Lab 0x02 - Tasks, Shares, and the Scheduler

New Attempt

- Due Oct 23 by 12:10pm
- Points 30
- Submitting a file upload
- File Types pdf and py

Overview

The purpose of this assignment is to incorporate the hardware drivers you wrote last week with a set of tasks running cooperatively using a priority scheduler. Your specific goal is to use the hardware drivers from the last lab along with new scheduler libraries to create a data-collection tool for benchmarking some of Romi's dynamics. That is, this lab will be like a combination of the techniques used in HW 0x00, Lab 0x00, and Lab 0x01.

Note: For many students this lab is the most challenging of the entire quarter because it is the lab where you are asked to do the most relative to your experience and progress in the class. Teams should recognize this ahead of time and manage their time accordingly to make the lab go as smooth as possible. All future labs will follow the same techniques and coding style as developed in this assignment so it is in your favor to start building a good foundation.

Recall the coding structure outlined recently in lecture, split into rows of the table below for high-level, middle-level, and low-level code. In this assignment you will spend most of your time in the middle-level layer, assuming that the low-level drivers you wrote in the previous lab function properly; you'll also spend a bit of time setting up the high-level main script to create and run tasks with the scheduler.

Lab Coding Structure

Level	Example Files	Purpose
High	`main.py`, `cotask.py`, `taskshare.py`	Main is used to create objects, create tasks, and then run the scheduler in perpetuity. No real implementation of program logic should go here.
Middle	`task1.py`, `task2.py`, `task3.py`	These files implement all the tasks for your program as separate generator functions and share data using objects from the share and queue classes to send data back and forth. There will likely be no hardware interaction at this layer.
Low	`motor.py`, `encoder.py`, `IR_sensor.py`	These files implement all the hardware interface features but should not interact with shared

Level	Example Files	Purpose
		variables. Each driver should be utilized exclusively by one of the tasks in the middle level above.

Familiarization

Begin this lab by downloading the scheduler library, the shares and queues library, and some example code, all available from Dr. Ridgely's GitHub Repository: **https://github.com/spluttflob/ME405-Support (https://github.com/spluttflob/ME405-Support)**. To run the scheduler you will need `cotask.py` and to use the shares and queues library you will need `task_share.py`. The example file is `basic_tasks.py`. You may also benefit from scanning through the repository for other files that may be helpful either in this lab or in future labs.

In addition to the files available on Dr. Ridgely's GitHub page, you can find detailed documentation for the code as well: **https://spluttflob.github.io/ME405-Support/annotated.html (https://spluttflob.github.io/ME405-Support/annotated.html)**.

Before you move on to the assignment, run the example file (renamed to `main.py`) on your Nucleo hardware. Analyze the code with your lab team while observing the output of the example, cross referencing with the documentation for the scheduler and the shares and queues libraries. Examine closely the syntax for creating tasks, shares, and queues, and also how the tasks are added to the scheduler to run at a prescribed period. Later, you can choose to modify this file in-place, or start a new file for the lab assignment.

Objectives

The goal for this assignment is first, to automate a testing procedure to characterize some of Romi's dynamics, and second, to scaffold future lab assignments in which you will be performing closed-loop motor control.

There is much freedom in how you go about this assignment, but consider the following "workflow" for collecting data from your Romi hardware.

1. An "operator", sitting at a PC issues a command to a user-interface to start data collection. This command should be issued to a Python program running on a PC with its own UI that will control the serial port and interact with the MCU on your behalf.

2. Once commanded, Romi should perform a step response test. That is, a step-change in duty-cycle should be applied to both motors on Romi so that it drives forward; it's probably wise to set the duty-

cycle back to zero after one or two seconds so that it only drives a meter or so on your benchtop; aim for the shortest test you can run to guarantee steady output behavior. During the step response test, the microcontroller should be recording data from the wheel encoders and storing them in arrays for later use.

3. Once Romi has completed the step response and fully buffered the test data, it should send the data in a CSV formatted fashion back across the serial connection to your Python script. This data should then be archived for future retrieval and also used to create step response plots, similar to what you presented in Lab 0x00, to help you characterize Romi's driving behavior.

4. Your team may, optionally, choose to have an additional second test variety that has Romi spin around in a pirouette instead of drive in a straight line.

The entire process from issuing a "go" command to generating step response plots should be fully automated. This will require that you run a Python program on your PC to handle serial input rather than using PuTTY. However, it is ***strongly*** recommend that you start by getting things working from PuTTY before you incorporate the PC script.

USB vs Bluetooth

Since it is impossible to open the same serial port in PuTTY and within a Python script, you will only be able to work with one or the other at a time. One option is to test everything manually in PuTTY and then switch over to testing your Python script without having a serial port open in PuTTY for debugging.

An alternative approach is to use a secondary serial port. For example, if a dedicated Bluetooth serial port is used for data collection, the PC script can open that serial port to read data while you use PuTTY to access the primary serial port over USB. This will allow you to debug your code in PuTTY while the data is sent over the Bluetooth serial connection.

Assignment

1. Before you begin writing code, work with your lab group to develop a task diagram; recall from lecture that task diagrams document the period and priority of each task along with the inter-task communication variables (shares and queues) used to transfer data between the tasks.

   Consider including the following tasks:

   - One or more tasks responsible for reading encoder data, performing motor actuation, and eventually closed-loop motor control. Will it make more sense to have separate tasks for each of these aspects, or one task that handles sensing, control, and actuation? Should each

motor/encoder pair have it's own task?

- A task dedicated to user interaction via PuTTY through either a Bluetooth VCP or a USB VCP.
- A task dedicated to data collection.

2. Once your team has a good sense of what tasks and how many will be beneficial to your program, consider how each task will be implemented. It is common to set up every task as a finite state machine (FSM) to promote consistency and scalability for adding features and new behavior in the future. Even tasks that may not need the full benefit of a FSM should be implemented as a FSM with one "stub" state to allow more features to be added easily in the future. For any tasks that your team selects to be implemented using a FSM, draw a state transition diagram to document your FSM design. These FSMs can and should refer to the shared data sent between tasks.

   Note that both the overall task diagram and the state transition diagrams for each task will be required deliverables for each lab starting with this one.

3. Only after your team has settled on a good design should you move on to the coding for this assignment.

Deliverables

You will need to submit deliverables twice for this assignment, as described below.

Preliminary Deliverables

A preliminary deliverable, *due at the end of the lab period*, will be the task diagram and the state transition diagrams used to design your program. You can submit those through the following Canvas submission link: **Lab 0x02 - Diagram Submission (https://canvas.calpoly.edu/courses/161863/assignments/1396020)** .

Final Deliverables

The final deliverables for this assignment will include a memorandum and several attachments, due before the start of the next lab.

Memo Write a detailed but concise memo presenting your design choices for this assignment including some discussion of your task structure and selection of shared variables. Why did you choose the task

periods and priorities that you did? Why did you pick the specific shares and queues that you did?

Diagrams Include in the body of your memo updated copies of the task diagram your team has constructed along with updated copies of the state transition diagrams for each task implemented as a FSM. These diagrams should be continuously updated as you modify your code to always reflect the implemented design.

If you feel that the full resolution images for each diagram fit poorly within the body of the memo also include larger high-resolution images as attachments at the end of your memo that each fill up an entire landscape-oriented page.

Shared Variables Include a table in your memo that lists out each shared variable with columns describing the data stored in that variable along with the motivation or purpose of that variable in your task structure.

Task Profile Include a second table in your memo that shows information from the profiler that is built into the provided scheduler library. Have all your tasks run on time? Are the periods and priorities causing the behavior that you want?

Program Functionality Provide evidence that your program achieves the objectives mentioned above by including example CSV outputs, generated plots, and analysis of your testing.

Source Code Include at the end of your memo a list and description of source files you created or used in this project. Also include the source files themselves as attachments through additional file submissions on Canvas.

The deliverables will need to be submitted on Canvas before the beginning of your lab section on Thursday, October 23.