

FZI Forschungszentrum Informatik

ros_bt_py

Behavior Tree Framework for
Robot Missions and Capabilities

David Oberacker
Oberacker@fzi.de



Motivation

Creating High Level Missions for Your Robot

Various **State Machine** options are available

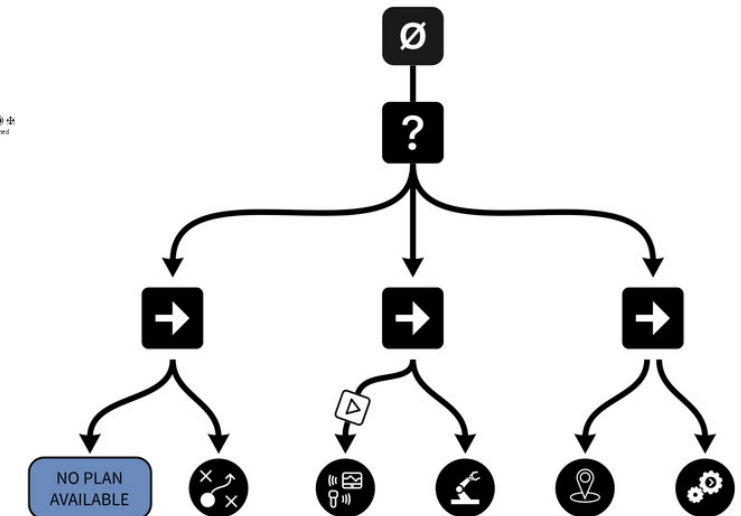
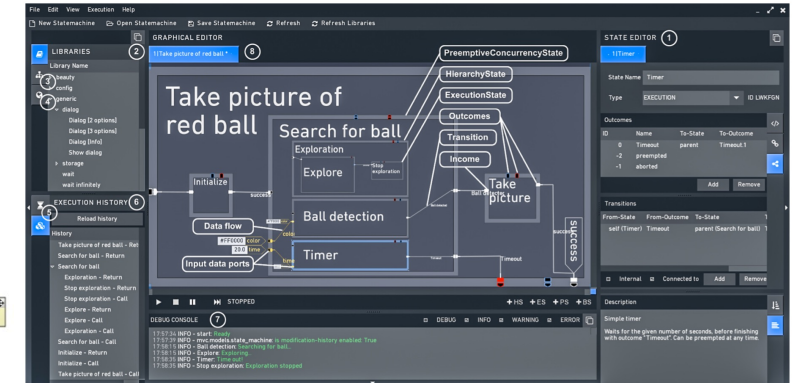
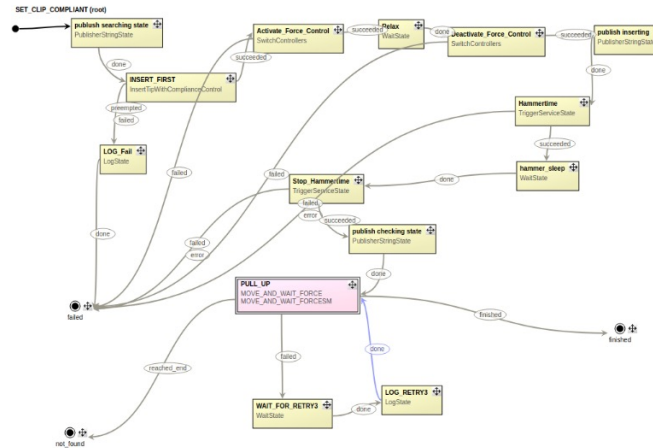
- i.E SMACH, FlexBE, RAFCON..
- Well tested, understood & mature tooling available

BUT:

- Tedious to use in larger Projects
- Difficult to change and modify on the fly

Behavior Trees as the new standard

- Easy to create and understand
- Task oriented instead of states
- Reactive and easy to modify & integrate



Motivation

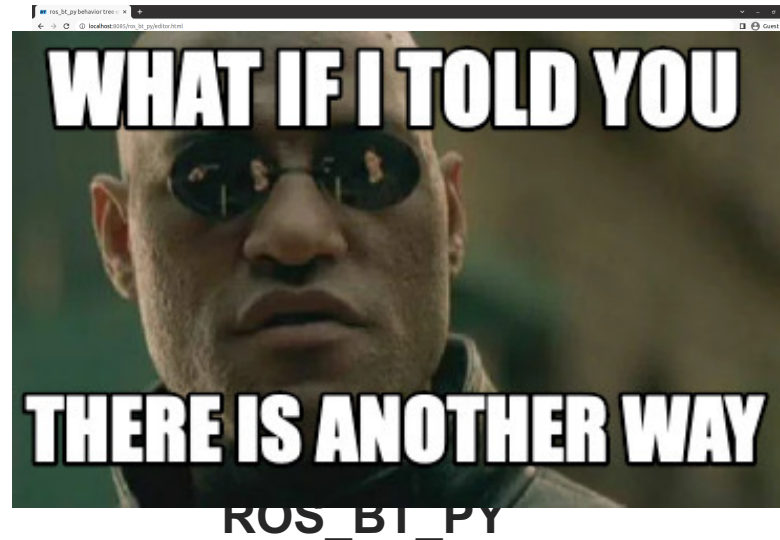
What we need from Behavior Trees

Current Behavior Tree Frameworks are

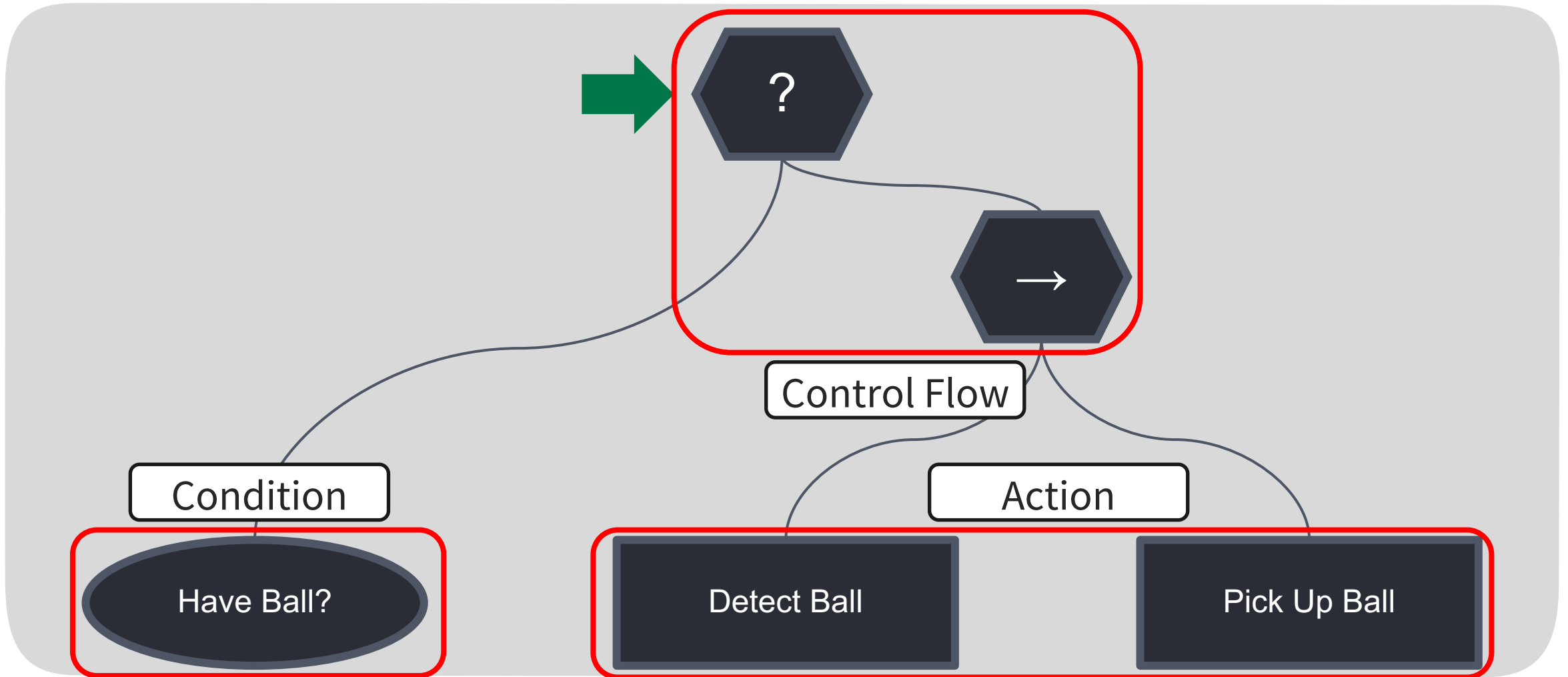
- less mature and provide fewer features than HSM
- often focussed in their application or project
- still difficult to use
- ROS Integration is often an afterthought

What do we want from a Behavior Tree Framework?

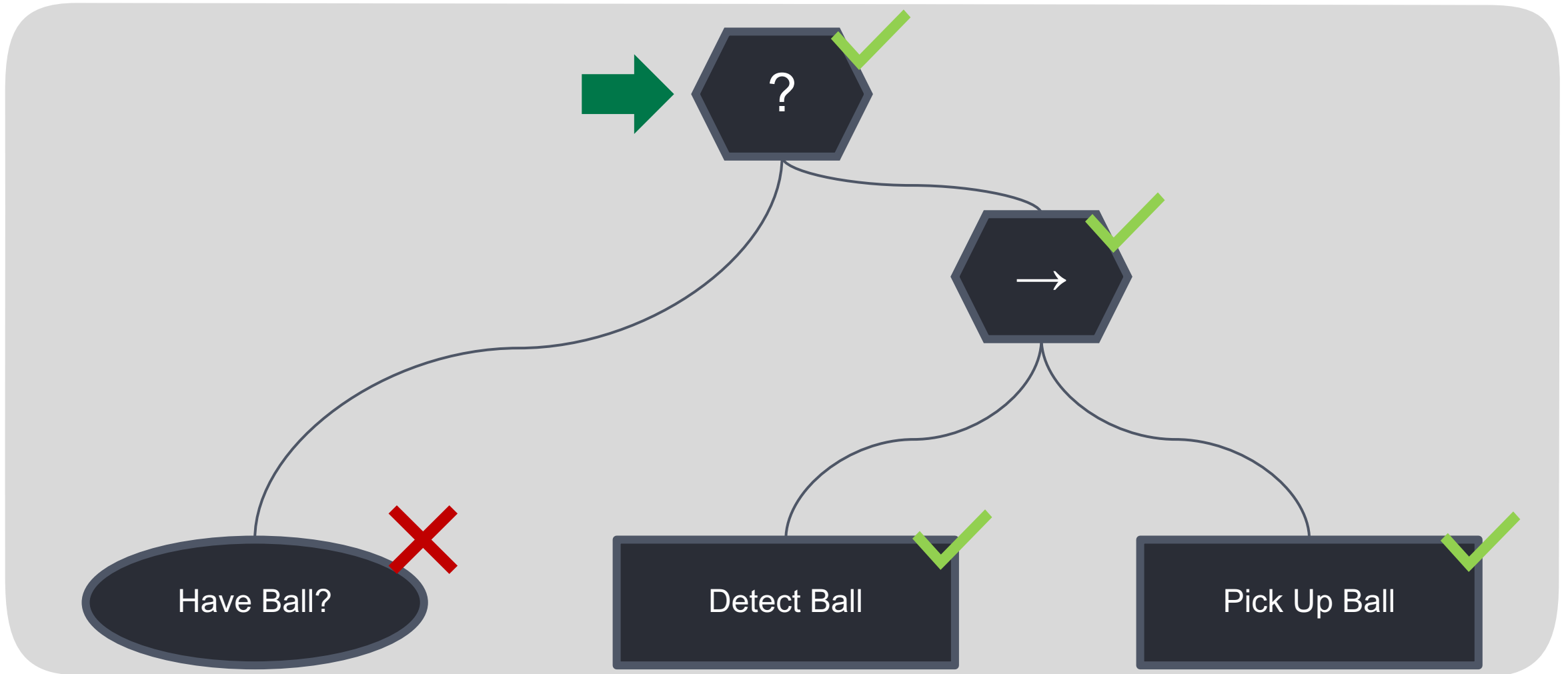
- Easy to use offline and online
- Modular, Flexible and Multi-Robot Enabled
- Rapid mission building for various robots and applications
- ROS Integration at every Step



Behavior Trees - Terminology



Behavior Trees - Execution & Ticking



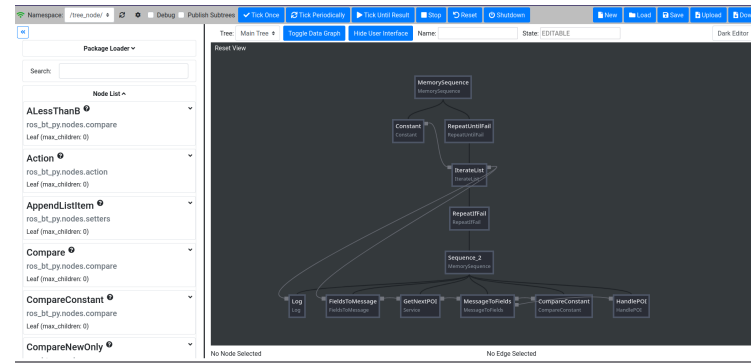
ros_bt_py - Library

```
topic_name : str],
inputs={},
outputs={'message': OptionRef('topic_type')},
max_children=0))
class TopicSubscriber(Leaf):
    """Subscribe to the specified topic and output the received messages

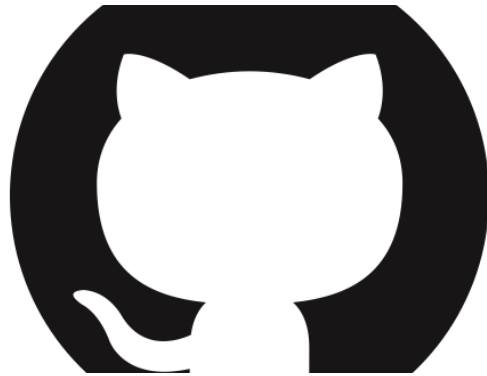
    This node will return RUNNING until a message is received on the topic.
    When a message is received, it outputs the message and returns SUCCEEDED.
    The message is then cleared for the next run.

    This node never returns FAILED.
    """
    def do_setup(self):
```

Python-based Implementation



Web-based Editor



Open-Source *

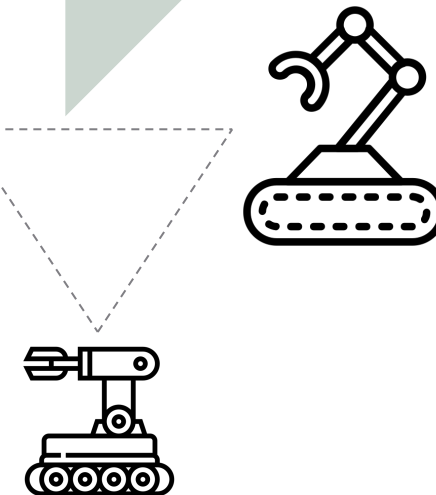
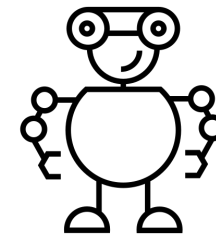
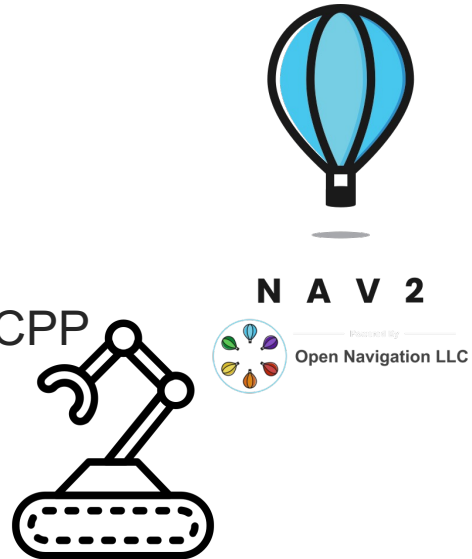
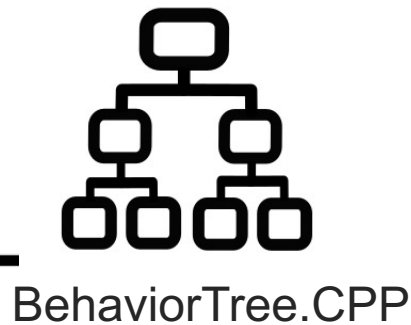
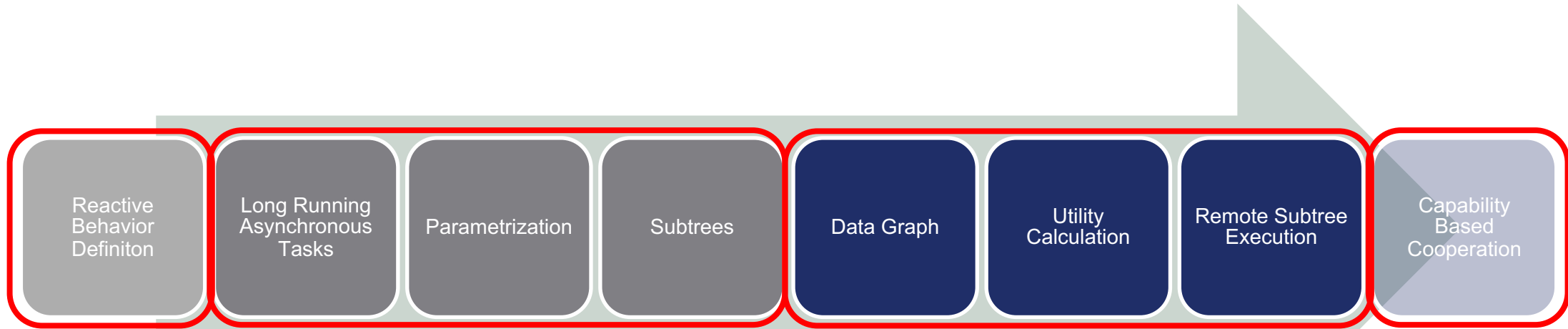


Native ROS 1/2 Integration

* https://github.com/fzi-forschungszentrum-informatik/ros_bt_py

Features

Behavior Tree-Perspective



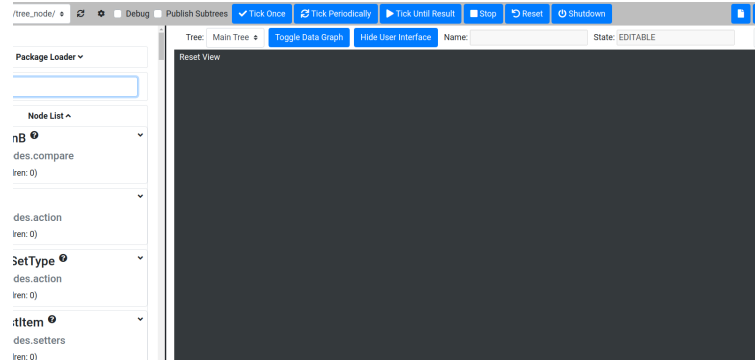
Features

Developer-Perspective

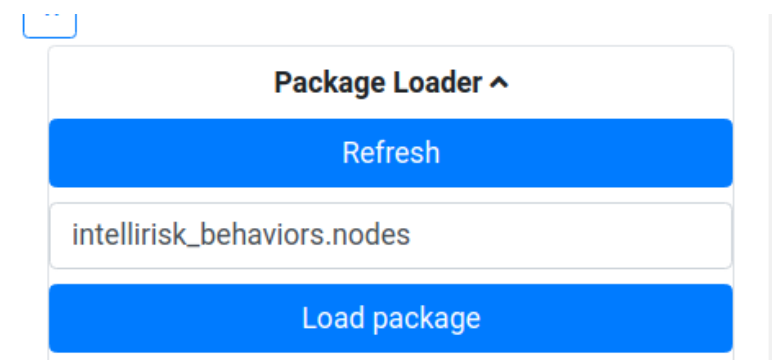


```
@define_bt_node(  
  NodeConfig(  
    version="0.9.0",  
    options={  
      "service_type": type,  
      "request_type": type,  
      "response_type": type,  
      "wait_for_response_seconds": float,  
    },  
    inputs={  
      "request": OptionRef("request_type"),  
      "service_name": str,  
    },  
    outputs={"response": OptionRef("response_type")},  
    max_children=0,  
  )  
)
```

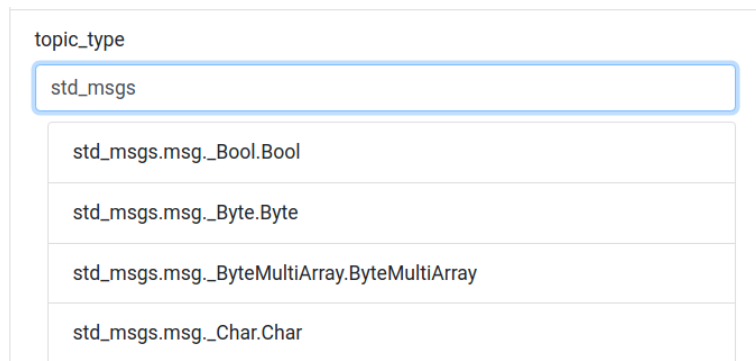
Python-based Nodes



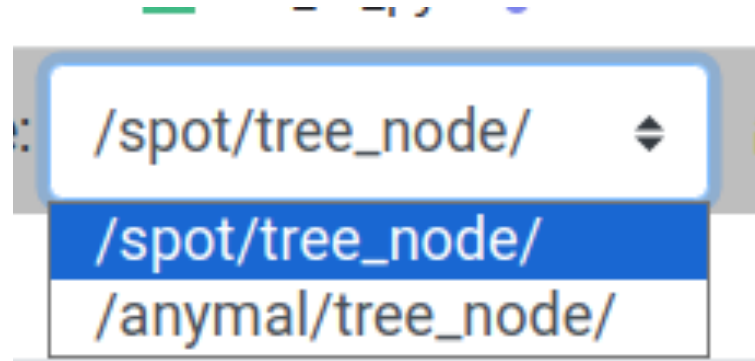
Live Web-Editor



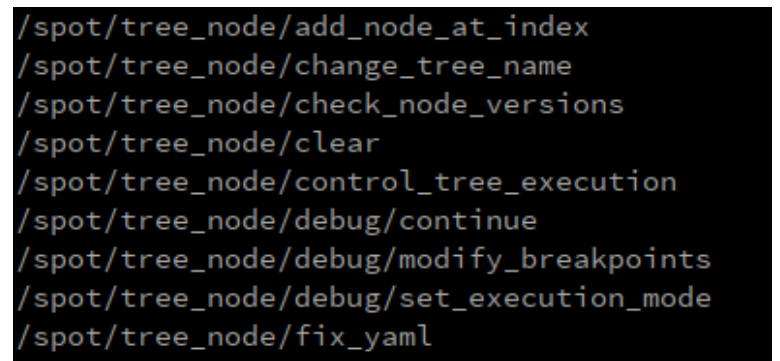
Hot-Reload of Nodes



ROS Message Autocompletion



“Remote”-Development



ROS-based Control Interface

Comparison to BehaviorTree.CPP

	ros_bt_py	BehaviorTree.Cpp
Typeing	✓	✓
Asynchronous Execution	✓	✓
Data Graph	✓	✗
Black Board	✗	✓
Execution Logging and Debugging	(✓)	✓
Remote Development	✓	✗
Node Hot-Reloading	✓	✗
ROS Integration	✓	✓
Dynamically Dispatched Capabilities	✓	✗
Development Philosophy	• “Online Development” • (Almost) No Code • Quick Prototyping	• “Offline Development” • Compiled Node-Library • Fast Runtime Performance

Let's build a BT with `ros_bt_py`!

Scenario

- (Simulated) Husky Robot
- Running Navigation Stack
- Objects that need to be grasped

Requirements

- Load object poses from file
- Move Husky to the poses
- Grasp the object at each pose

Goals YAML File

package://ros_bt_py_sim/config/goals.yaml

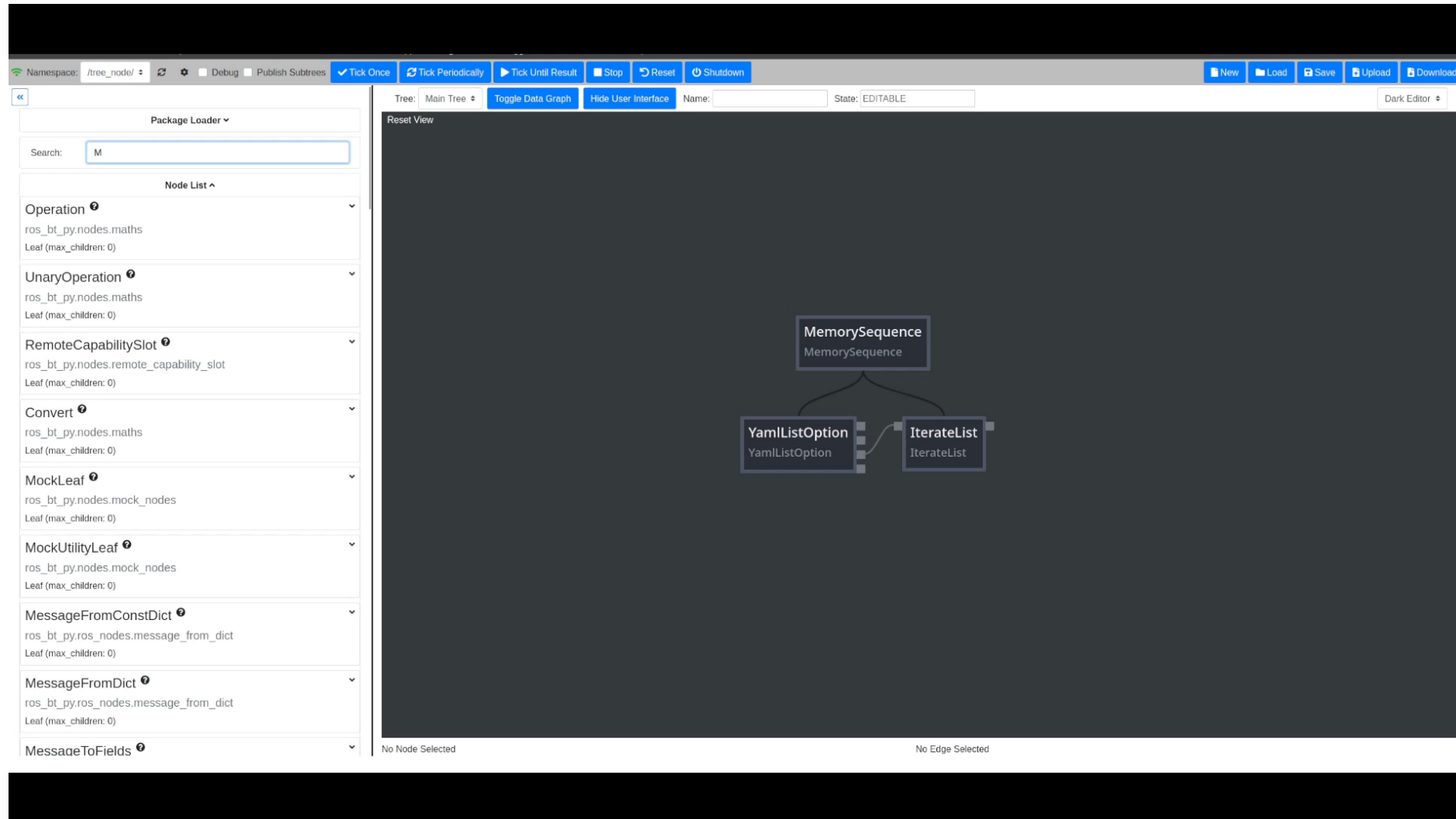
```
- {  
  header: {  
    frame_id: map  
  },  
  pose: {  
    position: {  
      x: 6.978711128234863,  
      y: 1.7688500881195068,  
      z: 0  
    },  
    orientation: {  
      x: 0,  
      y: 0,  
      z: 0.8030835388209147,  
      w: 0.5958664528859436  
    }  
  }  
}  
- {  
  header: {  
    ....
```

Web-Editor Overview

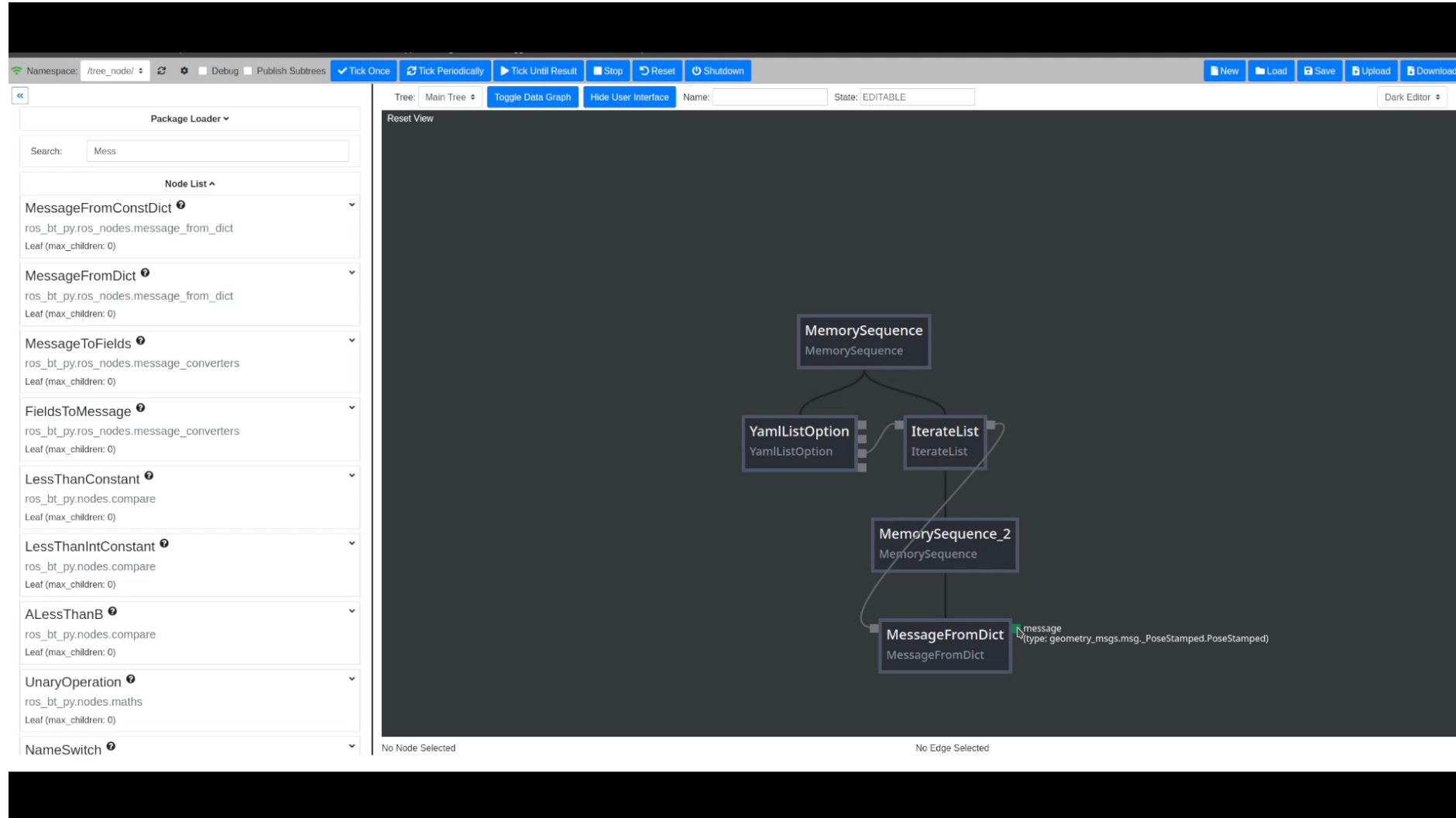


A screenshot of the Web-Editor interface. The interface is divided into several sections. At the top, there is a toolbar with buttons for 'Tick Once', 'Tick Periodically', 'Tick Until Result', 'Stop', 'Reset', and 'Shutdown'. Below this, there is a 'Package Loader' section with a search bar. To the left, there is a 'Node List' section containing a list of nodes: 'ALessThanB', 'Action', 'ActionForSetType', 'AppendListItem', 'Compare', 'CompareConstant', 'CompareNewOnly', 'Constant', and 'Convert'. Each node entry includes its name, a path (e.g., 'ros_bt_py.nodes.compare'), and a description (e.g., 'Leaf (max_children: 0)'). The main area of the interface is a large dark gray canvas labeled 'Reset View'. Above this canvas, there are buttons for 'Toggle Data Graph' and 'Hide User Interface', and input fields for 'Name' and 'State: EDITABLE'. At the bottom of the canvas, there are status indicators: 'No Node Selected' and 'No Edge Selected'. The interface also includes a 'Dark Editor' toggle in the top right corner.

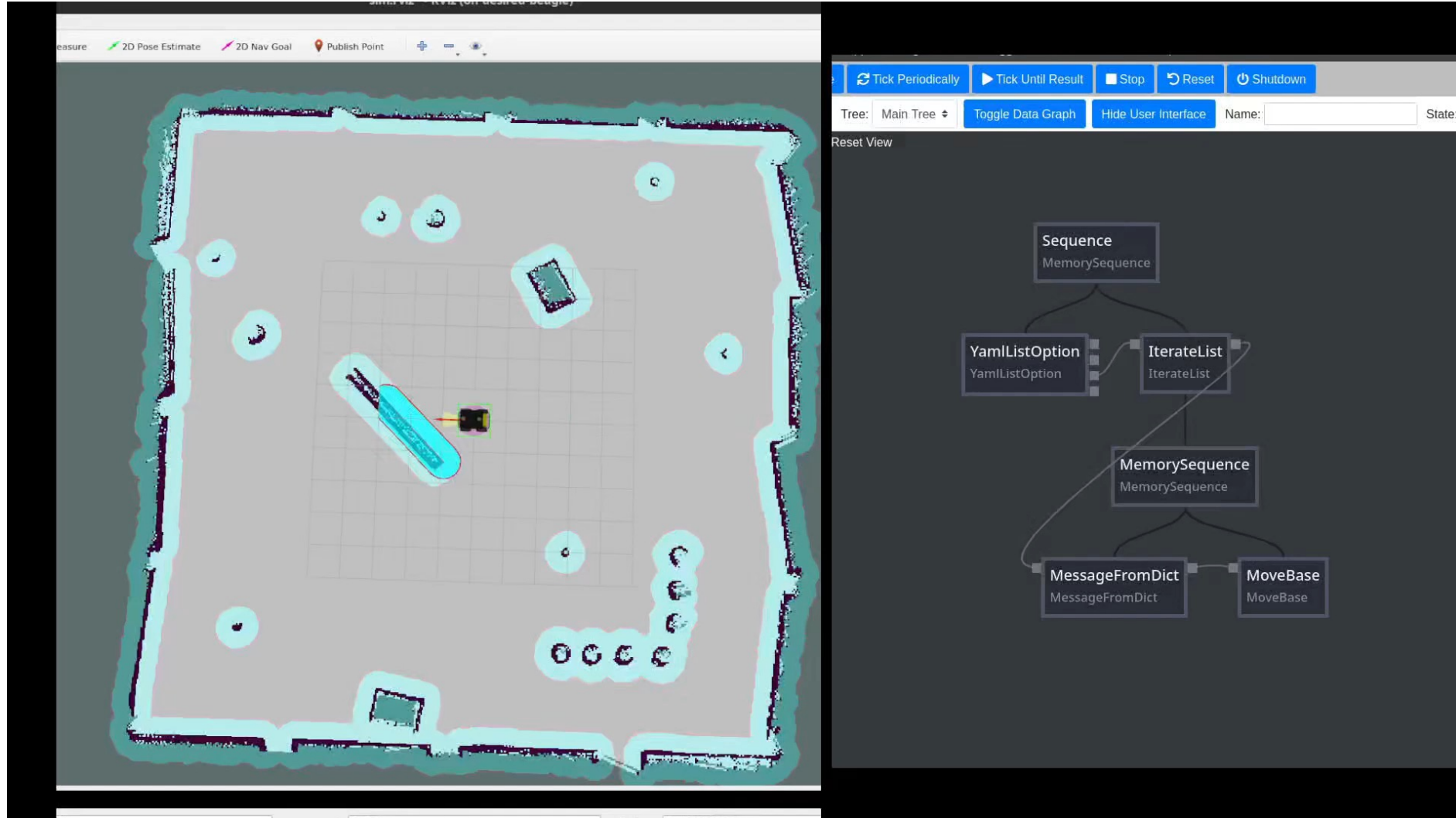
ROS-Messages Integration



Capability-based Dynamic Dispatch

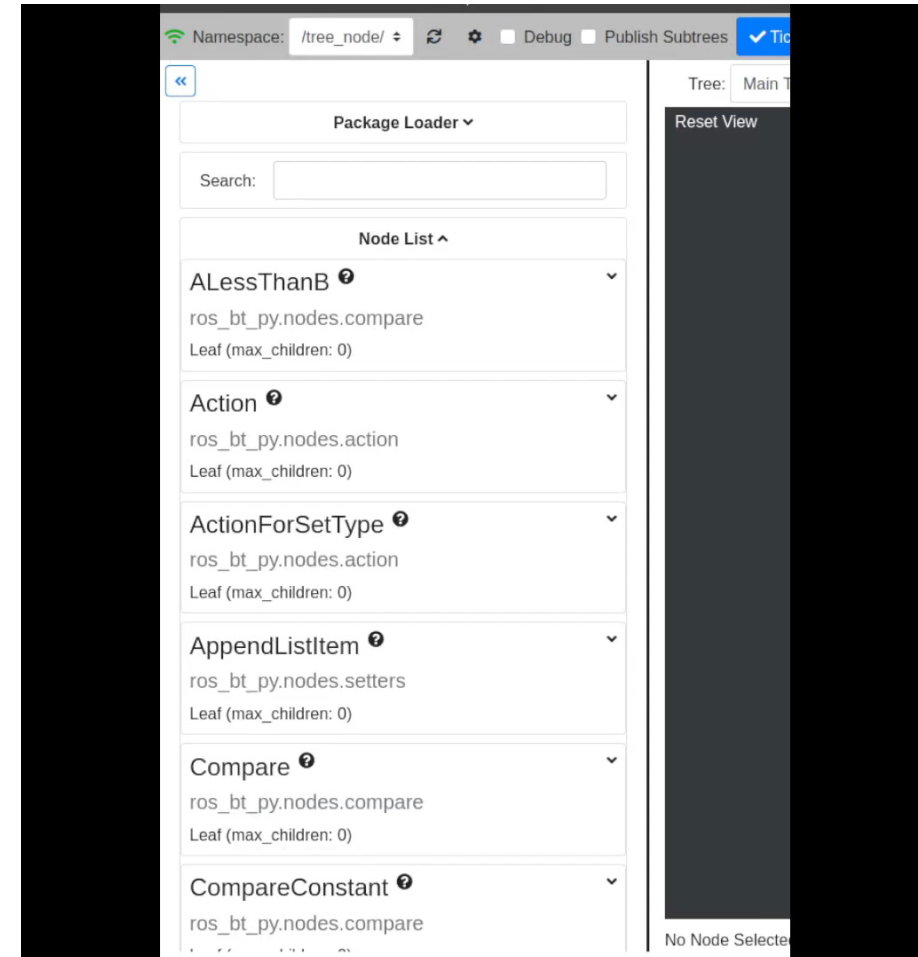


Running the Behavior Tree



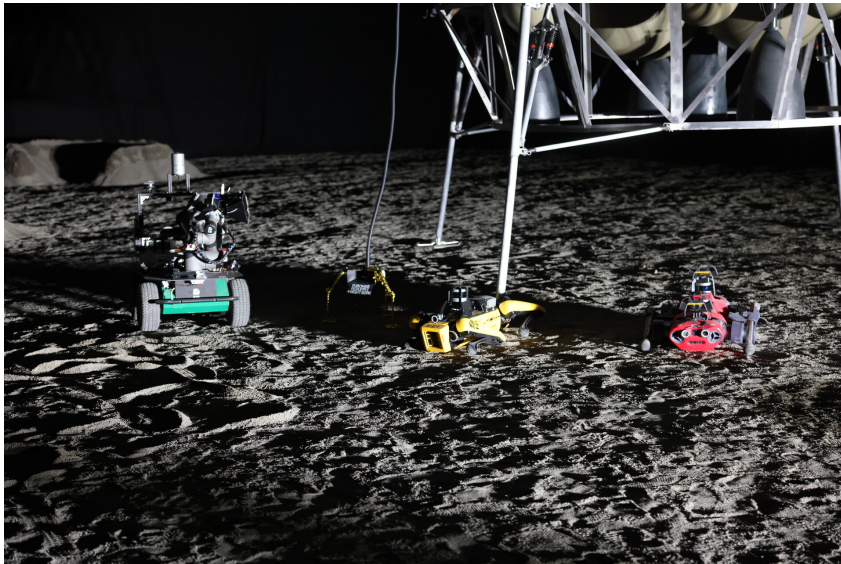
Defining a new Node

```
tabs 1 grasp.py 1 lookup_tf.py
1 from ros_bt_py.node import Leaf, define_bt_node
2 from ros_bt_py.node_config import NodeConfig
3
4 from ros_bt_py_msgs.msg import Node as NodeMsg
5 from ros_bt_py.ros_helpers import AsyncServiceProxy
6
7 @define_bt_node(
8     NodeConfig(
9         version="0.1.0",
10         options={
11             "timeout_secs": float,
12             "grasp_params": dict
13         },
14         inputs={
15             "object_index": int,
16         },
17         outputs={},
18         max_children=0,
19         optional_options=[],
20     )
21 )
22 class HuskyGrasp(Leaf):
23     """Grasp an object using the Husky robot."""
24 +--- 2 lines: def _do_setup(self):.....
25
26 +--- 27 lines: def _do_tick(self):.....
27
28 +--- 4 lines: def _do_untick(self):.....
29
30 +--- 6 lines: def _do_reset(self):.....
31
32 +--- 2 lines: def _do_shutdown(self):.....
```



Evaluation

Projects



**ESA Space Resources
Challenge**



intelliRisk 2

Future Improvements

- ROS2 Open Source Release
 - Feature Equality between ROS1 and ROS2
- Redesigned Web-Editor
 - Capability Editor
 - Robot-Team Capability Overview
- Improved Debugging and Logging Facilities
- Improved Multi-Robot Capability Allocation

Questions?

David Oberacker – Oberacker@fzi.de



ESA Space Resources Challenge

