Control of Mobile Robots
Project work
Prof. Luca Bascetta

Instructions

Each student should upload the project to WeBeep folder "Assignments/Project work", at least 3 working days before the date of the exam call, as a compressed zip file named "studentID_surname_name.zip", including:

- a PDF file, written in English or in Italian, answering to the questions (see below);
- a ROS workspace including all the code used to answer the questions.

If running a launch file or a script no results, or results that are different from the ones reported in the PDF file are generated, the project work gets zero points (projects are verified using ROS Noetic).
The project work can be submitted only once for each academic year. The mark obtained with the first submission is valid for all the academic year and cannot be changed submitting another solution (even in the case of zero points).

Problem

Consider a Turtlebot robot simulated using its unicycle kinematic model. The robot is provided with wheel motors and low-level controllers that allow to control the angular velocity of each wheel in order to guarantee that the robot performs the desired linear and angular velocities.
In order to simulate the motors and the low-level velocity controllers, the unicycle kinematic model should be extended with the following transfer functions

$$v(s) = \frac{1}{1+sT_a} v_{cmd}(s)$$
$$\omega(s) = \frac{1}{1+sT_a} \omega_{cmd}(s)$$

where v_{cmd} and ω_{cmd} are the robot linear and angular velocity commands, v and ω are the inputs to the unicycle kinematic model, and T_a is the velocity loops time constant. The value of T_a can be found in the parameter table.

Set up a ROS package "turtlebot_simulator", based on Boost Odeint library, to simulate the robot.
The package should include:

- a ROS node that simulates the robot using the extended unicycle kinematic model;
- a ROS node that allows to feed the simulator with simple (e.g., step) commands to test the model behaviour;
- a yaml file including model parameters;
- a launch file to run the simulator and the test node;
- a Python or MATLAB script to plot the test results starting from a bag file.

Set up a ROS package "turtlebot_traj_ctrl", implementing a trajectory tracking controller.
The controller is composed of a feedback linearisation law, based on the unicycle kinematic model, and a PI trajectory tracking controller with velocity feed-forward (PI controller can be implemented using Forward Euler discretisation). The trajectory tracking controller should be tested using an 8-shaped trajectory, generated according to the following equations:

$$x = a \sin\left(\frac{2\pi}{T}t\right)$$
$$y = a \sin\left(\frac{2\pi}{T}t\right) \cos\left(\frac{2\pi}{T}t\right)$$

where the values of a and T can be found in the parameter table.
The package should include:

- a ROS node that implements the trajectory tracking controller, including reference trajectory generation;
- a yaml file including model parameters;
- a launch file to run the simulator and the trajectory tracking controller;
- a Python or MATLAB script to plot the test results starting from a bag file.

Questions

1. Tune the trajectory tracking controller, selecting in an appropriate way the PI parameters and the sampling time, in order to achieve a good performance in terms of tracking error. Follow two steps: first use standard tools from control theory, then verify running simulations and varying the controller parameters if you can improve the first tuning. Report the values of:

   (a) the proportional gains K_{P_x}, K_{P_y};

   (b) the integral time constants T_{I_x}, T_{I_y};

   (c) the sampling time T_s;

   for both tuning steps, explaining the procedures you have followed to determine the controller requirements (crossover frequency, zero steady-state error, disturbance rejection, actuator effort limitation, etc.), and to tune the controller.

2. Report the results of a test of the trajectory tracking controller with the extended unicycle kinematic model. In particular, the following pictures should be reported:

   (a) a figure showing the reference and actual robot trajectory;

   (b) a figure showing the x and y components of the tracking error;

   (c) a figure showing the two control signals (velocity and steering);

   together with the value of the maximum tracking error in the x and y directions.
   Clearly and thoroughly comment the quality of the results, with particular reference to your tuning procedure or tuning requirements (are the results in accordance with the requirements? if no, why?).

3. Report:

   (a) the values of the parameters a, T, and T_a;

   (b) the names of the launch files and Python or MATLAB scripts one have to run in order to generate the results required in questions 1 and 2.

Evaluation

The project work is evaluated in this way.
First, a check is done on the solution to verify that all the questions have been answered, and the instructions have been fulfilled. Second, the solution is evaluated considering:

- the correctness of the requirements, and how they are explained and motivated;
- the correctness of the tuning procedure, and how it is explained and motivated;
- the coherence between the requirements and the results.

Note that, the aim is not to necessarily achieve the best possible tuning, but a good tuning supported by reasonable and well-motivated requirements.
Every choice in the design and tuning process should be thoroughly motivated.