CS639: Autonomous Robotics: Programming Assignment 1

Josiah Hanna

January 2026

1 Overview

In week 2, we learned about various simple control laws: bang-bang, P, and PID control. Now it's your turn to complete implementations of these algorithms on a simulated robot platform.

1.1 Note on Collaboration

You are free to discuss algorithmic details of implementations with each other, i.e., pseudocode. However, your final implementation must be your work alone – sharing code will result in a zero on the assignment.

1.2 AI Policy

You are free to use AI coding assistants on this assignment, however, the final submitted code will be viewed no differently than if you had written it yourself.

2 Setting up the Webots Simulator

Follow the instructions at https://cyberbotics.com/doc/guide/installing-webots to install the webots simulator on your computer. Once it is installed, your initial steps are:

1. Open Webots.
2. Download the provided project directory and open it in Webots. To open it, go to File >> Open World, and then select `<assignment_root>/worlds/turtle_world.wbt` where `assignment_root` is the top-level directory in the project directory.
3. You should see a TurtleBot in a rectangular, walled arena. If you run the simulation, the robot will move forward until it hits a wall.

3 Main Task

Your main task is to implement a controller that moves the TurtleBot around the perimeter of the walled arena for as long as the controller is ran. The robot should remain as close as possible to 0.25 meters away from the closest wall at all times. The robot can move either clockwise or counterclockwise around the arena. You will implement three different control laws to accomplish this and observe their relative strengths and weaknesses.

We have provided starter code in the project directory. To find this code, look under :

`<assignment_root>/controllers/turtle_controller/`

You should see the following files:

1. `turtle_controller.py` is a Webots robot controller that provides a wrapper around the robot's sensors and actuators. It serves two purposes: 1) to enable you to focus on implementing the control laws described in class without spending a lot of time learning the Webot's API and 2) to pre-process sensors and post-process controls to make them more interesting. DO NOT MODIFY THIS FILE!

2. `starter_controller.py` is where you will place your code. The main function in this class is called `step` and it is called by the `run()` function in `turtle_controller.py`. You will implement this function (and any necessary helper functions) so that the robot implements the desired behavior. A few helper functions have been written for you.

   (a) `get_filtered_lidar_reading()` filters out lidar values of `inf` which can occur when the robot is right next to a wall.

   (b) `compute_pose()` uses the filtered lidar values and robot's global heading to compute its global pose (x, y coordinate and orientation).

   You are not required to use these functions.

Note: you should not modify any provided files other than `starter_controller.py`. Doing so will cause your submission to fail when tested by the instructor.

4 Controllers

You will implement three types of controllers for wall following: bang-bang, P, and PID. Your Python file, `starter_controller.py`, implements a StudentController object. The constructor (`__init__()`) function takes an argument `control_law` that is cached in the variable `self.control_law`. Your implementation should use the specified control law for the robot to maintain the correct distance from the wall. At test time, we will run a test with each of the options "bang-bang", "P", and "PID."

Additionall requirements:

1. There is no required speed but your robot should complete a circuit around the walled arena in under 3 minutes (simulation time).

2. For all controllers, the input error signal is the distance to the closest wall (you may special case corners).

5 Rubric

This assignment is worth 10 points and these points are earned in the following way.

1. **Successful wall-following:** For all control laws, your robot approximately follows the prescribed path, repeatedly, around the arena. Points will be deducted for large deviations from the path, getting stuck in corners, etc. (4 points)

2. **Control Laws:** We will visualize a trajectory of the robot's path around the arena. Points are received for the path showing the characteristics of the controller used (e.g., bang-bang control will oscillate). (2 points each)

6 Hints and Answers to FAQs

1. You may modify any provided function in `starter_controller.py` as long as the API of `step()` remains the same.

2. The `compute_pose()` function is provided to help get you started. It uses a filtered lidar sensor to determine the robot's pose. While it mostly works, the lidar sensor occasionally fails and the pose is miscomputed. *You may need to detect and handle this!*

3. The pose is represented in a *right-handed coordinate system*. This means that the heading angle increases counterclockwise starting from the x-axis. So $\pi/2$ is pointing straight up.

4. The lidar angle uses a *left-handed coordinate system* relative to the robot's local coordinate system where the x-axis is straight ahead for the robot. So the lidar reading at positon 90 is directly to the right of the robot.

7 Final Submission

When you have completed your implementation, submit the file `starter_controller.py` on Gradescope.