FZI Forschungszentrum Informatik

ros_bt_py

Behavior Tree Framework for Robot Missions and Capabilities

David Oberacker
Oberacker@fzi.de

Motivation

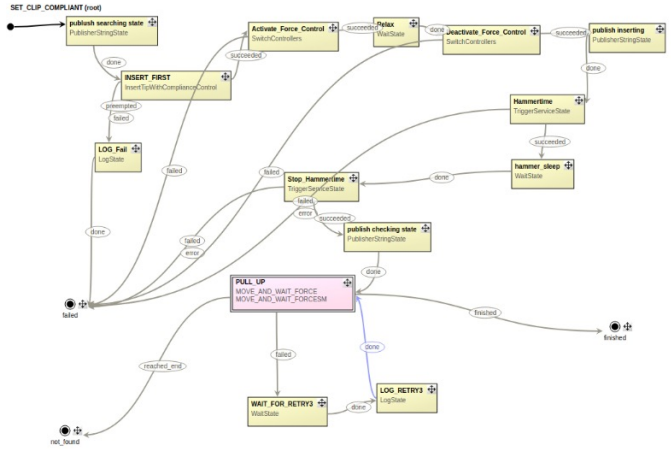
Creating High Level Missions for Your Robot

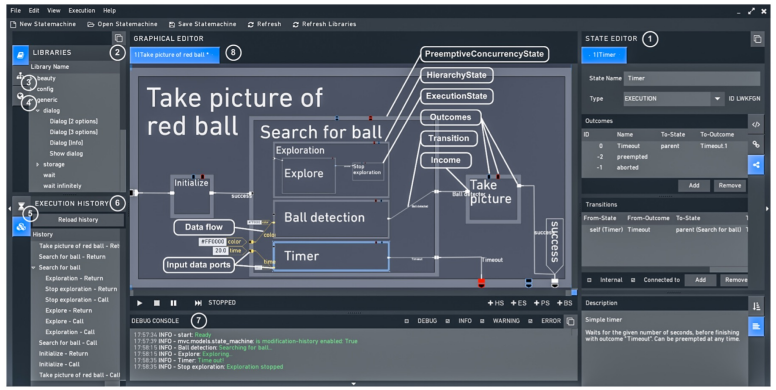
Various **State Machine** options are available

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

BUT:

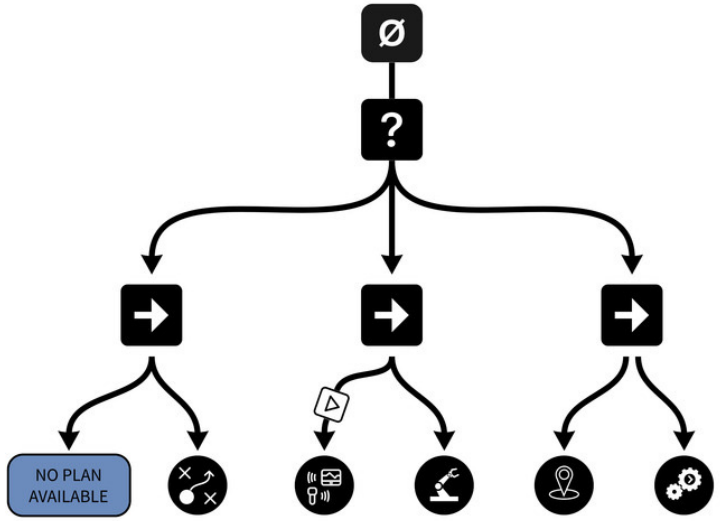
- Tedious to use in larger Projects
- Difficult to change and modify on the fly





Behavior Trees as the new standard

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



Motivation

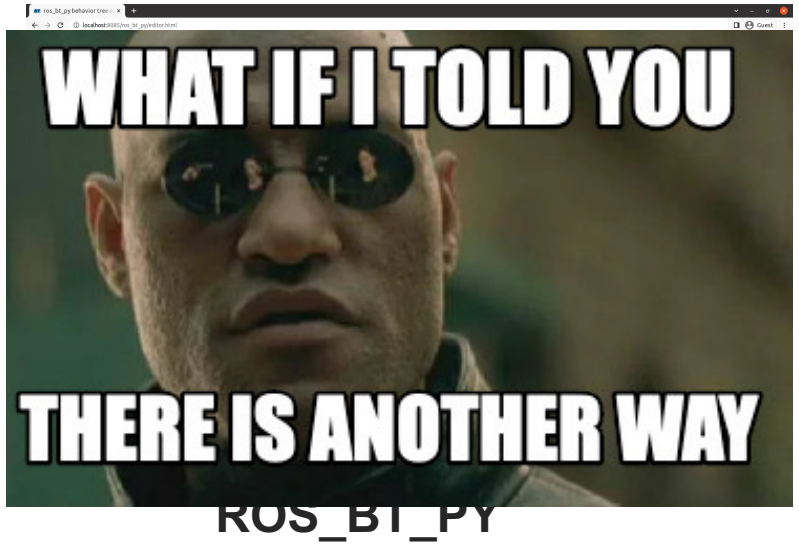
What we need from Behavior Trees

Current Behavior Tree Frameworks are

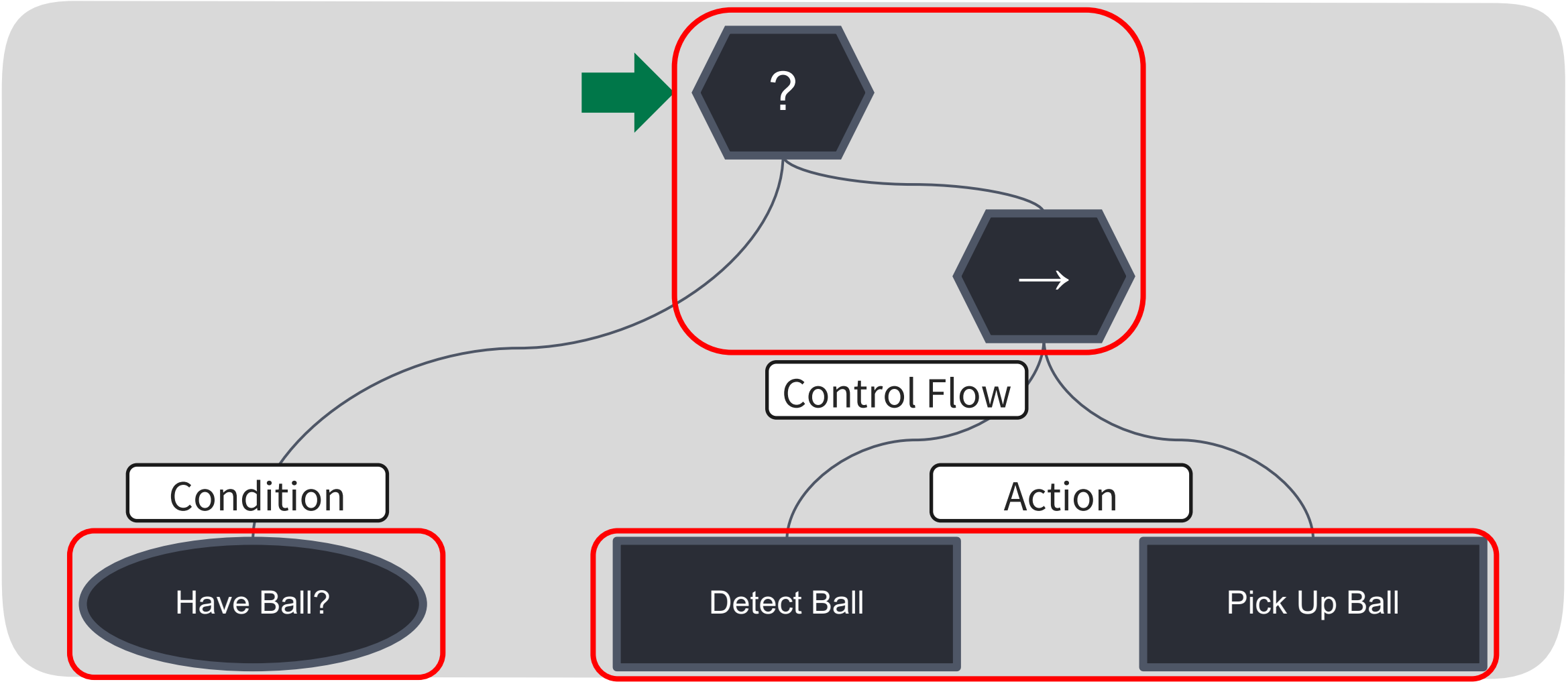
- less mature and provide fewer features than HSM
- often focussed in their application or project
- still diffcult to use
- ROS Integration is often an afterthought

What do we want from a Behavior Tree Framework?

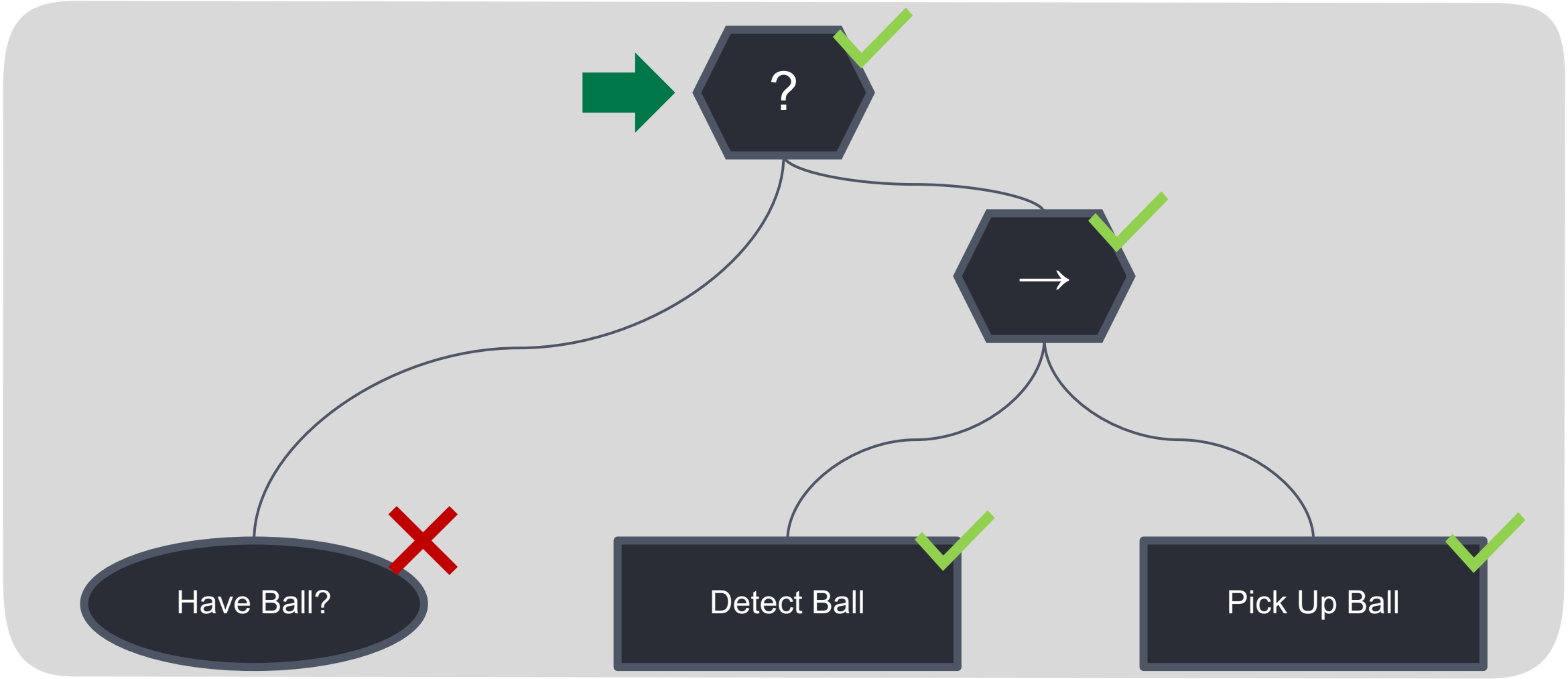
- Easy to use offline and online
- Modular, Flexible and Multi-Robot Enabled
- Rapid mission building for various robots and applications
- ROS Integration at every Step



Behavior Trees - Terminology



Behavior Trees - Execution & Ticking



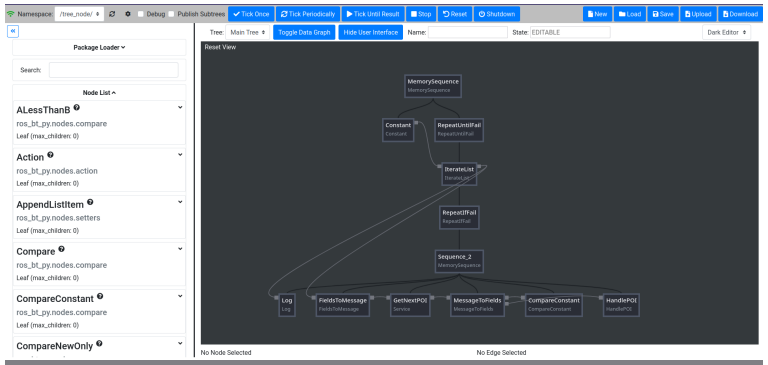
ros_bt_py - Library

```
        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 *

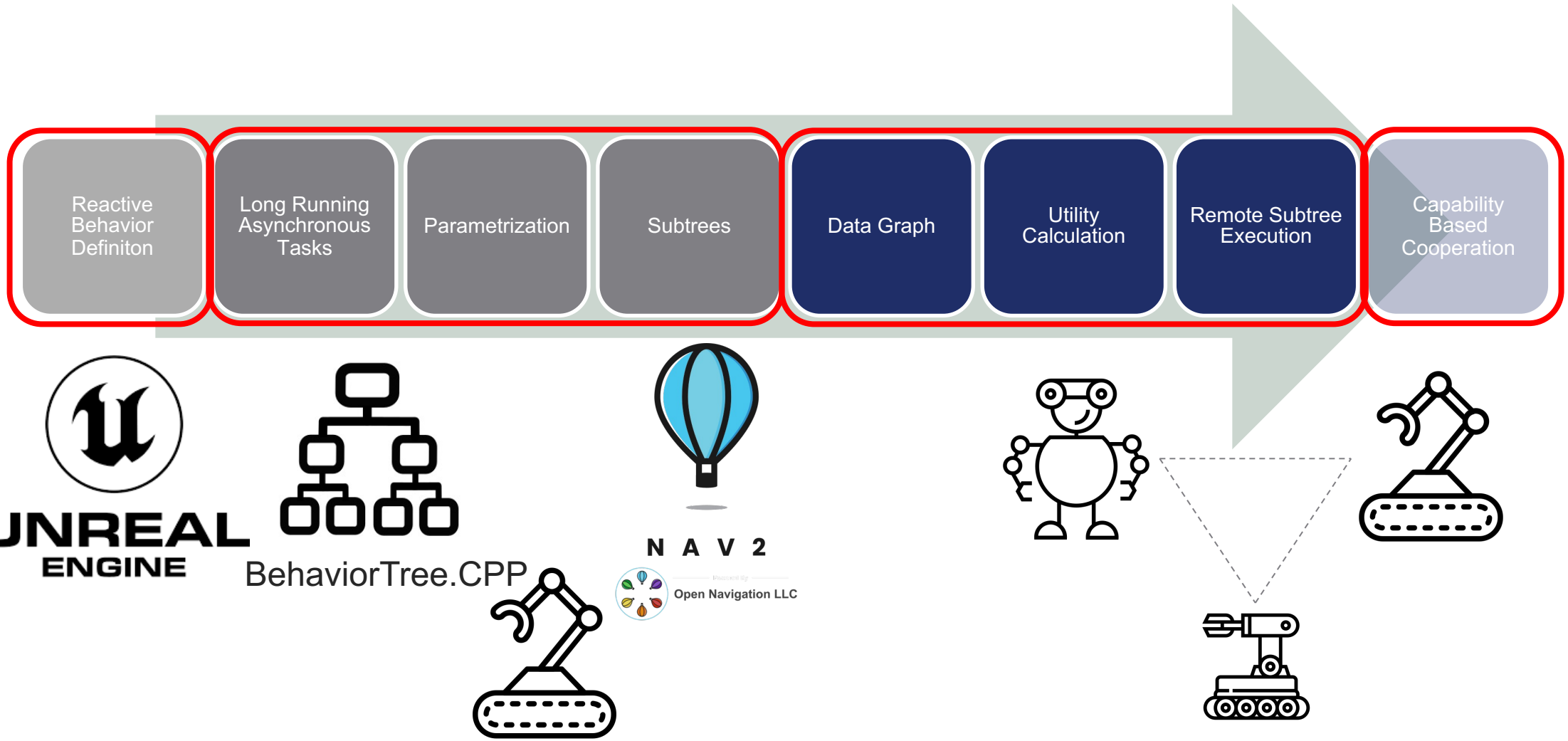
ROS

Native ROS 1/2 Integration

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

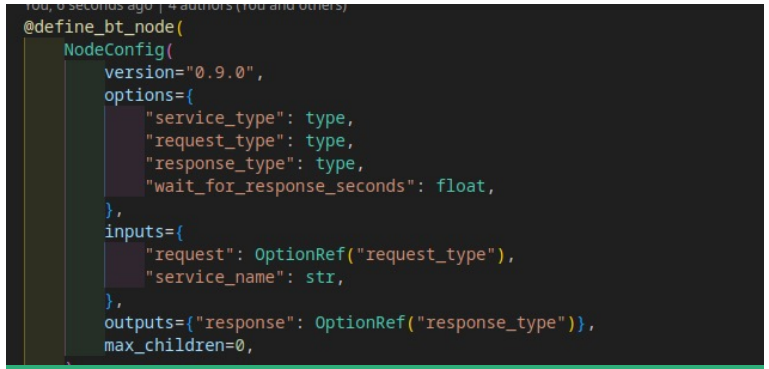
Features

Behavior Tree-Perspective

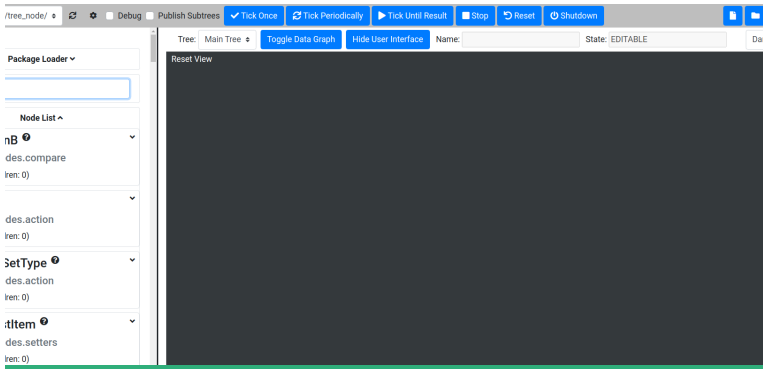


Features

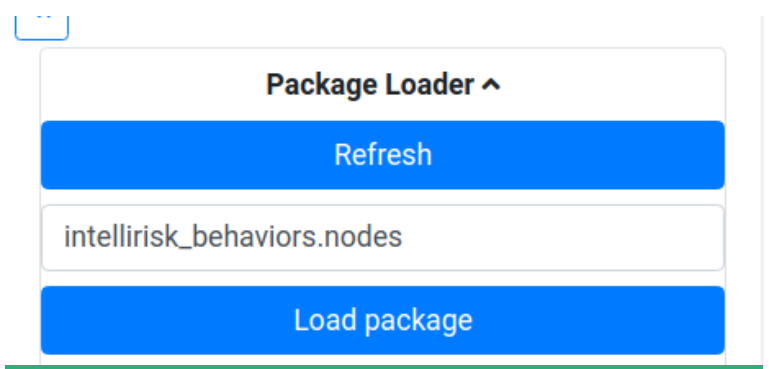
Developer-Perspective



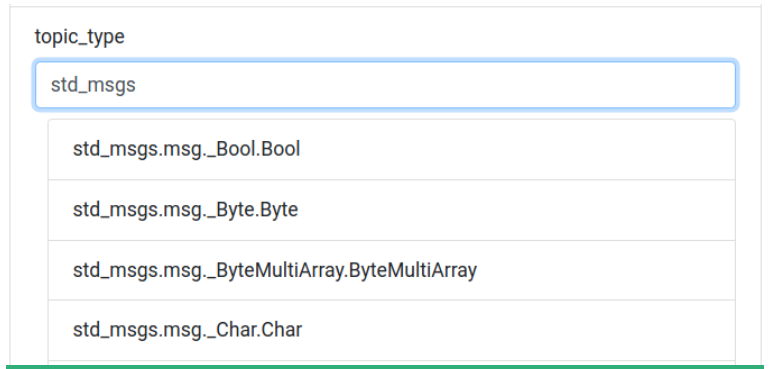
Python-based Nodes



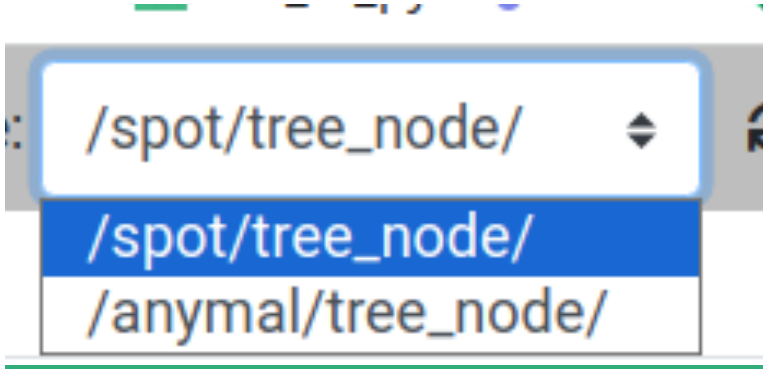
Live Web-Editor



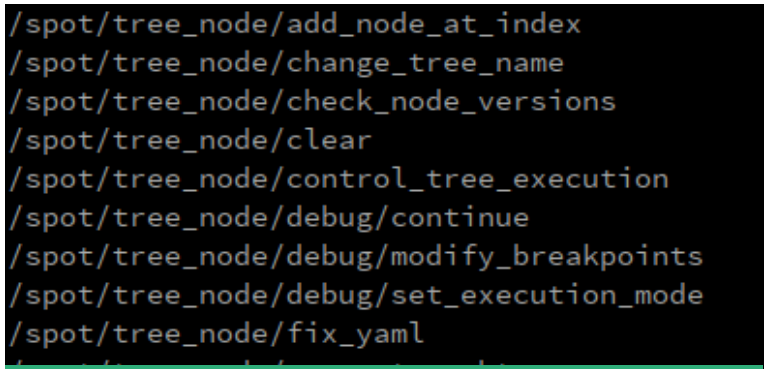
Hot-Reload of Nodes



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 Dispached 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!

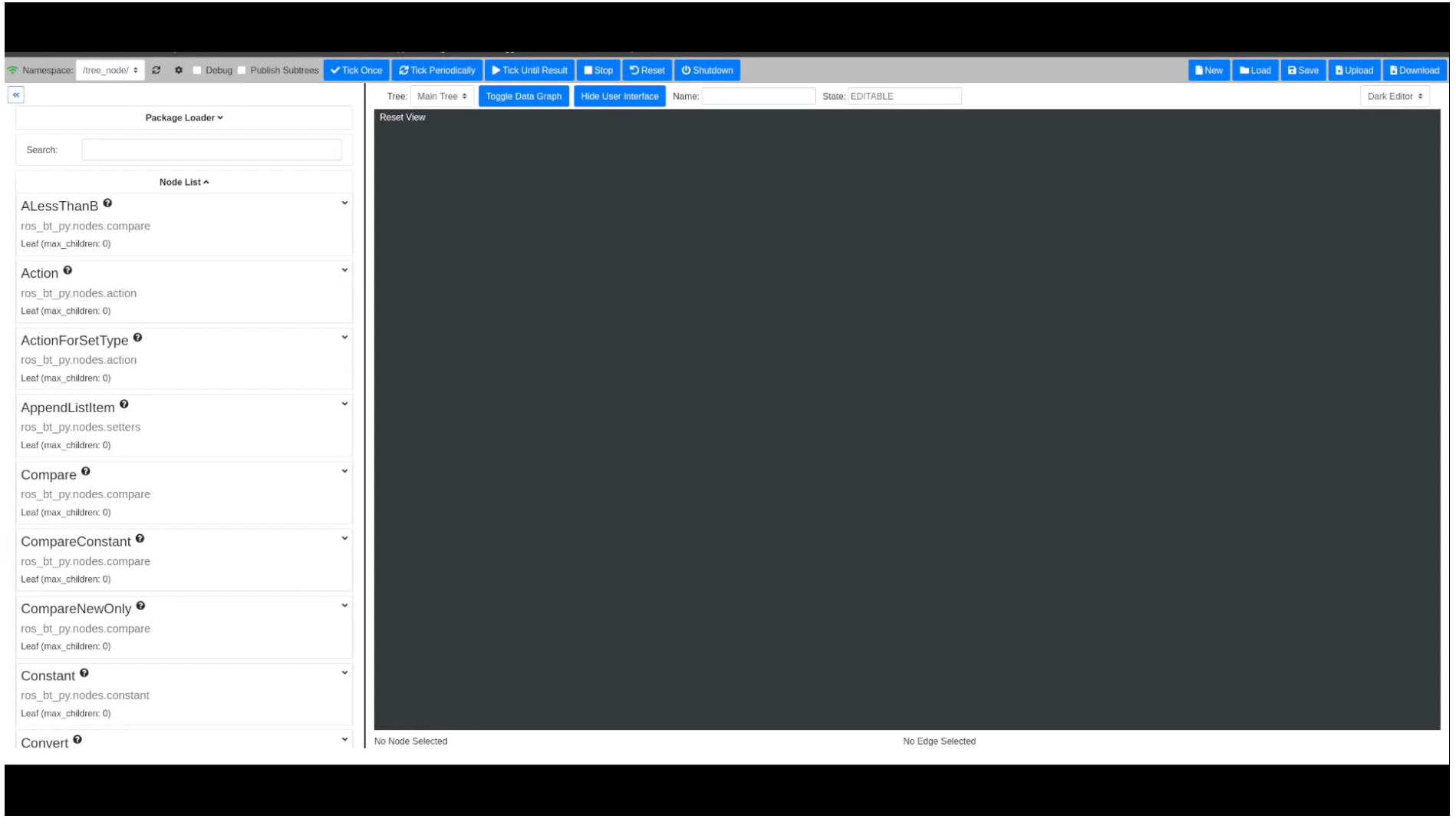
Scenario	– (Simulated) Husky Robot – Running Navigation Stack – Objects that need to be grasped
Requirements	– Load object poses from file – Move Husky to the poses – Grasp the object at each pose

Goals YAML File

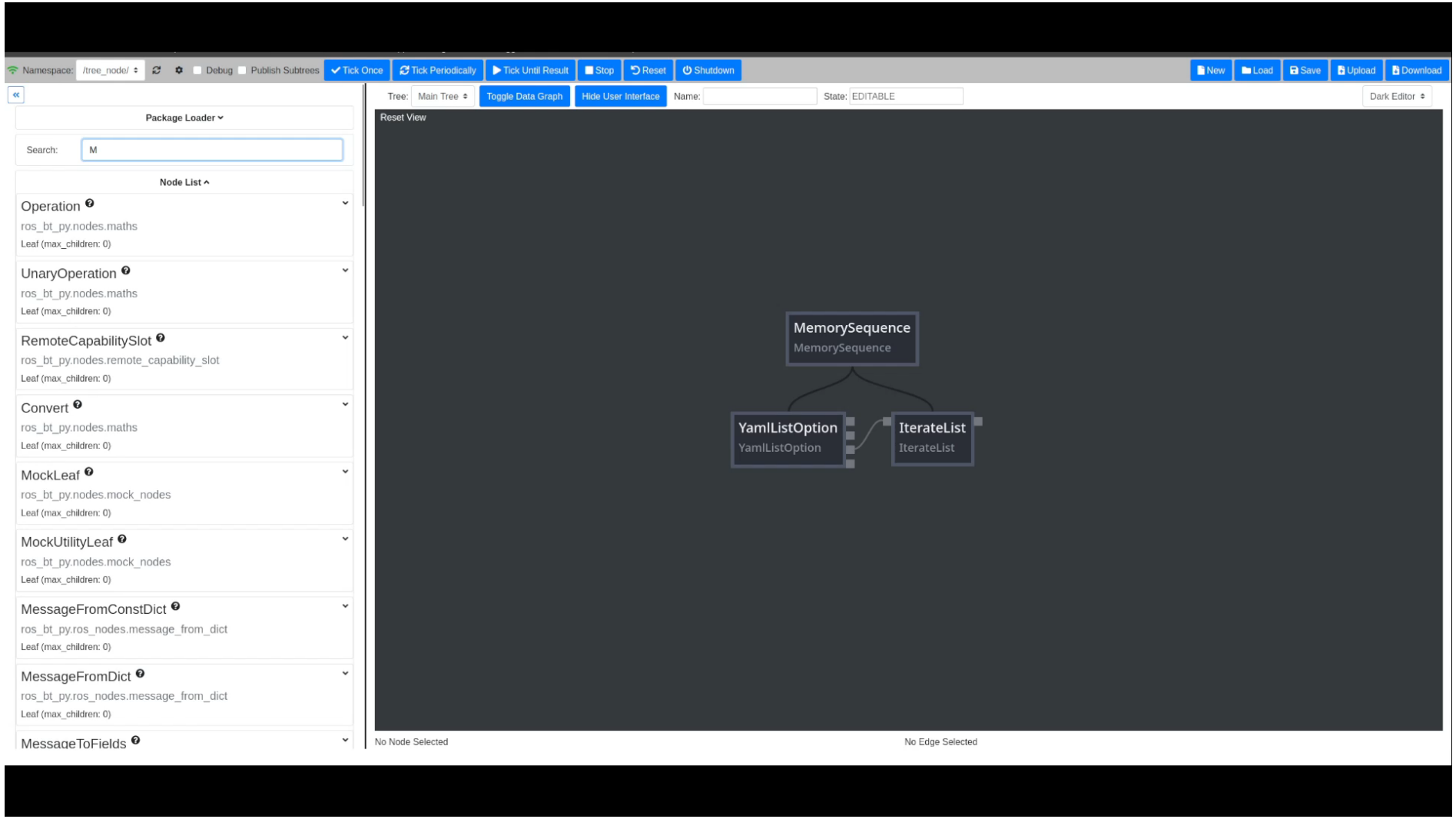
package://ros_bt_py_sim/config/goals.yaml

```
- {
  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: {
....
```

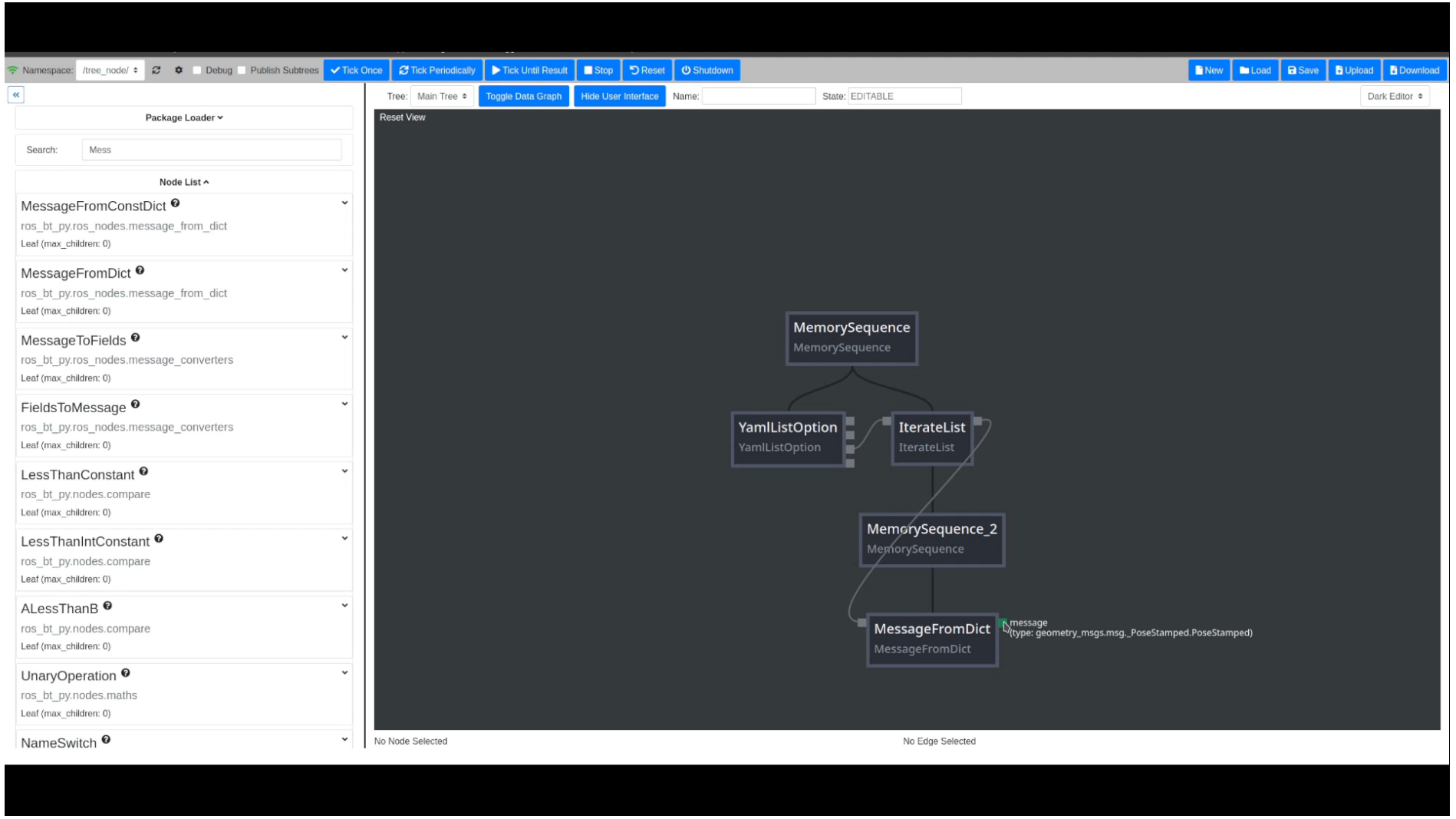
Web-Editor Overview



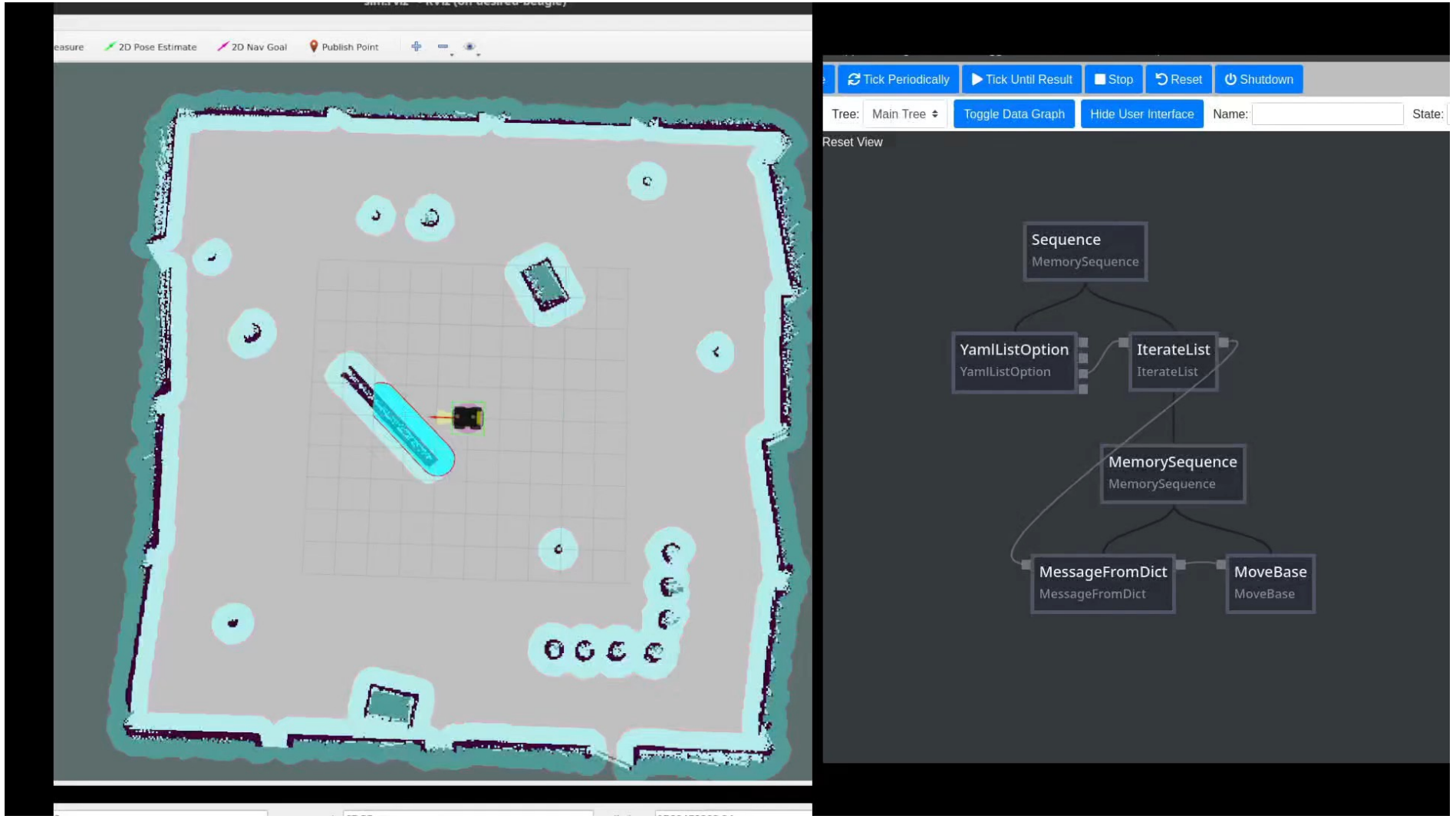
ROS-Messages Integration



Capability-based Dynamic Dispatch

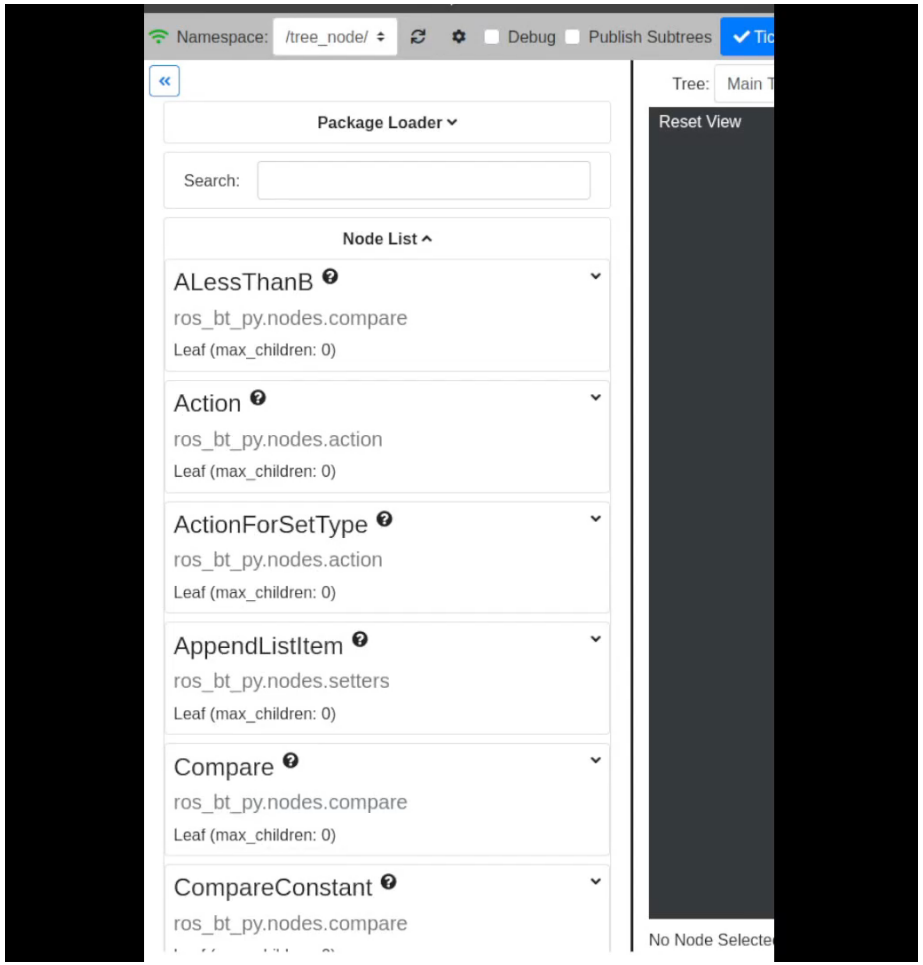


Running the Behavior Tree



Defining a new Node

```
tabs   1 grasp.py   1 lookup_tf.py
1  from ros_bt_py.node import Leaf, define_bt_node
   1 from ros_bt_py.node_config import NodeConfig
   2
   3 from ros_bt_py_msgs.msg import Node as NodeMsg
   4 from ros_bt_py.ros_helpers import AsyncServiceProxy
   5
   6 @define_bt_node(
   7     NodeConfig(
   8         version="0.1.0",
   9         options={
  10             "timeout_secs": float,
  11             "grasp_params": dict
  12         },
  13         inputs={
  14             "object_index": int,
  15         },
  16         outputs={},
  17         max_children=0,
  18         optional_options=[],
  19     )
  20 )
  21 class HuskyGrasp(Leaf):
  22     """Grasp an object using the Husky robot."""
  23
  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):·····
```

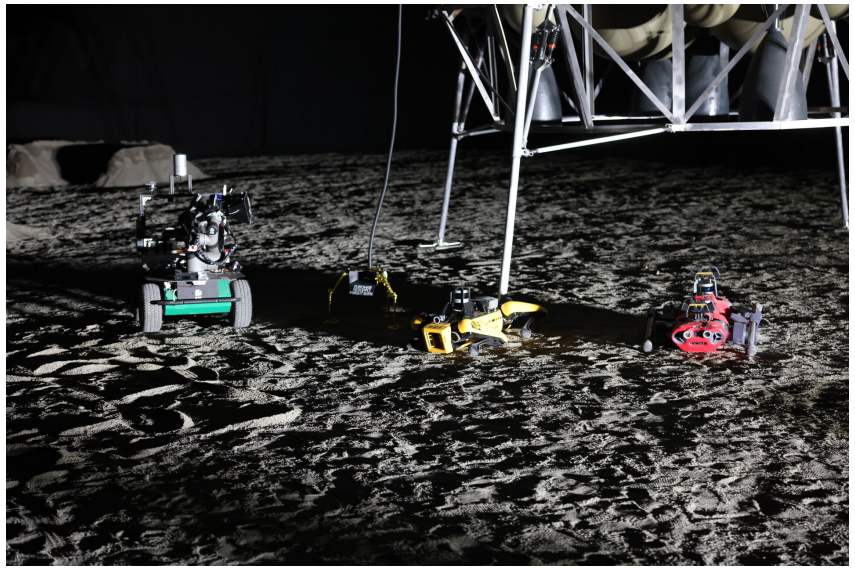


Evaluation

Projects

ROBDEKON

SeRoNet



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