

COLUMBIA UNIVERSITY EEME E6911 FALL '25

TOPICS IN CONTROL : PROBABILISTIC ROBOTICS

BEHAVIOR TREES

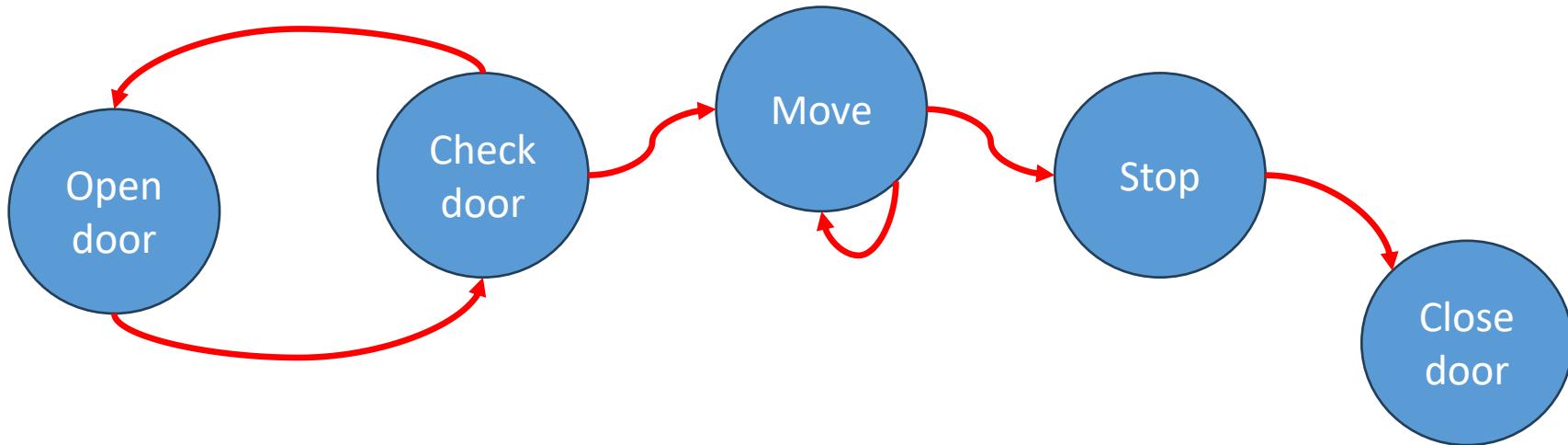
Instructor: Ilija Hadzic

Typical Mobile Robot Tasks

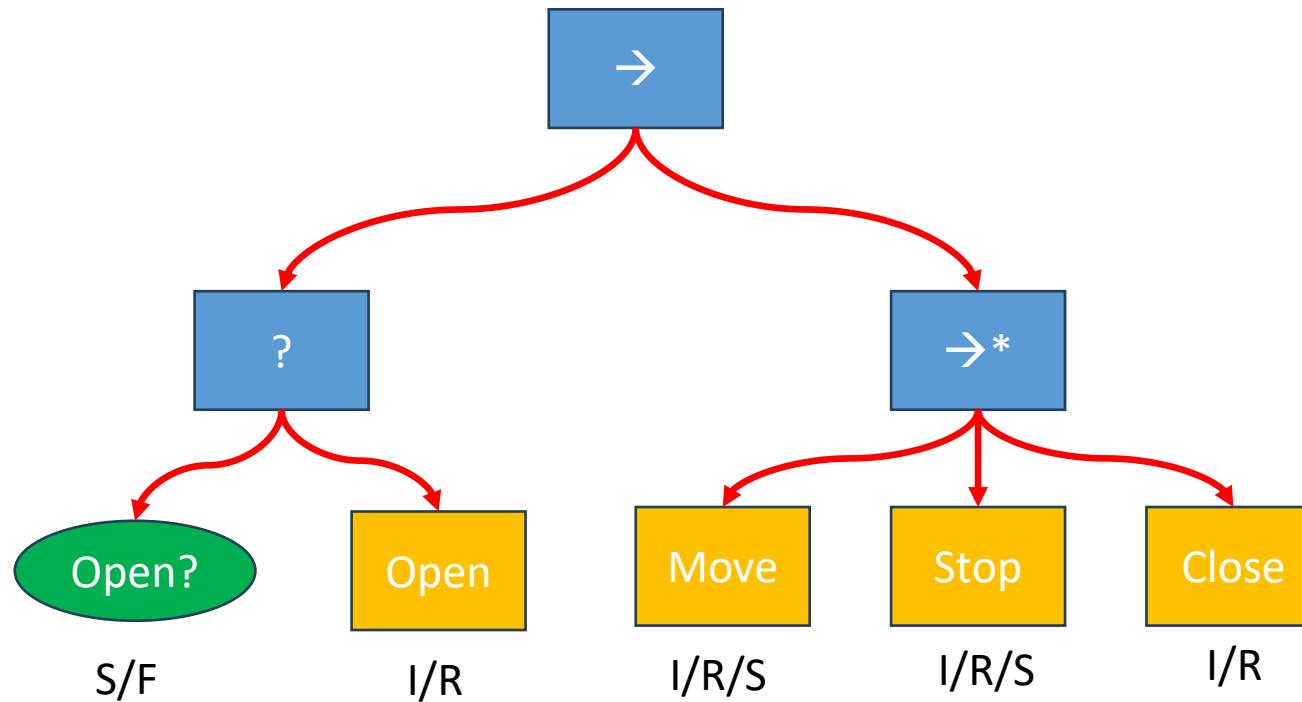
- Pursue a goal (while avoiding obstacles)
- Explore the area (e.g., autonomous mapping)
- Dock/undock (e.g., parking spot or charging station)
- Improve the localization
- Recovery behaviors (e.g., backout, spin, etc.)
- Handbrake
- Payload handling (e.g., load/unload the payload)
- Manipulation-related motion

Motivation

- Different tasks → different behavior
- Different behavior → different control algorithms
- Need a way to track and switch behaviors
- Recall the robot from Lab 2:



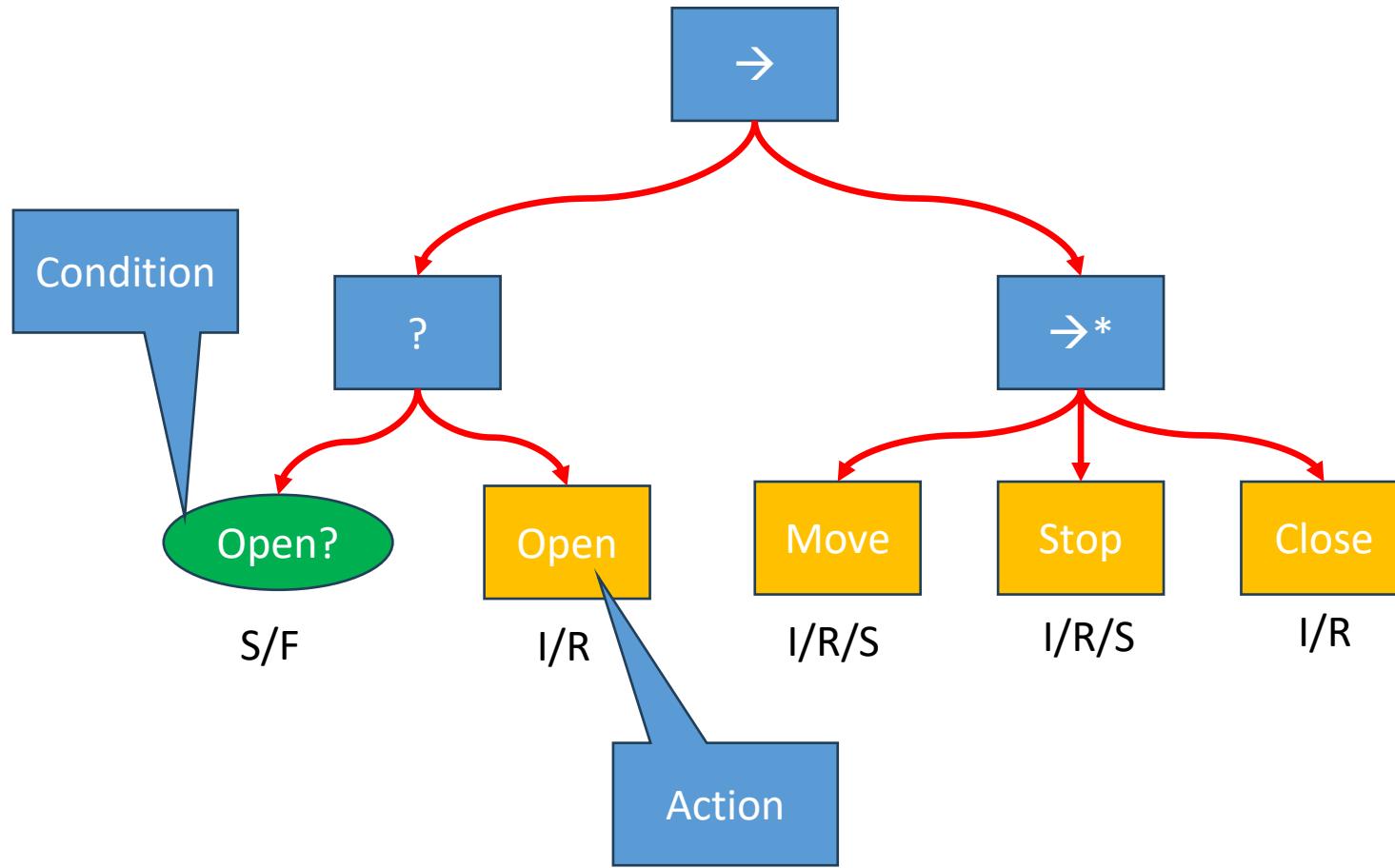
Behavior Tree: better abstraction



Basic Principles

- Main execution step: tick
- Tick propagates top-down
- Control flow nodes route the tick
- Condition node or action node is ticked
- Status propagates bottom-up
- Control flow nodes interpret and modify the status

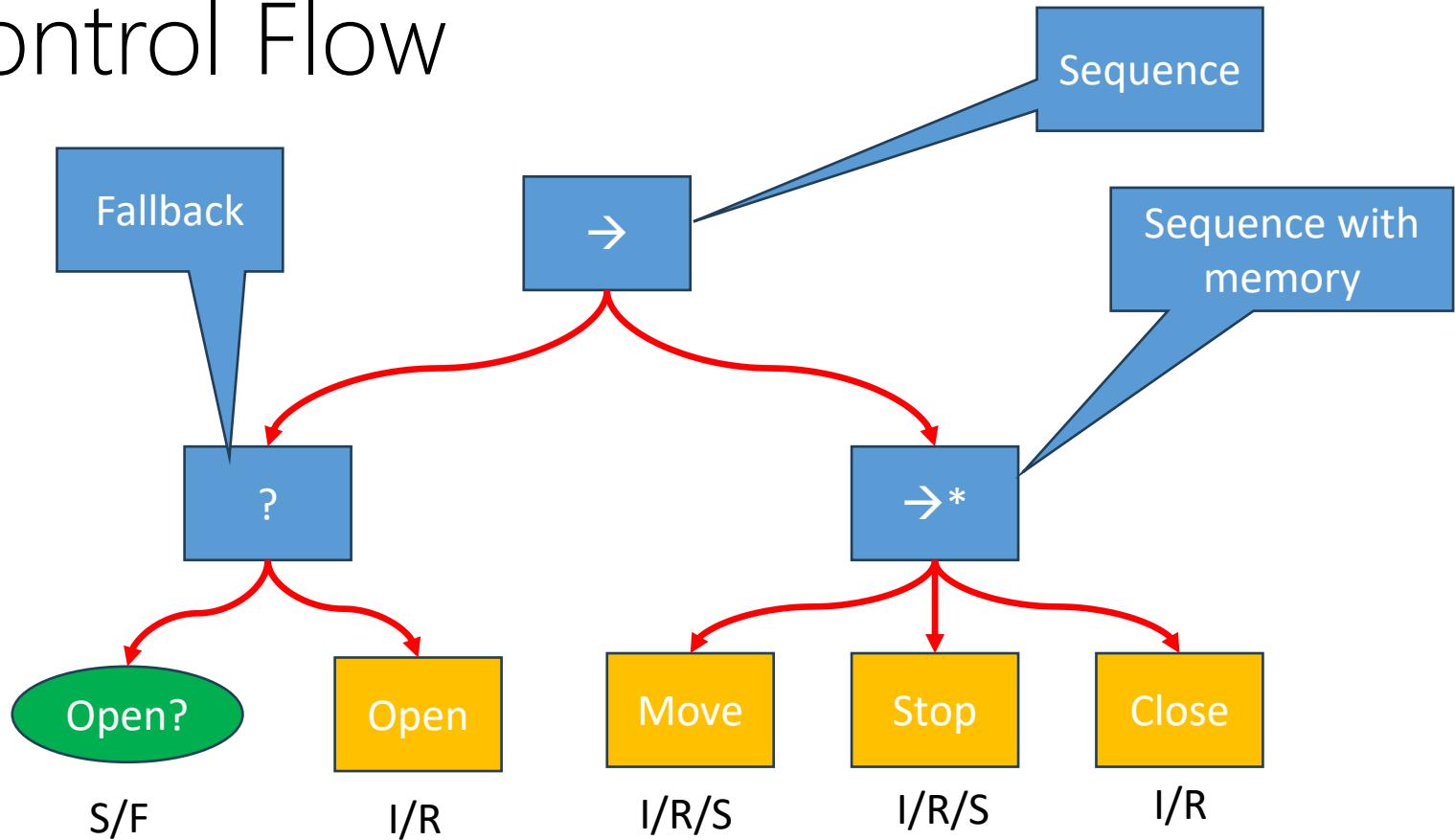
Action and Condition Nodes



Actions and Conditions

- Actions
- Stateful
 - Running: busy executing
 - Idle: not ticked yet
 - Success/Failure: done
- May return subset:
 - e.g., never fails
- Represent work:
 - e.g., pursuing a goal
- Conditions
- Stateless
 - Success/Failure
 - Returns immediately
- Never returns running
- Never returns idle
- Represent state checks
 - e.g., is door open?

Control Flow



Control Flow

Sequence:

```
for i 1 to N do
    childStatus <-Tick(child(i))
    if childStatus = Running then
        return Running
    else if childStatus = Failure then
        return Failure
    return Success
```

- All nodes must succeed

- Memory:

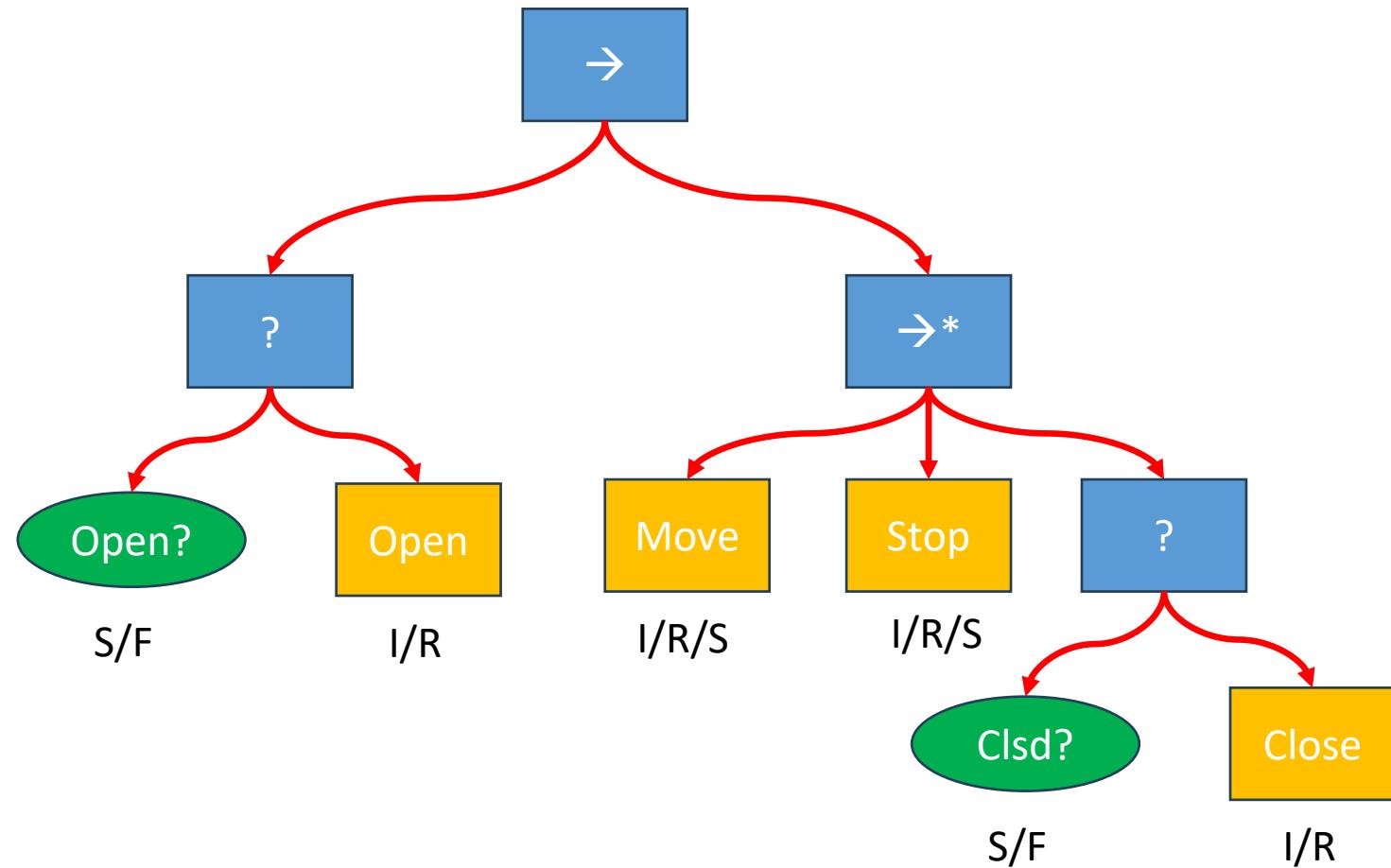
- Remember status
- No re-tick until the whole subtree completes

Fallback:

```
for i 1 to N do
    childStatus <-Tick(child(i))
    if childStatus = Running then
        return Running
    else if childStatus = Success then
        return Success
    return Failure
```

- Any node must succeed

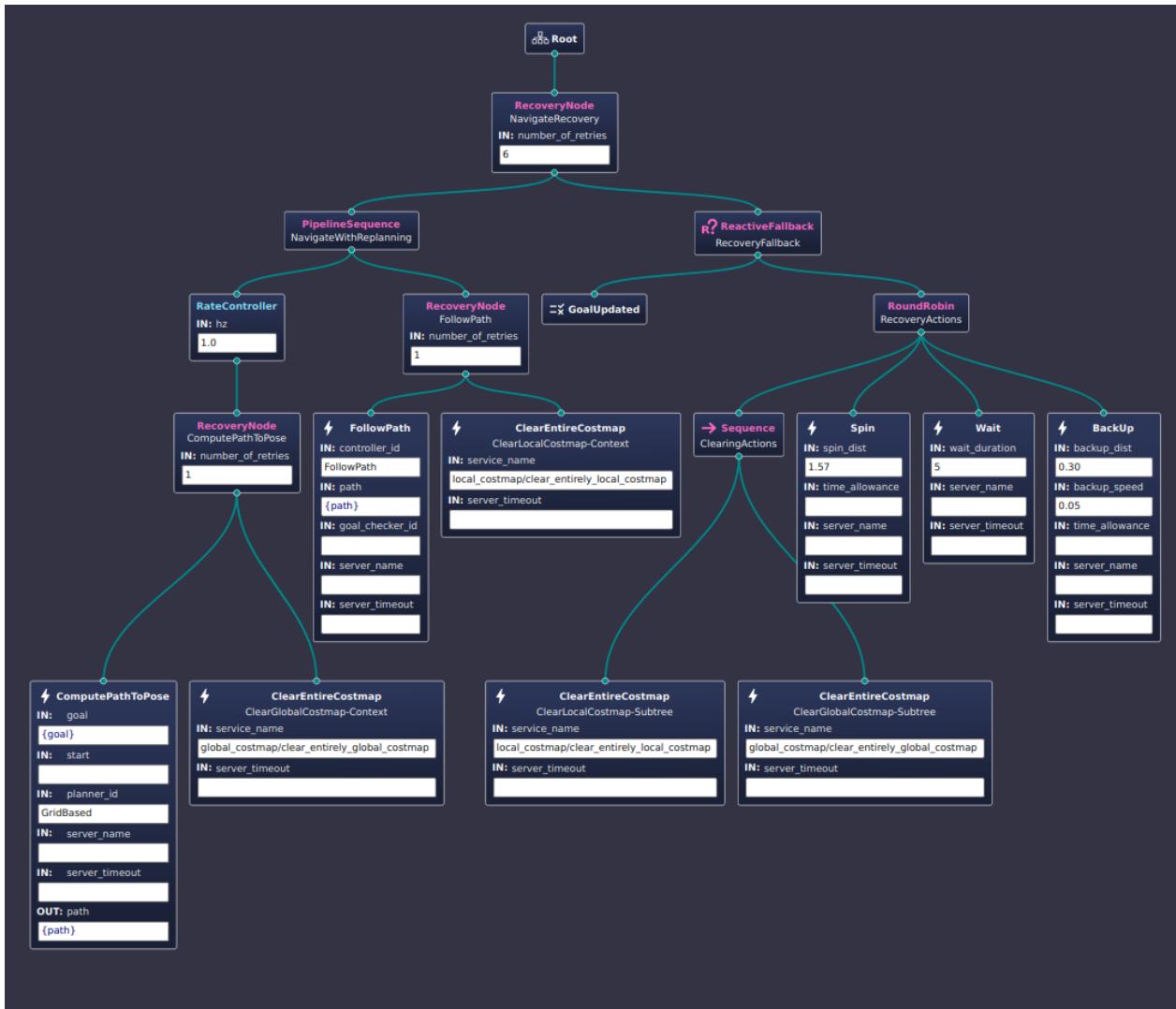
Adding New Behavior



In Practice

- BehaviorTree.CPP
 - C++ framework for implementing behavior trees
 - Specifies tree a XML
 - Graphical editors available
 - Implement actions and conditions using API (.so)
 - Spawn the runtime
- Nav2 (ROS)
 - Based on BehaviorTree.CPP
 - Implements Nav2-specific actions
- Both packages extend control flow structures
 - Reactive Sequence, Round-Robin, etc.

Example (from Nav2 package)



Example extensions

- Recovery Node:
 - Try first child
 - If it fails run second child (recovery)
 - After recovery, try first child again
 - Limit the number of retries
- Round Robin:
 - Like sequence but goes back to start
- Reactive Fallback:
 - Like fallback, but can be preempted
- Pipeline Sequence:
 - Like sequence but does not wait for completion

References

- Book:
 - Michele Colledanchise and Petter Ögren, Behavior Trees in Robotics and AI, 2017
 - <https://arxiv.org/abs/1709.00084>
- BehaviorTree.CPP:
 - <https://www.behaviortree.dev/>
- Nav2 documentation:
 - https://docs.nav2.org/behavior_trees/index.html

