Lab #3: Your Task is a Task

In this exercise, you will write a new task class. The exercise will take one week; you will work in teams as usual. Skills to practice in this lab include the following:

- Creating a task class with the ME405 task class structure
- Instantiating multiple tasks for multiple motors
- Inter-task communication in a thread-safe manner
- Documenting code with Doxygen
- Testing your code thoroughly to ensure that it is reliable

1 The Project

Write a task class which controls a motor using the motor driver you wrote in the previous lab. You should be able to create multiple motor drivers and motor tasks as follows, with as many parameters as needed:

```
// Create the motor driver objects, one for each motor
my_motor_driver* p_motor_1 = new my_motor_driver (param1A, param2A, param3A);
my_motor_driver* p_motor_2 = new my_motor_driver (param1B, param2B, param3B);

// Create motor driver tasks, one for each motor
new my_motor_task ("Motor1", ..., p_motor_1);
new my_motor_task ("Motor2", ..., p_motor_2);
```

Your motor task class should receive commands to set its power level through a thread-safe shared variable of type `shared_data<int16_t>`. For examples of how these shared variables work, look at the inter-task communication example code on PolyLearn. There should be another shared variable which controls the mode (braking, power, etc.) You can use a `bool`, a `uint8_t`, or an enumeration type (nice and readable) as you prefer. It is recommended to take an existing task class such as `task_brightness`, renaming and modifying its files – but make sure you don't leave any features of the old task when making your new task files, including comments and `#define`'s.

Next, modify `task_user` so that you can use the user interface to control the mode and power level of each motor independently. This means that the user interface task is writing the shared variables which are read by the motor control tasks. Note that if you would like to use one or two potentiometers with the A/D converter to control the motor speeds, that's fine; if you'd prefer to use clever keyboard commands, that's fine too. However, if you use a potentiometer, it must be read within the user interface task.

When this task has been completed, your motor driver tasks should be *very* modular: you can copy the motor driver and motor task files into any old AVR program and use them with little if any modification, as long as there are DC motors connected to VNH3SP30 or

compatible driver chips. The only difference between different motor driver tasks will be the parameters given to the task constructor as each task object is created.

You need to test your motor drivers thoroughly. *Try* to cause problems so that they can be fixed or, if you can't fix them before the due date, at least reported. Known bugs are better than hidden ones. Your testing should include running the motors at different speeds, braking one while powering the other, and so on in as many combinations as you can test.

2 References

These are some of the same references that were used in Lab 2, reprinted here for convenience. Information about the ME405 software library is at
`http://wind.calpoly.edu/ME405/doc/`.

Information about the FreeRTOS real-time operating system can be found at
`http://www.freertos.org/a00106.html`.

The code library used with our AVR compiler is avr-libc; there is a reference for all its available functions at
`http://www.nongnu.org/avr-libc/user-manual/modules.html`.

A webcomic which has computing related humor (often excellent) is at
`http://xkcd.com`.

3 Deliverables

On the due date, you'll demonstrate your program in lab and turn in the following:

- A one page memo describing what your program does. The memo should discuss the results of your tests in some detail.
- A printout of the Doxygen documentation for your new task class. This should not include the title page nor documentation for other classes. It should be printed double sided. *Have you looked over the printout before handing it in?*
- Printouts of the source files you have written or substantially modified, also double sided.