AA 274A: Principles of Robot Autonomy I
Final Project Check-in

Name: Alberto Guiggiani, Avrum Noor, Josie Oetjen, Karthik Pythireddi

SUID: alberto6, avrum, josieoet, karthik9

December 5, 2022

1 Backstory

In 2122, robots took over the world and kept all animals and humans, including Will Smith, hostage. However, among all the evil robots, one good one is still left, Turtlebot. Although admittedly inferior in terms of technology and design, his goodwill and courage make him the biggest hope for humankind. In a crucial mission, he has to rescue animals and humans from a so called zoo.

2 General Structure and Baseline Mission

We show the ROS graph diagram of our system in Figure 1. For the map exploration, we have set a collection of several waypoints needed to explore the map fully. The waypoint_publisher node publishes the collection of waypoints to our navigator node, which will be consumed by the planner and passes on a planned trajectory to the controller. Also, it implements the global state machine, which is shown in Figure 2. The navigator sends the velocity commands to the robot. Furthermore, the object_locator consumes the DNN output from the turtlebot detector to detect the animals, the stop signs, and the bananas. It publishes their respective pose and orientation to the waypoint_publisher and a zero velocity is commanded when locating a stop sign or banana to the navigator. The navigator switches to the STOP state whenever it receives a stop found message from the locator and handles the crossing logic to stop for 5 seconds. After stopping, it adds a crossing logic to ignore new objects for 3 seconds to prevent the turtlebot from stopping for the second time in front of the same object. Then, he switches back into the tracking state so he can stop at a new object again. Once the exploration is complete, the waypoint publisher prompts the user with the list of objects found and then creates navigation to the object locations based on the user selection. Arriving at the rescue position for each animal, the navigator switches to a stop state for 5 seconds. Lastly, the autonomous velocity command goes through a multiplexer that can ignore that command in favor of a manual command. We refer to the smart multiplexer extension for the details.

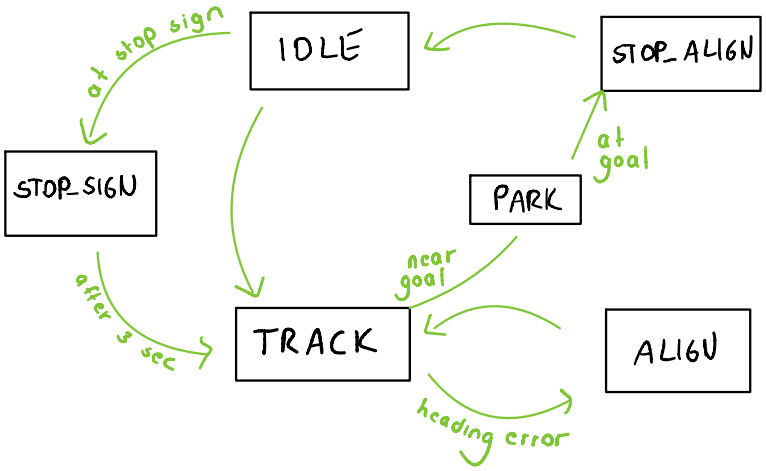

Figure 2: State Machine and Stopping Logic.
The figure shows the state diagram for the stopping logic.

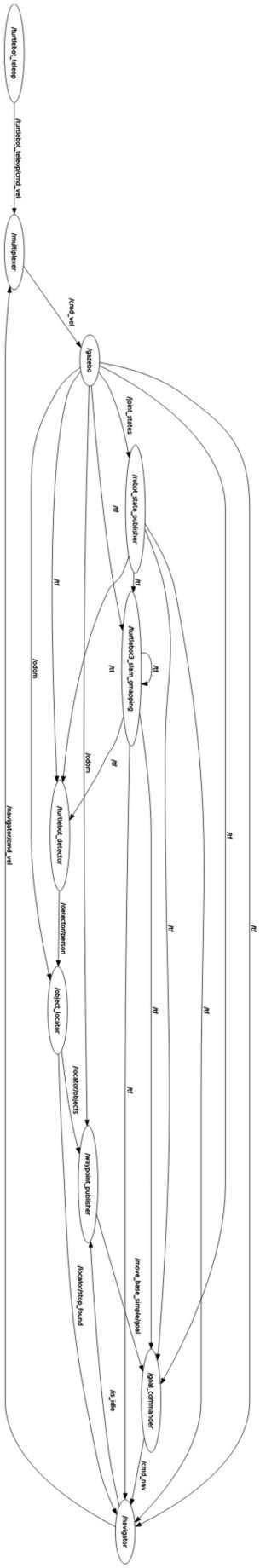

Figure 1: ROS graph diagram.
The figure shows ROS graph diagram of our system.

3 Extensions

3.1 Stopping at Stop Signs and Bananas

Despite his high focus on completing the mission quickly, turtlebot obeys traffic rules and stops at stop signs. Also, his weakness is his love of bananas, so when he finds one, he can't resist stopping briefly to snack on it. For the implementation, we refer to the preceding baseline chapter.

3.2 Smart Multiplexer

We designed a multiplexer with three modes of operation.

1. Autonomous control: It routes the robot autonomously by consuming the commanded velocity from the navigator.
2. Manual control: It uses the keyboard commands like "a s w x" to manually teleport the robot.
3. Hybrid control: It toggles the robot between the autonomous and manual modes seamlessly depending on the latest message received. We also added a 0.5-second grace period after receiving a manual input to avoid the autonomous system interrupting the user request.

We show a state diagram for the multiplexer in Figure 3.

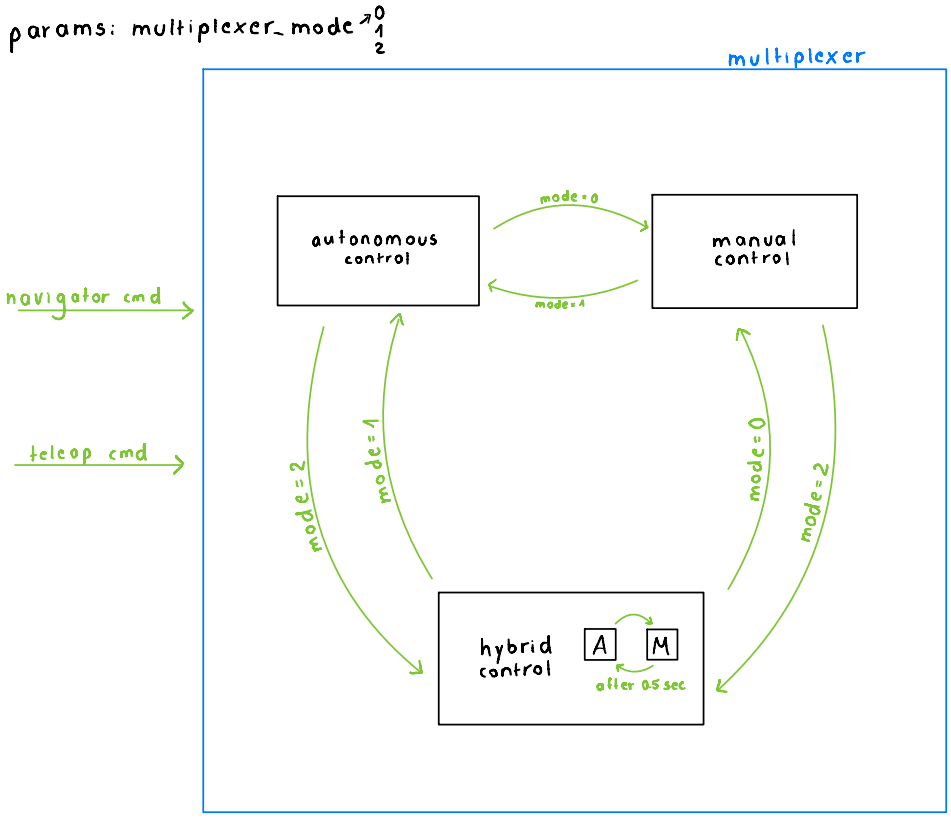

Figure 3: State Diagram Smart Multiplexer.
The figure shows the state diagram for the smart multiplexer extension.

3.3 Interactive Interface and Polite Animals

After the exploration, the system (aka turtlebot) informs the user which animals he found and asks which animals to rescue. The user then enters the animals he wants turtlebot to rescue with his keyboard in the desired order. If animals entered were not found, the system gives feedback on which animal is missing. Otherwise, turtlebot starts the rescue mission and prints the current status of the mission. Animals are delighted when rescued and say thank you with cute pictures. To implement this, we write a Python function that makes use of the input function and contains various ASCII art functions.

3.4 Live Data Introspection, Parameter Tuning, and Command Center Extension

We implement an extensive live data introspection with PlotJuggler. Having comprehensive data visualization is an essential tool for a robotic system. Plot Juggler is a lightweight visualization tool that directly connects to any running ROS topic and allows for extensive data visualization and live signal processing, like filters and math operations. Also, we integrate a complete GUI for live parameter tuning to maintain an overview and intuitive control over all parameters. Furthermore, to have a good overview in rviz, we visualize the point cloud, the goal marker, IMU, and a laser scan.