Implementing a Signal Processing Filter in BCI2000 using C++

Jürgen Mellinger
University of Tübingen, Germany

presented by

Jeremy Hill
Max Planck Institute for Biological Cybernetics
Tübingen, Germany
BCI2000 System Model

Operator

system configuration  visualization

Source

Storage

brain signals  event markers

Signal Processing

control signals  event markers

User Application

event markers
BCI2000 System Model

Operator

system configuration

visualization

Filter 1.1
Filter 1.2
Filter 1.3
Filter 1.4

Filter 2.1
Filter 2.2
Filter 2.3
Filter 2.4

Filter 3.1
Filter 3.2

brain signals

control signals

event markers

event markers

event markers

User

iciation

iciation
BCI2000 Code Base

• Written in C++
• Extensive use of STL
• Dependencies
  – Borland VCL GUI library   (but soon: Qt-based)
  – Win32 API
  – Borland C++ Builder Compiler required for complete build
• Parts (mex files, command line filters, many source files) may be built using gcc
// $Id: SpatialFilter.h 1751 2009-01-24 15:48:06E mellinger $ 
// Author: schalk@ecs.warwick.ac.uk, juergen.mellinger@uni-tuebingen.de 
// Description: The SpatialFilter computes a linear transformation of its 
// input signal, given by a matrix-valued parameter. 
// In this matrix, input channels correspond to columns, and output channels 
// to rows. 
// 
// (C) 2000-2008, ECI2000 Project 

#ifdef SPATIAL_FILTER_H
#define SPATIAL_FILTER_H

#include "GenericFilter.h"

class SpatialFilter : public GenericFilter
{
public:
  SpatialFilter();
  virtual ~SpatialFilter();
  virtual void Preflight ( const SignalProperties6, SignalProperties6 ) const;
  virtual void Initialize( const SignalProperties6, const SignalProperties6 );
  virtual void Process ( const GenericSignal6 Input, GenericSignal6 Output );

private:
  std::vector<std::vector<double> > mFilterMatrix;
};

#endif // SPATIAL_FILTER_H
BCI2000 System Model

Operator

system configuration  visualization

Filter 1.1  Filter 1.2  Filter 1.3  Filter 1.4

brain signals

control signals

event markers

event markers
BCI2000 Source Module

Inheritance:

Environment (Parameter, State)
GenericFilter (Initialize, StartRun, Process, ...)
GenericADC
MyHardwareADC

Diagram:

- ADC Driver Library
- ADC Component
- BCI2000 Framework
BCI2000 Signal Processing

Inheritance:

Environment
GenericFilter
MyFilter

(Parameter, State)
(Initialize, StartRun, Process, ...)

Parameterization
Data
GenericFilter Class Interface

- **GenericFilter** defines an interface
  - implemented by your code
  - called from the BCI2000 framework
- Interface elements are event handlers
  - Initialization
  - Processing
  - Helpers
- **C++**: virtual functions
**GenericFilter** Class Interface

- Blockwise data processing: `Process()`
- Helper functions
  - Constructor
  - Preflight()
  - Initialize()
- A few others
GenericFilter::Process()

• A filter's main function
• Called once for each block of data
• Single chain of filters:
  input received from preceding filter
  output fed into subsequent filter
• Filters not modifying their signal must do
  Output=Input;
GenericFilter::Initialize()

- Called when parameter settings have changed
- Adapt your filter’s member variables to reflect new parameter settings
- No need to check for configuration errors
GenericFilter::Preflight()

• Called when parameter settings need verification

• Prevent configuration errors
  • report an error if settings will lead to a crash
  • report a warning if parameters appear inconsistent
GenericFilter::StartRun()

• Called when a new run starts – user clicks “Start” or “Resume”

• Typical `StartRun()` examples
  – writing a run number into a log file
  – opening a new output file

• Any initialization should either fit into `Initialize()` or into `StartRun()`
GenericFilter::StopRun()

• Called when a run ends
• Only place where parameter values may be modified
• Modified parameter values will be propagated to operator and other modules automatically
Classification

• Generally:
  • Example to category
  • Continuous (data) to discrete (label)

• Classification function:
  • Multi-dimensional continuous data
  • Continuous 1D classification function
  • Discretization

• Linear classification
  \[ f(\mathbf{x}) = \sum_{i} x_i w_i \]

• Training algorithms
  LDA, linear SVM, Perceptron
BCI Classification

Continuous feedback:

brain state $\rightarrow$ cursor $\rightarrow$ target/class

\[ \tilde{x} \rightarrow f(\tilde{x}) \]

Data Acq   Sig Proc   Application
Tutorial Example: Linear Classifier

• Linear classification function
• Sparse representation: nonzero weights only
• Multiple output channels

\[ f(x) = \sum_{kl} w_{kl} x_{kl} \]

\[ f_j(x) = \sum_{kl} w_{jkl} x_{kl} \]
BCI2000 System Model

Operator

system configuration  visualization

Filter 1.1  Filter 1.2  Filter 1.3  Filter 1.4

Filter 2.1  Filter 2.2  Filter 2.3  Filter 2.4

Filter 3.1  Filter 3.2

brain signals  event markers

control signals  event markers

event markers
Tutorial Example: Linear Classifier

- Derive a new class `LinClassifier` from `GenericFilter`
- Implement its `Process()`, `Initialize()`, `Preflight()`, and constructor functions
- Add it to the filter chain
Declaring a 
LinClassifier Class

Contents of LinClassifier.h:

```cpp
#ifndef LIN_CLASSIFIER_H
#define LIN_CLASSIFIER_H

#include "GenericFilter.h"

class LinClassifier : public GenericFilter
{
  public:
    LinClassifier();
    ~LinClassifier();

    void Preflight( const SignalProperties&, SignalProperties& ) const;
    void Initialize( const SignalProperties&, const SignalProperties& );
    void Process( const GenericSignal&, GenericSignal& );
};
#endif // LIN_CLASSIFIER_H
```
void LinClassifier::Process(const GenericSignal& Input, 
                           GenericSignal& Output) 
{
  for (int ch = 0; ch < Output.Channels(); ++ch)
    for (int el = 0; el < Output.Elements(); ++el)
      Output(ch, el) = 0.0;

  for (size_t i = 0; i < mWeights.size(); ++i)
    Output(mOutputChannels[i], 0) += Input(mInputChannels[i], mInputElements[i])
       * mWeights[i];
}

\[
f_j(x) = \sum_{kl} w_{jkl} x_{kl}\]
Data Members

```cpp
#include <vector>
...

class LinClassifier : public GenericFilter
{
  public:
    ...
  private:
    std::vector<float> mOutputChannels,
    mInputChannels,
    mInputElements,
    mWeights;
};

f_j(x) = \sum_{kl} w_{jkl} x_{kl}
```
void LinClassifier::Initialize( const SignalProperties& Input,
               const SignalProperties& Output )
{
  const ParamRef& Classifier = Parameter( "Classifier" );
  size_t numEntries = Classifier->NumRows();
  mInputChannels.resize( numEntries );
  ...
  for( size_t entry = 0; entry < numEntries; ++entry
  {
    mInputChannels[ entry ] = Classifier( entry, 0 ) - 1;
    mInputElements[ entry ] = Classifier( entry, 1 ) - 1;
    mOutputChannels[ entry ] = Classifier( entry, 2 ) - 1;
    mWeights[ entry ] = Classifier( entry, 3 );
  }
}
LinClassifier::Preflight()

Potential errors:
• inappropriate dimensions of Classifier matrix parameter
• index out of range in input
• index out of range in output
LinClassifier::Preflight()

```cpp
void LinClassifier::Preflight(const SignalProperties& Input, SignalProperties& Output) const {
  // Check matrix format
  if( Parameter( "Classifier" )->NumColumns() != 4 )
    bcierr << "Classifier parameter must have 4 columns "
    << "(input channel, input element, "
    << "output channel, weight)"
    << endl;

  // Check indices and obtain max output channel
  ...
  // Request output dimensions
  Output = SignalProperties( maxChannel, 1 );
}
```
LinClassifier::LinClassifier()
{
    BEGIN_PARAMETER_DEFINITIONS
    "Filtering matrix Classifier= 2 "
    ": [ input%20channel input%20element%20(bin) output%20channel weight ] "
    " 1 4 1 1 
    " 1 6 2 1 
    " % % % // Linear classification matrix in sparse representation",
    END_PARAMETER_DEFINITIONS
}
Compiling LinClassifier.cpp

#include "PCHIncludes.h" // Make the compiler's Pre-Compiled
#pragma hdrstop          // Headers feature happy

#include "LinClassifier.h"
#include "BCIErrror.h"
#include <algorithm>     // for std::max()

using namespace std;

...
BCI2000 System Model

Operator

system configuration  visualization

Filter 1.1  Filter 1.2  Filter 1.3  Filter 1.4

Filter 2.1  Filter 2.2  Filter 2.3  Filter 2.4

Filter 3.1  Filter 3.2

brain signals  control signals

event markers  event markers  event markers
Instantiating **LinClassifier** in a Signal Processing module

PipeDefinition.cpp:

```cpp
#include "LinClassifier.h";
...
Filter( SpatialFilter, 2.A );
Filter( ARFilter, 2.B );
Filter( LinClassifier, 2.C );
Filter( NormalFilter, 2.D );
```

Filter chain:

1. Spatial Filter
2. AR Spectral Estimator
3. Classifier
4. Normalizer
Framework Recapitulation

- Parameter access
- State access
- Reporting errors and warnings
Parameter Access

- **Getting parameter values:**
  ```
  float myFloat = Parameter( "SamplingRate" );
  ```

- **Setting parameter values:**
  ```
  Parameter( "TimeConstant" ) = myFloat;
  ```

- **Matrix parameters:**
  ```
  myFloat = Parameter( "MUD" )( 0, 2 );
  myFloat = Parameter( "Audio" )( "Task", 2 );
  myFloat = OptionalParameter( "MUD", 1 )( 20, 30 );
  ```
State Access

• Getting state values:
  ```
  short targetCode = State( "TargetCode" );
  ```

• Setting state values:
  ```
  State( "ResultCode" ) = targetHit;
  ```

• Optional states:
  ```
  short artifact = OptionalState( "Artifact", 0 );
  ```

• Per-sample access:
  ```
  State( "Artifact" )( 3 ) = 1;
  ```
Reporting Errors and Warnings

using namespace std;

• C++ command-line programs use
  cout << "The result is " << result << "." << endl;
  cerr << "This is an error message." << endl;

• BCI2000 filters use
  bciout << "The result is " << result << "." << endl;
  bcierr << "TimeConstant must be greater 0." << endl;

• Side effects
  endl is \n forcing immediate display
  bcierr from Preflight(): Prevents Initialize()
  bcierr elsewhere: Terminates module
Signal Units and Labels

- Consistent choice of visualization scale
- Visualization in correct units
- User convenience
  - classifier configuration in terms of frequencies (SMR) or temporal offsets (ERP)
  - hardware-independence
- Automatic conversion
  
  ```python
  channel = Signal.ChannelIndex("Cz");
  element = Signal.ElementIndex("25Hz");
  ```
void LinClassifier::Initialize( const SignalProperties& Input,
    const SignalProperties& Output )
{
    const ParamRef& Classifier = Parameter( "Classifier" );
    ...
    for( size_t entry = 0; entry < numEntries; ++entry 
        
        
        
        mInputChannels[ entry ] = Input.ChannelIndex( Classifier( entry, 0 ) );
        mInputElements[ entry ] = Input.ElementIndex( Classifier( entry, 1 ) );
        ...
    }
}
Signal Units and Labels

Configuration ...

Classifier
SpatialFilter
SpatialFilterType
SpatialFilterCAROutput
SpatialFilter

Edit Matrix Classifier
Linear classification matrix in sparse representation

<table>
<thead>
<tr>
<th># of columns</th>
<th># of rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input channel</th>
<th>input element (bin)</th>
<th>output channel</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>6</td>
<td>-1.0</td>
</tr>
</tbody>
</table>
Signal Units and Labels

[Image of a software interface showing signal units and labels configuration]
“C3” and “C4”, huh? If you’re classifying bandpower features from EEG, then I hope you’ve done some careful spatial filtering first…
BCI2000 Application

- GraphObject
- Images, Text
- Audio Players
- 3D API
Conclusion

• BCI2000 is written in C++ and currently requires the Borland C++ environment.
• BCI2000 consists of four modules, each of which contains a number of filters.
• Extending BCI2000 is done by deriving your own filter class from GenericFilter.
• Begin with coding the core functionality into your filter’s Process() member function, then derive other member functions.

Share your ideas, suggestions, annoyances
juergen.mellinger@uni-tuebingen.de