Robotics Dojo

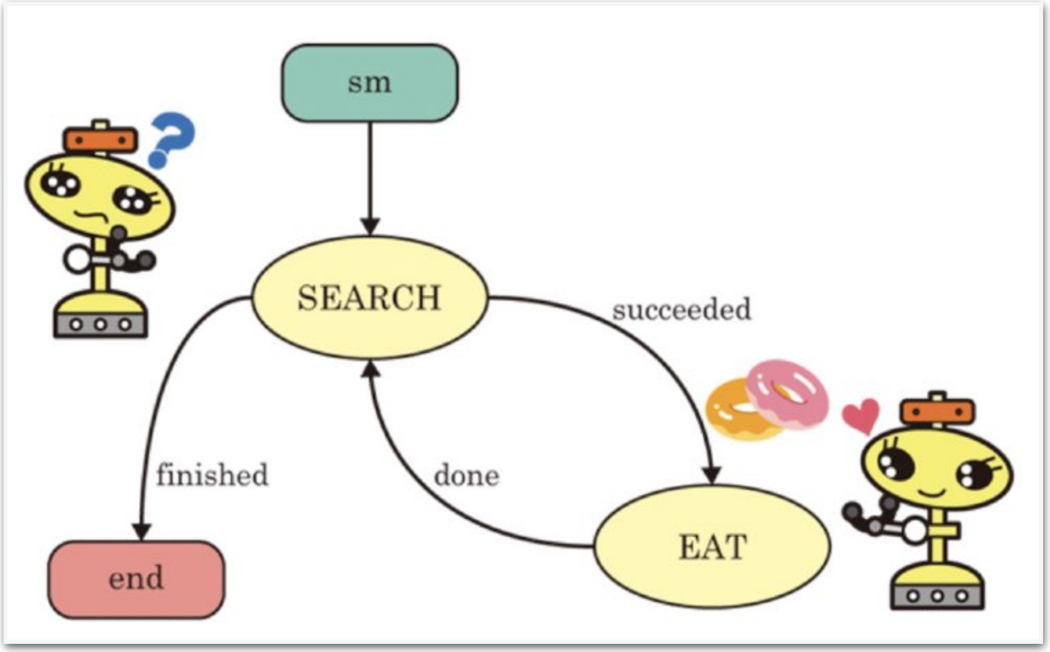
State machine basics for robot control

Dr. Shohei Aoki
Jomo Kenyatta University of Agriculture and Technology
Robotics Dojo

Table of contents

- State machine for robot control
- py_tree basics
- py_tree_ros basics
- Integration with Nav2

What is the state machine?



Example.
- Robot searches and eats sweets

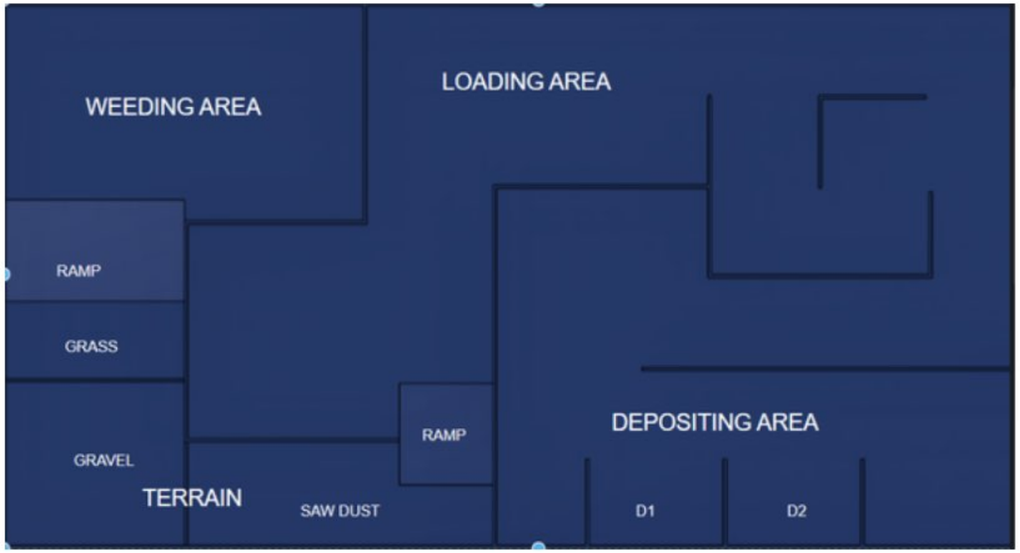
Two states:
SEARCH and EAT

1. Robot searches sweets
2. Once the robot finds the sweets, he/she will eat it

A state machine describes the **sequence of events**, and the **conditions of state transition**

Why will we need the state machine?

Dojo competition 2025



Robot moves to the weeding area
→ It weeds
→ When it completes weeding, it will move to the loading area
→ It will wait until the loading is done
→ When it completes loading, it will move to the depositing area
→...

The aim of Dojo competition 2025 is the **integration** of the **SLAM&Navigation** (2024) and **robot task** (2022,2023)

Fully autonomous robot!!

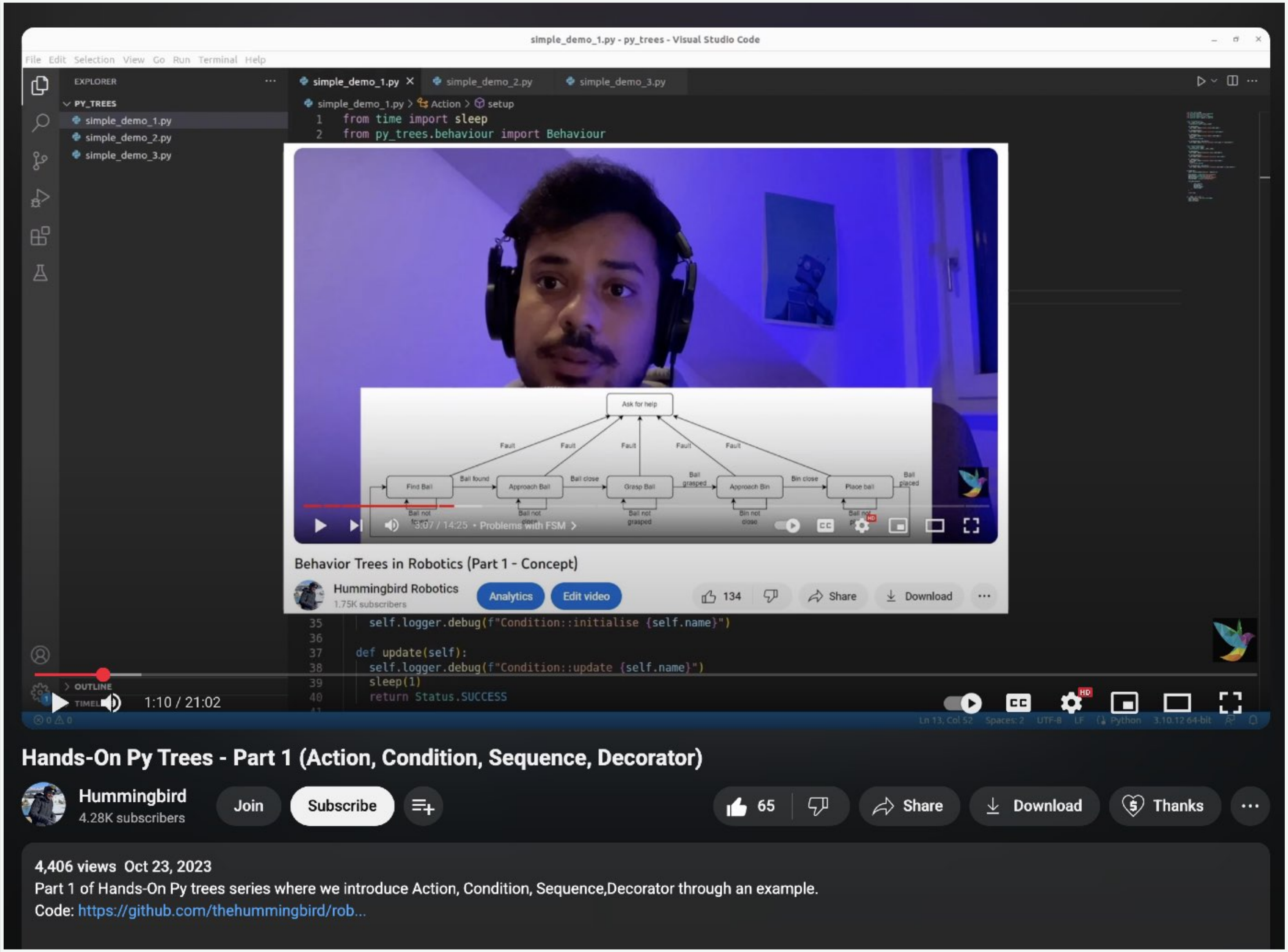
py_trees Python package

- BehaviorTree.cpp is commonly used for the state machine with ROS
  - It is implemented with C++
- py_trees is the Python implementation of the BehaviourTree.cpp

https://github.com/splintered-reality/py_trees
https://py-trees.readthedocs.io/en/devel/index.html

Py Trees tutorial

https://www.youtube.com/watch?v=vqbV7mysL84



```
from time import sleep
from py_trees.behaviour import Behaviour
```

Behavior Trees in Robotics (Part 1 - Concept)

Hummingbird Robotics
1.75K subscribers

```
    self.logger.debug(f"Condition::initialise {self.name}")

  def update(self):
    self.logger.debug(f"Condition::update {self.name}")
    sleep(1)
    return Status.SUCCESS
```

Hands-On Py Trees - Part 1 (Action, Condition, Sequence, Decorator)

Hummingbird
4.28K subscribers

4,406 views Oct 23, 2023
Part 1 of Hands-On Py trees series where we introduce Action, Condition, Sequence,Decorator through an example.
Code: https://github.com/thehummingbird/rob...

PyTrees Example

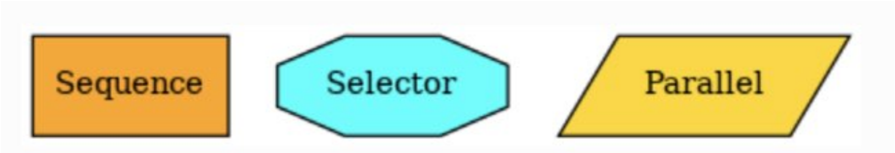
https://github.com/thehummingbird/robotics_demos/tree/main/py_trees

- simple_demo_1.py
  - Use `tick_one()` once
- simple_demo_2.py
  - Use `tick_one()` in the loop
- simple_demo_3.py
  - Use of `inverter`

Type of composites and decorators

- Composites
  - Sequence
  - Selector
  - Parallel
- Decorators
  - EternalGuard
  - SuccessIsRunning

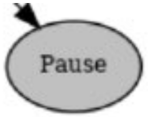
Composites



Decorators

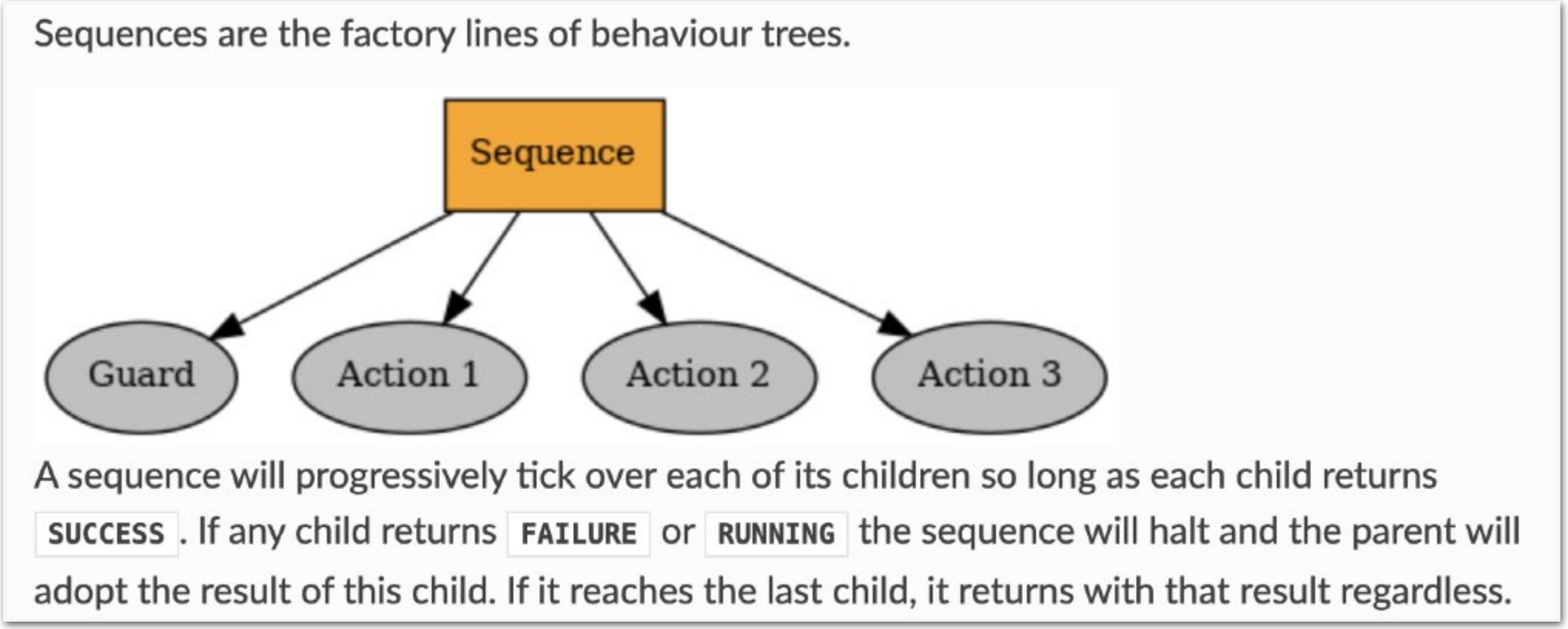
Battery Low?

Others



Sequence

https://py-trees.readthedocs.io/en/devel/composites.html

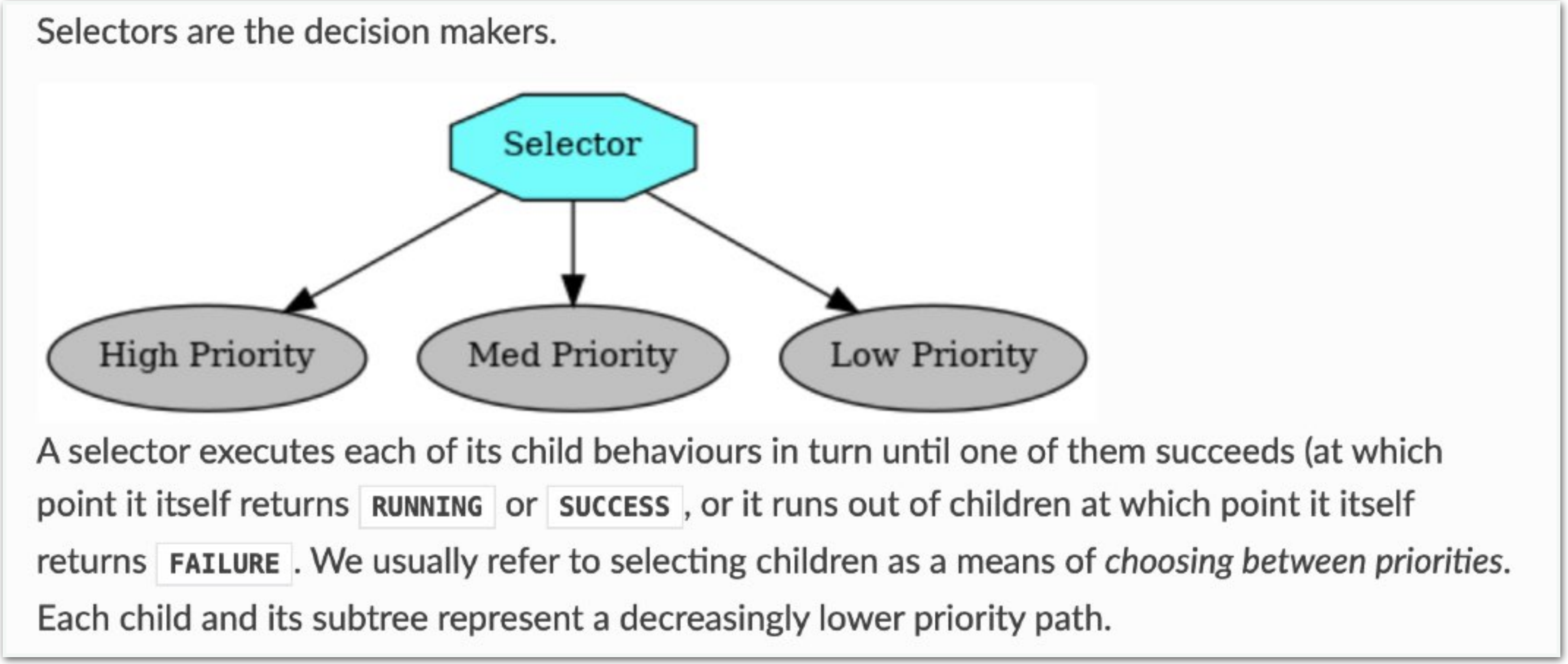


Sequences are the factory lines of behaviour trees.

A sequence will progressively tick over each of its children so long as each child returns `SUCCESS`. If any child returns `FAILURE` or `RUNNING` the sequence will halt and the parent will adopt the result of this child. If it reaches the last child, it returns with that result regardless.

Selector

https://py-trees.readthedocs.io/en/devel/composites.html

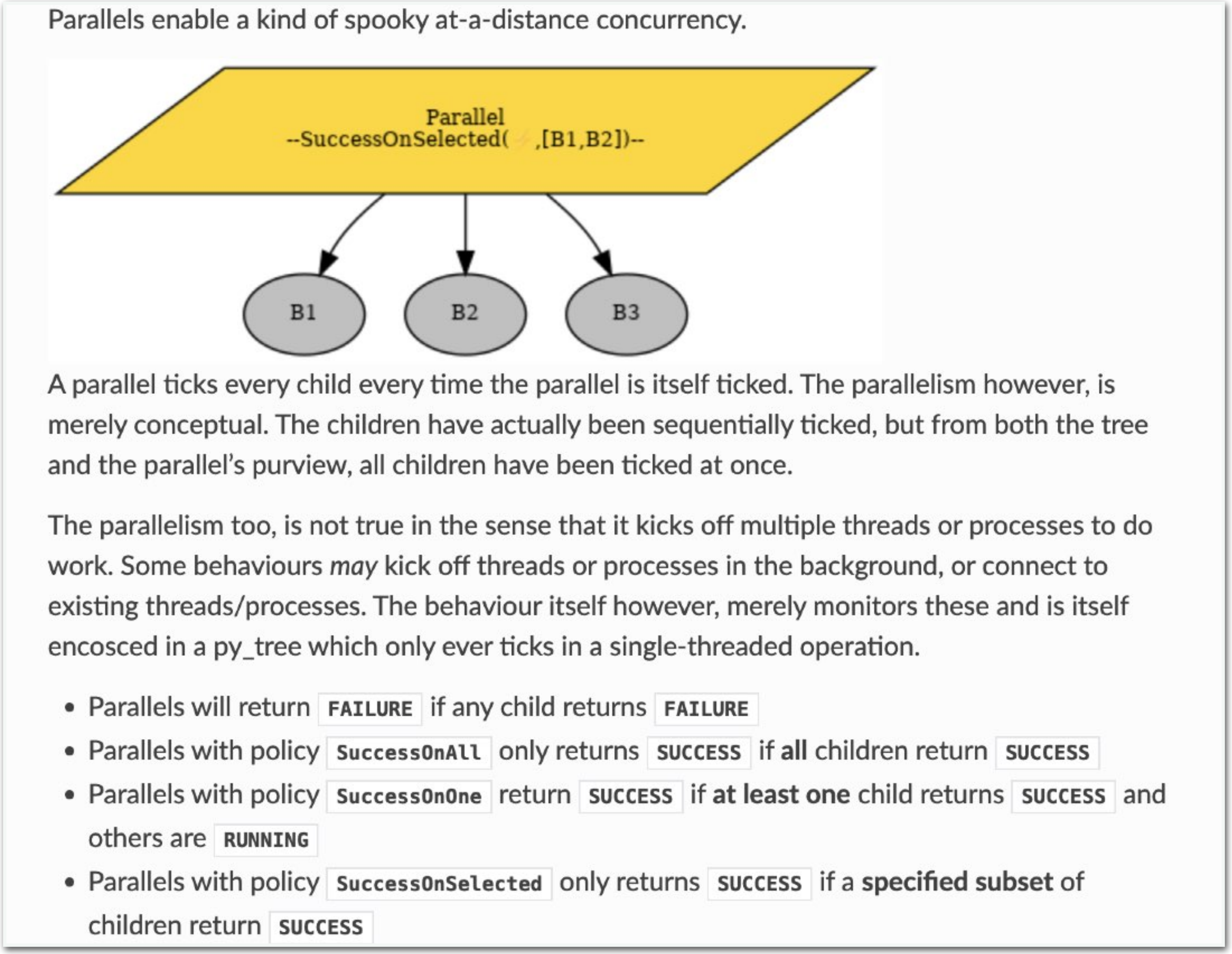


Selectors are the decision makers.

A selector executes each of its child behaviours in turn until one of them succeeds (at which point it itself returns `RUNNING` or `SUCCESS`, or it runs out of children at which point it itself returns `FAILURE`. We usually refer to selecting children as a means of *choosing between priorities*. Each child and its subtree represent a decreasingly lower priority path.

Parallel

https://py-trees.readthedocs.io/en/devel/composites.html



Parallels enable a kind of spooky at-a-distance concurrency.

A parallel ticks every child every time the parallel is itself ticked. The parallelism however, is merely conceptual. The children have actually been sequentially ticked, but from both the tree and the parallel's purview, all children have been ticked at once.

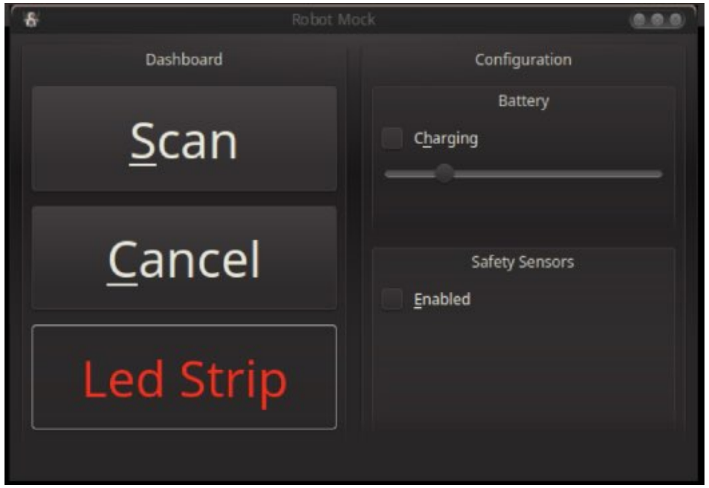
The parallelism too, is not true in the sense that it kicks off multiple threads or processes to do work. Some behaviours *may* kick off threads or processes in the background, or connect to existing threads/processes. The behaviour itself however, merely monitors these and is itself encosced in a py_tree which only ever ticks in a single-threaded operation.

- Parallels will return `FAILURE` if any child returns `FAILURE`
- Parallels with policy `SuccessOnAll` only returns `SUCCESS` if **all** children return `SUCCESS`
- Parallels with policy `SuccessOnOne` return `SUCCESS` if **at least one** child returns `SUCCESS` and others are `RUNNING`
- Parallels with policy `SuccessOnSelected` only returns `SUCCESS` if a **specified subset** of children return `SUCCESS`

`py_tree_ros`

ROS2 implementation of PyTrees

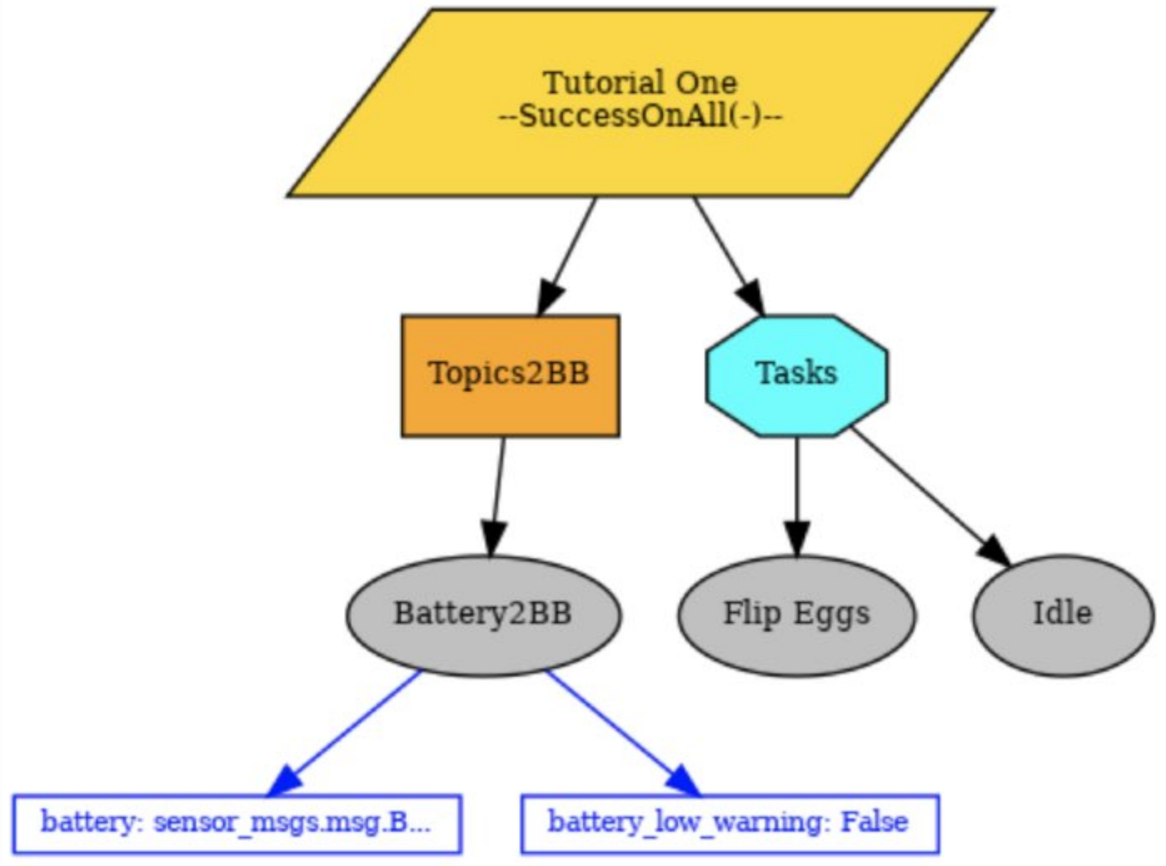
- 1. Data gathering
- 2. Battery check
- 3. Action clients



Mock robot console

https://py-trees-ros-tutorials.readthedocs.io/en/devel/tutorials.html

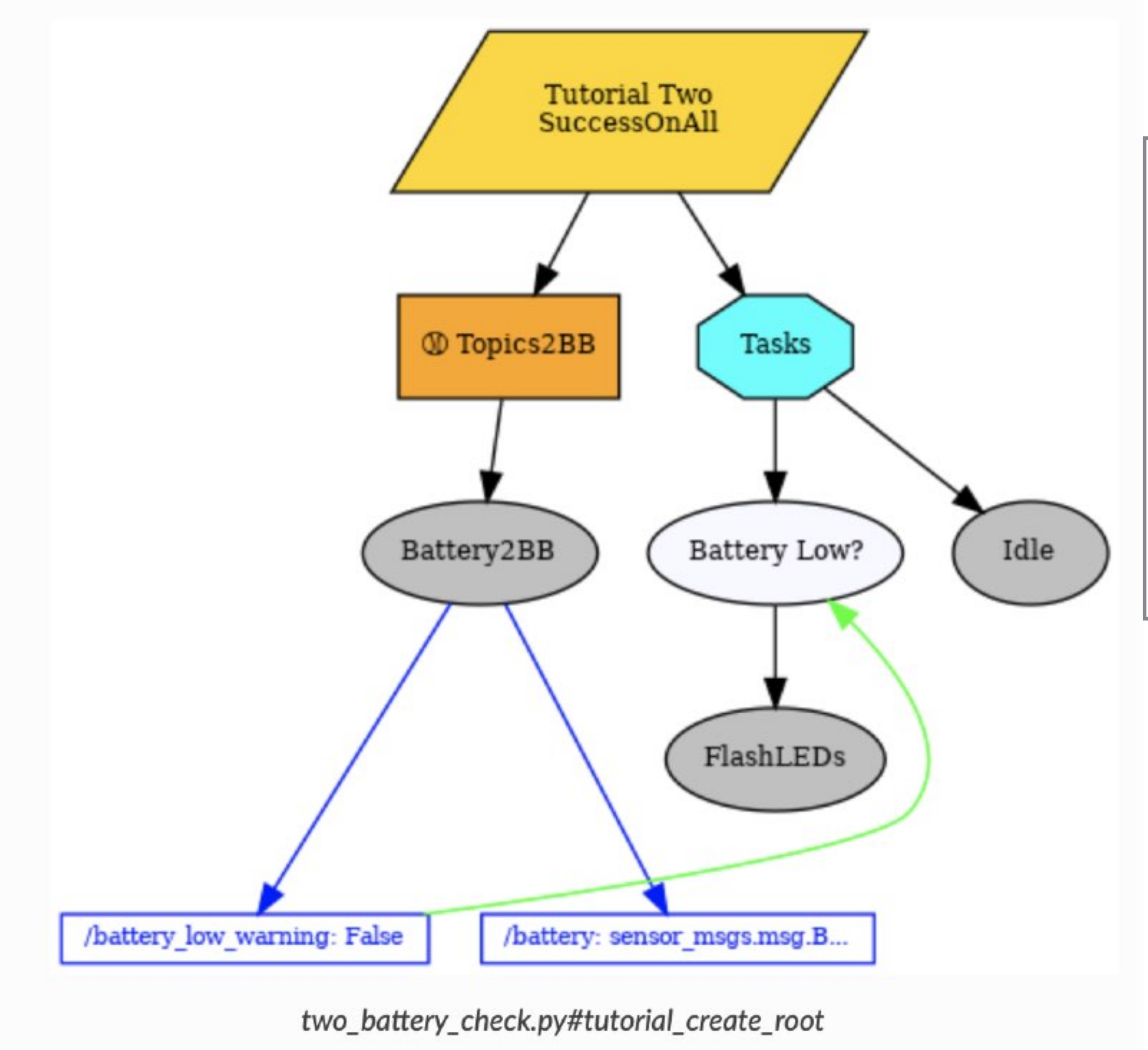
1. Data gathering



one_data_gathering.py#tutorial_create_root

Black Board (BB) is a global memory that can be referred from everywhere

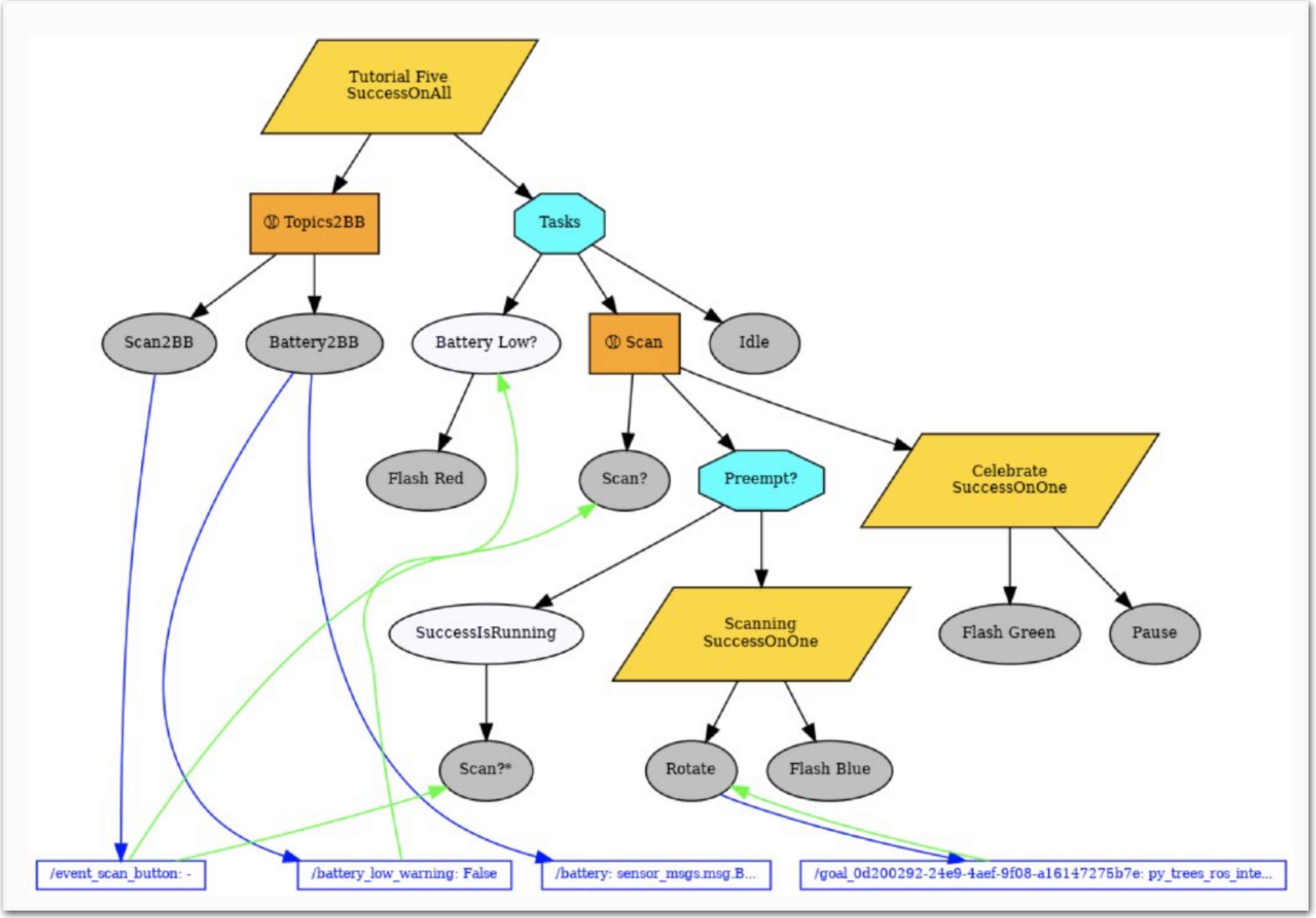
2. Battery check



two_battery_check.py#tutorial_create_root

The battery voltage is always monitored and an alert will be triggered when the remaining battery is less than 30%.

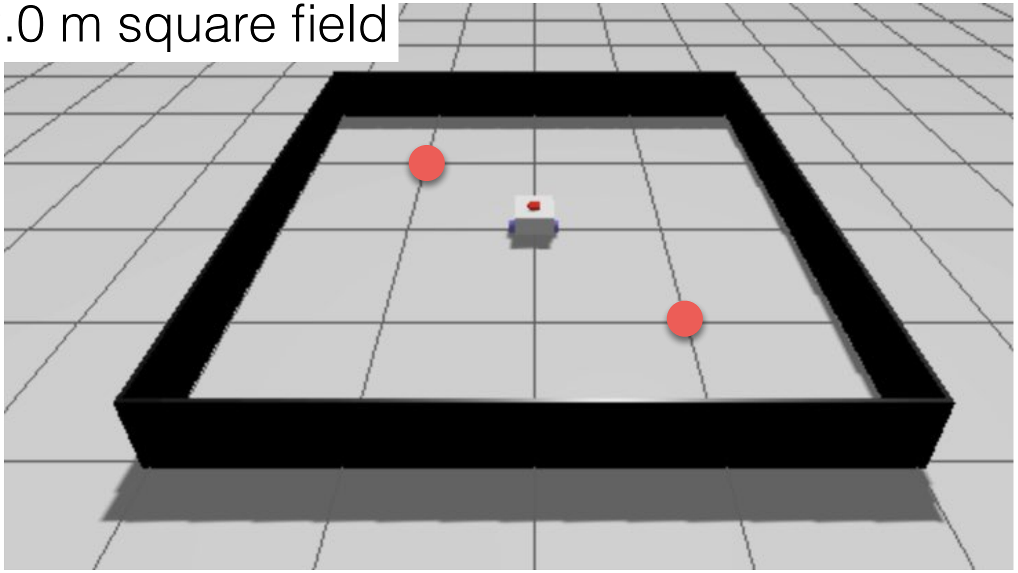
3. Action Clients



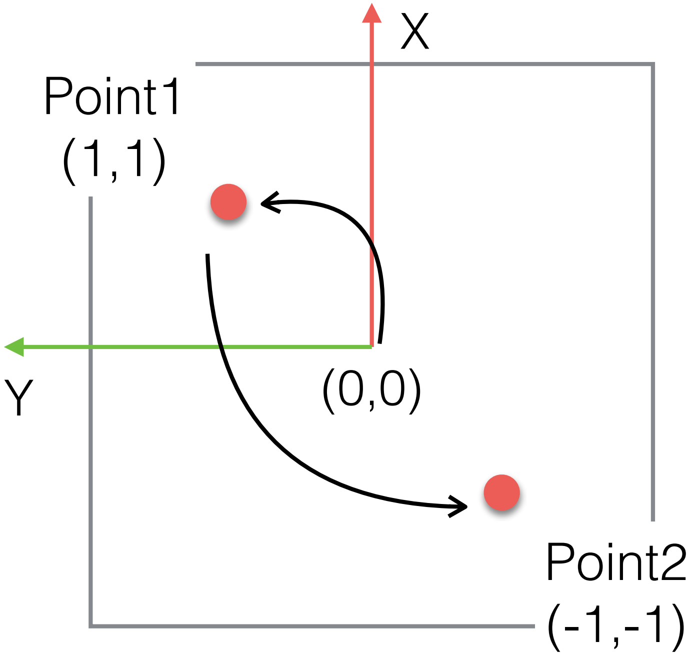
This is a fairly complex (so realistic) state transition.

- If you understand this state transition, you are ready to program your own robot!

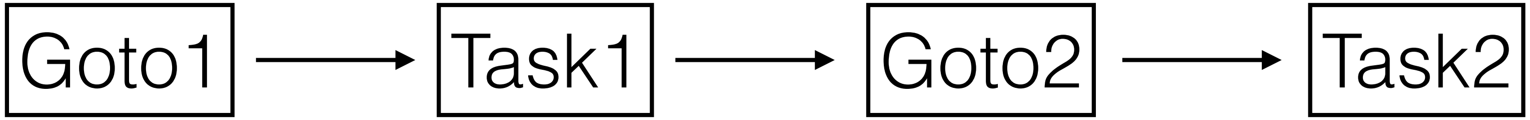
Demo program
State machine with navigation



2.0 m square field



Following **sequence**



Goto1 → Task1 → Goto2 → Task2

```
print('hello')
```

```
print('hi')
```

Movement is conducted by Nav2 package
State machine is defined by PyTrees package

How to run demo program

https://github.com/shohei/gazebo_ignition_fortress

- Run simulator (Ignition) and RViz2

```
$ ros2 launch ppp_bot launch_sim.launch.py
world_name:=simple_room
```

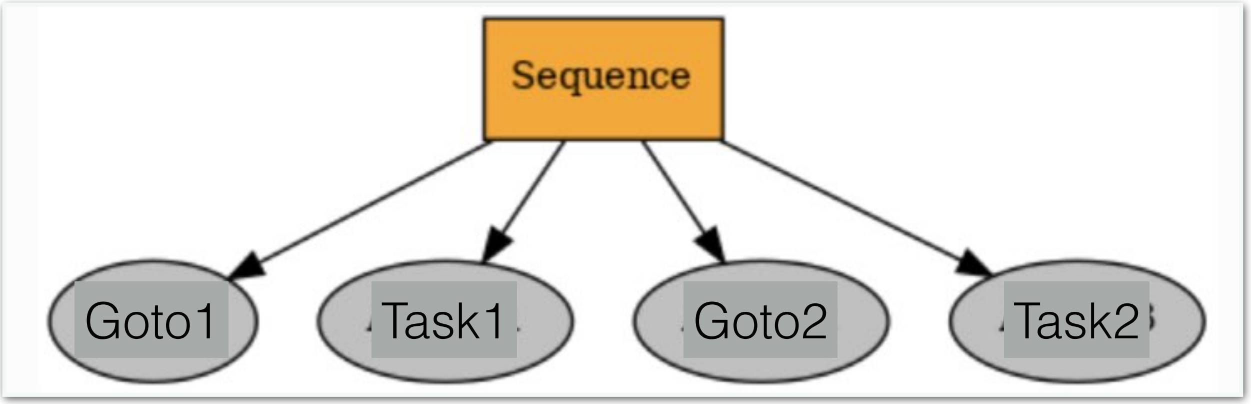
- Run navigation

```
$ ros2 launch ppp_bot navigation.launch.py
use_sim_time:=true
```

- Run demo script

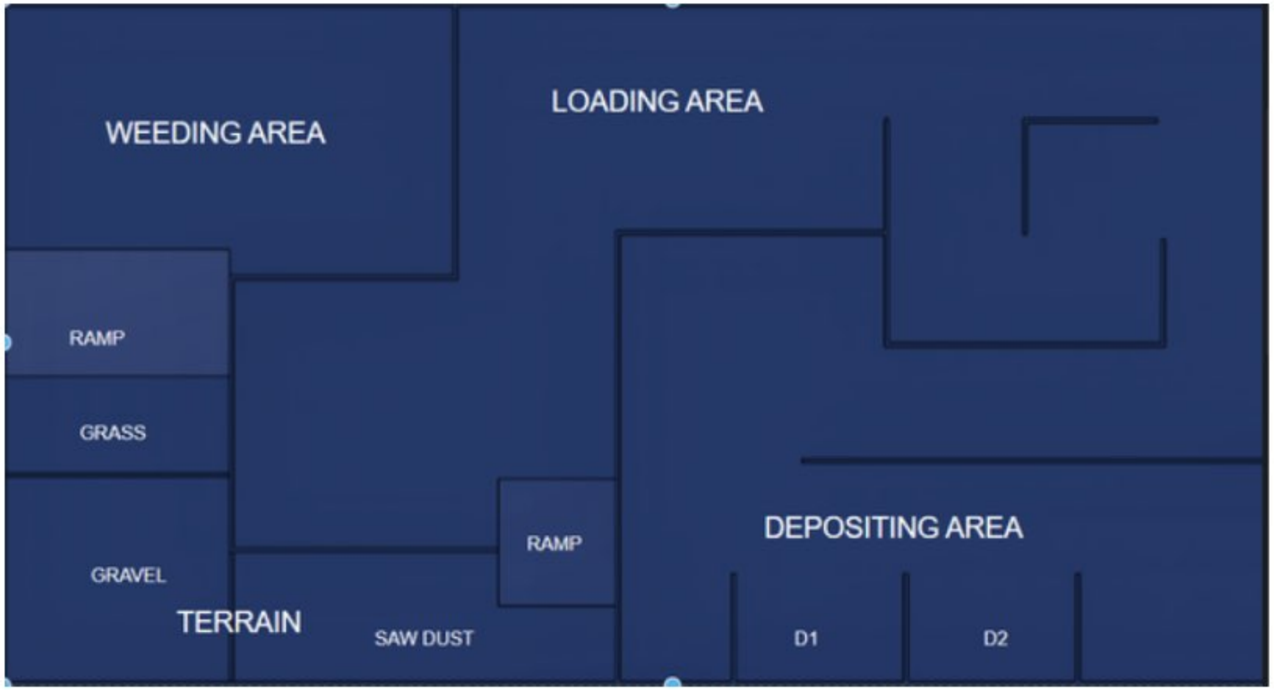
```
$ python test_folder/app.py
```

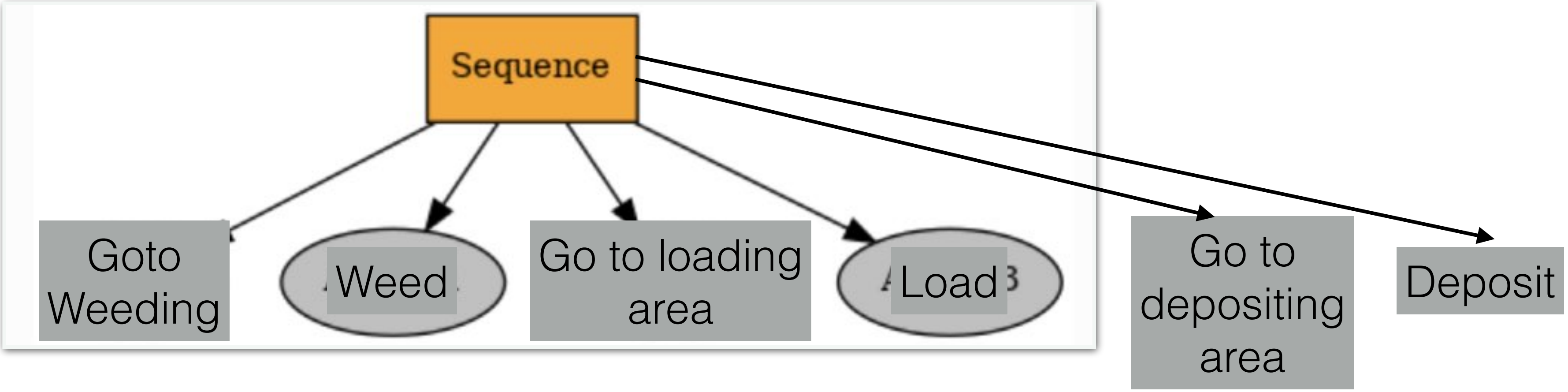
State machine definition



Simple sequence of 4 actions

How to customize the state machine

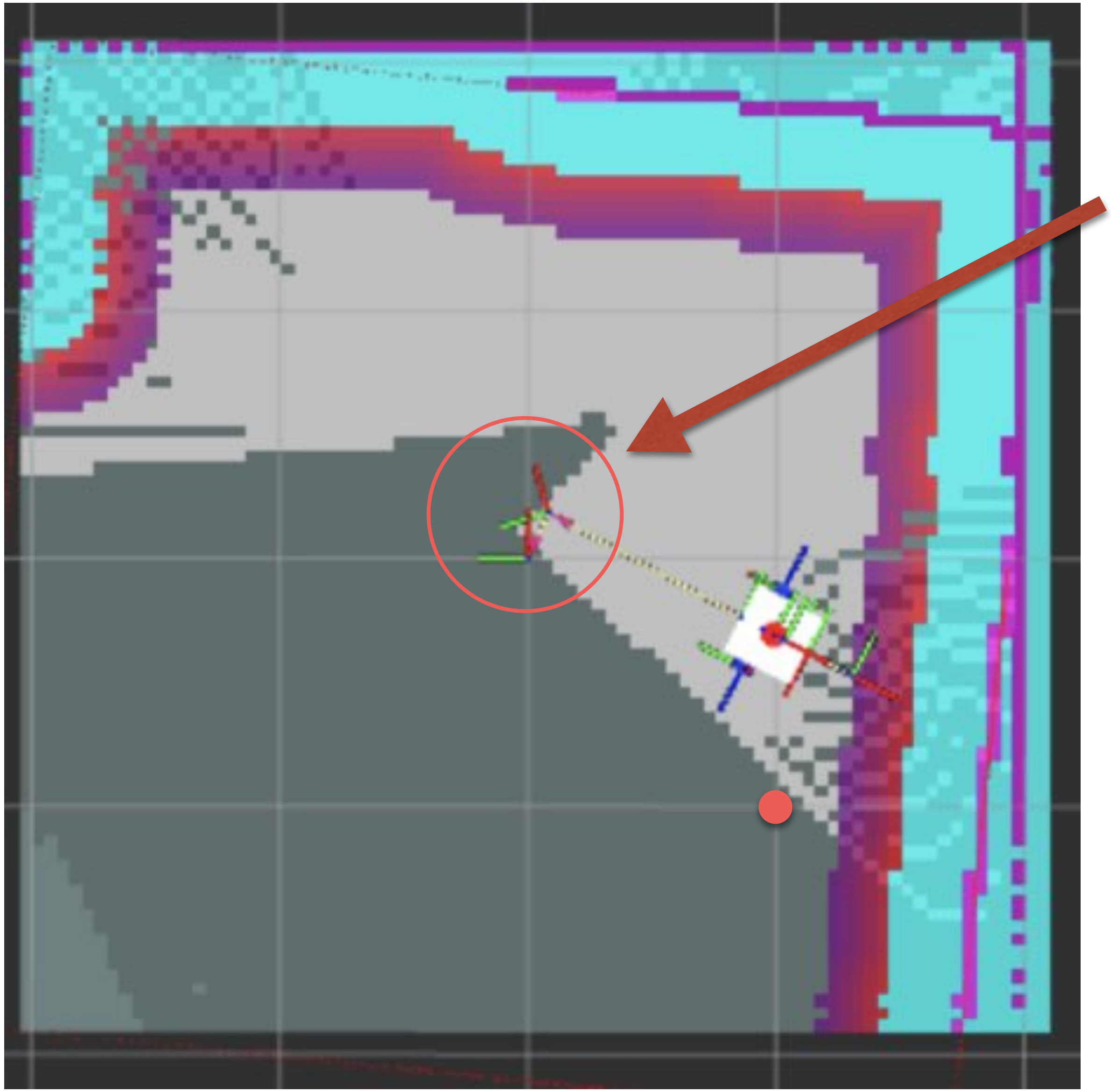




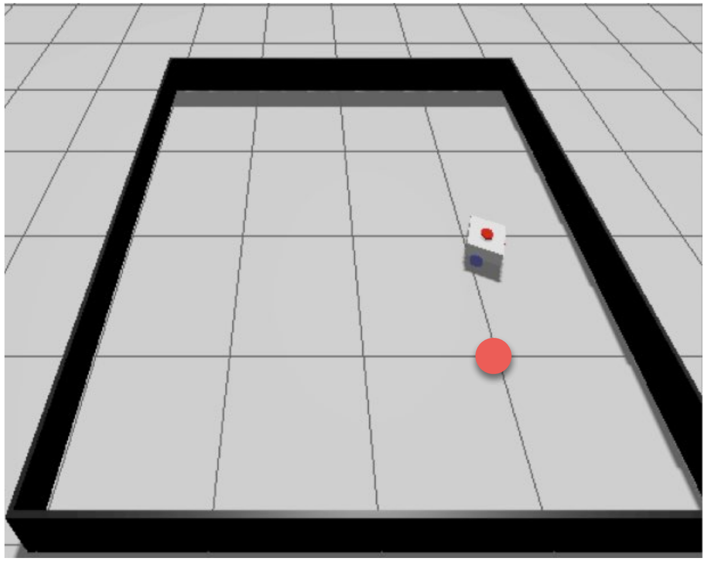
Tips

- You can adjust the **threshold** to judge if the robot has reached the goal

What needs to be improved: Localization error



The origin recognized by the Robot is offset from the actual origin



Error is observed
→Robot did not reach (-1,-1)