COMP1010 Lab Tasks Week 3

After the basic tasks from last week now we move on to more interesting (and challenging) problems. Note that to complete these you will have to build on what you already have in terms of code and logic. You are encouraged to reuse as much of your code as possible by creating header (and source) files that group together tasks that you are likely to repeat often. For example bundle all of the logic that deals with low-level driving instructions into one file and expose this functionality via a header file that you will include in subsequent tasks. This makes your code more modular and manageable. In order to prompt you to do this from this week on we will not be awarding credit for work submitted as a single big C file.

Tasks

- Task 2.0. Make the robot draw a right triangle using the motor encoders, much like before, but have it move faster.
- Task 2.1. Make the robot follow a wall.
- Task 2.2 (optional). Make the robot drive down the middle between two obstacles (eg cardboard boxes) passing equidistant between them.

Optional tasks are there just for fun - they are not graded. When submitting on Moodle make sure everything compiles cleanly. Compress all required files (header files, C files that go with the header files and main C files) and submit the archive. Only include source files!

Hints

To help you along we have a couple of suggestions.

- In order to complete Task 2.0 you will have to re-structure and remove duplication from your previous code for Task 1.3 (since we will not be accepting everything as one big file anymore). The robot behaves well at low speeds, but you will quickly find out (even in simulation) that at high speeds you need to be smart and dynamically manage the voltage of the motors. For example if you want to make the robot go fast exactly one meter you will have to start fast and slow down at some point in order to avoid overshooting.
- Before you take on Task 2.1 make sure you are comfortable reading the sensors. Spend some time playing with them. We do not (yet) pose any restrictions as to how fast you need to follow the wall or how far away from it you need to be so it should be relatively simple to come up with an algorithm to do the following (we are not looking for anything very complicated).