Biorob Coding Standards

When writing software in C or C++, please follow the coding standards as provided in this document. This allows your code to be more easily reused for further research purposes. If there is only one thing that you take away from this document, then it should be:

WHATEVER YOU DO, AT LEAST BE

CONSISTENT

Note: some rules can be bent, others can be broken...

---

1 Language

Any text, be it comments, function names, classes or variable names should be in English only. This applies even to debugging statements as this will make it easier for everybody to read your source code and help with any problems you might encounter.

2 Naming conventions

The naming schemes as listed in tables 1 and 2 should be used.

Table 1: C++ Naming Conventions

Type	Format	Example
Classes	Each word should be **title** cased	RobotController
Functions	Each word should be **title** cased. Don't include *get* in a getter function name	Weight()
Data Members	All data members in classes should start with **d_** to make it clear they are *data* members	d_filename
Variables	Start with **lower** case, each word should be **title** cased	inputFile
Constants	All **upper** case, separate words by underscores	STEP_SIZE
Enums	Each semantic word should be **title** cased	FormatLarge

Table 2: C Naming Conventions

Type	Format	Example
Functions	All **lower** case, separate words by underscores	calculate_weight
Variables	All **lower** case, separate words by underscores	input_file
Constants	All **upper** case, separate words by underscores	STEP_SIZE
Enums	All **upper** case, separate words by underscores	FORMAT_LARGE

3 Indentation

Always use tabs for level indenting. For statements that require a new level of indentation (if, else, else if, for, while, do, etc.). Always use curly braces (even for one line statements within the enclosing statement) and start the beginning curly brace on a new line.

Correct

```
if (true)
{
    cout << "I found the truth!" << endl;
}
```

Incorrect

```
if (true)
    cout << "I can't handle the truth!" << endl;

if (true) {
    cout << "I can't handle the truth!" << endl;
}
```

When breaking expressions over multiple lines, always use spaces to level out the remaining expression so that editors with different settings for the tab width still render the text with the correct indentation.

3.1 C function definition/declaration indentation

In C, function declarations should have all separate function arguments on a new line. The function return type should also be on a separate line from the function name. The first argument is on the same line as the function name, and any subsequent arguments are indented on the same level as the first argument. Make sure to align function argument types and names for better readability.

Correct

```
static void
my_function (char const *filename,
             unsigned    size)
{
    fprintf (stderr, "File: %s, %u\n", filename, size);
}
```

Incorrect

```
static void my_function (char const *filename, unsigned size) {
    fprintf (stderr, "This is wrong on so many levels...\n");
}
```

3.2 C++ class declarations

The same indentation rules as for statements hold for class declarations. Thus the starting curly brace should go on a new line. Also indent after the *private:*, *protected:*, *public:* keywords. Put data members at the top of the class declaration if possible. **Never** put *using namespace* statements in your class headers. Doing so could introduce name collisions when somebody includes your header.

Never make any data members public. Rather, provide accessor functions. Data should always be encapsulated by the class, and the class is responsible for maintaining its data and keeping it valid.

Correct

```
class Data
{
    std::string d_filename;
    size_t d_size;

    public:
        Data();
    private:
        void openFile(std::string const &filename);
};
```

Incorrect

```
class Data {
    public:
    std::string filename;
    size_t size;

    Data();
    private:
    void openFile(std::String const &filename);
};
```

4 Use of Spaces

Always use spaces around binary operators. Also **always** use a space after a statement and after a comma.

Correct

```
if (alpha + beta * theta < saturation && in_flux (theta, 2))
{
    /* Do something here */
}
```

Incorrect

```
if (alpha+beta*theta< saturation &&in_flux(theta,2)) {
    /* What is this unreadable statement! */
}
```

In C it is preferred to have a space after the function name of a function call, but not doing so is permitted (as long as you are **consistent**).

5 Documentation

As mentioned before, always write comments in English. Try to comment relevant and non-trivial parts of your code as well as possible, so that other people understand why you did something. For documenting reusable pieces of code (libraries or larger applications), please use Doxygen for documenting your functions and classes.

6 File Names

Use .hh and .cc for C++ headers and source files. Use .h and .c for C headers and source files.

7 Headers

Make sure to encapsulate declarations in C headers in "*extern "C"* {}" if you expect these to also be used from C++ programs. Also **always** put include guards in your header files.

Example C header

```
#ifndef MY_HEADER
#define MY_HEADER

#ifdef __cplusplus
extern "C"
{
#endif


void my_header_init ();


#ifdef __cplusplus
}
#endif

#endif /* MY_HEADER */
```

8 Compiling

When compiling, make sure to compile with generating warnings (-Wall) and while developing, try to compile with -Werror and make sure you resolve any warnings. Before releasing your code, **make sure** that your code compiles without any warnings! Try to use makefiles for compiling your code (in addition you can use utilities such as cmake or autotools), so that other people can just issue *make* to recompile the code if necessary.

9 Good coding conventions

- Avoid global variables. Rather pass variables to functions or declare variables static and provide functions that act on the variable instead
- Don't write large functions. Try to split up larger pieces of code in separate functions that each serve a specific purpose and focus on each function behaving correctly. You can avoid many, hard to find, issues this way
- Use **meaningful** names. Except for iteration variables (such as i), always use meaningful names for variables and functions. Don't use variables such as 'a', 'b' or function names such as 'do_it'

10 Common C++ coding practices

- When declaring a member function which is virtual in the parent class, repeat the *virtual* keyword in the inherited class. This makes it clear from looking at only that inherited class which of the functions in that class are virtual
- Initialize data members in the initializer part of the constructor
- Make good use of STL (in particular containers) in favour of the plain C ways of doing things
- Try to be ANSI C++ compliant to improve cross-platform quality of your code

The C++ Annotations (http://www.icce.rug.nl/documents/cplusplus/) are usually a good reference for C++ (there are also packages available on Ubuntu and Debian). Another good reference for coding practices in C++ is: http://www.parashift.com/c++-faq-lite/.

11 Licensing

Your source code should be licensed under a GPL 2 compatible license. The prefered license is GPL 2 which allows version 2 or any later version of the GPL license. Provide at least a COPYING file containing the full GPL 2 license in the root of your project directory. It is also good practice to include an excerpt of the license at the top of each source and header file of your project, although this is not strictly necessary.

Example License Excerpt

```
/*
 * <filename>
 * This file is part of <project>
 *
 * Copyright (C) <year> - <author>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330,
 * Boston, MA 02111-1307, USA.
 */
```