CSCE 436/836: Embedded Systems
Lab 4: PID, Serial Console, Scheduler

Instructor: Carrick Detweiler
carrick _at_ cse.unl.edu
University of Nebraska-Lincoln
Spring 2011

Started: March 29, 2011
Due: **Friday**, April 8, 2011 at **5pm**

1 Overview

This lab will consist of three main parts. As always, they are somewhat interrelated, so you should read through the complete lab before starting and plan how to approach the lab. In the first part of the lab, you will interface with a serial console library that gives you an easy way to include commands that you can execute over the serial port or radio. Next, you will learn to use a task scheduler library, which will allow you to execute tasks at specific intervals (milliseconds or seconds). Both of these libraries are from SEAMos, `http://code.google.com/p/seamos/`, which is a light-weight, configurable embedded operating system I have developed. Finally, you will implement proportional, integral, derivative (PID) control for the rotation of the hovercraft.

2 Safety

Remember and review the safety instructions from the previous labs.

3 Serial Console [30pts.]

The serial console provides a console/terminal-like interface for embedded systems. It lets you type in different commands (with arguments) to execute, similar to the setup you created in previous labs to allow rebooting the board and also commanding individual thrusters. You have also seen me use it in class and in some sample compiled code I sent you.

The console system consists of 4 files, `console.h`, `console.c`, `console_custom.h`, and `console_custom.c`. You should only need to modify the latter two files, although feel free to explore the other two files, which have the core code, to see how the console system functions. You will find examples of how to use the console system in `console_custom.c`. Note that `console_custom.c` assumes that you have a file `uart.h` that includes the standard-named uart functions we have been using and defaults to sending on both ports 0 and 1. This can be changed by editing `console_custom.c`.

The idea behind the console system is that you *add* or register functions that you want to be able to execute from the console. When you add them, you give a long name and short name that, when typed, will cause execution of a particular function. You also give a short description of any parameters and general help info. These are used when you type `h` or `help` from the console to output a set of available commands.

It will be up to you to read through the source files (mainly `console_custom.c`) to determine how to complete the following tasks. These should be completed on the 5V Atmel; you can optionally add the console system to the 3.3V processor if you find it useful. The first step will be to add the source files to your existing `SCons` build scrips.

Question: Create a console command to test each of the thrusters by briefly turning each one. Call it `testthrust` and have a short command of `tt`. This function should briefly turn on each thruster (one at a

time) so that you can see that they all are working. How did you initialize the console system? How did you tell it to process commands? How did you add a command?
Question: Create a console command to set the lift thruster to a specific thrust level from 0-100% (as an argument). Call this `lift` and have the short command be `l`. You may want to remap 0 thrust to the minimum thrust level that turns on the thruster and 100 to the maximum allowed thrust level. Describe how you were able to obtain the argument on the command line.
Question: Modify the `lift` command so that if no lift value is given, it will report the current lift level. How did you do this?

Implement similar functions to control each of the other thrusters. This is useful for testing, but not so much for direct hovercraft control. To control the hovercraft, create console command that switches from the console interface to a direct keyboard command interface (which you have already implemented). You should be able to switch back and forth between the two modes.
Question: How did you enable switching between the console control and direct control modes?
Question: Finally, implement console commands to read the sensors on the hoverboard (gyro, current, voltage, etc). Describe these functions.

4 Scheduler [30pts.]

A scheduler allows scheduling events to occur at a particular time or to repeat at a particular interval. You have been given a non-preemptive, multi-tasking scheduler. This means that you can schedule multiple tasks, but only one will run at a time. In addition, it will run until it completes, potentially blocking other tasks that would like to run. Because of this, you should make sure that no task runs for an extended period of time as this will starve any other tasks that are trying to execute.

The scheduler consists of the main `schedule.h` and `schedule.c`, as well as `scheduleConsoleCmds.h` and `scheduleConsoleCmds.c` which provide some console commands to add or delete events from the scheduler queue. Add these to your build system and make sure to initialize them.

The scheduler requires that `scheduleProcessMS(ms)` is called approximately every millisecond and is passed an argument which is the current time, since startup, in milliseconds. Use one of the timers to generate an interrupt every millisecond. In this interrupt, increment a variable that will serve as the global time in milliseconds. You should then call `scheduleProcessMS(ms)` from your main while loop (not in the interrupt). Pay close attention to the fact that you need to make sure that the global millisecond counter continues to increment, even when `scheduleProcessMS(ms)` is running. In addition, the `ms` variable needs to be carefully handled to prevent it from being updated at the same time that `scheduleProcessMS(ms)` is using it.
Question: Describe how you made sure your `ms` counter continually updated while `scheduleProcessMS(ms)` was running and how you made sure that the `ms` value was always valid when passed to `scheduleProcessMS(ms)`. Why would it be a bad idea to call this function directly from the interrupt code?

Similarly, there is a function `scheduleProcessSec(sec)` that handles events that are scheduled with second granularity. You can use the millisecond counter (divided by 1000) as input to this function. Setup your main loop so that both the millisecond and second schedule processing functions run.
Question: You can schedule an event to run at a particular number of milliseconds or seconds from now or periodically. What are the functions to do this? Describe their arguments.

To be able to add events from the console, you need to add a "possible event." You can do this by using the function `scheduleAddPossibleEvent(...)`. You can also add events or delete events from other events (an event can even delete itself).
Question: There are multiple ways to delete events. What are the methods?
Question: Create an event that blinks one of your LEDs briefly at 2Hz. How does this work?
Question: Implement a console command that uses the scheduler to drive straight for a specified amount of time. How did you achieve this?
Question: In the previous section, you created a console command that tests the thrusters by turning each one on for a short time. While it is doing this, however, nothing else can run (including scheduled events).

Create a new set of events that sequentially tests each thruster. You might turn on each thruster for a short period, perhaps 250ms, and then leaves all thrusters off for 1 second before moving to the next thruster to test. Describe how you implemented these sequences of thruster tests in the scheduler without blocking other events that may be running (such as the LED blinking event). You may also want to turn on your other LED when a thruster is supposed to be on so that you will know if a thruster is not working.

5 PID Control [30pts.]

In the previous lab you had to rotate to a particular angle. Some groups did this "open loop," by just creating a mapping for how long it takes to rotate a particular angle and just rotating for that time. Other groups use the gyro to obtain some feedback. This is what we are going to do in this section. We will do this by implementing a proportional, integral, derivative (PID) controller for rotation. This will enable your hovercraft to hold a particular angle even when it is disturbed (e.g. if you try to turn it, or to compensate for rotational moments caused by translational thrusters). You can read about PID control on the wikipedia page: `http://en.wikipedia.org/wiki/PID_controller`.

Here is a brief summary. You should make sure to read this whole section before starting as some of the work for the later questions will be helpful in answering the earlier ones. If you have a target angle a_t and a current angle (as read by your integrated gyro data) of a_i, then you can control the rotational thrust, r, of your hovercraft according to:

$$r = P(a_t - a_i) \tag{1}$$

P is the proportional scaling term of the controller (and can be positive or negative). With a P controller, if you are far off from your target angle, it will have high thrust and when it gets closer the thrust will decrease. This can work well, but if P is too low it will never reach the target and if it is too high, it will overshoot and oscillate.

By introducing a derivative, D, term oscillations can be damped:

$$r = P(a_t - a_i) + D(a_{i-1} - a_i) \tag{2}$$

where a_{i-1} is the angle from the previous time step and D is the derivative scaling term of the controller. The value $(a_{i-1} - a_i)$ is basically a numeric derivative of the error and it tends to slow how fast the output term r will change (damps oscillations).

A purely PD controller often works well and I recommend that you first try a PD controller before trying to introduce an integral, I, term. The problem is that a PD controller will not always achieve the set point [1]. Adding an integral term can ensure that you reach your target, but at the cost of potentially adding oscillations and having to deal with issues such as windup. Adding the I component will yield:

$$r = P(a_t - a_i) + I(integral_i) + D(a_{i-1} - a_i) \tag{3}$$

where, $integral_i = integral_{i-1} + (a_t - a_i)$, and $integral_0 = 0$. The windup problem is that if there is an offset (e.g. you hold your hovercraft in one position and don't let it rotate for a while), then the integral term will continue to grow unbounded. So typically, you need to bound the integral term by a maximum value to mitigate this problem.

In essence, a PID controller is not too difficult, however, there are a number of practical issues that you must address when using one. The first, is tuning the PID values. Wikipedia has a number of suggestions as to how to tune the controller. My preferred method is to first try a PD controller (almost always works better than a purely P controller). The idea is that you first set $D = I = 0$ and then tune P until it oscillates slightly, then add D until the oscillations are damped. On the hovercraft, the I term probably isn't needed, but if you find that there is an offset from the final position, then you can add a little bit of I to make sure it achieves the target.

[1] Think of a PD controller as pulling a car up a hill with a spring, no matter how stiff the spring, if you put one end where you want the car to end up, it will never get there.

Question: Did you need to use an I term? What were your final values for the PID parameters?

There are a number of other practical issues when controlling rotational systems by degrees. The first, is that there is a discontinuity between 0 and 360 degrees. The second, is if your target is 180 degrees off from your current position, should you go clockwise or counter clockwise? The problem is exacerbated if there is sensor noise (you could end up oscillating 180 degrees off of your target angle). To solve the second problem, one approach is to have a fixed thruster output whenever you are further than 90 degrees off of the target and then to switch to your PID control when you are closer to your target.

Question: Describe how you addressed rotations of near 180 degrees from your current angle.

To address the discontinuity between 0 and 360, you can always work in terms of relative angles when doing control instead of absolute angles. One way to think about this is to always have your target be 180 degrees. To rotate, you then just set your current angle to be offset from 180 degrees. For instance, if you are currently at 350 degrees and you want to go to 30 degrees, you can set your target angle to 180 degrees and set your current angle to 140 degrees. In essence, you are then working in relative angles far away from the discontinuity. There are other approaches to this as well.

Question: Describe how you addressed rotating around an absolute angle of 0 degrees.

Another problem to address is deadbands. If you leave your PID controller on all the time, then the hovercraft will maintain its angle. However, very small offsets may cause current to go to the thrusters without them actually moving. This is a waste of energy. Thus, it is good if you have deadbands so that the thrusters will never just be on a very small amount (and not turning). You can implement this in the PID controller or in the functions that control your thrusters.

Question: Create a new group of console commands (sub-commands) to set the P, I, and D values individually (call them `setp`, `seti`, and `setd`). If no value is given, then report the currently set value.

Question: Create a new function and console command that takes an offset angle and rotates that particular number of degrees, call the console command `rotate` or `rot` for short. It should take at least plus-minus 180 degrees. Describe how you implemented this function.

Question: Have your PID controller run as a scheduled event. It can run at up to 1KHz if run every millisecond. Do you need it to run this fast? What is the minimum frequency you can run it at and get good results?

Question: What happens if the PID controller does not run at the scheduled time? This could occur if some other event runs too long. How can you compensate for this?

Question: Plot outputs of your current angle, target, and rotation thrust levels for different PID values when changing the target angle. Make sure to include an output from your final PID controller. Also include plots from a purely P controller when it has a value that causes oscillation and when it does not cause oscillation. Include any other plots that you find useful.

Question: Implement a console command to drive an equilateral triangle. Do this by enabling the angle PID controller and then scheduling a series of events that enables the forward thruster, followed by a 120 turn, etc. Since the PID controller is always running, it will compensate for any angular drift that your forward thruster may cause.

6 To Hand In

You should designate one person from your group as the point person for this lab (each person needs to do this at least once over the semester). This person is responsible for organizing and handing in the report, but everyone must contribute to writing the text. You should list all group members and indicate who was the point person on this lab. Your lab should be submitted by email before the start of class on the due date. A pdf formatted document is preferred.

Your lab report should have an introduction and conclusion and address the various questions (highlighted as ***Question:***) throughout the lab in detail. It should be well written and have a logical flow. Including pictures, charts, and graphs may be useful in explaining the results. There is no set page limit, but you should make sure to answer questions in detail and explain how you arrived at your decisions. You are also

welcome to add additional insights and material to the lab beyond answering the required questions. The clarity, organization, grammar, and completeness of the report is worth **10 points** of your lab report grade.
Question: Please include your code with the lab report. Note that you will receive deductions if your code is not reasonably well commented. You should comment the code as you write it, do not leave writing comments until the end. I recommend structuring your code such that for every section of this lab, a different function is called from the main loop. Then you will be able to comment out just one function to change between sections in the lab and you won't have to delete working code from previous parts of the lab.
Question: For everyone in your group how many hours did each person spend on this part and the lab in total? Did you divide the work, if so how? Work on everything together?
Question: Please discuss and highlight any areas of this lab that you found unclear or difficult.