ECE 4122/6122 Hmk #5

(100 pts)

Section	Due Date
89313 - ECE 4122 - A	Nov 3rd, 2019 by 11:59 PM
89314 - ECE 6122 - A	Nov 3rd, 2019 by 11:59 PM
89340 - ECE 6122 - Q	Nov 5th, 2019 by 11:59 PM
89706 - ECE 6122 - QSZ	Nov 5th , 2019 by 11:59 PM

Notes:

A sample input file is provided with the assignment.

You can write, debug and test your code locally on your personal computer. However, the code you submit must compile and run correctly on the PACE-ICE server.

Submitting the Assignment:

See Appendix C.

Grading Rubric

AUTOMATIC GRADING POINT DEDUCTIONS :

Element	Percentage Deduction	Details
Undocked Yellow Jackets	-1 for each undocked Yellow Jacket	The yellow jackets may not be able to dock in time.
Destroyed Yellow Jackets	-2% for each one	The yellow jackets destroyed during the simulation.
Does Not Compile	40%	Code does not compile on PACE-ICE!
Does Not Execute as expected and/or Output is wrong format	10%-90%	The code compiles but does not produce correct outputs or does not execute as expected.
Clear Self-Documenting Coding Styles	10%-25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score - (20/24)*H	H = number of hours (ceiling function) passed deadline note : Sat/Sun count as one day; therefore $H = 0.5*H_{weekend}$

Battlestar Buzzy

Battlestar Buzzy has just finished defending GaTech from the evil space bulldogs. It was a fierce battle and only **seven** yellow jackets are still operational and need to dock as soon as possible. The yellow jackets are all damaged, flying blind, and having propulsion issues. Your mission is to develop a distributed **MPI** program that will be able to guide the yellow jackets safely back to Buzzy so that they can dock.

The following rules must be followed:

1) Create a PBS script file to allocate 4 nodes with 2 processors per node. Call the output file **out.dat**.

2) The main process (rank=0) reads in the following from an input file (all values are in MKS units).
    a. The name of the input file is **in.dat** and is in the same folder as the executable.
    b. The first line contains the length of time in seconds that the yellow jackets have to dock before Buzzy must jump to ludicrous speed.
    c. The second line contains the maximum thrust (F_{max}) in Newtons that each thruster can provide.
    d. The third line contains the x y z location, initial speed and the initial normalized direction vector of Battlestar Buzzy. The speed and direction of Buzzy will remain constant while the yellow jackets are docking.
    e. The next seven lines contain the same information for each of the yellow jackets.

3) The kinematics of each yellow jacket must be controlled by a different process, other than the main process. The main process calculates the new location for Buzzy every second and is used to gather and broadcast the location of Buzzy and the seven yellow jackets. Each yellow jacket is assume to have a mass of 10,000 kg.
    a. The yellow jackets have three thrusters. One for each direction (xyz), that can be used to change its direction by applying a one second burst of constant thrust from $-F_{max}$ to $+F_{max}$. The thrust force (F) can be changed every one second as needed. The thrusters are independent of each other and can be fired at the same time if needed with different thrust force values.

b. There is an issue with the propulsion system and the thrusters are randomly misfiring. Each time you use a thruster, the value you use needs to be multiplied by a random number between 0.8 to 1.2 to simulate the misfiring.

c. Use Newton's 2nd Law to determine the acceleration of the yellow jacket in each direction.

$$F_{x,y,z} = m * a_{x,y,z}$$

d. Use the equations of motion for constant acceleration in each direction (xyz) to determine the location and velocity of the yellow jacket every second.

$$x = x_o + v_{xo}t + \frac{1}{2}a_x t^2$$
$$v_x = v_{xo} + a_x * t$$

4) Each second the yellow jackets must report their current status, xyz location and their next F_x, F_y, F_z values to Buzzy in the main process.

5) Each second Buzzy in the main process broadcasts the status and location of itself and all seven yellow jackets, so that the yellow jackets are aware of the status and location of everyone. Battlestar Buzzy should always have a status of 1.

6) The yellow jackets must stay at least **250m** away from other "**active**" yellow jackets. If they come closer than 250m, it is considered a collision and both yellow jackets are considered destroyed. In this case, the processes for the two yellow jackets change their status to destroyed and their position stays constant at the point of destruction. Destroyed or docked yellow jackets are not a consideration for collision avoidance.

7) A yellow jacket will automatically dock with Buzzy once its distance is less than **50m** and the following two conditions are meet:

a. It must be flying in the same direction ($cos\theta > 0.8$) as Buzzy. You can calculate this value using the dot product between the velocity vector for Buzzy and the velocity vector of the yellow jacket.

$$cos\theta = \frac{\vec{v}_{Buzzy} \cdot \vec{v}_{Yello\ Jacket}}{|\vec{v}_{Buzzy}||\vec{v}_{Yello\ Jacket}|}$$

b. $|\vec{v}_{Yello\ Jacket}| < 1.1 \cdot |\vec{v}_{Buzzy}|$

8) If a yellow jacket comes within 50m of Buzzy and it does not meet conditions 7a and 7b then it is consider to have crashed into Buzzy and has been destroyed. In this case, the process for the yellow jacket changes its status to **destroyed** and the position stays constant at the point of destruction. If a yellow jacket has successfully docked then its status is updated to **docked** and its location is the same as the last broadcast Battlestar Buzzy location.

9) The program ends when one of the following occurs:
   a. The allowed time to dock has expired,
   b. All yellow jackets have either been destroyed or successfully docked with Buzzy.

10) While the program is running, the main process (rank=0) should output the status and location of each yellow jacket every second to the console. The output must be in the comma delimited format shown below:

    rankID, status, x, y, z, F_x, F_y, F_z

    RankID and status should be output as integers and x, y, z, F_x, F_y, F_z should be output using scientific notation with **6 decimal points** of precision. For example:

    1, 1, 2.345643e4, 8.765456e2, 5.879478e4, 1.001231e2, 2.456793e1, 4.003234e2

A yellow jacket can have one of three status values: **Active = 1, Docked = 2, Destroyed = 0**.

<u>Appendix A: Coding Standards</u>

<u>Indentation</u>:

When using *if/for/while* statements, make sure you indent 4 spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

<u>Camel Case:</u>

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird). This applies for functions and member functions as well! The main exception to this is class names, where the first letter should also be capitalized.

<u>Variable and Function Names</u>:

Your variable and function names should be clear about what that variable or function is. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:

Every file should have the following header at the top

```
/*
Author: <your name>
Class: ECE4122 or ECE6122
Last Date Modified: <date>

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.

Appendix B: Submitting Assignment

Step-By-Step Submitting a Tar.gz

PLEASE NOTE: All the following instructions are run on PACE. If you are struggling with any of the steps, please come see the TAs during office hours. Better to start and test early than wait until the last minute.

Below is a step-by-step tutorial on how to create the .tgz file for Homework 4 for ECE 4122/6122. Additionally, if you do not use a header file for a solution, you do not need to submit a header file. Please adjust these instructions accordingly. The tarball should contain a folder with your buzzid containing a subfolder for the problem with the corresponding .cpp and .h files.

Please make sure that the problem is in a subfolder with the naming conventions:
<FirstName_LastName>_Hmk5

Create a subdirectory named *buzzid* in the current working directory by executing the following command in the shell:

```
$ mkdir buzzed
```

Create a text file named **manifest.**

Please make sure that the **manifest** file is a plain text file and not a rich text file. Microsoft word will generate a rich text file. The easiest method to create a plain text file is to use a command line editor such as vi, vim, or nano.

(If you want a tutorial, these can be useful: vi: http://heather.cs.ucdavis.edu/~matloff/UnixAndC/Editors/ViIntro.html, vim: https://openvim.com/, nano: https://www.howtogeek.com/howto/42980/the-beginners-guide-to-nano-the-linux-command-line-text-editor/)

Populate the **manifest** file with the following:

```
buzzid/<FirstName_LastName>_Hmk5/*.cpp
buzzid/<FirstName_LastName>_Hmk5/*.h
```

(***PLEASE NOTE***: this is subject to change with depending on your class section. The wildcard (*) character will grab all the .cpp and .h files you generated for each problem.)

Now you need to construct the correct file structure for the submission. This means that you need to put all the homework files into your *buzzid* folder. Execute the following lines in the shell:

```
$ cp -a <FirstName_LastName>_* buzzid
```

Many people have had issue with this step. You need to make sure the naming conventions are consistent to avoid potential problems.

It is now time to tarball your submission. If all the other steps have gone smoothly execute the following command:

```
$ tar -zcvf buzzid-hmk5.tgz $(cat manifest)
```

You can now check the new tgz file just generated with the following command:

```
$ tar -ztvf buzzid-hmk5.tgz
```