Sensor fusion Software based on Principal components

SYSC 5709 Course Project
Coding Conventions

Course Supervisor:
Dr. Christina Ruiz martin
Department of Systems and Computer Engineering,
Carleton University, Ottawa, ON, Canada

Collaborators:
Scott Jordan, Tarun Panuganti

Revision 1.0

Student Name: Jaspal Singh
Student Name: Vinay Venkataramanachary

Submitted on Dec 17 2019

Table of Contents

Contents

1. Introduction

This Document outlines the coding conventions followed in the development of this sensor fusion software. Coding conventions were proposed by mutual agreement of relevant stakeholders involved and product owners.

2. Folder Structure

Below convention was used to build the folder structure of Git and GitHub repositories

```
/bin
/build
/doc
/src (can have subfolders for better organization)
/include (can have subfolders for better organization)
/data
/test
    /include (can have subfolders for better organization)
    /src (can have subfolders for better organization)
    /data
/lib (can have subfolders for better organization)
makefile (Just one makefile for the whole project)
README
```

Additional features will include additional folder as needed

3. Code Comments

1. Anyone reading the comments can easily understand the design and purpose of the code.
2. Consistency of the comment style. will use /* */
3. Every file should have a comment at the top describing its contents in a general way
4. Need to use Doxigen notation (the one that matches style /* */)
5. Punctuation, spelling, and grammar are important.

Should have at least one detailed description, one brief description, comments that would not be included in the documentation (when generated automatically from the source code), one detailed description after member, one brief description after member. Need to write comments for all the functions according to Doxigen notation

4. README File

The minimum requirements are as follows:

1. Use Markdown notation for better readability.
2. Organization: University name
3. Authors:
4. Brief description of what the program does
5. Source File Organization, list of folders and files
6. Instructions on how to compile/build the code and the test. You should also explain dependencies and refer the user/developer to the "place" where these instructions can be found (if they are too long).
7. Detailed instructions for the user and developers should be included in the documentation folder. In the readme file, you should specify where to find them

5. Coding Conventions

1. Variables and function names:

All variable and function names: snake style (i.e. lower case letters and numbers and underscore)

Should be self-explanatory

Example: int size_array

2. Constants:

All capital letters with "_" separator.

example: int SIZE, #define PI 3.14

3. Indentation:

4 spaces

4. If statements:

Always use curly brackets, even if there's only 1 statement.

Starting curly bracket at the end of the first line. The ending curly bracket should go on a new line in the same column where the if begins. Example:

```
if (value == 1) {
   value = 2;
}
```

5. Constants on RHS in equality/inequality check:

```
if (value == 1) {
   value = 2;
}
```

6. Order of function parameters:

Inputs, Inputs&Outputs, Outputs.

7. Lines < 80 characters (approx and indentation does not count towards the 80 characters)

8. File names:

Use the .h extension for header files.

Use .c for source files.

The names of the files must be lowercase without spaces to avoid operating system conflicts.

9. Avoid using global variables

10. Looping

Use the simplest and more readable form.

Always use curly brackets, even if there's only 1 statement.

11. Nesting: open and close brackets in the same column

12. Soft-Coding preferred over Hard-Coding

13. All files must have header guards

```
#ifndef __MY_HEADER__
#define __MY_HEADER__
... #endif // __ MY_HEADER__
```

14. Every file must include all other headers it needs

15. Use 0 for integers, 0.0 for reals, NULL for pointers, and '\0' for chars