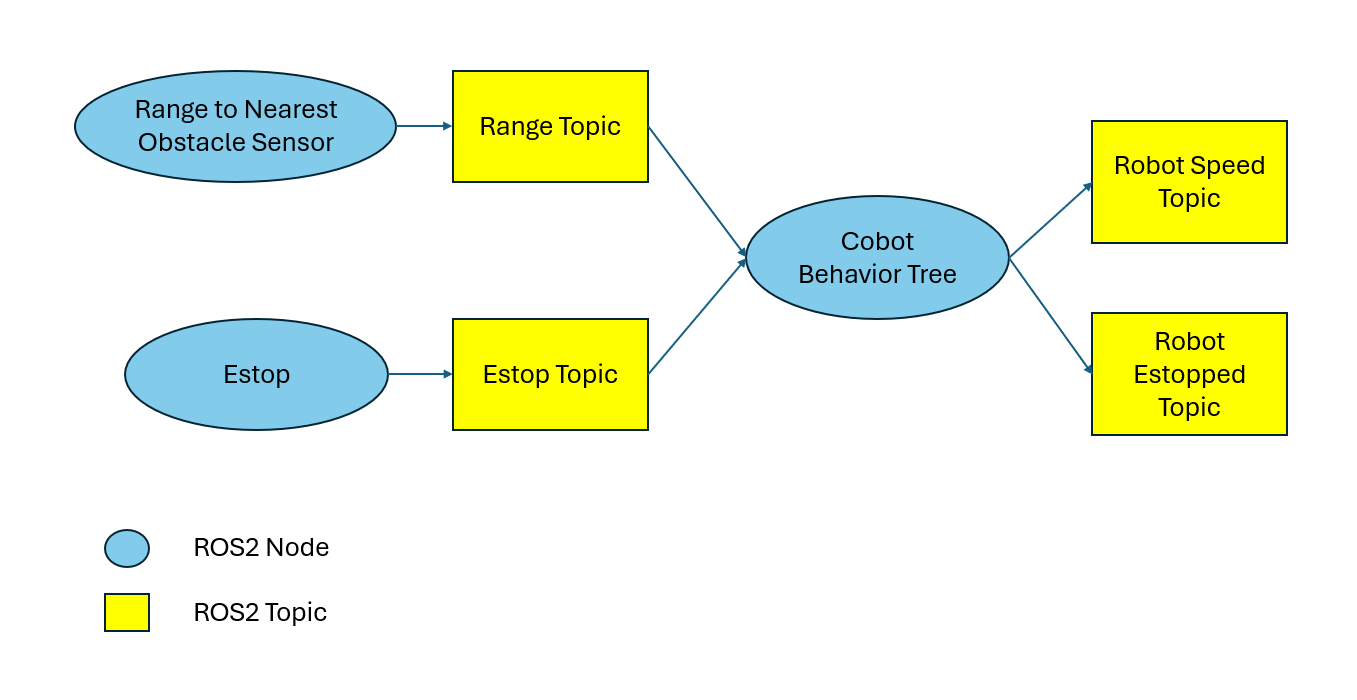
# Terminology

|  |  |
| --- | --- |
| Behavior Tree | A tree structure made up of action, condition, and decorator nodes. Similar to state machines, behavior trees allow modelling of complex behavior, however in many cases behavior trees are more intuitive and easier to understand. For example, in a state machine, global errors that can occur in any state are difficult to express because they potentially require a transition from every state to a failure state, which requires every node to have some knowledge of error handling, and clutters up the state machine. In behavior trees, global exceptions are handled in a much clearer fashion and can encapsulate this error knowledge into one piece of code. For details of behavior trees and the behavior tree library used in this design, see https://www.behaviortree.dev/ |
| Tick | A single execution of the logic and actions of the nodes of the behavior tree which returns either success or failure. Whether a node is ticked in tree execution and in what order is determined by the tree design. |
| Fallback Node | A fallback node is a condition node that will tick child nodes from left to right until one child succeeds, at which point it returns success. If all nodes failure, the fallback node returns failure. These are one of the standard condition nodes provided by behavior tree libraries. |
| Action Node | These are the nodes generally implemented by the user to represent complex logic and actions. They return either success or failure, which parent nodes then act on to determine the next steps in the tree. |
| Blackboard | Common data shared across all nodes, these can be read from or written to by any node. |
| ROS2 | Open source middleware commonly used for robotics software. |
| ROS2 topic | Publish/subscribe mechanism that allows ROS2 nodes to communicate with each other. |
| ROS2 node | A single separate unit of software used to complete a task in ROS2, run in its own process. Nodes can publish data, subscribe to data, and provide interfaces to the system and other nodes. |

# System Design

## Overall Cobot System Design



## Cobot Behavior Tree

A diagram of a flowchart

AI-generated content may be incorrect.

This behavior tree will execute estop. If there is an estop, the tree execution stops there. If there is no estop, the tree continues to process stop, slow, and full speed similarly.

# System Requirements

Provided in document: Robotics Software Engineer Take-Home Challenge

# Tools Used

ROS2 Jazzy https://docs.ros.org/en/jazzy/index.html

BehaviorTree.CPP version 4.6: <https://www.behaviortree.dev/>

C++17: Used for Behavior Tree implementation.

Python3: Used for Range sensor and estop modelling.

# ROS2 Nodes

Range Sensor: Publishes 16 bit integer proximity values to ROS2 topic “range”.

Estop: Publishes boolean value to ROS2 topic “estop” to simulate an asynchronous emergency stop signal.

Cobot: Takes input from “range” and “estop” topics and publishes a speed to ROS2 “speed”. If there is an estop signal, publishes it to the ROS2 “estopped” topic. “estopped” differs slightly from “estop” in that it indicates that the estop signal has been processed by the system.

# Logging

Uses ROS2 logging mechanism to produce Error, Warning, Information and Debug logs.

Publishes string representation of speed to ROS2 “speed” topic.

Publishes string representation of estop to ROS2 “estopped” topic.

# Future Work

Simulate full system startup: Currently nodes are started manually. A full system instantiation tool should be provided.

Automated Unit and Integration Tests: All tests are currently manual.

More complex input simulation: All data is randomly generated. A more realistic model, connected to a simulated robot would be better.

Create Docker container: This would improve portability.

Hysteresis: Avoid rapid toggling between speed states.

Visualization: ROS2 has multiple visualization tools, as does BehaviorTree.CPP. Evaluate and implement real time visualization of cobot behavior.