6.3 Domain-specific control frameworks

ROS 2 control provides the tools required to integrate a generic controller with its inputs and outputs in a modular way. In many cases, controllers need to work together with additional components in order to obtain the desired robot behaviors. A moving robot, for example, usually only follows commands that don't bring it in collision with its environment. At the same time, as the control architecture grows more and more complex, developers need additional tools in order to be able to introspect and debug the system.

Because of this, frameworks were developed that provide multiple additional components required to control robots of a specific domain. We will take a quick look at the most popular ones: Nav2 and MoveIt.

6.3.1 Nav2

Nav2[85], successor of the "ROS Navigation Stack", provides tools that allow a mobile robot to navigate between two poses in its environment. The required components usually are:

- One or more dynamic maps of the robot's environment, where obstacles can be continuously added or removed.
- A localization system, capable of understanding the current position of the robot in a map, or to generate a new map using the robot's sensor information.
- A global planner, capable of generating trajectories connecting the robot origin to its target while avoiding the obstacles modeled by the map.
- A local planner, capable of generating commands for the robot's actuators based on the target trajectory and the obstacles in the map[13].
- Actions that are functional to the navigation: tracking moving obstacles, resetting the maps state, etc.

13: This dual-planning strategy is often referred to as "Hybrid Planning".

In Nav2 all these components are orchestrated by Behavior trees(BT). In particular, the same behavior tree library that we described in Chapter 4.7.2 is used: "Behavior trees cpp".

In a typical execution flow, the navigation BTs will update the maps with the current obstacles, generate a trajectory to the target, then repeatedly call the local planner and update the map until the target is reached. This sequence, however, is not guaranteed to always work: the robot might encounter unexpected obstacles that prevent it from planning a trajectory or following an existing one, the local planner might get stuck in a local-minima and so on. Because of this, the navigation BTs additionally need to integrate mechanisms to handle these scenarios.

Different kinds of robots operating in different environments with different requirements, will almost certainly have different planning needs. Because of this, allowing for the customization of the order of execution of different tasks is essential to any modern robotic framework.

Similarly to what happened for ROS 2 control, in Nav2 components are integrated within a single framework using plugins. This not only allows

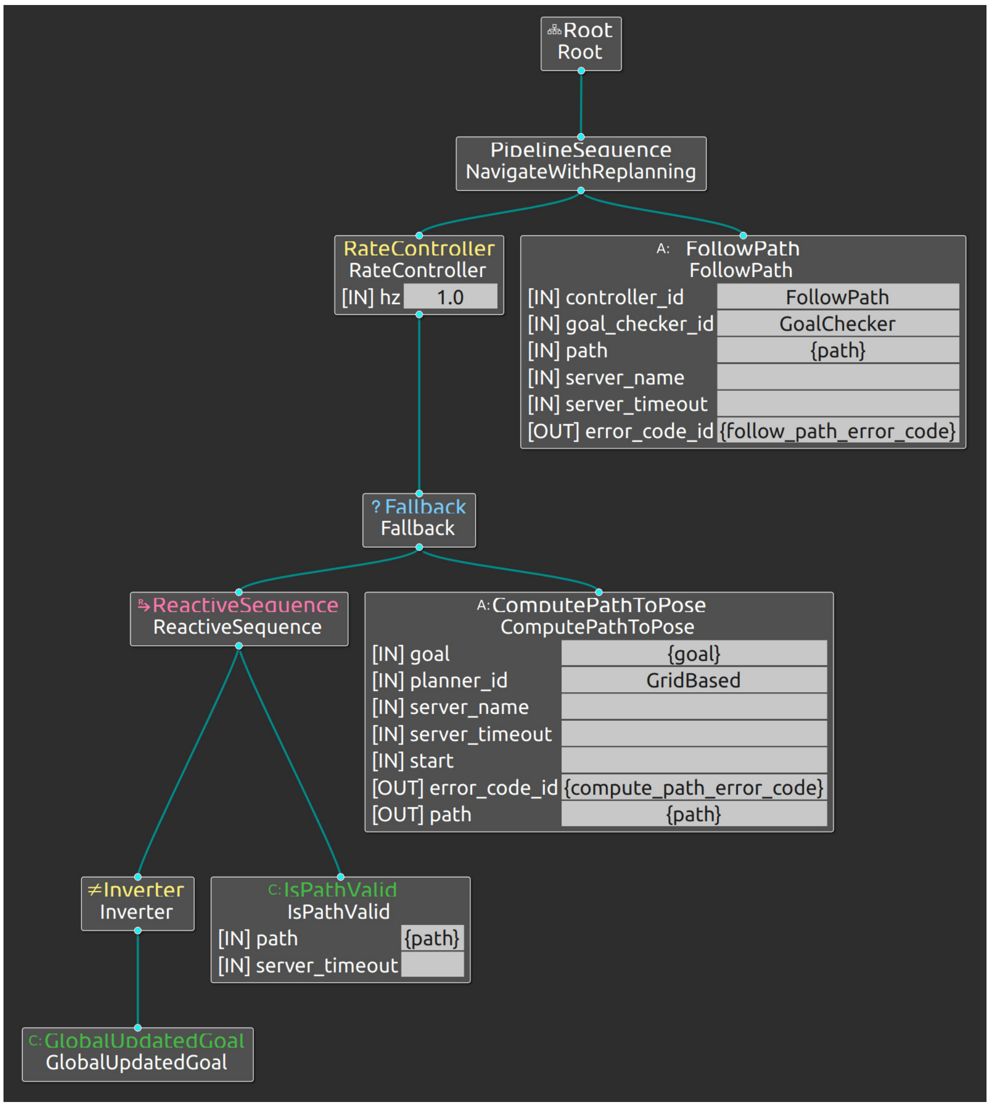

Figure 6.8: Example of simplified BT that generates a trajectory, follows it, and continuously checks for its validity.

to reduce the communication latency between them, but most importantly to share resources like the maps tracking the state of the obstacles in the environment (costmaps). The execution of all the components is synchronized using lifecycle-managed nodes.

For each of the required plugins, Nav2 provides a basic implementation to help you getting started with your project [14]. A complete list of the plugins implemented by the community can be found here[129].

14: Many of these aren't industry-ready solutions, but can easily be replaced with custom implementations.

If you are planning to integrate navigation functionalities within your robotic platform, using Nav2 will provide you a large set of tools ready to execute out of the box, while still allowing you to fully customize the system components and execution logic.

Since this is one of the largest open-source projects based on ROS 2, you should take a look at its architecture even if you are not planning to use it.

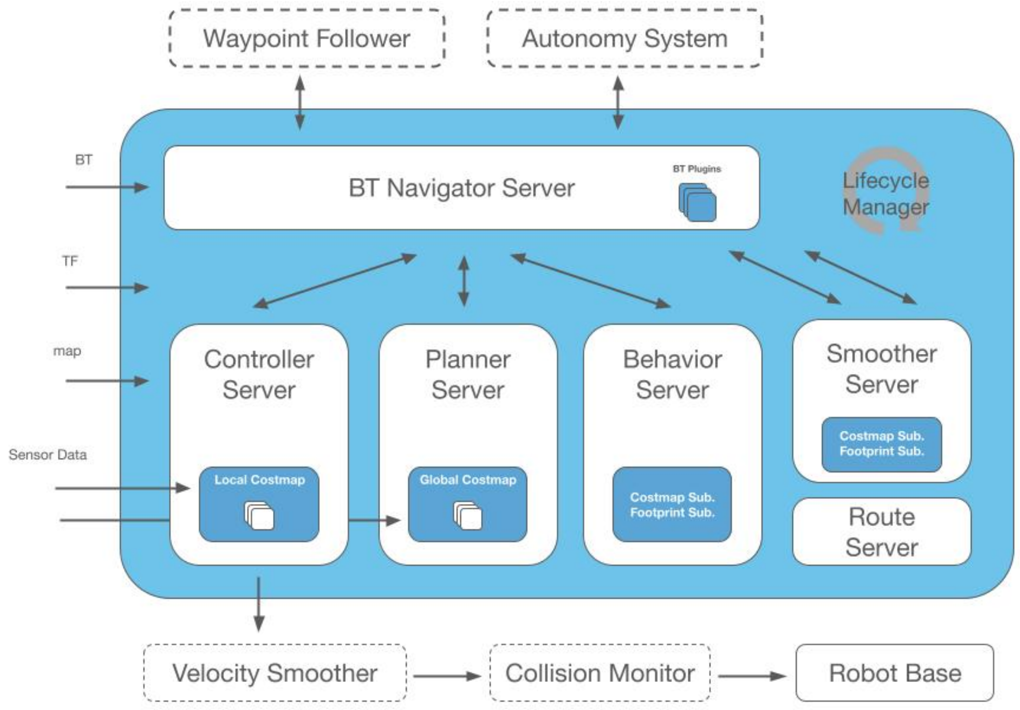

Figure 6.9: Nav2 architecture overview.
From: `https://docs.nav2.org/`