Vision Soil Analyzer

Product design of a vision based soil analyzer

Jelle Spijker

PUBLISHED BY ROYAL IHC

BOOK-WEBSITE.COM

First printing, September 2015

Contents

Part One

1. Introduction

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.

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

This statement requires citation [2]; this one is more specific [1, page 122].

2.2.1 Functional requirements

Name Description
Word Definition
Comment Elaboration

2.2.2 Technical requirements

Name Description
Word Definition
Comment Elaboration

3. Technical Design

3.1 Theorems

This is an example of theorems.

3.2 Several equations

This is a theorem consisting of several equations.

Theorem 3.2.1 — Name of the theorem. In $E = \mathbb{R}^n$ all norms are equivalent. It has the properties:

$$\big|\,||\mathbf{x}|| - ||\mathbf{y}||\,\big| \leq ||\mathbf{x}-\mathbf{y}|| \tag{3.1}$$

$$||\sum_{i=1}^{n} \mathbf{x}_i|| \leq \sum_{i=1}^{n} ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \tag{3.2}$$

3.3 Single Line

This is a theorem consisting of just one line.

Theorem 3.3.1 A set $\mathscr{D}(G)$ in dense in $L^2(G)$, $|\cdot|_0$.

4. Vision design

4.1 Definitions

This is an example of a definition. A definition could be mathematical or it could define a concept.

Definition 4.1.1 — Definition name. Given a vector space E, a norm on E is an application, denoted $||\cdot||$, E in $\mathbb{R}^+ = [0, +\infty[$ such that:

$$||\mathbf{x}|| = 0 \;\Rightarrow\; \mathbf{x} = \mathbf{0} \tag{4.1}$$

$$||\lambda \mathbf{x}|| = |\lambda| \cdot ||\mathbf{x}|| \tag{4.2}$$

$$||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}|| \tag{4.3}$$

4.2 Notations

Notation 4.1. *Given an open subset G of $\mathbb{R}^n$, the set of functions φ are:*

1. *Bounded support G;*
2. *Infinitely differentiable;*

a vector space is denoted by $\mathscr{D}(G)$.

II

Two

5. Technical Realisation

6. Vision realisation

6.1 Image Acquisition

This is an example of a remark.

R The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K} = \mathbb{R}$, however, established properties are easily extended to $\mathbb{K} = \mathbb{C}$.

6.2 Corollaries

This is an example of a corollary.

Corollary 6.2.1 — Corollary name. The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K} = \mathbb{R}$, however, established properties are easily extended to $\mathbb{K} = \mathbb{C}$.

6.3 Propositions

This is an example of propositions.

6.3.1 Several equations

Proposition 6.3.1 — Proposition name. It has the properties:

$$\big|\,||\mathbf{x}|| - ||\mathbf{y}||\,\big| \leq ||\mathbf{x} - \mathbf{y}|| \tag{6.1}$$

$$||\sum_{i=1}^{n} \mathbf{x}_i|| \leq \sum_{i=1}^{n} ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \tag{6.2}$$

6.3.2 Single Line

Proposition 6.3.2 Let $f, g \in L^2(G)$; if $\forall \varphi \in \mathscr{D}(G)$, $(f, \varphi)_0 = (g, \varphi)_0$ then $f = g$.

6.4 Examples

This is an example of examples.

6.4.1 Equation and Text

■ **Example 6.1** Let $G = \{x \in \mathbb{R}^2 : |x| < 3\}$ and denoted by: $x^0 = (1,1)$; consider the function:

$$f(x) = \begin{cases} e^{|x|} & \text{si } |x - x^0| \leq 1/2 \\ 0 & \text{si } |x - x^0| > 1/2 \end{cases} \tag{6.3}$$

The function f has bounded support, we can take $A = \{x \in \mathbb{R}^2 : |x - x^0| \leq 1/2 + \varepsilon\}$ for all $\varepsilon \in]0;5/2 - \sqrt{2}[$. ■

6.4.2 Paragraph of Text

■ **Example 6.2 — Example name.** 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.

■

6.5 Exercises

This is an example of an exercise.

Exercise 6.1 This is a good place to ask a question to test learning progress or further cement ideas into students' minds. ■

6.6 Problems

Problem 6.1 What is the average airspeed velocity of an unladen swallow?

6.7 Vocabulary

Define a word to improve a students' vocabulary.
Vocabulary 6.1 — Word. Definition of word.

Part Two

7. Presenting Information

7.1 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 7.1: Table caption

7.2 Figure

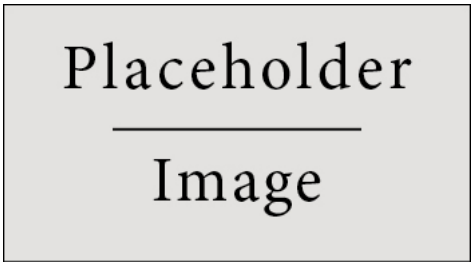

Figure 7.1: Figure caption

IV

Part Three

Bibliography

Books

[Smi12] John Smith. *Book title*. 1st edition. Volume 3. 2. City: Publisher, Jan. 2012, pages 123–200 (cited on page 10).

Articles

[Smi13] James Smith. “Article title”. In: 14.6 (Mar. 2013), pages 1–8 (cited on page 10).

Index

V

A. SoilMath Library

A.0.1 Genetic Algorithm Class

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  //! Genetic Algorithmes used for optimization problems
9  /*!
10   * Use this class for optimization problems. It's currently optimized for
11  * Neural Network optimzation
12   */
13 #pragma once
14
15 #include <bitset>
16 #include <random>
17 #include <string>
18 #include <algorithm>
19 #include <chrono>
20 #include <math.h>
21 #include <list>
22
23 //#include "NN.h"
24 #include "SoilMathTypes.h"
25 #include "MathException.h"
26
27 #include <QtCore/QObject>
28 #include <QDebug>
29 #include <QThread>
30 #include <QtConcurrent>
31
```

```
32 #include <boost/bind.hpp>
33
34 namespace SoilMath {
35
36 class GA : public QObject {
37   Q_OBJECT
38
39 public:
40   float MutationRate = 0.075f; /**< mutation rate*/
41   uint32_t Elitisme = 4;       /**< total number of the elite bastard*/
42   float EndError = 0.001f;     /**< acceptable error between last itteration*/
43   bool Revolution = true;
44
45   /*!
46  * \brief GA Standard constructor
47  */
48   GA();
49
50   /*!
51    * \brief GA Construction with a Neural Network initializers
52    * \param nnfunction the Neural Network prediction function which results will
53    * be optimized
54    * \param inputneurons the number of input neurons in the Neural Network don't
55    * count the bias
56    * \param hiddenneurons the number of hidden neurons in the Neural Network
57    * don't count the bias
58    * \param outputneurons the number of output neurons in the Neural Network
59    */
60   GA(NNfunctionType nnfunction, uint32_t inputneurons, uint32_t hiddenneurons,
61      uint32_t outputneurons);
62
63   /*!
64    * \brief GA standard de constructor
65    */
66   ~GA();
67
68   /*!
69    * \brief Evolve Darwin would be proud!!! This function creates a population
70    * and itterates
71    * through the generation till the maximum number off itterations has been
72    * reached of the
73    * error is acceptable
74    * \param inputValues complex vector with a reference to the inputvalues
75    * \param weights reference to the vector of weights which will be optimized
76    * \param rangeweights reference to the range of weights, currently it doesn't
77    * support indivudal ranges
78    * this is because of the crossing
79    * \param goal target value towards the Neural Network prediction function
80    * will be optimized
81    * \param maxGenerations maximum number of itterations default value is 200
82    * \param popSize maximum number of population, this should be an even number
83    */
84   void Evolve(const InputLearnVector_t &inputValues, Weight_t &weights,
85               MinMaxWeight_t rangeweights, OutputLearnVector_t &goal,
86               uint32_t maxGenerations = 200, uint32_t popSize = 30);
87 signals:
```

```
88    void learnErrorUpdate(double newError);
89
90  private:
91    NNfunctionType NNfuction; /**< The Neural Net work function*/
92    uint32_t inputneurons;    /**< the total number of input neurons*/
93    uint32_t hiddenneurons;   /**< the total number of hidden neurons*/
94    uint32_t outputneurons;   /**< the total number of output neurons*/
95
96    MinMaxWeight_t rangeweights;
97    InputLearnVector_t inputValues;
98    OutputLearnVector_t goal;
99
100   float minOptim = 0;
101   float maxOptim = 0;
102   uint32_t oldElit = 0;
103   float oldMutation = 0.;
104   std::list<double> last10Gen;
105   uint32_t currentGeneration = 0;
106   bool revolutionOngoing = false;
107
108   /*!
109    * \brief Genesis private function which is the spark of live, using a random
110    * seed
111    * \param weights a reference to the used Weight_t vector
112    * \param rangeweights pointer to the range of weights, currently it doesn't
113    * support indivudal ranges
114    * \param popSize maximum number of population, this should be an even number
115    * \return
116    */
117   Population_t Genesis(const Weight_t &weights, uint32_t popSize);
118
119   /*!
120    * \brief CrossOver a private function where the partners mate with each other
121    * The values or PopMember_t are expressed as bits or ar cut at the point
122    * CROSSOVER
123    * the population members are paired with the nearest neighbor and new members
124    * are
125    * created pairing the Genome_t of each other at the CROSSOVER point.
126    * Afterwards all
127    * the top tiers partners are allowed to mate again.
128    * \param pop reference to the population
129    */
130   void CrossOver(Population_t &pop);
131
132   /*!
133    * \brief Mutate a private function where individual bits from the Genome_t
134    * are mutated
135    * at a random uniform distribution event defined by the MUTATIONRATE
136    * \param pop reference to the population
137    */
138   void Mutate(Population_t &pop);
139
140   /*!
141    * \brief GrowToAdulthood a private function where the new population members
142    * serve as the
143    * the input for the Neural Network prediction function. The results are
```

```
144     * weight against
145     * the goal and this weight determine the fitness of the population member
146     * \param pop reference to the population
147     * \param inputValues a InputLearnVector_t with a reference to the inputvalues
148     * \param rangeweights pointer to the range of weights, currently it doesn't
149     * support indivudal ranges
150     * \param goal a Predict_t type with the expected value
151     * \param totalFitness a reference to the total population fitness
152     */
153    void GrowToAdulthood(Population_t &pop, float &totalFitness);
154
155    /*!
156     * \brief SurvivalOfTheFittest a private function where a battle to the death
157     * commences
158     * The fittest population members have the best chance of survival. Death is
159     * instigated
160     * with a random uniform distibution. The elite members don't partake in this
161     * desctruction
162     * The ELITISME rate indicate how many top tier members survive this
163     * catastrophic event.
164     * \param inputValues a InputLearnVector_t with a reference to the inputvalues
165     * \param totalFitness a reference to the total population fitness
166     * \return
167     */
168    bool SurvivalOfTheFittest(Population_t &pop, float &totalFitness);
169
170    /*!
171     * \brief PopMemberSort a private function where the members are sorted
172     * according to
173     * there fitness ranking
174     * \param i left hand population member
175     * \param j right hand population member
176     * \return true if the left member is closser to the goal as the right member.
177     */
178    static bool PopMemberSort(PopMember_t i, PopMember_t j) {
179      return (i.Fitness < j.Fitness);
180    }
181
182    /*!
183     * \brief Conversion of the value of type T to Genome_t
184     * \details Usage: Use <tt>ConvertToGenome<Type>(type, range)</tt>
185     * \param value The current value wich should be converted to a Genome_t
186     * \param range the range in which the value should fall, this is to have a
187     * Genome_t
188     * which utilizes the complete range 0000...n till 1111...n
189     */
190    template <typename T>
191    inline Genome_t ConvertToGenome(T value, std::pair<T, T> range) {
192      uint32_t intVal = static_cast<uint32_t>(
193          (UINT32_MAX * (range.first + value)) / (range.second - range.first));
194      Genome_t retVal(intVal);
195      return retVal;
196    }
197
198    /*!
199     * \brief Conversion of the Genome to a value
```

```
200     * \details Usage: use <tt>ConvertToValue<Type>(genome, range)
201     * \param gen is the Genome which is to be converted
202     * \param range is the range in which the value should fall
203     */
204    template <typename T>
205    inline T ConvertToValue(Genome_t gen, std::pair<T, T> range) {
206      T retVal =
207          range.first +
208          (((range.second - range.first) * static_cast<T>(gen.to_ulong())) /
209           UINT32_MAX);
210      return retVal;
211    }
212  };
213  }
```

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  #include "GA.h"
9
10 namespace SoilMath {
11 GA::GA() {}
12
13 GA::GA(NNfunctionType nnfunction, uint32_t inputneurons, uint32_t hiddenneurons,
14         uint32_t outputneurons) {
15   this->NNfuction = nnfunction;
16   this->inputneurons = inputneurons;
17   this->hiddenneurons = hiddenneurons;
18   this->outputneurons = outputneurons;
19 }
20
21 GA::~GA() {}
22
23 void GA::Evolve(const InputLearnVector_t &inputValues, Weight_t &weights,
24                 MinMaxWeight_t rangeweights, OutputLearnVector_t &goal,
25                 uint32_t maxGenerations, uint32_t popSize) {
26   minOptim = goal[0].OutputNeurons.size();
27   minOptim = -minOptim;
28   maxOptim = 2 * goal[0].OutputNeurons.size();
29   oldElit = Elitisme;
30   oldMutation = MutationRate;
31   this->inputValues = inputValues;
32   this->rangeweights = rangeweights;
33   this->goal = goal;
34
35   // Create the population
36   Population_t pop = Genesis(weights, popSize);
37   float totalFitness = 0.0;
38   for (uint32_t i = 0; i < maxGenerations; i++) {
39     CrossOver(pop);
40     Mutate(pop);
41     totalFitness = 0.0;
42     GrowToAdulthood(pop, totalFitness);
43     if (SurvivalOfTheFittest(pop, totalFitness)) {
44       break;
45     }
46   }
47   weights = pop[0].weights;
48 }
49
50 Population_t GA::Genesis(const Weight_t &weights, uint32_t popSize) {
51   if (popSize < 1)
52     return Population_t();
53
54   Population_t pop;
55   unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
```

```
56    std::default_random_engine gen(seed);
57    std::uniform_real_distribution<float> dis(rangeweights.first,
58                                              rangeweights.second);
59
60    for (uint32_t i = 0; i < popSize; i++) {
61      PopMember_t I;
62      for (uint32_t j = 0; j < weights.size(); j++) {
63        I.weights.push_back(dis(gen));
64        I.weightsGen.push_back(
65            ConvertToGenome<float>(I.weights[j], rangeweights));
66      }
67      pop.push_back(I);
68    }
69    return pop;
70  }
71
72  void GA::CrossOver(Population_t &pop) {
73    Population_t newPop; // create a new population
74    PopMember_t newPopMembers[2];
75    SplitGenome_t Split[2];
76
77    for (uint32_t i = 0; i < pop.size(); i += 2) {
78
79      for (uint32_t j = 0; j < pop[i].weights.size(); j++) {
80        // Split A
81        Split[0].first = std::bitset<CROSSOVER>(
82            pop[i].weightsGen[j].to_string().substr(0, CROSSOVER));
83        Split[0].second = std::bitset<GENE_MAX - CROSSOVER>(
84            pop[i].weightsGen[j].to_string().substr(CROSSOVER,
85                                                    GENE_MAX - CROSSOVER));
86
87        // Split B
88        Split[1].first = std::bitset<CROSSOVER>(
89            pop[i + 1].weightsGen[j].to_string().substr(0, CROSSOVER));
90        Split[1].second = std::bitset<GENE_MAX - CROSSOVER>(
91            pop[i + 1].weightsGen[j].to_string().substr(CROSSOVER,
92                                                        GENE_MAX - CROSSOVER));
93
94        // Mate A and B to AB and BA
95        newPopMembers[0].weightsGen.push_back(
96            Genome_t(Split[0].first.to_string() + Split[1].second.to_string()));
97        newPopMembers[1].weightsGen.push_back(
98            Genome_t(Split[1].first.to_string() + Split[0].second.to_string()));
99      }
100     newPop.push_back(newPopMembers[0]);
101     newPop.push_back(newPopMembers[1]);
102     newPopMembers[0].weightsGen.clear();
103     newPopMembers[1].weightsGen.clear();
104   }
105
106   // Allow the top tiers population partners to mate again
107   uint32_t halfN = pop.size() / 2;
108   for (uint32_t i = 0; i < halfN; i++) {
109     for (uint32_t j = 0; j < pop[i].weights.size(); j++) {
110       Split[0].first = std::bitset<CROSSOVER>(
111           pop[i].weightsGen[j].to_string().substr(0, CROSSOVER));
```

```
112         Split[0].second = std::bitset<GENE_MAX - CROSSOVER>(
113             pop[i].weightsGen[j].to_string().substr(CROSSOVER,
114                                                     GENE_MAX - CROSSOVER));
115
116         Split[1].first = std::bitset<CROSSOVER>(
117             pop[i + 2].weightsGen[j].to_string().substr(0, CROSSOVER));
118         Split[1].second = std::bitset<GENE_MAX - CROSSOVER>(
119             pop[i + 2].weightsGen[j].to_string().substr(CROSSOVER,
120                                                         GENE_MAX - CROSSOVER));
121
122         newPopMembers[0].weightsGen.push_back(
123             Genome_t(Split[0].first.to_string() + Split[1].second.to_string()));
124         newPopMembers[1].weightsGen.push_back(
125             Genome_t(Split[1].first.to_string() + Split[0].second.to_string()));
126       }
127       newPop.push_back(newPopMembers[0]);
128       newPop.push_back(newPopMembers[1]);
129       newPopMembers[0].weightsGen.clear();
130       newPopMembers[1].weightsGen.clear();
131     }
132     pop = newPop;
133   }
134
135   void GA::Mutate(Population_t &pop) {
136     unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
137     std::default_random_engine gen(seed);
138     std::uniform_real_distribution<float> dis(0, 1);
139
140     std::default_random_engine genGen(seed);
141     std::uniform_int_distribution<int> disGen(0, (GENE_MAX - 1));
142
143     QtConcurrent::blockingMap<Population_t>(pop, [&](PopMember_t &P) {
144       for (uint32_t j = 0; j < P.weightsGen.size(); j++) {
145         if (dis(gen) < MutationRate) {
146           P.weightsGen[j][disGen(genGen)].flip();
147         }
148       }
149     });
150   }
151
152   void GA::GrowToAdulthood(Population_t &pop, float &totalFitness) {
153
154     QtConcurrent::blockingMap<Population_t>(pop, [&](PopMember_t &P) {
155       // std::for_each(pop.begin(), pop.end(), [&](PopMember_t &P) {
156       for (uint32_t j = 0; j < P.weightsGen.size(); j++) {
157         P.weights.push_back(ConvertToValue<float>(P.weightsGen[j], rangeweights));
158       }
159       Weight_t iWeight(P.weights.begin(),
160                        P.weights.begin() + ((inputneurons + 1) * hiddenneurons));
161       Weight_t hWeight(P.weights.begin() + ((inputneurons + 1) * hiddenneurons),
162                        P.weights.end());
163
164       for (uint32_t j = 0; j < inputValues.size(); j++) {
165         Predict_t results = NNfuction(inputValues[j], iWeight, hWeight,
166                                       inputneurons, hiddenneurons, outputneurons);
167         // See issue #85
```

```
168         bool allGood = true;
169         float fitness = 0.0;
170         for (uint32_t k = 0; k < results.OutputNeurons.size(); k++) {
171           bool resultSign = std::signbit(results.OutputNeurons[k]);
172           bool goalSign = std::signbit(goal[j].OutputNeurons[k]);
173           fitness += results.OutputNeurons[k] / goal[j].OutputNeurons[k];
174           if (resultSign != goalSign) {
175             allGood = false;
176           }
177         }
178         fitness += (allGood) ? results.OutputNeurons.size() : 0;
179         P.Fitness += fitness;
180       }
181     });
182
183     for_each(pop.begin(), pop.end(), [&](PopMember_t &P) {
184       P.Fitness /= inputValues.size();
185       totalFitness += P.Fitness;
186     });
187   }
188
189   bool GA::SurvivalOfTheFittest(Population_t &pop, float &totalFitness) {
190     bool retVal = false;
191     uint32_t decimationCount = pop.size() / 2;
192
193     unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
194     std::default_random_engine gen(seed);
195
196     std::sort(pop.begin(), pop.end(),
197               [](const PopMember_t &L, const PopMember_t &R) {
198                 return L.Fitness < R.Fitness;
199               });
200
201     float maxFitness = pop[pop.size() - 1].Fitness * pop.size();
202     uint32_t i = Elitisme;
203     while (pop.size() > decimationCount) {
204       if (i == pop.size()) {
205         i = Elitisme;
206       }
207       std::uniform_real_distribution<float> dis(0, maxFitness);
208       if (dis(gen) > pop[i].Fitness) {
209         totalFitness -= pop[i].Fitness;
210         pop.erase(pop.begin() + i);
211       }
212       i++;
213     }
214
215     std::sort(pop.begin(), pop.end(),
216               [](const PopMember_t &L, const PopMember_t &R) {
217                 return L.Fitness > R.Fitness;
218               });
219
220     float learnError = 1 - ((pop[0].Fitness - minOptim) / (maxOptim - minOptim));
221
222     // Viva la Revolution
223     if (currentGeneration > 9) {
```

```
224       double avg = 0;
225       for_each(last10Gen.begin(), last10Gen.end(), [&](double &G) { avg += G; });
226       avg /= 10;
227       double minMax[2] = {avg * 0.98, avg * 1.02};
228       if (learnError > minMax[0] && learnError < minMax[1]) {
229         if (!revolutionOngoing) {
230           qDebug() << "Viva la revolution!";
231           oldElit = Elitisme;
232           Elitisme = 0;
233           oldMutation = MutationRate;
234           MutationRate = 0.25;
235           revolutionOngoing = true;
236         }
237       } else if (revolutionOngoing) {
238         qDebug() << "Peace has been restort";
239         Elitisme = oldElit;
240         MutationRate = oldMutation;
241         revolutionOngoing = false;
242       }
243       last10Gen.pop_front();
244       last10Gen.push_back(learnError);
245     } else {
246       last10Gen.push_back(learnError);
247     }
248     currentGeneration++;
249     emit learnErrorUpdate(static_cast<double>(learnError));
250     if (learnError < EndError) {
251       retVal = true;
252     }
253     return retVal;
254 }
255 }
```

A.0.2 Fast Fourier Transform Class

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  #pragma once
9
10 #include <vector>
11 #include <complex>
12 #include <cmath>
13 #include <valarray>
14 #include <array>
15 #include <deque>
16 #include <queue>
17 #include <iterator>
18 #include <algorithm>
19 #include <stdint.h>
20 #include <opencv2/core.hpp>
21 #include "SoilMathTypes.h"
22 #include "MathException.h"
23
24 namespace SoilMath {
25 /*!
26  * \brief Fast Fourier Transform class
27  * \details Use this class to transform a black and white blob presented as a
28  * cv::Mat with values 0 or 1 to a vector of complex values representing the Fouri
29  * Descriptors.
30  */
31 class FFT {
32 public:
33   /*!
34  * \brief Standard constructor
35  */
36   FFT();
37
38   /*!
39   * \brief Standard deconstructor
40   */
41   ~FFT();
42
43   /*!
44    * \brief Transforming the img to the frequency domain and returning the
45    * Fourier Descriptors
46    * \param img contour in the form of a cv::Mat type CV_8UC1. Which should
47    * consist of a continous contour. \f$ \{ img \in \mathbb{Z} | 0 \leq img \leq
48    * 1 \} \f$
49    * \return a vector with complex values, represing the contour in the
50    * frequency domain, expressed as Fourier Descriptors
51    */
52   ComplexVect_t GetDescriptors(const cv::Mat &img);
53
54 private:
```

```
55    ComplexVect_t
56        fftDescriptors; /**< Vector with complex values which represent the
57                           descriptors*/
58    ComplexVect_t
59        complexcontour; /**< Vector with complex values which represent the
60                           contour*/
61    cv::Mat Img;        /**< Img which will be analysed*/
62
63    /*!
64     * \brief Contour2Complex a private function which translates a continous
65     * contour image
66     * to a vector of complex values. The contour is found using a depth first
67     * search with
68     * extension list. The alghorithm is based upon <a
69     * href="http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/
70     * opencourseware
71     * 6-034-artificial-intelligence lecture 4</a>
72     * \param img contour in the form of a cv::Mat type CV_8UC1. Which should
73     * consist of a continous contour. \f$ \{ img \in \mathbb{Z} | 0 \leq img \leq
74     * 1 \} \f$
75     * \param centerCol centre of the contour X value
76     * \param centerRow centre of the contour Y value
77     * \return a vector with complex values, represing the contour as a function
78     */
79    ComplexVect_t Contour2Complex(const cv::Mat &img, float centerCol,
80                                  float centerRow);
81    /*!
82     * \brief Neighbors a private function returning the neighboring pixels which
83     * belong to a contour
84     * \param O uchar pointer to the data
85     * \param pixel current counter
86     * \param columns total number of columns
87     * \param rows total number of rows
88     * \return
89     */
90    iContour_t Neighbors(uchar *O, int pixel, uint32_t columns, uint32_t rows);
91
92    /*!
93     * \brief fft a private function calculating the Fast Fourier Transform
94     * let \f$ m \f$ be an integer and let \f$ N=2^m \f$ also
95     * \f$ CA=[x_0,\ldots,x_{N-1}] \f$ is an \f$ N \f$ dimensional complex vector
96     * let \f$ \omega=\exp({-2\pi i\over N}) \f$
97     * then \f$ c_k={\frac{1}{N}}\sum_{j=0}^{j=N-1}CA_j\omega^{jk} \f$
98     * \param CA a \f$ CA=[x_0,\ldots,x_{N-1}] \f$ is an \f$ N \f$ dimensional
99     * complex vector
100    */
101   void fft(ComplexArray_t &CA);
102
103   /*!
104    * \brief ifft
105    * \param CA
106    */
107   void ifft(ComplexArray_t &CA);
108 };
109 }
```

```
/* 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 *0, 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) {
```

```
56        break;
57      }
58      if (O[count] == 1)
59        neighbors.push_back(count);
60    }
61    return neighbors;
62  }
63
64  ComplexVect_t FFT::Contour2Complex(const cv::Mat &img, float centerCol,
65                                     float centerRow) {
66    uchar *O = img.data;
67    uint32_t pEnd = img.cols * img.rows;
68
69    std::deque<std::deque<uint32_t>> sCont;
70    std::deque<uint32_t> eList;
71
72    // Initialize the queue
73    for (uint32_t i = 0; i < pEnd; i++) {
74      if (O[i] == 1) {
75        std::deque<uint32_t> tmpQ;
76        tmpQ.push_back(i);
77        sCont.push_back(tmpQ);
78        break;
79      }
80    }
81
82    if (sCont.front().size() < 1) {
83      throw Exception::MathException(EXCEPTION_NO_CONTOUR_FOUND,
84                                     EXCEPTION_NO_CONTOUR_FOUND_NR);
85    } // Exception handling
86
87    uint32_t prev = -1;
88
89    // Extend path on queue
90    for (uint32_t i = sCont.front().front(); i < pEnd;) {
91      iContour_t nBors =
92          Neighbors(O, i, img.cols, img.rows); // find neighboring pixels
93      std::deque<uint32_t> cQ = sCont.front(); // store first queue;
94      sCont.erase(sCont.begin());              // erase first queue from beginning
95      if (cQ.size() > 1) {
96        prev = cQ.size() - 2;
97      } else {
98        prev = 0;
99      }
100     // Loop through each neighbor
101     for (uint32_t j = 0; j < nBors.size(); j++) {
102       if (nBors[j] != cQ[prev]) // No backtracking
103       {
104         if (nBors[j] == cQ.front() && cQ.size() > 8) {
105           i = pEnd;
106         } // Back at first node
107         if (std::find(eList.begin(), eList.end(), nBors[j]) ==
108             eList.end()) // Check if this current route is extended elsewhere
109         {
110           std::deque<uint32_t> nQ = cQ;
111           nQ.push_back(nBors[j]); // Add the neighbor to the queue
```

```
112             sCont.push_front(nQ);   // add the sequence to the front of the queue
113           }
114         }
115       }
116       if (nBors.size() > 2) {
117         eList.push_back(i);
118       } // if there are multiple choices put current node in extension List
119       if (i != pEnd) {
120         i = sCont.front().back();
121       } // If it isn't the end set i to the last node of the first queue
122       if (sCont.size() == 0) {
123         throw Exception::MathException(EXCEPTION_NO_CONTOUR_FOUND,
124                                        EXCEPTION_NO_CONTOUR_FOUND_NR);
125       }
126     }
127
128     // convert the first queue to a complex normalized vector
129     Complex_t cPoint;
130     ComplexVect_t contour;
131     float col = 0.0;
132     // Normalize and convert the complex function
133     for_each(
134         sCont.front().begin(), sCont.front().end(),
135         [&img, &cPoint, &contour, &centerCol, &centerRow, &col](uint32_t &e) {
136           col = (float)((e % img.cols) - centerCol);
137           if (col == 0.0) {
138             cPoint.real(1.0);
139           } else {
140             cPoint.real((float)(col / centerCol));
141           }
142           cPoint.imag((float)((floorf(e / img.cols) - centerRow) / centerRow));
143           contour.push_back(cPoint);
144         });
145
146     return contour;
147   }
148
149   void FFT::fft(ComplexArray_t &CA) {
150     const size_t N = CA.size();
151     if (N <= 1) {
152       return;
153     }
154
155     //!< Divide and conquor
156     ComplexArray_t even = CA[std::slice(0, N / 2, 2)];
157     ComplexArray_t odd = CA[std::slice(1, N / 2, 2)];
158
159     fft(even);
160     fft(odd);
161
162     for (size_t k = 0; k < N / 2; ++k) {
163       Complex_t ct = std::polar(1.0, -2 * M_PI * k / N) * odd[k];
164       CA[k] = even[k] + ct;
165       CA[k + N / 2] = even[k] - ct;
166     }
167   }
```

```
168
169 void FFT::ifft(ComplexArray_t &CA) {
170   CA = CA.apply(std::conj);
171   fft(CA);
172   CA = CA.apply(std::conj);
173   CA /= CA.size();
174 }
175 }
```

A.0.3 Neural Network Class

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  #pragma once
9
10 #include <stdint.h>
11 #include <vector>
12 #include <string>
13 #include <fstream>
14
15 #include <boost/archive/xml_iarchive.hpp>
16 #include <boost/archive/xml_oarchive.hpp>
17 #include <boost/serialization/vector.hpp>
18 #include <boost/serialization/version.hpp>
19
20 #include "GA.h"
21 #include "MathException.h"
22 #include "SoilMathTypes.h"
23 #include "FFT.h"
24
25 #include <QtCore/QObject>
26
27 namespace SoilMath {
28 /*!
29  * \brief The Neural Network class
30  * \details This class is used to make prediction on large data set. Using self
31  * learning algoritmes
32  */
33 class NN : public QObject {
34   Q_OBJECT
35
36 public:
37   /*!
38  * \brief NN constructor for the Neural Net
39  * \param inputneurons number of input neurons
40  * \param hiddenneurons number of hidden neurons
41  * \param outputneurons number of output neurons
42  */
43   NN(uint32_t inputneurons, uint32_t hiddenneurons, uint32_t outputneurons);
44
45   /*!
46    * \brief NN constructor for the Neural Net
47    */
48   NN();
49
50   /*!
51    * \brief ~NN virtual deconstructor for the Neural Net
52    */
53   virtual ~NN();
54
```

```
55    /*!
56     * \brief Predict The prediction function.
57     * \details In this function the neural net is setup and the input which are
58     * the complex values descriping the contour in the frequency domein serve as
59     * input. The absolute value of these im. number because I'm not interrested
60     * in the orrientation of the particle but more in the degree of variations.
61     * \param input vector of complex input values, these're the Fourier
62     * descriptors
63     * \return a real valued vector of the output neurons
64     */
65    Predict_t Predict(ComplexVect_t input);
66
67    /*!
68     * \brief PredictLearn a static function used in learning of the weights
69     * \details It starts a new Neural Network object and passes all the
70     * paramaters in to this newly created object. After this the predict function
71     * is called and the value is returned. This work around was needed to pass
72     * the neural network to the Genetic Algorithm class.
73     * \param input a complex vector of input values
74     * \param inputweights the input weights
75     * \param hiddenweights the hidden weights
76     * \param inputneurons the input neurons
77     * \param hiddenneurons the hidden neurons
78     * \param outputneurons the output neurons
79     * \return
80     */
81    static Predict_t PredictLearn(ComplexVect_t input, Weight_t inputweights,
82                                  Weight_t hiddenweights, uint32_t inputneurons,
83                                  uint32_t hiddenneurons, uint32_t outputneurons);
84
85    /*!
86     * \brief SetInputWeights a function to set the input weights
87     * \param value the real valued vector with the values
88     */
89    void SetInputWeights(Weight_t value) { iWeights = value; }
90
91    /*!
92     * \brief SetHiddenWeights a function to set the hidden weights
93     * \param value the real valued vector with the values
94     */
95    void SetHiddenWeights(Weight_t value) { hWeights = value; }
96
97    /*!
98     * \brief SetBeta a function to set the beta value
99     * \param value a floating value ussualy between 0.5 and 1.5
100    */
101   void SetBeta(float value) { beta = value; }
102   float GetBeta() { return beta; }
103
104   /*!
105    * \brief Learn the learning function
106    * \param input a vector of vectors with complex input values
107    * \param cat a vector of vectors with the know output values
108    * \param noOfDescriptorsUsed the total number of descriptos which should be
109    * used
110    */
```

```
111   void Learn(InputLearnVector_t input, OutputLearnVector_t cat,
112              uint32_t noOfDescriptorsUsed);
113
114   /*!
115    * \brief SaveState Serialize and save the values of the Neural Net to disk
116    * \details Save the Neural Net in XML valued text file to disk so that a
117    * object can
118    * be reconstructed on a latter stadia.
119    * \param filename a string indicating the file location and name
120    */
121   void SaveState(std::string filename);
122
123   /*!
124    * \brief LoadState Loads the previouse saved Neural Net from disk
125    * \param filename a string indicating the file location and name
126    */
127   void LoadState(std::string filename);
128
129   Weight_t iWeights; /**< a vector of real valued floating point input weights*/
130   Weight_t hWeights; /**< a vector of real valued floating point hidden weight*/
131
132   uint32_t MaxGenUsedByGA = 200;
133   uint32_t PopulationSizeUsedByGA = 30;
134   float MutationrateUsedByGA = 0.075f;
135   uint32_t ElitismeUsedByGA = 4;
136   float EndErrorUsedByGA = 0.001;
137   float MaxWeightUsedByGA = 50;
138   float MinWeightUSedByGa = -50;
139
140   uint32_t GetInputNeurons() { return inputNeurons; }
141   void SetInputNeurons(uint32_t value);
142
143   uint32_t GetHiddenNeurons() { return hiddenNeurons; }
144   void SetHiddenNeurons(uint32_t value);
145
146   uint32_t GetOutputNeurons() { return outputNeurons; }
147   void SetOutputNeurons(uint32_t value);
148
149   bool studied =
150       false; /**< a value indicating if the weights are a results of a
151                 learning curve*/
152
153 signals:
154   void learnErrorUpdate(double newError);
155
156 private:
157   GA *optim = nullptr;
158   std::vector<float> iNeurons; /**< a vector of input values, the bias is
159                                   included, the bias is included and
160                                     is the first value*/
161   std::vector<float>
162       hNeurons; /**< a vector of hidden values, the bias is included and
163                    is the first value*/
164   std::vector<float> oNeurons; /**< a vector of output values*/
165
166   uint32_t hiddenNeurons = 50; /**< number of hidden neurons minus bias*/
```

```
167   uint32_t inputNeurons = 20;  /**< number of input neurons minus bias*/
168   uint32_t outputNeurons = 18; /**< number of output neurons*/
169   float beta; /**< the beta value, this indicates the steepness of the sigmoid
170                  function*/
171
172   friend class boost::serialization::access; /**< a private friend class so the
173                                                  serialization can access all
174                                                  the needed functions*/
175   /*!
176    * \brief serialization function
177    * \param ar the object
178    * \param version the version of the class
179    */
180   template <class Archive>
181   void serialize(Archive &ar, const unsigned int version) {
182     if (version == 0) {
183       ar &BOOST_SERIALIZATION_NVP(inputNeurons);
184       ar &BOOST_SERIALIZATION_NVP(hiddenNeurons);
185       ar &BOOST_SERIALIZATION_NVP(outputNeurons);
186       ar &BOOST_SERIALIZATION_NVP(iWeights);
187       ar &BOOST_SERIALIZATION_NVP(hWeights);
188       ar &BOOST_SERIALIZATION_NVP(beta);
189       ar &BOOST_SERIALIZATION_NVP(studied);
190       ar &BOOST_SERIALIZATION_NVP(MaxGenUsedByGA);
191       ar &BOOST_SERIALIZATION_NVP(PopulationSizeUsedByGA);
192       ar &BOOST_SERIALIZATION_NVP(MutationrateUsedByGA);
193       ar &BOOST_SERIALIZATION_NVP(ElitismeUsedByGA);
194       ar &BOOST_SERIALIZATION_NVP(EndErrorUsedByGA);
195       ar &BOOST_SERIALIZATION_NVP(MaxWeightUsedByGA);
196       ar &BOOST_SERIALIZATION_NVP(MinWeightUSedByGa);
197     }
198   }
199 };
200 }
201 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);
}

```

```
56 Predict_t NN::Predict(ComplexVect_t input) {
57   if (input.size() != inputNeurons) {
58     throw Exception::MathException(EXCEPTION_SIZE_OF_INPUT_NEURONS,
59                                    EXCEPTION_SIZE_OF_INPUT_NEURONS_NR);
60   }
61   if (!studied) {
62     throw Exception::MathException(EXCEPTION_NEURAL_NET_NOT_STUDIED,
63                                    EXCEPTION_NEURAL_NET_NOT_STUDIED_NR);
64   }
65
66   iNeurons.clear();
67   hNeurons.clear();
68   oNeurons.clear();
69
70   // Set the bias in the input and hidden vector to 1 (real number)
71   iNeurons.push_back(1.0f);
72   hNeurons.push_back(1.0f);
73
74   Predict_t retVal;
75   uint32_t wCount = 0;
76
77   // Init the network
78   for (uint32_t i = 0; i < inputNeurons; i++) {
79     iNeurons.push_back(static_cast<float>(abs(input[i])));
80   }
81   for (uint32_t i = 0; i < hiddenNeurons; i++) {
82     hNeurons.push_back(0.0f);
83   }
84   for (uint32_t i = 0; i < outputNeurons; i++) {
85     oNeurons.push_back(0.0f);
86   }
87
88   for (uint32_t i = 1; i < hNeurons.size(); i++) {
89     wCount = i - 1;
90     for (uint32_t j = 0; j < iNeurons.size(); j++) {
91       hNeurons[i] += iNeurons[j] * iWeights[wCount];
92       wCount += hNeurons.size() - 1;
93     }
94     hNeurons[i] = 1 / (1 + pow(2.71828f, (-hNeurons[i] * beta)));
95   }
96
97   for (uint32_t i = 0; i < oNeurons.size(); i++) {
98     wCount = i;
99     for (uint32_t j = 0; j < hNeurons.size(); j++) {
100      oNeurons[i] += hNeurons[j] * hWeights[wCount];
101      wCount += oNeurons.size();
102    }
103    oNeurons[i] =
104        (2 / (1.0f + pow(2.71828f, (-oNeurons[i] * beta)))) -
105        1; // Shift plus scale so the learning function can be calculated
106  }
107
108  retVal.OutputNeurons = oNeurons;
109  retVal.ManualSet = false;
110  return retVal;
111 }
```

```
112
113 void NN::Learn(InputLearnVector_t input, OutputLearnVector_t cat,
114                 uint32_t noOfDescriptorsUsed __attribute__((unused))) {
115   if (optim == nullptr) {
116       optim = new SoilMath::GA(PredictLearn, inputNeurons, hiddenNeurons, outputNe
117     }
118   connect(optim, SIGNAL(learnErrorUpdate(double)), this, SIGNAL(learnErrorUpdate(d
119
120   optim->Elitisme = ElitismeUsedByGA;
121   optim->EndError = EndErrorUsedByGA;
122   optim->MutationRate = MutationrateUsedByGA;
123
124   ComplexVect_t inputTest;
125   std::vector<Weight_t> weights;
126   Weight_t weight(((inputNeurons + 1) * hiddenNeurons) +
127                       ((hiddenNeurons + 1) * outputNeurons),
128                   0);
129   // loop through each case and adjust the weights
130   optim->Evolve(input, weight,
131                 MinMaxWeight_t(MinWeightUSedByGa, MaxWeightUsedByGA), cat,
132                 MaxGenUsedByGA, PopulationSizeUsedByGA);
133
134   this->iWeights = Weight_t(
135       weight.begin(), weight.begin() + ((inputNeurons + 1) * hiddenNeurons));
136   this->hWeights = Weight_t(
137       weight.begin() + ((inputNeurons + 1) * hiddenNeurons), weight.end());
138   studied = true;
139 }
140
141 void NN::SetInputNeurons(uint32_t value) {
142   if (value != inputNeurons) {
143     inputNeurons = value;
144     iNeurons.clear();
145     iNeurons.reserve(inputNeurons + 1);
146     studied = false;
147   }
148 }
149
150 void NN::SetHiddenNeurons(uint32_t value) {
151   if (value != hiddenNeurons) {
152     hiddenNeurons = value;
153     hNeurons.clear();
154     hNeurons.reserve(hiddenNeurons + 1);
155     studied = false;
156   }
157 }
158
159 void NN::SetOutputNeurons(uint32_t value) {
160   if (value != outputNeurons) {
161     outputNeurons = value;
162     oNeurons.clear();
163     oNeurons.reserve(outputNeurons);
164     studied = false;
165   }
166 }
167 }
```

A.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*/
```

```
55    T1 binRange = 0;          /**< the range of a single bin*/
56    float Std = 0.0;          /**< standard deviation*/
57    T3 Sum = 0;               /**< total sum of all the data values*/
58    uint16_t Rows = 0;        /**< number of rows from the data matrix*/
59    uint16_t Cols = 0;        /**< number of cols from the data matrix*/
60    bool StartAtZero = true;  /**< indication of the minimum value starts at zero
61                                 or could be less*/
62    double *BinRanges = nullptr;
63    double HighestPDF = 0.;
64
65    uint32_t *begin() { return &bins[0]; }    /**< pointer to the first bin*/
66    uint32_t *end() { return &bins[noBins]; } /**< pointer to the last + 1 bin*/
67
68    /*!
69     * \brief WelchTest Compare the sample using the Welch's Test
70     * \details (source:
71     * http://www.boost.org/doc/libs/1_57_0/libs/math/doc/html/math_toolkit/stat_tut
72     * \param statComp Statiscs Results of which it should be tested against
73     * \return
74     */
75    bool WelchTest(SoilMath::Stats<T1, T2, T3> &statComp) {
76      double alpha = 0.05;
77      // Degrees of freedom:
78      double v = statComp.Std * statComp.Std / statComp.n +
79                 this->Std * this->Std / this->n;
80      v *= v;
81      double t1 = statComp.Std * statComp.Std / statComp.n;
82      t1 *= t1;
83      t1 /= (statComp.n - 1);
84      double t2 = this->Std * this->Std / this->n;
85      t2 *= t2;
86      t2 /= (this->n - 1);
87      v /= (t1 + t2);
88      // t-statistic:
89      double t_stat = (statComp.Mean - this->Mean) /
90                      sqrt(statComp.Std * statComp.Std / statComp.n +
91                           this->Std * this->Std / this->n);
92      //
93      // Define our distribution, and get the probability:
94      //
95      boost::math::students_t dist(v);
96      double q = cdf(complement(dist, fabs(t_stat)));
97
98      bool rejected = false;
99      // Sample 1 Mean == Sample 2 Mean test the NULL hypothesis, the two means
100     // are the same
101     if (q < alpha / 2)
102       rejected = false;
103     else
104       rejected = true;
105     return rejected;
106   }
107
108   /*!
109    * \brief Stats Constructor
110    * \param rhs Right hand side
```

```
111     */
112   Stats(const Stats &rhs)
113       : bins{new uint32_t[rhs.noBins]{0}}, CFD{new double[rhs.noBins]{}},
114         BinRanges{new double[rhs.noBins]{}} {
115     this->binRange = rhs.binRange;
116     this->Calculated = rhs.Calculated;
117     this->Cols = rhs.Cols;
118     this->EndBin = rhs.EndBin;
119     this->isDiscrete = rhs.isDiscrete;
120     this->max = rhs.max;
121     this->Mean = rhs.Mean;
122     this->min = rhs.min;
123     this->n = rhs.n;
124     this->noBins = rhs.noBins;
125     this->n_end = rhs.n_end;
126     this->Range = rhs.Range;
127     this->Rows = rhs.Rows;
128     this->Startbin = rhs.Startbin;
129     this->Std = rhs.Std;
130     this->Sum = rhs.Sum;
131     std::copy(rhs.bins, rhs.bins + rhs.noBins, this->bins);
132     std::copy(rhs.CFD, rhs.CFD + rhs.noBins, this->CFD);
133     std::copy(rhs.BinRanges, rhs.BinRanges + rhs.noBins, this->BinRanges);
134     this->Data = rhs.Data;
135     this->StartAtZero = rhs.StartAtZero;
136     this->HighestPDF = rhs.HighestPDF;
137   }
138
139   /*!
140    * \brief operator = Assigmnet operator
141    * \param rhs right hand side
142    * \return returns the right hand side
143    */
144   Stats &operator=(Stats const &rhs) {
145     if (&rhs != this) {
146       Data = rhs.Data;
147
148       if (bins != nullptr) {
149         delete[] bins;
150         bins = nullptr;
151       }
152       if (CFD != nullptr) {
153         delete[] CFD;
154         CFD = nullptr;
155       }
156       if (BinRanges != nullptr) {
157         delete[] BinRanges;
158         BinRanges = nullptr;
159       }
160
161       bins = new uint32_t[rhs.noBins];     // leak
162       CFD = new double[rhs.noBins];        // leak
163       BinRanges = new double[rhs.noBins]; // leak
164       this->binRange = rhs.binRange;
165       this->Calculated = rhs.Calculated;
166       this->Cols = rhs.Cols;
```

```
167         this->EndBin = rhs.EndBin;
168         this->isDiscrete = rhs.isDiscrete;
169         this->max = rhs.max;
170         this->Mean = rhs.Mean;
171         this->min = rhs.min;
172         this->n = rhs.n;
173         this->noBins = rhs.noBins;
174         this->n_end = rhs.n_end;
175         this->Range = rhs.Range;
176         this->Rows = rhs.Rows;
177         this->Startbin = rhs.Startbin;
178         this->Std = rhs.Std;
179         this->Sum = rhs.Sum;
180         this->Data = &rhs.Data[0];
181         std::copy(rhs.bins, rhs.bins + rhs.noBins, this->bins);
182         std::copy(rhs.CFD, rhs.CFD + rhs.noBins, this->CFD);
183         std::copy(rhs.BinRanges, rhs.BinRanges + rhs.noBins, this->BinRanges);
184         this->StartAtZero = rhs.StartAtZero;
185         this->HighestPDF = rhs.HighestPDF;
186       }
187       return *this;
188     }
189
190     /*!
191      * \brief Stats Constructor
192      * \param noBins number of bins with which to build the histogram
193      * \param startBin starting value of the first bin
194      * \param endBin end value of the second bin
195      */
196     Stats(int noBins = 256, T1 startBin = 0, T1 endBin = 255) {
197       min = std::numeric_limits<T1>::max();
198       max = std::numeric_limits<T1>::min();
199       Range = std::numeric_limits<T1>::max();
200       Startbin = startBin;
201       EndBin = endBin;
202       this->noBins = noBins;
203       bins = new uint32_t[noBins]{0};    // leak
204       CFD = new double[noBins]{};        // leak
205       BinRanges = new double[noBins]{}; // leak
206
207       if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) ||
208           typeid(T1) == typeid(long double)) {
209         isDiscrete = false;
210         binRange = static_cast<T1>((EndBin - Startbin) / noBins);
211       } else {
212         isDiscrete = true;
213         binRange = static_cast<T1>(round((EndBin - Startbin) / noBins));
214       }
215     }
216
217     /*!
218      * \brief Stats constructor
219      * \param data Pointer to the data
220      * \param rows Number of rows
221      * \param cols Number of Columns
222      * \param noBins Number of bins
```

```
223     * \param startBin Value of the start bin
224     * \param startatzero bool indicating if the bins should be shifted from zero
225     */
226    Stats(T1 *data, uint16_t rows, uint16_t cols, int noBins = 256,
227          T1 startBin = 0, bool startatzero = true) {
228      min = std::numeric_limits<T1>::max();
229      max = std::numeric_limits<T1>::min();
230      Range = max - min;
231
232      Startbin = startBin;
233      EndBin = startBin + noBins;
234      StartAtZero = startatzero;
235
236      if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) ||
237          typeid(T1) == typeid(long double)) {
238        isDiscrete = false;
239      } else {
240        isDiscrete = true;
241      }
242
243      Data = data;
244      Rows = rows;
245      Cols = cols;
246      bins = new uint32_t[noBins]{0};
247      CFD = new double[noBins]{};
248      BinRanges = new double[noBins]{};
249      this->noBins = noBins;
250      if (isDiscrete) {
251        BasicCalculate();
252      } else {
253        BasicCalculateFloat();
254      }
255    }
256
257    /*!
258     * \brief Stats Constructor
259     * \param data Pointer the data
260     * \param rows Number of rows
261     * \param cols Number of Columns
262     * \param mask the mask should have the same size as the data a value of zero
263     * indicates that the data pointer doesn't exist. A 1 indicates that the data
264     * pointer is to be used
265     * \param noBins Number of bins
266     * \param startBin Value of the start bin
267     * \param startatzero indicating if the bins should be shifted from zero
268     */
269    Stats(T1 *data, uint16_t rows, uint16_t cols, uchar *mask, int noBins = 256,
270          T1 startBin = 0, bool startatzero = true) {
271      min = std::numeric_limits<T1>::max();
272      max = std::numeric_limits<T1>::min();
273      Range = max - min;
274
275      Startbin = startBin;
276      EndBin = startBin + noBins;
277      StartAtZero = startatzero;
278
```

```
279     if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) ||
280         typeid(T1) == typeid(long double)) {
281       isDiscrete = false;
282     } else {
283       isDiscrete = true;
284     }
285
286     Data = data;
287     Rows = rows;
288     Cols = cols;
289     bins = new uint32_t[noBins]{0};
290     CFD = new double[noBins]{};
291     BinRanges = new double[noBins]{};
292     this->noBins = noBins;
293     if (isDiscrete) {
294       BasicCalculate(mask);
295     } else {
296       BasicCalculateFloat(mask);
297     }
298   }
299
300   /*!
301    * \brief Stats Constructor
302    * \param binData The histogram data
303    * \param startC start counter
304    * \param endC end counter
305    */
306   Stats(T2 *binData, uint16_t startC, uint16_t endC) {
307     noBins = endC - startC;
308     Startbin = startC;
309     EndBin = endC;
310     uint32_t i = noBins;
311
312     if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) ||
313         typeid(T1) == typeid(long double)) {
314       isDiscrete = false;
315       throw Exception::MathException(EXCEPTION_TYPE_NOT_SUPPORTED,
316                                      EXCEPTION_TYPE_NOT_SUPPORTED_NR);
317     } else {
318       isDiscrete = true;
319     }
320
321     bins = new uint32_t[noBins]{0};
322     CFD = new double[noBins]{};
323     BinRanges = new double[noBins]{};
324     while (i-- > 0) {
325       bins[i] = binData[i];
326       n += binData[i];
327     }
328     BinCalculations(startC, endC);
329   }
330
331   ~Stats() {
332     Data == nullptr;
333     if (bins != nullptr) {
334       delete[] bins;
```

```
335       bins = nullptr;
336     }
337     if (CFD != nullptr) {
338       delete[] CFD;
339       CFD = nullptr;
340     }
341     if (BinRanges != nullptr) {
342       delete[] BinRanges;
343       BinRanges = nullptr;
344     }
345   }
346
347   /*!
348    * \brief BasicCalculateFloat execute the basic float data calculations
349    */
350   void BasicCalculateFloat() {
351     float sum_dev = 0.0;
352     n = Rows * Cols;
353     for (uint32_t i = 0; i < n; i++) {
354       if (Data[i] > max) {
355         max = Data[i];
356       }
357       if (Data[i] < min) {
358         min = Data[i];
359       }
360       Sum += Data[i];
361     }
362     binRange = (max - min) / noBins;
363     uint32_t index = 0;
364     Mean = Sum / (float)n;
365     Range = max - min;
366
367     if (StartAtZero) {
368       for (uint32_t i = 0; i < n; i++) {
369         index = static_cast<uint32_t>(Data[i] / binRange);
370         if (index == noBins) {
371           index -= 1;
372         }
373         bins[index]++;
374         sum_dev += pow((Data[i] - Mean), 2);
375       }
376     } else {
377       for (uint32_t i = 0; i < n; i++) {
378         index = static_cast<uint32_t>((Data[i] - min) / binRange);
379         if (index == noBins) {
380           index -= 1;
381         }
382         bins[index]++;
383         sum_dev += pow((Data[i] - Mean), 2);
384       }
385     }
386     Std = sqrt((float)(sum_dev / n));
387     getCFD();
388     Calculated = true;
389   }
390
```

```
391   /*!
392    * \brief BasicCalculateFloat execute the basic float data calculations with a
393    * mask
394    * \param mask uchar mask type 0 don't calculate, 1 calculate
395    */
396   void BasicCalculateFloat(uchar *mask) {
397     float sum_dev = 0.0;
398     n = Rows * Cols;
399     uint32_t nmask = 0;
400     for (uint32_t i = 0; i < n; i++) {
401       if (mask[i] != 0) {
402         if (Data[i] > max) {
403           max = Data[i];
404         }
405         if (Data[i] < min) {
406           min = Data[i];
407         }
408         Sum += Data[i];
409         nmask++;
410       }
411     }
412     binRange = (max - min) / noBins;
413     uint32_t index = 0;
414     Mean = Sum / (float)nmask;
415     Range = max - min;
416     if (StartAtZero) {
417       for (uint32_t i = 0; i < n; i++) {
418         if (mask[i] != 0) {
419           index = static_cast<uint32_t>(Data[i] / binRange);
420           if (index == noBins) {
421             index -= 1;
422           }
423           bins[index]++;
424           sum_dev += pow((Data[i] - Mean), 2);
425         }
426       }
427     } else {
428       for (uint32_t i = 0; i < n; i++) {
429         if (mask[i] != 0) {
430           index = static_cast<uint32_t>((Data[i] - min) / binRange);
431           if (index == noBins) {
432             index -= 1;
433           }
434           bins[index]++;
435           sum_dev += pow((Data[i] - Mean), 2);
436         }
437       }
438     }
439     Std = sqrt((float)(sum_dev / nmask));
440     getCFD();
441     Calculated = true;
442   }
443
444   /*!
445    * \brief BasicCalculate execute the basic discrete data calculations
446    */
```

```
447   void BasicCalculate() {
448     double sum_dev = 0.0;
449     n = Rows * Cols;
450     for (uint32_t i = 0; i < n; i++) {
451       if (Data[i] > max) {
452         max = Data[i];
453       }
454       if (Data[i] < min) {
455         min = Data[i];
456       }
457       Sum += Data[i];
458     }
459     binRange = static_cast<T1>(ceil((max - min) / static_cast<float>(noBins)));
460     if (binRange == 0) {
461       binRange = 1;
462     }
463     Mean = Sum / (float)n;
464     Range = max - min;
465 
466     uint32_t index;
467     if (StartAtZero) {
468       std::for_each(Data, Data + n, [&](T1 &d) {
469         index = static_cast<uint32_t>(d / binRange);
470         if (index == noBins) {
471           index -= 1;
472         }
473         bins[index]++;
474         sum_dev += pow((d - Mean), 2);
475       });
476     } else {
477       std::for_each(Data, Data + n, [&](T1 &d) {
478         index = static_cast<uint32_t>((d - min) / binRange);
479         if (index == noBins) {
480           index -= 1;
481         }
482         bins[index]++;
483         sum_dev += pow((d - Mean), 2);
484       });
485     }
486     Std = sqrt((float)(sum_dev / n));
487     getCFD();
488     Calculated = true;
489   }
490 
491   /*!
492    * \brief BasicCalculate execute the basic discrete data calculations with
493    * mask
494    * \param mask uchar mask type 0 don't calculate, 1 calculate
495    */
496   void BasicCalculate(uchar *mask) {
497     double sum_dev = 0.0;
498     n = Rows * Cols;
499     uint32_t nmask = 0;
500     uint32_t i = 0;
501     std::for_each(Data, Data + n, [&](T1 &d) {
502       if (mask[i++] != 0) {
```

```
503           if (d > max) {
504             max = d;
505           }
506           if (d < min) {
507             min = d;
508           }
509           Sum += d;
510           nmask++;
511         }
512       });
513       binRange = static_cast<T1>(ceil((max - min) / static_cast<float>(noBins)));
514       Mean = Sum / (float)nmask;
515       Range = max - min;
516
517       uint32_t index;
518       if (StartAtZero) {
519         i = 0;
520         std::for_each(Data, Data + n, [&](T1 &d) {
521           if (mask[i++] != 0) {
522             index = static_cast<uint32_t>(d / binRange);
523             if (index == noBins) {
524               index -= 1;
525             }
526             bins[index]++;
527             sum_dev += pow((d - Mean), 2);
528           }
529         });
530       } else {
531         i = 0;
532         std::for_each(Data, Data + n, [&](T1 &d) {
533           if (mask[i++] != 0) {
534             index = static_cast<uint32_t>((d - min) / binRange);
535             if (index == noBins) {
536               index -= 1;
537             }
538             bins[index]++;
539             sum_dev += pow((d - Mean), 2);
540           }
541         });
542       }
543       Std = sqrt((float)(sum_dev / nmask));
544       getCFD();
545       Calculated = true;
546     }
547
548     /*!
549      * \brief BinCalculations excute the cacluations with the histogram
550      * \param startC start counter
551      * \param endC end counter
552      */
553     void BinCalculations(uint16_t startC, uint16_t endC __attribute__((unused))) {
554       float sum_dev = 0.0;
555       // Get the Sum
556       uint32_t i = 0;
557       for_each(begin(), end(), [&](uint32_t &b) { Sum += b * (startC + i++); });
558
```

```
559       // Get Mean
560       Mean = Sum / (float)n;
561
562       // Get max
563       for (int i = noBins - 1; i >= 0; i--) {
564         if (bins[i] != 0) {
565           max = i + startC;
566           break;
567         }
568       }
569
570       // Get min
571       for (uint32_t i = 0; i < noBins; i++) {
572         if (bins[i] != 0) {
573           min = i + startC;
574           break;
575         }
576       }
577
578       // Get Range;
579       Range = max - min;
580
581       // Calculate Standard Deviation
582       i = 0;
583       for_each(begin(), end(), [&](uint32_t &b) {
584         sum_dev += b * pow(((i++ + startC) - Mean), 2);
585       });
586       Std = sqrt((float)(sum_dev / n));
587       getCFD();
588       Calculated = true;
589     }
590
591     uint32_t HighestFrequency() {
592       uint32_t freq = 0;
593       std::for_each(begin(), end(), [&](uint32_t &B) {
594         if (B > freq) {
595           freq = B;
596         }
597       });
598       return freq;
599     }
600
601     void GetPDFfunction(std::vector<double> &xAxis, std::vector<double> &yAxis,
602                         double Step, double start = 0, double stop = 7) {
603       uint32_t resolution;
604       resolution = static_cast<uint32_t>(((stop - start) / Step) + 0.5);
605
606       xAxis.push_back(start);
607       double yVal0 = (1 / (Std * 2.506628274631)) *
608                      exp(-(pow((start - Mean), 2) / (2 * pow(Std, 2))));
609       yAxis.push_back(yVal0);
610       HighestPDF = yVal0;
611       for (uint32_t i = 1; i < resolution; i++) {
612         double xVal = xAxis[xAxis.size() - 1] + Step;
613         xAxis.push_back(xVal);
614         double yVal = (1 / (Std * 2.506628274631)) *
```

```
615                     exp(-(pow((xVal - Mean), 2) / (2 * pow(Std, 2))));
616       yAxis.push_back(yVal);
617       if (yVal > HighestPDF) {
618         HighestPDF = yVal;
619       }
620     }
621   }
622
623 protected:
624   uint32_t n_end = 0; /**< data end counter used with mask*/
625
626   /*!
627    * \brief getCFD get the CFD matrix;
628    */
629   void getCFD() {
630     uint32_t *sumBin = new uint32_t[noBins];
631     sumBin[0] = bins[0];
632     CFD[0] = (static_cast<double>(sumBin[0]) / static_cast<double>(n)) * 100.;
633     for (uint32_t i = 1; i < noBins; i++) {
634       sumBin[i] = (sumBin[i - 1] + bins[i]);
635       CFD[i] = (static_cast<double>(sumBin[i]) / static_cast<double>(n)) * 100.;
636       if (CFD[i] > HighestPDF) {
637         HighestPDF = CFD[i];
638       }
639     }
640     delete[] sumBin;
641   }
642
643   friend class boost::serialization::access; /**< Serialization class*/
644
645   /*!
646    * \brief serialize the object
647    * \param ar argument
648    * \param version
649    */
650   template <class Archive>
651   void serialize(Archive &ar, const unsigned int version) {
652     if (version == 0) {
653       ar &isDiscrete;
654       ar &n;
655       ar &noBins;
656       for (size_t dc = 0; dc < noBins; dc++) {
657         ar &bins[dc];
658       }
659       for (size_t dc = 0; dc < noBins; dc++) {
660         ar &CFD[dc];
661       }
662       for (size_t dc = 0; dc < noBins; dc++) {
663         ar &BinRanges[dc];
664       }
665       ar &Calculated;
666       ar &Mean;
667       ar &Range;
668       ar &min;
669       ar &max;
670       ar &Startbin;
```

```
671         ar &EndBin;
672         ar &binRange;
673         ar &Std;
674         ar &Sum;
675         ar &Rows;
676         ar &Cols;
677         ar &StartAtZero;
678         ar &HighestPDF;
679       }
680     }
681 };
682 }
683
684 typedef SoilMath::Stats<float, double, long double>
685       floatStat_t; /**< floating Stat type*/
686 typedef SoilMath::Stats<uchar, uint32_t, uint64_t>
687       ucharStat_t; /**< uchar Stat type*/
688 typedef SoilMath::Stats<uint16_t, uint32_t, uint64_t>
689       uint16Stat_t; /**< uint16 Stat type*/
690 typedef SoilMath::Stats<uint32_t, uint32_t, uint64_t>
691       uint32Stat_t; /**< uint32 Stat type*/
692 BOOST_CLASS_VERSION(floatStat_t, 0)
693 BOOST_CLASS_VERSION(ucharStat_t, 0)
694 BOOST_CLASS_VERSION(uint16Stat_t, 0)
695 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) {
```

```
56        ar &boost::serialization::base_object<
57            SoilMath::Stats<double, double, long double>>(*this);
58      }
59    }
60
61  public:
62    PSD() : SoilMath::Stats<double, double, long double>() {}
63
64    PSD(double *data, uint32_t nodata, double *binranges, uint32_t nobins,
65        uint32_t endbin)
66        : SoilMath::Stats<double, double, long double>(nobins, 0, endbin) {
67      std::copy(binranges, binranges + nobins, BinRanges);
68      Data = data;
69      Rows = nodata;
70      Cols = 1;
71
72      BasicCalculatePSD();
73    }
74  };
75  }
76  BOOST_CLASS_VERSION(SoilMath::PSD, 0)
```

A.0.5 General project file

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  /*! \brief Collection of the public SoilMath headers
9   * Commonpractice is to include this header when you want to add Soilmath
10  * routines
11  */
12 #pragma once
13
14 #include "Stats.h"
15 #include "Sort.h"
16 #include "FFT.h"
17 #include "NN.h"
18 #include "GA.h"
19 #include "CommonOperations.h"
20 #include "SoilMathTypes.h"
21 #include "psd.h"
22 #include "Mat_archive.h"
23 #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;
```

```
56    return u.d;
57  }
58
59  static inline double quick_pow2(int n) {
60    static double pow2[256] = {
61        0,     1,     4,     9,     16,    25,    36,    49,    64,
    81,
62        100,   121,   144,   169,   196,   225,   256,   289,   324,
    361,
63        400,   441,   484,   529,   576,   625,   676,   729,   784,
    841,
64        900,   961,   1024,  1089,  1156,  1225,  1296,  1369,  1444,
    1521,
65        1600,  1681,  1764,  1849,  1936,  2025,  2116,  2209,  2304,
    2401,
66        2500,  2601,  2704,  2809,  2916,  3025,  3136,  3249,  3364,
    3481,
67        3600,  3721,  3844,  3969,  4096,  4225,  4356,  4489,  4624,
    4761,
68        4900,  5041,  5184,  5329,  5476,  5625,  5776,  5929,  6084,
    6241,
69        6400,  6561,  6724,  6889,  7056,  7225,  7396,  7569,  7744,
    7921,
70        8100,  8281,  8464,  8649,  8836,  9025,  9216,  9409,  9604,
    9801,
71        10000, 10201, 10404, 10609, 10816, 11025, 11236, 11449, 11664, 11881,
72        12100, 12321, 12544, 12769, 12996, 13225, 13456, 13689, 13924, 14161,
73        14400, 14641, 14884, 15129, 15376, 15625, 15876, 16129, 16384, 16641,
74        16900, 17161, 17424, 17689, 17956, 18225, 18496, 18769, 19044, 19321,
75        19600, 19881, 20164, 20449, 20736, 21025, 21316, 21609, 21904, 22201,
76        22500, 22801, 23104, 23409, 23716, 24025, 24336, 24649, 24964, 25281,
77        25600, 25921, 26244, 26569, 26896, 27225, 27556, 27889, 28224, 28561,
78        28900, 29241, 29584, 29929, 30276, 30625, 30976, 31329, 31684, 32041,
79        32400, 32761, 33124, 33489, 33856, 34225, 34596, 34969, 35344, 35721,
80        36100, 36481, 36864, 37249, 37636, 38025, 38416, 38809, 39204, 39601,
81        40000, 40401, 40804, 41209, 41616, 42025, 42436, 42849, 43264, 43681,
82        44100, 44521, 44944, 45369, 45796, 46225, 46656, 47089, 47524, 47961,
83        48400, 48841, 49284, 49729, 50176, 50625, 51076, 51529, 51984, 52441,
84        52900, 53361, 53824, 54289, 54756, 55225, 55696, 56169, 56644, 57121,
85        57600, 58081, 58564, 59049, 59536, 60025, 60516, 61009, 61504, 62001,
86        62500, 63001, 63504, 64009, 64516, 65025};
87    return pow2[(n >= 0) ? n : -n];
88  }
89
90  static inline long float2intRound(double d) {
91    d += 6755399441055744.0;
92    return reinterpret_cast<int &>(d);
93  }
94
95  /*!
96   * \brief calcVolume according to ISO 9276-6
97   * \param A
98   * \return
99   */
100 static inline float calcVolume(float A) {
101   return (pow(A, 1.5)) / 10.6347f;
```

```
102 }
103
104 static inline std::vector<float> makeOutput(uint8_t value, uint32_t noNeurons) {
105   std::vector<float> retVal(noNeurons, -1);
106   retVal[value - 1] = 1;
107   return retVal;
108 }
109
110 /*!
111  * \brief calcDiameter according to ISO 9276-6
112  * \param A
113  * \return
114  */
115 static inline float calcDiameter(float A) {
116   //return sqrt((4 * A) / M_PI);
117   return 1.1283791670955 * sqrt(A);
118 }
119 }
```

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7  #pragma once
8
9  #define GENE_MAX 32  /**< maximum number of genes*/
10 #define CROSSOVER 16 /**< crossover location*/
11
12 #include <stdint.h>
13 #include <bitset>
14 #include <vector>
15 #include <complex>
16 #include <valarray>
17 #include <array>
18
19 typedef unsigned char uchar;   /**< unsigned char*/
20 typedef unsigned short ushort; /**< unsigned short*/
21 typedef unsigned int uint32_t;
22
23 typedef std::complex<double> Complex_t;       /**< complex vector of doubles*/
24 typedef std::vector<Complex_t> ComplexVect_t; /**< vector of Complex_t*/
25 typedef std::valarray<Complex_t> ComplexArray_t; /**< valarray of Complex_t*/
26 typedef std::vector<uint32_t> iContour_t;        /**< vector of uint32_t*/
27 typedef std::bitset<GENE_MAX> Genome_t; /**< Bitset repressenting a genome*/
28 typedef std::pair<std::bitset<CROSSOVER>, std::bitset<GENE_MAX - CROSSOVER>>
29     SplitGenome_t; /**< a matted genome*/
30
31 typedef std::vector<float> Weight_t;     /**< a float vector*/
32 typedef std::vector<Genome_t> GenVect_t; /**< a vector of genomes*/
33 typedef struct PopMemberStruct {
34   Weight_t weights;       /**< the weights the core of a population member*/
35   GenVect_t weightsGen;   /**< the weights as genomes*/
36   float Calculated = 0.0; /**< the calculated value*/
37   float Fitness = 0.0;    /**< the fitness of the population member*/
38 } PopMember_t;            /**< a population member*/
39 typedef std::vector<PopMember_t> Population_t; /**< Vector with PopMember_t*/
40 typedef std::pair<float, float>
41     MinMaxWeight_t; /**< floating pair weight range*/
42
43 typedef struct Predict_struct {
44   uint8_t Category = 1; /**< the category number */
45   float RealValue = 1.;  /**< category number as float in order to estimate how
46                         precise to outcome is*/
47   float Accuracy = 1.;   /**< the accuracy of the category*/
48   std::vector<float> OutputNeurons; /**< the output Neurons*/
49   bool ManualSet = true;
50 } Predict_t;                        /**< The prediction results*/
51 typedef Predict_t (*NNfunctionType)(
52     ComplexVect_t, Weight_t, Weight_t, uint32_t, uint32_t,
53     uint32_t); /**< The prediction function from the Neural Net*/
54
55 typedef std::vector<ComplexVect_t>
```

```
56      InputLearnVector_t; /**< Vector of a vector with complex values*/
57  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-bo
#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 __attrib
  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-bo
#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 __attr
  ar &P.Accuracy;
  ar &P.Category;
  ar &P.OutputNeurons;
  ar &P.RealValue;
}
}
}
```

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  #define EXCEPTION_MATH "Math Exception!"
9  #define EXCEPTION_MATH_NR 0
10 #define EXCEPTION_NO_CONTOUR_FOUND
   \
11   "No continuous contour found, or less then 8 pixels long!"
12 #define EXCEPTION_NO_CONTOUR_FOUND_NR 1
13 #define EXCEPTION_SIZE_OF_INPUT_NEURONS
   \
14   "Size of input unequal to input neurons exception!"
15 #define EXCEPTION_SIZE_OF_INPUT_NEURONS_NR 2
16 #define EXCEPTION_NEURAL_NET_NOT_STUDIED "Neural net didn't study exception!"
17 #define EXCEPTION_NEURAL_NET_NOT_STUDIED_NR 3
18 #define EXCEPTION_TYPE_NOT_SUPPORTED
   \
19   "Type not supported for operation exception!"
20 #define EXCEPTION_TYPE_NOT_SUPPORTED_NR 4
21
22 #pragma once
23 #include <exception>
24 #include <string>
25
26 namespace SoilMath {
27 namespace Exception {
28 class MathException : public std::exception {
29 public:
30   MathException(std::string m = EXCEPTION_MATH, int n = EXCEPTION_MATH_NR)
31       : msg(m), nr(n){};
32   ~MathException() _GLIBCXX_USE_NOEXCEPT{};
33   const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };
34   const int *id() const _GLIBCXX_USE_NOEXCEPT { return &nr; }
35
36 private:
37   std::string msg;
38   int nr;
39 };
40 }
41 }
```

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  #pragma once
9  #include <stdint.h>
10
11 namespace SoilMath {
12 /*!
13  * \brief The Sort template class
14  */
15 class Sort {
16 public:
17   Sort() {}
18   ~Sort() {}
19
20   /*!
21    * \brief QuickSort a static sort a Type T array with i values
22    * \details Usage: QuickSort<type>(*type , i)
23    * \param arr an array of Type T
24    * \param i the number of elements
25    */
26   template <typename T> static void QuickSort(T *arr, int i) {
27     if (i < 2)
28       return;
29
30     T p = arr[i / 2];
31     T *l = arr;
32     T *r = arr + i - 1;
33     while (l <= r) {
34       if (*l < p) {
35         l++;
36       } else if (*r > p) {
37         r--;
38       } else {
39         T t = *l;
40         *l = *r;
41         *r = t;
42         l++;
43         r--;
44       }
45     }
46     Sort::QuickSort<T>(arr, r - arr + 1);
47     Sort::QuickSort<T>(l, arr + i - l);
48   }
49
50   /*!
51    * \brief QuickSort a static sort a Type T array with i values where the key
52    * are also changed accordingly
53    * \details Usage: QuickSort<type>(*type *type , i)
54    * \param arr an array of Type T
55    * \param key an array of 0..i-1 representing the index
```

```
56      * \param i the number of elements
57      */
58    template <typename T> static void QuickSort(T *arr, T *key, int i) {
59      if (i < 2)
60        return;
61
62      T p = arr[i / 2];
63
64      T *l = arr;
65      T *r = arr + i - 1;
66
67      T *lkey = key;
68      T *rkey = key + i - 1;
69
70      while (l <= r) {
71        if (*l < p) {
72          l++;
73          lkey++;
74        } else if (*r > p) {
75          r--;
76          rkey--;
77        } else {
78         if (*l != *r) {
79            T t = *l;
80            *l = *r;
81            *r = t;
82
83            T tkey = *lkey;
84            *lkey = *rkey;
85            *rkey = tkey;
86          }
87
88          l++;
89          r--;
90
91          lkey++;
92          rkey--;
93        }
94      }
95      Sort::QuickSort<T>(arr, key, r - arr + 1);
96      Sort::QuickSort<T>(l, lkey, arr + i - l);
97    }
98  };
99  }
```

B. Hardware Library

B.0.1 Microscope Class

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  /*! \class Microscope
9  Interaction with the USB 5 MP microscope
10 */
11
12 #pragma once
13
14 #include <stdint.h>
15 #include <vector>
16 #include <string>
17 #include <utility>
18 #include <algorithm>
19
20 #include <sys/stat.h>
21 #include <sys/utsname.h>
22 #include <sys/ioctl.h>
23 #include <fstream>
24 #include <fcntl.h>
25
26 #include <linux/videodev2.h>
27 #include <linux/v4l2-controls.h>
28 #include <linux/v4l2-common.h>
29
30 #include <boost/filesystem.hpp>
31 #include <boost/regex.hpp>
```

```
32
33 #include <opencv2/photo.hpp>
34 #include <opencv2/imgcodecs.hpp>
35 #include <opencv2/opencv.hpp>
36 #include <opencv2/core.hpp>
37
38 #include "MicroscopeNotFoundException.h"
39 #include "CouldNotGrabImageException.h"
40
41 namespace Hardware {
42 class Microscope {
43 public:
44   enum Arch { ARM, X64 };
45
46   enum PixelFormat { YUYV, MJPG };
47
48   struct Resolution_t {
49     uint16_t Width = 2048;
50     uint16_t Height = 1536;
51     PixelFormat format = PixelFormat::MJPG;
52     std::string to_string() {
53       std::string retVal = std::to_string(Width);
54       retVal.append(" x ");
55       retVal.append(std::to_string(Height));
56       if (format == PixelFormat::MJPG) {
57           retVal.append(" - MJPG");
58         }
59       else {
60           retVal.append(" - YUYV");
61         }
62       return retVal;
63     }
64     uint32_t ID;
65   };
66
67   struct Control_t {
68     std::string name;
69     int minimum;
70     int maximum;
71     int step;
72     int default_value;
73     int current_value;
74     uint32_t ID = V4L2_CID_BASE;
75     bool operator==(Control_t &rhs) {
76       if (this->name.compare(rhs.name) == 0) {
77         return true;
78       } else {
79         return false;
80       }
81     }
82     bool operator!=(Control_t &rhs) {
83       if (this->name.compare(rhs.name) != 0) {
84         return true;
85       } else {
86         return false;
87       }
```

```
88      }
89    };
90
91    typedef std::vector<Control_t> Controls_t;
92
93    struct Cam_t {
94      std::string Name;
95      std::string devString;
96      uint32_t ID;
97      std::vector<Resolution_t> Resolutions;
98      uint32_t delaytrigger = 1;
99      Resolution_t *SelectedResolution = nullptr;
100     Controls_t Controls;
101     int fd;
102     bool operator==(Cam_t const &rhs) {
103       if (this->ID == rhs.ID || this->Name == rhs.Name) {
104         return true;
105       } else {
106         return false;
107       }
108     }
109     bool operator!=(Cam_t const &rhs) {
110       if (this->ID != rhs.ID && this->Name != rhs.Name) {
111         return true;
112       } else {
113         return false;
114       }
115     }
116   };
117
118   std::vector<Cam_t> AvailableCams;
119   Cam_t *SelectedCam = nullptr;
120   Arch RunEnv;
121
122   Microscope();
123   Microscope(const Microscope &rhs);
124
125   ~Microscope();
126
127   Microscope operator=(Microscope const &rhs);
128
129   bool IsOpened();
130   bool openCam(Cam_t *cam);
131   bool openCam(int &cam);
132   bool openCam(std::string &cam);
133
134   bool closeCam(Cam_t *cam);
135
136   void GetFrame(cv::Mat &dst);
137   void GetHDRFrame(cv::Mat &dst, uint32_t noframes = 3);
138
139   Control_t *GetControl(const std::string name);
140   void SetControl(Control_t *control);
141
142   Cam_t *FindCam(std::string cam);
143   Cam_t *FindCam(int cam);
```

```
144
145 private:
146   cv::VideoCapture *cap = nullptr;
147
148   std::vector<cv::Mat> HDRframes;
149
150   std::vector<Cam_t> GetAvailableCams();
151   Arch GetCurrentArchitecture();
152   int fd;
153 };
154 }
```

```
/* 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) {
```

```
56      string videoln = itr->path().string();
57      videoln.append("/name");
58      if (boost::filesystem::exists(videoln)) {
59        Cam_t currentCam;
60        std::ifstream camName;
61        camName.open(videoln);
62        std::getline(camName, currentCam.Name);
63        camName.close();
64        currentCam.ID =
65            std::atoi(itr->path().string().substr(28, std::string::npos).c_str());
66
67        // Open Cam
68        currentCam.devString = path_ss_dev + std::to_string(currentCam.ID);
69        if ((currentCam.fd = open(currentCam.devString.c_str(), O_RDWR)) == -1) {
70          throw Exception::MicroscopeException(EXCEPTION_NOCAMS,
71                                               EXCEPTION_NOCAMS_NR);
72        }
73
74        // Get controls
75        memset(&queryctrl, 0, sizeof(queryctrl));
76        memset(&controlctrl, 0, sizeof(controlctrl));
77        for (queryctrl.id = V4L2_CID_BASE; queryctrl.id < V4L2_CID_LASTP1;
78             queryctrl.id++) {
79
80          if (ioctl(currentCam.fd, VIDIOC_QUERYCTRL, &queryctrl) == 0) {
81            if (!(queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)) {
82              Control_t currentControl;
83              currentControl.ID = queryctrl.id;
84              currentControl.name = (char *)queryctrl.name;
85              currentControl.minimum = queryctrl.minimum;
86              currentControl.maximum = queryctrl.maximum;
87              currentControl.default_value = queryctrl.default_value;
88              currentControl.step = queryctrl.step;
89              controlctrl.id = queryctrl.id;
90              if (ioctl(currentCam.fd, VIDIOC_G_CTRL, &controlctrl) == 0) {
91                currentControl.current_value = controlctrl.value;
92              }
93              currentCam.Controls.push_back(currentControl);
94            }
95          } else {
96            if (errno == EINVAL)
97              continue;
98            throw Exception::MicroscopeException(EXCEPTION_QUERY,
99                                                 EXCEPTION_QUERY_NR);
100         }
101       }
102
103       // Get image formats
104       struct v4l2_format format;
105       memset(&format, 0, sizeof(format));
106
107       uint32_t width[5] = {640, 800, 1280, 1600, 2048};
108       uint32_t height[6] = {480, 600, 960, 1200, 1536};
109
110       uint32_t ResolutionID = 0;
111
```

```
112         // YUYV
113         for (uint32_t i = 0; i < 5; i++) {
114           format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
115           format.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
116           format.fmt.pix.width = width[i];
117           format.fmt.pix.height = height[i];
118           int ret = ioctl(currentCam.fd, VIDIOC_S_FMT, &format);
119           if (ret != -1 && format.fmt.pix.height == height[i] &&
120               format.fmt.pix.width == width[i]) {
121             Resolution_t res;
122             res.Width = format.fmt.pix.width;
123             res.Height = height[i];
124             res.ID = ResolutionID++;
125             res.format = PixelFormat::YUYV;
126             currentCam.Resolutions.push_back(res);
127           }
128         }
129
130         // MJPEG
131         for (uint32_t i = 0; i < 5; i++) {
132           format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
133           format.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
134           format.fmt.pix.width = width[i];
135           format.fmt.pix.height = height[i];
136           int ret = ioctl(currentCam.fd, VIDIOC_S_FMT, &format);
137           if (ret != -1 && format.fmt.pix.height == height[i] &&
138               format.fmt.pix.width == width[i]) {
139             Resolution_t res;
140             res.Width = format.fmt.pix.width;
141             res.Height = format.fmt.pix.height;
142             res.ID = ResolutionID++;
143             res.format = PixelFormat::MJPG;
144             currentCam.Resolutions.push_back(res);
145           }
146         }
147
148         close(currentCam.fd);
149         retVal.push_back(currentCam);
150       }
151     }
152
153     for (uint32_t i = 0; i < retVal.size(); i++) {
154       if (retVal[i].Resolutions.size() == 0) {
155         retVal.erase(retVal.begin() + i);
156         i--;
157       }
158     }
159
160     return retVal;
161 }
162
163 bool Microscope::IsOpened() {
164     if (cap == nullptr) {
165       return false;
166     } else {
167       return cap->isOpened();
```

```
168   }
169 }
170
171 bool Microscope::openCam(Cam_t *cam) {
172   for (uint32_t i = 0; i < AvailableCams.size(); i++) {
173     if (AvailableCams[i] == *cam) {
174       closeCam(SelectedCam);
175       SelectedCam = cam;
176       cap = new cv::VideoCapture(SelectedCam->ID);
177       if (!cap->isOpened()) {
178         throw Exception::MicroscopeException(EXCEPTION_NOCAMS,
179                                              EXCEPTION_NOCAMS_NR);
180       }
181       cap->set(CV_CAP_PROP_FRAME_WIDTH, SelectedCam->SelectedResolution->Width);
182       cap->set(CV_CAP_PROP_FRAME_HEIGHT,
183                SelectedCam->SelectedResolution->Height);
184       for (Controls_t::iterator it = SelectedCam->Controls.begin();
185            it != SelectedCam->Controls.end(); ++it) {
186         SetControl(&*it);
187       }
188       return true;
189     }
190   }
191   return false;
192 }
193
194 bool Microscope::openCam(std::string &cam) { return openCam(FindCam(cam)); }
195
196 bool Microscope::openCam(int &cam) { return openCam(FindCam(cam)); }
197
198 Microscope::Cam_t *Microscope::FindCam(int cam) {
199   for (uint32_t i = 0; i < AvailableCams.size(); i++) {
200     if (cam == AvailableCams[i].ID) {
201       return &AvailableCams[i];
202     }
203   }
204   return nullptr;
205 }
206
207 Microscope::Cam_t *Microscope::FindCam(string cam) {
208   for (uint32_t i = 0; i < AvailableCams.size(); i++) {
209     if (cam.compare(AvailableCams[i].Name) == 0) {
210       return &AvailableCams[i];
211     }
212   }
213   return nullptr;
214 }
215
216 bool Microscope::closeCam(Cam_t *cam) {
217   if (cap != nullptr) {
218     if (cap->isOpened()) {
219       cap->release();
220     }
221     delete cap;
222     cap = nullptr;
223   }
```

```
224 }
225
226 void Microscope::GetFrame(cv::Mat &dst) {
227   openCam(SelectedCam);
228   sleep(SelectedCam->delaytrigger);
229   if (RunEnv == Arch::ARM) {
230     for (uint32_t i = 0; i < 2; i++) {
231       if (!cap->grab()) {
232         throw Exception::CouldNotGrabImageException();
233       }
234       sleep(SelectedCam->delaytrigger);
235     }
236     cap->retrieve(dst);
237   } else {
238     for (uint32_t i = 0; i < 2; i++) {
239       if (!cap->read(dst)) {
240         throw Exception::CouldNotGrabImageException();
241       }
242     }
243   }
244 }
245
246 void Microscope::GetHDRFrame(cv::Mat &dst, uint32_t noframes) {
247   // create the brightness steps
248   Control_t *brightness = GetControl("Brightness");
249   Control_t *contrast = GetControl("Contrast");
250
251   uint32_t brightnessStep =
252       (brightness->maximum - brightness->minimum) / noframes;
253   int8_t currentBrightness = brightness->current_value;
254   int8_t currentContrast = contrast->current_value;
255   contrast->current_value = contrast->maximum;
256
257   cv::Mat currentImg;
258   // take the shots at different brightness levels
259   for (uint32_t i = 1; i <= noframes; i++) {
260     brightness->current_value = brightness->minimum + (i * brightnessStep);
261     GetFrame(currentImg);
262     HDRframes.push_back(currentImg);
263   }
264
265   // Set the brightness and back to the previous used level
266   brightness->current_value = currentBrightness;
267   contrast->current_value = currentContrast;
268
269   // Perform the exposure fusion
270   cv::Mat fusion;
271   cv::Ptr<cv::MergeMertens> merge_mertens = cv::createMergeMertens();
272   merge_mertens->process(HDRframes, fusion);
273   fusion *= 255;
274   fusion.convertTo(dst, CV_8UC1);
275 }
276
277 Microscope::Control_t *Microscope::GetControl(const string name) {
278   for (Controls_t::iterator it = SelectedCam->Controls.begin();
279        it != SelectedCam->Controls.end(); ++it) {
```

```
280       if (name.compare(it->name) == 0) {
281         return &*it;
282       }
283     }
284     return nullptr;
285 }
286
287 void Microscope::SetControl(Control_t *control) {
288     if ((SelectedCam->fd = open(SelectedCam->devString.c_str(), O_RDWR)) == -1) {
289       throw Exception::MicroscopeException(EXCEPTION_NOCAMS, EXCEPTION_NOCAMS_NR);
290     }
291
292     struct v4l2_queryctrl queryctrl;
293     struct v4l2_control controlctrl;
294
295     memset(&queryctrl, 0, sizeof(queryctrl));
296     queryctrl.id = control->ID;
297     if (ioctl(SelectedCam->fd, VIDIOC_QUERYCTRL, &queryctrl) == -1) {
298       if (errno != EINVAL) {
299         close(SelectedCam->fd);
300         throw Exception::MicroscopeException(EXCEPTION_QUERY, EXCEPTION_QUERY_NR);
301       } else {
302         close(SelectedCam->fd);
303         throw Exception::MicroscopeException(EXCEPTION_CTRL_NOT_FOUND,
304                                              EXCEPTION_CTRL_NOT_FOUND_NR);
305       }
306     } else if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED) {
307       close(SelectedCam->fd);
308       throw Exception::MicroscopeException(EXCEPTION_CTRL_NOT_FOUND,
309                                            EXCEPTION_CTRL_NOT_FOUND_NR);
310     } else {
311       memset(&controlctrl, 0, sizeof(controlctrl));
312       controlctrl.id = control->ID;
313       controlctrl.value = control->current_value;
314
315       if (ioctl(SelectedCam->fd, VIDIOC_S_CTRL, &controlctrl) == -1) {
316         // Fails on auto white balance
317         // throw Exception::MicroscopeException(EXCEPTION_CTRL_VALUE,
318         //                                      EXCEPTION_CTRL_VALUE_NR);
319       }
320     }
321     close(SelectedCam->fd);
322 }
323 }
```

B.0.2 Beaglebone Black Class

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  /*! \class BBB
9  The core BeagleBone Black class used for all hardware related classes.
10 Consisting of universal used method, functions and variables. File operations,
11 polling and threading
12 */
13
14 #pragma once
15
16 #define SLOTS
   \
17   "/sys/devices/bone_capemgr.9/slots" /*!< Beaglebone capemanager slots file*/
18
19 #include <fstream>
20 #include <sstream>
21 #include <string>
22 #include <sys/stat.h>
23 #include <pthread.h>
24 #include <unistd.h>
25 #include <sys/epoll.h>
26 #include <fcntl.h>
27 #include <regex>
28 #include <stdexcept>
29
30 #include "GPIOReadException.h"
31 #include "FailedToCreateGPIOPollingThreadException.h"
32 #include "ValueOutOfBoundsException.h"
33
34 using namespace std;
35
36 namespace Hardware {
37 typedef int (*CallbackType)(
38     int); /*!< CallbackType used to pass a function to a thread*/
39
40 class BBB {
41 public:
42   int debounceTime; /*!< debounce time for a button in milliseconds*/
43
44   BBB();
45   ~BBB();
46
47 protected:
48   bool threadRunning;             /*!< used to stop the thread*/
49   pthread_t thread;               /*!< The thread*/
50   CallbackType callbackFunction; /*!< the callbakcfunction*/
51
52   bool DirectoryExist(const string &path);
53   bool CapeLoaded(const string &shield);
```

```
54
55    string Read(const string &path);
56    void Write(const string &path, const string &value);
57
58    /*! Converts a number to a string
59    \param Number as typename
60    \returns the number as a string
61    */
62    template <typename T> string NumberToString(T Number) {
63      ostringstream ss;
64      ss << Number;
65      return ss.str();
66    };
67
68    /*! Converts a string to a number
69    \param Text the string that needs to be converted
70    \return the number as typename
71    */
72    template <typename T> T StringToNumber(string Text) {
73      stringstream ss(Text);
74      T result;
75      return ss >> result ? result : 0;
76    };
77  };
78  }
```

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  #include "BBB.h"
9
10 namespace Hardware {
11 /*! Constructor*/
12 BBB::BBB() {
13   threadRunning = false;
14   callbackFunction = NULL;
15   debounceTime = 0;
16   thread = (pthread_t)NULL;
17 }
18
19 /*! De-constructor*/
20 BBB::~BBB() {}
21
22 /*! Reads the first line from a file
23 \param path constant string pointing towards the file
24 \returns this first line
25 */
26 string BBB::Read(const string &path) {
27   ifstream fs;
28   fs.open(path.c_str());
29   if (!fs.is_open()) {
30     throw Exception::GPIOReadException(("Can't open: " + path).c_str());
31   }
32   string input;
33   getline(fs, input);
34   fs.close();
35   return input;
36 }
37
38 /*! Writes a value to a file
39 \param path a constant string pointing towards the file
40 \param value a constant string which should be written in the file
41 */
42 void BBB::Write(const string &path, const string &value) {
43   ofstream fs;
44   fs.open(path.c_str());
45   if (!fs.is_open()) {
46     throw Exception::GPIOReadException(("Can't open: " + path).c_str());
47   }
48   fs << value;
49   fs.close();
50 }
51
52 /*! Checks if a directory exist
53 \returns true if the directory exists and false if not
54 */
55 bool BBB::DirectoryExist(const string &path) {
```

```
56    struct stat st;
57    if (stat((char *)path.c_str(), &st) != 0) {
58      return false;
59    }
60    return true;
61  }
62
63  /*! Checks if a cape is loaded in the file /sys/devices/bone_capemgr.9/slots
64  \param shield a const search string which is a (part) of the shield name
65  \return true if the search string is found otherwise false
66  */
67  bool BBB::CapeLoaded(const string &shield) {
68    bool shieldFound = false;
69
70    ifstream fs;
71    fs.open(SLOTS);
72    if (!fs.is_open()) {
73      throw Exception::GPIOReadException("Can't open SLOTS");
74    }
75
76    string line;
77    while (getline(fs, line)) {
78      if (line.find(shield) != string::npos) {
79        shieldFound = true;
80        break;
81      }
82    }
83    fs.close();
84    return shieldFound;
85  }
86  }
```

B.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);

```

```
55    bool isExported(int number, Direction &dir, Edge &edge);
56    bool ExportPin(int number);
57    bool UnexportPin(int number);
58
59    Direction ReadsDirection(const string &gpiopath);
60    void WritesDirection(const string &gpiopath, Direction direction);
61
62    Edge ReadsEdge(const string &gpiopath);
63    void WritesEdge(const string &gpiopath, Edge edge);
64
65    Value ReadsValue(const string &gpiopath);
66    void WritesValue(const string &gpiopath, Value value);
67 };
68
69 void *threadedPollGPIO(void *value);
70 }
```

```
/* 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;

```

```
56    if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) {
57      throw Exception::FailedToCreateGPIOPollingThreadException(
58          "GPIO: Failed to add control interface!");
59    }
60
61    while (count <= 1) {
62      i = epoll_wait(epollfd, &ev, 1, -1);
63      if (i == -1) {
64        close(fd);
65        return -1;
66      } else {
67        count++;
68      }
69    }
70    close(fd);
71    return 0;
72  }
73
74  GPIO::Value GPIO::GetValue() { return ReadsValue(gpiopath); }
75  void GPIO::SetValue(GPIO::Value value) { WritesValue(gpiopath, value); }
76
77  GPIO::Direction GPIO::GetDirection() { return direction; }
78  void GPIO::SetDirection(Direction direction) {
79    this->direction = direction;
80    WritesDirection(gpiopath, direction);
81  }
82
83  GPIO::Edge GPIO::GetEdge() { return edge; }
84  void GPIO::SetEdge(Edge edge) {
85    this->edge = edge;
86    WritesEdge(gpiopath, edge);
87  }
88
89  bool GPIO::isExported(int number __attribute__((unused)), Direction &dir, Edge &ed
90    // Checks if directory exist and therefore is exported
91    if (!DirectoryExist(gpiopath)) {
92      return false;
93    }
94
95    // Reads the data associated with the pin
96    dir = ReadsDirection(gpiopath);
97    edge = ReadsEdge(gpiopath);
98    return true;
99  }
100
101 bool GPIO::ExportPin(int number) {
102   Write(EXPORT_PIN, NumberToString<int>(number));
103   usleep(250000);
104 }
105
106 bool GPIO::UnexportPin(int number) {
107   Write(UNEXPORT_PIN, NumberToString<int>(number));
108 }
109
110 GPIO::Direction GPIO::ReadsDirection(const string &gpiopath) {
111   if (Read(gpiopath + DIRECTION) == "in") {
```

```
112       return Input;
113     } else {
114       return Output;
115     }
116   }
117
118   void GPIO::WritesDirection(const string &gpiopath, Direction direction) {
119     switch (direction) {
120     case Hardware::GPIO::Input:
121       Write((gpiopath + DIRECTION), "in");
122       break;
123     case Hardware::GPIO::Output:
124       Write((gpiopath + DIRECTION), "out");
125       break;
126     }
127   }
128
129   GPIO::Edge GPIO::ReadsEdge(const string &gpiopath) {
130     string reader = Read(gpiopath + EDGE);
131     if (reader == "none") {
132       return None;
133     } else if (reader == "rising") {
134       return Rising;
135     } else if (reader == "falling") {
136       return Falling;
137     } else {
138       return Both;
139     }
140   }
141
142   void GPIO::WritesEdge(const string &gpiopath, Edge edge) {
143     switch (edge) {
144     case Hardware::GPIO::None:
145       Write((gpiopath + EDGE), "none");
146       break;
147     case Hardware::GPIO::Rising:
148       Write((gpiopath + EDGE), "rising");
149       break;
150     case Hardware::GPIO::Falling:
151       Write((gpiopath + EDGE), "falling");
152       break;
153     case Hardware::GPIO::Both:
154       Write((gpiopath + EDGE), "both");
155       break;
156     default:
157       break;
158     }
159   }
160
161   GPIO::Value GPIO::ReadsValue(const string &gpiopath) {
162     string path(gpiopath + VALUE);
163     int res = StringToNumber<int>(Read(path));
164     return (Value)res;
165   }
166
167   void GPIO::WritesValue(const string &gpiopath, Value value) {
```

```
168   Write(gpiopath + VALUE, NumberToString<int>(value));
169 }
170
171 void *threadedPollGPIO(void *value) {
172   GPIO *gpio = static_cast<GPIO *>(value);
173   while (gpio->threadRunning) {
174     gpio->callbackFunction(gpio->WaitForEdge());
175     usleep(gpio->debounceTime * 1000);
176   }
177   return 0;
178 }
179 }
```

B.0.4 PWM 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 "BBB.h"
#include <dirent.h>

#define OCP_PATH "/sys/devices/ocp.3/"
#define P8_13_FIND "bs_pwm_test_P8_13"
#define P8_19_FIND "bs_pwm_test_P8_19"
#define P9_14_FIND "bs_pwm_test_P9_14"
#define P9_16_FIND "bs_pwm_test_P9_16"

#define PWM_CAPE "am33xx_pwm"
#define P8_13_CAPE "bspwm_P8_13" //_14
#define P8_19_CAPE "bspwm_P8_13" //_14
#define P9_14_CAPE "bspwm_P9_14" //_16
#define P9_16_CAPE "bspwm_P9_16" //_16

#define P8_13_CAPE_LOAD "bspwm_P8_13_14"
#define P8_19_CAPE_LOAD "bspwm_P8_13_14"
#define P9_14_CAPE_LOAD "bspwm_P9_14_16"
#define P9_16_CAPE_LOAD "bspwm_P9_16_16"

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);

```

```
55    int GetDuty() { return duty; };
56    void SetDuty(int value);
57    void SetIntensity();
58
59    Run GetRun() { return run; };
60    void SetRun(Run value);
61
62    Polarity GetPolarity() { return polarity; };
63    void SetPolarity(Polarity value);
64
65    PWM(Pin pin);
66    ~PWM();
67
68  private:
69    int period;          // current period
70    int duty;            // current duty
71    float intensity;     // current intensity
72    uint8_t pixelvalue; // current pixelvalue
73    Run run;             // current run state
74    Polarity polarity;  // current polaity
75
76    string basepath;      // the basepath ocp.3
77    string dutypath;      // base + duty path
78    string periodpath;    // base + period path
79    string runpath;       // base + run path
80    string polaritypath; // base + polarity path
81
82    void calcIntensity();
83    string FindPath(string value);
84  };
85  }
```

```
/* 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
  basepath = OCP_PATH;
  switch (pin) {
  case Hardware::PWM::P8_13:
    if (!CapeLoaded(P8_13_CAPE)) {
      Write(SLOTS, P8_13_CAPE_LOAD);
    }
    basepath.append(FindPath(P8_13_FIND));
    break;
  case Hardware::PWM::P8_19:
    if (!CapeLoaded(P8_19_CAPE)) {
      Write(SLOTS, P8_19_CAPE_LOAD);
    }
    basepath.append(FindPath(P8_19_FIND));
    break;
  case Hardware::PWM::P9_14:
    if (!CapeLoaded(P9_14_CAPE)) {
      Write(SLOTS, P9_14_CAPE_LOAD);
    }
    basepath.append(FindPath(P9_14_FIND));
    break;
  case Hardware::PWM::P9_16:
    if (!CapeLoaded(P9_16_CAPE)) {
      Write(SLOTS, P9_16_CAPE_LOAD);
    }
    basepath.append(FindPath(P9_16_FIND));
    break;
  }

  // Get the working paths
  dutypath = basepath + "/duty";
  periodpath = basepath + "/period";
  runpath = basepath + "/run";
```

```
56    polaritypath = basepath + "/polarity";
57
58    // Give Linux time to setup directory structure;
59    usleep(250000);
60
61    // Read current values
62    period = StringToNumber<int>(Read(periodpath));
63    duty = StringToNumber<int>(Read(dutypath));
64    run = static_cast<Run>(StringToNumber<int>(Read(runpath)));
65    polarity = static_cast<Polarity>(StringToNumber<int>(Read(polaritypath)));
66
67    // calculate the current intensity
68    calcIntensity();
69  }
70
71  PWM::~PWM() {}
72
73  /// <summary>
74  /// Calculate the current intensity
75  /// </summary>
76  void PWM::calcIntensity() {
77    if (polarity == Normal) {
78      if (duty == 0) {
79        intensity = 0.0f;
80      } else {
81        intensity = (float)period / (float)duty;
82      }
83    } else {
84      if (period == 0) {
85        intensity = 0.0f;
86      } else {
87        intensity = (float)duty / (float)period;
88      }
89    }
90  }
91
92  /// <summary>
93  /// Set the intensity level as percentage
94  /// </summary>
95  /// <param name="value">floating value multipication factor</param>
96  void PWM::SetIntensity(float value) {
97    if (polarity == Normal) {
98      SetDuty(static_cast<int>((value * duty) + 0.5));
99    } else {
100     SetPeriod(static_cast<int>((value * period) + 0.5));
101   }
102 }
103
104 /// <summary>
105 /// Set the output as a corresponding uint8_t value
106 /// </summary>
107 /// <param name="value">pixel value 0-255</param>
108 void PWM::SetPixelValue(uint8_t value) {
109   if (period != 255) {
110     SetPeriod(255);
111   }
```

```
112     SetDuty(255 - value);
113     pixelvalue = value;
114   }
115
116   /// <summary>
117   /// Set the period of the signal
118   /// </summary>
119   /// <param name="value">period : int</param>
120   void PWM::SetPeriod(int value) {
121     string valstr = NumberToString<int>(value);
122     Write(periodpath, valstr);
123     period = value;
124
125     calcIntensity();
126   }
127
128   /// <summary>
129   /// Set the duty of the signal
130   /// </summary>
131   /// <param name="value">duty : int</param>
132   void PWM::SetDuty(int value) {
133     string valstr = NumberToString<int>(value);
134     Write(dutypath, valstr);
135     duty = value;
136
137     calcIntensity();
138   }
139
140   /// <summary>
141   /// Run the signal
142   /// </summary>
143   /// <param name="value">On or Off</param>
144   void PWM::SetRun(Run value) {
145     int valInt = static_cast<int>(value);
146     string valstr = NumberToString<int>(valInt);
147     Write(runpath, valstr);
148     run = value;
149   }
150
151   /// <summary>
152   /// Set the polarity
153   /// </summary>
154   /// <param name="value">Normal or Inverted signal</param>
155   void PWM::SetPolarity(Polarity value) {
156     int valInt = static_cast<int>(value);
157     string valstr = NumberToString<int>(valInt);
158     Write(runpath, valstr);
159     polarity = value;
160   }
161
162   /// <summary>
163   /// Find the current PWM path in the OCP.3 directory
164   /// </summary>
165   /// <param name="value">part a the path name</param>
166   /// <returns>Returns the first found value</returns>
167   string PWM::FindPath(string value) {
```

```
168   auto dir = opendir(OCP_PATH);
169   auto entity = readdir(dir);
170   while (entity != NULL) {
171     if (entity->d_type == DT_DIR) {
172       string str = static_cast<string>(entity->d_name);
173       if (str.find(value) != string::npos) {
174         return str;
175       }
176     }
177     entity = readdir(dir);
178   }
179   return "";
180 }
181 }
```

B.0.5 General project file

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  #pragma once
9
10 #include "ADC.h"
11 #include "EC12P.h"
12 #include "eqep.h"
13 #include "GPIO.h"
14 #include "PWM.h"
15 #include "SoilCape.h"
16 #include "Microscope.h"
17 #include "CouldNotGrabImageException.h"
18 #include "ADCReadException.h"
19 #include "FailedToCreateGPIOPollingThreadException.h"
20 #include "FailedToCreateThreadException.h"
21 #include "GPIOReadException.h"
22 #include "MicroscopeNotFoundException.h"
23 #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;
};
}
}
```

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  #pragma once
9
10 #include <exception>
11 #include <string>
12
13 using namespace std;
14
15 namespace Hardware {
16 namespace Exception {
17 class FailedToCreateGPIOPollingThreadException : public std::exception {
18 public:
19   FailedToCreateGPIOPollingThreadException(
20       string m = "Failed to create GPIO polling thread!")
21       : msg(m){};
22   ~FailedToCreateGPIOPollingThreadException() _GLIBCXX_USE_NOEXCEPT{};
23   const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };
24
25 private:
26   string msg;
27 };
28 }
29 }
```

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7
8  #pragma once
9
10 #include <exception>
11 #include <string>
12
13 using namespace std;
14
15 namespace Hardware {
16 namespace Exception {
17 class FailedToCreateThreadException : public std::exception {
18 public:
19   FailedToCreateThreadException(string m = "Couldn't create the thread!")
20       : msg(m){};
21   ~FailedToCreateThreadException() _GLIBCXX_USE_NOEXCEPT{};
22   const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };
23
24 private:
25   string msg;
26 };
27 }
28 }
```

```
1  /* Copyright (C) Jelle Spijker - All Rights Reserved
2   * Unauthorized copying of this file, via any medium is strictly prohibited
3   * and only allowed with the written consent of the author (Jelle Spijker)
4   * This software is proprietary and confidential
5   * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6   */
7  #define EXCEPTION_OPENCAM "Exception could not open cam!"
8  #define EXCEPTION_OPENCAM_NR 0
9  #define EXCEPTION_NOCAMS "Exception no cam available!"
10 #define EXCEPTION_NOCAMS_NR 1
11 #define EXCEPTION_QUERY "Exception could not query device!"
12 #define EXCEPTION_QUERY_NR 3
13 #define EXCEPTION_FORMAT_RESOLUTION "Exception No supported formats and resolutior
14 #define EXCEPTION_FORMAT_RESOLUTION_NR 4
15 #define EXCEPTION_CTRL_NOT_FOUND "Control not found!"
16 #define EXCEPTION_CTRL_NOT_FOUND_NR 5
17 #define EXCEPTION_CTRL_VALUE "Control value not set!"
18 #define EXCEPTION_CTRL_VALUE_NR 5
19
20
21 #pragma once
22 #include <exception>
23 #include <string>
24
25 using namespace std;
26
27 namespace Hardware {
28 namespace Exception {
29 class MicroscopeException : public std::exception {
30 public:
31   MicroscopeException(string m = EXCEPTION_OPENCAM,
32                       int n = EXCEPTION_OPENCAM_NR) : msg{m}, nr{n} {
   }
33   ~MicroscopeException() _GLIBCXX_USE_NOEXCEPT {}
34   const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); }
35   const int *id() const _GLIBCXX_USE_NOEXCEPT { return &nr; }
36
37 private:
38   string msg;
39   int nr;
40 };
41 }
42 }
```

```
/* 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;
};
}
}
```