"Standard" project proposal

This is the **standard** project relating to topics covered in class. Alternatively, it is possible to carry out an Advanced project, to be agreed individually with the teacher.

The proposed assignment for the exam project consists of TWO exercises.

- Solving the FIRST exercise is a NECESSARY condition for passing the exam and SUFFICIENT for achieving a basic score (depending only on the score obtained in written and oral). The first exercise is of a strictly individual nature.
- The proposal of a solution for the second exercise is ONLY considered provided that the first. It is a NECESSARY condition to aspire to a higher score.

1. Checking the Cart-Pole system.

For the first exercise, you are asked to scrupulously follow the instructions (specified below). However, it is required that the candidate is able to demonstrate adequate knowledge of the proposed solution and the techniques used.

2. Basic autonomous navigation.

For the second exercise a project outline is provided. The candidate is free to propose any variation from the example track, as long as you are able to justify your choices. The final judgment will take into account the complexity of the proposed problem and the quality of the execution.

DELIVERY

EXERCISE 1.

Develop a controller for the physical cart-pole system on a simulated environment.

- The Cart-Pole system consists of an inverted pendulum mounted on a moving cart. The cart is free to move back and forth along a fixed axis (y).
- The controller to be implemented has the objective of stabilizing the pendulum in the vertical position, acting on the movement of the trolley.
- Use the provided simulation environment (https://github.com/cscribano/gazebo_polecart_ros), based on Gazebo-classic and ROS2 for communication with the simulator.

We *recommend* proceeding as follows:

- Carefully study the documentation provided for the simulator (Readme from the github repo).
- Clone, then compile the provided ROS2 package using the colcon system, then run the environment via ROS2 as indicated by the documentation.
- Check that the ROS2 topics used by the simulator are actually visible, using the ROS2 command line interface (tip: list published topics, publish manual control as per documentation).

- Create a NEW ROS2 package to use for implementing the proposed solution.
- Implement the solution in C++ using the ROS2 APIs seen in class.
- Implement a new ROS node with publishers and subscribers for the appropriate topics.
- In particular:

- Read the pendulum state from the /cart/pole_state topic
- Update the control loop
- Post the control value to the /cart/force topic
- Implement the control loop by means of a simple PID controller

Suggestions:

- The state of the pendulum, published on the topic /cart/pole_state as a JointState message, identifies the angle (in radians) relative to the vertical position of the pendulum (as indicated in the documentation). The set-point is therefore a value of 0.0 radians.
- It is possible to move the cart by applying a force along the y axis (positive or negative) by publishing a Wrench type message on the /cart/force topic (as indicated in the documentation).

- Before starting the control cycle, it is recommended to reset the simulation to start from the initial condition with the pendulum vertical. For this purpose it is recommended to invoke the reset service (indicated in the documentation). It is advisable to invoke the reset when the implemented node starts, using the C++ API (ROS2 service CLIENT).
- As pointed out in the documentation, the effect of the applied force does NOT disappear even after the simulator is reset. It is therefore necessary to apply a force of 0.0 N on the y axis if you intend to cancel the effects of the previous checks and stop the carriage.
- The time delta (dt) useful for derivation and integration operations can be assumed equal to 50 milliseconds (taking into account a publication frequency of 20hz). • It is also possible (but not mandatory) to obtain dt as the difference between the Timestamps of two subsequent status messages.

Final notes:

“Tuning” the PID controller may require several attempts to find the optimal gains for stable control. IF it is not possible to tune the controller correctly (and therefore the pendulum will not be completely stable) the solution MAY still be acceptable PROVIDED THAT THE IMPLEMENTED SOLUTION IS STILL TECHNICALLY CORRECT.

EXERCISE 2.

Below is a POSSIBLE outline for the second exercise.

The candidate can follow the indications of the track or propose all the variations he wishes, provided that the level of complexity is comparable or higher and that the same fundamental technologies are used.

The only fundamental requirement is that a robot's autonomous navigation problem in a simulated environment is addressed, using ROS2 and similar tools covered in class.

EXAMPLE track

A simulated environment is set up using the Turtlebot3 simulator (seen in class) based on Gazebo. We want to create an autonomous system to navigate the surrounding environment in search of a known marker (of the ArUco type), which we want to locate within the map.

Using ROS2 and the Nav2 stack, implement the following behavior:

- The simulated environment is started, the Nav2 navigation stack is initialized if necessary
- The cube with the aruco markers (ref.) is positioned (also manually) at a point on the map. slide)
- Initially the robot navigates the map autonomously.
- Simultaneously the topic of the simulated room is analyzed to identify the Aruco marker.
- When the marker is identified, the pose is estimated (ref. slide), and the transformation (TF) of the marker is published in the global map system.

Suggested variations:

- The robot can navigate by visiting a series of fixed points provided in succession (EASY) or it is possible to dynamically calculate the next point to head towards (example: using the laser scanner, MORE ADVANCED).
- When the marker is found it would be preferable to automatically set the navigation to stop near the located marker. Alternatively, it is possible to stop on the spot or continue navigation in search of other possible markers.

Suggestions:

- Please refer to the slides of 7/6 for further details. The cubic marker model to be inserted into Gazebo and the example code for marker pose estimation are provided in the **"starter pack"** folder (on Moodle)
- Explicit use or creation of Behavior Tree is not necessary (topic not covered in lesson).

Final notes:

- The ability to propose and resolve variations with respect to the suggested scenario will be evaluated positively.

PROJECT PRESENTATION

- Projects can be discussed on a date to be agreed.
- For exercise 1 it is sufficient to bring the code and demonstrate in-depth knowledge of it.
- For exercise 2 it is possible to help yourself with a simple presentation, in particular if you original variations are proposed with respect to the track.

**** F.A.Q ****

1. Is it necessary to develop exclusively in C++?

- Yes. In the unfortunate event of unbridgeable gaps, contact the teacher.

2. I don't have a computer / my computer isn't up to par.

- In this case it is possible to agree on a project suited to the resources, or a project that uses the resources that can be provided to you (e.g., board jetson...)

3. I completed exercise 1, but exercise 2 is partial and doesn't work very well.

- Even if partial it is still more than nothing.

For any doubts not covered by this document do not hesitate to contact carmelo.scribano@unimore.it to obtain clarification.