Jafar, a C/C++ framework

Nizar Sallem

November 1, 2009

Jafar is A Framework for Algorithms [development] in Robotics

Jafar is a C/C++ framework, also environment, which aims to ease development of **your** algorithms in the robotics field.

Framework

A collection of libraries

Algorithm

Focused on your *issues* $\Rightarrow$ invent all but the wheel

Robotics

Dedicated to robotics $\Rightarrow$ coupled to some free libraries focused on vision, algebra, geometry, etc.

Cool but how?

To achieve previous goals, jafar brings these features:

- Support for C/C++
- A build system
- Interactive shell, tcl or ruby
- Modular environment
- Bindings with swig
- Errors reporting
- Unit testing
- Documentation

Jafar Structure

- Directories

Jafar Structure

- Directories
  - bin : various tools

Jafar Structure

- Directories
  - bin : various tools
  - modules : all the installed modules

Jafar Structure

- Directories
  - bin : various tools
  - modules : all the installed modules
  - doc : the documentation

Jafar Structure

- Directories
  - bin : various tools
  - modules : all the installed modules
  - doc : the documentation
  - share : tools to share work

Jafar Structure

- Directories
  - bin : various tools
  - modules : all the installed modules
  - doc : the documentation
  - share : tools to share work
  - lib, include

Jafar Structure

- Directories
  - bin : various tools
  - modules : all the installed modules
  - doc : the documentation
  - share : tools to share work
  - lib, include
  - aclocal, config, autom4te.cache, Makefile...

Jafar Structure

- Directories
  - bin : various tools
  - modules : all the installed modules
  - doc : the documentation
  - share : tools to share work
  - lib, include
  - aclocal, config, autom4te.cache, Makefile...
- Subversion layout

Jafar Structure

- Directories
  - bin : various tools
  - modules : all the installed modules
  - doc : the documentation
  - share : tools to share work
  - lib, include
  - aclocal, config, autom4te.cache, Makefile...
- Subversion layout
  - svn+ssh://svn.laas.fr/svn/jafar/jafarBackbone/trunk/ : the heart of jafar

Jafar Structure

- Directories
  - bin : various tools
  - modules : all the installed modules
  - doc : the documentation
  - share : tools to share work
  - lib, include
  - aclocal, config, autom4te.cache, Makefile...
- Subversion layout
  - svn+ssh://svn.laas.fr/svn/jafar/jafarBackbone/trunk/ : the heart of jafar
  - svn+ssh://**yourlogin**@svn.laas.fr/svn/jafar/jafarModules/trunk/ : all the modules

A Jafar's module is:

A C/C++ library

header files ⇒ modulname/**include**/modulname

source files ⇒ modulname/**src**

A Jafar's module is:

A C/C++ library

header files ⇒ modulname/**include**/modulname
source files ⇒ modulname/**src**

A set of tcl/ruby scripts

.rb or .tcl files in modulname/**macro**

A Jafar's module is:

A C/C++ library

header files ⇒ modulname/**include**/modulname
source files ⇒ modulname/**src**

A set of tcl/ruby scripts

.rb or .tcl files in modulname/**macro**

Documentation

modulname.doxy in modulname/**doc** and doxygen formatted comments in headers

A Jafar's module is:

A C/C++ library

header files ⇒ modulname/**include**/modulname
source files ⇒ modulname/**src**

A set of tcl/ruby scripts

.rb or .tcl files in modulname/**macro**

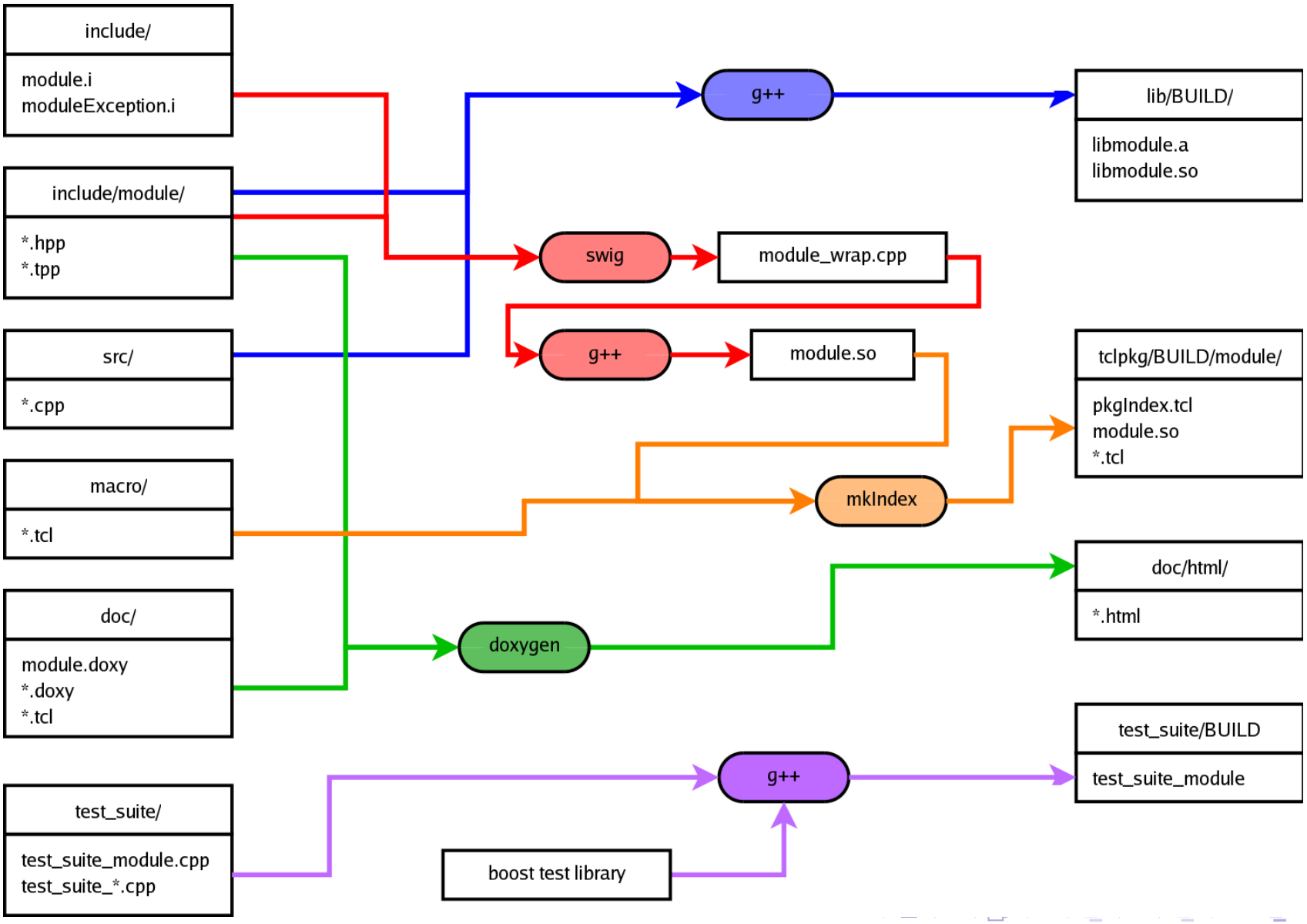
Documentation

modulname.doxy in modulname/**doc** and doxygen formatted comments in headers

Unit tests

test_suite_modulename.cpp in modulname/**test_suite**

Directory structure



New module, how to?

- create it

```
1  cd ${JAFAR_DIR}/modules
2  ../bin/jafar --module -c playmodule
```

New module, how to?

- create it

```
1  cd ${JAFAR_DIR}/modules
2  ../bin/jafar --module -c playmodule
```

- what is in there

```
1  cd playmodule
2  tree .
```

```
.
|-- COPYRIGHT
|-- Makefile
|-- README
|-- User.make
|-- doc
|   `-- playmodule.doxy
|-- include
|   |-- playmodule
|   |   `-- playmoduleException.hpp
|   |-- playmodule.i
|   |-- playmoduleException.i
|   `-- playmoduleTools.i
|-- macro
|   `-- playmodule.rb
|-- src
|   |-- playmoduleException.cpp
|   `-- ruby
|       `-- extconf.rb
`-- test_suite
    `-- test_suite_playmodule.cpp

7 directories, 13 files
```

User.make

Module specific make instructions

```
# $Id$ #

#
# User part of the makefiles
# module playmodule
#
```

```
# module version
MODULE_VERSION = 0
MODULE_REVISION = 1
```

information on the module version

```
# modules dependencies
REQUIRED_MODULES = kernel
OPTIONAL_MODULES =
```

module intrinsic dependencies compilation time

```
# external libraries dependencies
REQUIRED_EXTLIBS =
OPTIONAL_EXTLIBS =
```

module extrinsic dependecies compilation time

```
# LDFLAGS +=
```

module extrinsic dependecies link time

```
LIBS += -lkernel
```

module intrinsic dependencies link time

```
# CPPFLAGS += -DJFR_NDEBUG
CPPFLAGS += $(BOOST_CPPFLAGS)

CXXFLAGS += -g -ggdb -Wall
```

mixed flags for compilation

include/playmodule.i

SWIG interface file for wrapping instructions

```
/** swig interface file for module playmodule.
 *
 * \file playmodule.i
 * \ingroup playmodule
 */

%module playmodule

%{
/* ruby defines ALLOC which conflicts with boost */
#undef ALLOC

/*
   * headers necessary to compile the wrapper
   */

#include "jafarConfig.h"

// using namespace jafar::playmodule;

%}

%include "jafar.i"
%include "playmoduleException.i"

/*
 * headers to be wrapped goes here
 */

// %include "playmoduleTools.i"
// instantiate some print functions
// replace "Type" with appropriate class name
// %template(print) jafar::playmodule::print<jafar::playmodule::Type>;
```

} here is where to place headers to compile module's wrapper

} other interface file generally coming from elsewhere

} here is where to place headers that module's wrapper will provide

SWIG is a powerfull tool but has its Achilles heel: template functions and classes have to be instantiated. Here is where to do it

include/playmoduleException.hpp
src/playmoduleException.cpp

C++ provides exceptions handling, yet you still need to define them.

```
1  #ifndef PLAYMODULE_PLAYMODULE_EXCEPTION_HPP
2  #define PLAYMODULE_PLAYMODULE_EXCEPTION_HPP
3
4  #include "kernel/jafarException.hpp"
5
6  namespace jafar {
7    namespace playmodule {
8      class PlaymoduleException : public ::jafar::kernel::Exception {
9
10     public:
11
12       enum ExceptionId {
13         //          MY_ERROR /**< my error */
14       };
15
16     }; // class PlaymoduleException
17   } // namespace playmodule
18 } // namespace jafar
19 #endif // PLAYMODULE_PLAYMODULE_EXCEPTION_HPP
```

doc/playmodule.doxy

Documentation that will be seen on module's welcome page

```
1  /* $Id$ */
2
3  /*!
4    \addtogroup playmodule Module playmodule
5
6  \version 0.1
7
8  \author
9    jafar@laas.fr
10
11   Short description of the module goes here...
12
13 \section secPlaymoduleHistory  History
14
15   - 0.1 (2009-10-29) - Initial version
16
17 \section secPlaymoduleRequirements Requirements
18
19   Other module or external libraires dependences...
20
21 \section secPlaymoduleMacro Macro
22
23   Extra doc for macro can go here... (you can delete this section if
24 not relevant)
25
26 \section secPlaymoduleInterface Tcl interface (generated by swig)
27
28   The interface of the module is generated from the following files:
29     - playmodule.i defines the wrapped classes and functions,
30     - playmoduleException.i defines the \c try { } \c catch block
```

macro/playmodule.rb macro/playmodule.tcl

Macros are scripts written in ruby or tcl to provide "executive" tasks

in ruby

```
1  require 'jafar/kernel'
2  require 'jafar/playmodule/playmodule'
3  Jafar.register_module Jafar::Playmodule
```

macro/playmodule.rb macro/playmodule.tcl

Macros are scripts written in ruby or tcl to provide "executive" tasks

in ruby

```
1  require 'jafar/kernel'
2  require 'jafar/playmodule/playmodule'
3  Jafar.register_module Jafar::Playmodule
```

in tcl

```
1  package require playmodule
2
3  namespace eval playmodule {
4  }
5
6  package provide playmodule 0.1
```

test_suite/test_suite_playmodule.cpp

Unit tests to ensure module *aimed* functioning

```
1   /* $Id$ */
2
3   // boost unit test includes
4   #define BOOST_TEST_MAIN
5   #define BOOST_TEST_DYN_LINK
6   #include <boost/test/auto_unit_test.hpp>
7   using boost::unit_test_framework::test_suite;
8   using boost::unit_test_framework::test_case;
9
10  #include "kernel/jafarDebug.hpp"
11
12  BOOST_AUTO_TEST_CASE( dummy ){}
13
14  test_suite*
15  init_unit_test_suite( int, char* [] ) {
16
17    // we set the debug level to Warning
18    jafar::debug::DebugStream::setDefaultLevel(jafar::debug::DebugStream::Warning);
19    return 0;
20  }
```

Compilation

- Compile it

```
1 make
```

Compilation

- Compile it

```
1 make
```

- Load it

```
1 require 'jafar/playmodule'
```

Compilation

- Compile it

```
1 make
```

- Load it

```
1 require 'jafar/playmodule'
```

- So what is available...

A first class: header

```
1  #ifndef _MY_FIRST_CLASS_
2  #define _MY_FIRST_CLASS_
3  namespace jafar {
4    namespace playmodule {
5      class MyFirstClass {
6        public:
7          int add(int u, int v) const;
8      };
9    }
10 }
11 #endif
```

A first class: source

```
1  #include "playmodule/MyFirstClass.hpp"
2
3  using namespace jafar::playmodule;
4
5  int MyFirstClass::add(int u, int v) const
6  {
7    return u + v;
8  }
```

Bind it

Open file *playmodule.i* and add:

```
1  #include "playmodule/MyFirstClass.hpp"
```

And:

```
1  %include "playmodule/MyFirstClass.hpp"
```

Use it

```
1  require 'jafar/playmodule'
2  obj = Playmodule::MyFirstClass.new
3  obj.add( 1, 2)
```

Features

- Debug
- Usefull macros
- Configuration file
- Timing tool

Debug (1/2)

- DataLogger : to log data into a file

Debug (1/2)

- DataLogger : to log data into a file
- Debug macro : JFR_DEBUG

```
1    JFR_DEBUG( u << " + " << v << " = " << (u+v) );
```

Disable debug:

```
1    Jafar::Kernel::DebugStream::moduleOff("playmodule");
```

Debug (2/2)

- JFR_ASSERT / JFR_PRED_ERROR : check that a parameter is correct

```
1  int MyFirstClass::div(int u, int v) const
2  {
3    JFR_ASSERT( v != 0, "Can't divide by 0");
4    return u / v;
5  }
```

Usefull macros

For instance JFR_FOREACH:

```
1  std::vector< CoolObject > coolObjects;
2  JFR_FOREACH( CoolObject& coolObject, coolObjects )
3  {
4    coolObject.soSomethingCool();
5  }
```

Instead of:

```
1  std::vector< CoolObject > coolObjects;
2  for( std::vector< CoolObject >::iterator it
3       = coolObjects.begin();
4    it = coolObjects.end(); ++it )
5  {
6    it->soSomethingCool();
7  }
```

Configuration file (1/3)

Exemple of configuration file:

```
1  MyValue: 10
2  OtherValue: hello
```

Exemple of code to read file:

```
1  KeyValueFile configFile;
2  configFile.readFile( "test.cfg");
3  int val;
4  configFile.getItem( "MyValue", val );
5  std::string val2;
6  configFile.getItem( "OtherValue", val2 );
```

Configuration file (2/3)

KeyValueFileSave: an object which can save its parameters.

```
1   class CoolAlgorithm : public KeyValueFileSave {
2     public:
3       virtual void saveKeyValueFile(
4       jafar::kernel::KeyValueFile& keyValueFile)
5       {
6         keyValueFile.setItem("MyParameter", m_parameter );
7       }
8     public:
9       int m_parameter;
10  };
11
12  CoolAlgorithm coolAlgorithm;
13  coolAlgorithm.save("algo.cfg");
```

Configuration file (3/3)

KeyValueFileLoad: an object which can load its parameters.

```
1   class CoolAlgorithm : public KeyValueFileLoad {
2     public:
3       virtual void loadKeyValueFile(
4       jafar::kernel::KeyValueFile const& keyValueFile)
5       {
6         keyValueFile.getItem("MyParameter", m_parameter );
7       }
8     public:
9       int m_parameter;
10  };
11
12  CoolAlgorithm coolAlgorithm;
13  coolAlgorithm.load("algo.cfg");
```

Timing tools

- Chrono

```
1    Chrono chrono;
2    chrono.start();
3    // Do some extensive computation
4    JFR_DEBUG(chrono.elapsed());
```

Timing tools

- Chrono

```
1    Chrono chrono;
2    chrono.start();
3    // Do some extensive computation
4    JFR_DEBUG(chrono.elapsed());
```

- Framerate

Features

- matrix and vectors computation
- use lapack
- linear solvers
- least-square optimization

Create vectors and matrix

- Bounded vectors

```
1  jblas::vec2 vec_2;
2  jblas::vec3 vec_3;
3  jblas::vec4 vec_4;
```

Create vectors and matrix

- Bounded vectors
- Unbounded vectors

```
1  jblas::vec vec_n( 10 );
```

Create vectors and matrix

- Bounded vectors
- Unbounded vectors
- Bounded matrix

```
1   jblas::mat22 mat_22;
2   jblas::mat33 mat_33;
3   jblas::mat44 mat_44;
```

Create vectors and matrix

- Bounded vectors
- Unbounded vectors
- Bounded matrix
- Unbounded matrix

```
1  jblas::mat mat_nn(100,400);
```

Create vectors and matrix

- Bounded vectors
- Unbounded vectors
- Bounded matrix
- Unbounded matrix
- Zero, Scalar and Identity matrix

```
1  jblas::mat mat = jblas::zero_mat(5);
2  jblas::mat mat = jblas::identity_mat(5);
3  jblas::mat mat = 5.0 * jblas::scalar_mat(5); // Matrix filled with 5.0
```

Vectors and matrix operations

- Addition, substraction

```
1   jblas::vec2 v1, v2, v3;
2   v1 = v1 + v2 - v3;
```

Vectors and matrix operations

- Addition, substraction
- Multiplication

```
1   jblas::mat m1, m2, m3;
2   m3 = ublas::prod( m1, m2 );
3   m3 = ublas::prod( m1,
4   jblas::mat( ublas::prod( m3, m2 ) );
```

Vectors and matrix operations

- Addition, substraction
- Multiplication
- Transposition

```
1  jblas::mat m4 = ublas::trans(m3);
2  m4 = ublas::prod( ublas::trans(m2), m1 );
```

Vectors and matrix operations

- Addition, substraction
- Multiplication
- Transposition
- Inversion

```
1   ublasExtra::inv( m4 );
```

Vectors and matrix operations

- Addition, substraction
- Multiplication
- Transposition
- Inversion
- Dot and cross product

```
1   ublas::outer_prod( v1, v2 ); // cross product
2   ublas::inner_prod( v1, v2 ); // dot product
```

Vectors and matrix operations

- Addition, substraction
- Multiplication
- Transposition
- Inversion
- Dot and cross product
- Determinant

```
1   ublasExtra::det( m4 );
```

Symmetric matrix

- Bounded symmetric matrix:

```
1  jblas::sym_mat22 mat_22;
2  jblas::sym_mat33 mat_33;
3  jblas::sym_mat44 mat_44;
```

Symmetric matrix

- Bounded symmetric matrix:
- Unbounded symmetric matrix:

```
1  jblas::sym_mat mat_nn(100);
```

Symmetric matrix

- Bounded symmetric matrix:
- Unbounded symmetric matrix:
- Create symmetrix matrix from non symmetric matrix

```
1  jblas::sym_mat mat_nn(100);
2  jblas::sym_mat smat_nn =
3      ublas::symmetric_adaptor<jblas::mat44,
4          ublas::lower>( mat_nn );
5  jblas::sym_mat smat_nn =
6      ublas::symmetric_adaptor<jblas::mat44,
7          ublas::upper>( mat_nn );
```

Symmetric matrix

- Bounded symmetric matrix:
- Unbounded symmetric matrix:
- Create symmetrix matrix from non symmetric matrix
- Access elements

```
1  jblas::sym_mat22 mat_22;
2  mat_22(0,1) = 10.0;
3  // Warning:
4  mat_22(1,0) = 12.0;
5  JFR_DEBUG( mat_22(1,0) ); // will display 10.0 !
```

Use lapack

- To compute SVD and eigen values

Use lapack

- To compute SVD and eigen values
- Warning: use column major with Lapack

```
1  jblas::mat A( 30, 3 );
2  jblas::mat_column_major m_A( A );
3  jblas::vec s(3);
4  jblas::mat_column_major U(30, 3);
5  jblas::mat_column_major VT(3, 3);
6
7  int ierr = lapack::gesdd(m_A,s,U,VT);
```

Linear least square

- Find x that minimize $\|A.x - b\|^2$

Linear least square

- Find x that minimize $\|A.x - b\|^2$
- LinearLeastSquares

```
1   LinearLeastSquares lls;
2   lls.setSize( 3 /* model size */,
3                10 /* number of points */ );
4   jblas::vec valueOfA;
5   double valueOfB;
6   lls.setData( 0 /* index of point */,
7                valueOfA,
8                valueOfB );
9   ...
10  lls.solve();
11  lls.x(); // return the value of x
12  lls.xCov(); // return the covariance
```

Linear least square

- Find x that minimize $\|A.x - b\|^2$
- LinearLeastSquares
- VariableSizeLinearLeastSquares

```
1  VariableSizeLinearLeastSquares vsll
2                ( 3 /* model size */ );
3  jblas::vec valueOfA;
4  double valueOfB;
5  vsll.addMeasure( valueOfA, valueOfB );
6  ...
7  vsll.solve();
8  vsll.x(); // return the value of x
```

Features

- T3D: 3D Transformation

Features

- T3D: 3D Transformation
- Geometric classes such as Point, Lines, Boxes...

T3D: 3D Transformation

- Support for Euler and Quaternion

```
1  jblas::vec transfoX(6);
2  transfoX(0) = x;
3  transfoX(1) = y;
4  transfoX(2) = z;
5  transfoX(3) = yaw;
6  transfoX(4) = pitch;
7  transfoX(5) = roll;
8  geom::T3DEuler transfo( transfoX );
```

T3D: 3D Transformation

- Support for Euler and Quaternion
- Composition

```
1  geom::T3DEuler robotToWorld = something;
2  geom::T3DEuler sensorToRobot = something;
3  geom::T3DEuler sensorToWorld;
4  geom::T3D::compose( sensorToRobot,
5                      robotToWorld,
6                      sensorToWorld);
```

T3D: 3D Transformation

- Support for Euler and Quaternion
- Composition
- Invert

```
1  geom::T3DEuler robotToWorld = something;
2  geom::T3DEuler worldToRobot;
3  geom::T3D::invert( robotToWorld, worldToRobot );
```

T3D: 3D Transformation

- Support for Euler and Quaternion
- Composition
- Invert
- Transform a vector

```
1  geom::T3DEuler sensorToWorld = something;
2  jblas::vec X_sensor;
3  jblas::vec X_world = ublas::prod( sensorToWorld.getM(), X_sensor );
```

Geometric classes

- Points

```
1   jblas::vec v = coordinates of the point;
2   geom::Point<3> p1(
3                 new Point<3>::EuclideanDriver( v ) );
```

Geometric classes

- Points
- Lines

```
1   jblas::vec v = coordinates of the origin;
2   jblas::vec v = coordinates of the vector director;
3   geom::Line<3> l1(
4                 new Line<3>::EuclideanDriver( v ) );
5   geom::Point<3> p1;
6   geom::Point<3> p2;
7   geom::Line<3> l2(
8       new Line<3>::TwoPointsPointerDriver( &p1, &p2 ) );
9   geom::Line<3> l2(
10                  new Line<3>::TwoPointsDriver( p1, p2 ) );
```

Geometric classes

- Points
- Lines
- Segments, polylines, planes, facets...

Geometric classes

- Points
- Lines
- Segments, polylines, planes, facets...
- Operations

```
1   geom::distance( p1, p2 );
2   geom::distance( p1, l2 );
3   geom::angle( l1, l2 );
```

Geometric classes

- Points
- Lines
- Segments, polylines, planes, facets...
- Operations
- Bounding box

```
1  geom::Segment segment( {1,1,1}, {-1,-1,-1} );
2  segment.boundingBox(); // Return {1,1,1}, {-1,-1,-1}
```

Geometric classes : VoxelSpace (1/2)

```
1   class Object
2   {
3     public:
4       Object(const geom::Atom<3>& atom_)
5         : m_atom(atom_)
6       {
7       }
8       const geom::Atom<3>& atom() const
9       { return m_atom; }
10    private:
11      const geom::Atom<3>& m_atom;
12  };
```

Geometric classes : VoxelSpace (2/2)

```
1   geom::VoxelSpace<dimension, Object,
2       geom::AtomBoundingBoxGetter<dimension, Object> >
3       voxelSpace;
4   geom::Point<3> pt;
5   Object* obj1 = new Object(pt);
6   voxelSpace.insertObject( obj1 );
7   geom::Line<3> li;
8   Object* obj2 = new Object(li);
9   voxelSpace.insertObject( obj2 );
10  geom::BoundingBox<3> bb( onePoint, oneAnotherPoint );
11  std::list<Object*> objects =
12                          voxelSpace.objectsIn( bb );
```

Features

- load/read images

Features

- load/read images
- access to the whole OpenCV API

Features

- Create an image

```
1  image::Image dx( width, height, IPL_DEPTH_16S, JfrImage_CS_GRAY );
```

Features

- Create an image

```
1   image::Image dx( width, height, IPL_DEPTH_16S, JfrImage_CS_GRAY );
```

- Use a functin from OpenCV

```
1   image::Image myImage;
2   myImage.loadImage("MyFile.png");
3   cvSobel( myImage, dx, 1, 0, 3);
```

Features

Camera model:

- Pinhole
- Barreto
- Stereo bench

Features

- Read images

Features

- Read images
- Read positions

Default configuration

- set the base directory

```
1   Jafar::Datareader::DataReader
2       .setDefaultBasePath("/home/cyrille/laas/Data")
```

Default configuration

- set the base directory

```
1  Jafar::Datareader::DataReader
2      .setDefaultBasePath("/home/cyrille/laas/Data")
```

- set the series name

```
1  Jafar::Datareader::DataReader
2      .setDefaultSeriesName("pelican")
```

Default configuration

- set the base directory

```
1  Jafar::Datareader::DataReader
2      .setDefaultBasePath("/home/cyrille/laas/Data")
```

- set the series name

```
1  Jafar::Datareader::DataReader
2      .setDefaultSeriesName("pelican")
```

- set the serie number

```
1  Jafar::Datareader::DataReader
2      .setDefaultSerieNumber(11)
```

Read data

```
1  dr = Datareader::DataReader.new
2  sr = dr.getStereoReader( 0 )
3  img = sr.left.loadImage( 0 )
4  Qdisplay::showimage( img )
```

Features

- display images

Features

- display images
- display vectors overlay

Use

```
1   dr = Datareader::DataReader.new
2   sr = dr.getStereoReader( 0 )
3   imgL = sr.left.loadImage( 0 )
4   imgR = sr.right.loadImage( 0 )
5
6   viewer = Jafar::Qdisplay::Viewer.new
7   imageviewL = Jafar::Qdisplay::ImageView.new(imgL)
8   viewer.setImageView(imageviewL)
9   imageviewR = Jafar::Qdisplay::ImageView.new(imgR)
10  viewer.setImageView(imageviewR, 1, 0)
11
12  shape = Qdisplay::Shape.new(Qdisplay::Shape::ShapeRectangle, 10, 10, 5, 5)
13  shape.setColor(0,255,0)
14  shape.setLabel("Hello World !")
15  imageviewL.addShape(shape)
```

Why modules?

Definition

a modular software promotes *the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. This practice brings about an equally wide-ranging degree of benefits in both the short-term and the long-term* **Wikipedia, component-based software engineering**

Why modules?

Definition

a modular software promotes *the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. This practice brings about an equally wide-ranging degree of benefits in both the short-term and the long-term* **Wikipedia, component-based software engineering**

Benefits

1. ease of maintenance
2. increase reuse
3. functional barrier between algorithms
4. dependencies optimization
5. a shelf for each box

Implications

- At conception, do a functional separation in your work:

Implications

- At conception, do a functional separation in your work:
  - Your aim: a nice soft for meshing

Implications

- At conception, do a functional separation in your work:
  - Your aim: a nice soft for meshing
  - Naive method: one module

Implications

- At conception, do a functional separation in your work:
  - Your aim: a nice soft for meshing
  - Naive method: one module
  - Good method: geometric precepts module, drawing tools module, meshing algos module

Implications

- At conception, do a functional separation in your work:
  - Your aim: a nice soft for meshing
  - Naive method: one module
  - Good method: geometric precepts module, drawing tools module, meshing algos module
- At implementation, do check previous work:

Implications

- At conception, do a functional separation in your work:
  - Your aim: a nice soft for meshing
  - Naive method: one module
  - Good method: geometric precepts module, drawing tools module, meshing algos module
- At implementation, do check previous work:
  - Doesn't exist: create at the right place

Implications

- At conception, do a functional separation in your work:
  - Your aim: a nice soft for meshing
  - Naive method: one module
  - Good method: geometric precepts module, drawing tools module, meshing algos module
- At implementation, do check previous work:
  - Doesn't exist: create at the right place
  - Doesn't suit: extend

Implications

- At conception, do a functional separation in your work:
  - Your aim: a nice soft for meshing
  - Naive method: one module
  - Good method: geometric precepts module, drawing tools module, meshing algos module
- At implementation, do check previous work:
  - Doesn't exist: create at the right place
  - Doesn't suit: extend
  - Conflicts: conciliate

Collab, what?

Definition

Collaborative software is software designed to help people involved in a common task achieve their goals

Benefits, Metcalfe's law

the more people who use something, the more valuable it becomes

Implications

- on you

Implications

- on you
  - you are a developer

Implications

- on you
  - you are a developer

Implications

- on you
  - you are a developer
  - you a user

Implications

- on you
  - you are a developer
  - you a user
  - you are both

Implications

- on you
  - you are a developer
  - you a user
  - you are both
- from you

Implications

- on you
  - you are a developer
  - you a user
  - you are both
- from you
  - understand and use svn: from shell or from gui

Implications

- on you
  - you are a developer
  - you a user
  - you are both
- from you
  - understand and use svn: from shell or from gui
  - write *well*: adopt the C++ coding standard http://www.possibility.com/Cpp/CppCodingStandard.html
  - *document* your work: we provide you doxygen, "just" comment your code

Implications

- on you
  - you are a developer
  - you a user
  - you are both
- from you
  - understand and use svn: from shell or from gui
  - write *well*: adopt the C++ coding standard http://www.possibility.com/Cpp/CppCodingStandard.html
  - *document* your work: we provide you doxygen, "just" comment your code
  - write unit tests: these are important

What are unit tests?

From wikipedia: *In computer programming, unit testing is a method of testing that verifies the individual units of source code are working properly.* **Wikipedia, Unit testing**

- Test the behavior of individual functions

What are unit tests?

From wikipedia: *In computer programming, unit testing is a method of testing that verifies the individual units of source code are working properly.* **Wikipedia, Unit testing**

- Test the behavior of individual functions
- As much as possible independent tests

What are unit tests?

From wikipedia: *In computer programming, unit testing is a method of testing that verifies the individual units of source code are working properly.* **Wikipedia, Unit testing**

- Test the behavior of individual functions
- As much as possible independent tests
- Automatic

Why are unit tests important?

- Make sure your code does what you want it to do

Why are unit tests important?

- Make sure your code does what you want it to do
- Speed up development and optimizations/refactoring

Why are unit tests important?

- Make sure your code does what you want it to do
- Speed up development and optimizations/refactoring
- Make sure nobody else breaks your feature

Why are unit tests important?

- Make sure your code does what you want it to do
- Speed up development and optimizations/refactoring
- Make sure nobody else breaks your feature
- Tests are documentation

Write an unit test

Add a file test_suite/test_MyFirstClass.cpp :

```
1  #include <boost/test/auto_unit_test.hpp>
2  #include <kernel/jafarTestMacro.hpp>
3  #include "playmodule/MyFirstClass.hpp"
4
5  BOOST_AUTO_TEST_CASE( test_MyFirstClass )
6  {
7    MyFirstClass mfc;
8    JFR_CHECK_EQUAL( mfc.add(1, 2), 3 );
9  }
```

Then:

```
1      make test
```

Documentation: a brief introduction to doxygen

Generatl tags:

- **@ingroup** declare a function to be part of a group
- **@ref** give a reference to an other function/class

Function tags:

- **@param** describe a parameter
- **@return** describe the return parameter

Lets document MyFirstClass

```
1  /**
2   * This is my first class in Jafar. @ref add is
3   * the most important function.
4   * @ingroup playmodule
5   */
6  class MyFirstClass {
7    public:
8      /**
9       * This function add two numbers.
10      * @param u first number
11      * @param v second number
12      * @return the addition of u with v
13      */
14     int add(int u, int v) const;
15 };
```

Conclusion

Jafar is yours, it is up to you to make it improve or collapse