Software Documentation

Software name: SensorFusion_Demo

Version: 1.0

GitHub Repository: *https://github.com/Monbert/sensorFusion_Demo*

Organization: Carleton University

Authors: Kai Xia & Hui Tang

Date: Dec 18, 2019

Contents

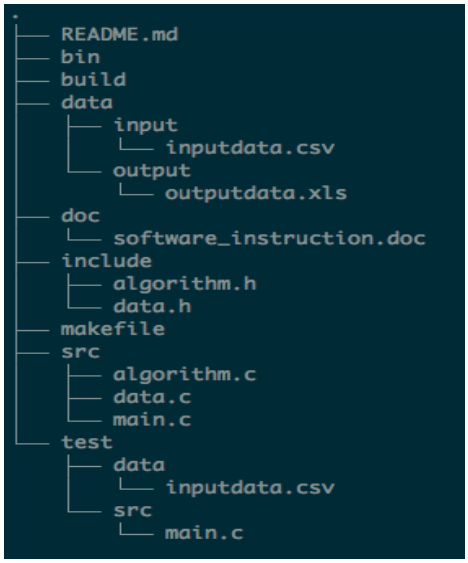
1. Introduction

Sensor fusion, is defined as the process of integrating information from multiple sources to produce the most specific and comprehensive unified data about an entity, activity or event, which can not be obtain by a single sensor. By properly administrating multiple sensors and their measurements, it can achieve improved accuracy and more specific inferences than those only using a single sensor.

This software can calculate a single and weighted value from your input data which include many sensors values. And the value generated by the algorithm is representative of the correct raw data, which means there are some wrong measures in raw data.

2. For Users

2.1 Folder structure



bin/: Binary files of the project will be stored and maintained here

build/: Contains executable files

data/: Contains input and output data files

doc/: Contains software documentation

include/: Contains header files

src/: Contains source code

test/: Contains test source code to test program

makefile: Contains how to compile and link the program

README: Contains general description of the software and usage instructions

2.2 Pre-Requisites to run the software

Platform compatibility: Mac OS 10.12.6, Linux Ubuntu 18.04, Windows 10

1. GNU packages (make, cmake)

2. gcc compiler

2.3 Steps of Installation and Running

Step 1: Download the whole project
Step 2: cd to this project location in the terminal
Step 3: clean residual files -- terminal command "make clean
Step 4: compile all files -- terminal command "make all"
Step 5: run the executable file -- terminal command "make run"

2.4 Steps of Testing Functions

Here we prepare two set of sensor data to test functions, comparing expected results and actual results to tell the function PASS or FAIL in the part of testing.

Step 1: cd to this project location in the terminal
Step 2: clean residual files -- terminal command "make clean"
Step 3: compile all files used in testing-- terminal command "make test"
Step 4: run the executable testing file -- terminal command "make run_test"
Step 5: Input 1 or 2 which set of senor data we prepared in advance to use testing function implementations in the algorithm, example testing results as below pictures:

```
(base) Albert@dhcp-129-136:~/Desktop/compile_test/sensorFusion_Demo_folder_structure$ make run_test
./bin/tests
Please choose which set of data is going to be tested: (input 1 or 2, then press enter)
1
```

```
**test compute_fused_output function**
**param[in]- sensor - it is a struct of storing all sensor data,*weight - it is a pointer to an array of
weight coefficient of every valid sensor,n - it is a number of all sensors**
**param[out]-the value of fused output as the fianl result**
**expected result:if it run successfully : the fused_output is exactly what we expected**
**expected result:if it fails: the fused_output is not what we expected**
**Factual result:53.008965**
**actual output:0.000000**
**test result of this function:FAIL**
```

2.5 Format of Input and Output

Input:

All input values are saved in a data/input/inputdata.csv file. Users should follow the pattern of input file like below example:

	A	B	C
1	time	sensor_name	value
2	13.2	1	53.2
3	13.2	2	52.6
4	13.2	3	52.7
5	13.2	4	53.2
6	13.2	5	52.8
7	13.2	6	53.3
8	13.2	7	49.7
9	13.2	8	53.1
10	13.5	1	53
11	13.5	2	53.1
12	13.5	3	52.6
13	13.5	4	53.2
14	13.5	5	52.8
15	13.5	6	53.6
16	13.5	7	55.7
17	13.5	8	53.5
18			

Output:

The results computed and fused are stored in a data/output/outputdata.xls file with certain style like below example:

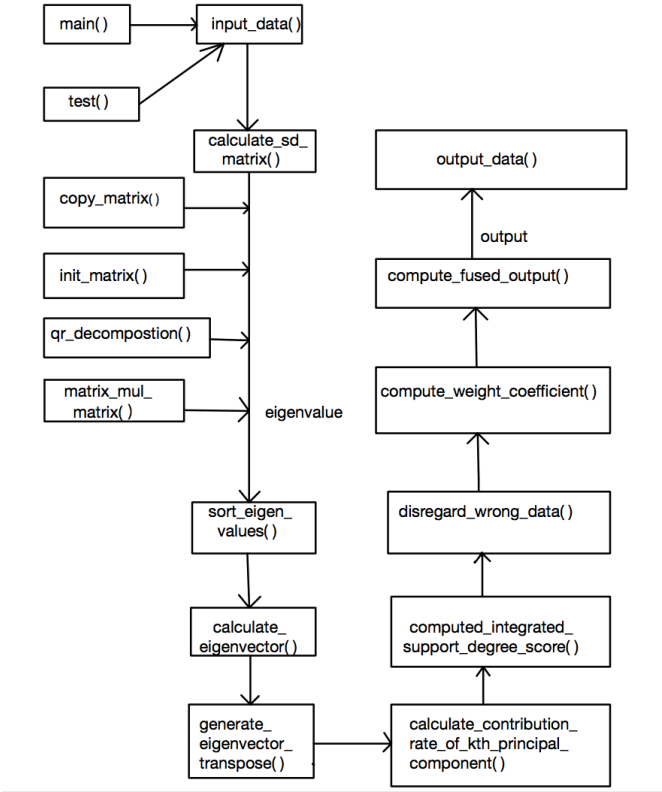
	A	B
1	time_24h	fused_value
2	13.2	53.008965
3	13.5	53.133483
4		

Notes: if there is no correct named input file in your data/input folder, users will get error so that they can check it.

```
./bin/executable
error! There is no inputdata.csv file, please check if it is existed.
```

3. For Developers

3.1 Software Design Diagram



3.2 Details of Function Implementation

data.c: (the following functions are included in data.c)

input_data(): *read the values of each set of sensor data from inputdata.csv file, and put those data into a struct called sensor_list*

output_data(): *get the fused values from algorithm and output them into a output.xls file respectively*

repeat(): *loop the main algorithm to get several timestamp fused results*

*function details of parameters in and return values can be found in every function comment where it is on the top of every function in source code file src/data.c.

algorithm.c: (the following functions are included in algorithm.c)

calculate_sd_matrix(): *get the value of the support degree matrix that is given by every two different sensor data in a timestamp*

copy_matrix(): *copy a matrix to a same matrix, that we use it in the part of calculating eigenvalues*

init_matrix(): *initialize the struct we called matrix*

qr_decompostion(): *QR decompostion is a methods to decompose a matrix into an orthogonal matrix Q and an upper triangular matrix R, that helps to calculate eigenvalues*

matrix_mul_matrix(): *multiply a matrix to another matrix with same value, that helps to calculate eigenvalues*

sort_eigen_values(): *sort all eigenvalues we calculated descendingly*

calculate_eigenvector(): *calculate eigenvectors using eigenvalues we calculated*

generate_eigenvector_transpose(): *transpose eigenvectors to help following computing*

calculate_contribution_rate_of_kth_principal_component(): *get a vector of values of alpha_k*

computed_integrated_support_degree_score(): *get a vector of integrated support degree score*

disregard_wrong_data(): *since there are some invalid sensor data from input, it helps to disregard those invalid sensor and shows the number of them*

compute_weight_coefficient(): *get a vector of weight coefficient*

compute_fused_output(): *get the final results of fusing sensor data in a timestamp*

*function details of parameters in and return values can be found in every function comment where it is on the top of every function in source code file src/algorithm.c.

3.3 Coding Convention

In the source codes of this project, we follow the certain coding conventions which announced on cuLearn. Generally can be written down as following:

1. Variables and function names use snake.

2. All constants use capital letters with "_" separator.

3. Indentation uses 4 spaces.

4. 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.

5. Constants on RHS in equality/inequality check.

6. Order of function parameters: Inputs, Inputs&Outputs, Outputs.

7. Each Line should around less than 80 characters.

8. The names of the files must be lowercase without spaces to avoid operating system conflicts. Use the .h extension for header files. Use .c for source files.

9. Avoid using global variables.

10. Use the simplest and more readable form of looping. Always use curly brackets, even if there's only 1 statement.

11. Open and close brackets in the same column when use nesting.

12. Soft-Coding preferred over Hard-Coding.

13. All files must have header guards.

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.

3.4 Code Comments Rules

1. Comments should be easily understanding the design and purpose of the code.

2. Consistency of the comment style.

3. Every file should have a comment at the top describing its contents in a general way

4. Use Doxigen notation to write comments for all the functions which at least need one detailed description and one brief description.

5. Take care about punctuation, spelling, and grammar are important.