AI ROBOTICS FOR ALL

AIAT ARTIFICIAL INTELLIGENCE ASSOCIATION OF THAILAND

KU KASETSART UNIVERSITY

TRS THAI ROBOTICS SOCIETY

Slide

https://kasets.art/6f6ly0

Robot Manipulation

Noppanut Thongton

Robot Manipulation

REF: https://www.red-dot.org/project/kuka-kr-quantec-210-r2700-45875

Kinematics

Kinematics for robot manipulation.

4

Kinematics

- แปลเป็นภาษาไทยคือ จลนศาสตร์
- เป็นศาสตร์ของการเคลื่อนไหว ทำการศึกษากับวัตถุที่เคลื่อนไหวโดยที่ไม่คำถึงถึงแรงที่ก่อให้เกิดการเคลื่อนไหวนั้น
- โดยเป็นการศึกษาตำแหน่ง ความเร็ว ความเร่ง และพจน์ที่เป็น higher order derivative ของตำแหน่งตามเวลา

Kinematics

- โดยการศึกษา kinematics ของ manipulation นั้นเป็นการอ้างถึงเรขาคณิต และ คุณสมบัติตามเวลาของการเคลื่อนไหว

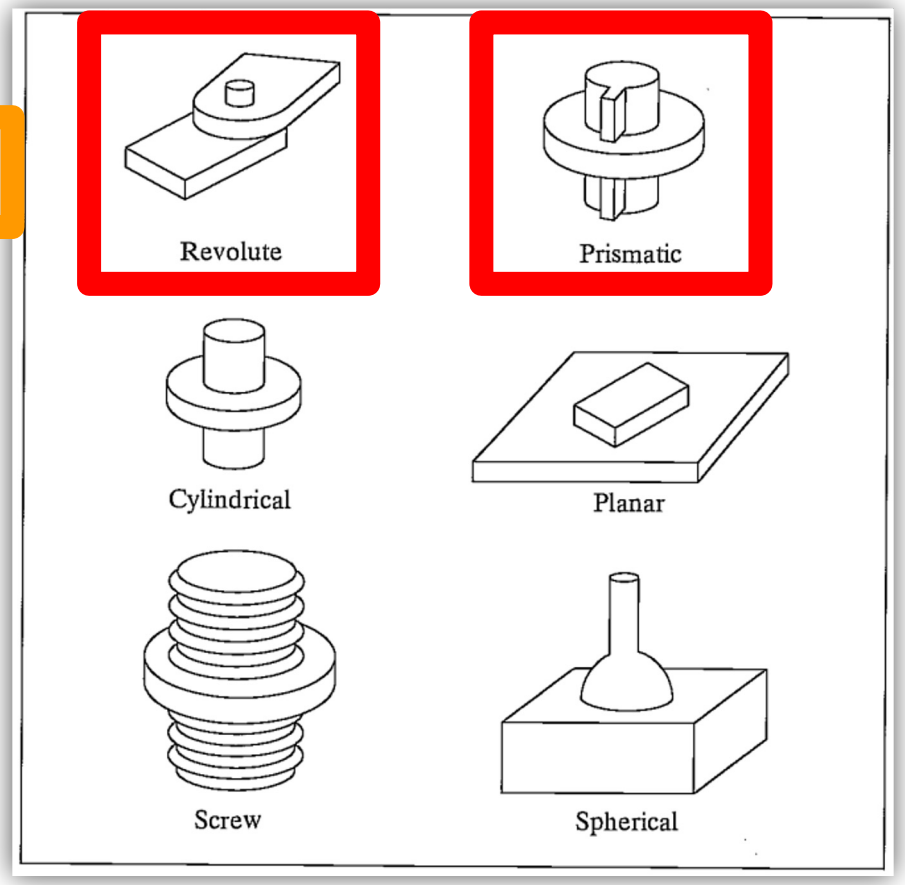
Link description

The description of Link.

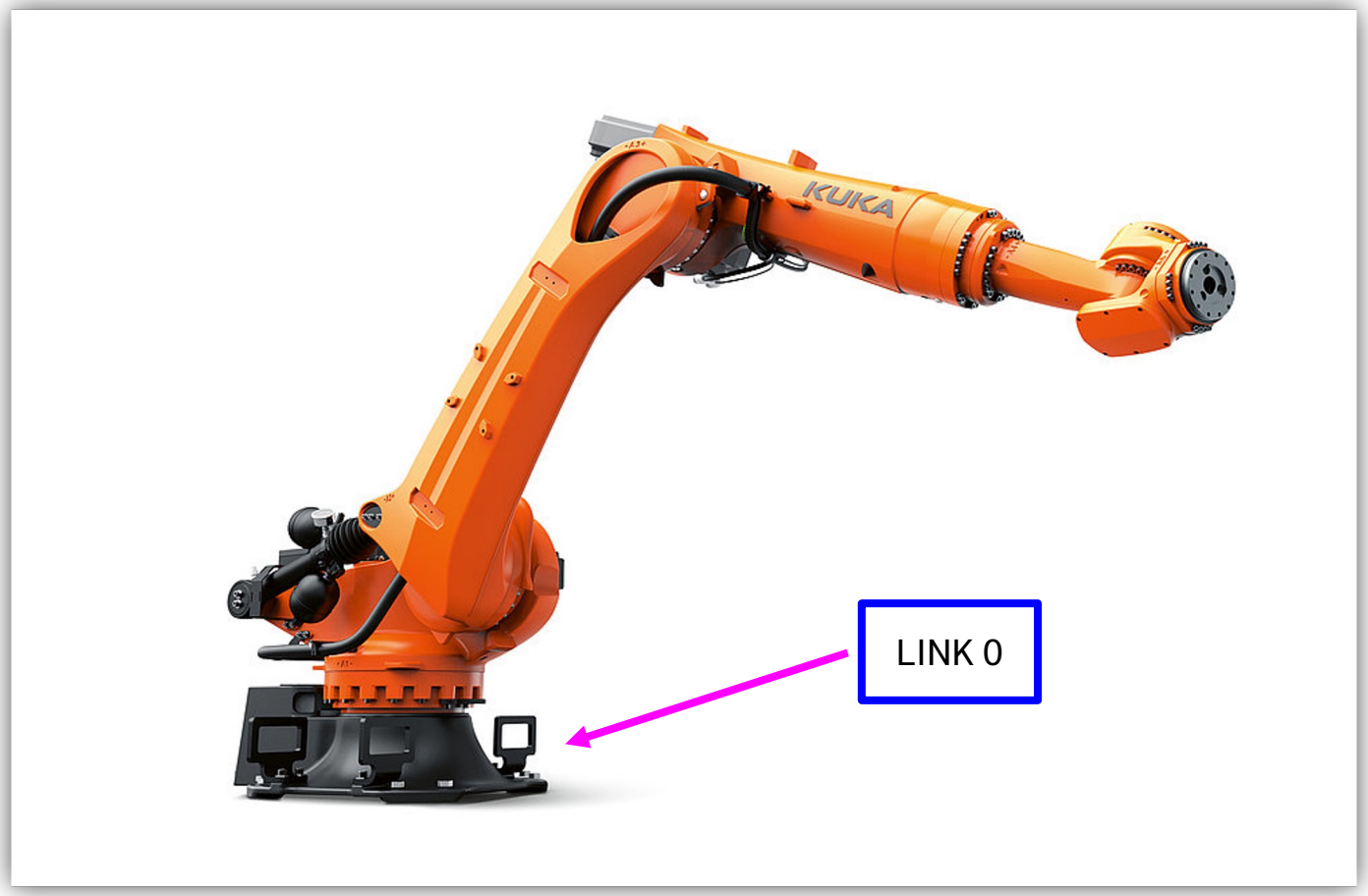
7

Link description

Popular

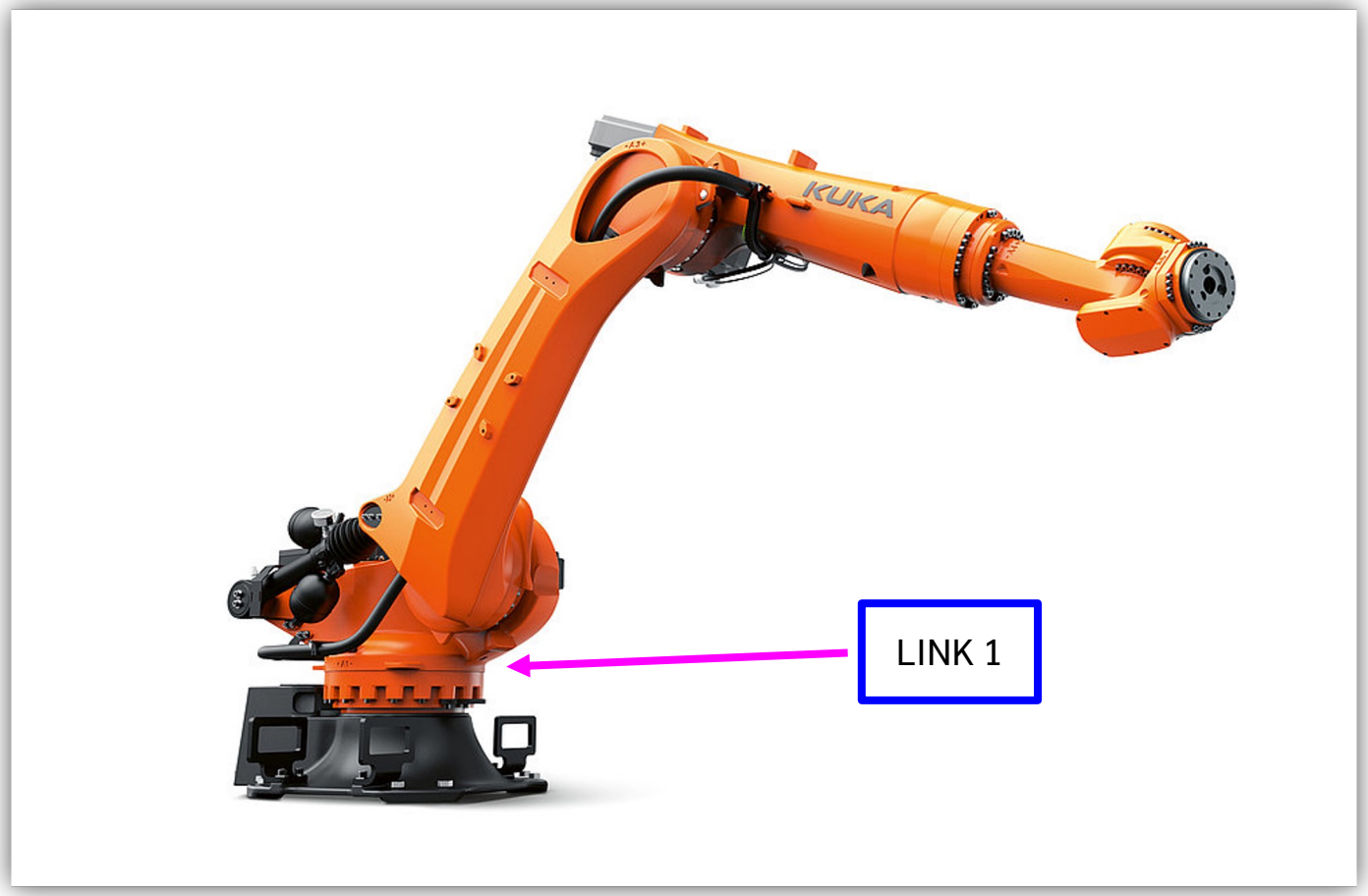


Link description



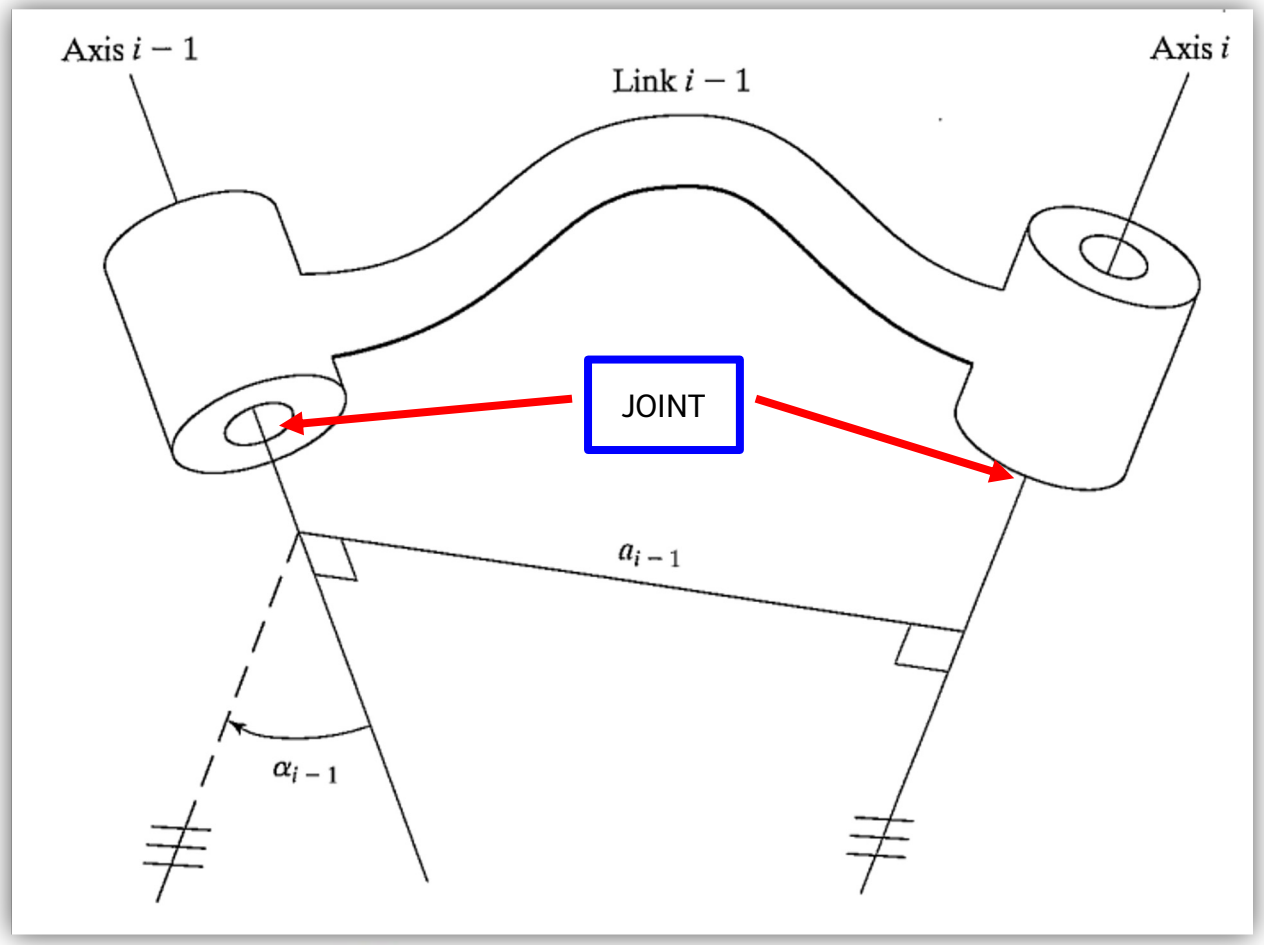
REF: https://www.red-dot.org/project/kuka-kr-quantec-210-r2700-45875

Link description



REF: https://www.red-dot.org/project/kuka-kr-quantec-210-r2700-45875

Link description

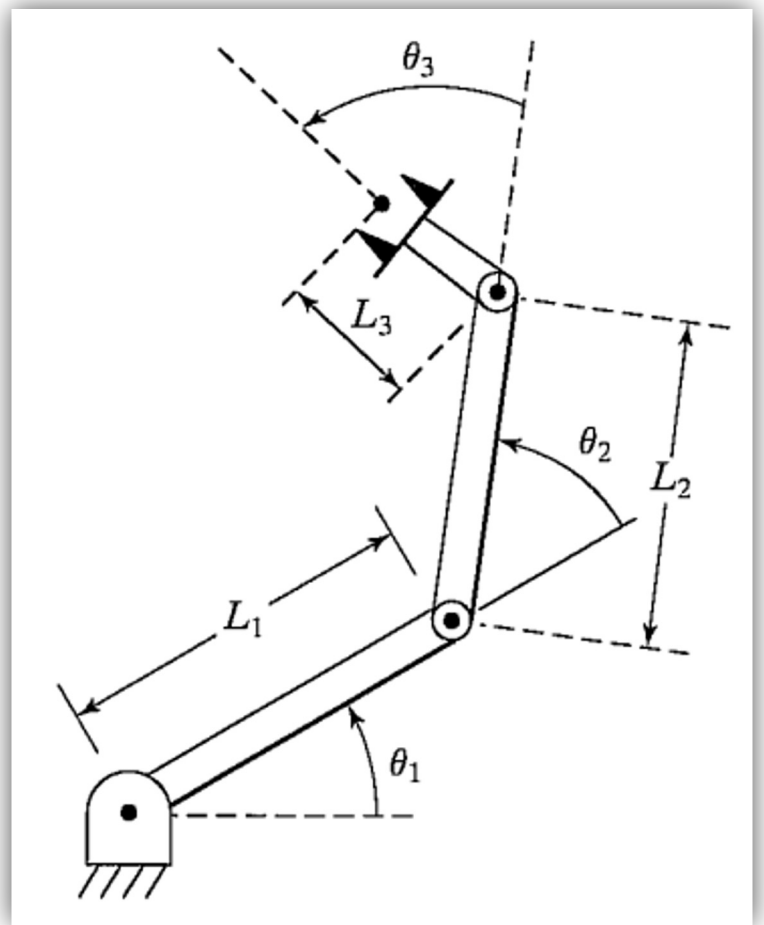


Forward kinematics

Forward kinematics

12

Forward kinematics

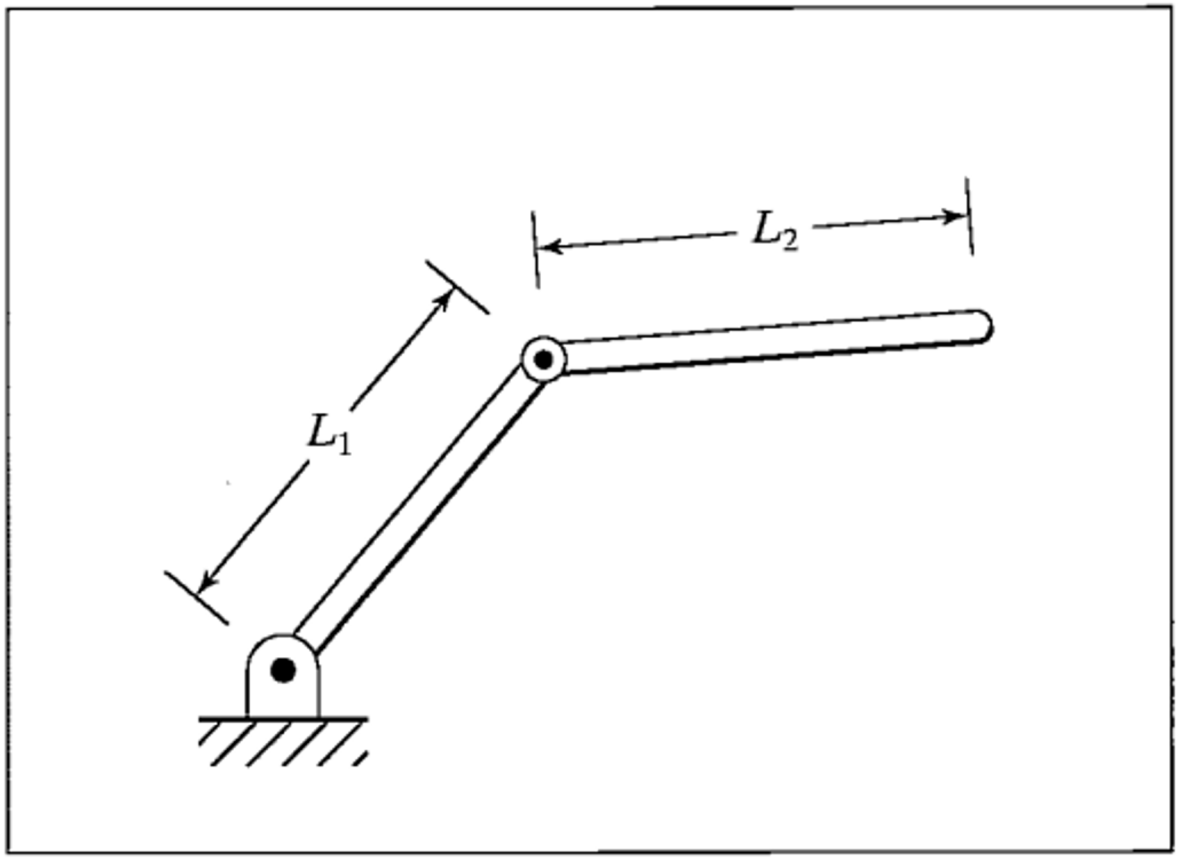


Inverse kinematics

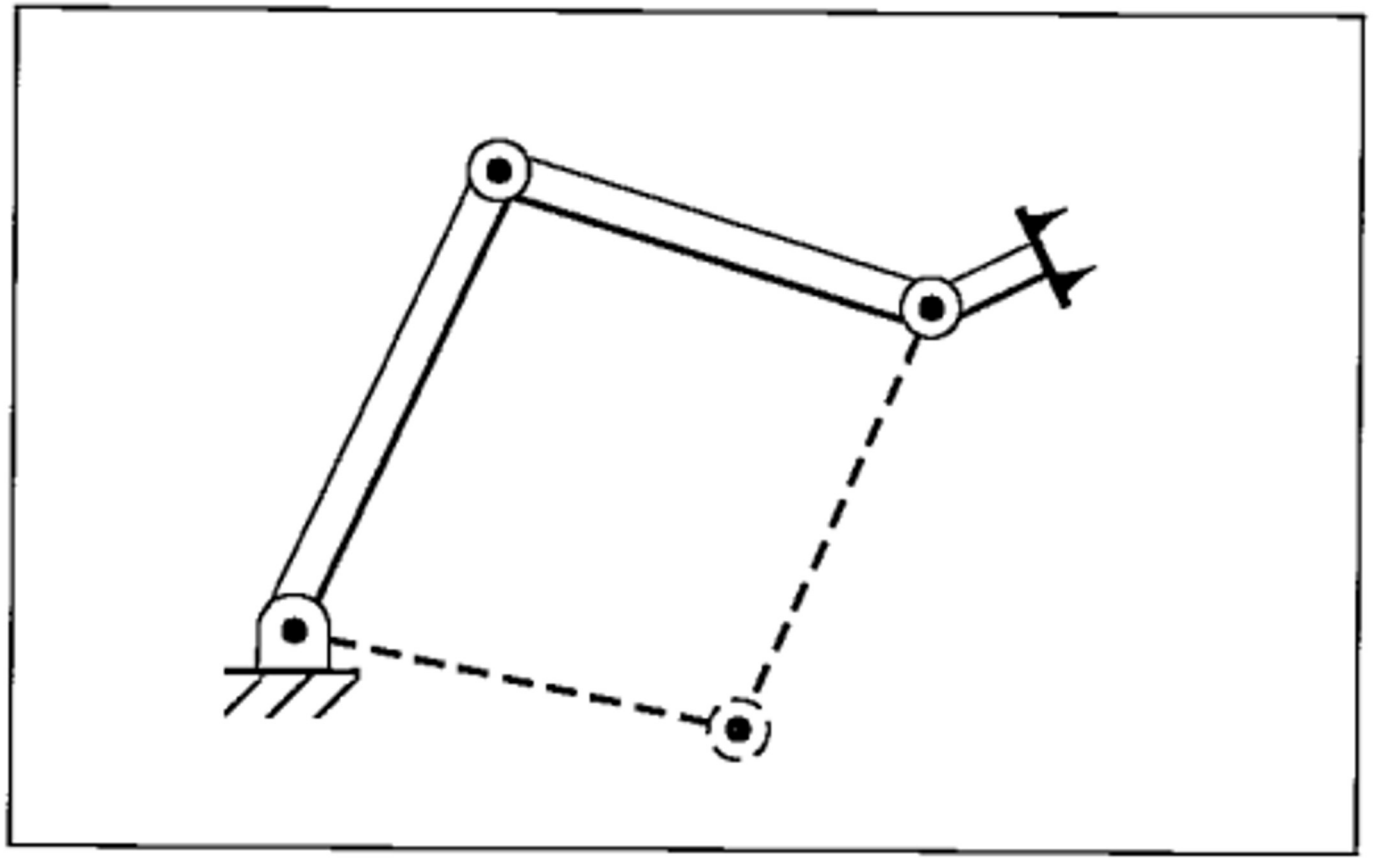
Inverse kinematics

14

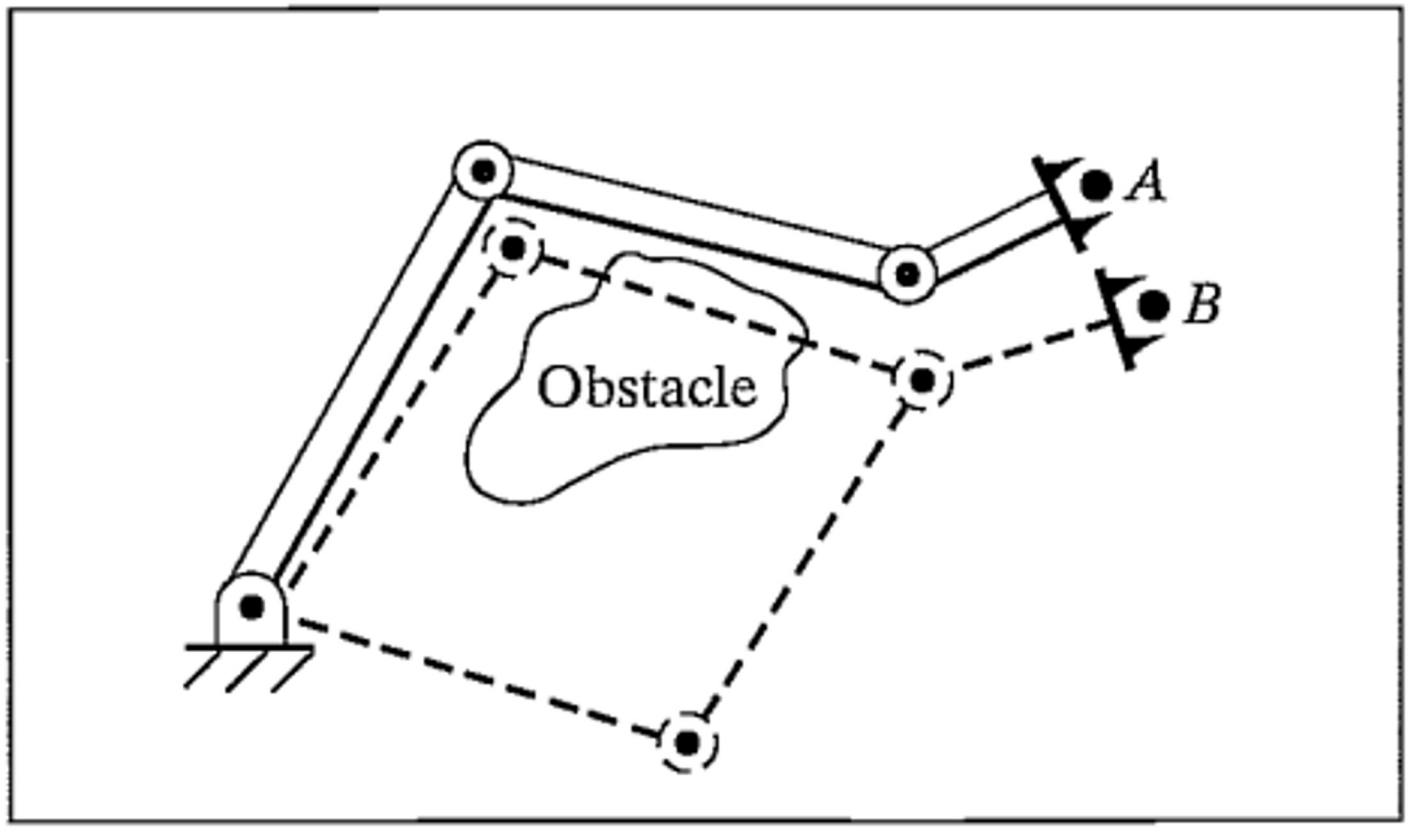
Inverse kinematics



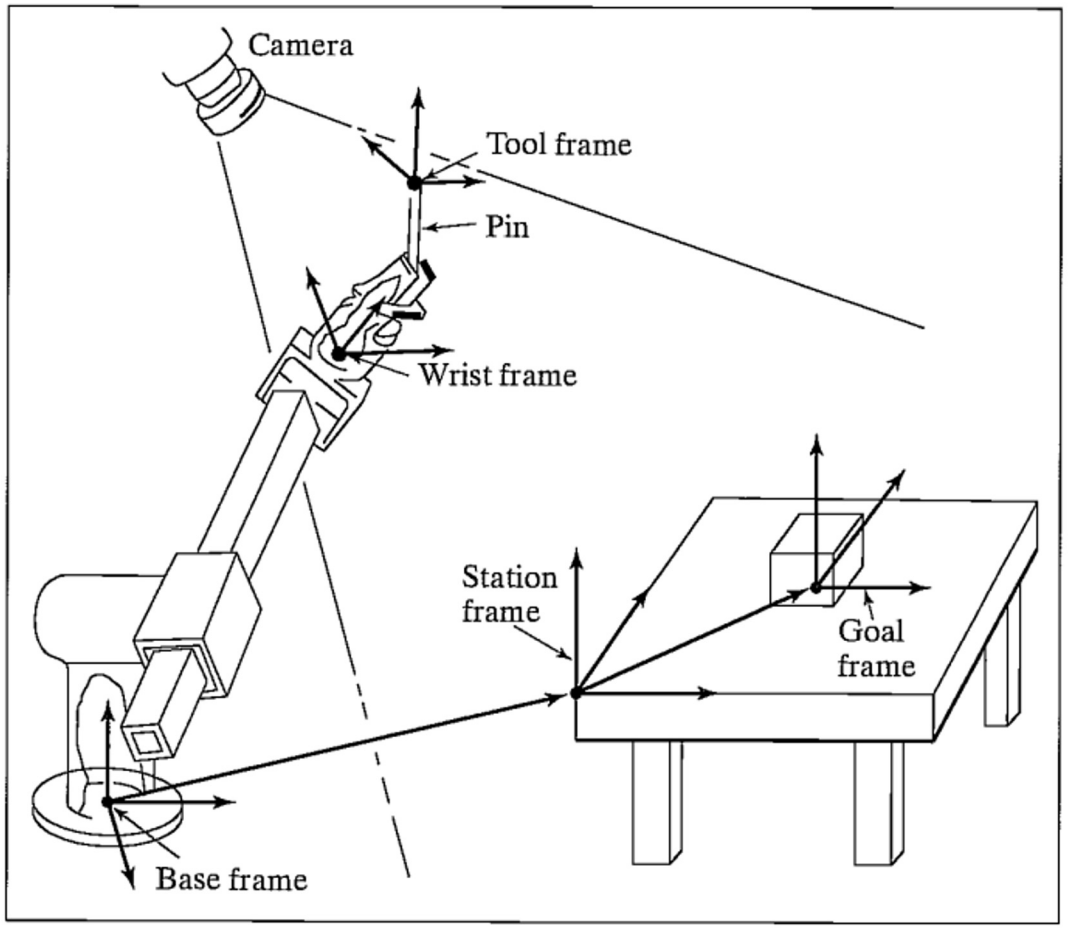
Inverse kinematics

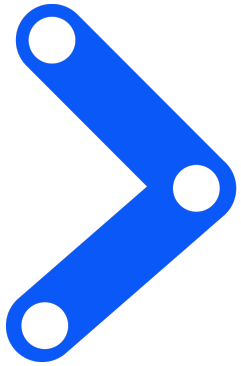


Inverse kinematics



Inverse kinematics





MoveIt

MoveIt! Installation

Install MoveIt!

19

MoveIt! Installation

```
$ rosdep update
```

MoveIt! Installation

```
$ sudo apt update
```

MoveIt! Installation

```
$ sudo apt install
```

MoveIt! Installation

```
$ sudo apt install ros-noetic-moveit
```

MoveIt! Installation

```
$ git clone https://github.com/ros-planning/moveit_tutorials.git -b master
```

MoveIt! Installation

```
$ git clone https://github.com/ros-planning/panda_moveit_config.git -b noetic-devel
```

MoveIt! Installation

```
$ rosdep install -y --from-paths . --ignore-src --rosdistro
noetic
```

MoveIt! Installation

```
$ sudo apt remove ros-noetic-moveit
```

https://ros-planning.github.io/moveit_tutorials/doc/getting_started/getting_started.html

MoveIt! Installation

```
$ cd <workspace>
```

MoveIt! Installation

```
$ catkin_make
```

MoveIt! Installation

```
$ rospack profile
```

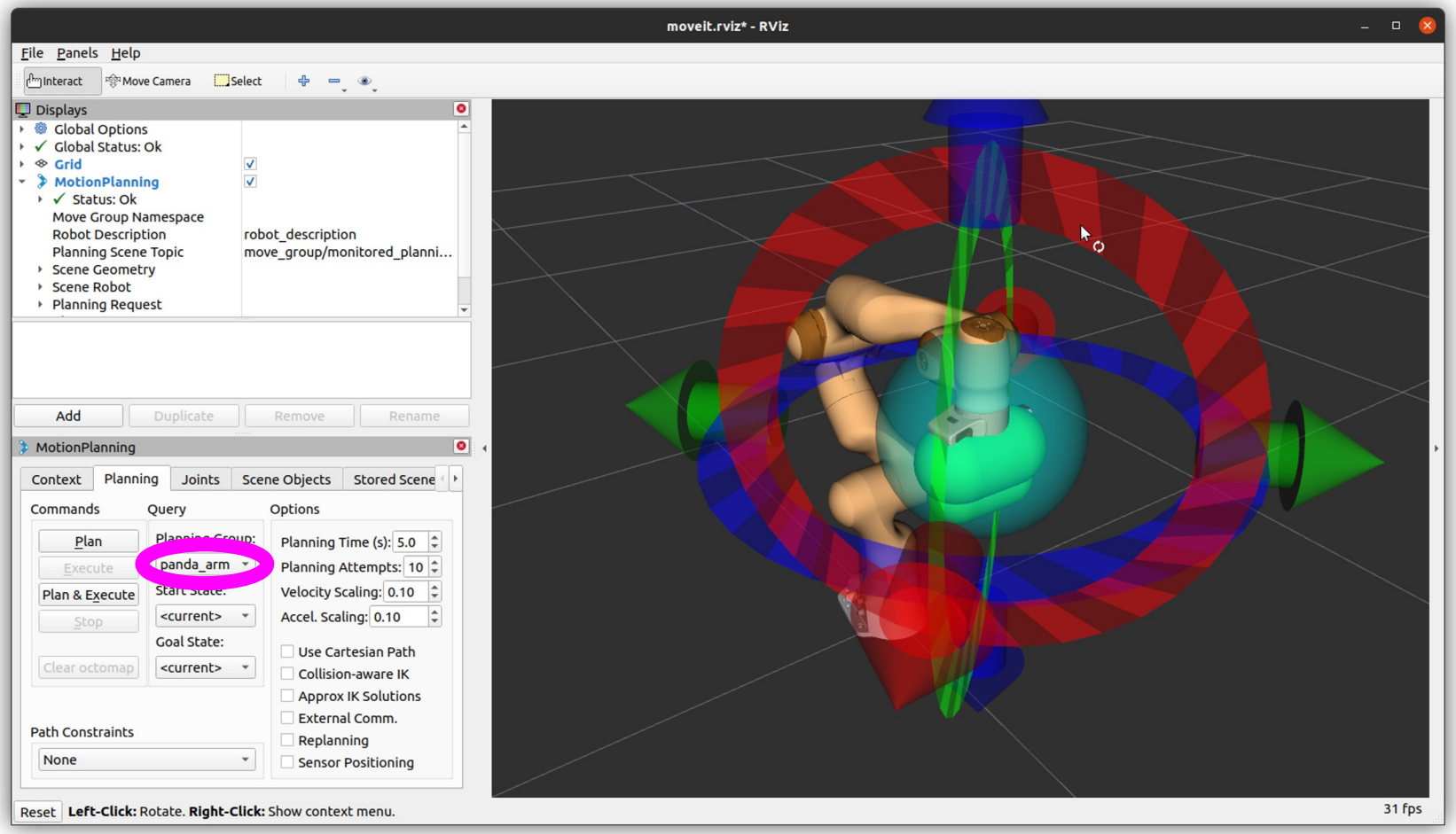
Panda arm demo

Using Moveit with Panda arm.

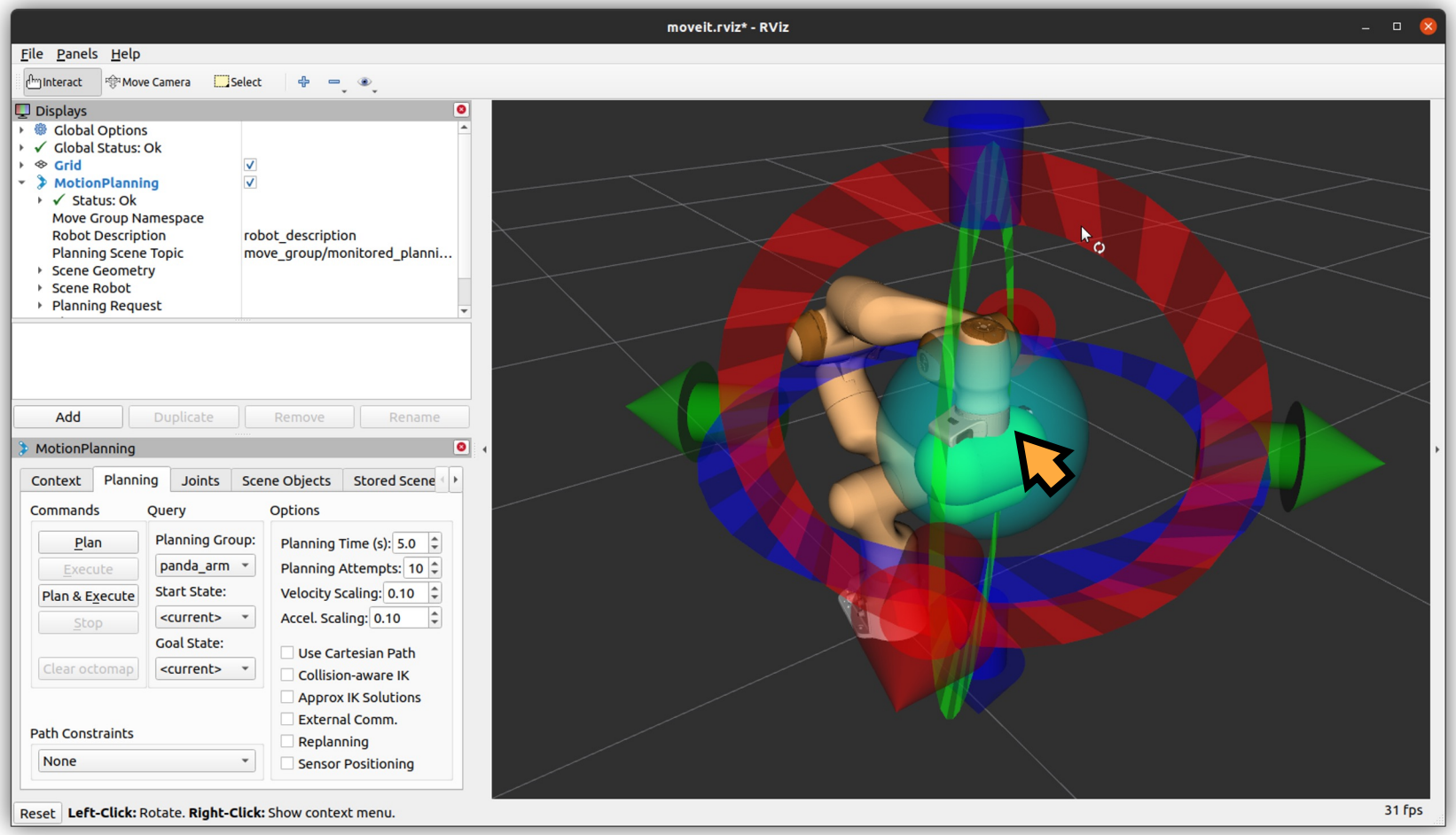
Panda arm demo

```
$ roslaunch panda_moveit_config demo.launch
```

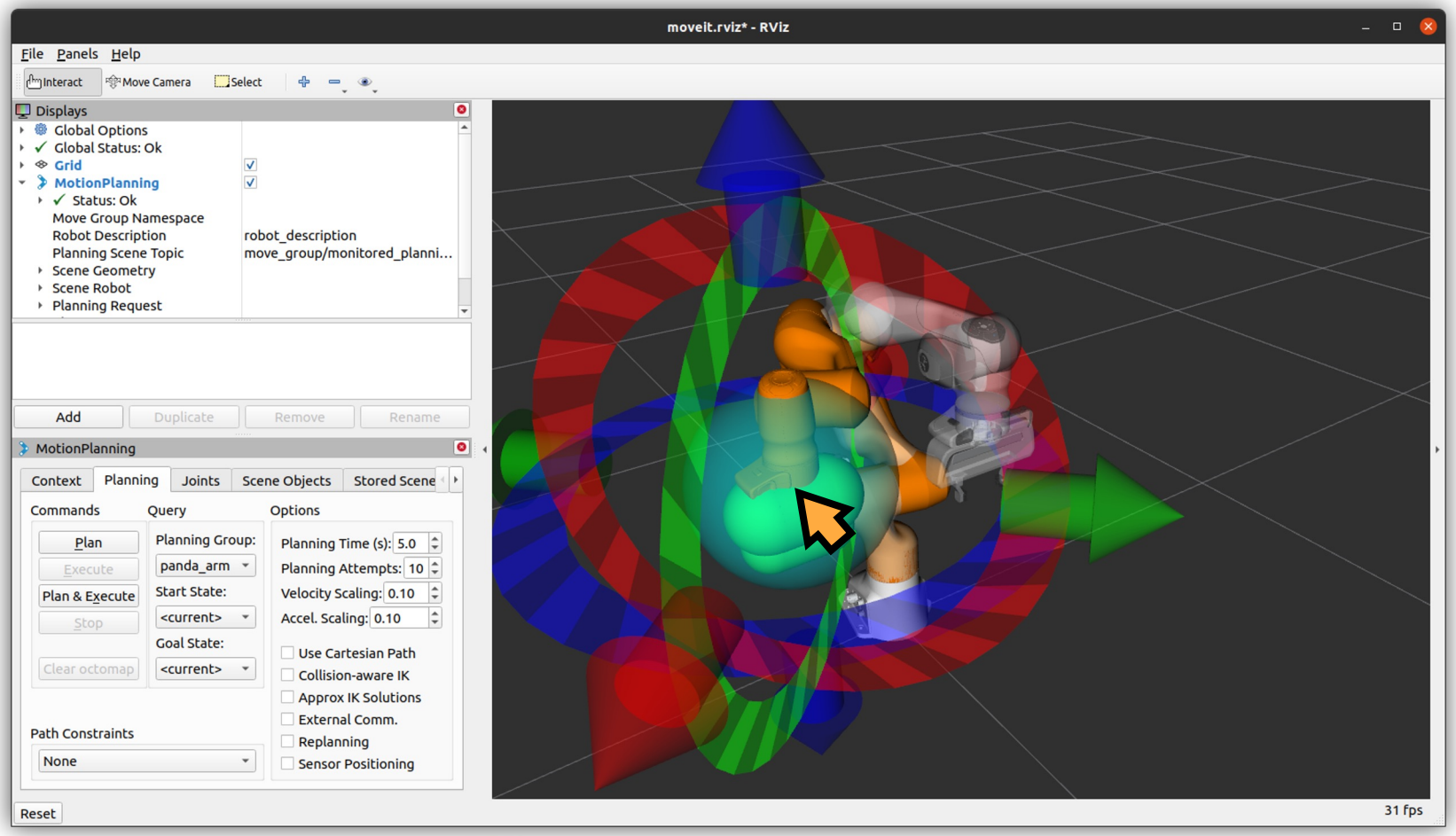
Panda arm demo



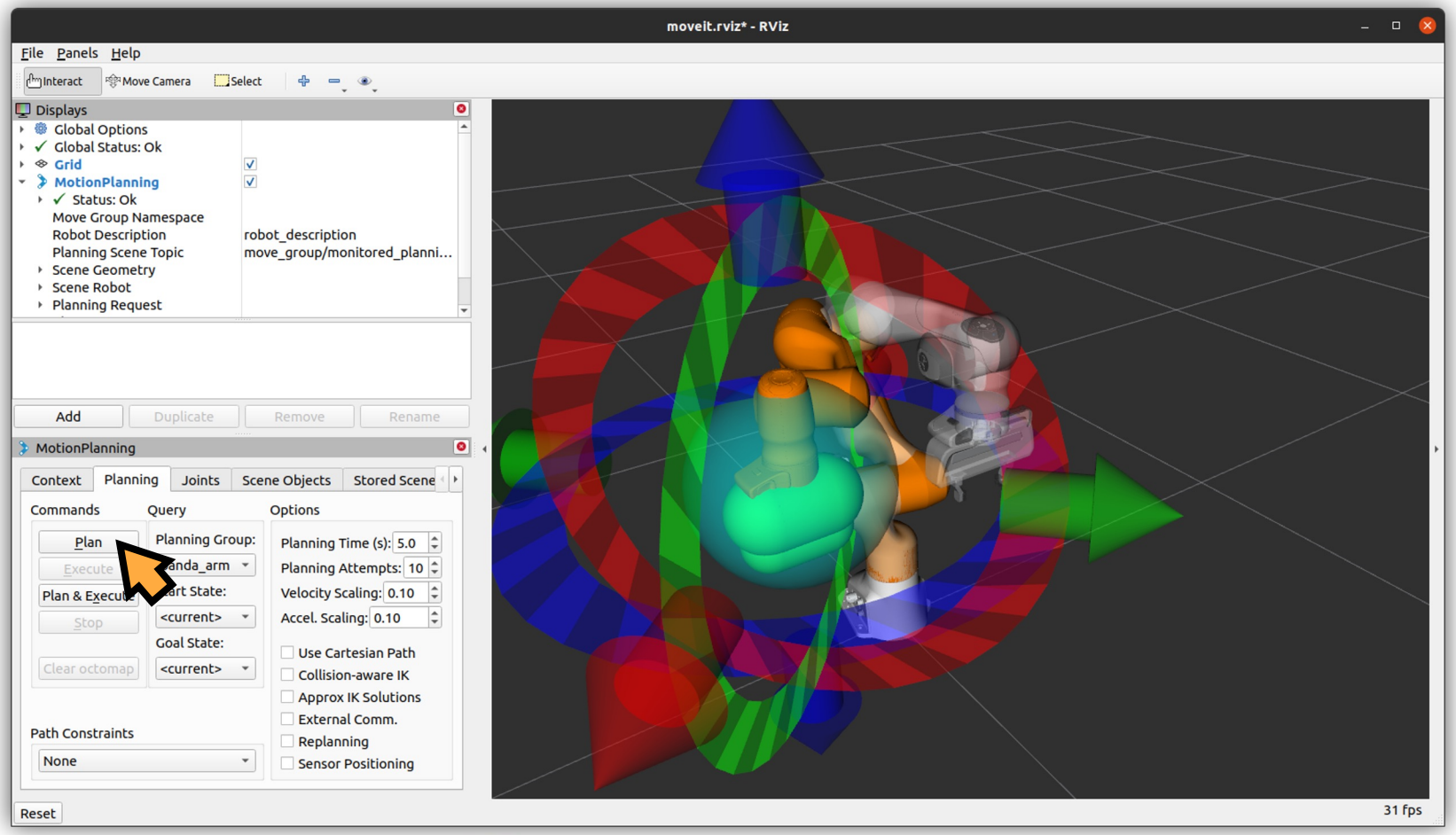
Panda arm demo



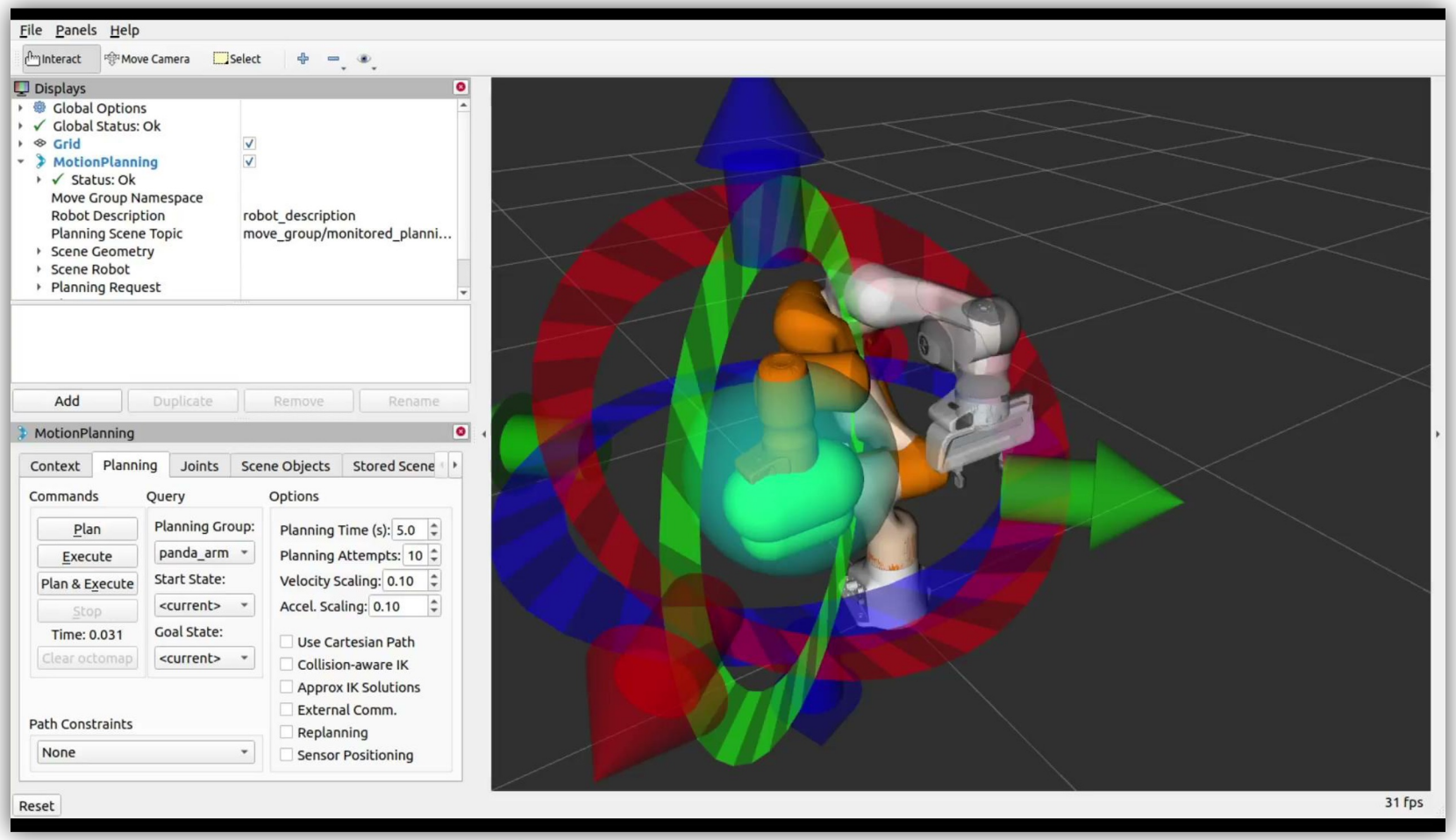
Panda arm demo



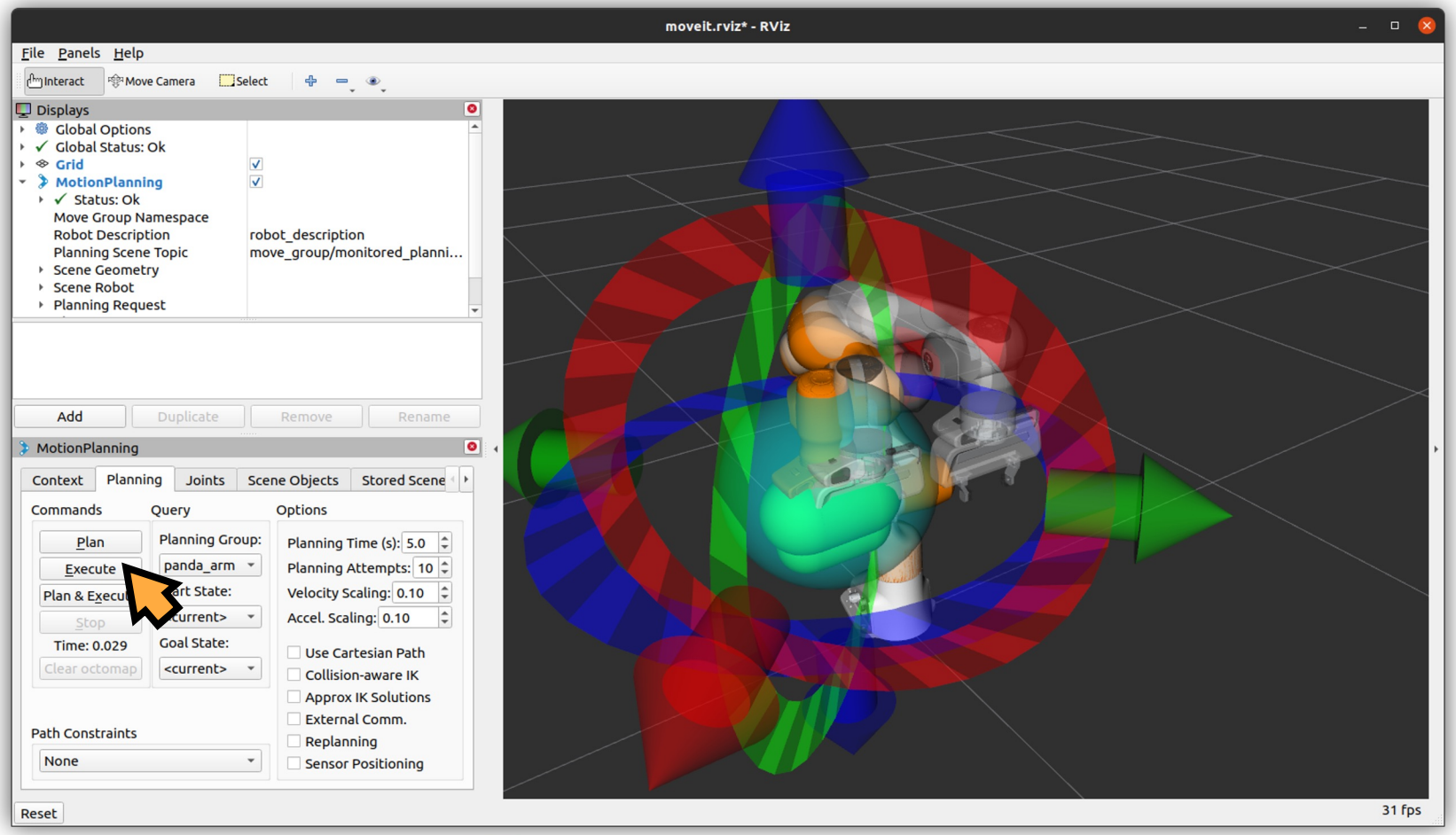
Panda arm demo



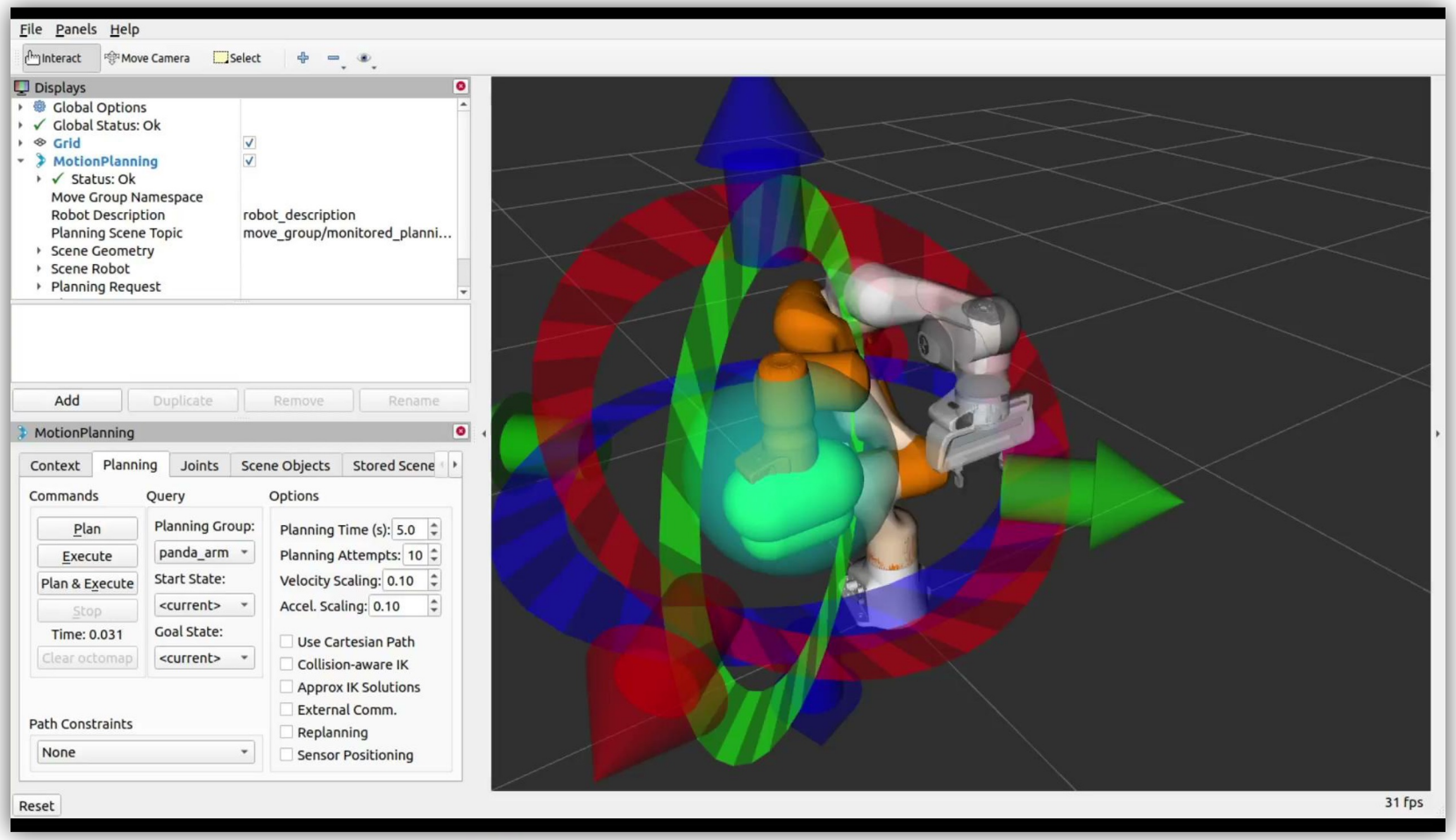
Panda arm demo



Panda arm demo



Panda arm demo



MoveIt Setup Assistant

Tools for help you setup your arm in Moveit!

40

MoveIt Setup Assistant

```
$ cd tutorial_ws/src/
```

MoveIt Setup Assistant

```
$ git clone https://github.com/ros-industrial/kuka_experimental.git
```

MoveIt Setup Assistant

```
$ cd ..
```

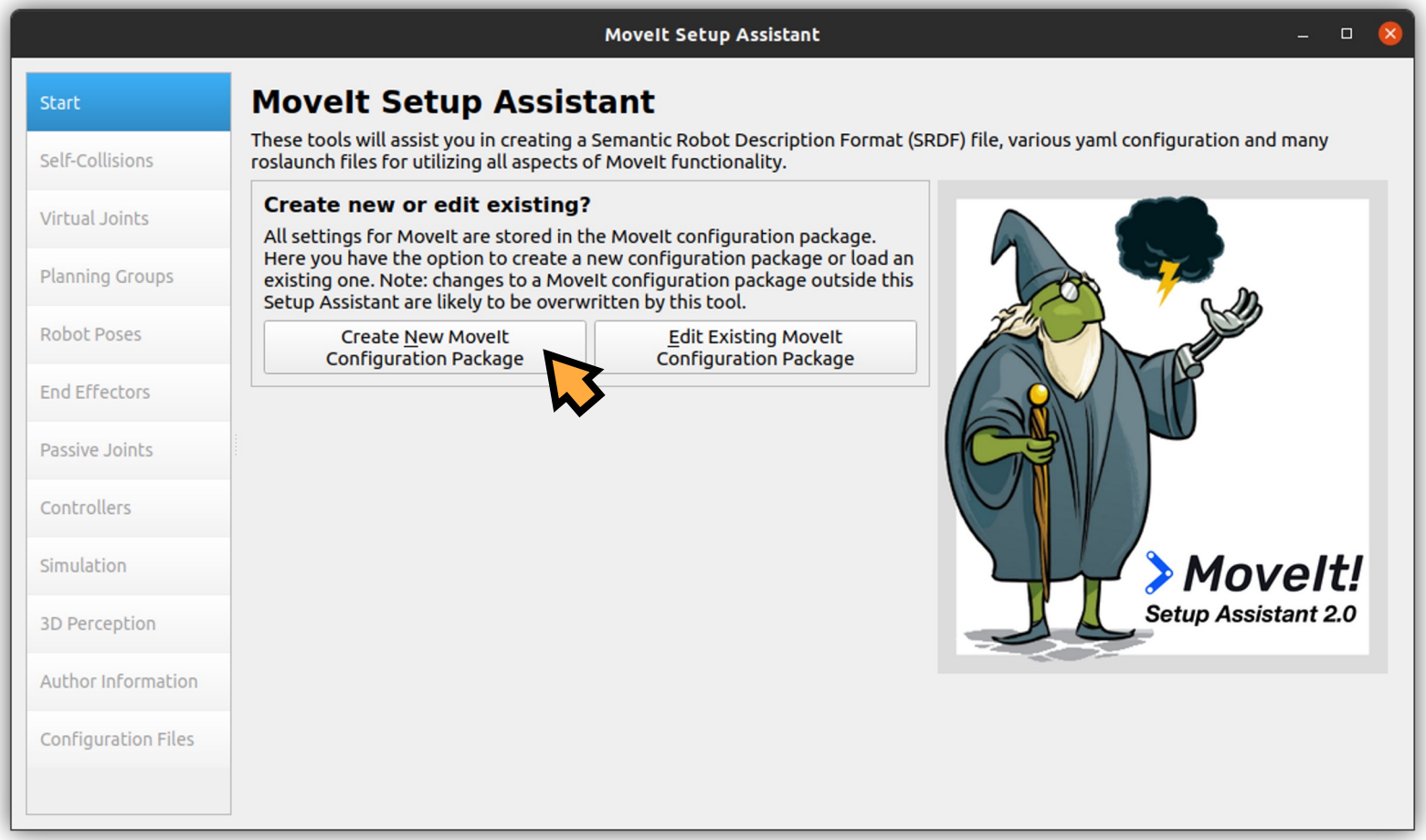
MoveIt Setup Assistant

```
$ catkin_make
```

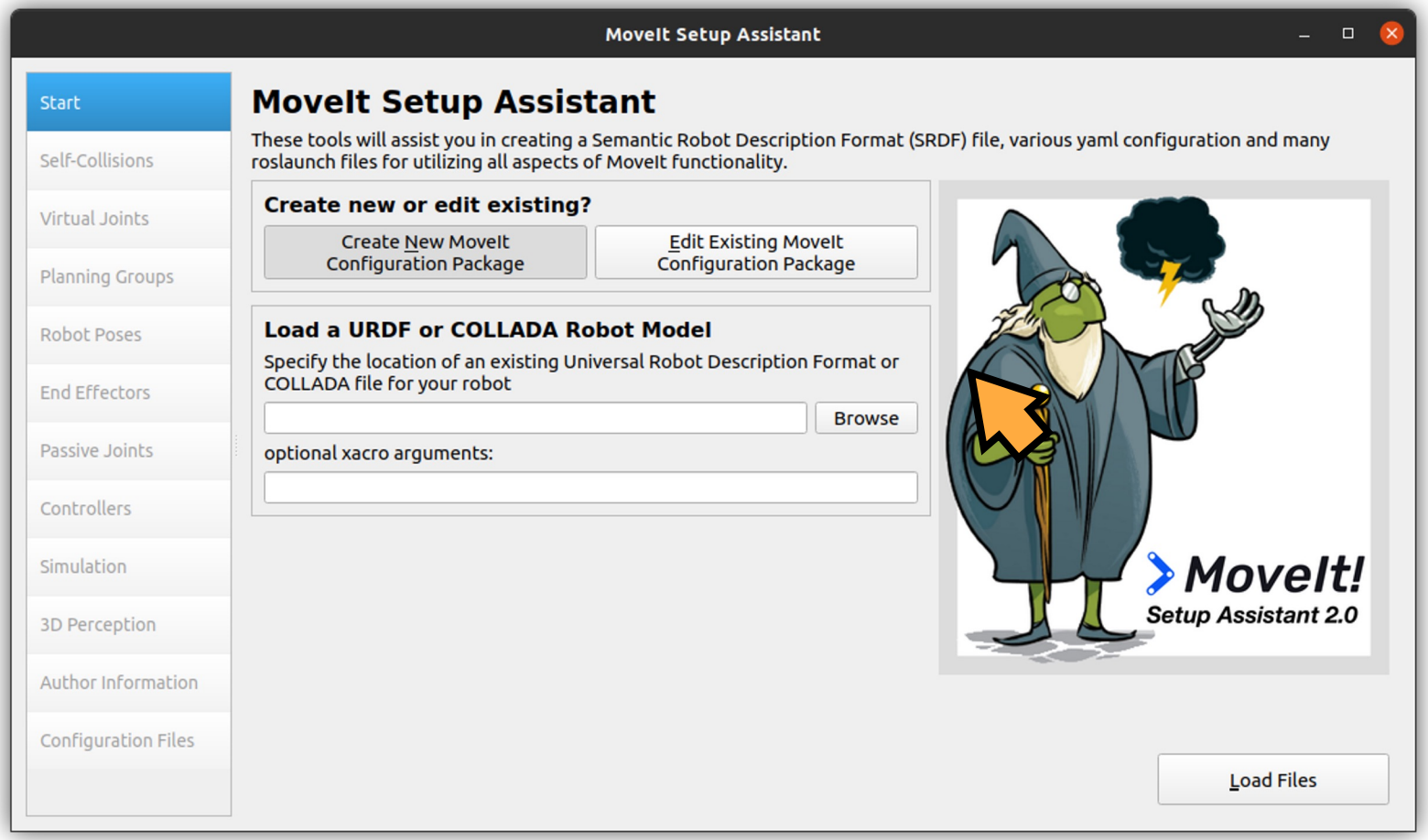
MoveIt Setup Assistant

```
$ roslaunch moveit_setup_assistant setup_assistant.launch
```

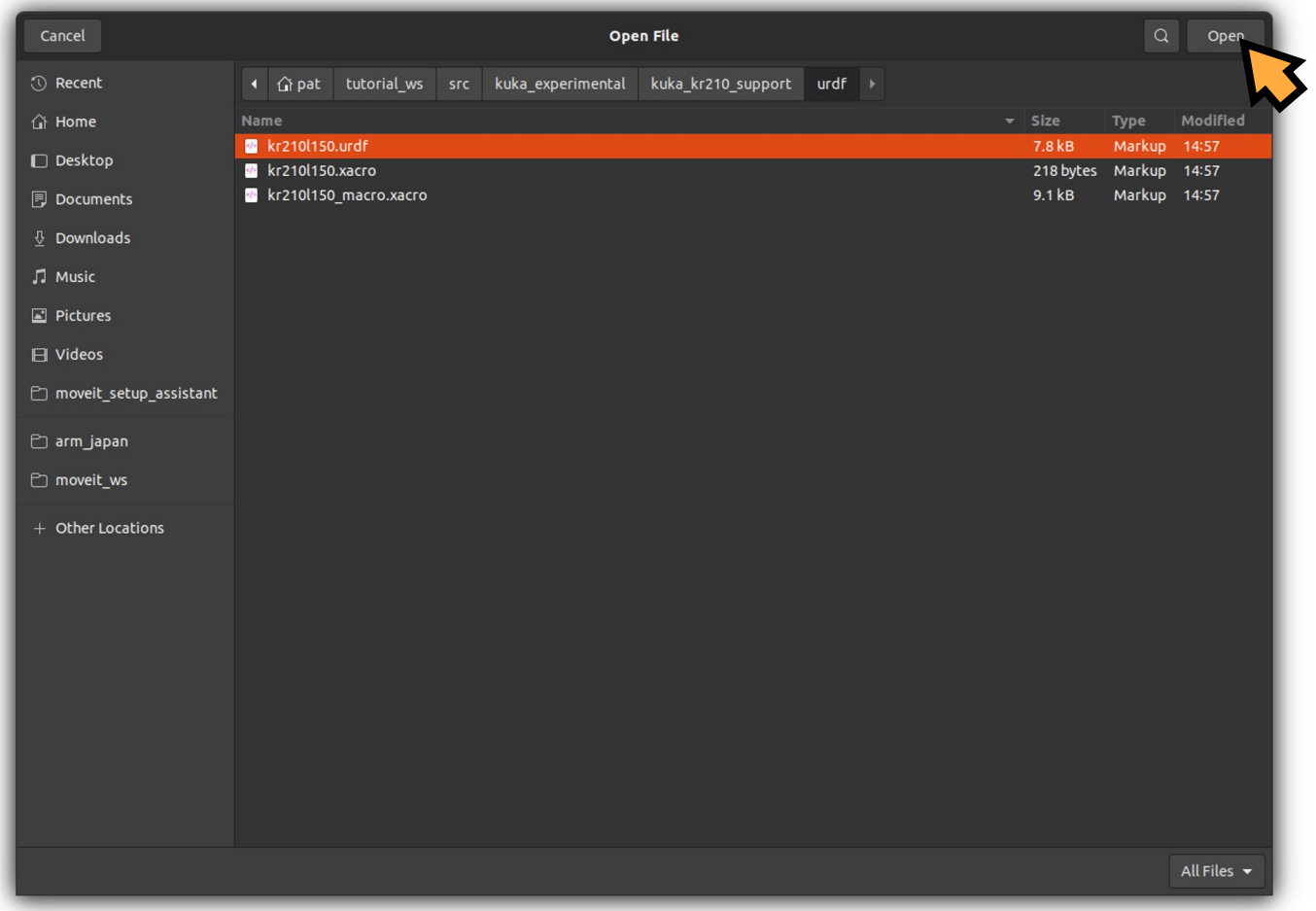
MoveIt Setup Assistant



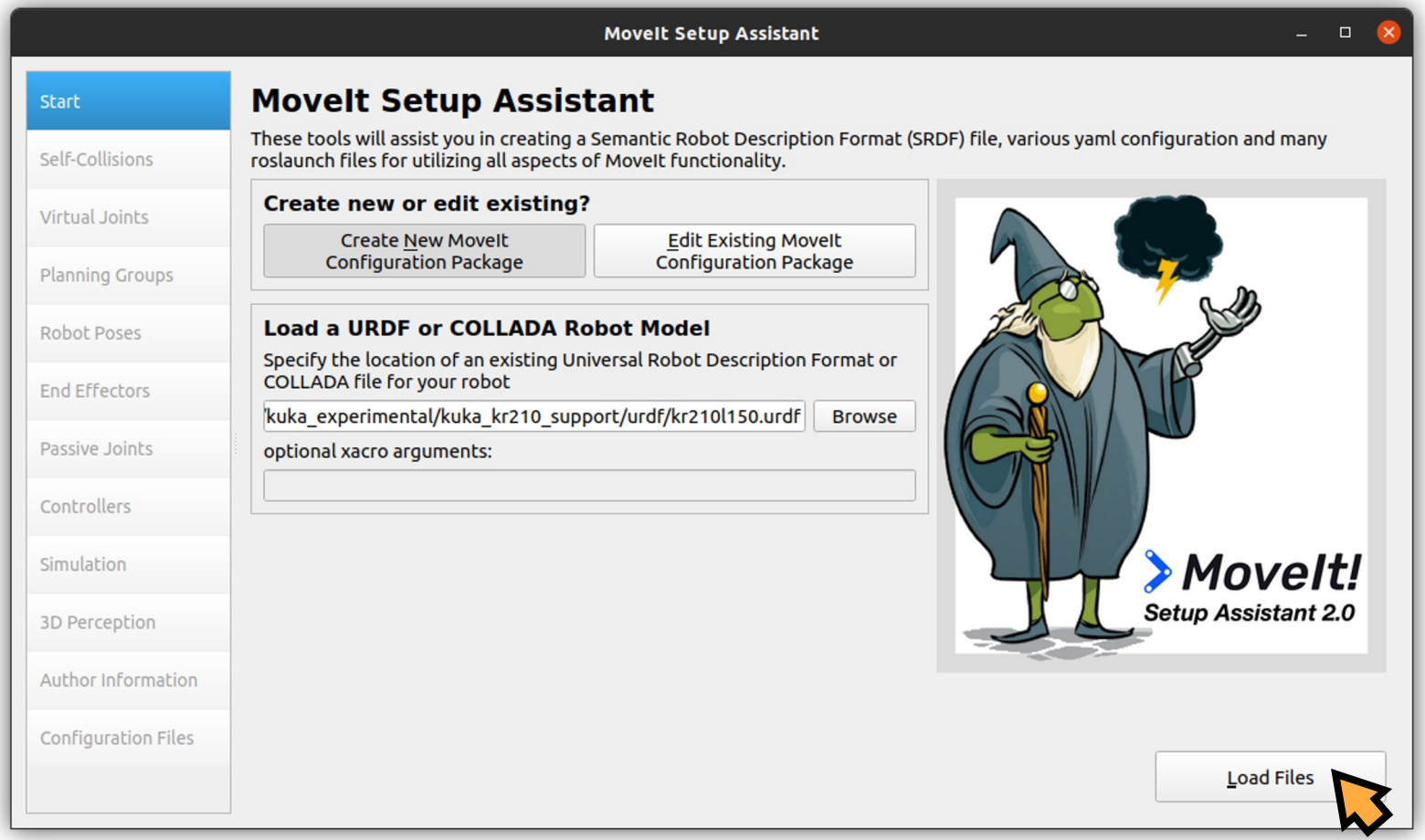
MoveIt Setup Assistant



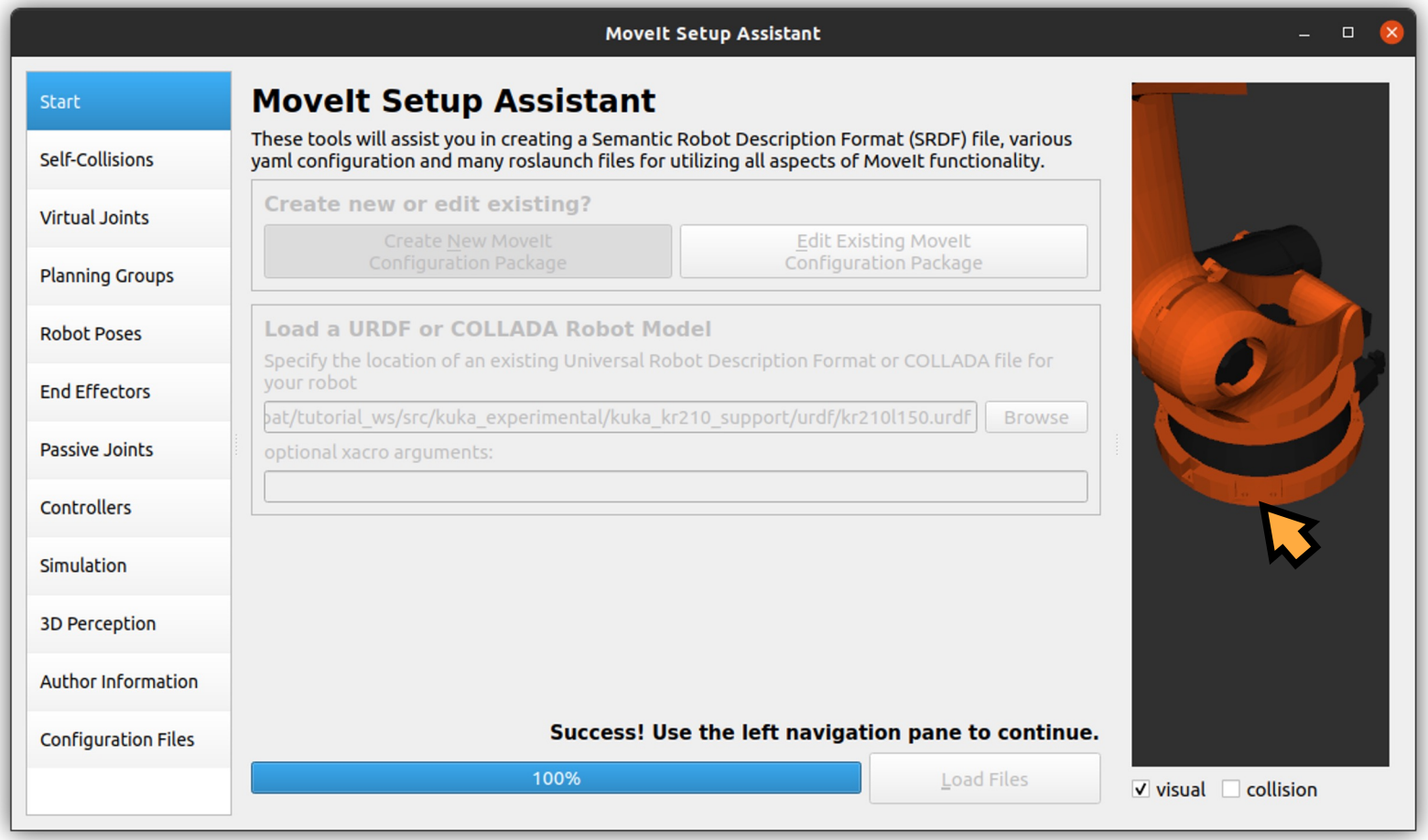
MoveIt Setup Assistant



MoveIt Setup Assistant

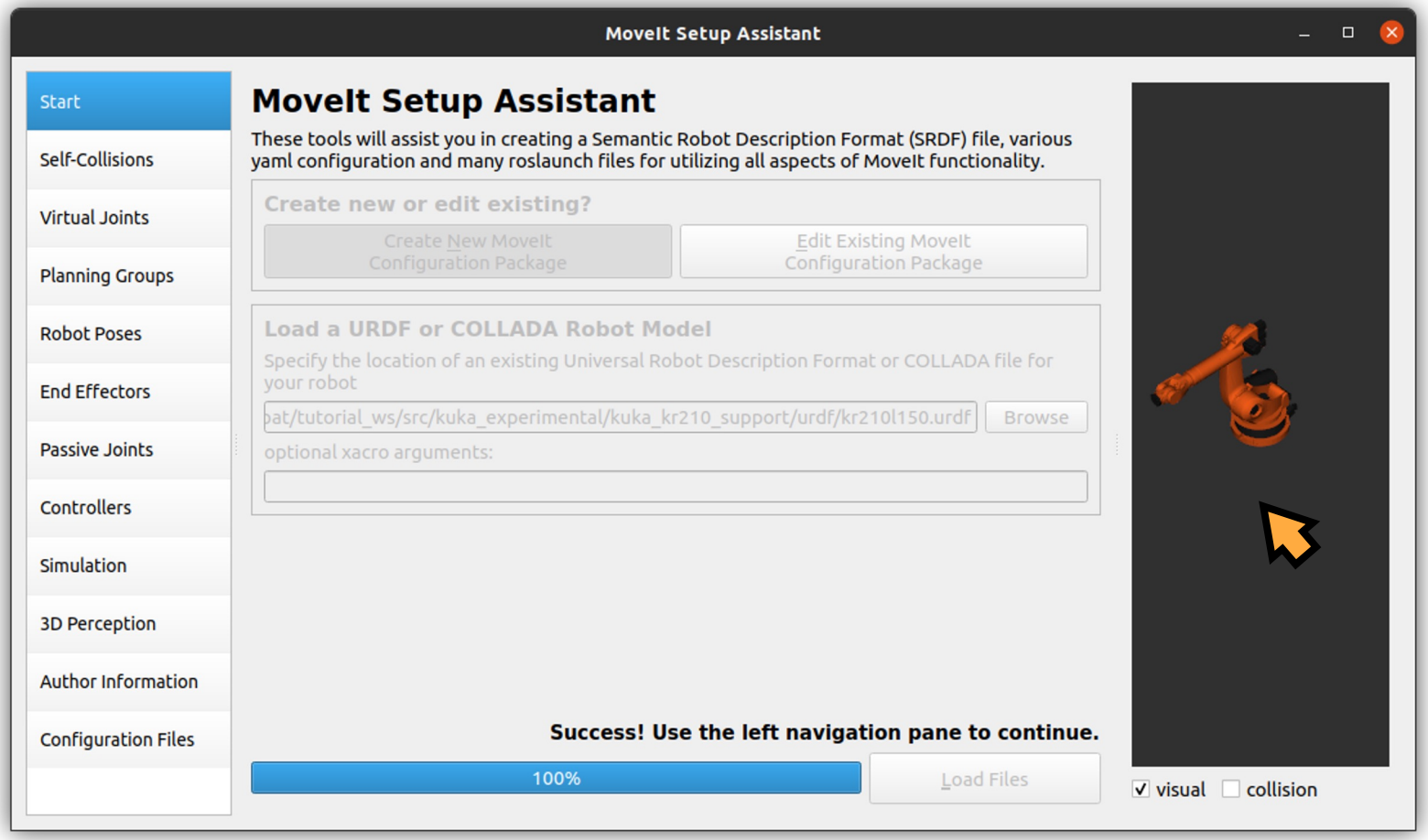


MoveIt Setup Assistant

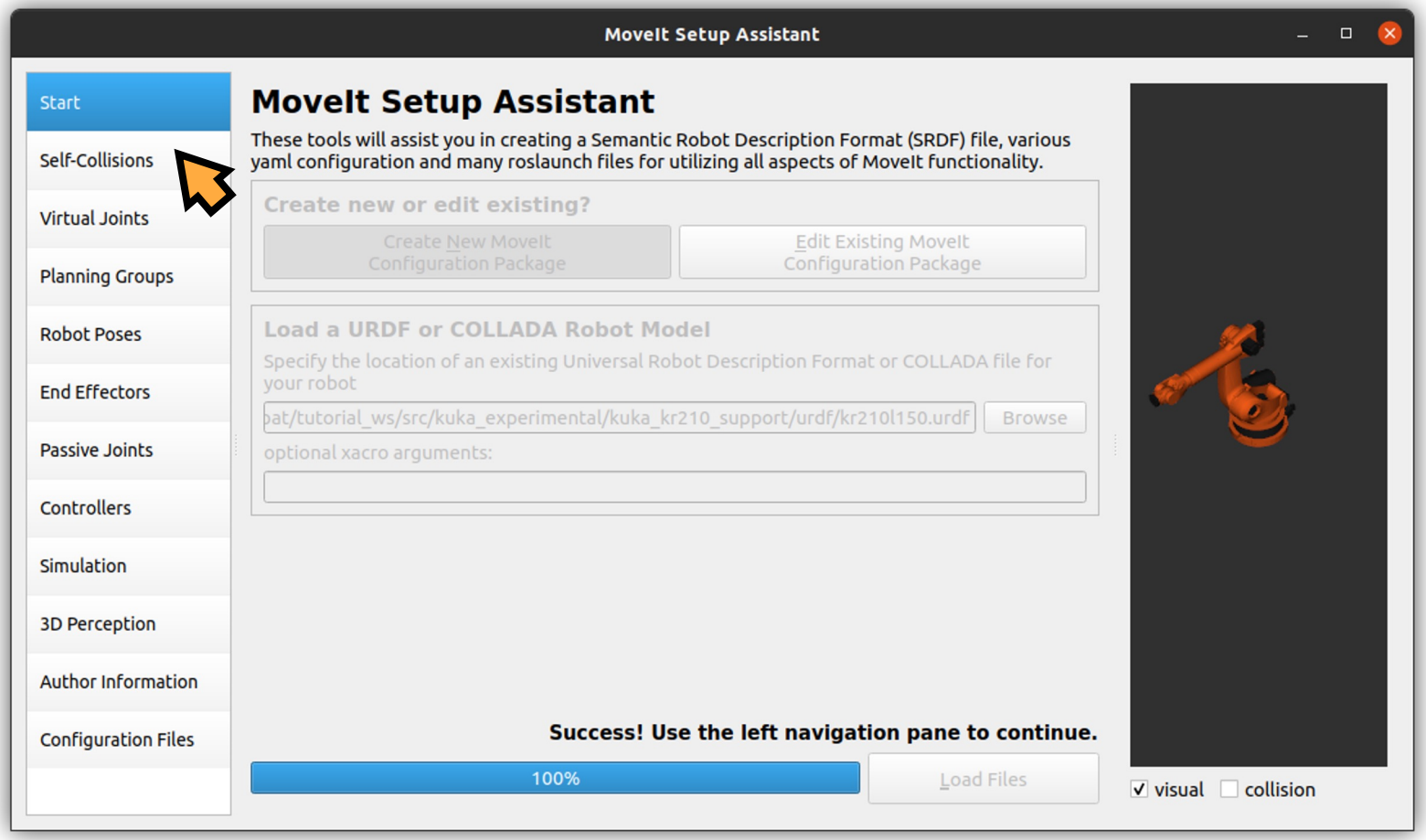


MoveIt Setup Assistant

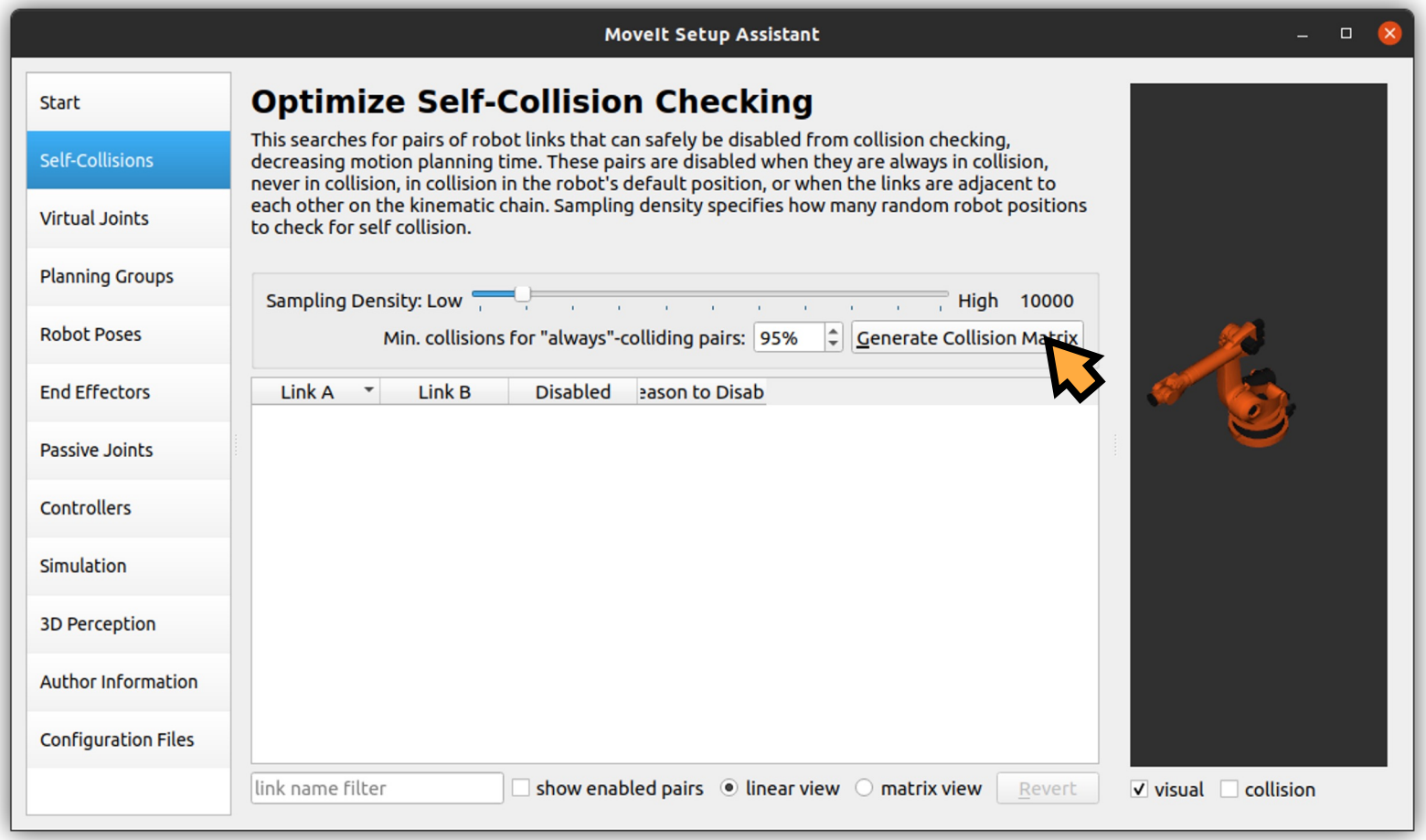
MoveIt Setup Assistant



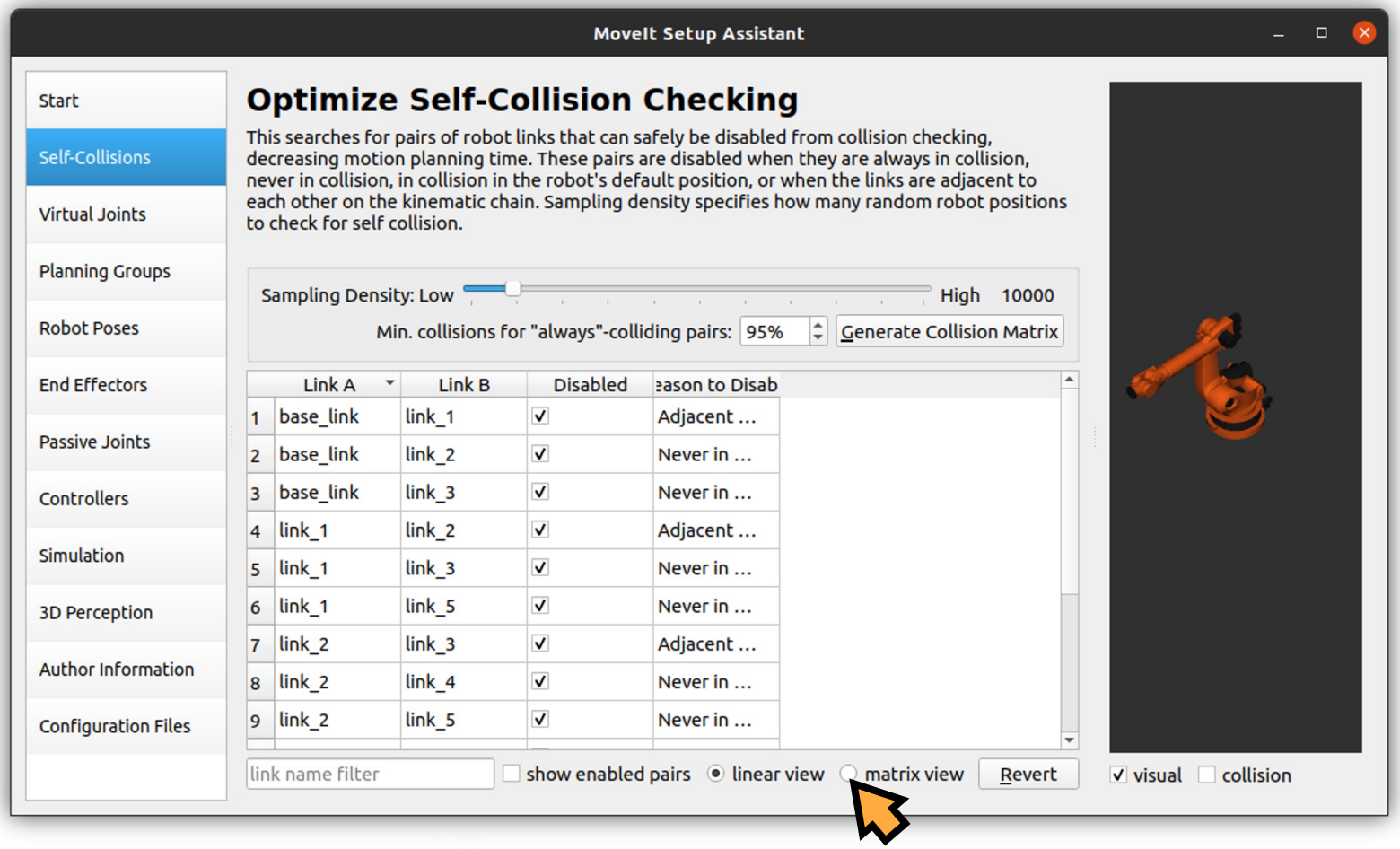
MoveIt Setup Assistant



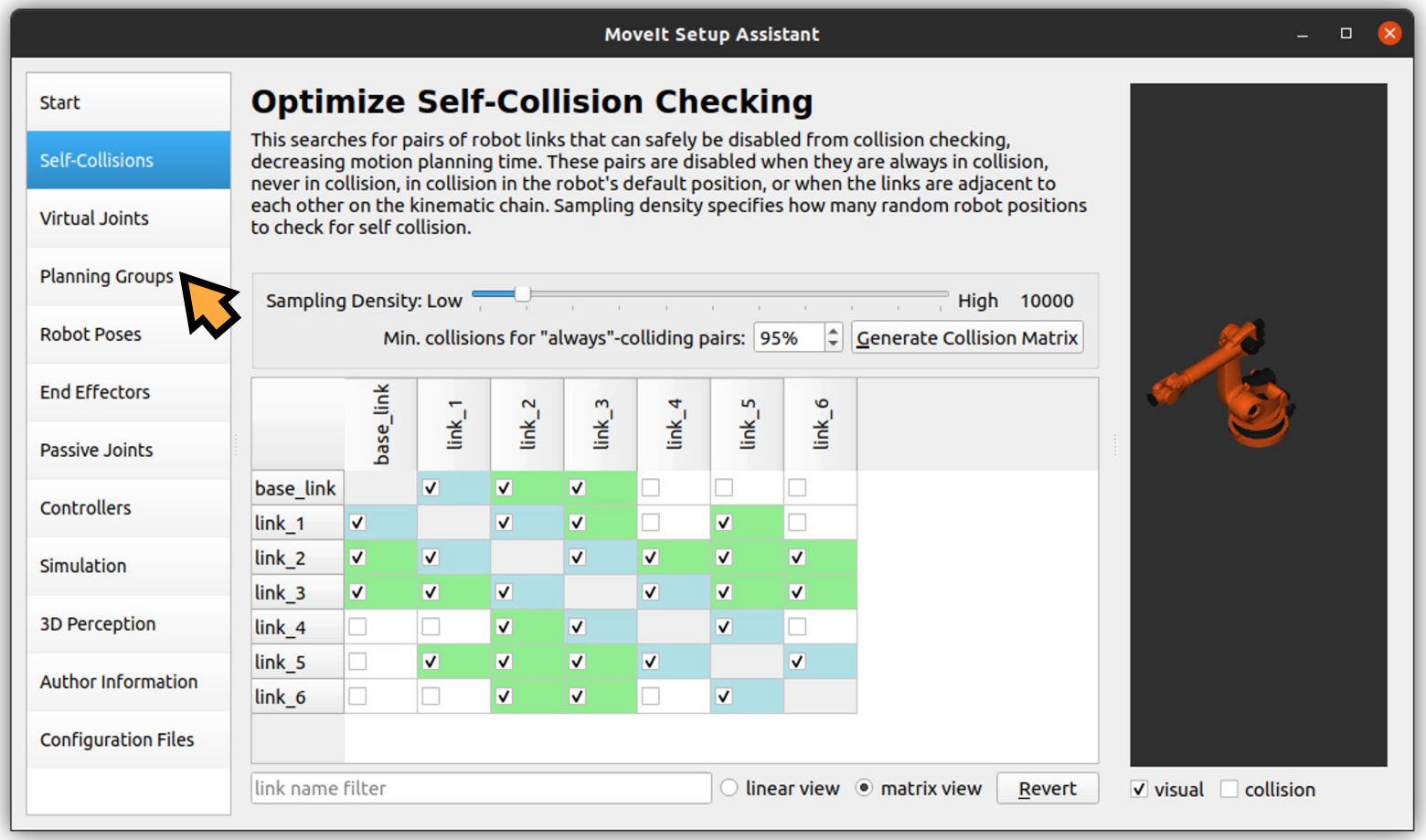
MoveIt Setup Assistant



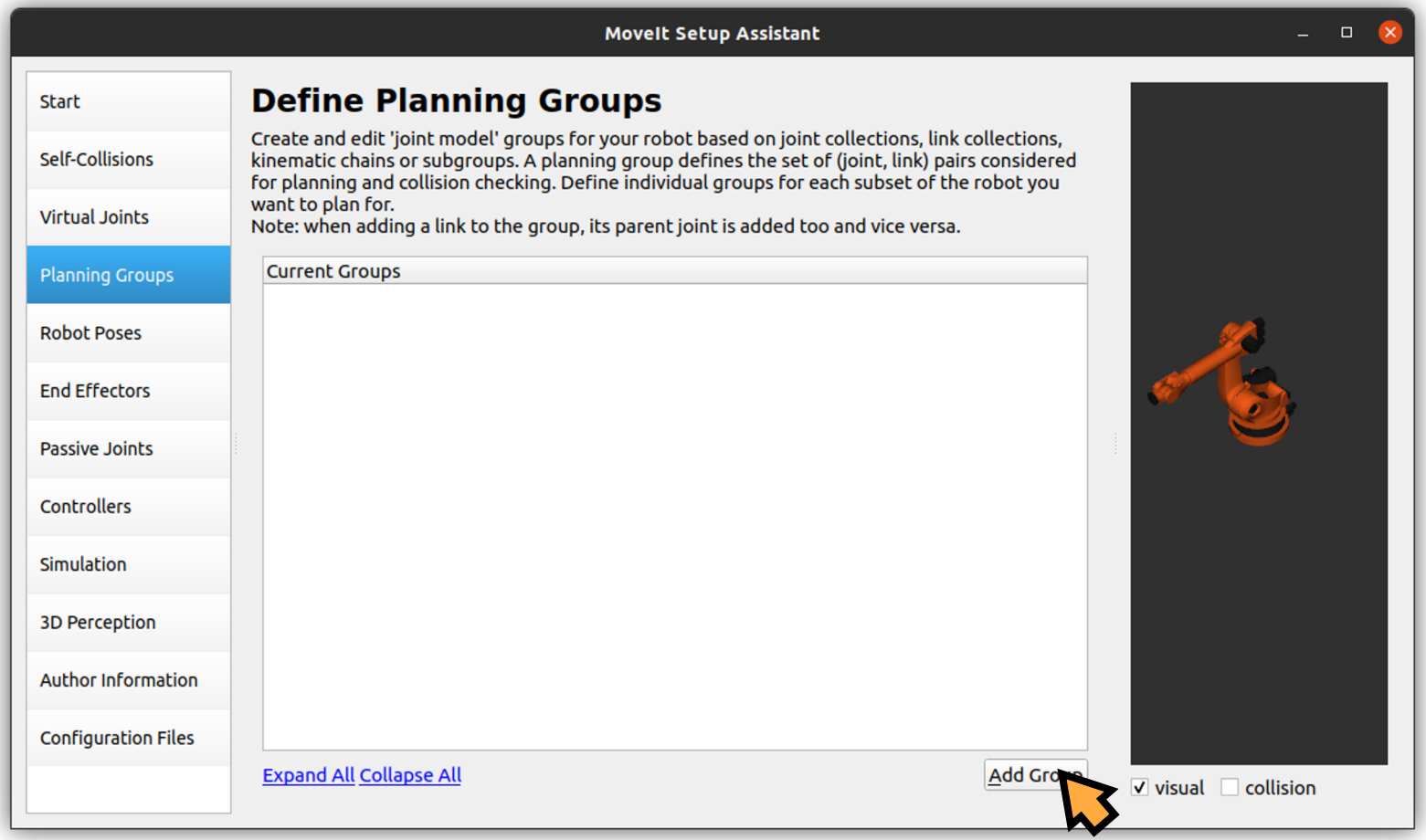
MoveIt Setup Assistant

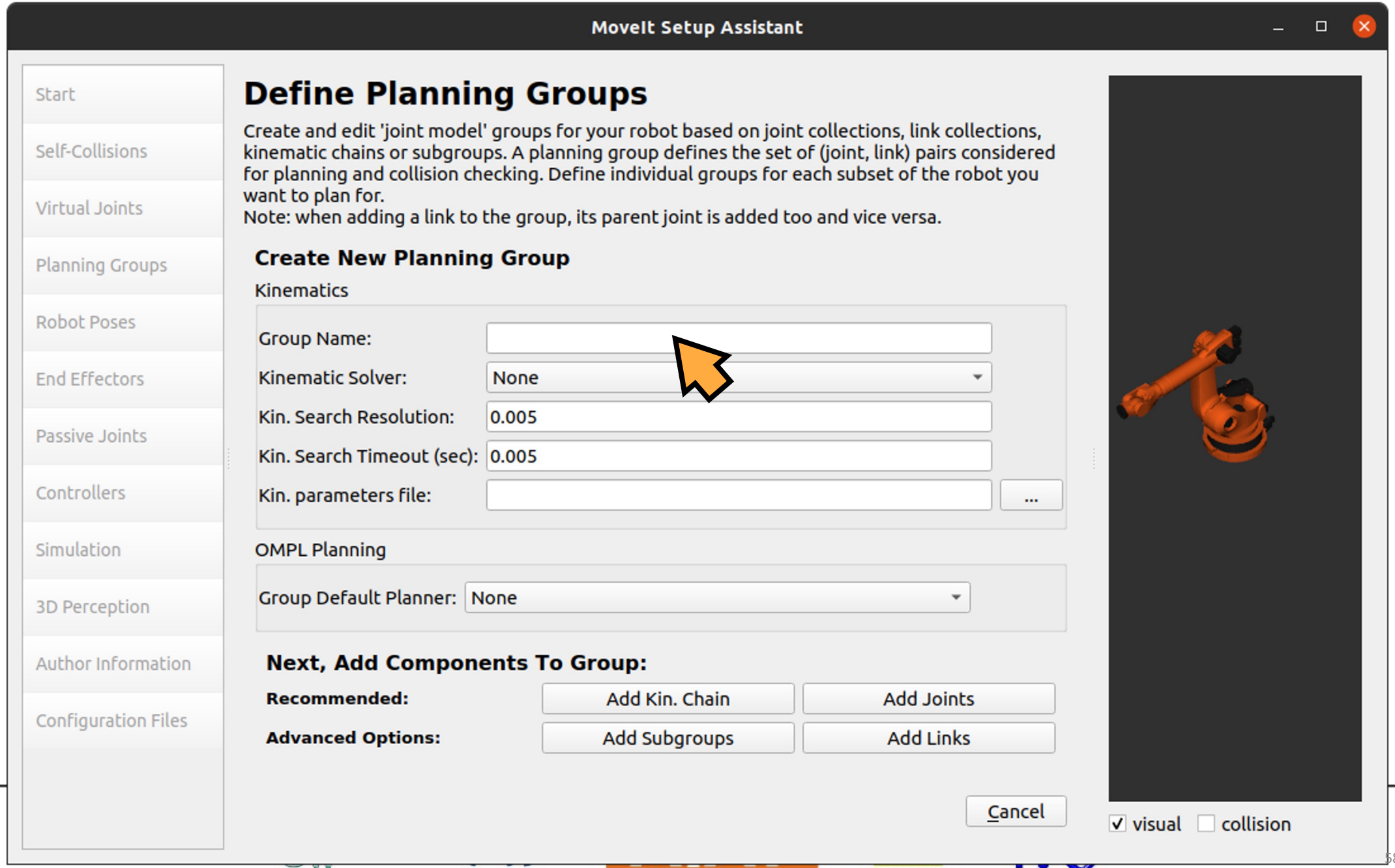


MoveIt Setup Assistant

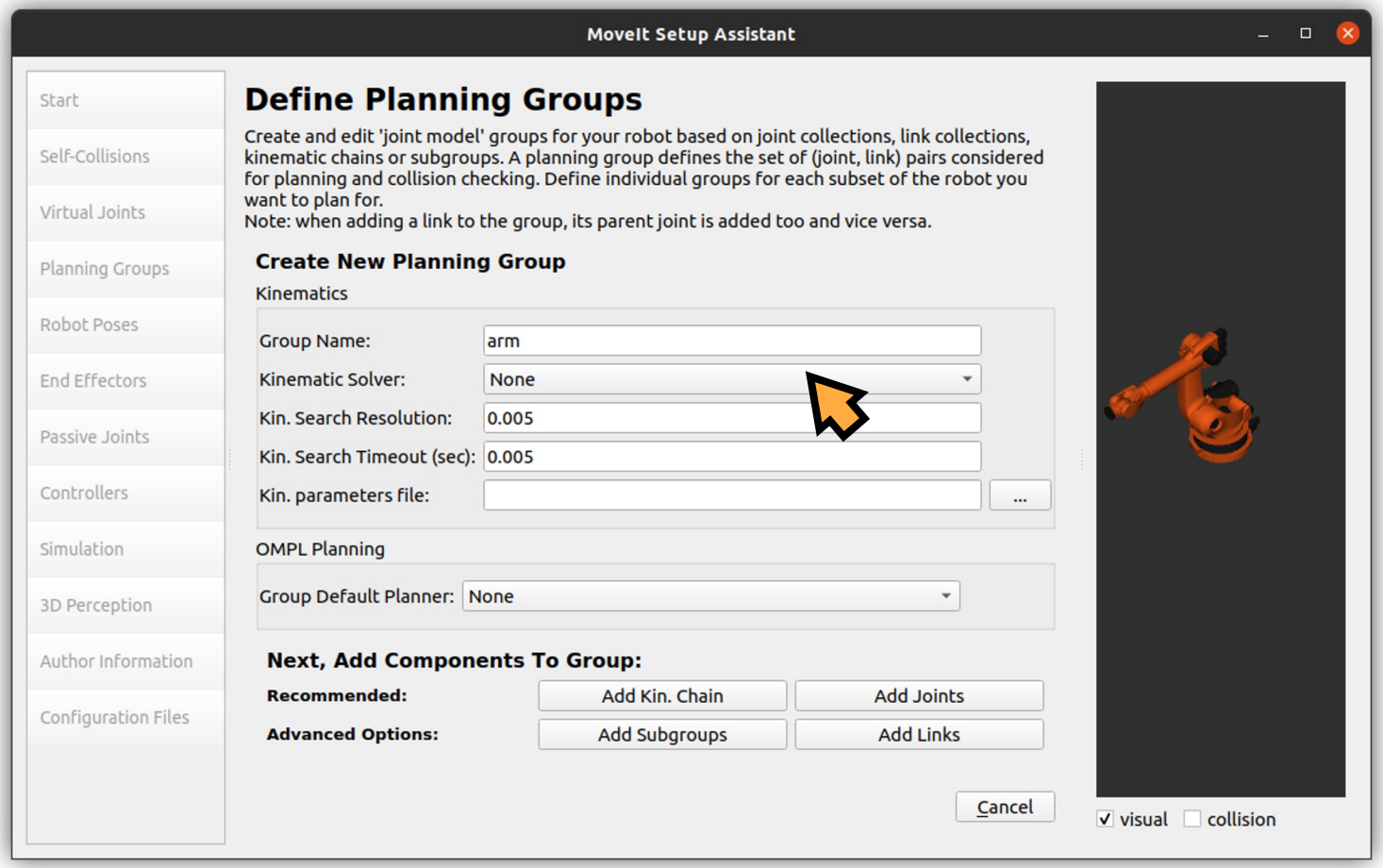


MoveIt Setup Assistant

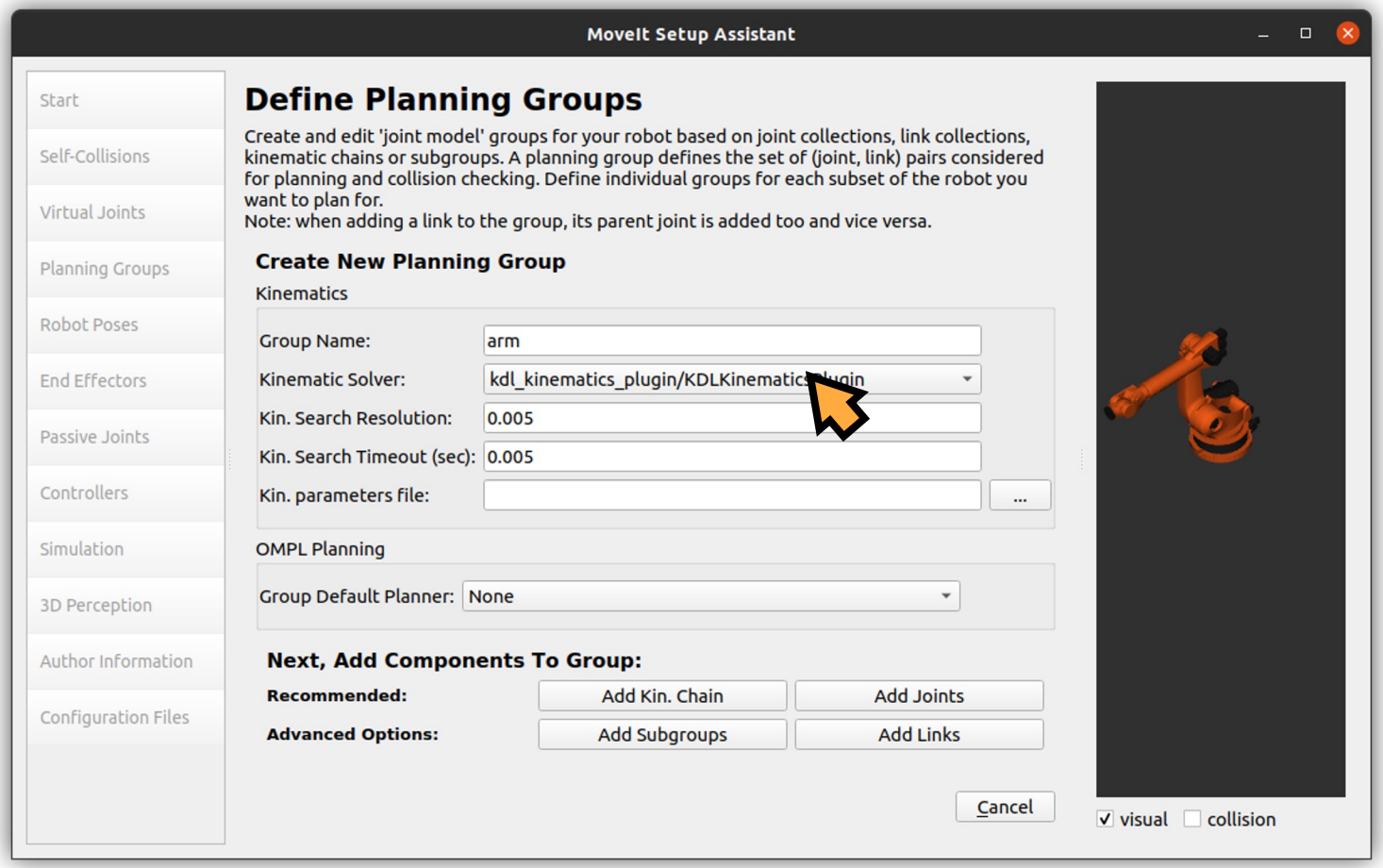



MoveIt Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
Controllers
Simulation
3D Perception
Author Information
Configuration Files
Define Planning Groups
Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for.
Note: when adding a link to the group, its parent joint is added too and vice versa.
Create New Planning Group
Kinematics
Group Name:
Kinematic Solver:
None
Kin. Search Resolution:
0.005
Kin. Search Timeout (sec):
0.005
Kin. parameters file:
...
OMPL Planning
Group Default Planner:
None
Next, Add Components To Group:
Recommended:
Add Kin. Chain
Add Joints
Advanced Options:
Add Subgroups
Add Links
Cancel
visual
collision

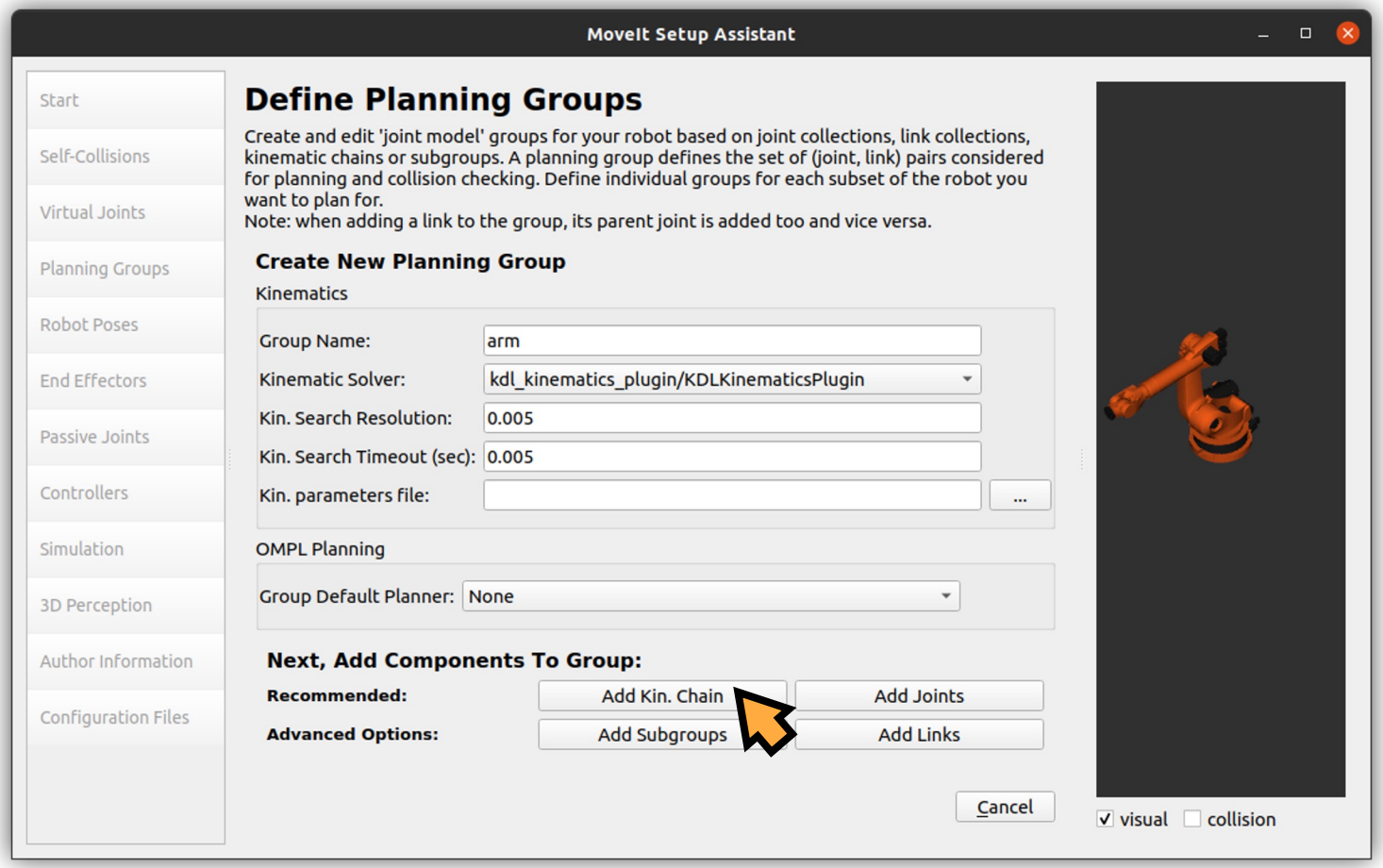
MoveIt Setup Assistant



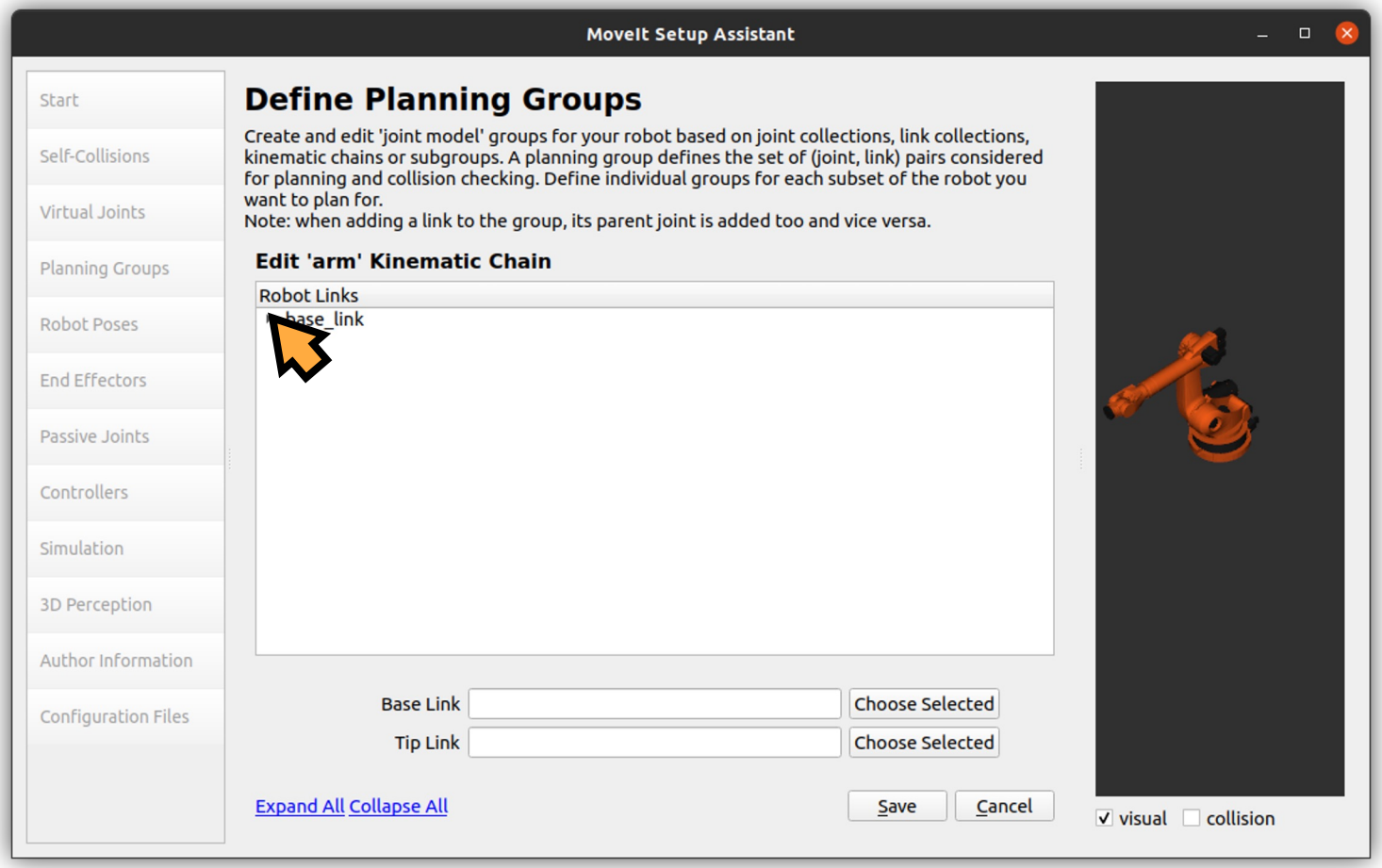
MoveIt Setup Assistant



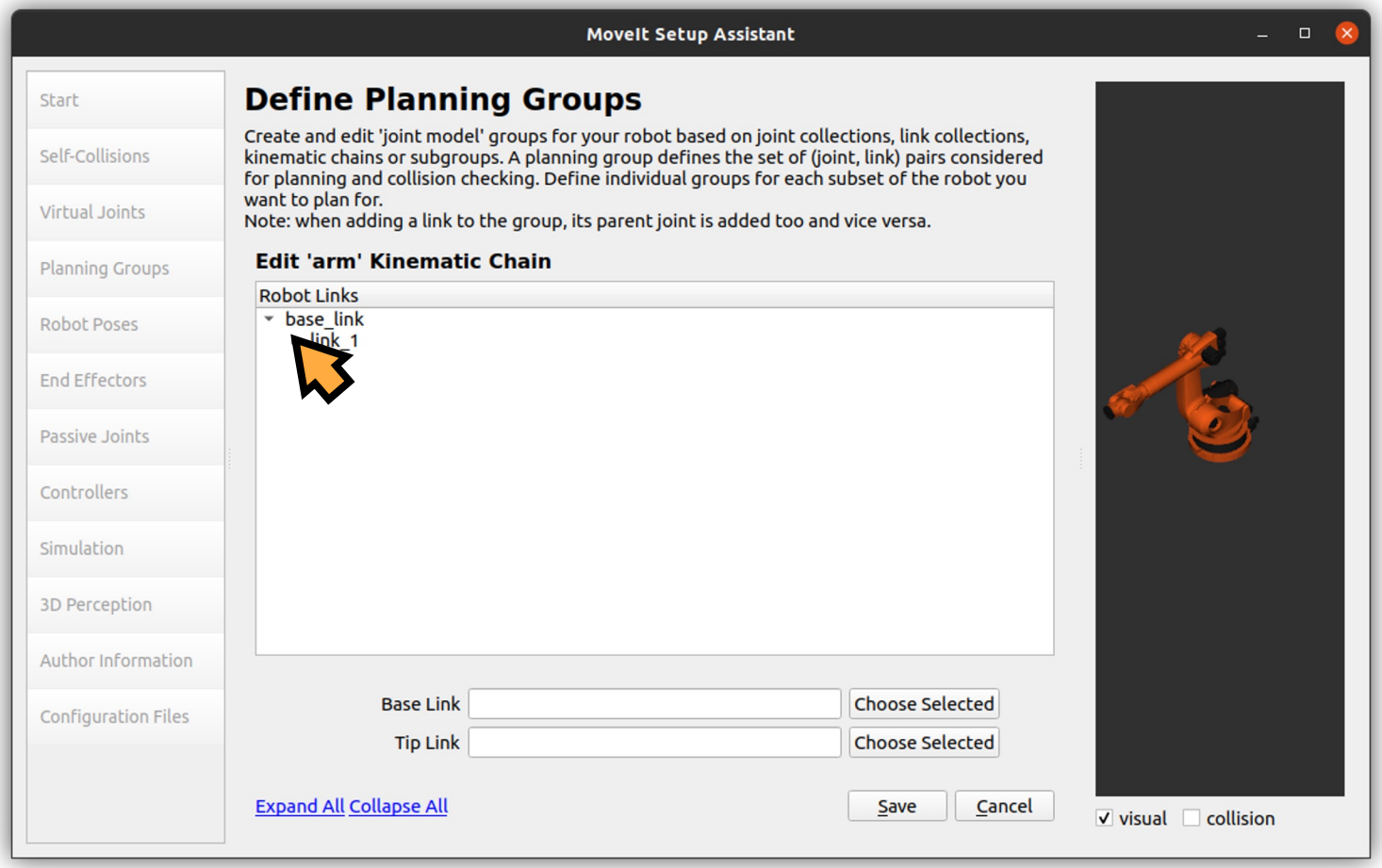
MoveIt Setup Assistant



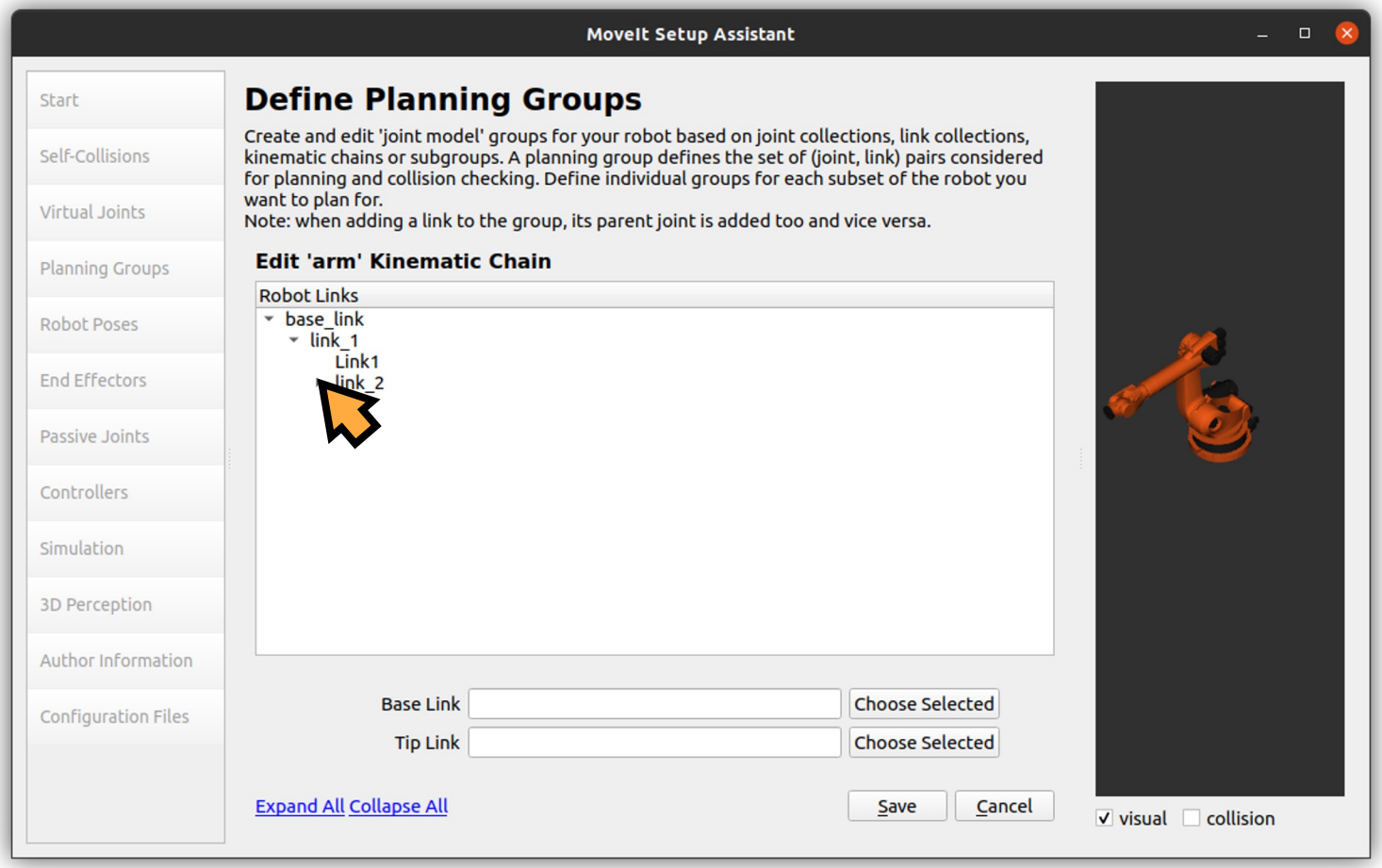
MoveIt Setup Assistant



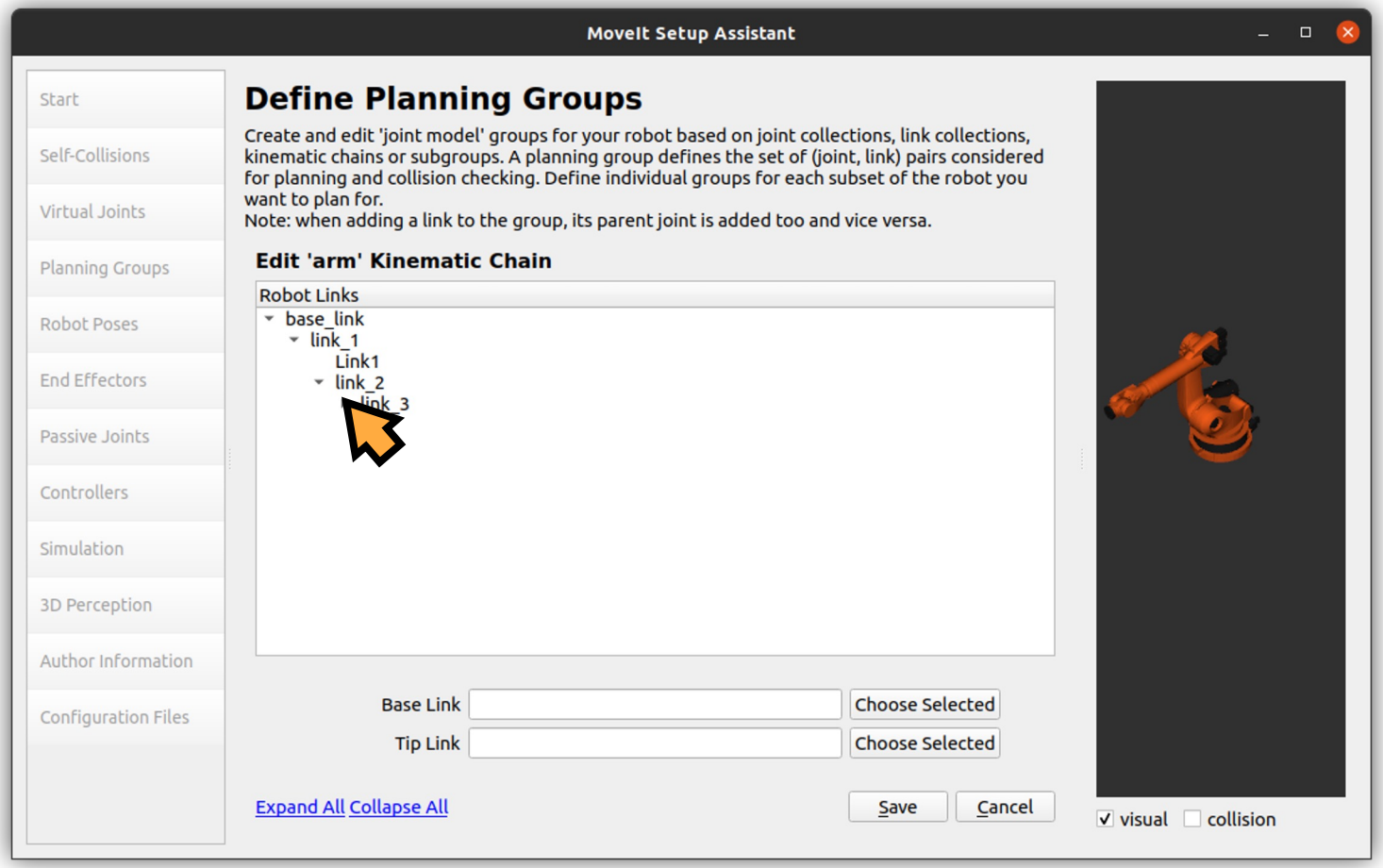
MoveIt Setup Assistant



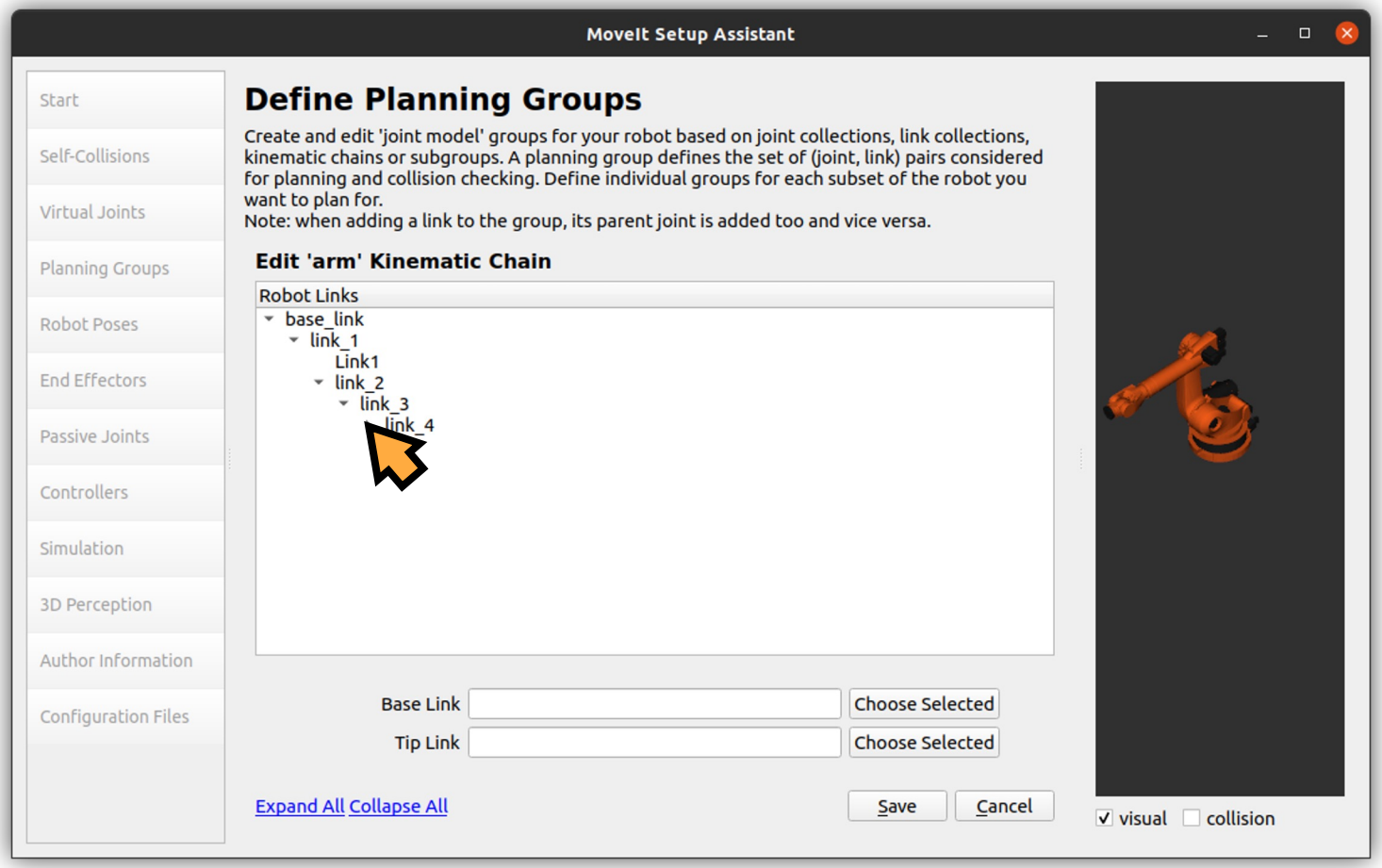
MoveIt Setup Assistant



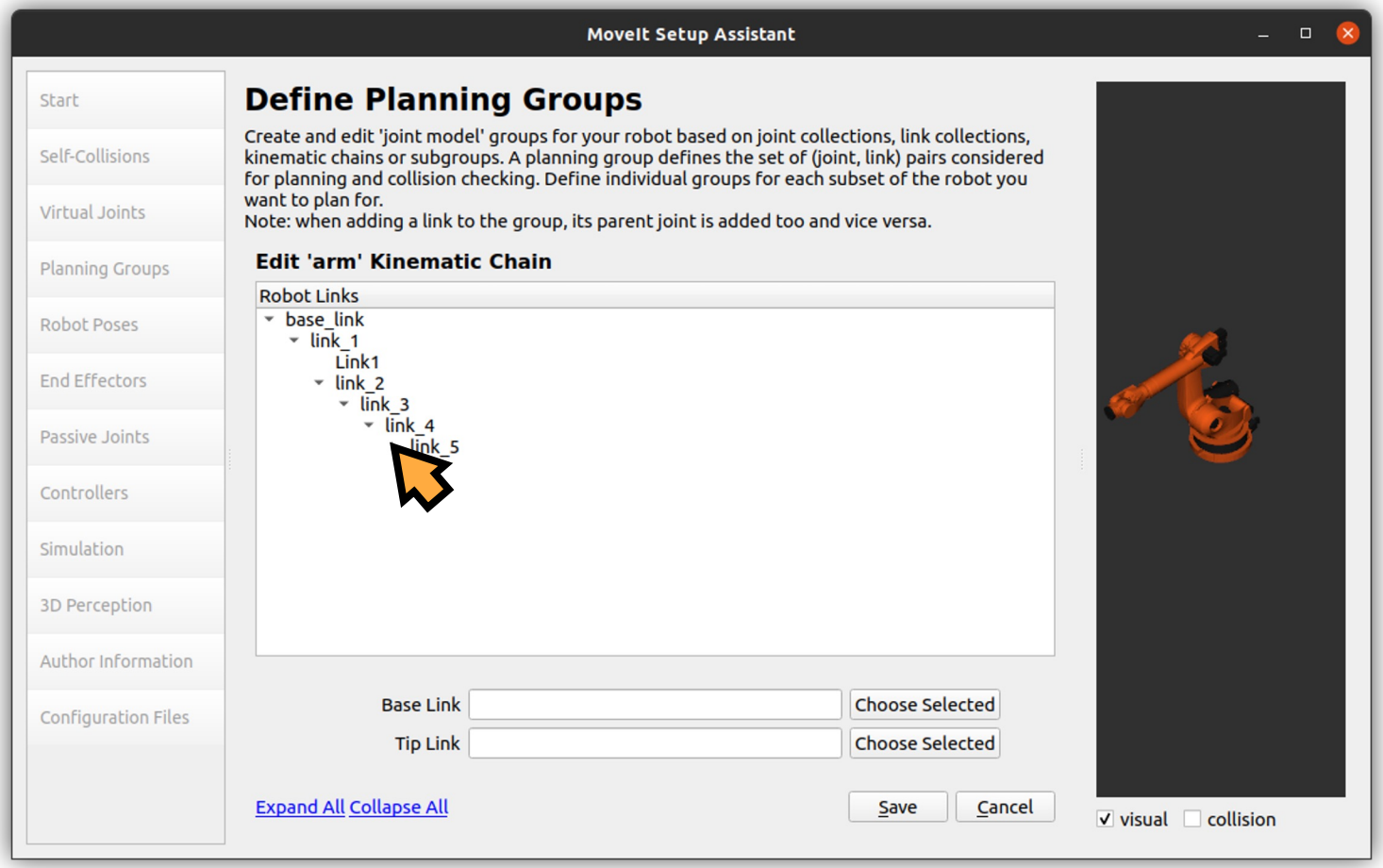
MoveIt Setup Assistant



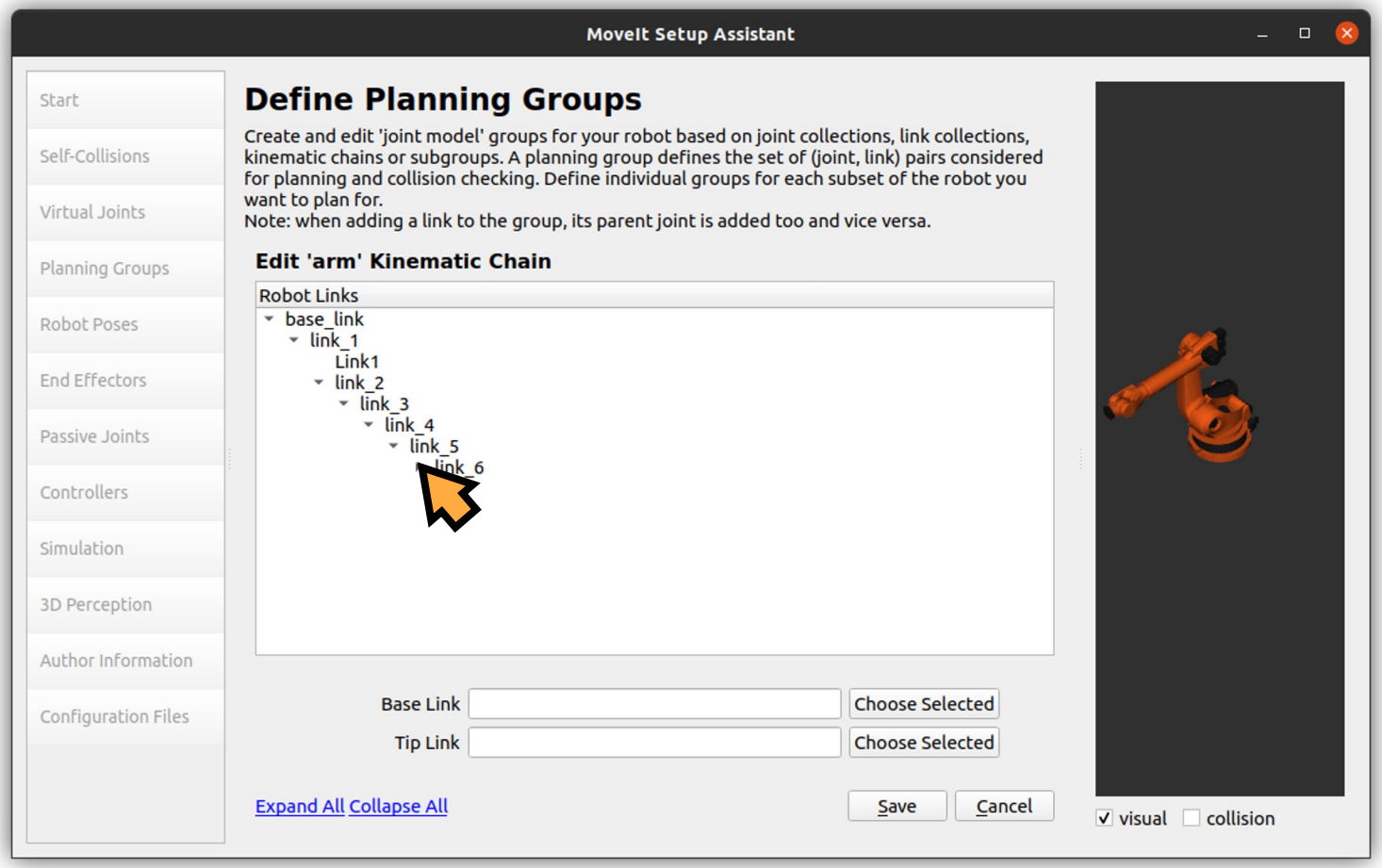
MoveIt Setup Assistant



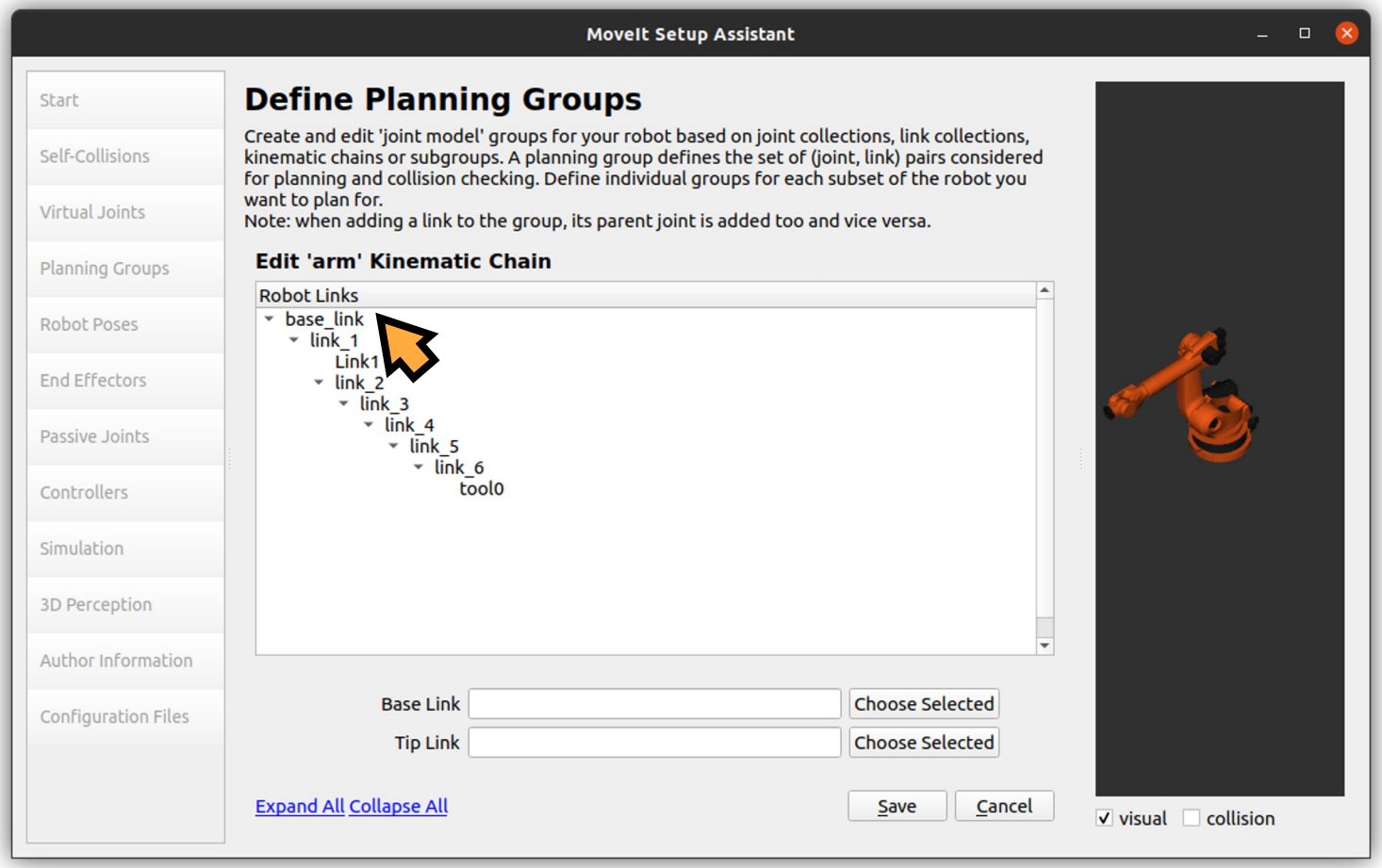
MoveIt Setup Assistant



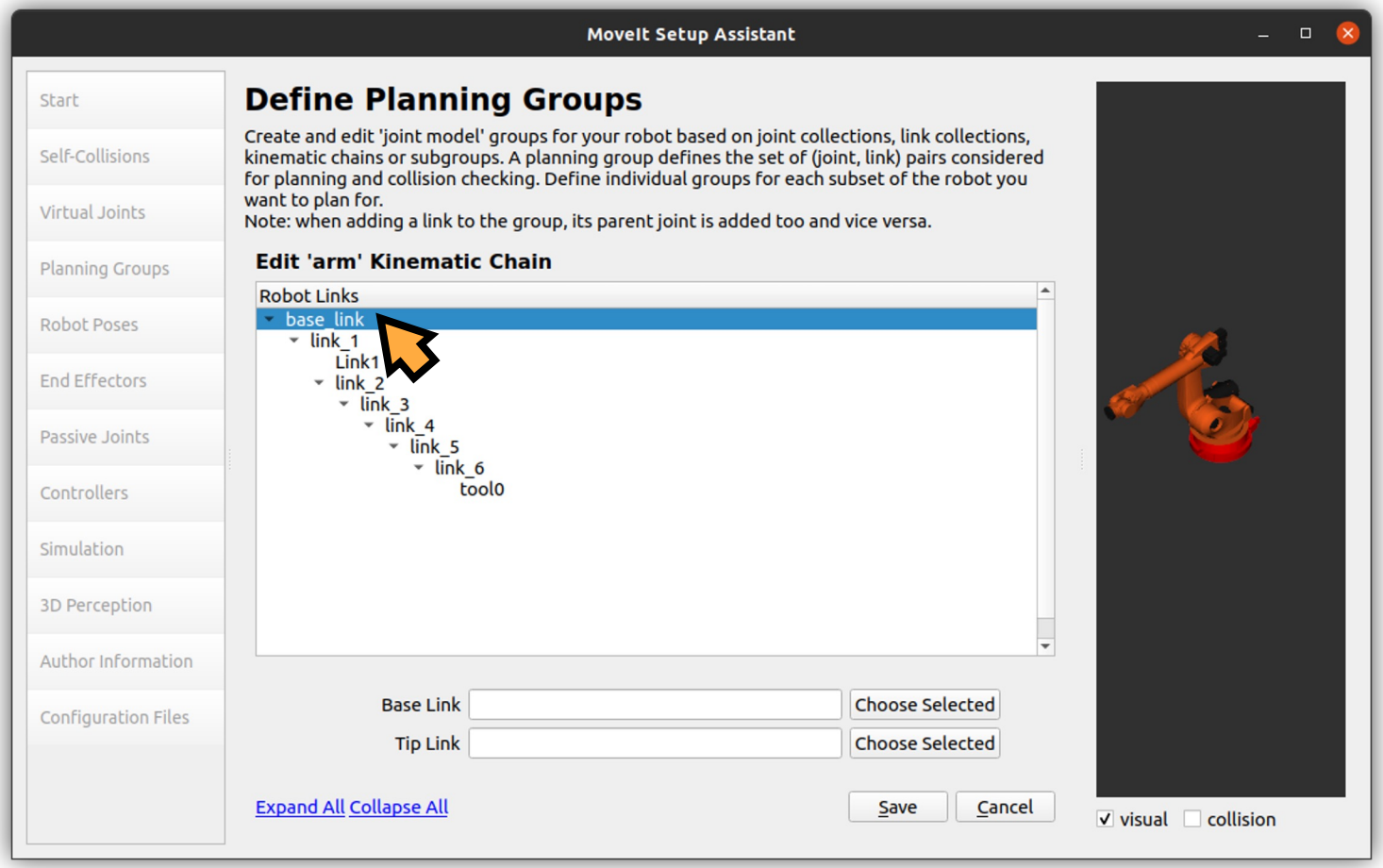
MoveIt Setup Assistant



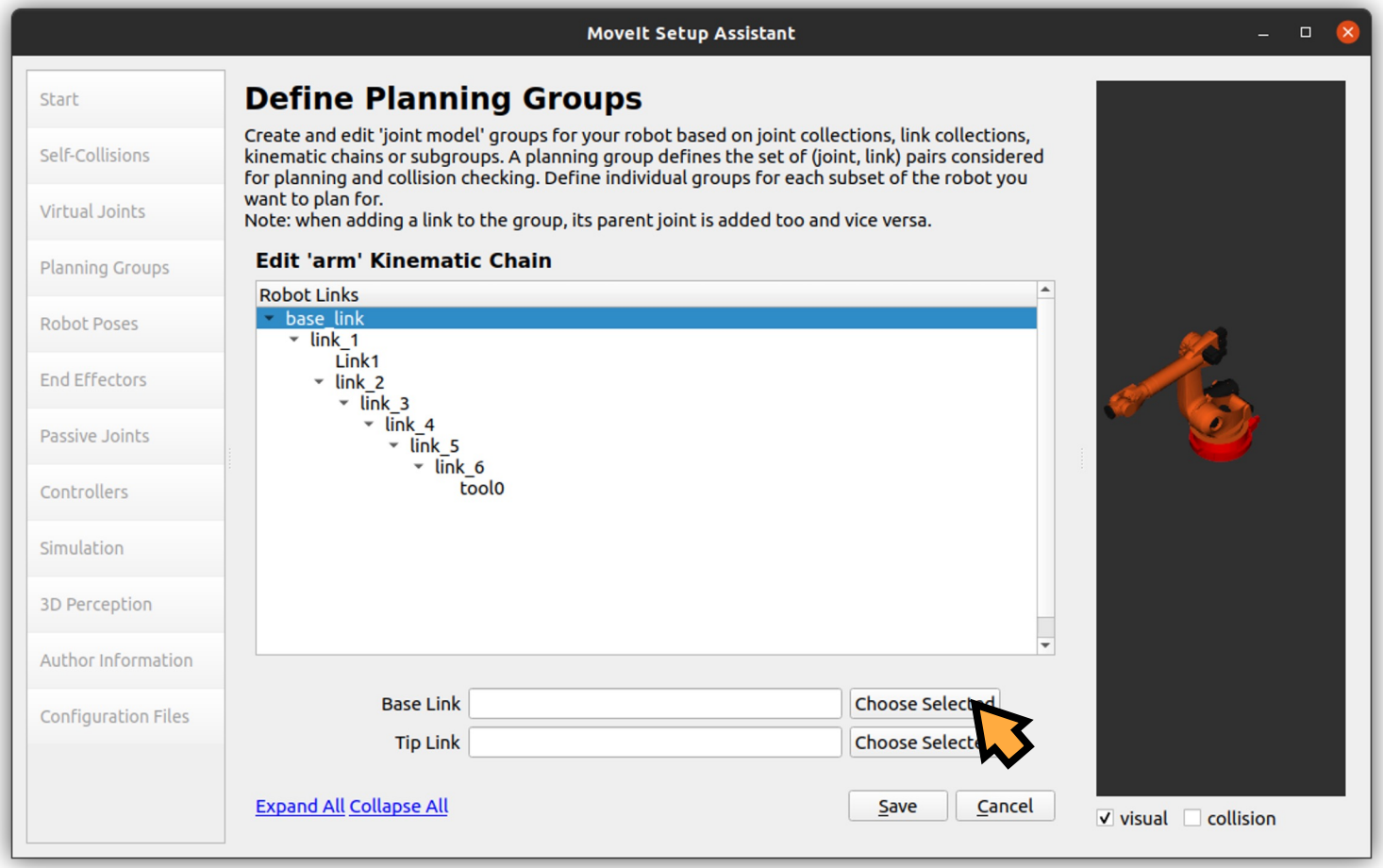
MoveIt Setup Assistant



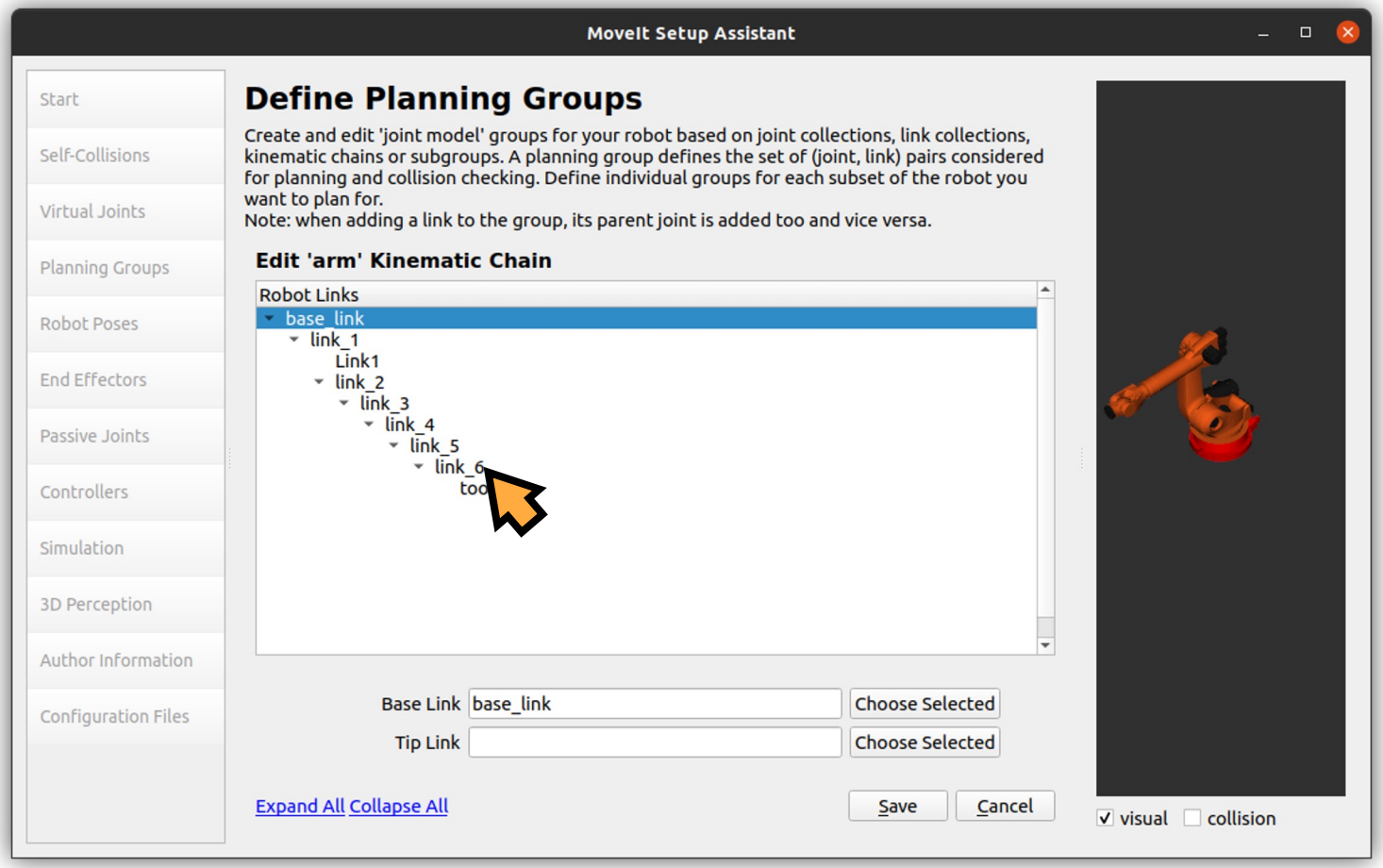
MoveIt Setup Assistant



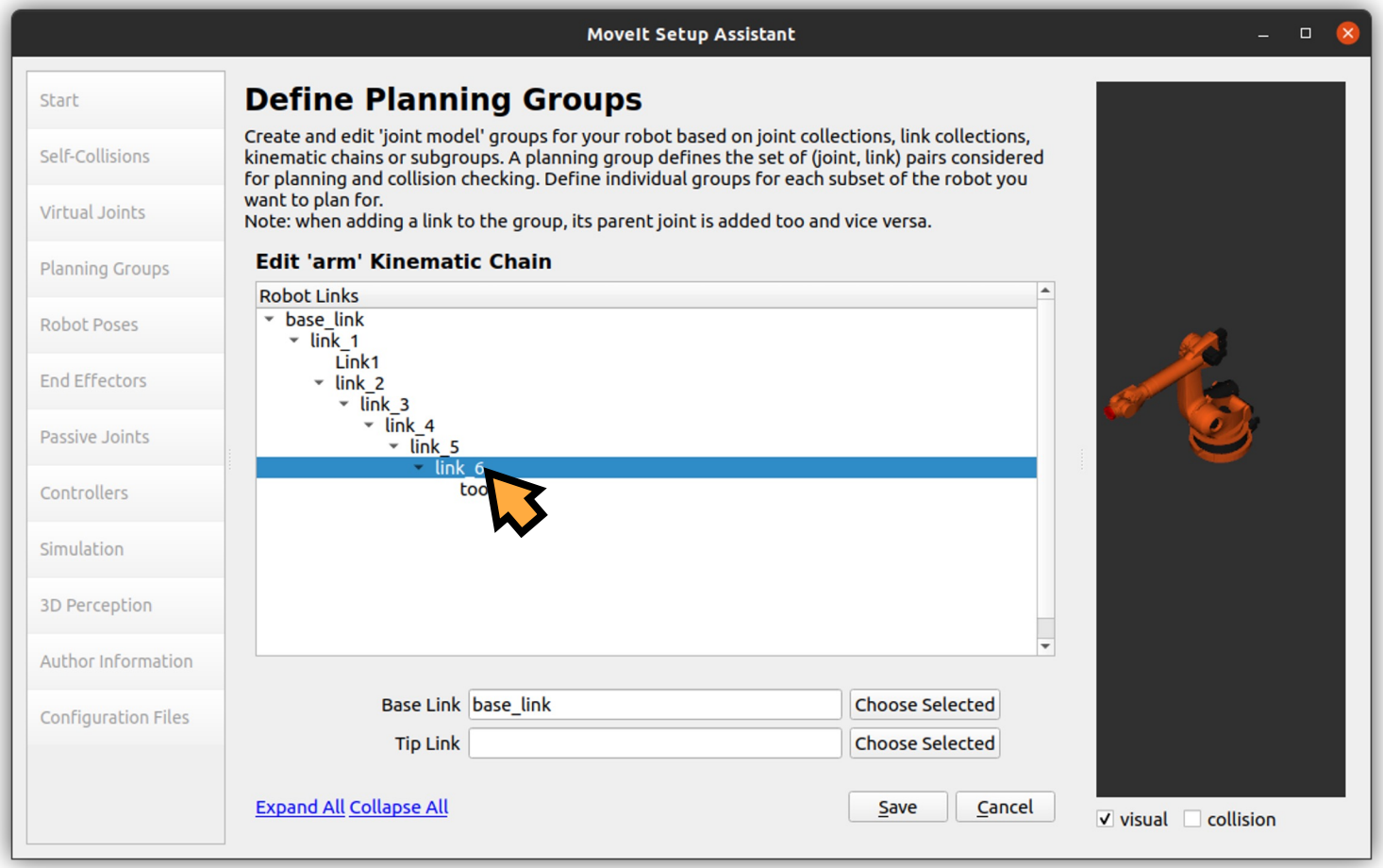
MoveIt Setup Assistant



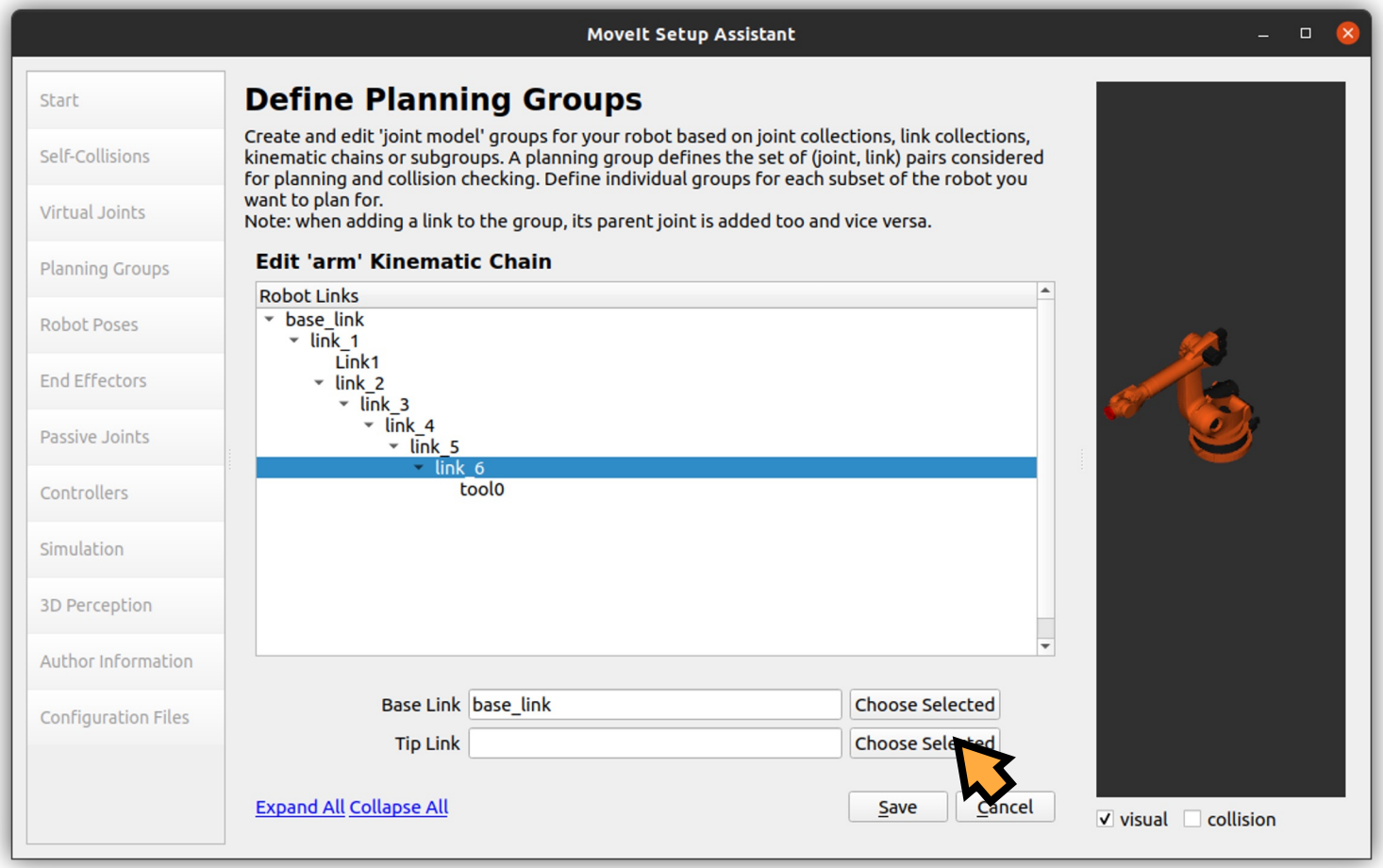
MoveIt Setup Assistant



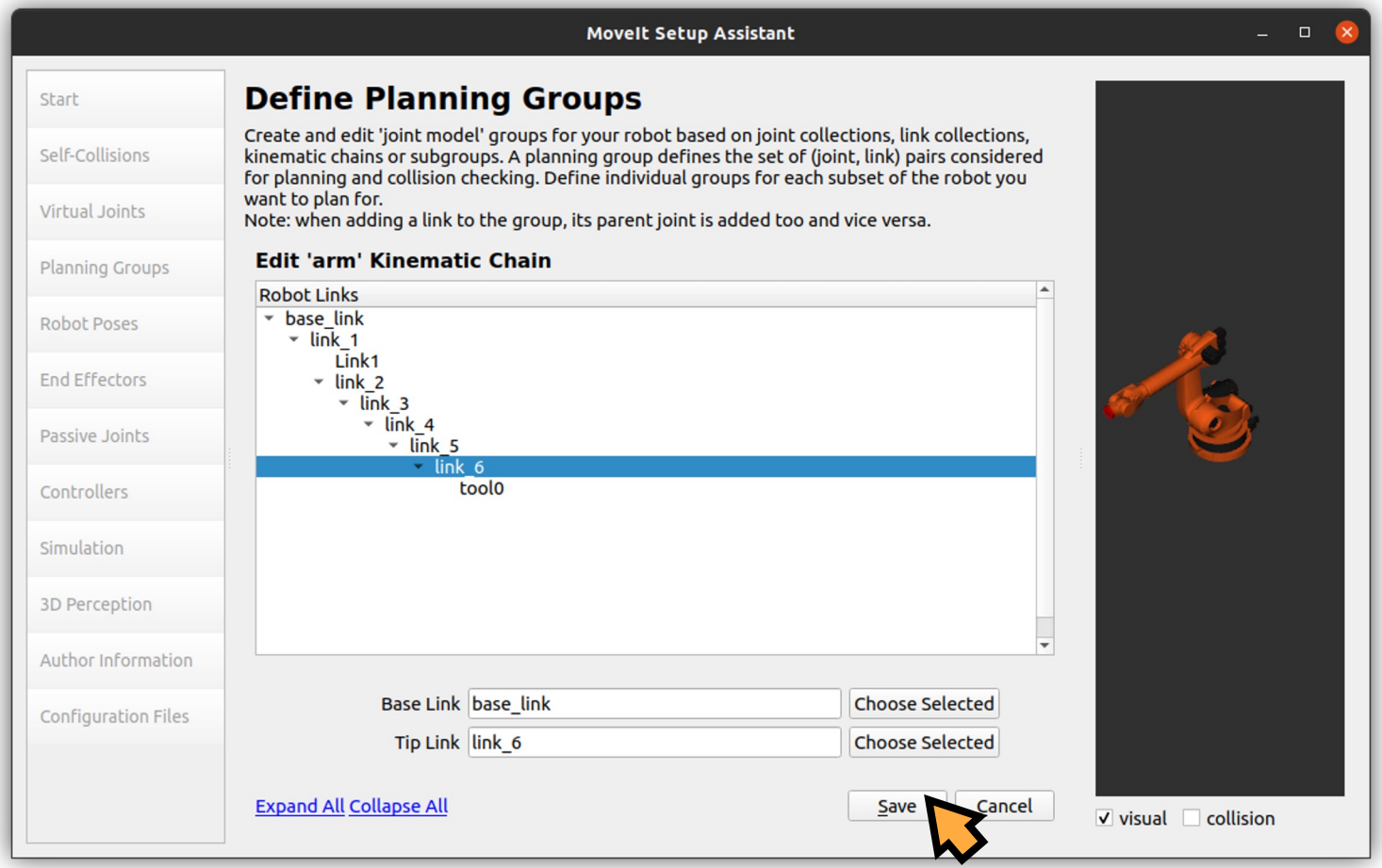
MoveIt Setup Assistant



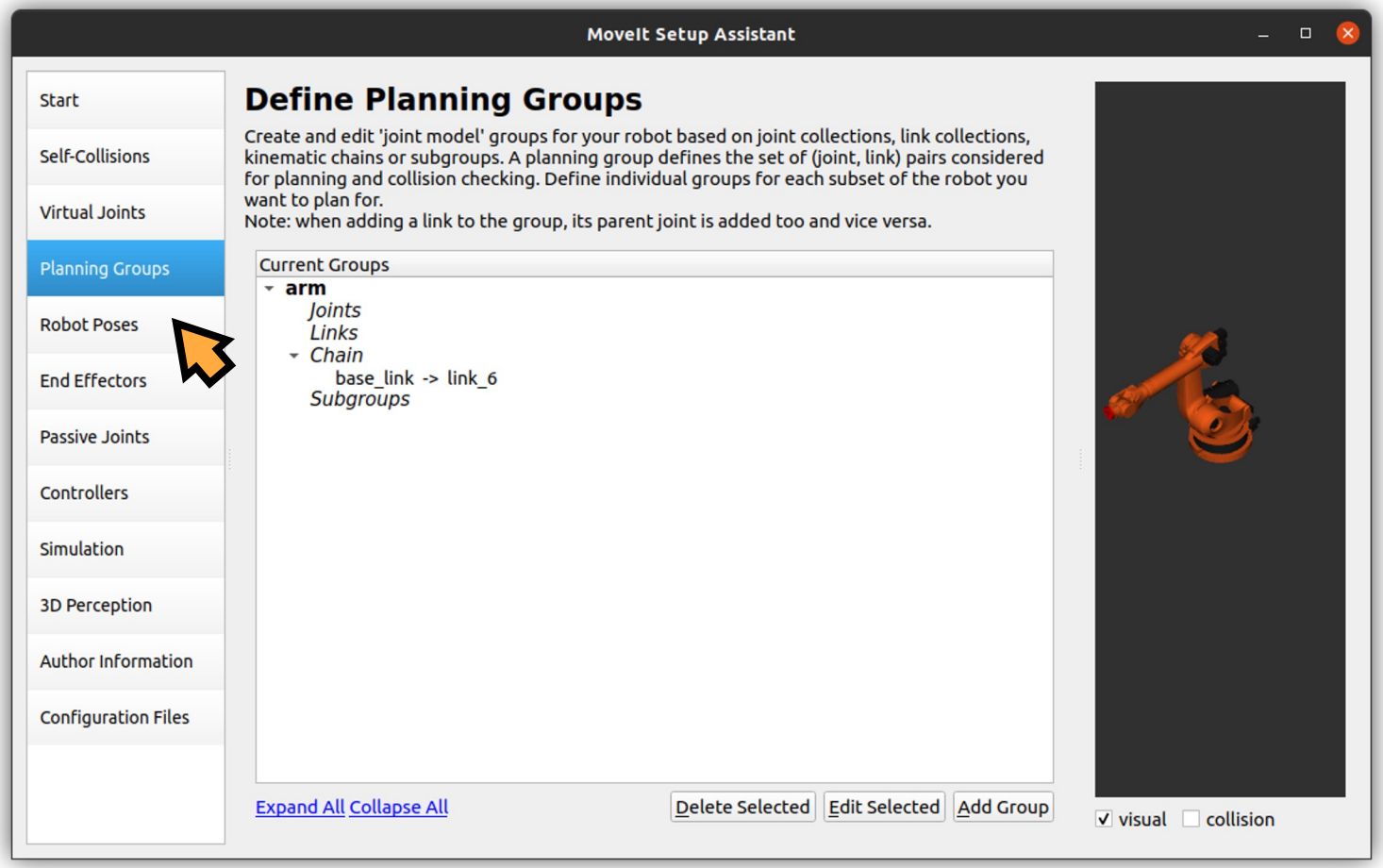
MoveIt Setup Assistant



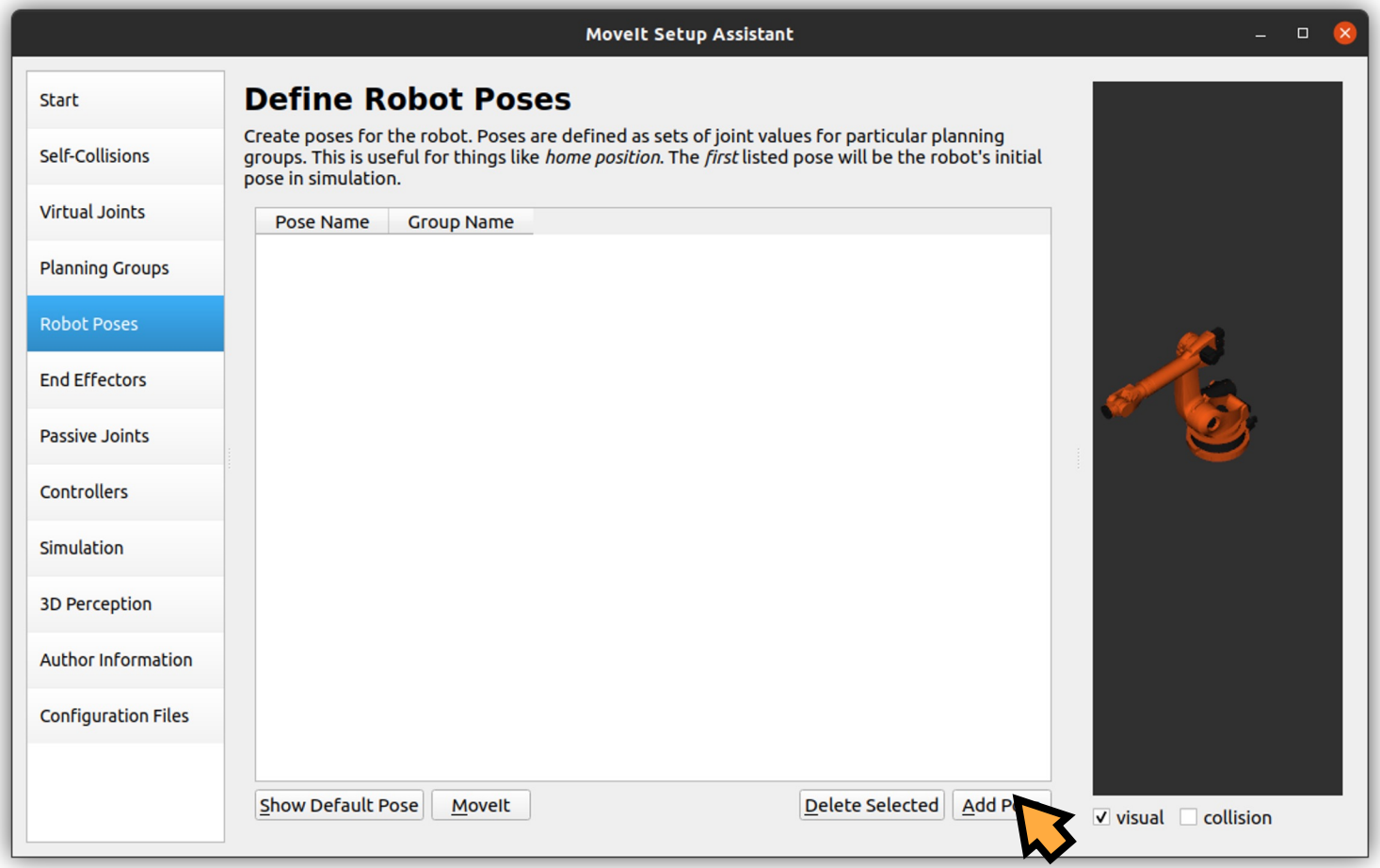
MoveIt Setup Assistant



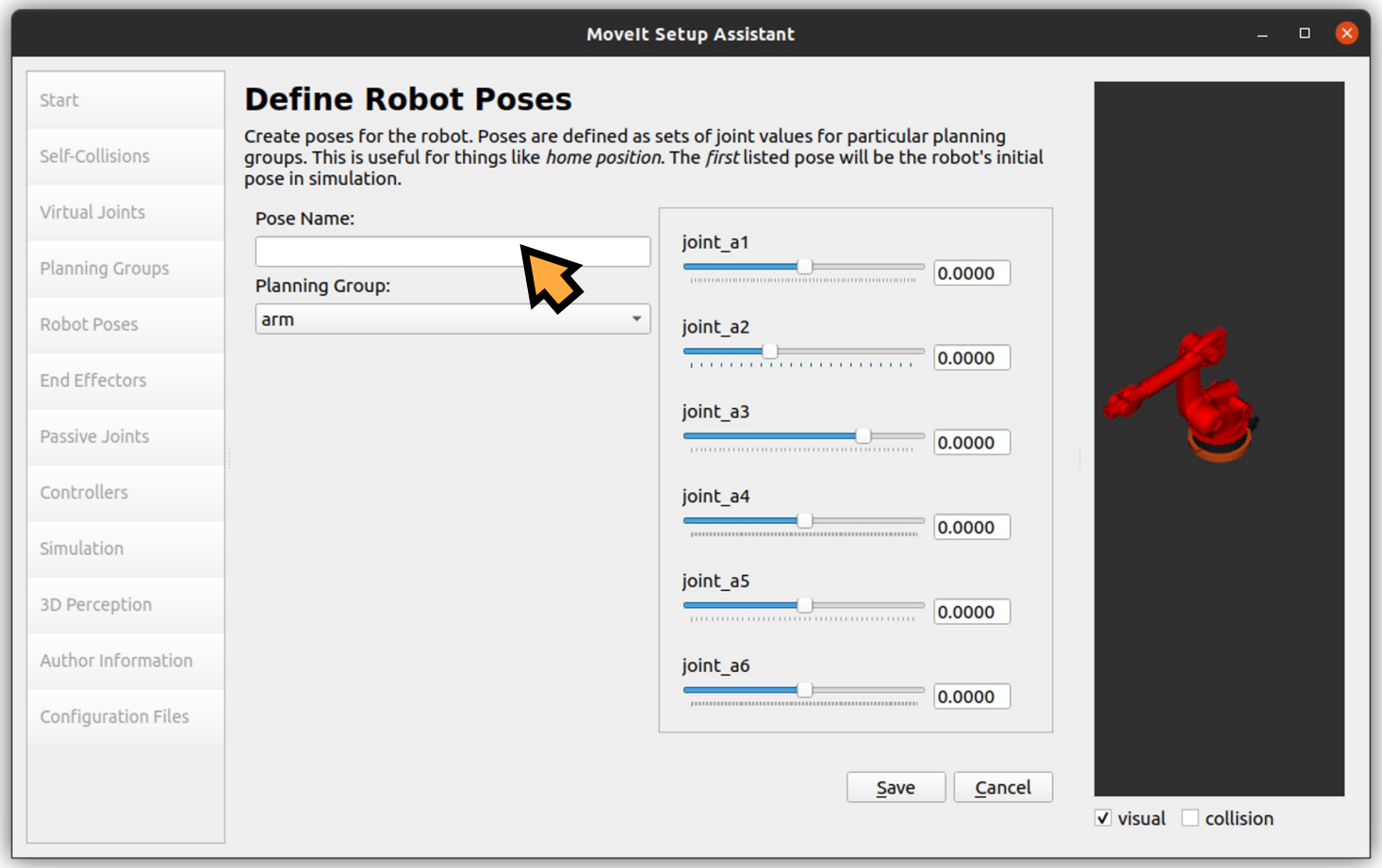
MoveIt Setup Assistant



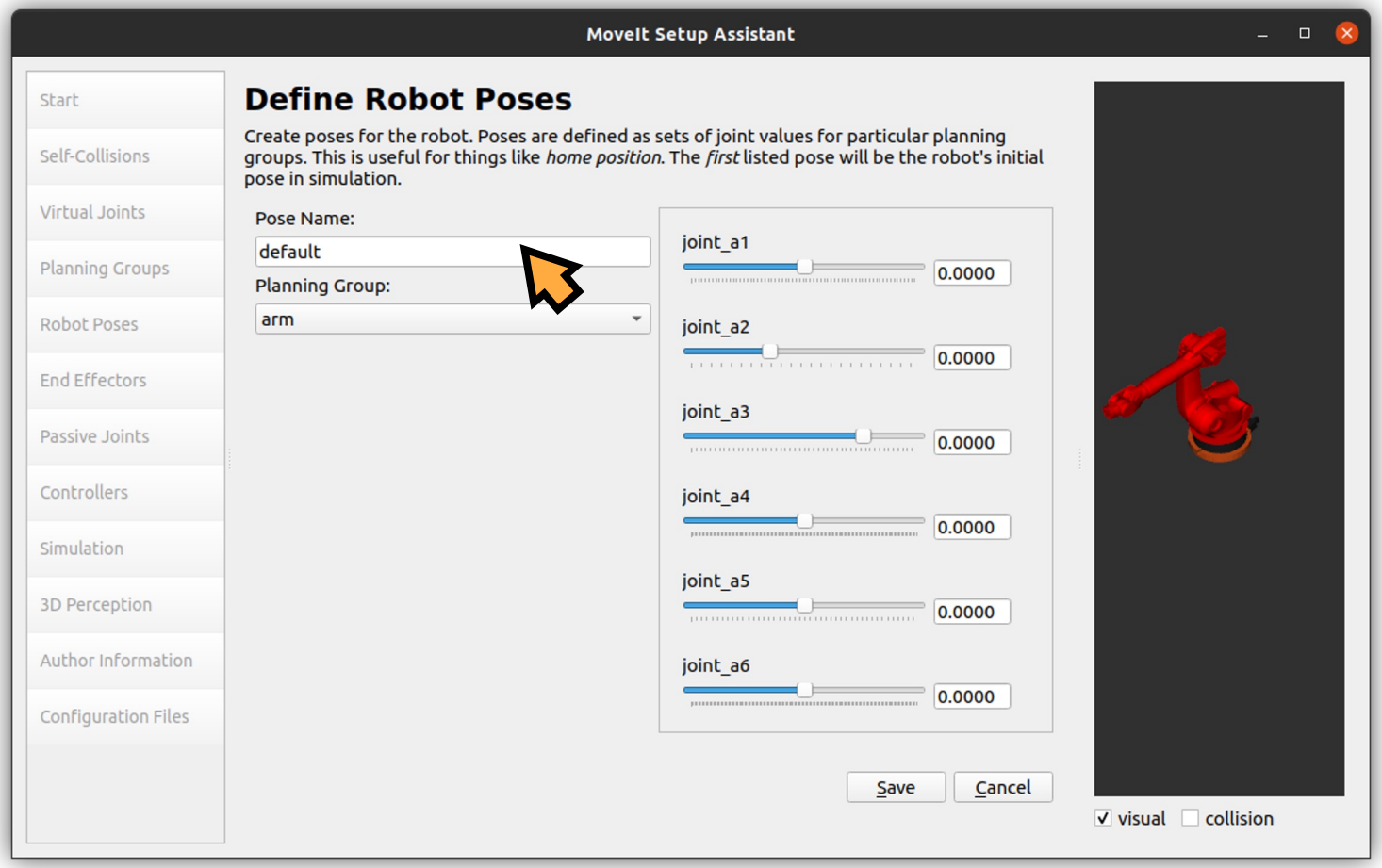
MoveIt Setup Assistant



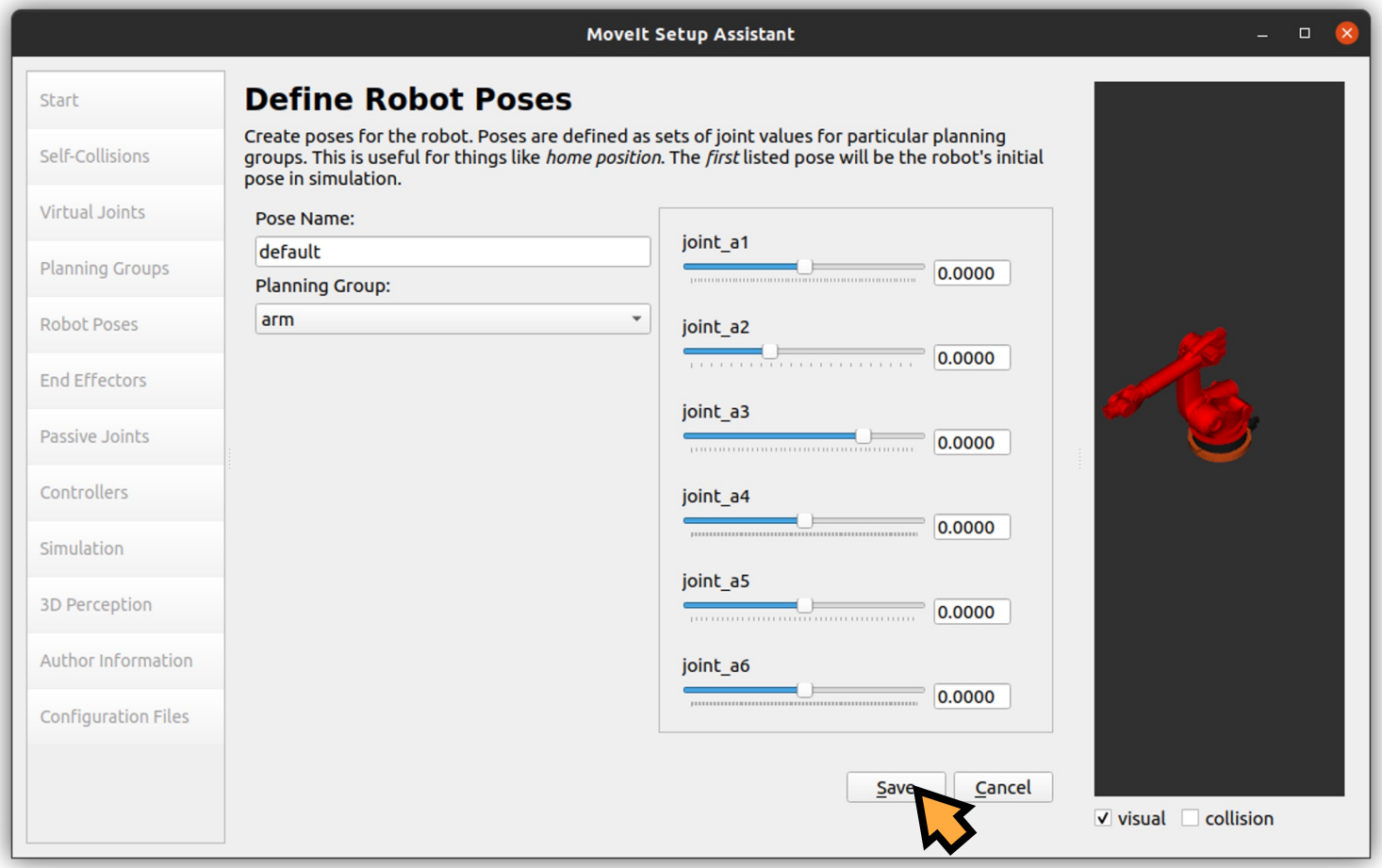
MoveIt Setup Assistant



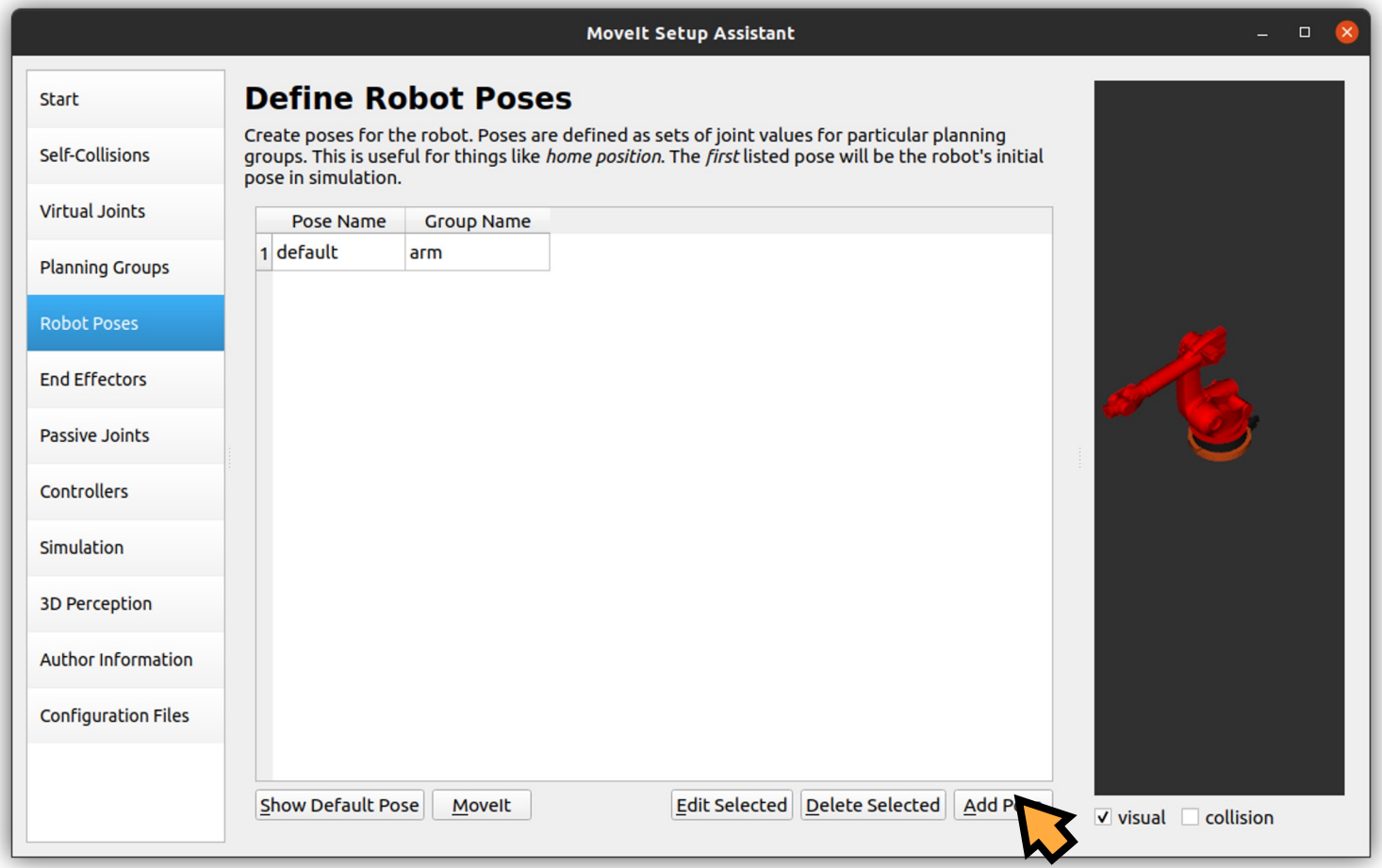
MoveIt Setup Assistant



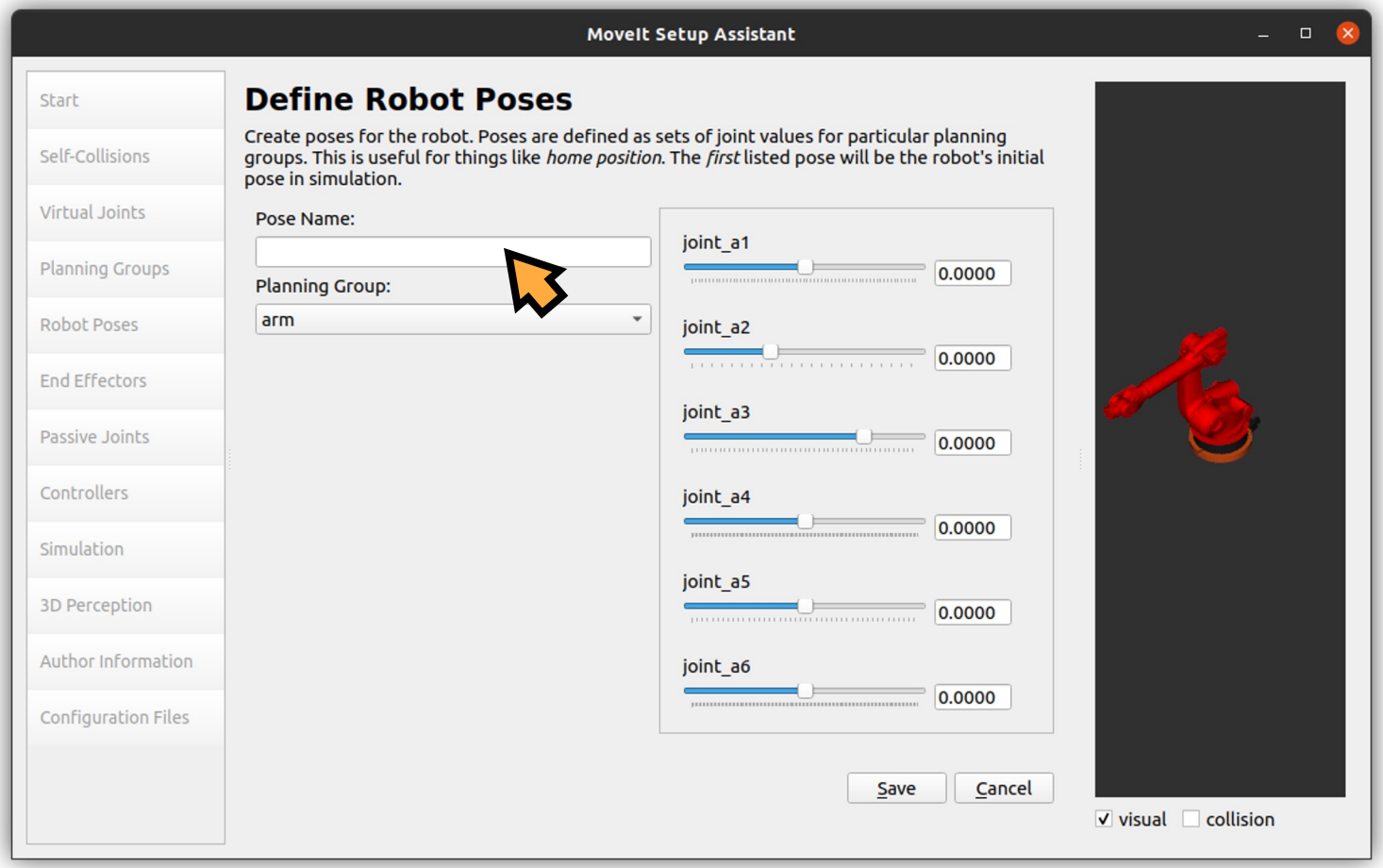
MoveIt Setup Assistant



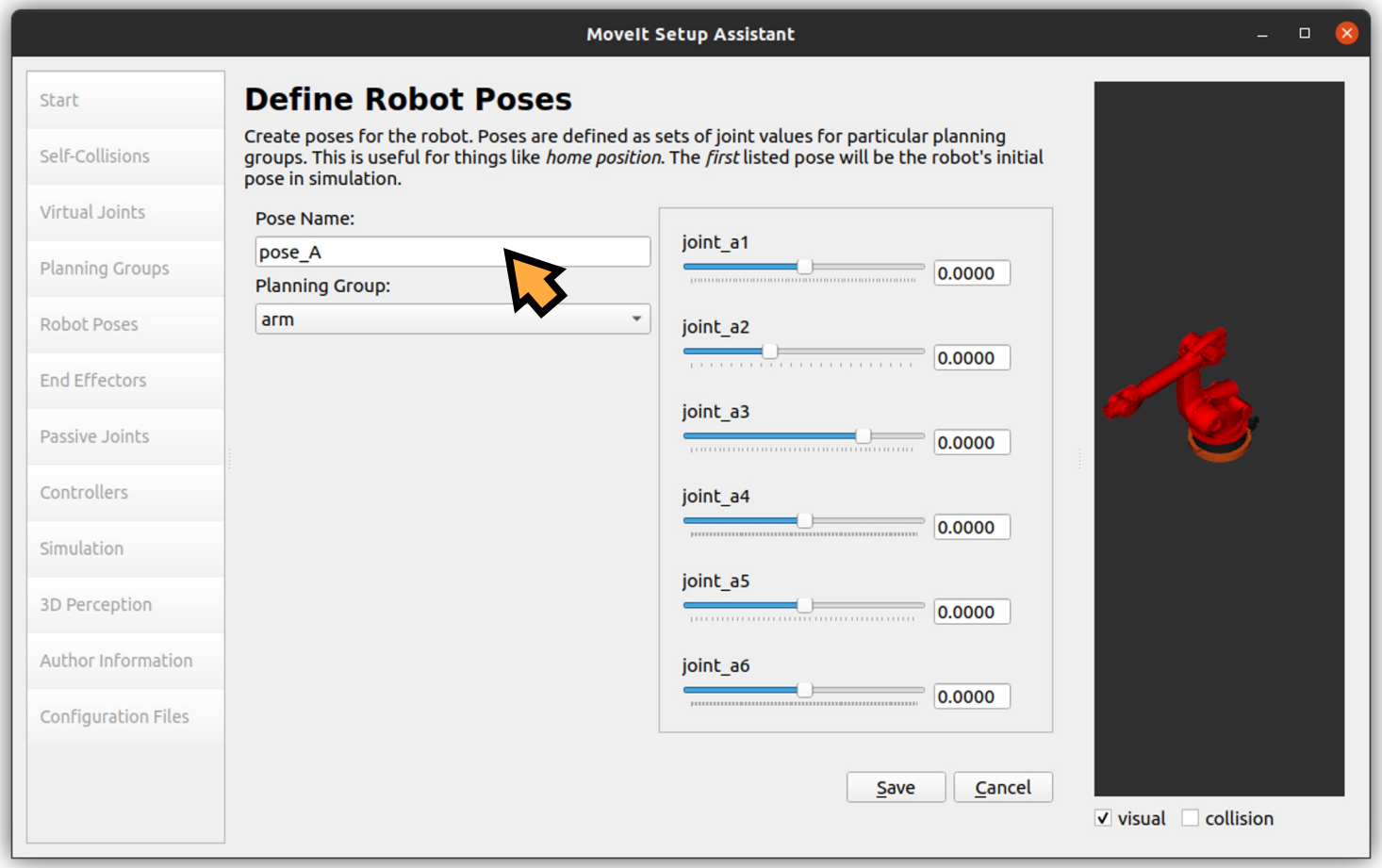
MoveIt Setup Assistant



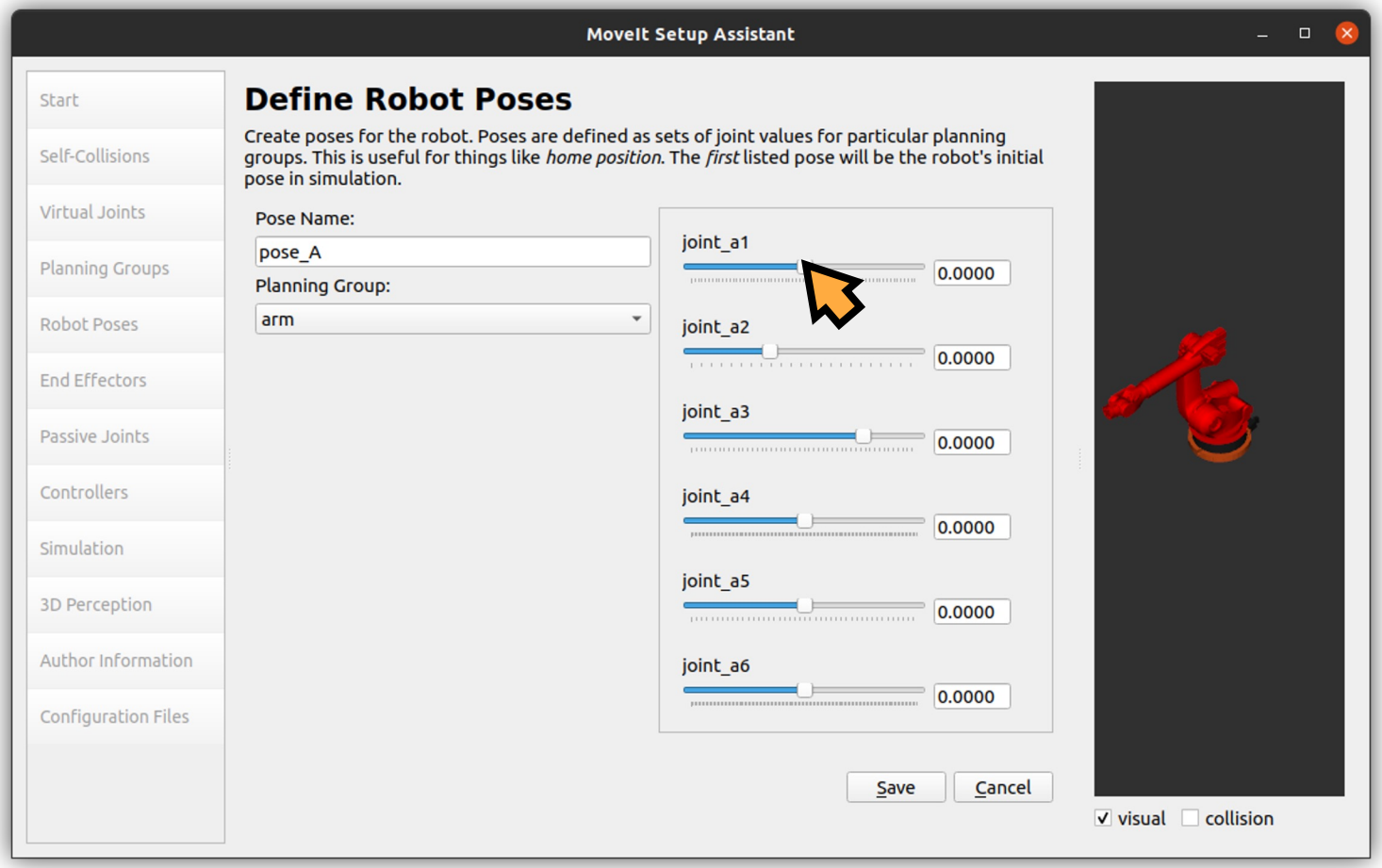
MoveIt Setup Assistant



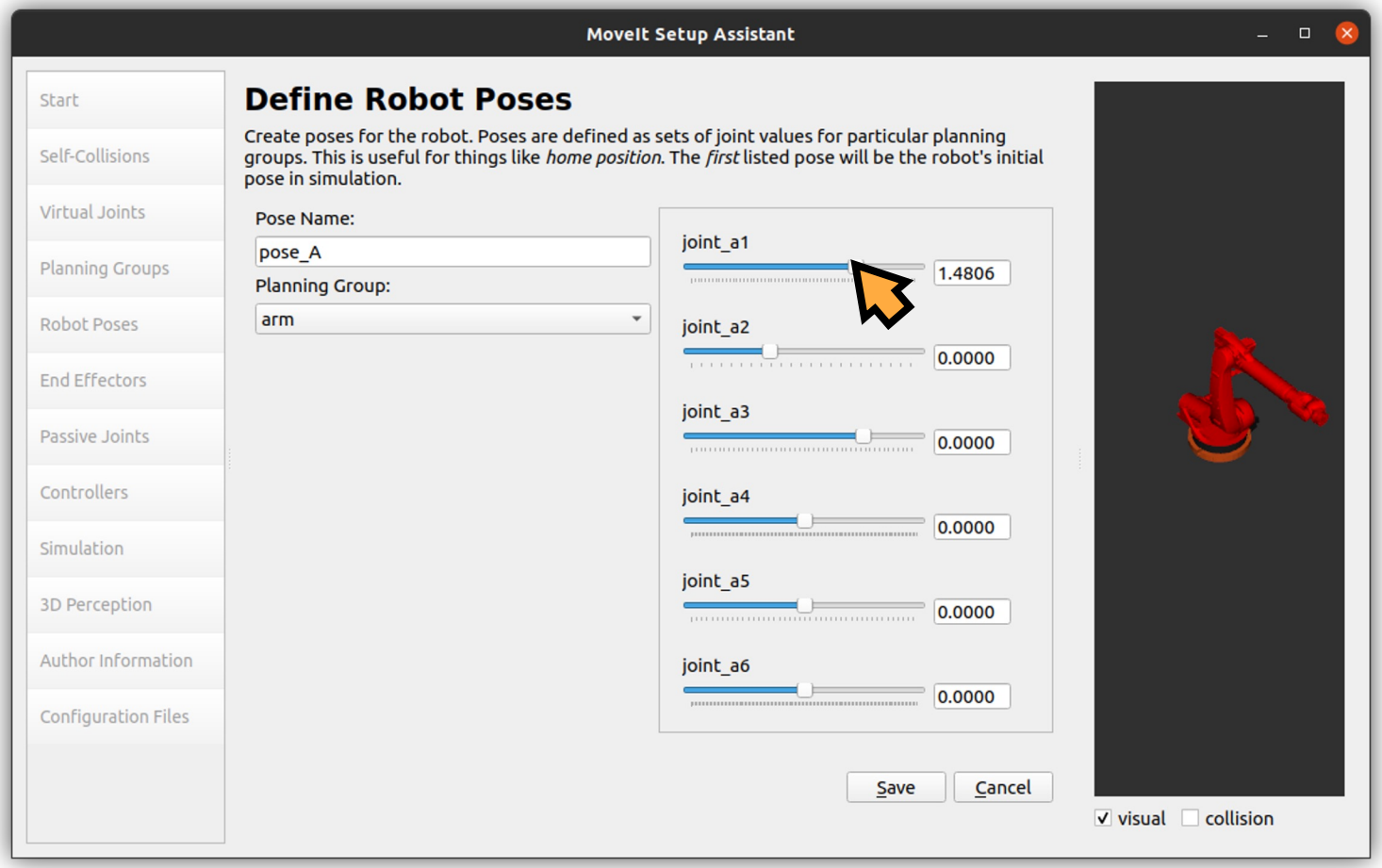
MoveIt Setup Assistant



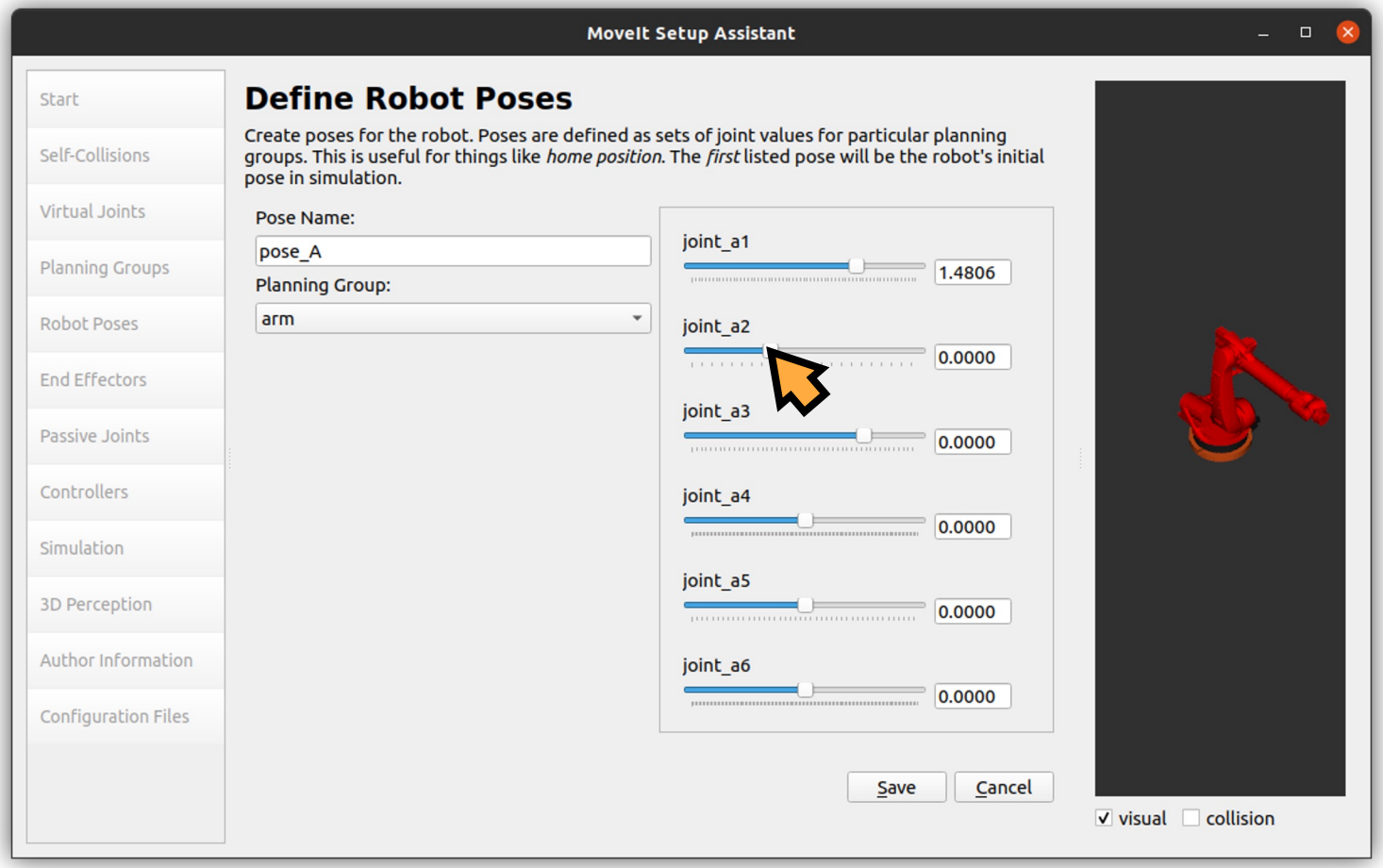
MoveIt Setup Assistant



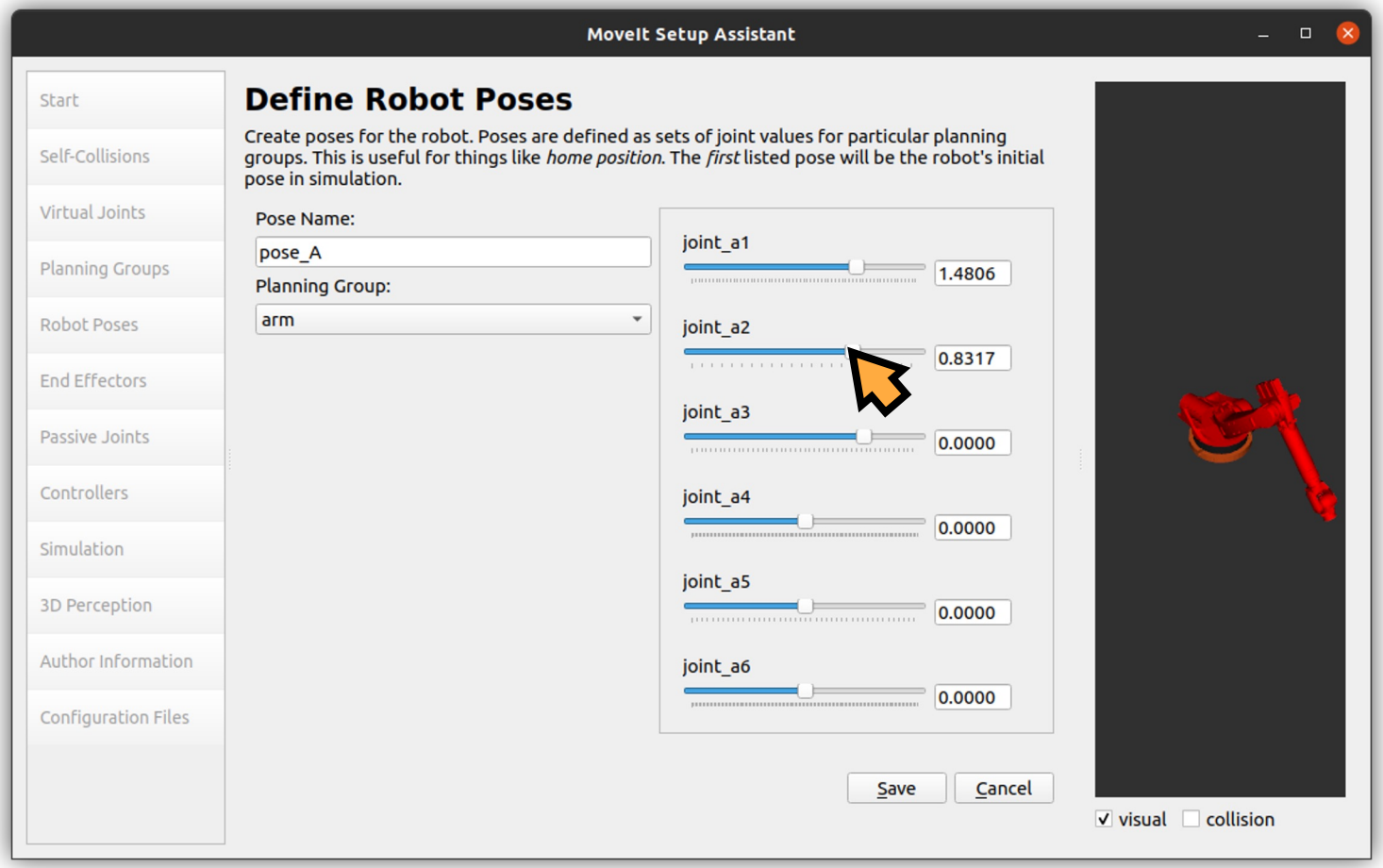
MoveIt Setup Assistant



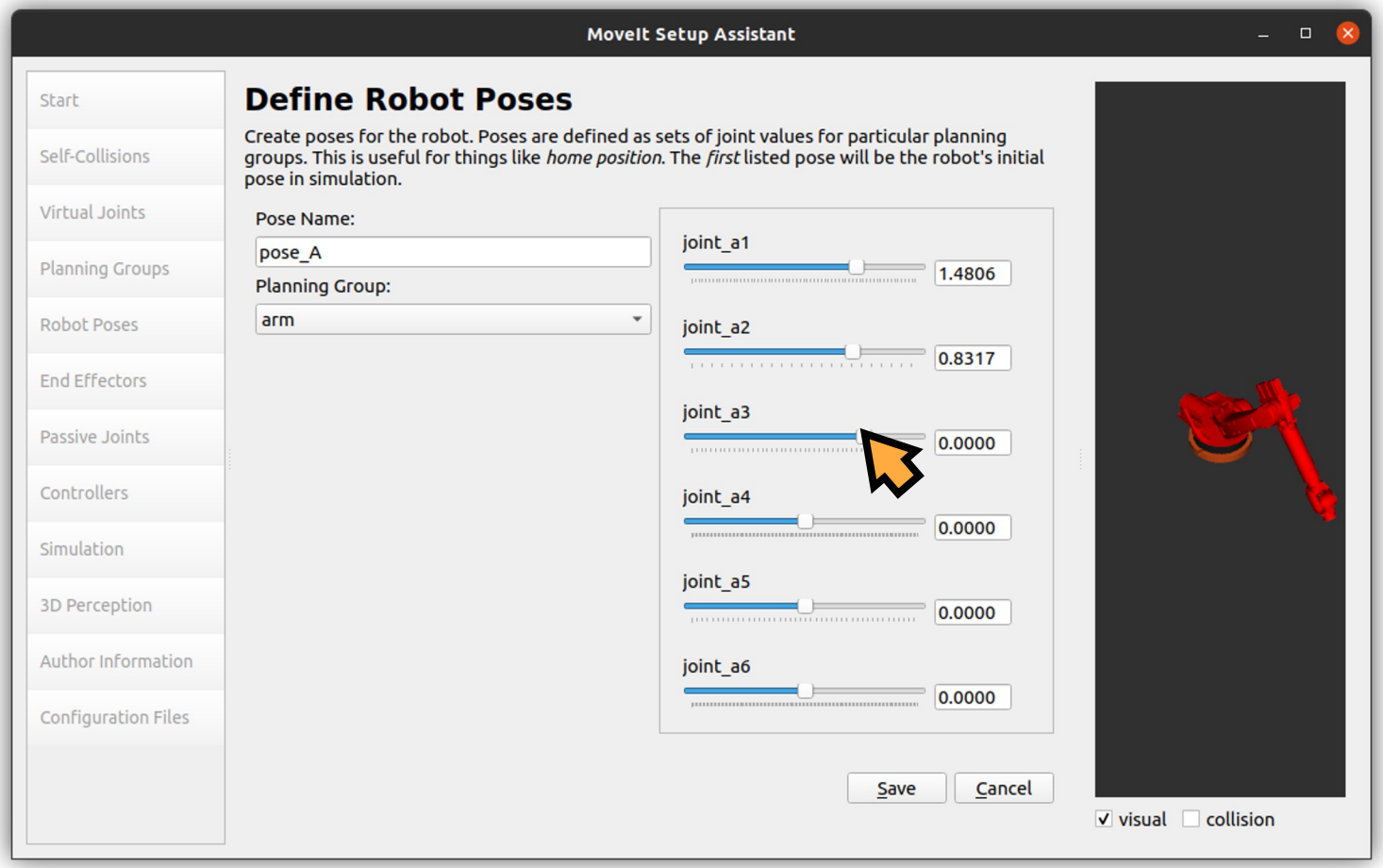
MoveIt Setup Assistant



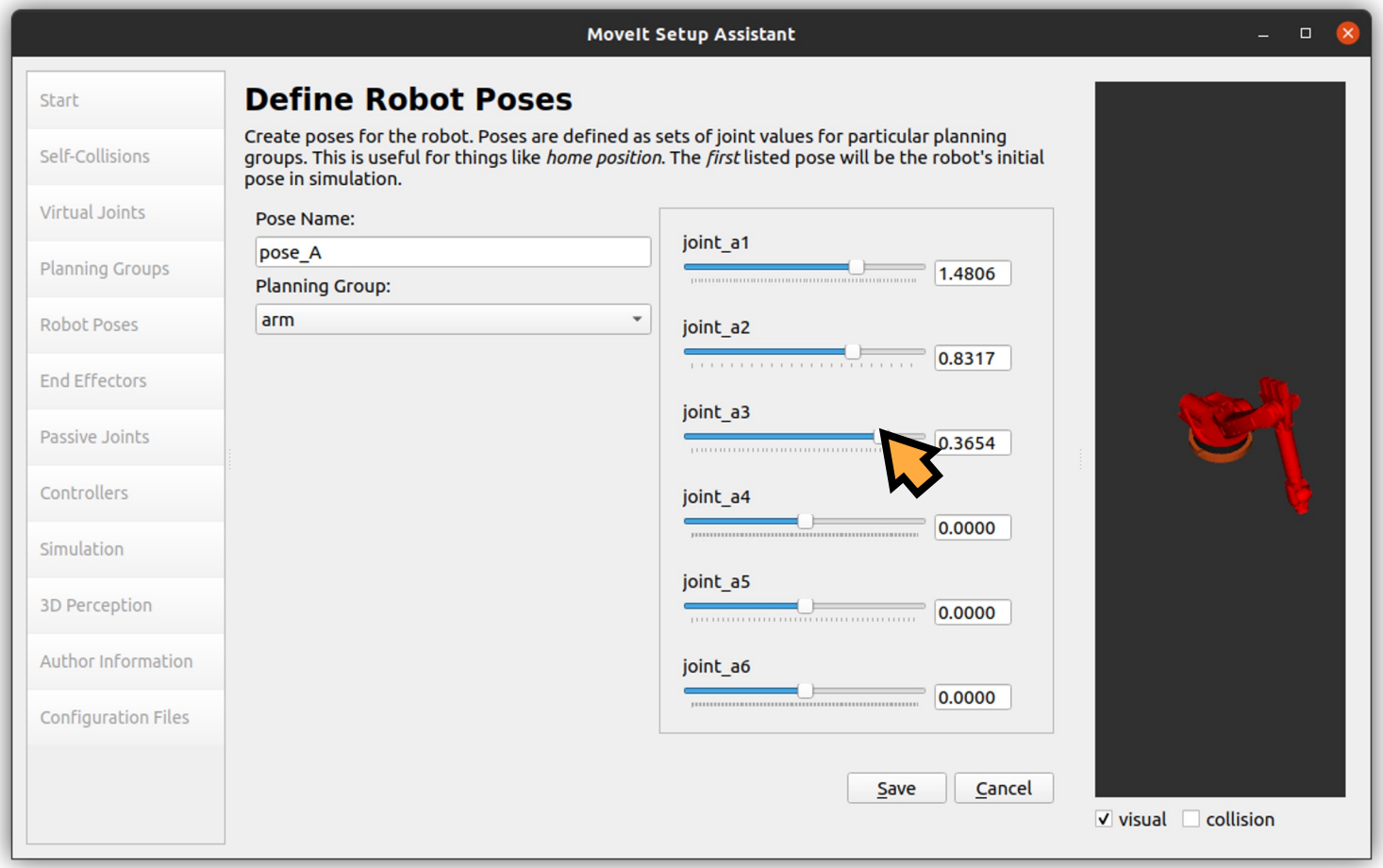
MoveIt Setup Assistant



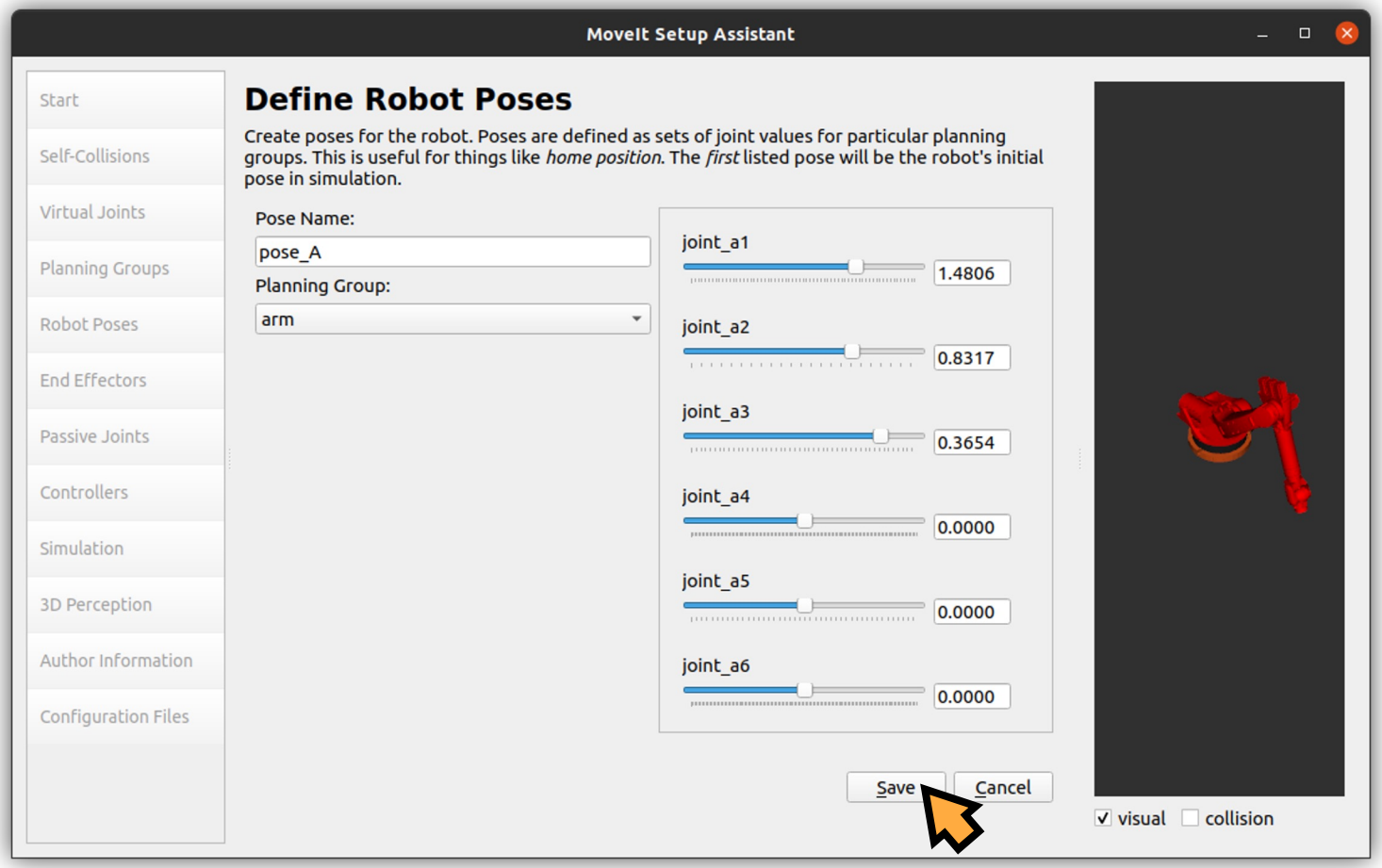
MoveIt Setup Assistant



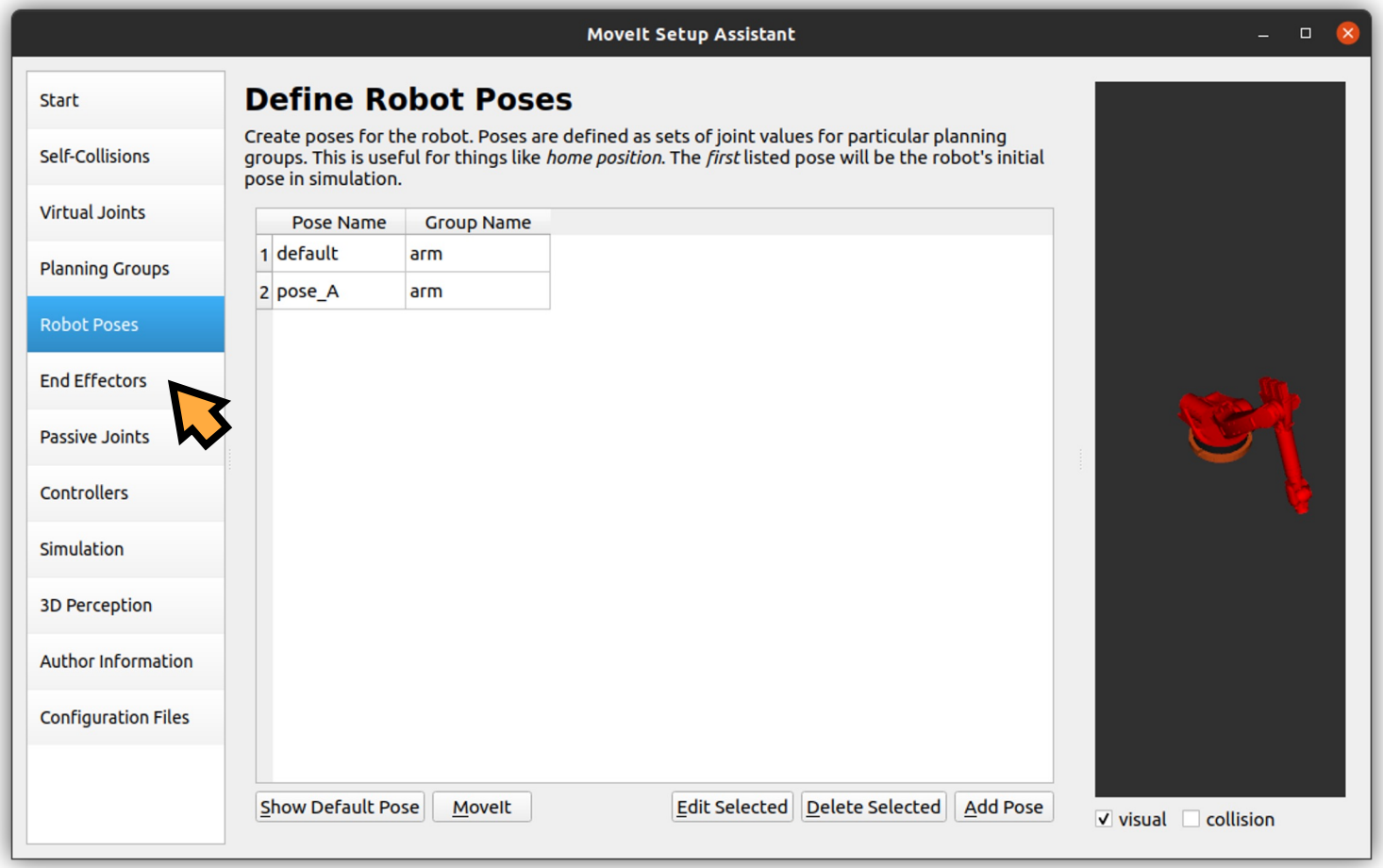
MoveIt Setup Assistant



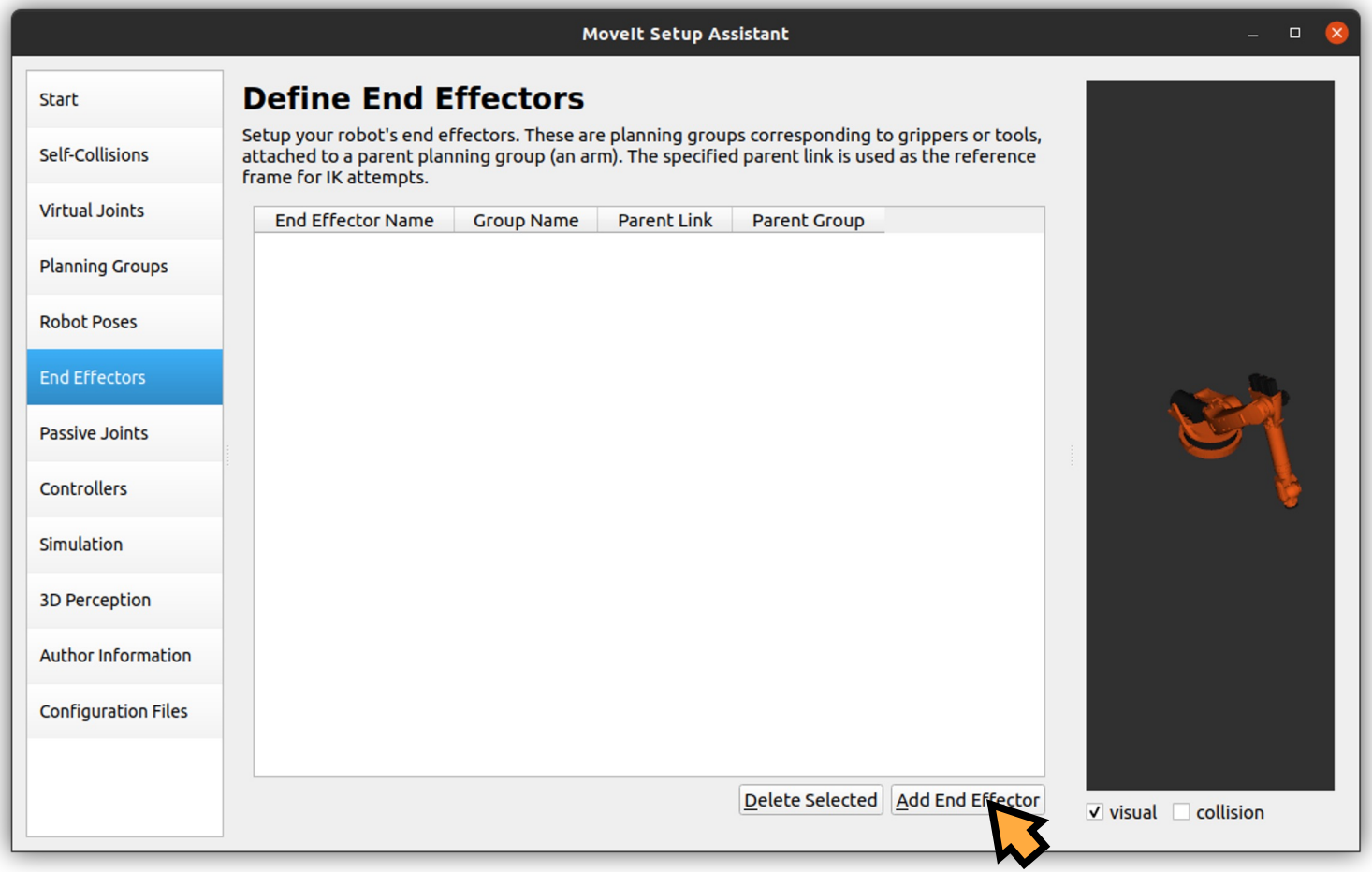
MoveIt Setup Assistant



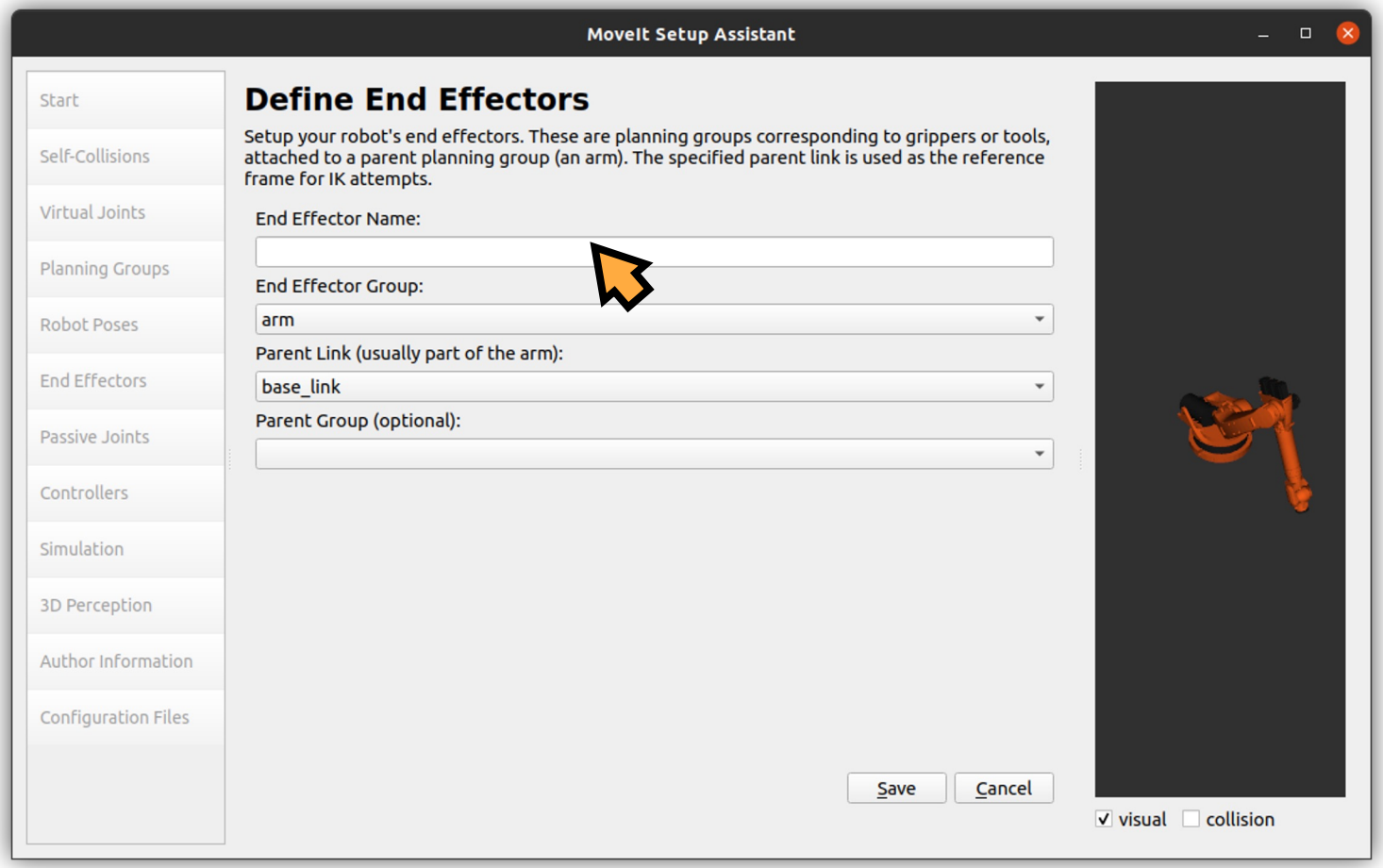
MoveIt Setup Assistant



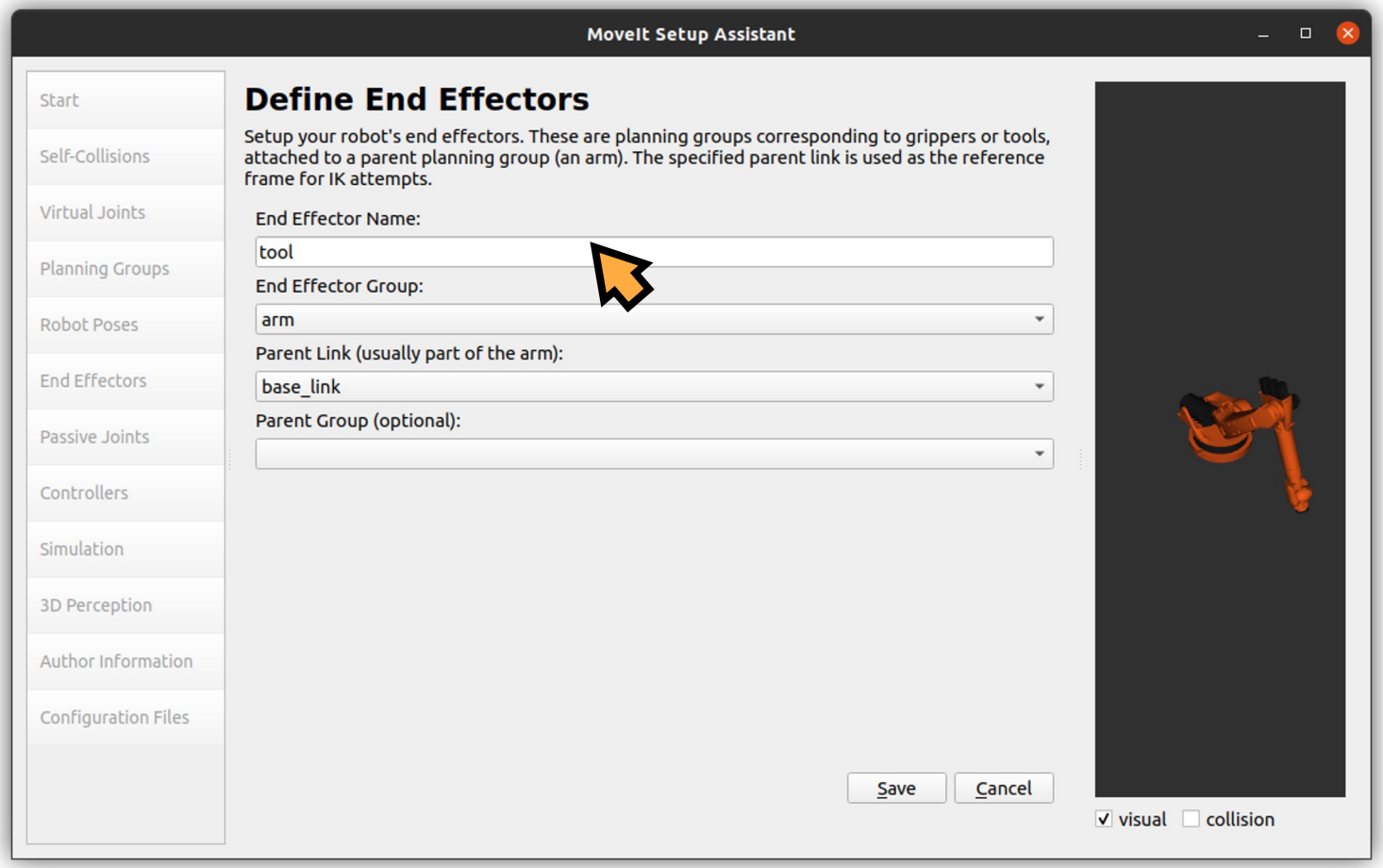
MoveIt Setup Assistant



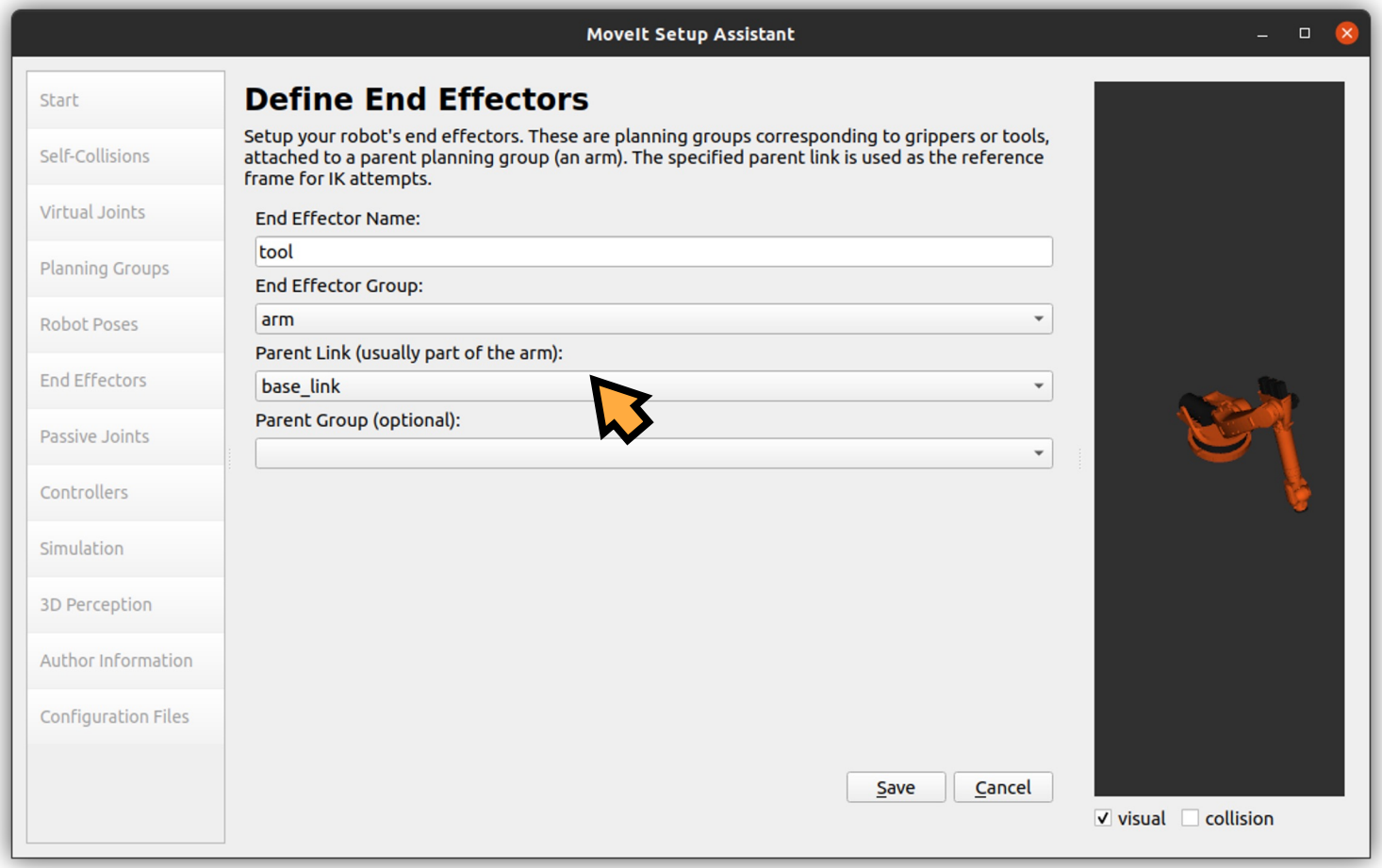
MoveIt Setup Assistant



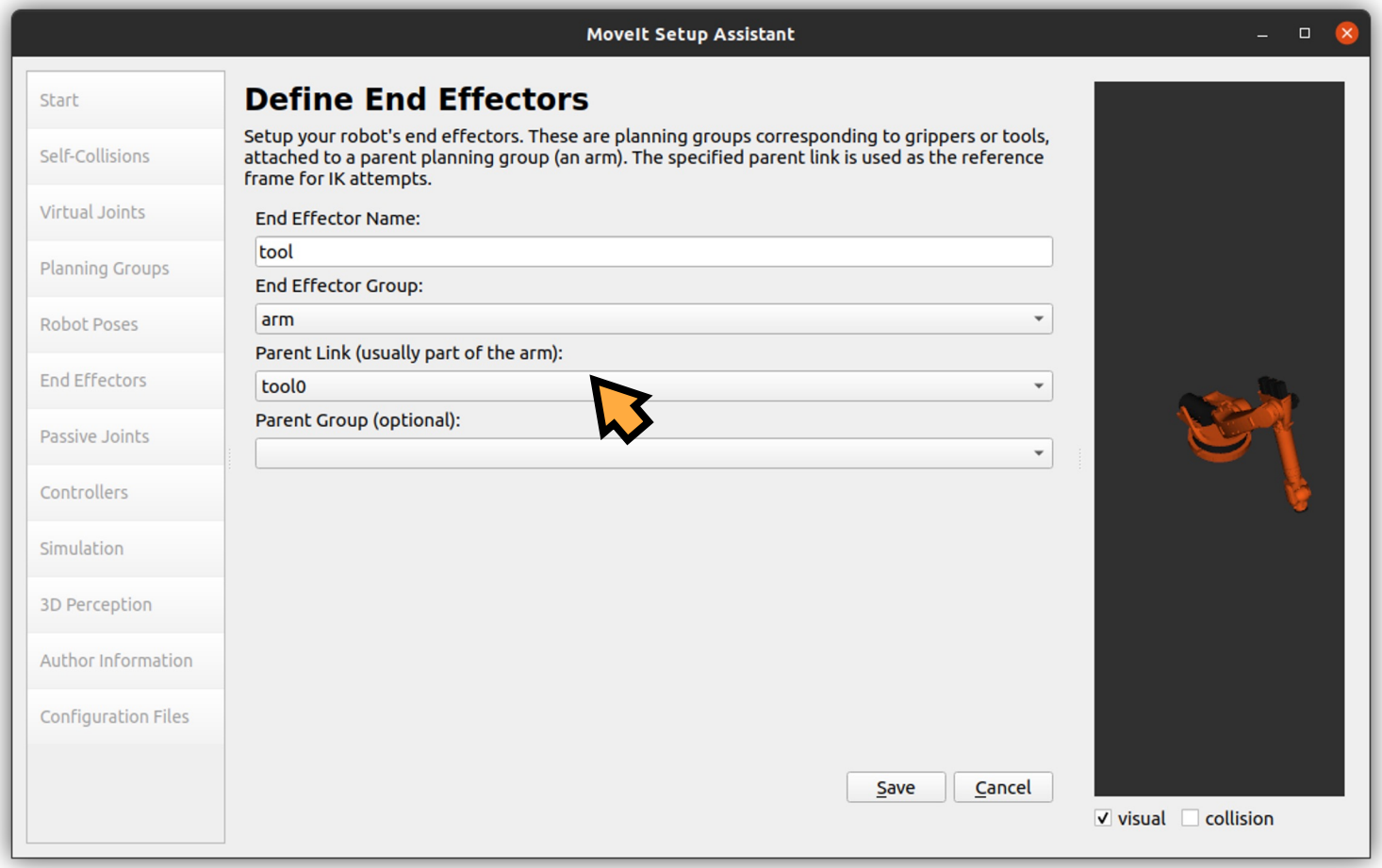
MoveIt Setup Assistant



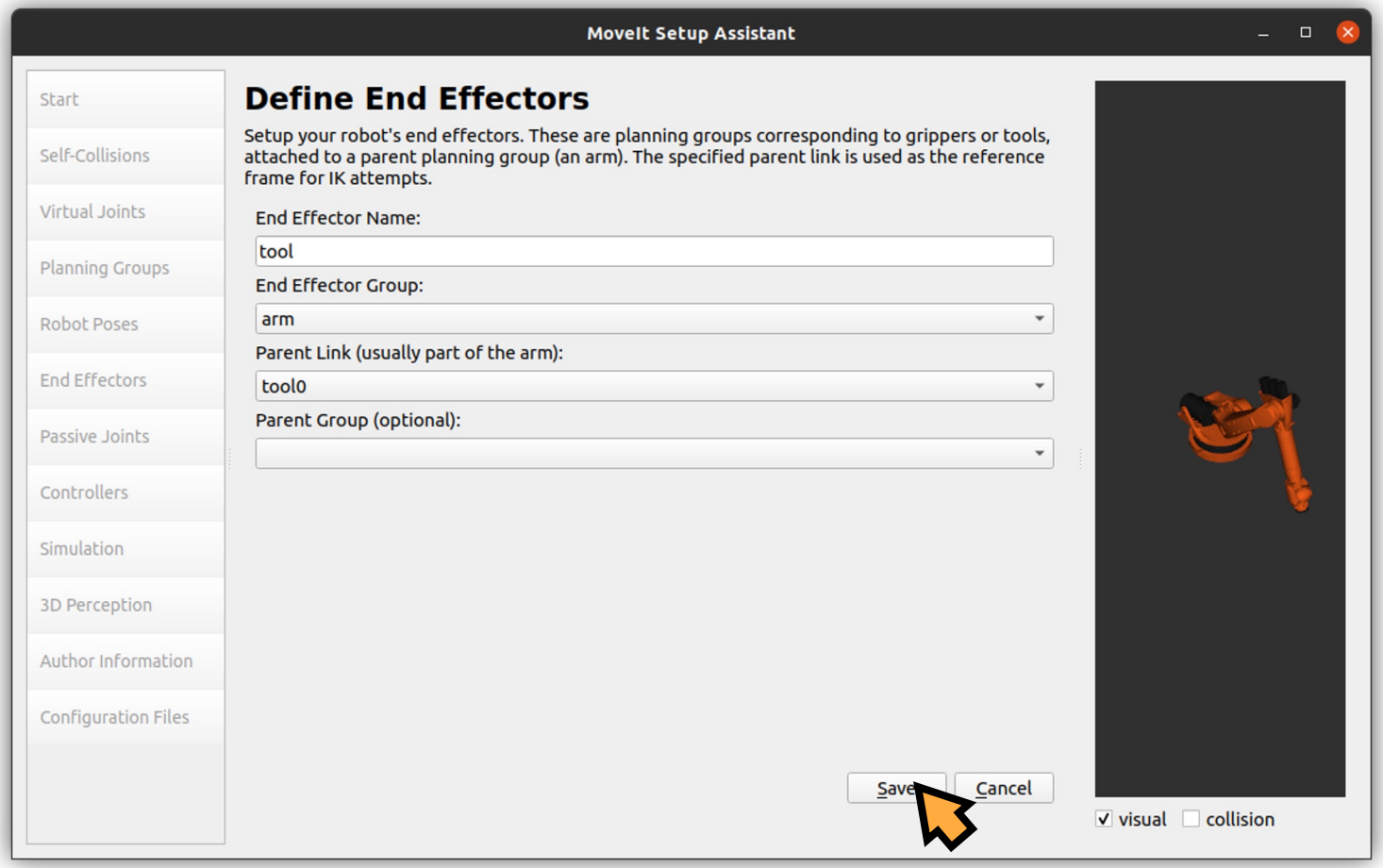
MoveIt Setup Assistant



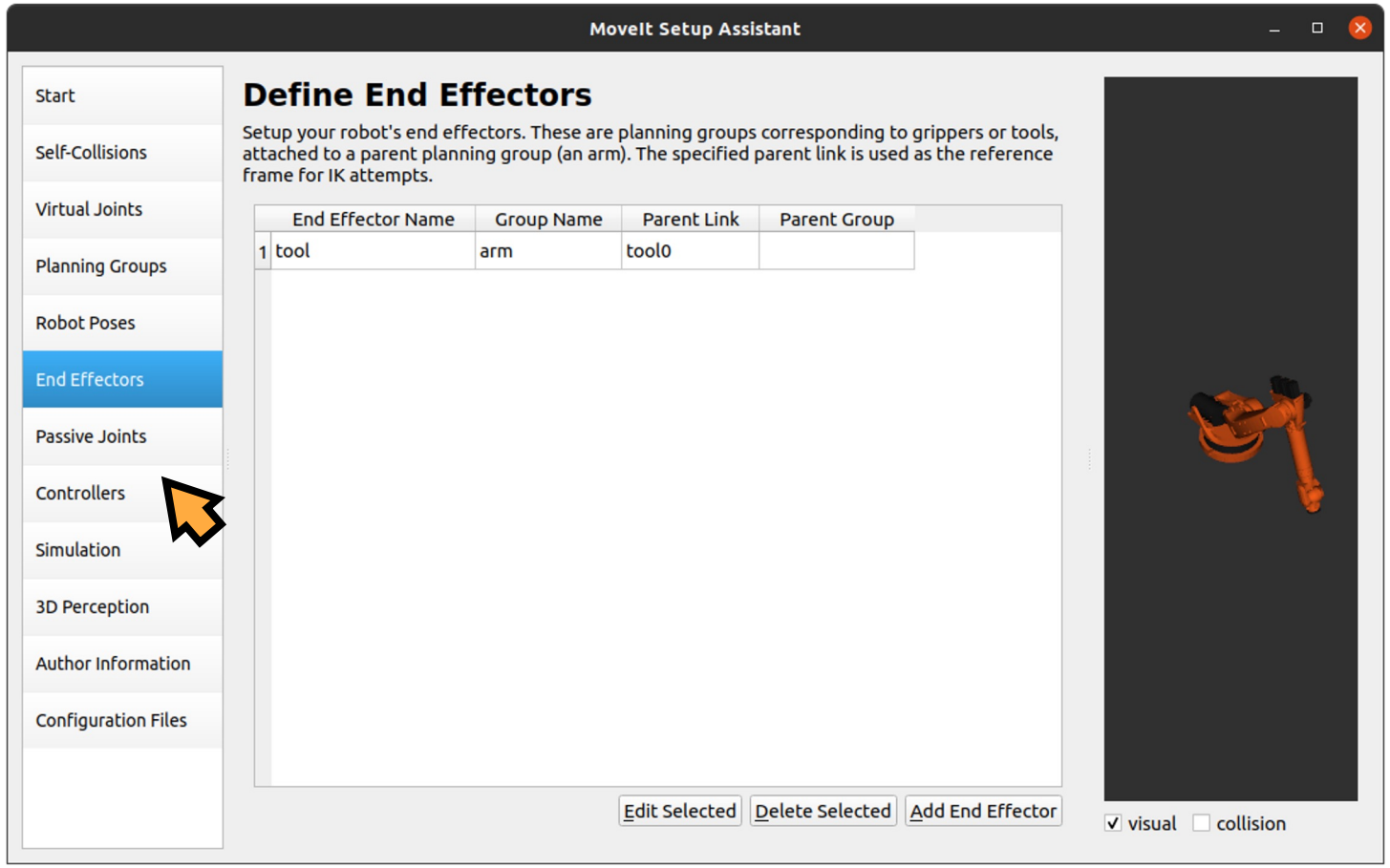
MoveIt Setup Assistant



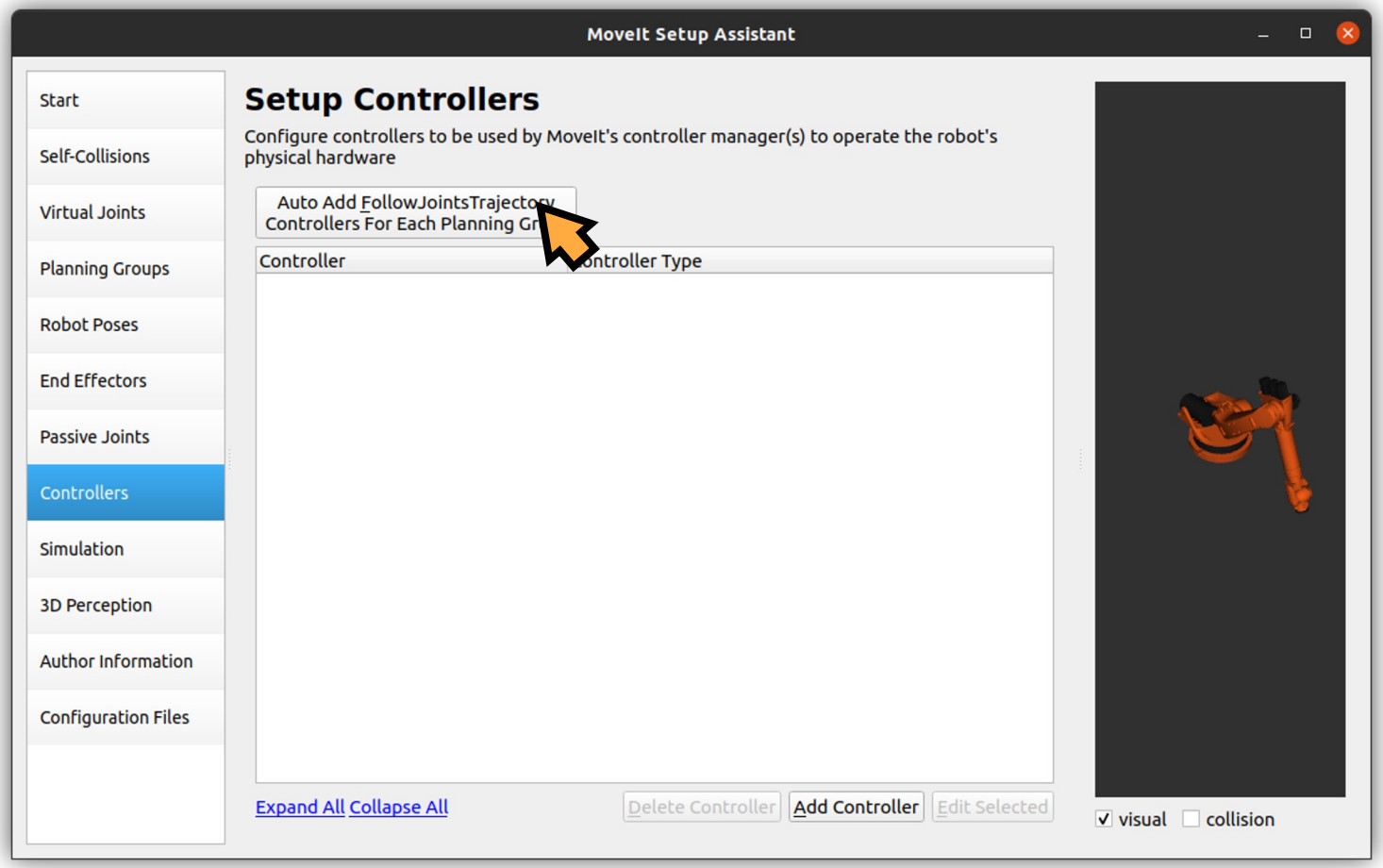
MoveIt Setup Assistant



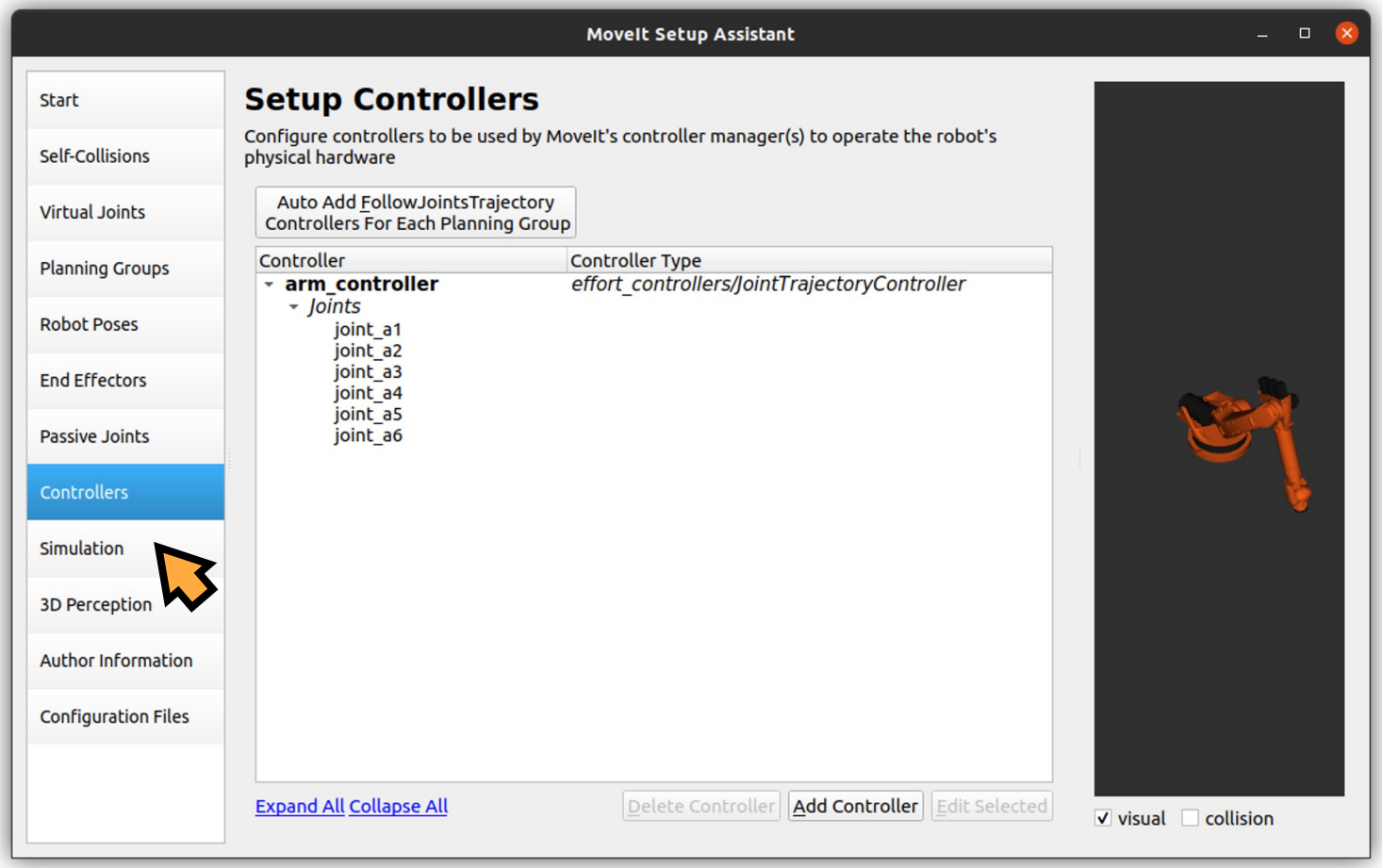
MoveIt Setup Assistant



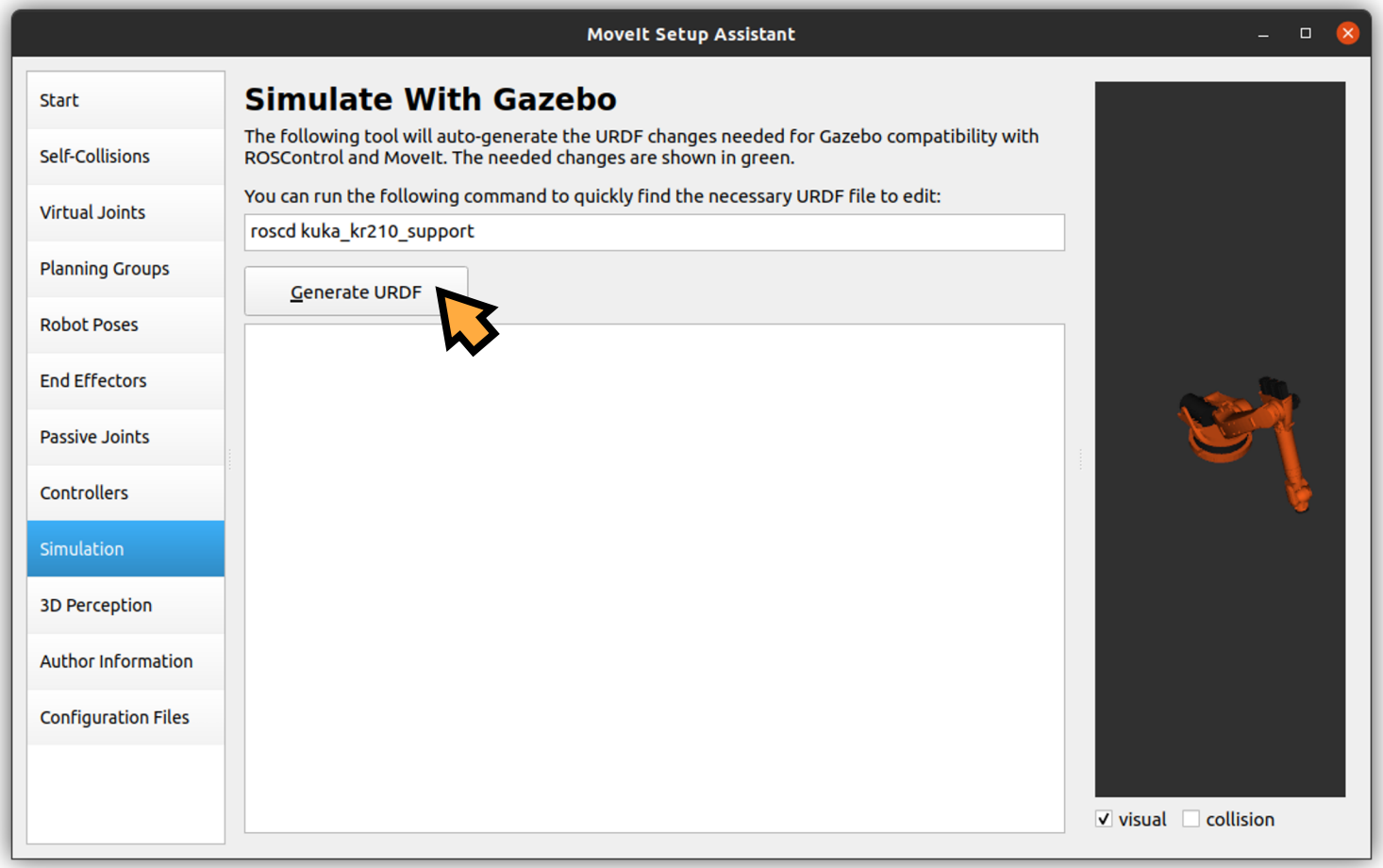
MoveIt Setup Assistant



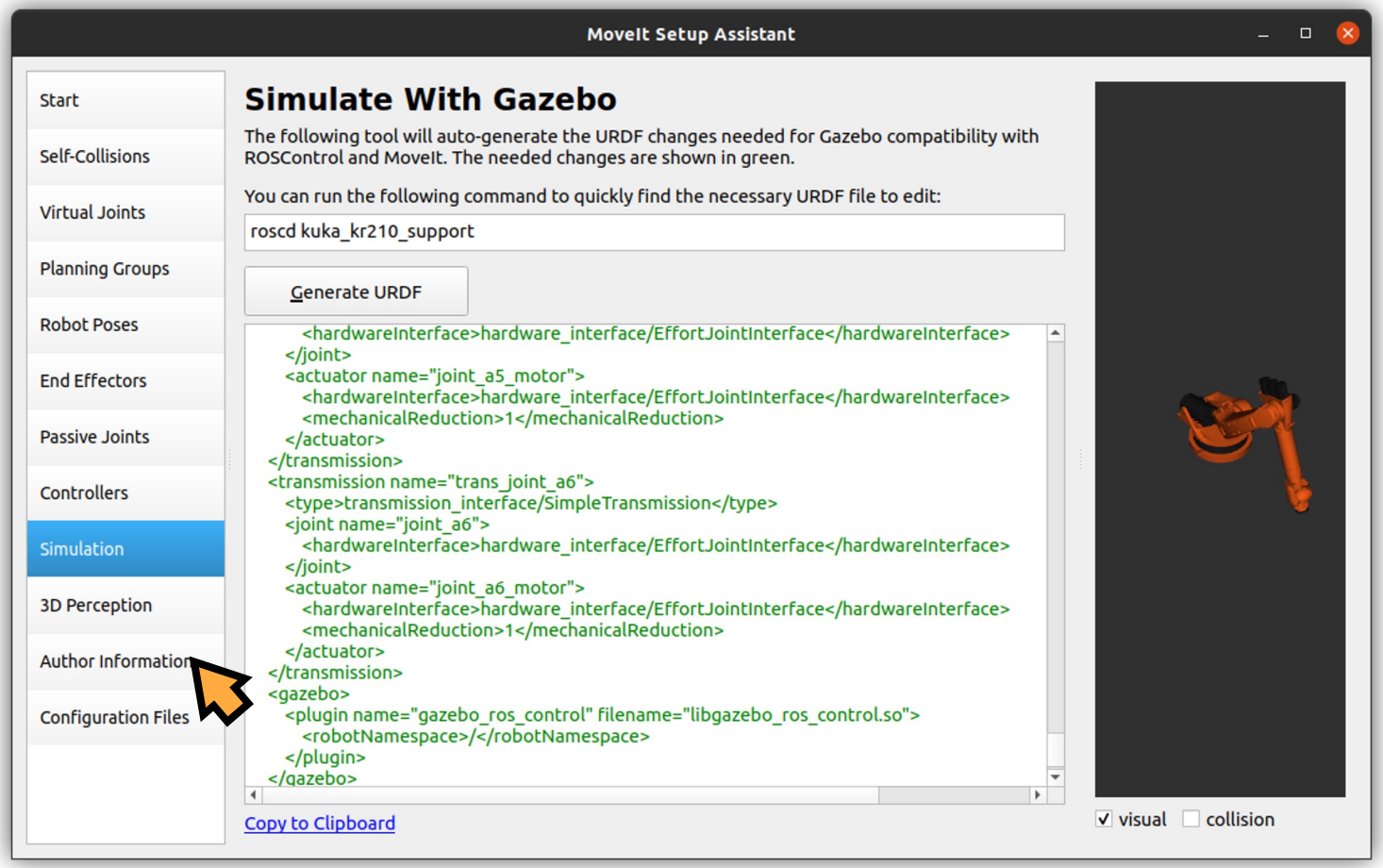
MoveIt Setup Assistant



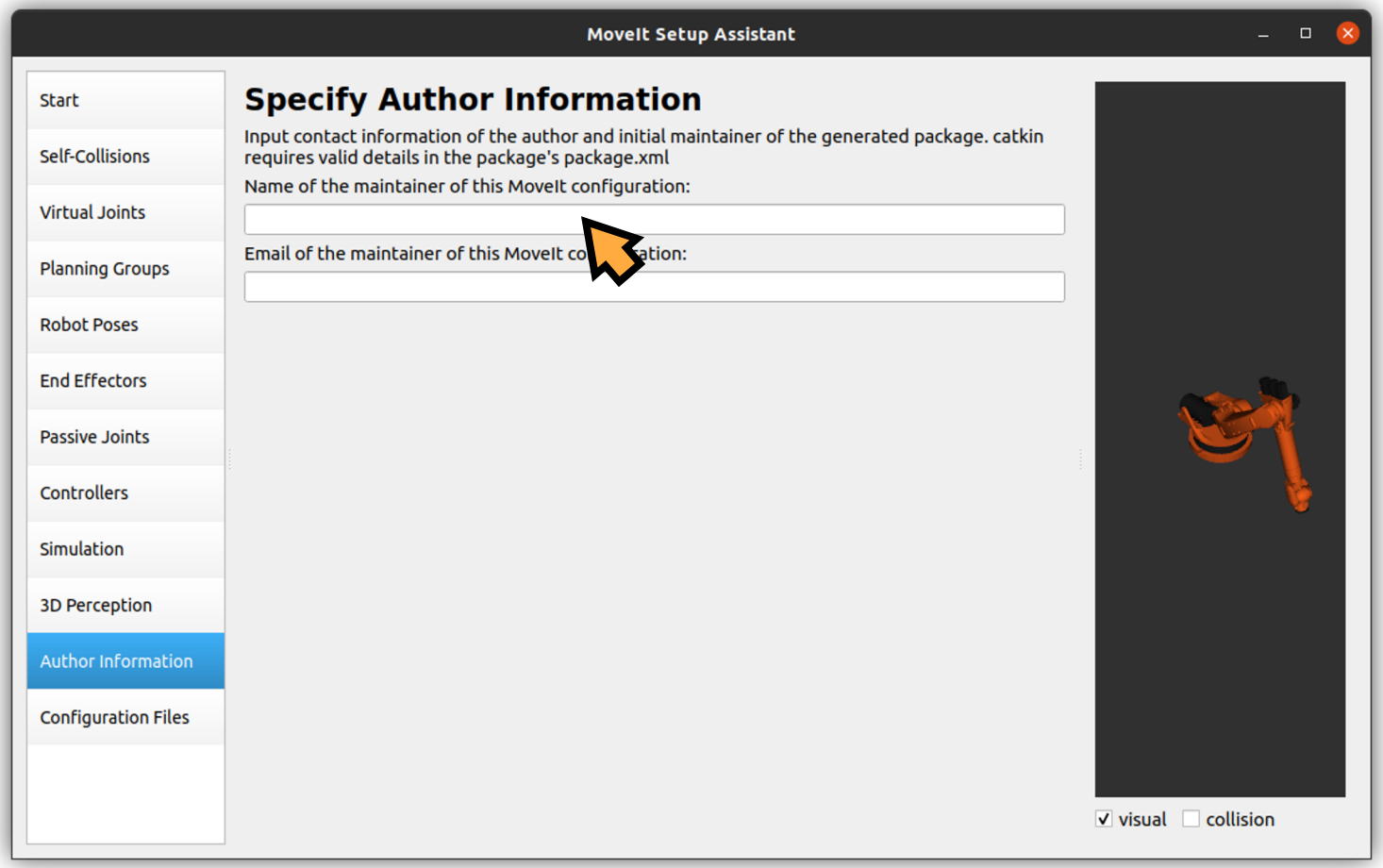
MoveIt Setup Assistant



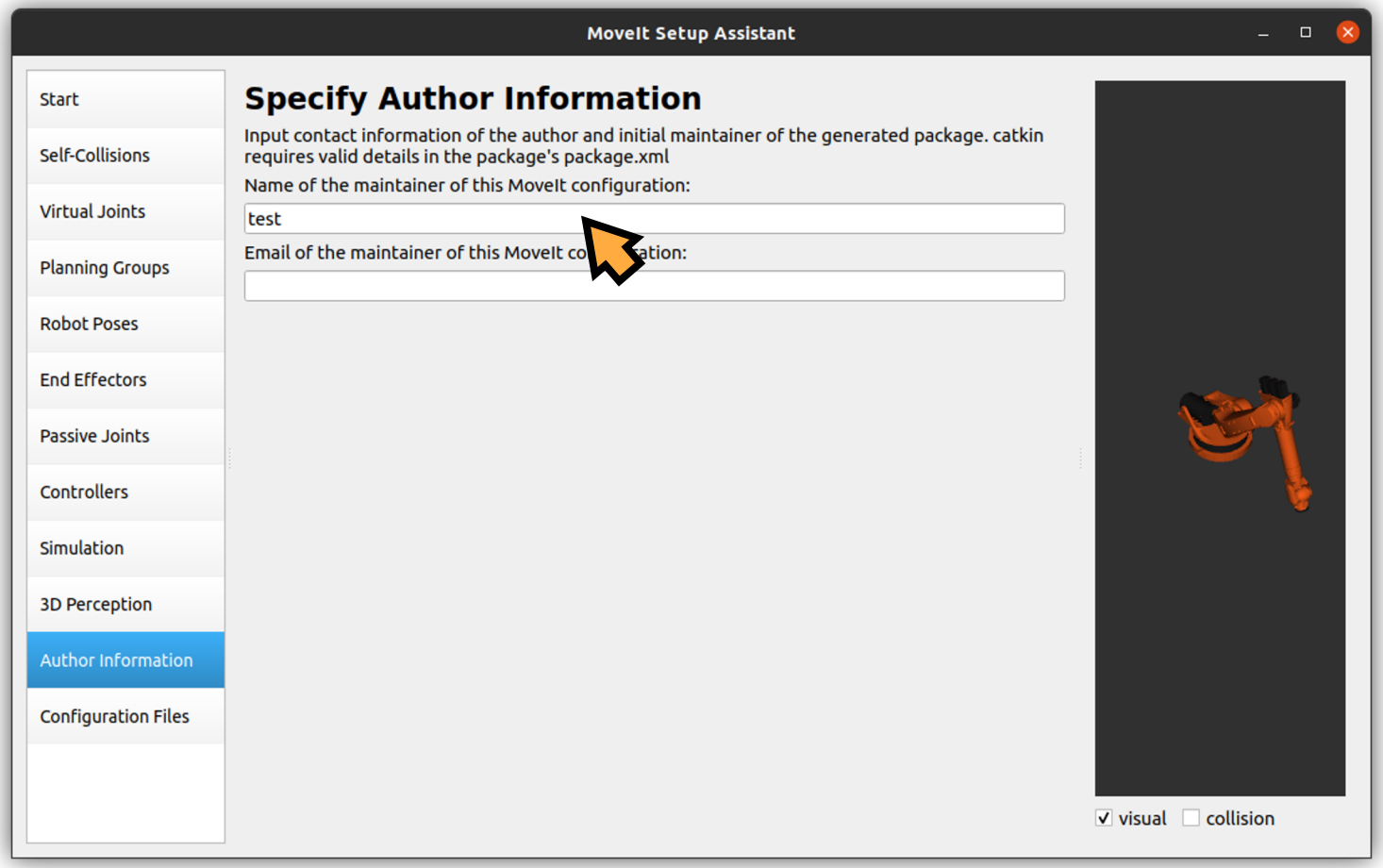
MoveIt Setup Assistant



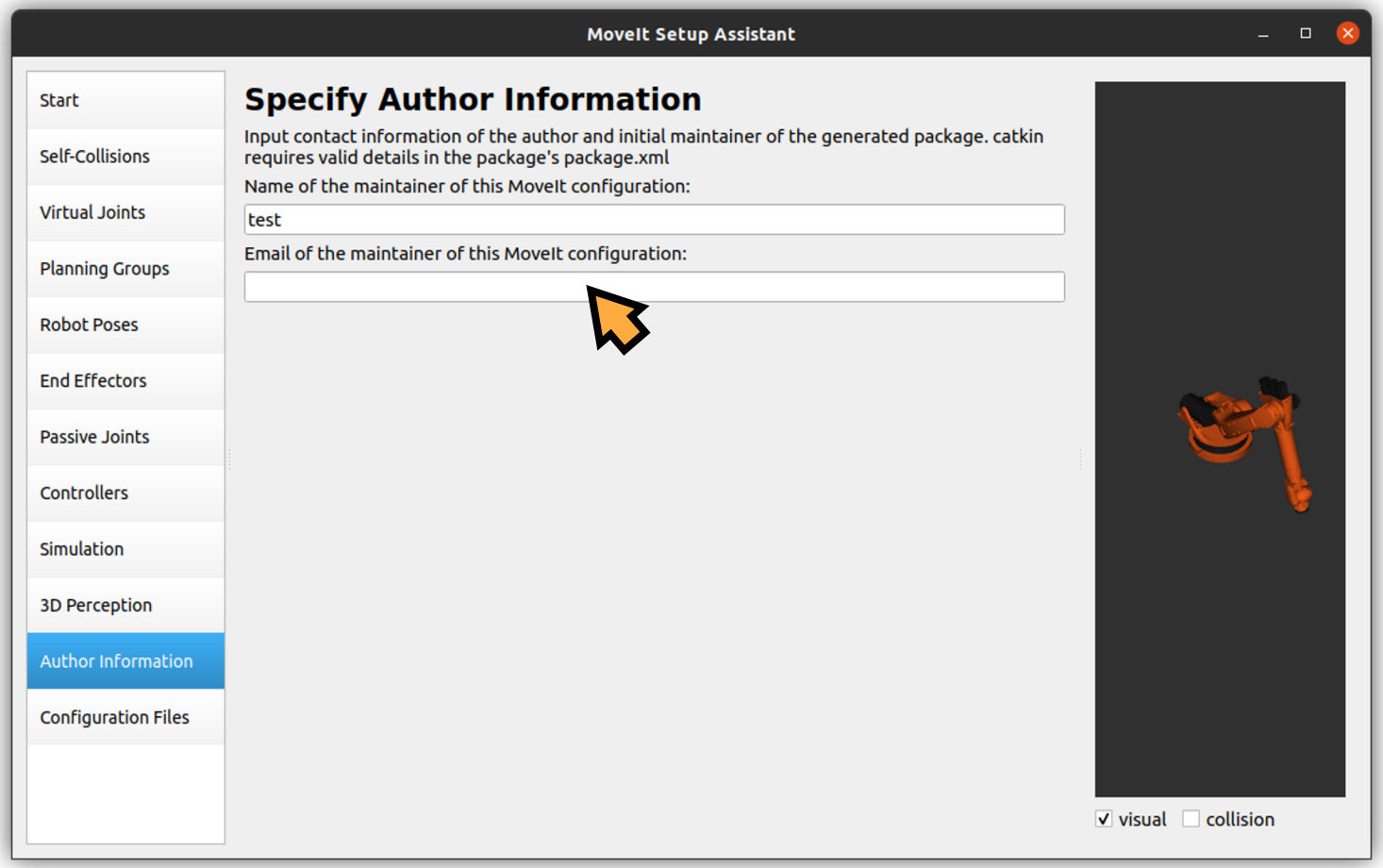
MoveIt Setup Assistant



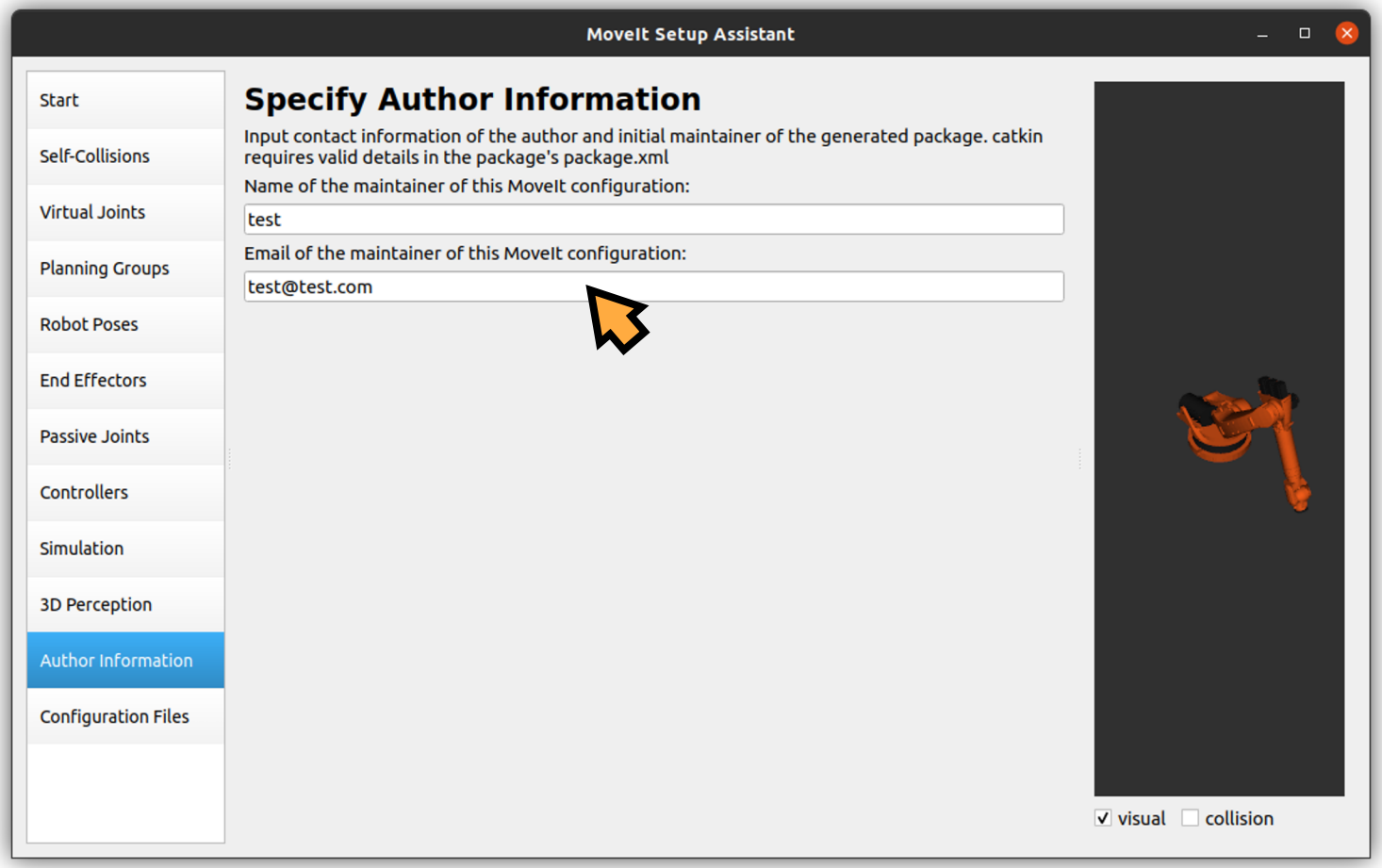
MoveIt Setup Assistant



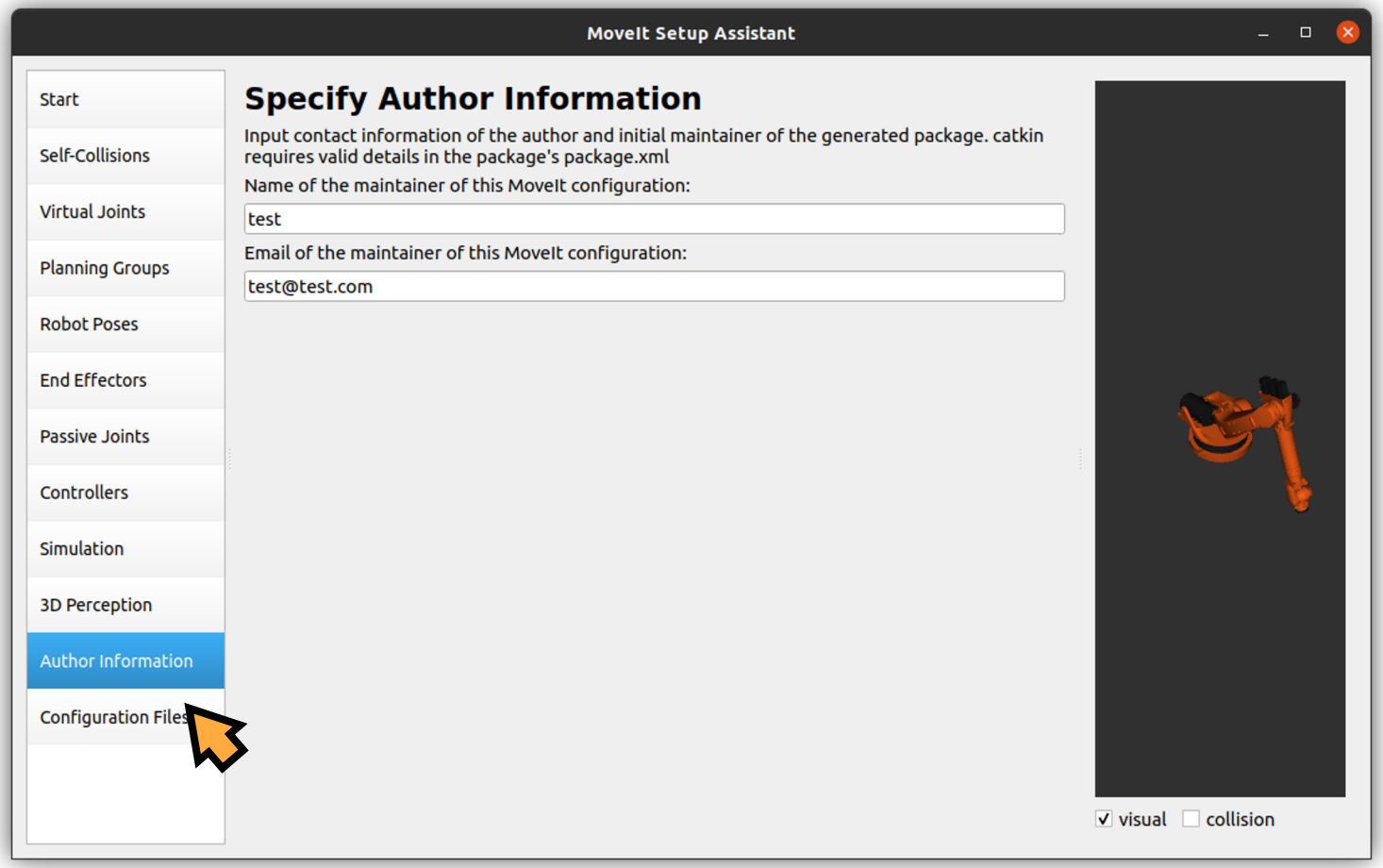
MoveIt Setup Assistant



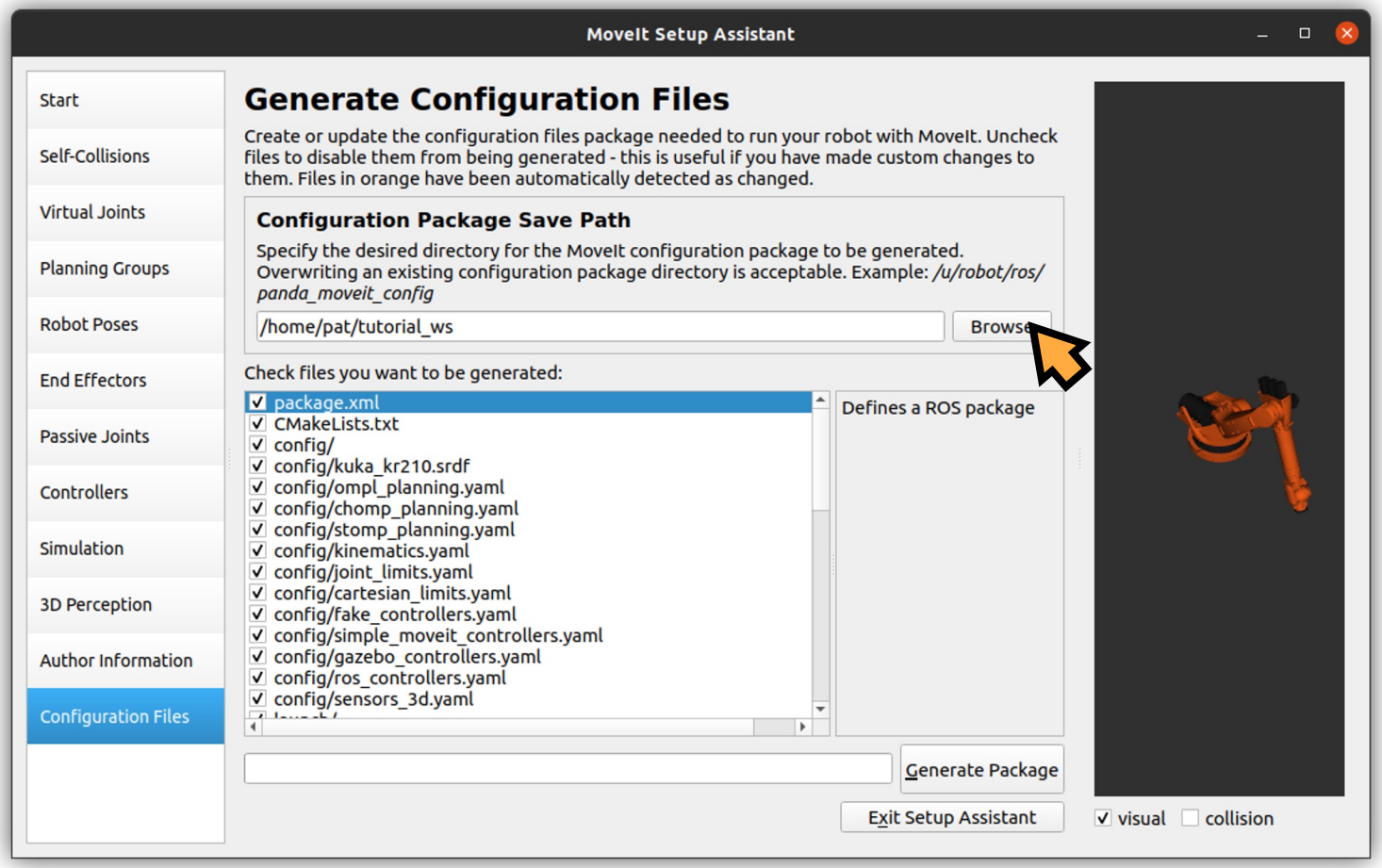
MoveIt Setup Assistant



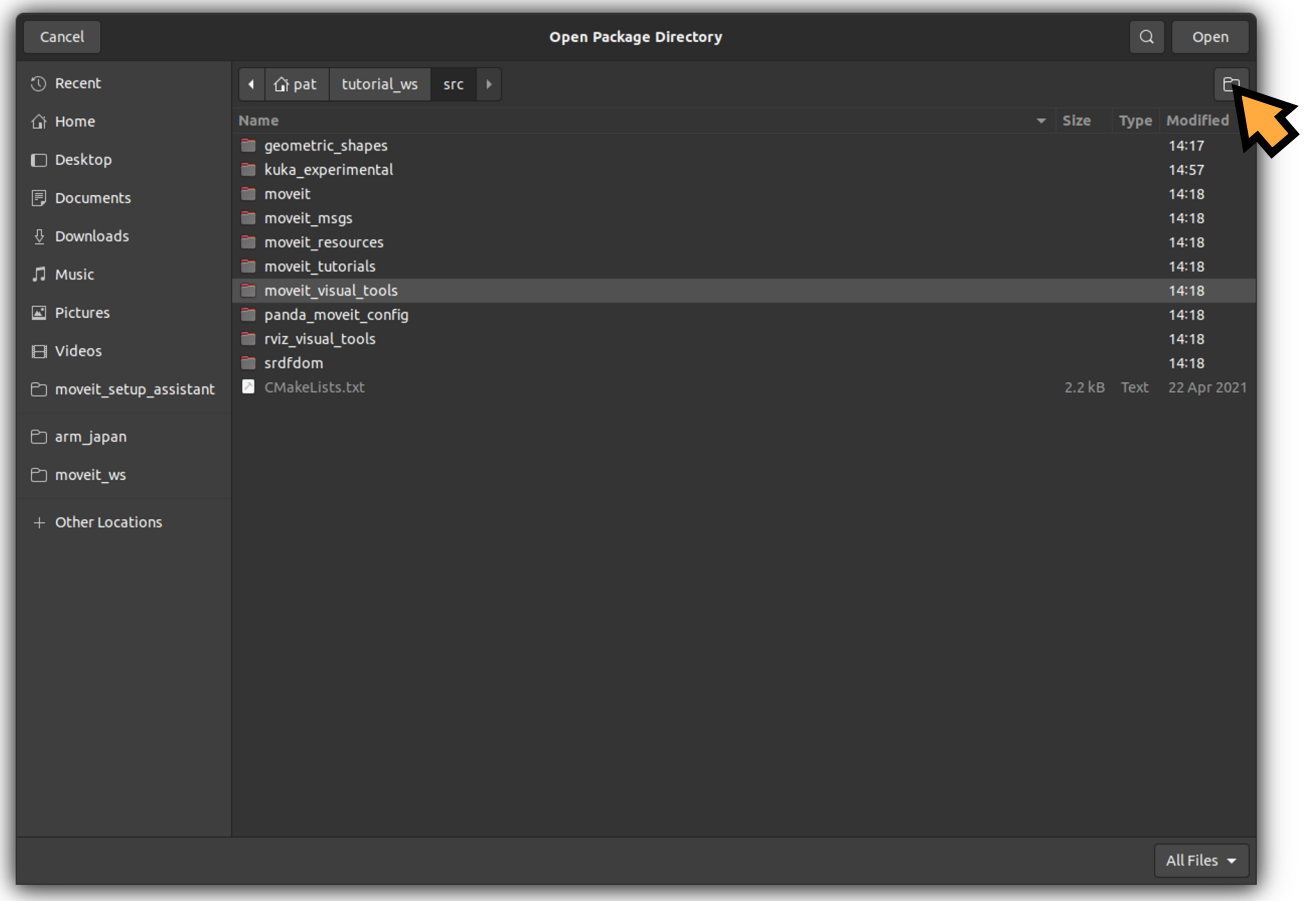
MoveIt Setup Assistant



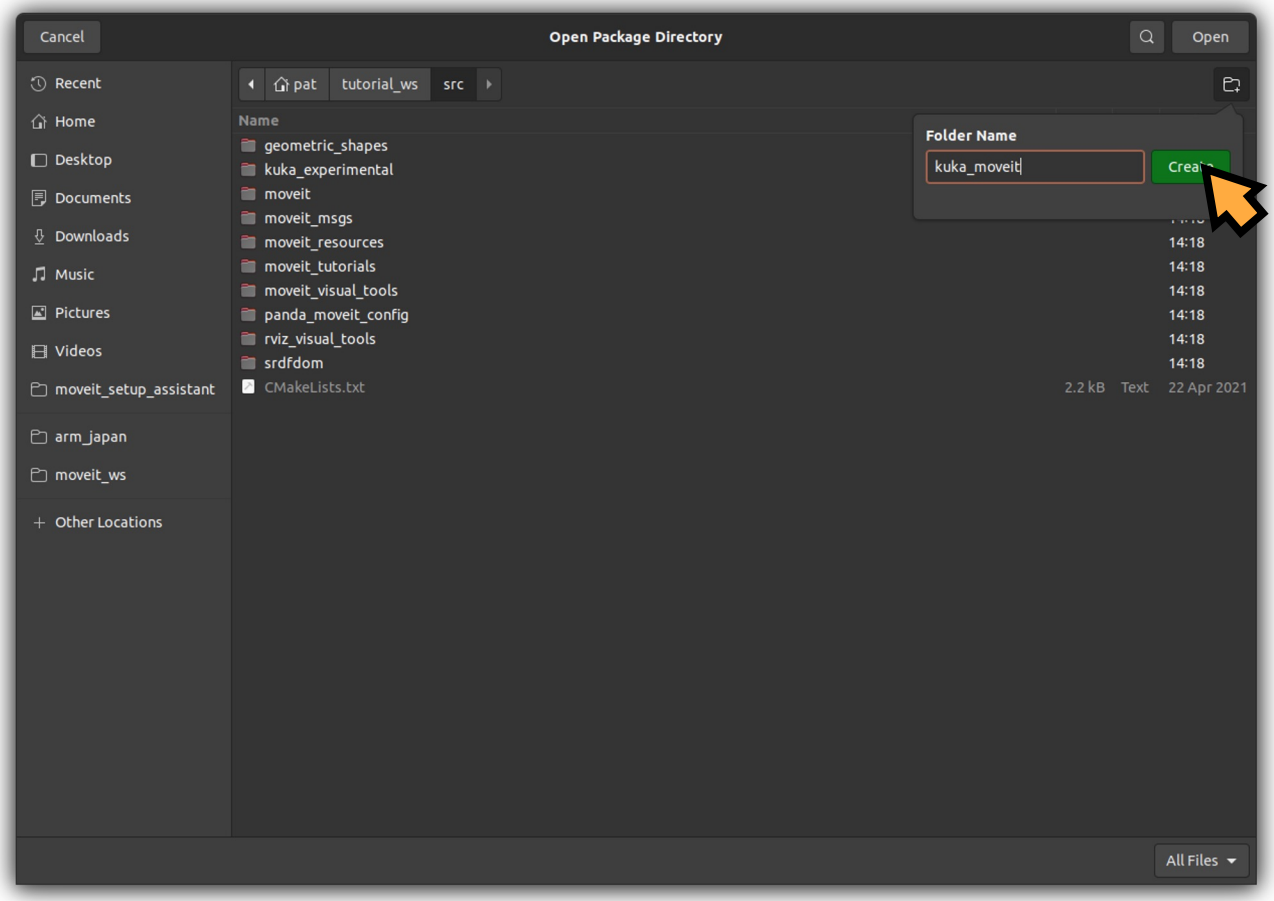
MoveIt Setup Assistant



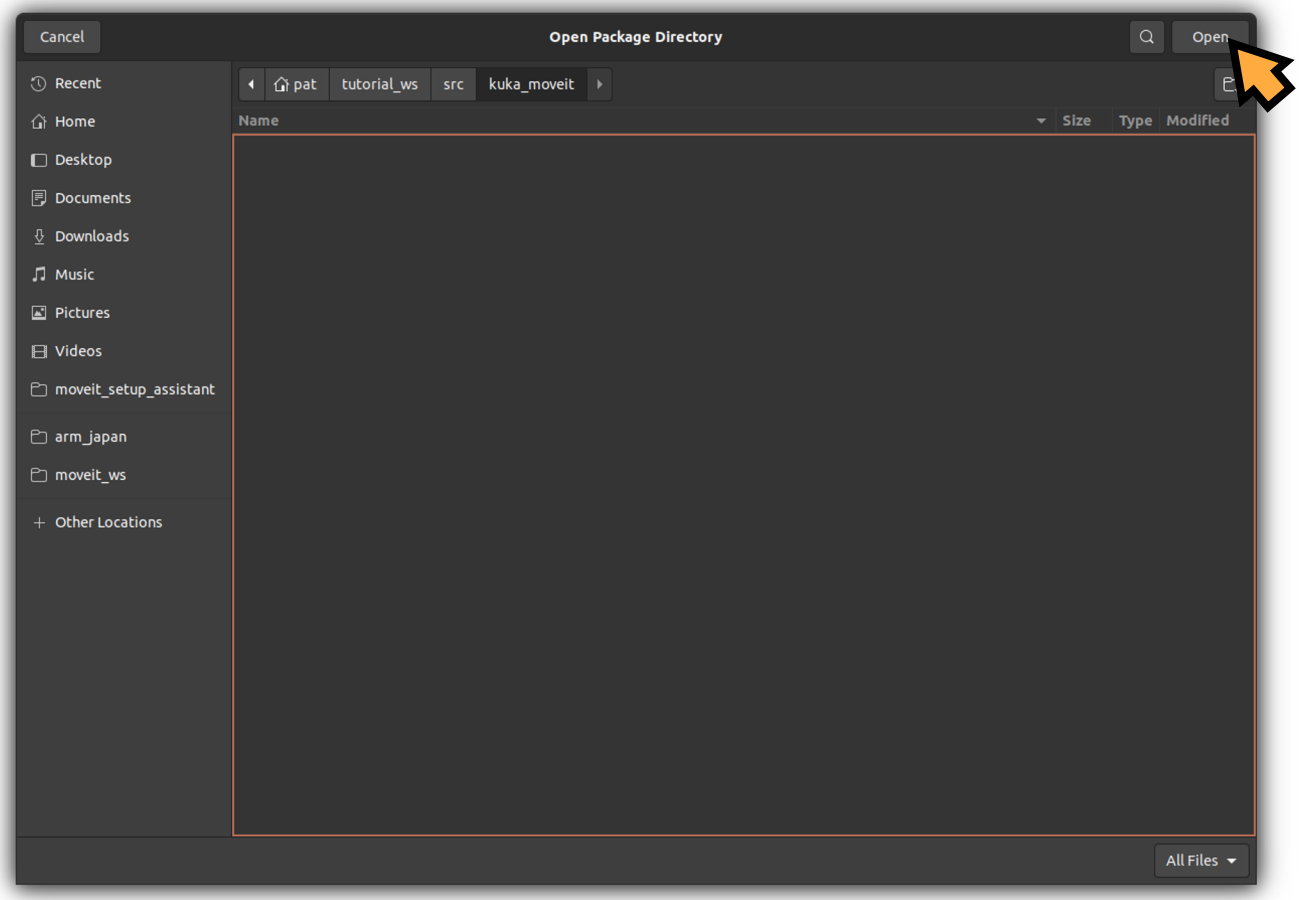
MoveIt Setup Assistant



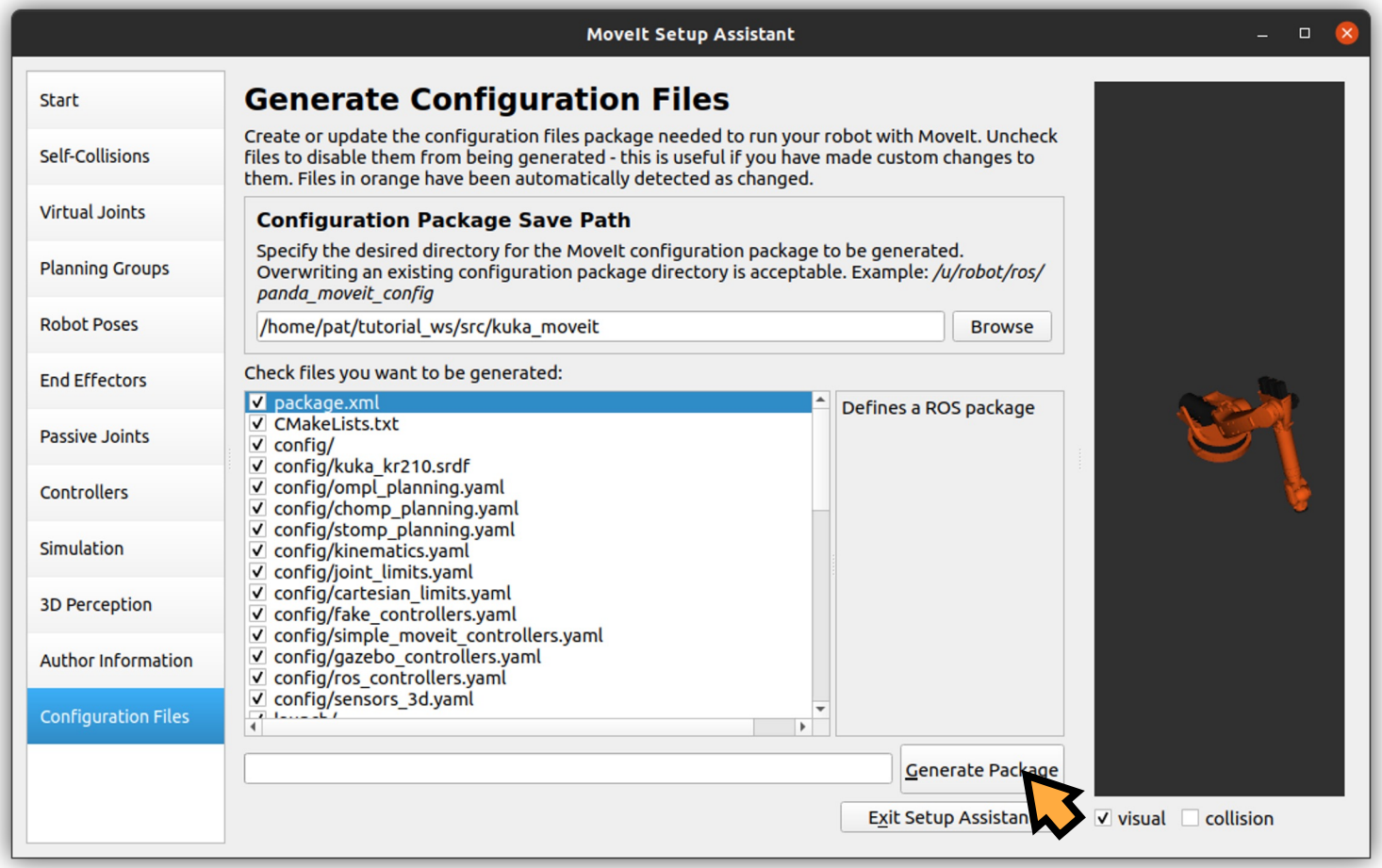
MoveIt Setup Assistant



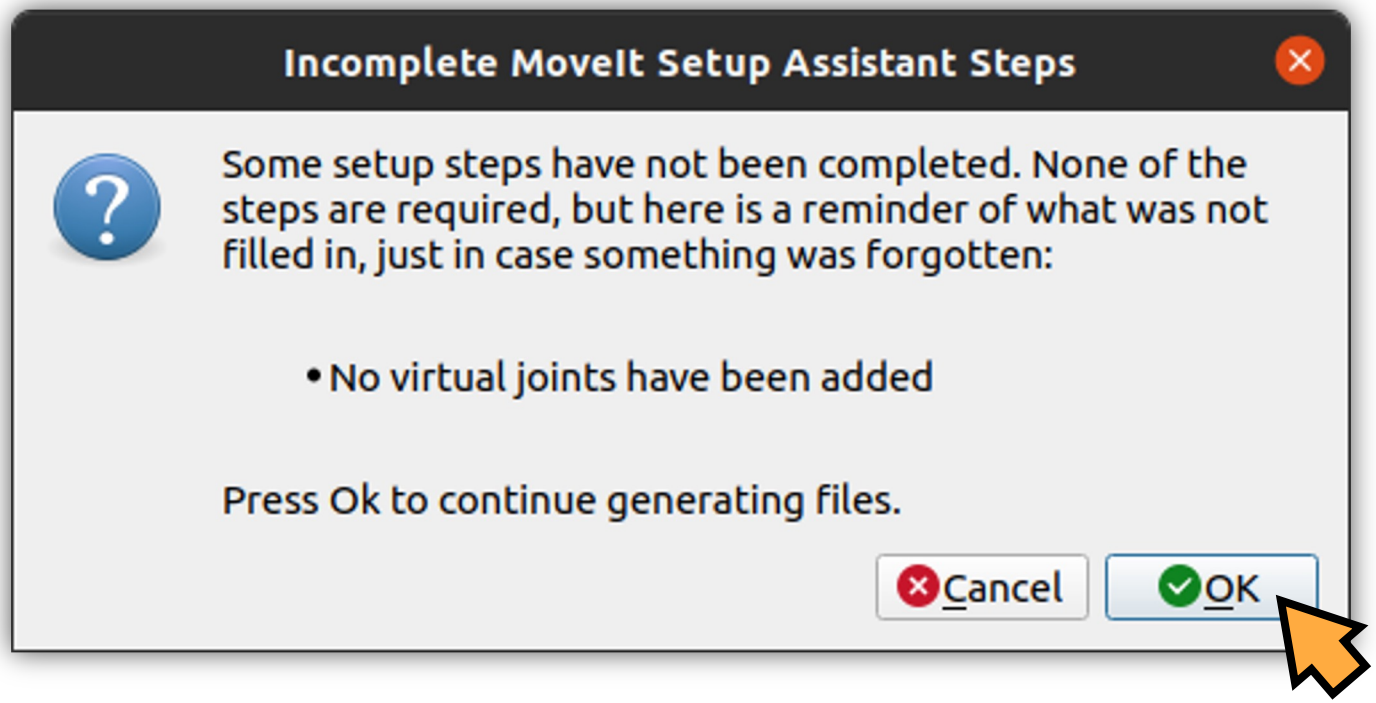
MoveIt Setup Assistant



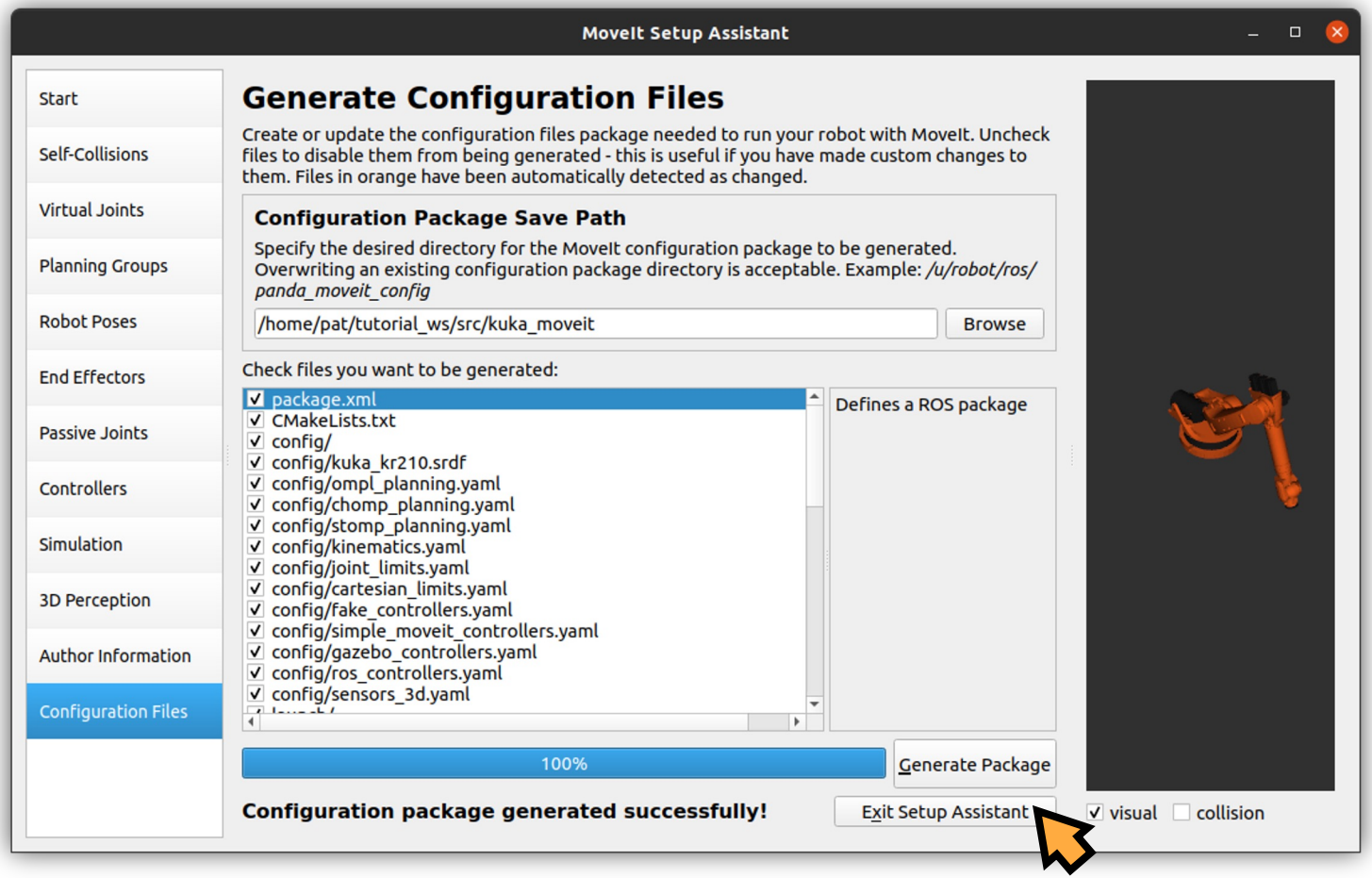
MoveIt Setup Assistant



MoveIt Setup Assistant



MoveIt Setup Assistant



MoveIt Setup Assistant

```
$ cd tutorial_ws/
```

MoveIt Setup Assistant

```
$ catkin_make
```

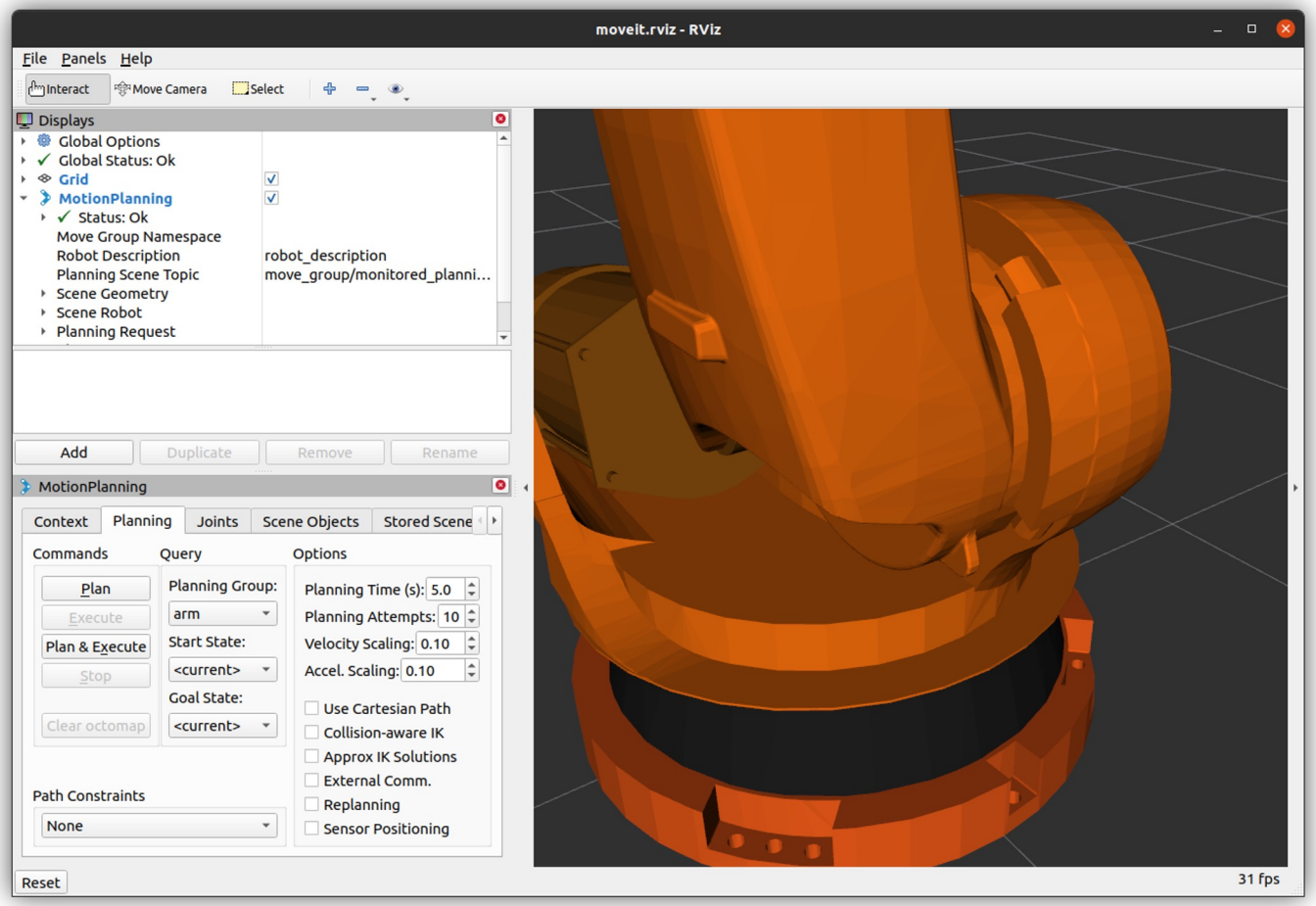
MoveIt Setup Assistant

```
$ rospack profile
```

MoveIt Setup Assistant

```
$ roslaunch kuka_moveit demo.launch
```

MoveIt Setup Assistant



Solve by inverse kinematics

Using inverse kinematics

120

Fixed posture

MoveIt Commander Scripting

Using commander scripting with Moveit.

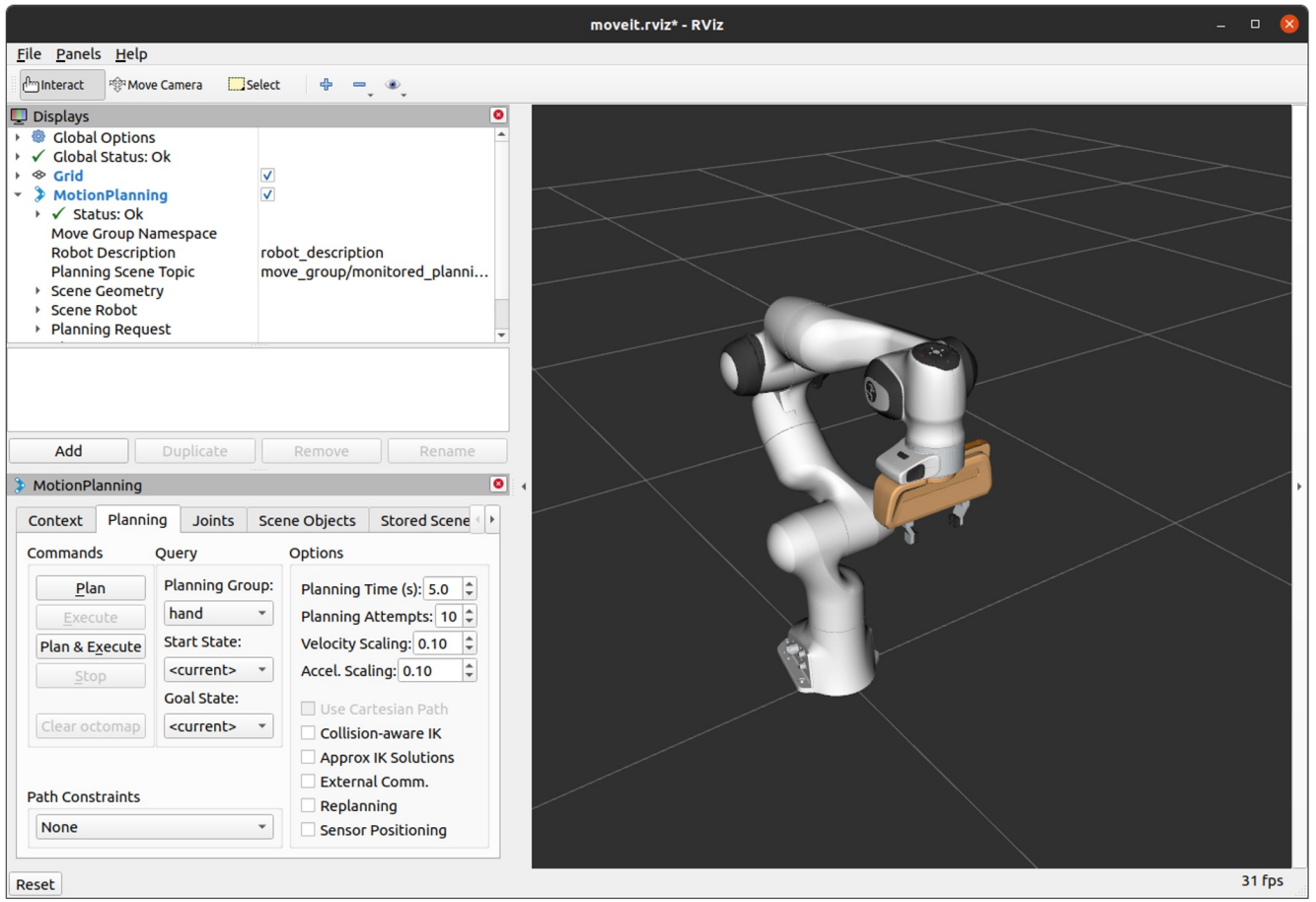
MoveIt Commander Scripting

```
$ roslaunch panda_moveit_config demo.launch
```

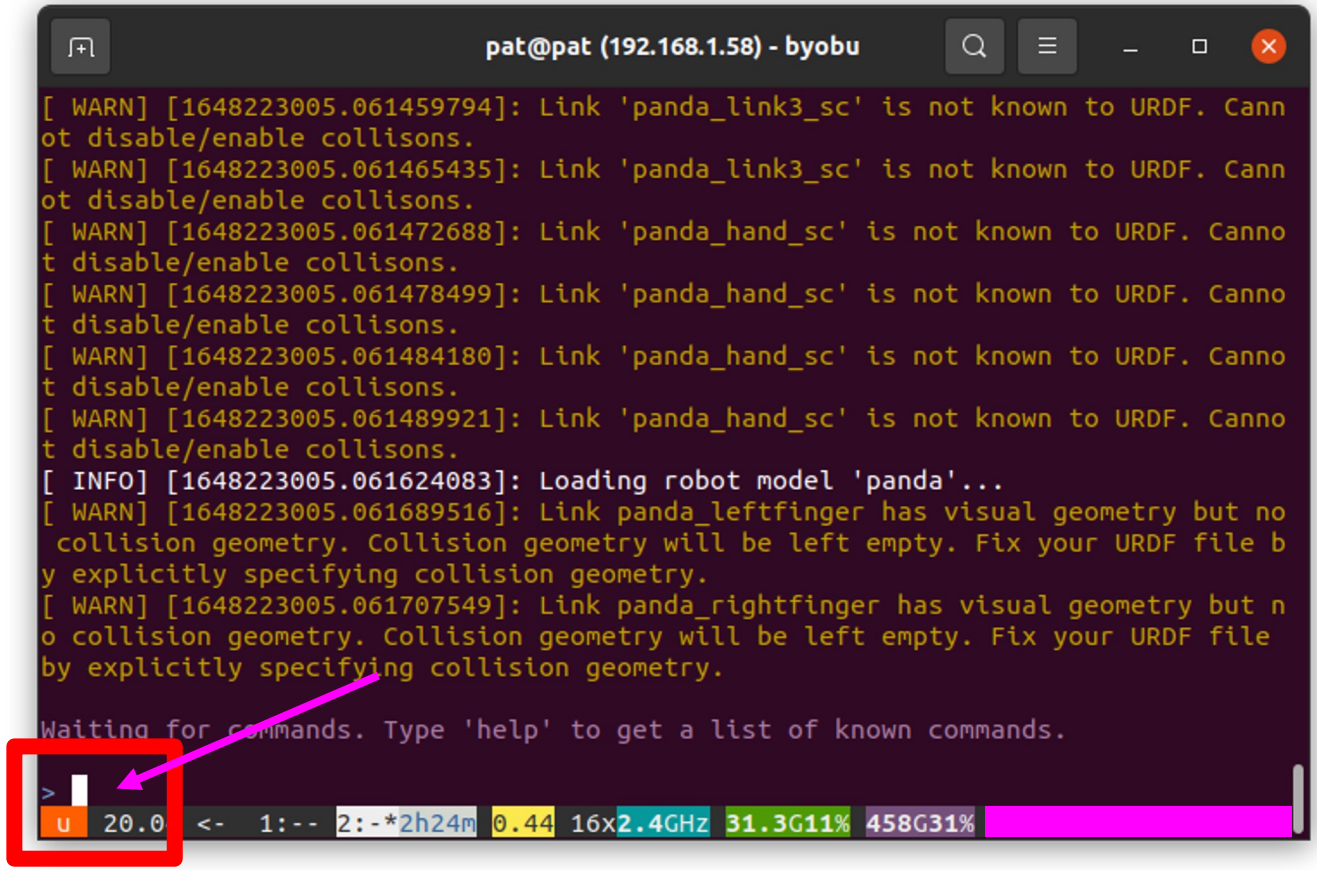
MoveIt Commander Scripting

```
$ rosrun moveit_commander moveit_commander_cmdline.py
```

MoveIt Commander Scripting



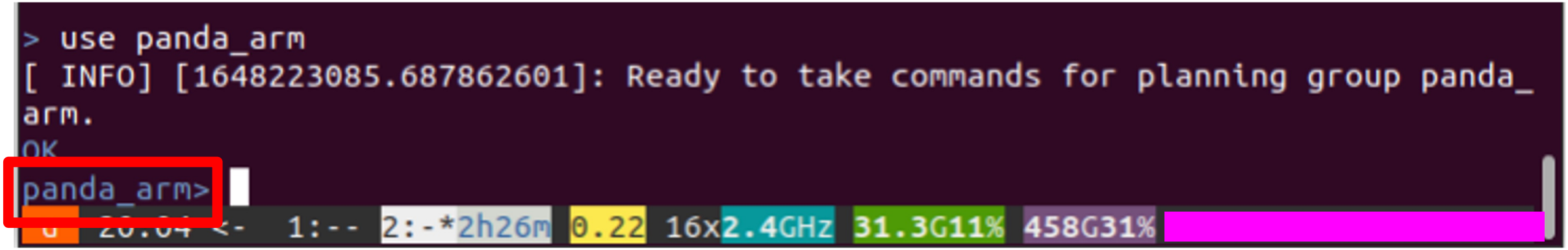
MoveIt Commander Scripting



MoveIt Commander Scripting

```
> use panda_arm
```

MoveIt Commander Scripting



MoveIt Commander Scripting

```
> current
```

MoveIt Commander Scripting

```
joints = [0.0 -0.7853981633974483 0.0 -2.356194490192345 0.0 1.5707963267948966
0.7853981633974483]
header:
  seq: 0
  stamp:
    secs: 1648223187
    nsecs: 440111398
  frame_id: "panda_link0"
pose:
  position:
    x: 0.30689056659294117
    y: -2.3679704553216237e-16
    z: 0.5902820523028393
  orientation:
    x: 0.9238795325112867
    y: -0.3826834323650899
    z: 3.1799184314772466e-17
    w: 1.5677392968614206e-16 ]
panda_link8_RPY = [3.141592653589793, -0.0, -0.7853981633974485]
panda_arm>
 u  20.04 <-  1:--  2:-*2h27m  0.87  16x2.6GHz  31.3G11%  458G31%
```

MoveIt Commander Scripting

```
panda_arm> current
joints = [0.0 -0.7853981633974483 0.0 -2.356194490192345 0.0 1.5707963267948966
0.7853981633974483]
panda_link8 pose = [
header:
  seq: 0
  stamp:
    secs: 1648223187
    nsecs: 440111398
  frame_id: "panda_link0"
pose:
  position:
    x: 0.30689056659294117
    y: -2.36797045532162 37e-16
    z: 0.5902820523028393
  orientation:
    x: 0.9238795325112867
    y: -0.3826834323650899
    z: 3.1799184314772466e-17
    w: 1.5677392968614206e-16 ]
panda_arm>
 u  20.04 <-  1:--  2:-*2h27m  0.87  16x2.6GHz  31.3G11%  458G31%
```

MoveIt Commander Scripting

```
panda_arm> current
joints = [0.0 -0.7853981633974483 0.0 -2.356194490192345 0.0 1.5707963267948966
0.7853981633974483]
panda_link8 pose = [
header:
  seq: 0
  stamp:
    secs: 1648223187
    nsecs: 440111398
  frame_id: "panda_link0"
pose:
  position:
    x: 0.30689056659294117
    y: -2.3679704553216237e-16
    z: 0.5902820523028393
  orientation:
    x: 0.9238795325112867
    y: -0.3826834323650899
    z: 3.1799184314772466e-17
    w: 1.5677329686142069e-16 ]
panda_link8_RPY = [3.141592653589793, -0.0, -0.7853981633974485]
panda_arm>
 u  20.04 <-  1:--  2:-*2h27m  0.87  16x2.6GHz  31.3G11%  458G31%
```

MoveIt Commander Scripting

```
> rec c
```

MoveIt Commander Scripting

```
panda_arm> rec c
Remembered current joint values under the name c
```

MoveIt Commander Scripting

```
> c
```

MoveIt Commander Scripting

```
panda_arm> c
[0.0 -0.7853981633974483 0.0 -2.356194490192345 0.0 1.5707963267948966 0.7853981
633974483]
```

MoveIt Commander Scripting

```
joints = [0.0 -0.7853981633974483 0.0 -2.356194490192345 0.0 1.5707963267948966
0.7853981633974483]
header:
  seq: 0
  stamp:
    secs: 1648223187
    nsecs: 440111398
  frame_id: "panda_link0"
pose:
  position:
    x: 0.30689056659294117
    y: -2.3679704553216237e-16
    z: 0.5902820523028393
  orientation:
    x: 0.9238795325112867
    y: -0.3826834323650899
    z: 3.1799184314772466e-17
    w: 1.5677392968614206e-16 ]
panda_link8_RPY = [3.141592653589793, -0.0, -0.7853981633974485]
panda_arm>
 u  20.04 <-  1:--  2:-*2h27m  0.87  16x2.6GHz  31.3G11%  458G31%
```

MoveIt Commander Scripting

```
> goal = c
```

MoveIt Commander Scripting

```
panda_arm> goal = c
goal is now the same as c
```

MoveIt Commander Scripting

```
> goal[0] = 0.2
```

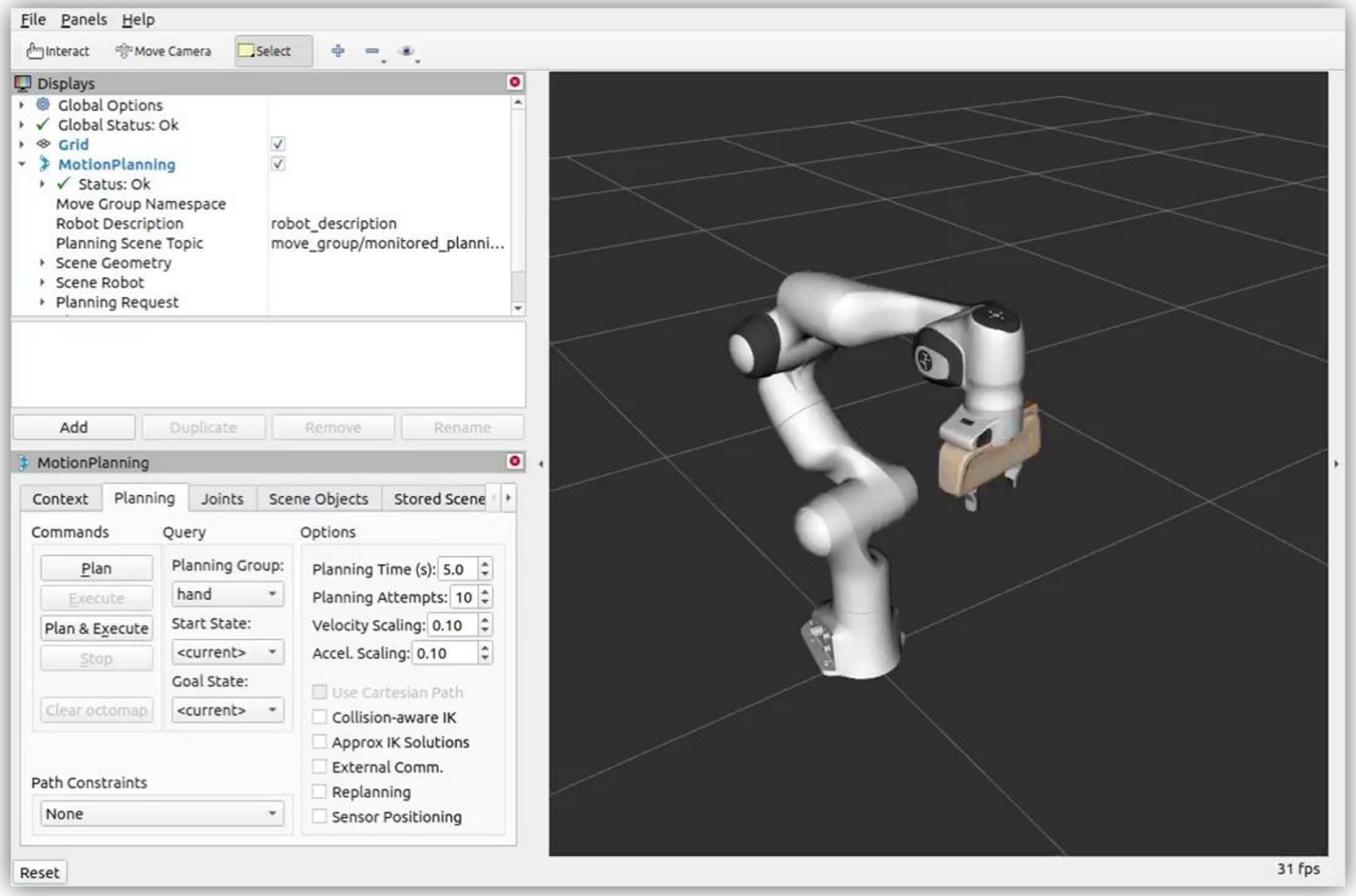
MoveIt Commander Scripting

```
panda_arm> goal[0] = 0.2
Updated goal[0]
```

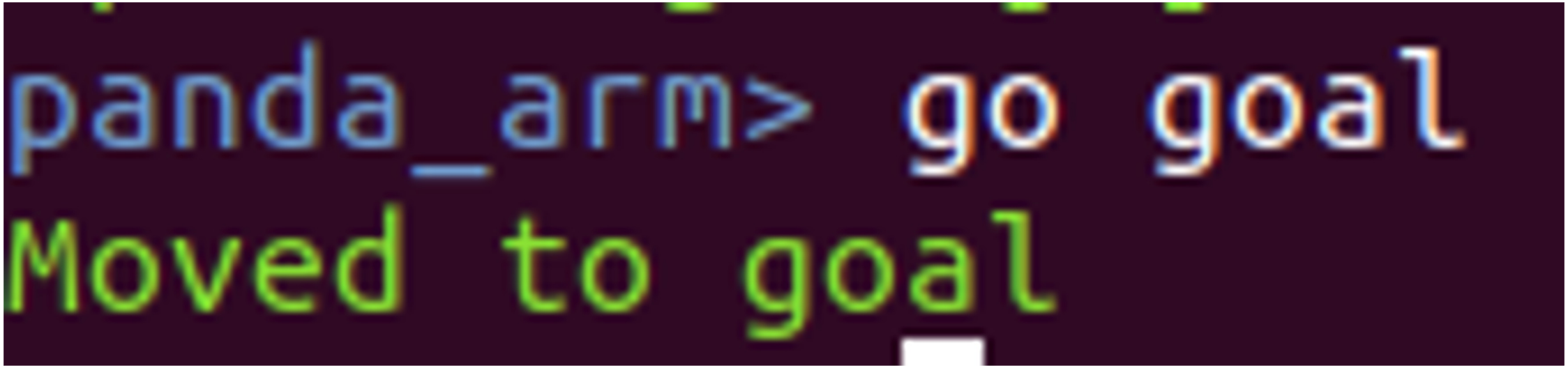
MoveIt Commander Scripting

```
> go goal
```

MoveIt Commander Scripting



MoveIt Commander Scripting



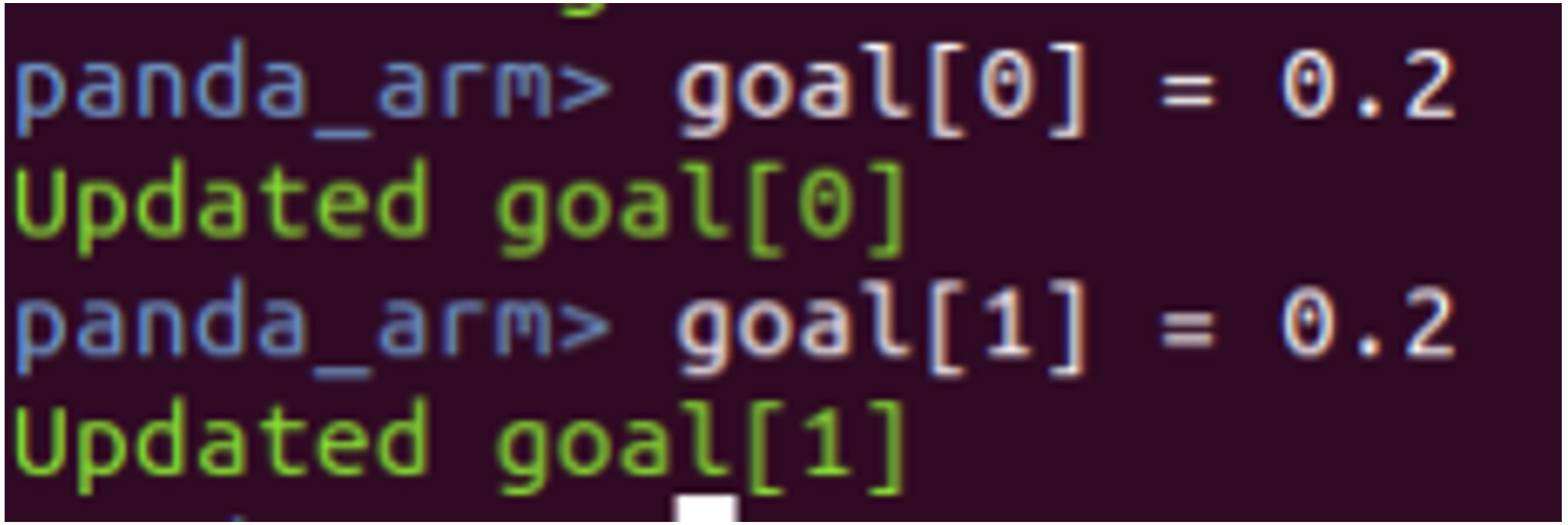
MoveIt Commander Scripting

```
> goal[0] = 0.2
```

MoveIt Commander Scripting

```
> goal[1] = 0.2
```

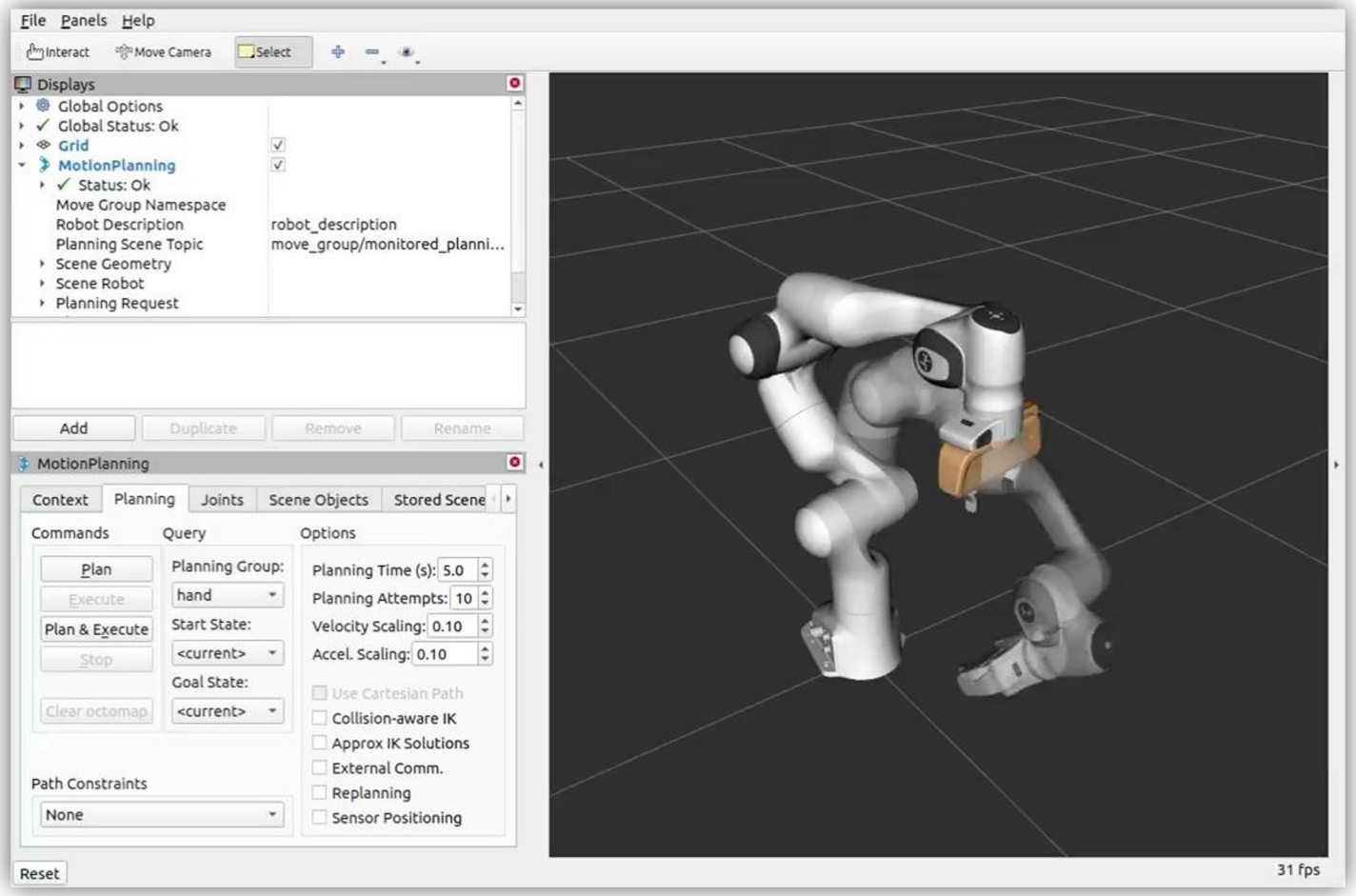
MoveIt Commander Scripting



MoveIt Commander Scripting

```
> plan goal
```

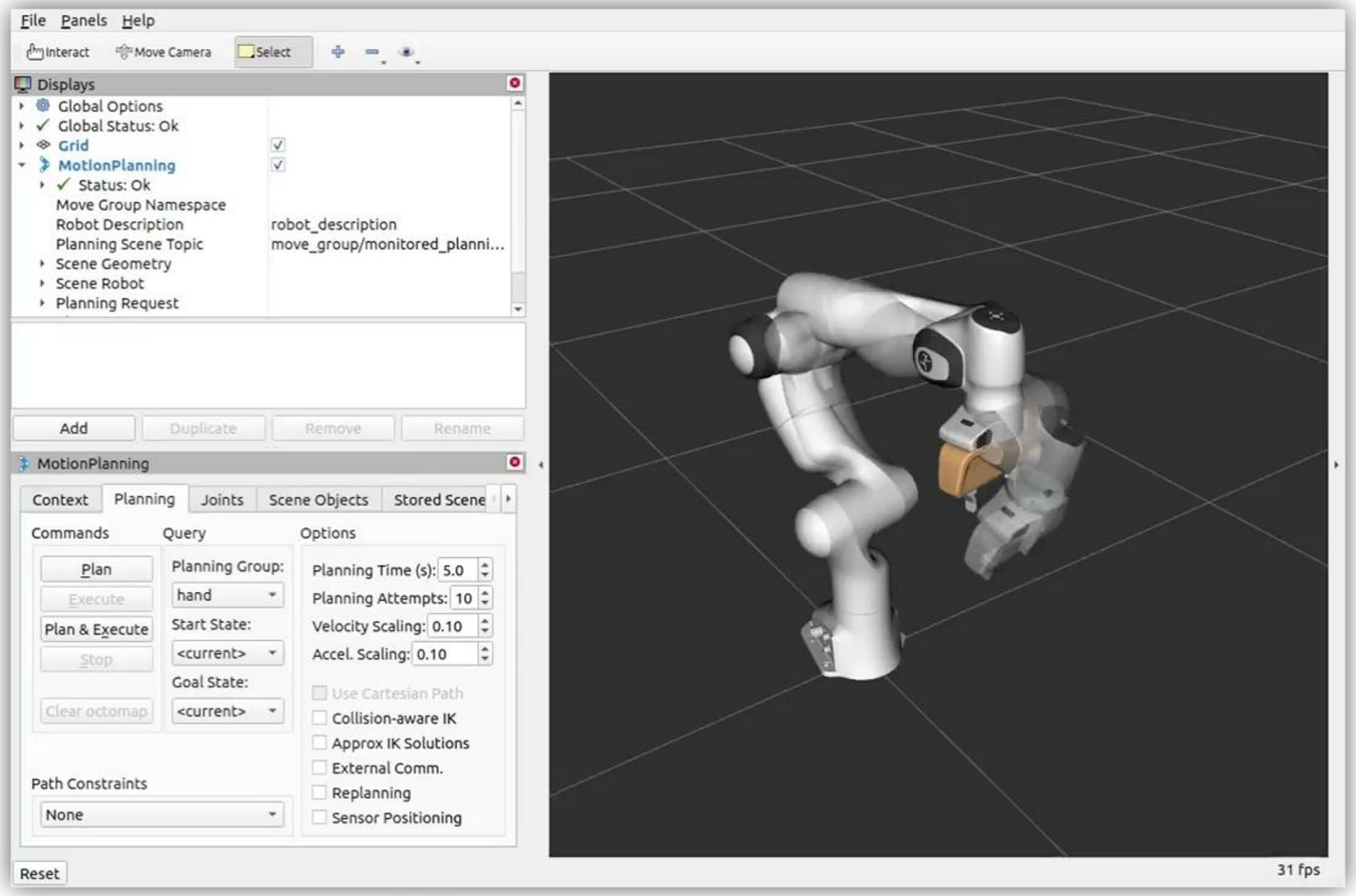
MoveIt Commander Scripting



MoveIt Commander Scripting

```
> execute
```

MoveIt Commander Scripting



MoveIt Commander Scripting

```
> quit
```

MoveIt Commander Scripting

Ctrl + c in all terminal

Move Group Python Interface

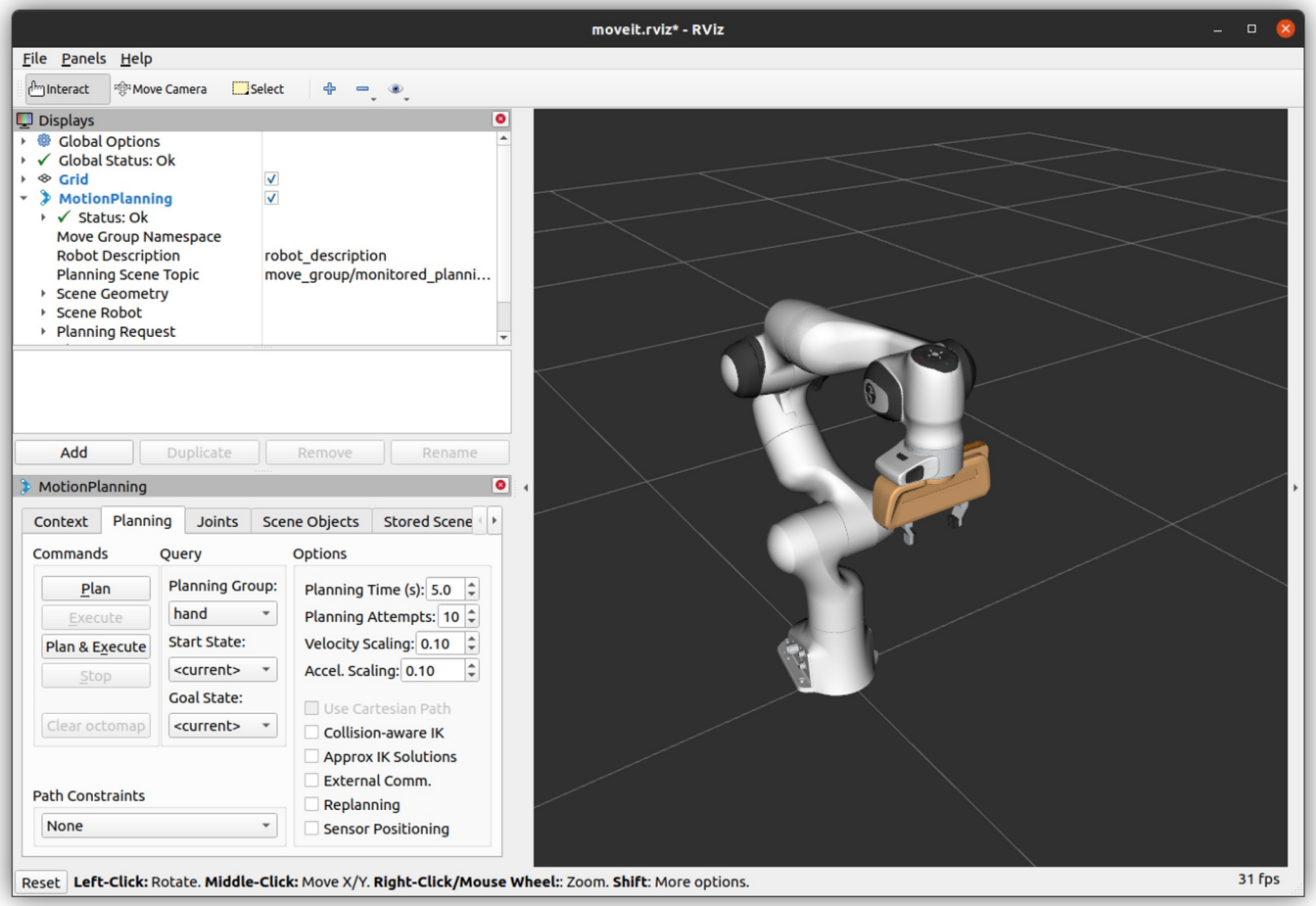
Using python with moveit

154

Move Group Python Interface

```
$ roslaunch panda_moveit_config demo.launch
```

Move Group Python Interface



Move Group Python Interface

```
$ rosrun moveit_tutorials move_group_python_interface_tutorial.py
```

Move Group Python Interface

```
def main():
    try:
        print("")
        print("----------------------------------------------------------")
        print("Welcome to the MoveIt MoveGroup Python Interface Tutorial")
        print("----------------------------------------------------------")
        print("Press Ctrl-D to exit at any time")
        print("")
        input(
            "============ Press `Enter` to begin the tutorial by setting up the moveit_commander ..."
        )
        tutorial = MoveGroupPythonInterfaceTutorial()

        input(
            "============ Press `Enter` to execute a movement using a joint state goal ..."
        )
```

Move Group Python Interface

```python
class MoveGroupPythonInterfaceTutorial(object):
    """MoveGroupPythonInterfaceTutorial"""

    def __init__(self):
        super(MoveGroupPythonInterfaceTutorial, self).__init__()

        ## BEGIN_SUB_TUTORIAL setup
        ##
        ## First initialize `moveit_commander`_ and a `rospy`_ node:
        moveit_commander.roscpp_initialize(sys.argv)
        rospy.init_node("move_group_python_interface_tutorial", anonymous=True)

        ## Instantiate a `RobotCommander`_ object. Provides information such as the robot's
        ## kinematic model and the robot's current joint states
        robot = moveit_commander.RobotCommander()

        ## Instantiate a `PlanningSceneInterface`_ object.  This provides a remote interface
        ## for getting, setting, and updating the robot's internal understanding of the
        ## surrounding world:
        scene = moveit_commander.PlanningSceneInterface()

        ## Instantiate a `MoveGroupCommander`_ object.  This object is an interface
        ## to a planning group (group of joints).  In this tutorial the group is the primary
        ## arm joints in the Panda robot, so we set the group's name to "panda_arm".
        ## If you are using a different robot, change this value to the name of your robot
        ## arm planning group.
        ## This interface can be used to plan and execute motions:
        group_name = "panda_arm"
        move_group = moveit_commander.MoveGroupCommander(group_name)

        ## Create a `DisplayTrajectory`_ ROS publisher which is used to display
        ## trajectories in Rviz:
        display_trajectory_publisher = rospy.Publisher(
            "/move_group/display_planned_path",
            moveit_msgs.msg.DisplayTrajectory,
            queue_size=20,
        )
```

Move Group Python Interface

```
planning_frame = move_group.get_planning_frame()
print("============ Planning frame: %s" % planning_frame)

# We can also print the name of the end-effector link for this group:
eef_link = move_group.get_end_effector_link()
print("============ End effector link: %s" % eef_link)

# We can get a list of all the groups in the robot:
group_names = robot.get_group_names()
print("============ Available Planning Groups:", robot.get_group_names())

# Sometimes for debugging it is useful to print the entire state of the
# robot:
print("============ Printing robot state")
print(robot.get_current_state())
print("")
## END_SUB_TUTORIAL

# Misc variables
self.box_name = ""
self.robot = robot
self.scene = scene
self.move_group = move_group
self.display_trajectory_publisher = display_trajectory_publisher
self.planning_frame = planning_frame
self.eef_link = eef_link
self.group_names = group_names
```

Move Group Python Interface

```
tutorial.go_to_joint_state()
```

Move Group Python Interface

```python
def go_to_joint_state(self):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    move_group = self.move_group

    ## BEGIN_SUB_TUTORIAL plan_to_joint_state
    ##
    ## Planning to a Joint Goal
    ## ^^^^^^^^^^^^^^^^^^^^^^^^
    ## The Panda's zero configuration is at a `singularity <https://www.quora.com/Robotics-What-is-meant-by-kinematic-singularity>`_, so the first
    ## thing we want to do is move it to a slightly better configuration.
    ## We use the constant `tau = 2*pi <https://en.wikipedia.org/wiki/Turn_(angle)#Tau_proposals>`_ for convenience:
    # We get the joint values from the group and change some of the values:
    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = 0
    joint_goal[1] = -tau / 8
    joint_goal[2] = 0
    joint_goal[3] = -tau / 4
    joint_goal[4] = 0
    joint_goal[5] = tau / 6  # 1/6 of a turn
    joint_goal[6] = 0

    # The go command can be called with joint values, poses, or without any
    # parameters if you have already set the pose or joint target for the group
    move_group.go(joint_goal, wait=True)

    # Calling ``stop()`` ensures that there is no residual movement
    move_group.stop()

    ## END_SUB_TUTORIAL

    # For testing:
    current_joints = move_group.get_current_joint_values()
    return all_close(joint_goal, current_joints, 0.01)
```

Move Group Python Interface

```
input("============ Press `Enter` to execute a movement using a pose goal ...")
tutorial.go_to_pose_goal()
```

Move Group Python Interface

```python
def go_to_pose_goal(self):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    move_group = self.move_group

    ## BEGIN_SUB_TUTORIAL plan_to_pose
    ##
    ## Planning to a Pose Goal
    ## ^^^^^^^^^^^^^^^^^^^^^^^
    ## We can plan a motion for this group to a desired pose for the
    ## end-effector:
    pose_goal = geometry_msgs.msg.Pose()
    pose_goal.orientation.w = 1.0
    pose_goal.position.x = 0.4
    pose_goal.position.y = 0.1
    pose_goal.position.z = 0.4

    move_group.set_pose_target(pose_goal)

    ## Now, we call the planner to compute the plan and execute it.
    plan = move_group.go(wait=True)
    # Calling `stop()` ensures that there is no residual movement
    move_group.stop()
    # It is always good to clear your targets after planning with poses.
    # Note: there is no equivalent function for clear_joint_value_targets()
    move_group.clear_pose_targets()

    ## END_SUB_TUTORIAL

    # For testing:
    # Note that since this section of code will not be included in the tutorials
    # we use the class variable rather than the copied state variable
    current_pose = self.move_group.get_current_pose().pose
    return all_close(pose_goal, current_pose, 0.01)
```

Move Group Python Interface

```
input("============ Press `Enter` to plan and display a Cartesian path ...")
cartesian_plan, fraction = tutorial.plan_cartesian_path()

input(
    "============ Press `Enter` to display a saved trajectory (this will replay the Cartesian path)  ..."
)
tutorial.display_trajectory(cartesian_plan)

input("============ Press `Enter` to execute a saved path ...")
tutorial.execute_plan(cartesian_plan)
```

Move Group Python Interface

```python
def plan_cartesian_path(self, scale=1):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    move_group = self.move_group

    ## BEGIN_SUB_TUTORIAL plan_cartesian_path
    ##
    ## Cartesian Paths
    ## ^^^^^^^^^^^^^^^
    ## You can plan a Cartesian path directly by specifying a list of waypoints
    ## for the end-effector to go through. If executing  interactively in a
    ## Python shell, set scale = 1.0.
    ##
    waypoints = []

    wpose = move_group.get_current_pose().pose
    wpose.position.z -= scale * 0.1  # First move up (z)
    wpose.position.y += scale * 0.2  # and sideways (y)
    waypoints.append(copy.deepcopy(wpose))

    wpose.position.x += scale * 0.1  # Second move forward/backwards in (x)
    waypoints.append(copy.deepcopy(wpose))

    wpose.position.y -= scale * 0.1  # Third move sideways (y)
    waypoints.append(copy.deepcopy(wpose))

    # We want the Cartesian path to be interpolated at a resolution of 1 cm
    # which is why we will specify 0.01 as the eef_step in Cartesian
    # translation.  We will disable the jump threshold by setting it to 0.0,
    # ignoring the check for infeasible jumps in joint space, which is sufficient
    # for this tutorial.
    (plan, fraction) = move_group.compute_cartesian_path(
        waypoints, 0.01, 0.0  # waypoints to follow  # eef_step
    )  # jump_threshold

    # Note: We are just planning, not asking move_group to actually move the robot yet:
    return plan, fraction
```

Move Group Python Interface

```python
def display_trajectory(self, plan):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    robot = self.robot
    display_trajectory_publisher = self.display_trajectory_publisher

    ## BEGIN_SUB_TUTORIAL display_trajectory
    ##
    ## Displaying a Trajectory
    ## ^^^^^^^^^^^^^^^^^^^^^^^
    ## You can ask RViz to visualize a plan (aka trajectory) for you. But the
    ## group.plan() method does this automatically so this is not that useful
    ## here (it just displays the same trajectory again):
    ##
    ## A `DisplayTrajectory`_ msg has two primary fields, trajectory_start and trajectory.
    ## We populate the trajectory_start with our current robot state to copy over
    ## any AttachedCollisionObjects and add our plan to the trajectory.
    display_trajectory = moveit_msgs.msg.DisplayTrajectory()
    display_trajectory.trajectory_start = robot.get_current_state()
    display_trajectory.trajectory.append(plan)
    # Publish
    display_trajectory_publisher.publish(display_trajectory)

    ## END_SUB_TUTORIAL
```

Move Group Python Interface

```python
def execute_plan(self, plan):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    move_group = self.move_group

    ## BEGIN_SUB_TUTORIAL execute_plan
    ##
    ## Executing a Plan
    ## ^^^^^^^^^^^^^^^^
    ## Use execute if you would like the robot to follow
    ## the plan that has already been computed:
    move_group.execute(plan, wait=True)

    ## **Note:** The robot's current joint state must be within some tolerance of the
    ## first waypoint in the `RobotTrajectory`_ or ``execute()`` will fail
    ## END_SUB_TUTORIAL
```

Move Group Python Interface

```
input("============ Press `Enter` to add a box to the planning scene ...")
tutorial.add_box()
```

Move Group Python Interface

```python
def add_box(self, timeout=4):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    box_name = self.box_name
    scene = self.scene

    ## BEGIN_SUB_TUTORIAL add_box
    ##
    ## Adding Objects to the Planning Scene
    ## ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    ## First, we will create a box in the planning scene between the fingers:
    box_pose = geometry_msgs.msg.PoseStamped()
    box_pose.header.frame_id = "panda_hand"
    box_pose.pose.orientation.w = 1.0
    box_pose.pose.position.z = 0.11  # above the panda_hand frame
    box_name = "box"
    scene.add_box(box_name, box_pose, size=(0.075, 0.075, 0.075))

    ## END_SUB_TUTORIAL
    # Copy local variables back to class variables. In practice, you should use the class
    # variables directly unless you have a good reason not to.
    self.box_name = box_name
    return self.wait_for_state_update(box_is_known=True, timeout=timeout)
```

Move Group Python Interface

```
input("============ Press `Enter` to attach a Box to the Panda robot ...")
tutorial.attach_box()
```

Move Group Python Interface

```python
def attach_box(self, timeout=4):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    box_name = self.box_name
    robot = self.robot
    scene = self.scene
    eef_link = self.eef_link
    group_names = self.group_names

    ## BEGIN_SUB_TUTORIAL attach_object
    ##
    ## Attaching Objects to the Robot
    ## ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    ## Next, we will attach the box to the Panda wrist. Manipulating objects requires the
    ## robot be able to touch them without the planning scene reporting the contact as a
    ## collision. By adding link names to the ``touch_links`` array, we are telling the
    ## planning scene to ignore collisions between those links and the box. For the Panda
    ## robot, we set ``grasping_group = 'hand'``. If you are using a different robot,
    ## you should change this value to the name of your end effector group name.
    grasping_group = "hand"
    touch_links = robot.get_link_names(group=grasping_group)
    scene.attach_box(eef_link, box_name, touch_links=touch_links)
    ## END_SUB_TUTORIAL

    # We wait for the planning scene to update.
    return self.wait_for_state_update(
        box_is_attached=True, box_is_known=False, timeout=timeout
    )
```

Move Group Python Interface

```
input(
    "============ Press `Enter` to plan and execute a path with an attached collision object ..."
)
cartesian_plan, fraction = tutorial.plan_cartesian_path(scale=-1)
tutorial.execute_plan(cartesian_plan)
```

Move Group Python Interface

```
input("============ Press `Enter` to detach the box from the Panda robot ...")
tutorial.detach_box()
```

Move Group Python Interface

```python
def detach_box(self, timeout=4):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    box_name = self.box_name
    scene = self.scene
    eef_link = self.eef_link

    ## BEGIN_SUB_TUTORIAL detach_object
    ##
    ## Detaching Objects from the Robot
    ## ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    ## We can also detach and remove the object from the planning scene:
    scene.remove_attached_object(eef_link, name=box_name)
    ## END_SUB_TUTORIAL

    # We wait for the planning scene to update.
    return self.wait_for_state_update(
        box_is_known=True, box_is_attached=False, timeout=timeout
    )
```

Move Group Python Interface

```
input(
    "============ Press `Enter` to remove the box from the planning scene ..."
)
tutorial.remove_box()
```

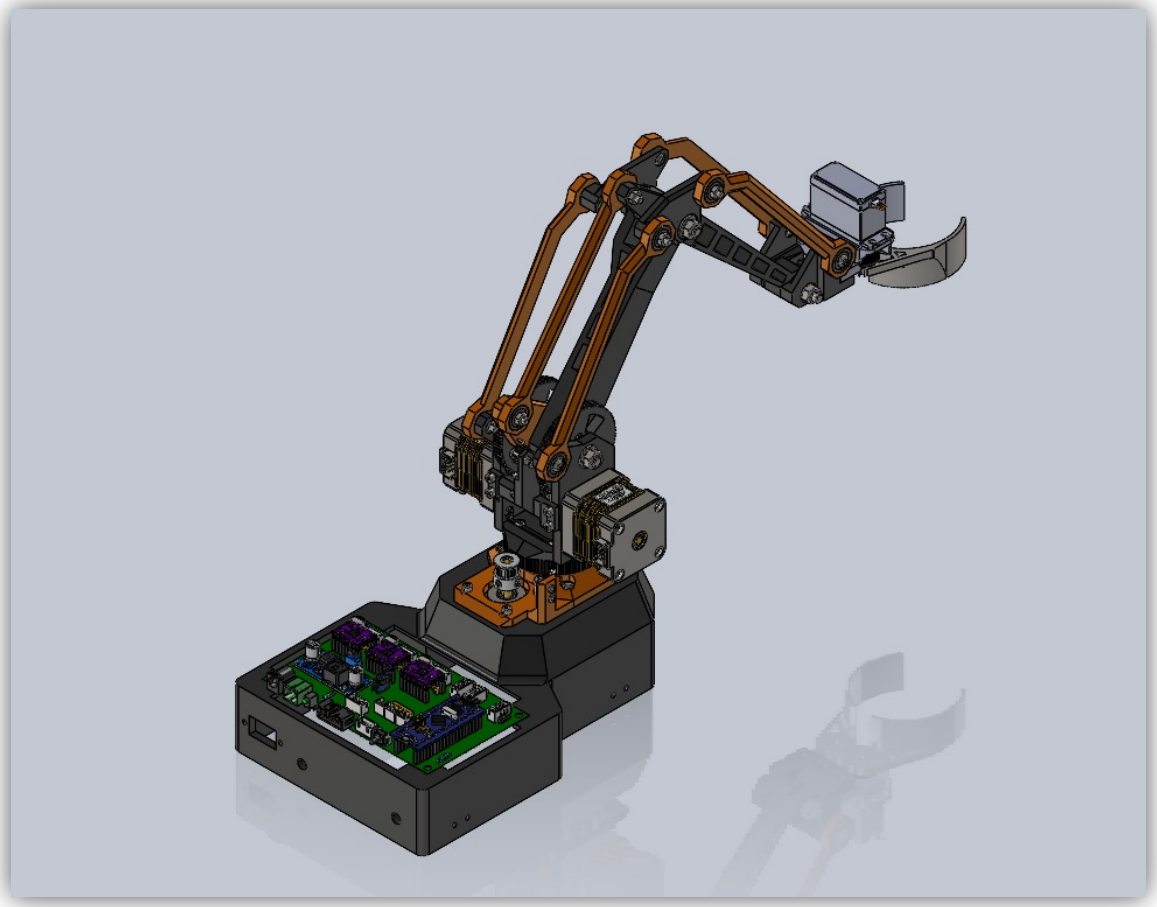
Move Group Python Interface

```python
def remove_box(self, timeout=4):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    box_name = self.box_name
    scene = self.scene

    ## BEGIN_SUB_TUTORIAL remove_object
    ##
    ## Removing Objects from the Planning Scene
    ## ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    ## We can remove the box from the world.
    scene.remove_world_object(box_name)

    ## **Note:** The object must be detached before we can remove it from the world
    ## END_SUB_TUTORIAL

    # We wait for the planning scene to update.
    return self.wait_for_state_update(
        box_is_attached=False, box_is_known=False, timeout=timeout
    )
```

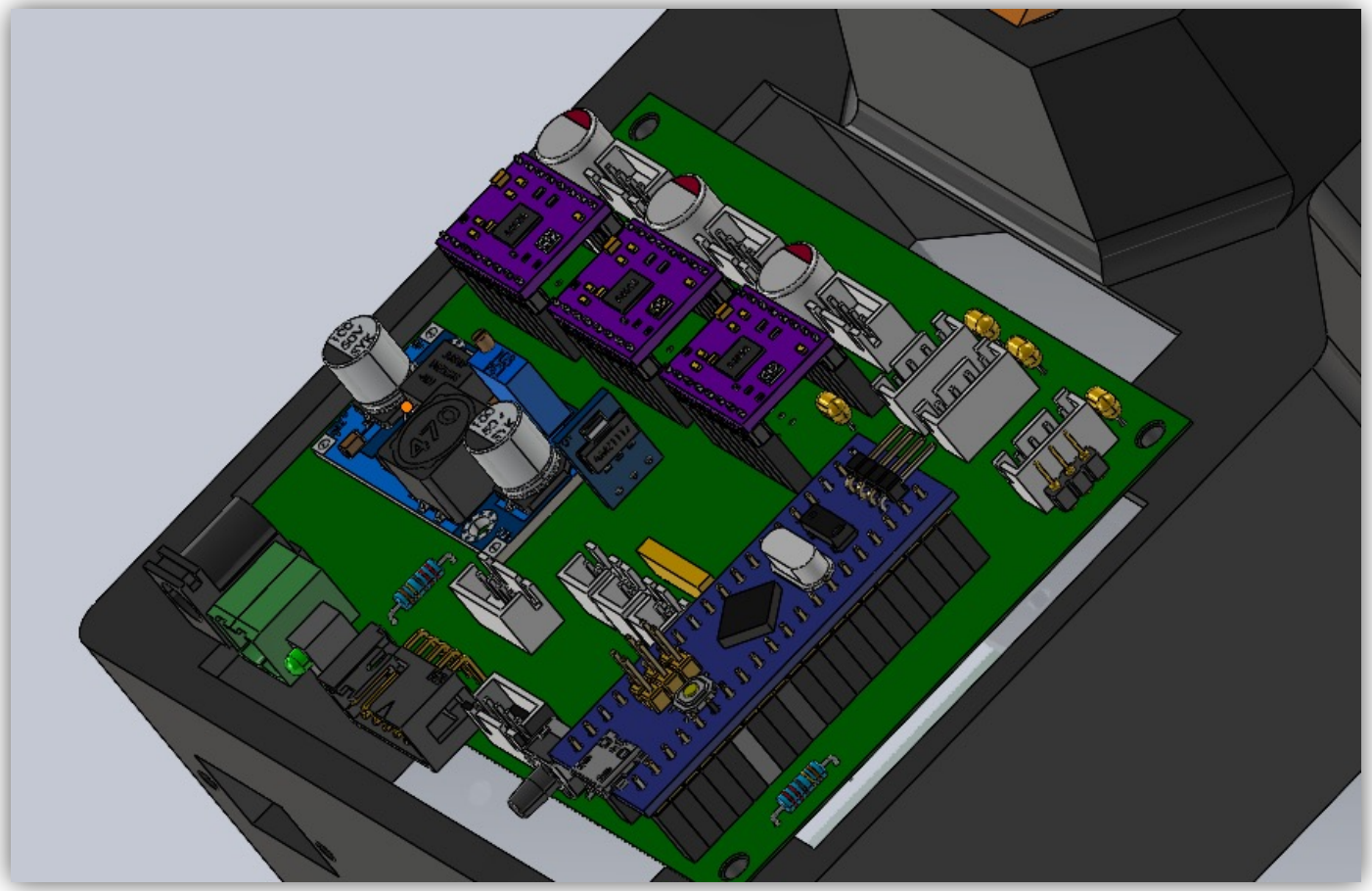
Integrate real arm with ROS

Using real robot with ROS

Integrate real arm with ROS



Integrate real arm with ROS



Integrate real arm with ROS

Roboarm Control Protocol (FW ver 1.0)

Action Command

ACTION ENCODER_TARGET 3 [X][Y][Z]
Rotate each arm of the robot to a specified encoder position.

CMD: #AE3 [X] [Y] [Z]|
X: Target Encoder Position (for Base)
Y: Target Encoder Position (for Main arm)
Z: Target Encoder Position (for Sub arm)

Ex: #AE3 3000 2000 -100|
Rotate each arm. For base, main arm and sub arm encoder positions are 3000,2000, and -100 respectively.

Return:
"OK: [X] [Y] [Z]"
"Err: MSG not recognized"

ACTION ANGLE_TARGET 3 [X] [Y] [Z]
Rotate each arm of the robot to a specified angle position.

CMD: #AA3 [X] [Y] [Z]|
X: Target Angle (for Base)
Y: Target Angle (for Main arm)
Z: Target Angle (for Sub arm)

Ex: #AA3 25.50 12.50 20.30|
Rotate each arm. For base, main arm and sub arm angles are 25.50°,12.50°, and 20.30° respectively.

Return:
"OK: [X] [Y] [Z]"
"Err: MSG not recognized"

ACTION GRIPPER 1 [Lvl]
Control the grip level of the gripper.

CMD: #AG1 [Lvl]|
Lvl: Grip level (0 more grip - 4 less grip)

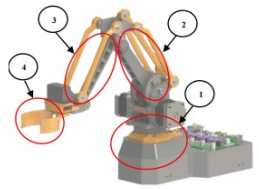
Ex: #AG1 2|
set grip level equal to 2

Return:
"OK: GRIPPER LVL [Lvl]"
"Err: MSG not recognized"
"Err: GRIPPER LVL NOT IN RANGE"

Command structure

#[Type] [Subtype] [argument number] [arg]...|

Robot Part Definition



1. Base (Arm index = 0)
2. Main Arm (Arm index = 1)
3. Sub Arm (Arm index = 2)
4. Gripper

ACTION ORIGIN 0
Set each robot arm to the origin(home) position.

CMD: #AO0|

Ex: #AO0|
Rotate all arms to its origin position (home)

Return:
"OK: RESET TO ORIGIN..."
"OK: AT ORIGIN"

Note: Green LED of controller is **ON** during this operation. All other commands will be ignored.

Parameter Setup Command

SET PID_GAIN 4 [Arm idx] [Pval] [Ival] [Dval]
Set PID gain for a specified arm index

CMD: #SP4 [Arm idx] [Pval] [Ival] [Dval]|
Arm: target arm index (0-Base 1-Main arm 2-Sub arm)
Pval: P_gain
Ival: I_gain
Dval: D_gain

Ex.#SP4 1 3.00 0.50 0.10|
Set PID for controlling main arm. P gain is 3.00, I gain is 0.50 and D gain is 0.10

Return:
"OK: ARM = [Arm idx] P = [P_val] I = [I_val] D = [D_val]"
"Err: MSG not recognized"
"Err: ARM IDX is out of range"

Note: PID controller of this robot arm increase or reduce pulse frequency which is used to be the input of the stepper driver causing a change in rotation speed. Changing rotation speed isn't guaranteed to increase driving power.

Read Command

READ ENCODER_POSITION 0
Read the encoder position of base, main arm, and sub arm respectively.

CMD: #RE0|

Ex: #RE0|
Read encoder value from all encoders.

Return:
"[base Encoder] [main arm Encoder] [sub arm Encoder]"
"Err : MSG not recognized"

READ GRIPPER_LEVEL 0
Read the grip level of the gripper.

CMD: #RG0|

Ex: #RG0|
Read the grip level of the gripper.

Return:
"[Gripper level (0 less grip -4 more grip)]
"Err : MSG not recognized"

READ ANGLE_POSITION 0
Read the angle of base, main arm, and sub arm respectively.

CMD: #RA0|

Ex: #RA0|
Read angle translated from all encoders.

Return:
"[base Angle] [main arm Angle] [sub arm Angle]"
"Err : MSG not recognized"

READ PID 1 [Arm idx]
Read PID gain for specified arm index.

CMD: #RP1 [Arm idx]|
Arm idx: target arm index (0-Base 1-Main arm 2-Sub arm)

Ex: #RP1 0|
Read PID gain from base motor controller.

Return:
"[Arm idx P gain] [Arm idx I gain] [Arm idx D gain]"
"Err : MSG not recognized"
"Err: ARM IDX is out of range"

Integrate real arm with ROS

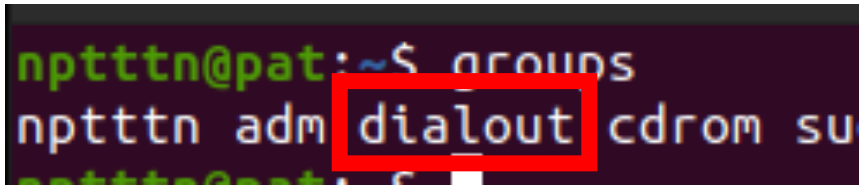
```
$ cd <your_workspace>/src
```

```
$ git clone https://gitlab.com/robotcitizens/workshop/ku_belt_arm_driver.git
```

```
$ cd ..
```

```
$ catkin_make
```

Integrate real arm with ROS




```
$ groups
```

```
nptttn@pat:~/tutorial_ws$ groups
nptttn adm cdrom sudo dip plugdev lpadmin lxd sambashare
```

If don’t have dialout in output, please do the command below.

```
$ sudo usermod -aG dialout $USER
```

```
$ sudo reboot
```

Integrate real arm with ROS

```python
class ArmDriver(object):
    def __init__(self):

        usb_port = os.getenv("ARM_PORT")
        if usb_port is None:
            usb_port = "/dev/ttyACM0"
        try:
            self.board = serial.Serial(usb_port, 115200, rtscts=True)
        except serial.serialutil.SerialException: # type: ignore
            raise(Exception(f"Can't find port {usb_port}"))

        self.current_pose = None
        self.goal_position = None
        self.ros_path = RosPack().get_path('ku_arm_belt_driver')

        self.go_home()
        wait_for_avaliable(self.board)
        rospy.loginfo("Driver is ready!")

        self.joint_state_pub = rospy.Publisher("/AiForAll/manipulation/joint_states", JointState, queue_size=1)
        self.go_to_jointstate_sub = rospy.Subscriber("/AiForAll/manipulation/go_joint_states", JointState, self.update_jointstate_cb)
        self.go_home_ser = rospy.Service("/AiForAll/manipulation/go_home", Empty, self.go_home_cb)
        self.go_home_ser = rospy.Service("/AiForAll/manipulation/open_gripper", Empty, self.open_gripper_cb)
        self.go_home_ser = rospy.Service("/AiForAll/manipulation/close_gripper", Empty, self.close_gripper_cb)
        self.save_pose = rospy.Service("/AiForAll/manipulation/save_pose", SaveArmPose, self.save_pose_cb)
        self.go_to_pose = rospy.Service("/AiForAll/manipulation/go_to_pose", GoToArmPose, self.go_to_pose_cb)
```

Integrate real arm with ROS

```python
    def read_angle(self):
        pkg = b"#RA0|"
        self.board.flush()
        self.board.write(pkg)

        time.sleep(0.2)
        if self.board.in_waiting > 0:
            # self.board.write(pkg)
            data = self.board.read_all()
            data = data.decode() # type: ignore
            data = data.split("\r\n")
            rospy.logdebug(data)
            try:
                RA_index = data.index("RA0") + 1
                data = data[RA_index].split(" ")
                for i in range(len(data)):
                    data[i] = float(data[i]) # type: ignore
            except ValueError:
                rospy.logdebug("Error can't get data from encoder.")
                return 1
            message = JointState()
            message.header.stamp = rospy.Time.now()
            message.name = ["motor_1", "motor_2", "motor_3"]
            message.position = data
            self.joint_state_pub.publish(message)
            self.current_pose = data
```

Integrate real arm with ROS

```python
    def update_jointstate_cb(self, data):
        self.goal_position = data

    def go_to_jointstate(self):
        if self.goal_position is None:
            return -1

        rospy.logdebug(f"Goal position: {self.goal_position}")
        if self.goal_position.position[3] > 4 * math.pi / 5:
            gripper_lv = 0
        elif self.goal_position.position[3] > 3 * math.pi / 5:
            gripper_lv = 1
        elif self.goal_position.position[3] > 2 * math.pi / 4:
            gripper_lv = 2
        elif self.goal_position.position[3] > math.pi / 4:
            gripper_lv = 3
        else:
            gripper_lv = 4

        pkg = f"#AG1 {gripper_lv}|".encode()
        rospy.logdebug(f"package write: {pkg}")
        self.board.flush()
        self.board.write(pkg)

        wait_for_avaliable(self.board)

        position = list(self.goal_position.position)
        for i in range(len(position)):
            position[i] = str((position[i]) * 180 / math.pi)
        position = " ".join(position[0:3])
        pkg = f"#AA3 {position}|".encode()
        rospy.logdebug(f"package write: {pkg}")
        self.board.write(pkg)

        wait_for_avaliable(self.board)
```

Integrate real arm with ROS

```
118     def go_home_cb(self, data):
119         self.go_home()
120         return EmptyResponse()
121 
122     def go_home(self):
123         pkg = b"#A00|"
124         rospy.logdebug(f"package write: {pkg}")
125         self.board.write(pkg)
126         time.sleep(0.2)
127         while self.board.in_waiting > 0:
128             data = self.board.read_all()
129             data = data.decode() # type: ignore
130             rospy.logdebug(data)
131             if "OK: AT ORIGIN" in data:
132                 break
```

Integrate real arm with ROS

```python
    def save_pose_cb(self, data):
        pose_name = data.pose_name
        current_pose = self.current_pose
        rospy.logdebug(f"current pose: {current_pose}")
        data = [pose_name] + current_pose # type: ignore

        try:

            with open(f'{self.ros_path}/csv/pose.csv', mode='a') as pose_file:
                pose_writter = csv.writer(pose_file)
                pose_writter.writerow(data) # type: ignore
                pose_file.close()

        except Exception as e:
            rospy.logerr(e)
            return SaveArmPoseResponse(False)

        return SaveArmPoseResponse(True)

    def go_to_pose_cb(self, data):
        pose_name = data.pose_name

        with open(f'{self.ros_path}/csv/pose.csv', mode='r') as pose_file:
                csv_reader = csv.reader(pose_file, delimiter=',')
                for row in csv_reader:
                    if row[0] == pose_name:
                        pkg = f"#AA3 {row[1]} {row[2]} {row[3]}|".encode()
                        rospy.logdebug(f"package write: {pkg}")
                        self.board.write(pkg)
                        wait_for_avaliable(self.board)

        return GoToArmPoseResponse(True)
```

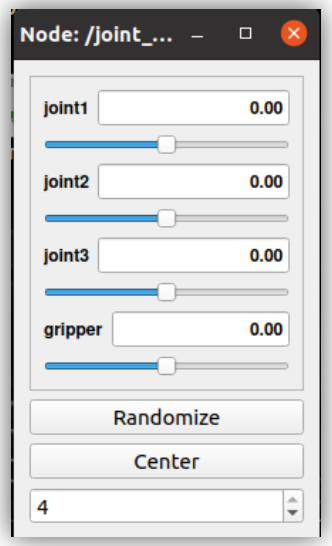
Integrate real arm with ROS

```
167     def open_gripper_cb(self, data):
168         pkg = f"#AG1 4|".encode()
169         rospy.logdebug(f"package write: {pkg}")
170         self.board.write(pkg)
171         wait_for_avaliable(self.board)
172         return EmptyResponse()
173 
174     def close_gripper_cb(self, data):
175         pkg = f"#AG1 0|".encode()
176         rospy.logdebug(f"package write: {pkg}")
177         self.board.write(pkg)
178         wait_for_avaliable(self.board)
179         return EmptyResponse()
```

Integrate real arm with ROS

```
$ rospack profile
```

```
$ roslaunch ku_arm_belt_driver driver.launch joint_state_publisher_enable:=true
```



Integrate real arm with ROS

```
$ roslaunch ku_arm_belt_driver driver.launch
```

```
nptttn@pat:~$ roslaunch ku_arm_belt_driver driver.launch
... logging to /home/nptttn/.ros/log/83302fbe-016d-11ee-8e1b-cb02c4ab28e8/roslaunch-pat-8334.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://pat:34013/

SUMMARY
========

PARAMETERS
 * /robot_description: <?xml version="1....
 * /rosdistro: noetic
 * /rosversion: 1.15.15

NODES
  /
    arm_driver (ku_arm_belt_driver/arm_driver.py)

ROS_MASTER_URI=http://localhost:11311

process[arm_driver-1]: started with pid [8349]
[INFO] [1685728323.117868]: Driver is ready!
```

Integrate real arm with ROS

```
$ rosservice call /AiForAll/manipulation/open_gripper "{}"
```

```
$ rosservice call /AiForAll/manipulation/close_gripper "{}"
```

Integrate real arm with ROS

Close the switch then drag your arm to desired position and run command below

```
$ rosservice call /AiForAll/manipulation/save_pose "pose_name: 'user_1'"
```

Open the switch then the arm will move to the previous position and run command below to move to the position, you save.

```
$ rosservice call /AiForAll/manipulation/go_to_pose "pose_name: 'user_1'"
```