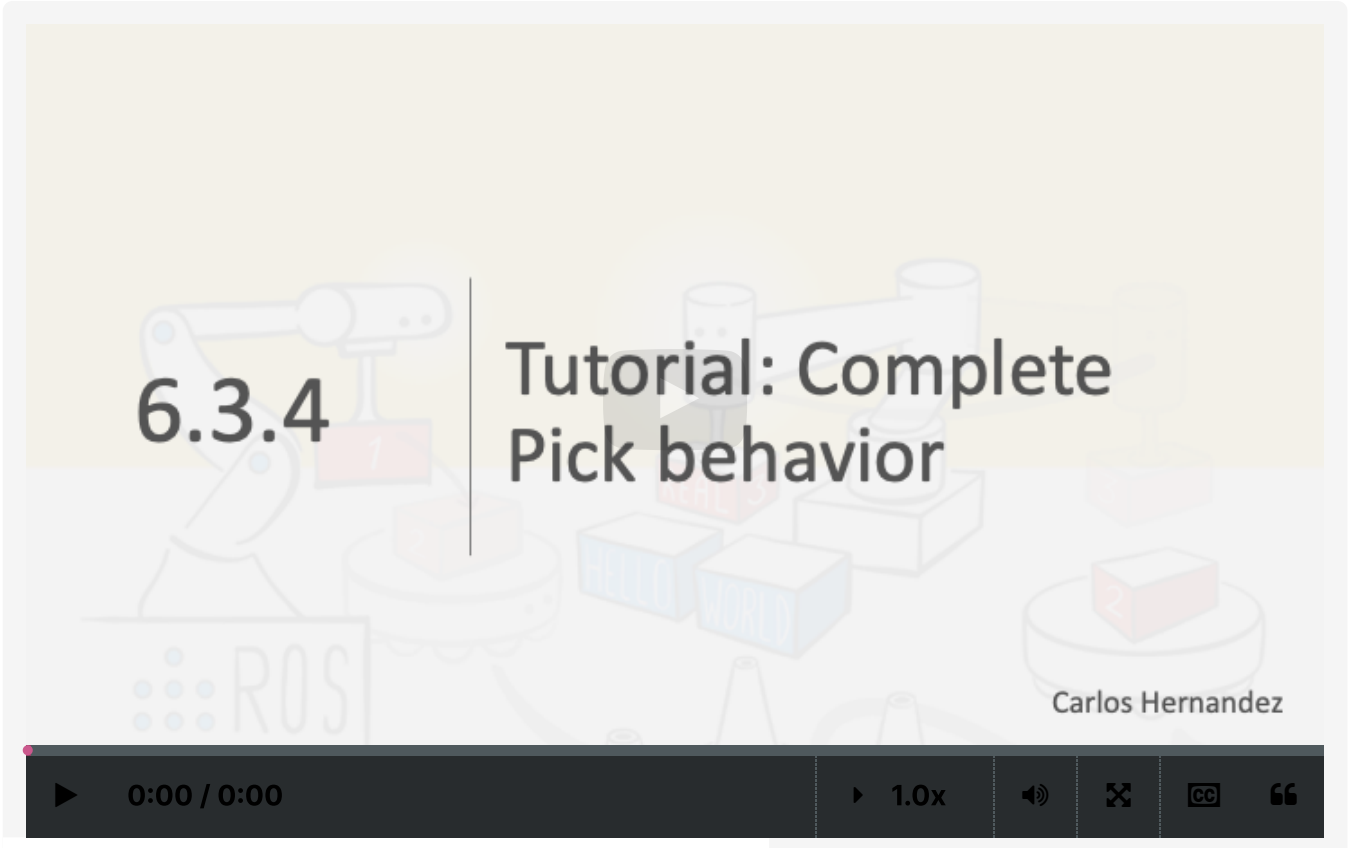
In this video is a tutorial on how to set up a FlexBE behavior. It will show you the steps required and give you some tips on what to think of when configuring a FlexBE behavior .

Important Note:

If you want to reproduce the behavior of the video, screenshots of the required configuration are provided after the summary.

Tutorial Compl. Pick behavior



Video ▼ Subtitles ▼

Other ▼

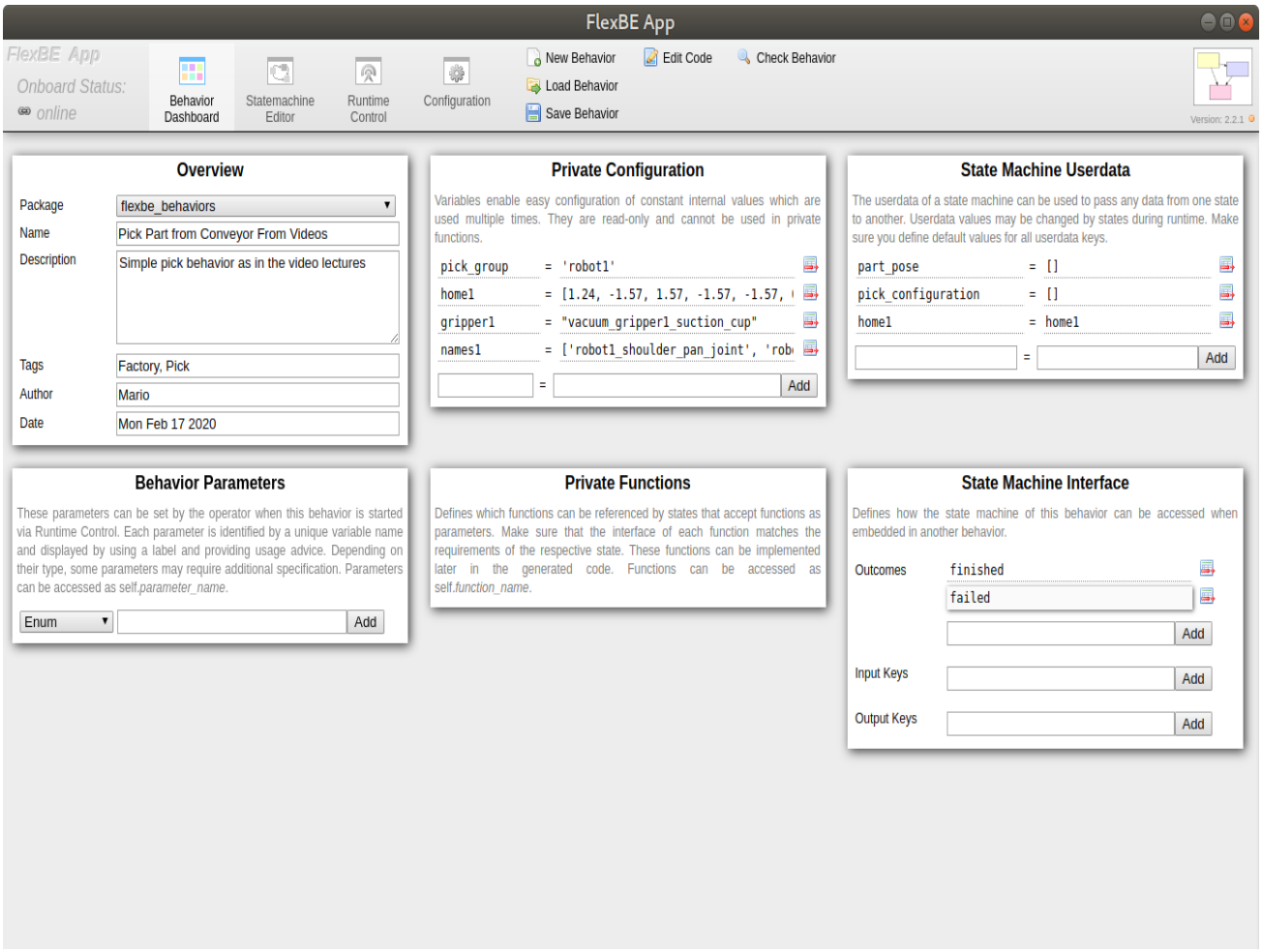
In this video you were shown how to configure a complete FlexBE behavior, by entering the values for all state parameters and input and output keys.

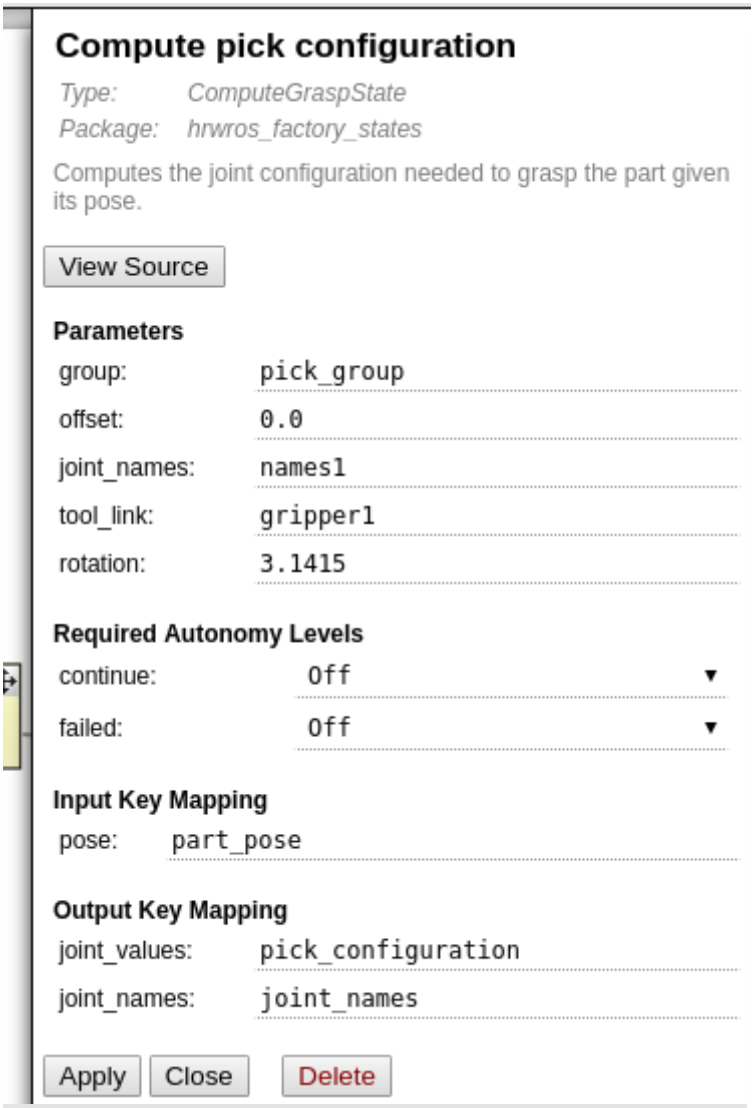
- Don't forget to use quotation marks when entering literal *string* values.
- Don't forget to hit the *Apply* button when your done configuring a state.
- When stumbling across undetermined *output keys*, you can do the following: hover over the name of the output key. This will show you what data type the output key expects. Next, you create a new user data variable with that data type.
- Sometimes it might seem that configuring the states of your behavior in FlexBE takes a lot of clicking. Often using the same variables. But think of how much work all this would be coding this in your own nodes or scripts, passing values trough functions and assigning them to variables. This would also be more prone to errors.
- *State parameters* (or input and output keys) can have the same or similar names the value you need to enter for another state parameter, but they are two different things. By coincidence names might be the same or similar.
- Don't forget to save the changes you made to your behavior!

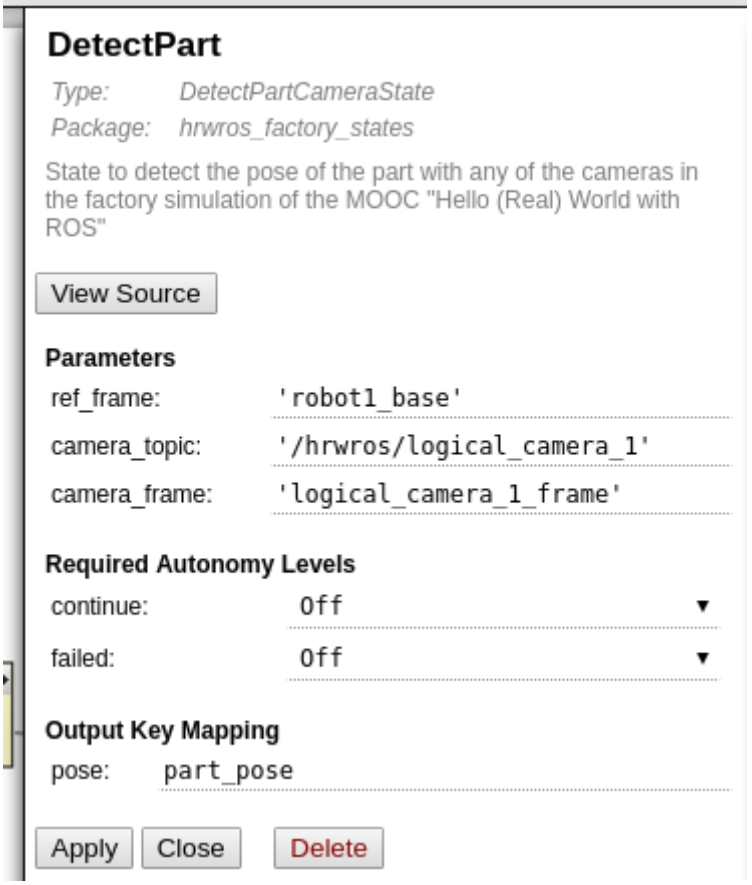
In the next video you will learn how to execute the FlexBE behavior we just created.

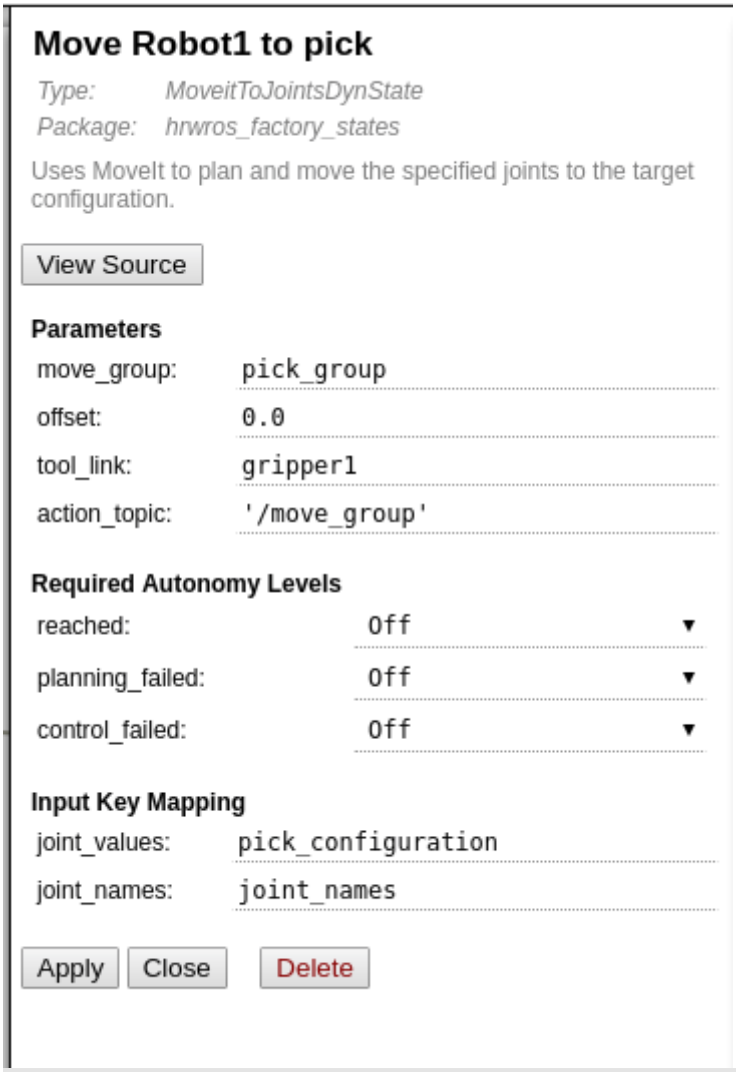
Here you will find the Behavior Dashboard, as well as the configuration for each state, so you can reproduce the same behavior as in the video.

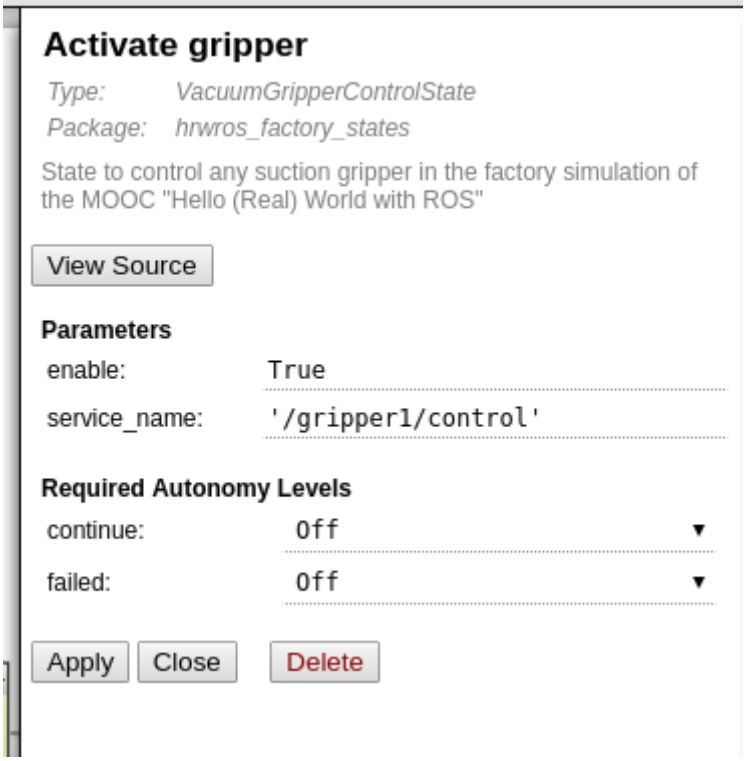
Behavior Dashboard


FlexBE App
FlexBE App
Onboard Status:
online
Behavior Dashboard
Statemachine Editor
Runtime Control
Configuration
New Behavior
Load Behavior
Save Behavior
Edit Code
Check Behavior
Version: 2.2.1
Overview
Package
flexbe_behaviors
Name
Pick Part from Conveyor From Videos
Description
Simple pick behavior as in the video lectures
Tags
Factory, Pick
Author
Mario
Date
Mon Feb 17 2020
Private Configuration
Variables enable easy configuration of constant internal values which are used multiple times. They are read-only and cannot be used in private functions.
pick_group = 'robot1'
home1 = [1.24, -1.57, 1.57, -1.57, -1.57,
gripper1 = "vacuum_gripper1_suction_cup"
names1 = ['robot1_shoulder_pan_joint', 'rob
Add
State Machine Userdata
The userdata of a state machine can be used to pass any data from one state to another. Userdata values may be changed by states during runtime. Make sure you define default values for all userdata keys.
part_pose = []
pick_configuration = []
home1 = home1
Add
Behavior Parameters
These parameters can be set by the operator when this behavior is started via Runtime Control. Each parameter is identified by a unique variable name and displayed by using a label and providing usage advice. Depending on their type, some parameters may require additional specification. Parameters can be accessed as self.parameter_name.
Enum
Add
Private Functions
Defines which functions can be referenced by states that accept functions as parameters. Make sure that the interface of each function matches the requirements of the respective state. These functions can be implemented later in the generated code. Functions can be accessed as self.function_name.
State Machine Interface
Defines how the state machine of this behavior can be accessed when embedded in another behavior.
Outcomes
finished
failed
Add
Input Keys
Add
Output Keys
Add


Compute pick configuration
Type: ComputeGraspState
Package: hrwros_factory_states
Computes the joint configuration needed to grasp the part given its pose.
View Source
Parameters
group: pick_group
offset: 0.0
joint_names: names1
tool_link: gripper1
rotation: 3.1415
Required Autonomy Levels
continue: Off
failed: Off
Input Key Mapping
pose: part_pose
Output Key Mapping
joint_values: pick_configuration
joint_names: joint_names
Apply
Close
Delete


DetectPart
Type: DetectPartCameraState
Package: hrwros_factory_states
State to detect the pose of the part with any of the cameras in the factory simulation of the MOOC "Hello (Real) World with ROS"
View Source
Parameters
ref_frame: 'robot1_base'
camera_topic: '/hrwros/logical_camera_1'
camera_frame: 'logical_camera_1_frame'
Required Autonomy Levels
continue: Off
failed: Off
Output Key Mapping
pose: part_pose
Apply
Close
Delete


Move Robot1 to pick
Type: MoveitToJointsDynState
Package: hrwros_factory_states
Uses MoveIt to plan and move the specified joints to the target configuration.
View Source
Parameters
move_group: pick_group
offset: 0.0
tool_link: gripper1
action_topic: '/move_group'
Required Autonomy Levels
reached: Off
planning_failed: Off
control_failed: Off
Input Key Mapping
joint_values: pick_configuration
joint_names: joint_names
Apply
Close
Delete


Activate gripper
Type: VacuumGripperControlState
Package: hrwros_factory_states
State to control any suction gripper in the factory simulation of the MOOC "Hello (Real) World with ROS"
View Source
Parameters
enable: True
service_name: '/gripper1/control'
Required Autonomy Levels
continue: Off
failed: Off
Apply
Close
Delete

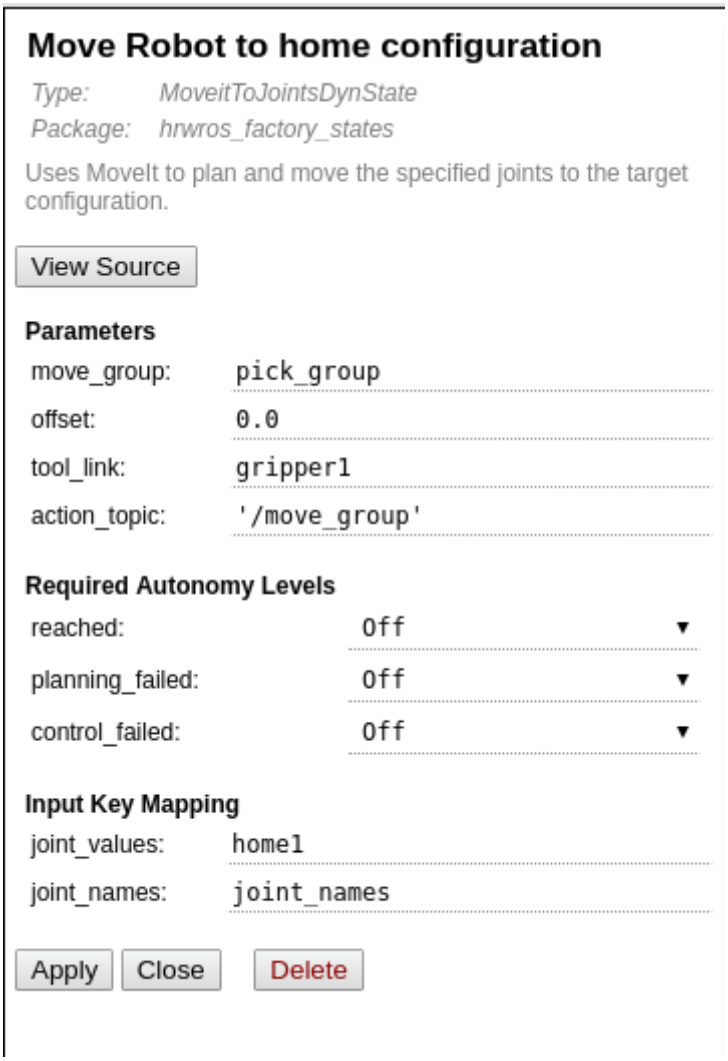


Figure A for Question 1

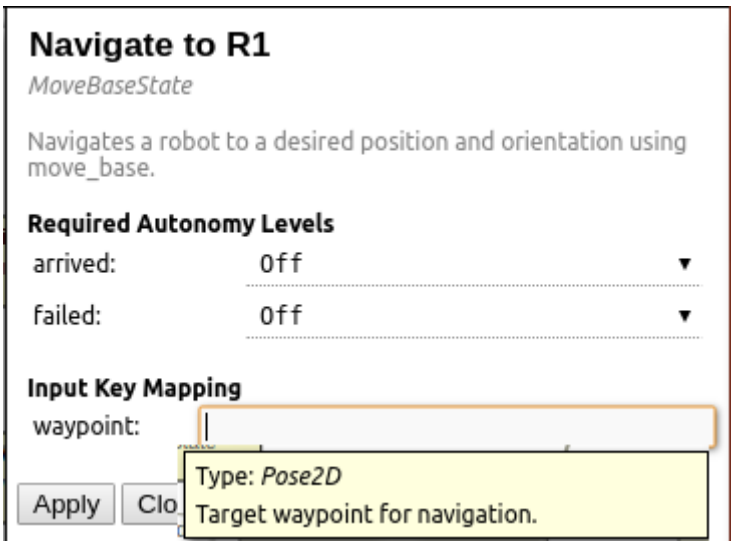
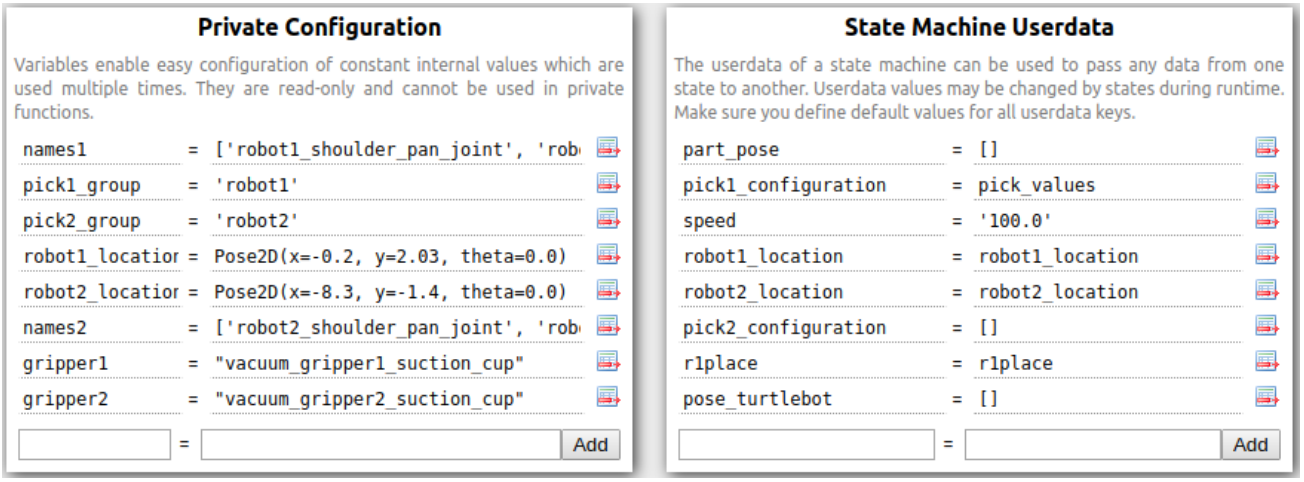


Figure B for Question 1



6.3.4 Question 1

1 point possible (ungraded)

What value should we set for the input key "waypoint" in Figure A (below), if this state should make the turtle navigate to the location of robot1, and considering the configuration of variables for the behavior in Figure B?

- robot1_location
- '100.0'
- pose_turtlebot
- robot2_location

Submit