Assignment1 Implementation Guide

Sept. 9th, 2019
Tiancheng Ge

For setting up the MBot, please refer to the document located on Canvas under `files/Specs/Mbot document.pdf`. There is an overview of the code structure in that document as well.

This implementation guide will point you to the code for each task, and give you some hints. For this lab, the master program you will run for all the tasks (except Task 0 in the implementation guide) is `mobilebot` on the beaglebone. It subscribes to motor commands, publishes odometry, and starts the main control loop. However, you will need to write and run other programs together with `mobilebot`, to complete all the tasks.

Task 0: Test the motors

Connect to the beaglebone and run the test_motors program in `mobilebot-f19`, ensuring that the motors and motor drivers work.
Hint: You may need to revise the macros in `mobilebot-f19/commom/mb_defs.h`.
If the left and right motor are flipped, change `LEFT_MOTOR` and `RIGHT_MOTOR`.
If the rotation direction of the motor is wrong, change the polarity of the motor. You may need to change the polarity of the encoder, to make the odometry correct.

Task I: Implement Odometry

The code you need to modify is in `mobilebot-f19/commom/mb_odometry.*`.
Hint: You can check the correctness of the implementation, by running `mobilebot`. Try to print additional states in `mobilebot` for debugging.

Task II: Implement Your Visualization Tool

The code you need to modify is in `botlab-f19/src/apps/botgui`.
Hint: You may need to publish additional messages to indicate where the robot is making a turn, and let `botgui` subscribe to that message, and draw a square at the location when turning.

To subscribe to the messages published from the Mbot, you will need to install `bot-lcm-tunnel` on your laptop, which is available in the libbot library (https://github.com/RobotLocomotion/libbot.git)
Install it with `sudo make BUILD_PREFIX=/usr/local` to run it from the command line directly.
Run `bot-lcm-tunnel YOUR_MBOT_IP` to start the connection.

Task III: Implement Closed-Loop Velocity Control

The PID controllers are implemented in `mobilebot-f19/commom/mb_controller.*`. You will need to tune the PID parameters located in `mobilebot-f19/bin/pid.cfg`.

Hint: To generate the plots to submit, you will need to modify `mobilebot-f19/commom/mb_controller.*, mobilebot-f19/commom/mb_structs.*, mobilebot-f19/mobilebot`, etc. You may need to publish additional lcm messages to command the wheel velocities, and let `mobilebot` subscribe. These additional programs and messages may be reused by Task IV, but they will not be reused for Task V and VI.
To visualize the step response in real-time for debugging and tuning, you can run lcm-spy on your laptop, and click on the message you want to visualize. The gui of lcm-spy will then show up a small plot that visualizes all the scalers in the message.
You may log the velocities, and use MATLAB or Python to plot the step responses.

Task IV: Implement Open-Loop Control

Reuse the additional lcm messages and programs you write in Task III, write a controller that commands left and right wheel velocities separately. Run `botgui` on your laptop to record the odometry trajectory. The code written for this task will not be reused by Task V and Task VI.

Task V: Implement Closed-Loop Control with Odometry

The main piece of code you need to modify is `botlab-f19/src/mbot/motion_controller.cpp.`
This program runs on the Raspberry Pi. It commands forward and angular velocities directly. Accommodating to this one more level of abstraction, you will need to modify `mobilebot-f19/commom/mb_controller.c`, to calculate set-point wheel velocities, based on the commanded forward and angular velocities.
You will need to write additional messages, to get the forward and angular velocity feedbacks in `botlab-f19/src/mbot/motion_controller.cpp.` Implement the feedback controllers for driving straight and turning in `motion_controller.cpp` as well.
Hint: There is another program, `botlab-f19/src/mbot/drive_square.cpp,` implemented for you. Running this program together with `motion_controller` will make the robot drive in a square. What level of abstraction is `drive_square` providing? The `motion_controller` will be reused by A2, and this level of abstraction is very useful then.

Task VI: Implement Closed-Loop Control with Laser Scans

Modify `botlab-f19/src/mbot/rplidar_driver.cpp` to make it publish the pose of the robot in the world frame.
Modify `motion_controller.cpp` so that it takes the feedback of the pose in the world frame, instead of the odometry feedback.