MoveIt Studio

Creating Behaviors // Guidelines & Standards

Index

Naming & Descriptions

Ports

Grouping & Subtrees

This presentation is meant to highlight the general problems seen from a non-target user and non-roboticist perspective.

All observations and proposals are strictly from the view of general usability and best-practices based on the current implementation of the Behavior Tree system.

1

Naming & Descriptions

Similar names and/or similar behaviors in different sub-categories:

Behaviors → Perception → Write Calibrated Pose to YAML

Write pose (x,y,z, roll, pitch, yaw) to YAML file. This behavior is meant to be called after the Calibrate Pose Action. Note: The behavior saves the calibrated_pose into the ~/.config/moveit_studio/calibration folder

Behaviors → Perception → Calibrate Pose Action

Calls a robot_calibration_msgs::action::CalibratePose action server and outputs the results on the calibrated_poses port. For now this behavior only supports sending a single frame for calibration.

Behaviors → Miscellaneous → Write Pose to YAML

Write pose (x,y,z,qx,qy,qz,qw) to YAML file.

Similar names and/or similar behaviors in different sub-categories (continued):

- Why are the two “Write Pose” behaviors in different sub-categories?
  - If ‘Calibrate Pose Action’ is directly related to ‘Write Pose to YAML’ shouldn’t they be in the same group?
  - If the actions are completely different but the behaviors are still related, shouldn’t we alert the user of where to go to find that related action?
- What’s the difference between writing a pose and writing a “calibrated” pose?
  - Is there ever a scenario I would want the written pose to not be calibrated?
    - If yes, how do I know what that is without any explanation of when to use it?
    - If no, why does a non-calibrated version exist?
- Is ‘Write Calibrated Pose to YAML’ the same as running ‘Calibrate Pose Action’ and ‘Write Pose to YAML’?
  - If no, why don’t we explain when to use each in the definition and/or provide an option to send them directly to the correct one?
  - If yes, why do we have three behaviors for one (or possibly two) action in multiple sub-categories?

Nearly identical behaviors and descriptions:

Setup MTC Interpolate To Joint State

Given an existing MTC Task object and a joint state, appends MTC stages to describe a joint-interpolated motion plan to that joint state.

Setup MTC Cartesian Move To Joint State

Given an existing MTC Task object and a joint state, appends MTC stages to describe a cartesian motion plan to that joint state.

- Assuming I know the difference between the “joint-interpolated” and “cartesian” motions, how should I decide which to use?
- While consistent descriptions and terminology are good, altering one word that’s the same as the behavior name itself isn’t helpful to my understanding of what each is going to do.

Generic information that may be confusing and/or is too similar to other behaviors:

Get Camera Info

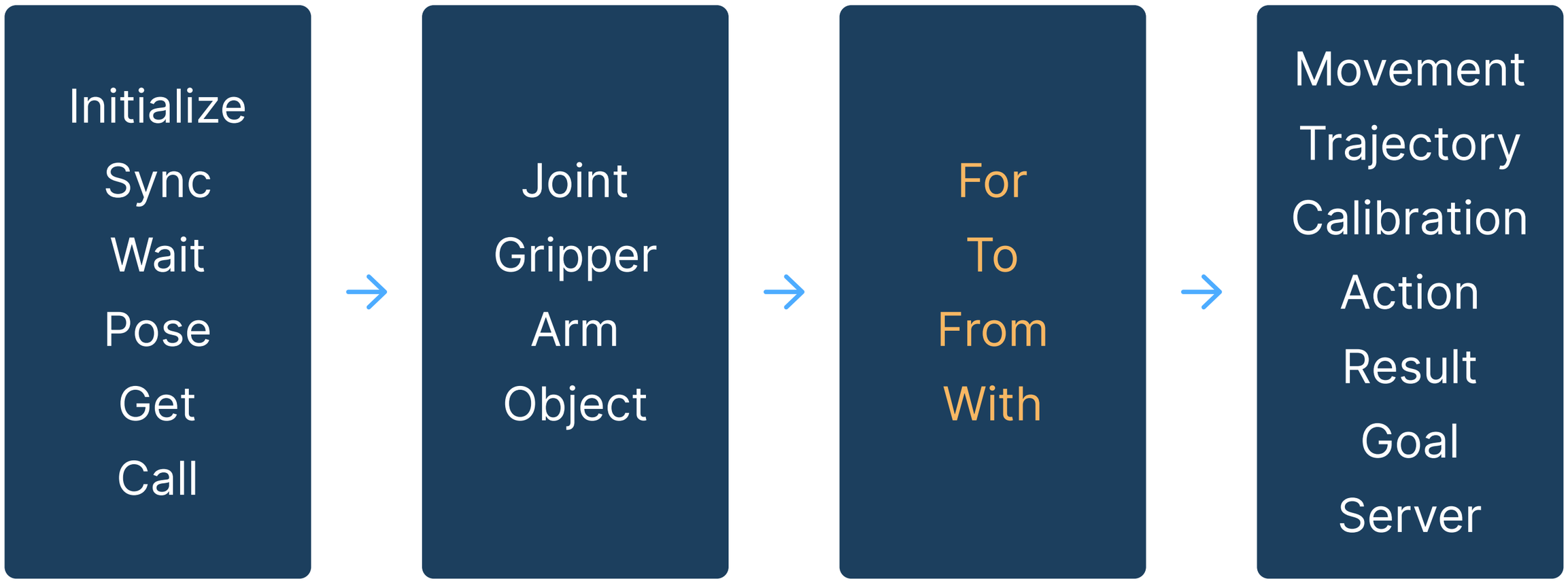
Captures camera information and makes it available on an output port.

Get Synchronized Camera Topics

Time-synchronized multiple topics from the camera system and exposes them on output ports.

- Both objectives provide camera information, but what exactly?
  - Camera information: What is the information?
  - Time-synchronized multiple topics: What are the topics?
- Both link the information to output ports, so why does it read differently?
  - Makes it available on an an output port - Available for me to do what?
  - Exposes them on output ports - What’s the difference between being “available” and being “exposed?”

When creating new custom behaviors in MoveIt Studio we currently use a (sometimes complex) combination set up actions (verbs) & setups (nouns). While the overall idea is good, we need to form some rules around it by coming up with standard definitions for each word as it relates directly to the behaviors. This will hopefully reduce any confusion and also unnecessary duplicates or similar behaviors by avoiding multiple users making assumptions on the meanings.



Example (Final descriptions will need to be decided and agreed upon by the team):

Move: Moves the selected arm/actuator/torso at the set jog speed.

Joint: One or more hinge points on the arm/actuator/torso.

State: The overall physical placement of the selected arm/actuator/torso.

- Provide the user with set definitions of each word to explain the base function.
- Combine this by listing needed inputs from the user and ports required to make it work (which should also be defined).
- Finish with why the user would want to do this and suggest a follow-up action or actions.

Move To Joint State

Moves the selected arm/actuator/torso at the set jog speed by manipulating the joints to a specific end state by taking the input from the joint state message and optional information from the planning group, controllers and constraints to end at [X] or prepare for [X] action.

We should also attempt to reduce the overall word count on some behaviors and redundant nouns/verbs.

Behaviors → Task Planning

There are 18 behaviors that start with “Setup MTC...” and another 5 with “MTC” or “Task Plan” in the name.

- If there are more than [X] number of behaviors using the same words we should consider adding a new sub-group for them.
- In the scenario above, a total of 23 of the 26 behaviors have some version of “MTC” or “Task Plan” in the name despite already being in a sub-category named “Task Planning.”
  - We should attempt to eliminate the redundant words to shorten the names of behaviors making both the list and the behavior tree easier to scan quickly. If the meaning might be lost when viewed in the behavior tree vs. within the sub-category in the list we could add the sub-category label to the node itself or potentially add alternate icons.
    - Example: Instead of “Setup MTC Pick Object” we have an icon and/or smaller text specific for the sub-category of ‘Task Planning’ so the name itself is just “Pick Object” or “Setup Pick Object”

Creating a concise set of terminology and definitions will help to:

- Alleviate any confusion over what a specific word means as it relates to our software.
- Build procedural memory with our users as they will see consistent terminology used for behavior naming.
- Enable faster building of objectives, even with new behaviors, as they will be able to predict what that behavior will do more accurately based on repeat terminology.
- Assist the user in where they should start looking to find a new or previously unused behavior if they are already familiar with a similar one based on consistent naming alone.

2

Ports

Unclear capabilities and/or uses:

Direct feedback from NASA JSC users

IN/OUT: detect_april_tags

“What does this actually do? What is it detecting? We looked for another objective that was using it to see if we could figure out what it’s supposed to do.”

- Similar issues with general behavior naming and descriptions.
  - What are the common ports and how would I know where to find information on them?
  - Are naming conventions the same or similar to behaviors naming?
    - If yes, all of the terminology should be added to a single place.
    - If no, a dedicated port naming structure and descriptions should be defined and available to the user.

No validation and no reason to write code here vs. the XML file:

Direct feedback from NASA JSC users

> “The Objective Builder provides a good visual representation of the structure when I’m explaining it (the objective) to someone but I find it easier to just edit the behaviors in XML.”

- Certain ports should have more precise interactions, aka, if it’s a TRUE/FALSE input, why am I typing that and running the risk of a typo?
- Why isn’t there in-line validation on inputs that can only have a select number of possible correct entries?
  - I tested this by editing ‘Move to Pose,’ selecting the ‘Setup MTC Move to Pose’ node, and altering the ‘IN: use_all_planners’ port from “false” to “tru” and saving without issue.
- Why are port some code entries in brackets and others not?
  - How do I know when to input one over the other, especially with no in-line validation?
- Users are frequently commenting lines of code in/out for testing purposes, why do we not allow this in port code?
- Users can bulk edit the same line of code in most editors across multiple ports, so why does this have to be done manually one-by-one on behaviors?

No descriptions or instructions:

parameter_name

interpolate_to_joint_state.target_joint_state

controller_names

/joint_trajectory_controller /robotiq_gripper_controller

- Is the code shown here an ID, direct code, a path to code, a file, or set of files it’s using?
  - How do I know what I should be entering?
- When it’s “name” vs “names” shouldn’t there be multiple field options?
  - If this is a path to a file or group of names, how can I see what they actually are?
- How do these relate to the other ports in the behavior?
- If these ports are used across multiple behaviors (especially within the same objective), why can’t I automatically select to duplicate the information?