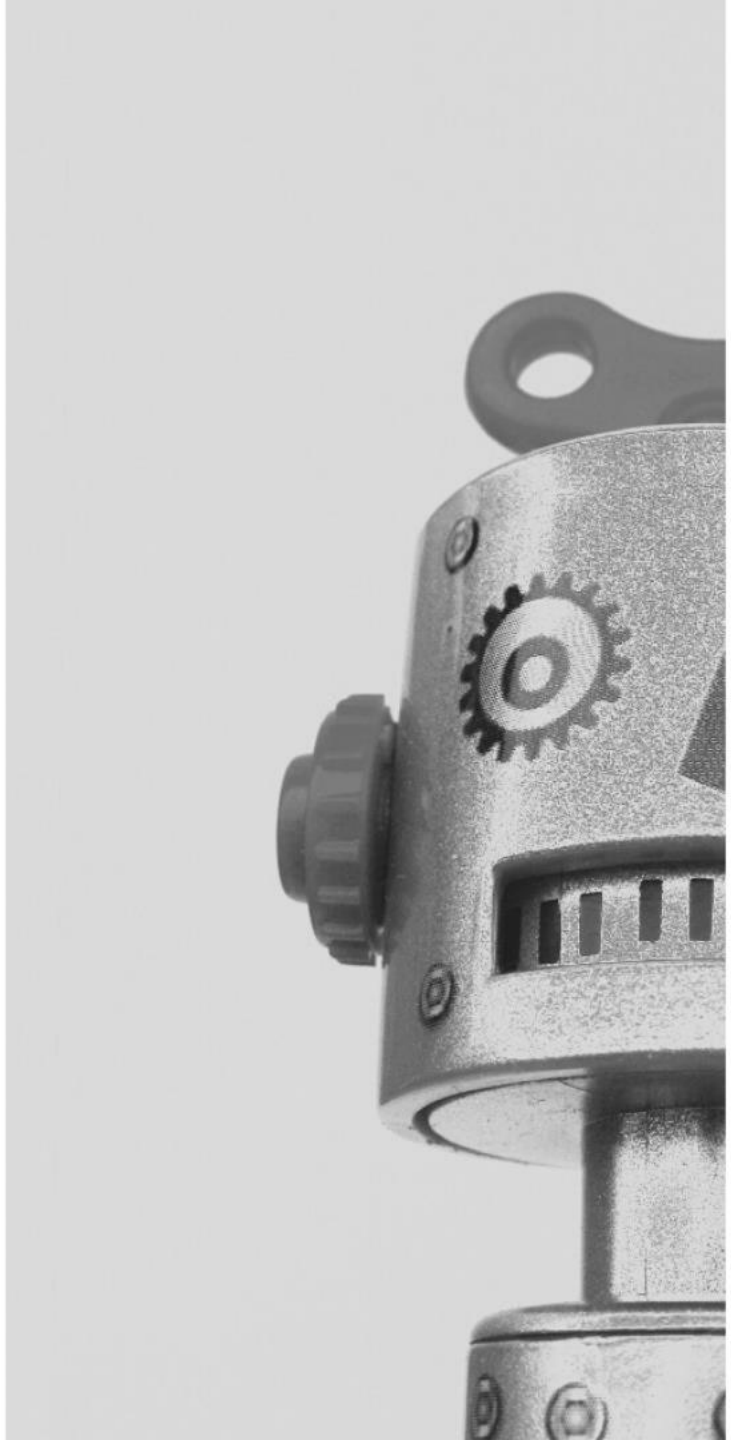
BY robotcitizens



Robot Arm

KASETSART UNIVERSITY

OVERVIEW

Robot Manipulator Concept

- Manipulation Concept

MoveIt!

- MoveIt! Overview
- Moveit! Setup assistant
- Plan and execution motion using “move_group” node

Robot Manipulator Concept

KU
KASETSART
UNIVERSITY

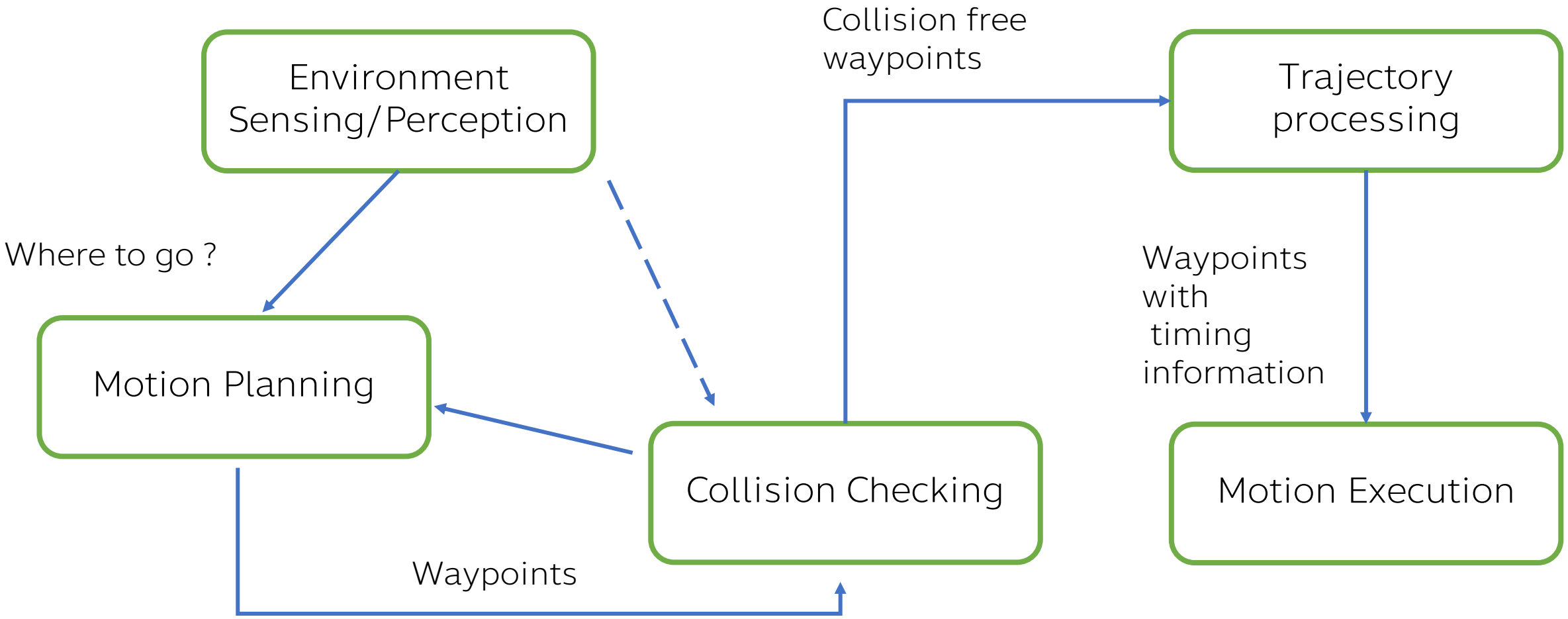
Robot Manipulator Concept

Interact with the environment and modify it using a robot to perform useful task

- Pick and place object
- CNC milling, wire cut
- Fasten two parts of car

Robot Manipulator Concept

Manipulation – function modules

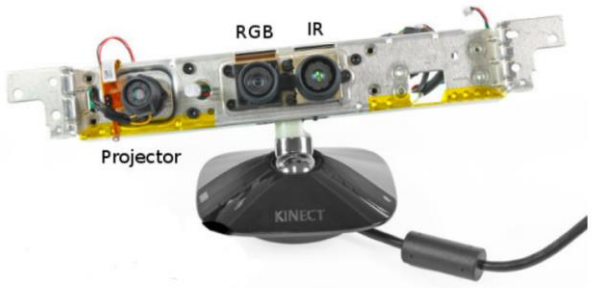


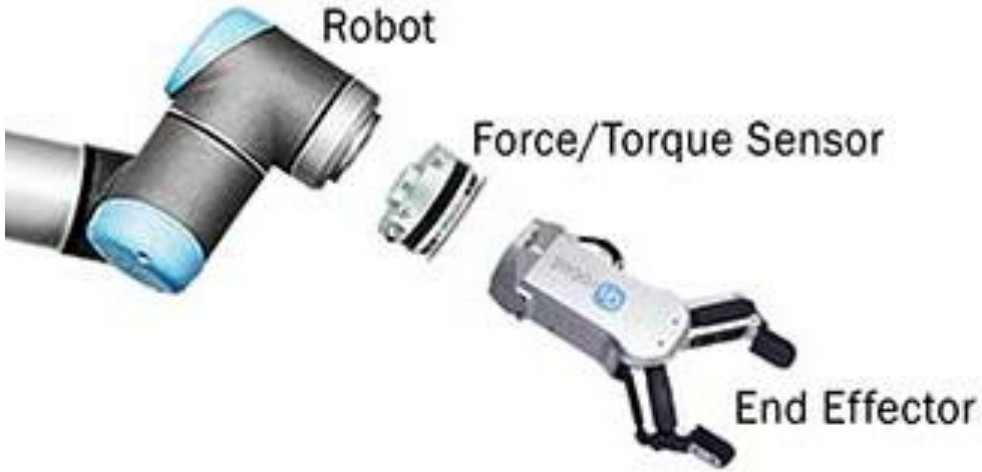
Robot Manipulator Concept

- Kinect
- Lidar

Environment Sensing/Perception

3D sensor, Camera, Force sensor

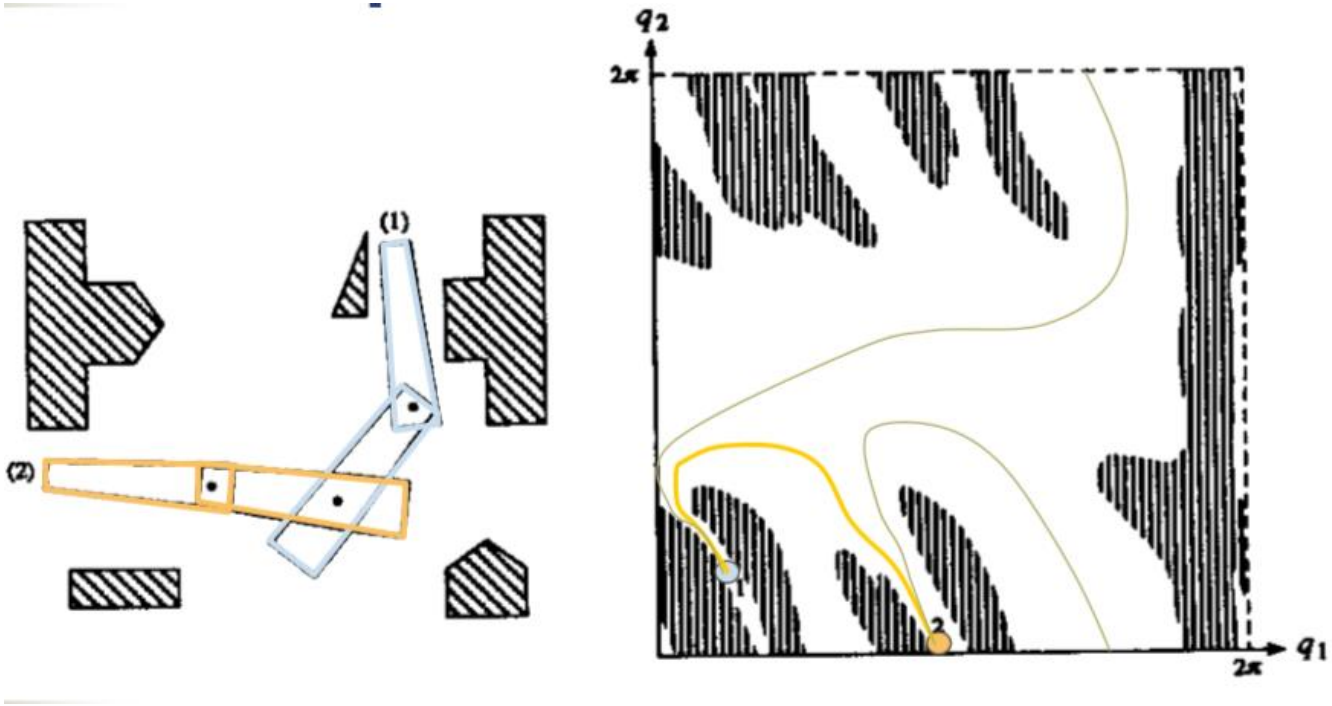


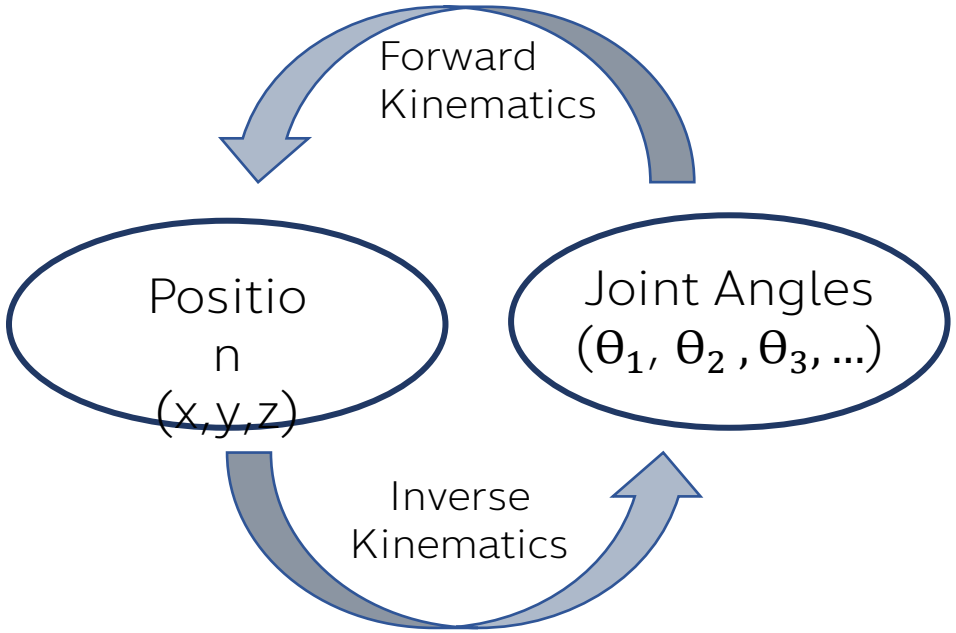


Robot Manipulator Concept

- OMPL library
- CHOMP library

Motion Planning

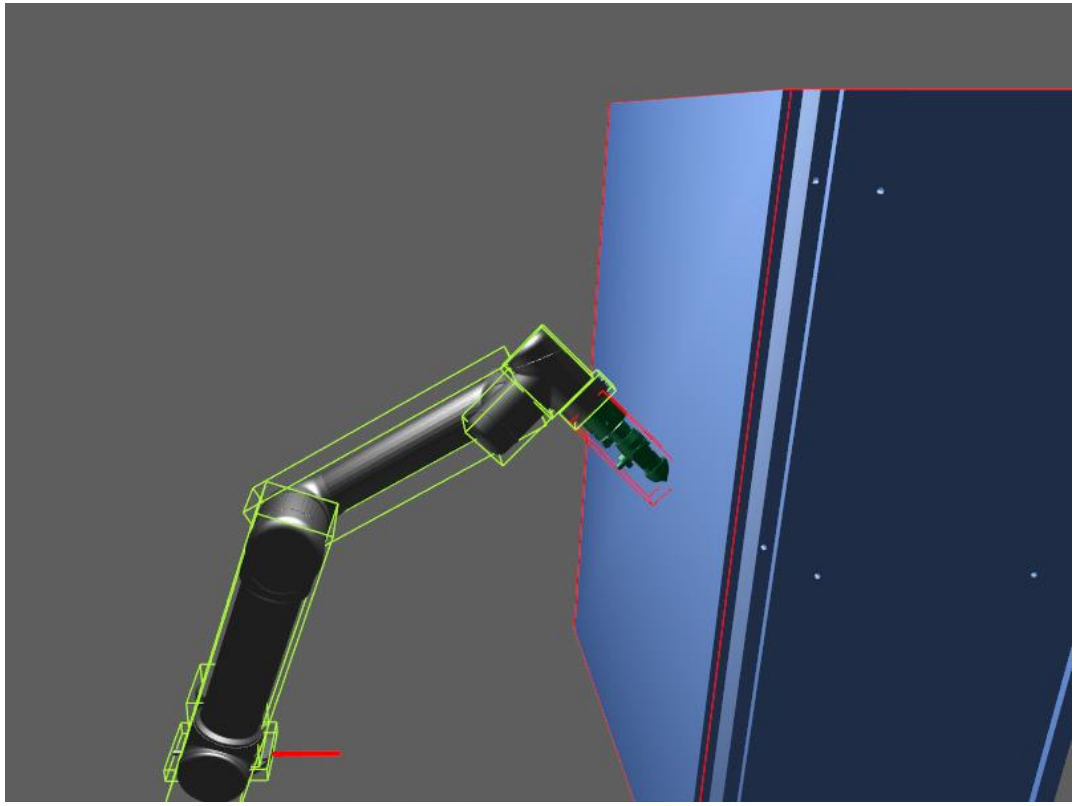


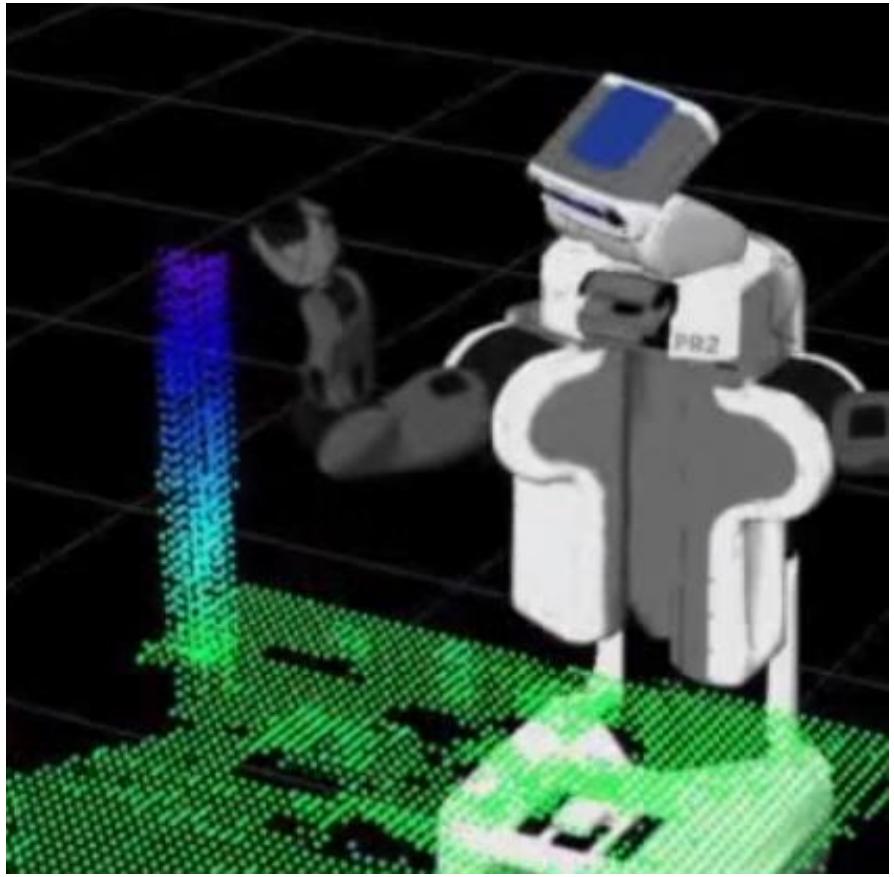


Robot Manipulator Concept

Collision Checking

- FCL library
- Bullet library

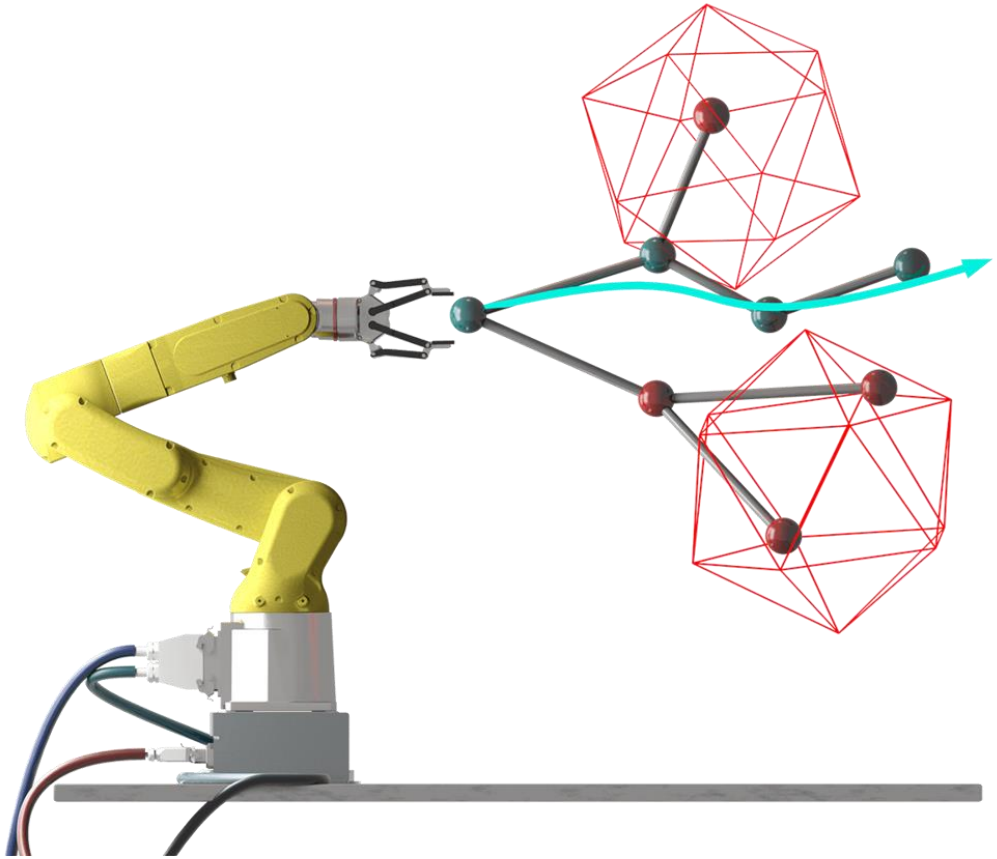


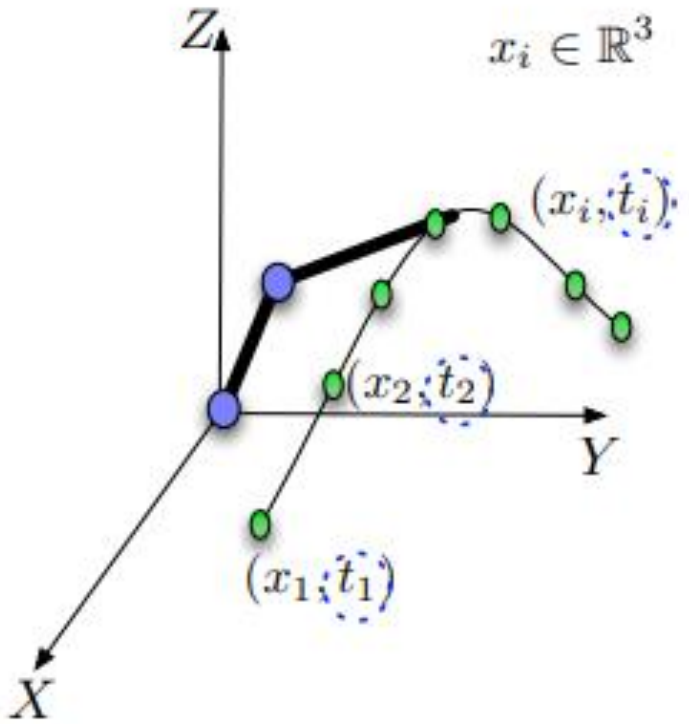


Robot Manipulator Concept

- IPTP library
- TOPP library

Trajectory processing

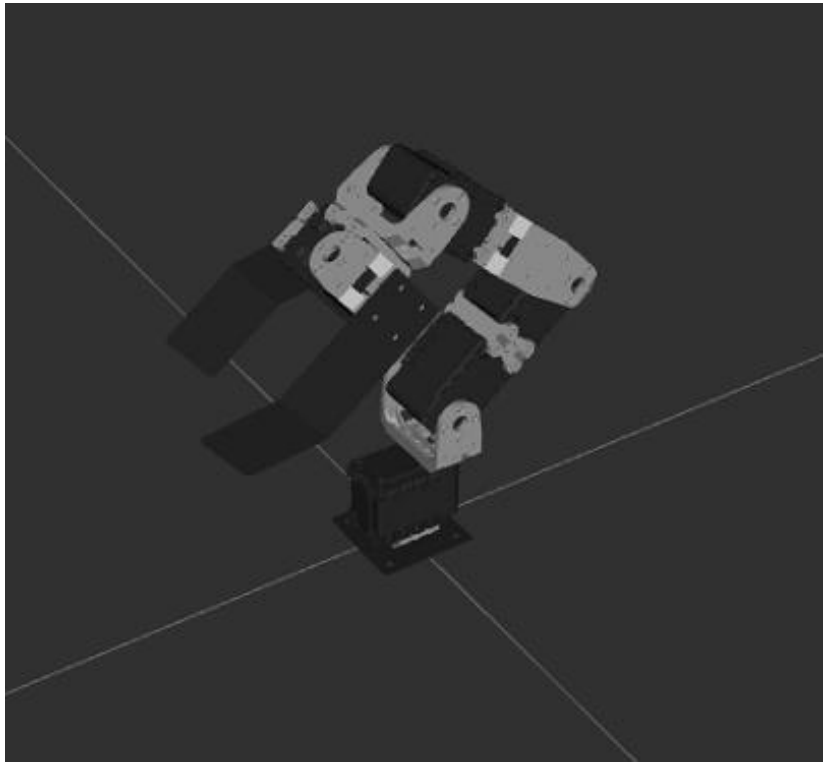




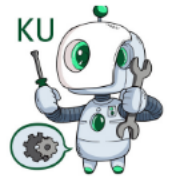
Robot Manipulator Concept

- Real Robot
  (Dynamixel Controller)
- Simulation

Motion Execution

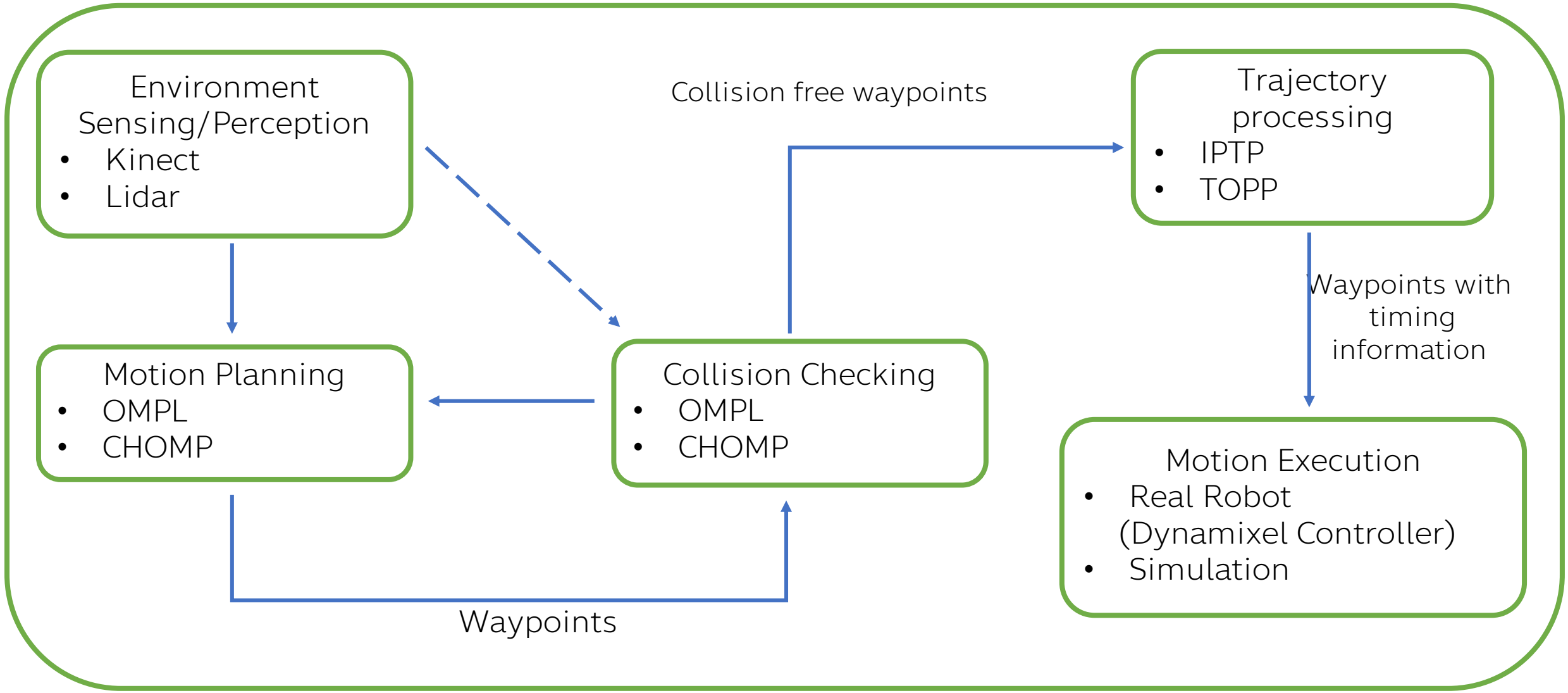


MoveIt!



KU
KASETSART
UNIVERSITY

MoveIt!


Environment Sensing/Perception
• Kinect
• Lidar
Collision free waypoints
Trajectory processing
• IPTP
• TOPP
Waypoints with timing information
Motion Planning
• OMPL
• CHOMP
Collision Checking
• OMPL
• CHOMP
Motion Execution
• Real Robot (Dynamixel Controller)
• Simulation
Waypoints
MoveIt!

MoveIt!

Install MoveIt! for ROS kinetic

```
$ sudo apt-get install ros-kinetic-moveit
```

ROBOTIS DYNAMIXEL

Install Dynamixel controller, msg, driver

```
$ sudo apt-get install ros-kinetic-dynamixel-*
```

MoveIt!

Go to source directory in your workspace

```
$ cd ~/ros_workshop_ws/src
```

Download URDF

```
$ git clone https://github.com/gbiggs/crane_plus_arm.git
```

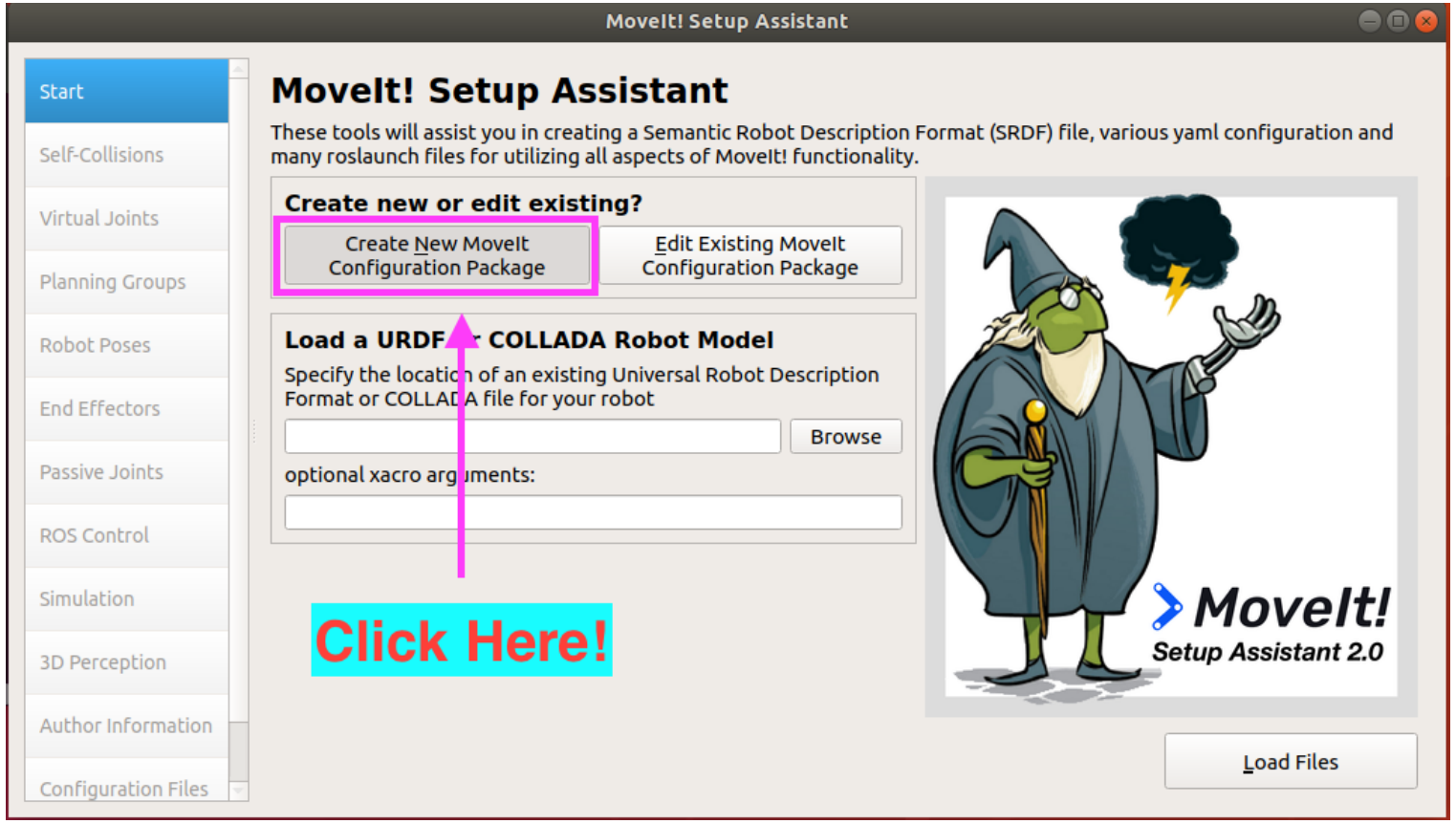
Then go to your workspace and run

```
$ cd ..
```

```
$ catkin_make
```

MoveIt!

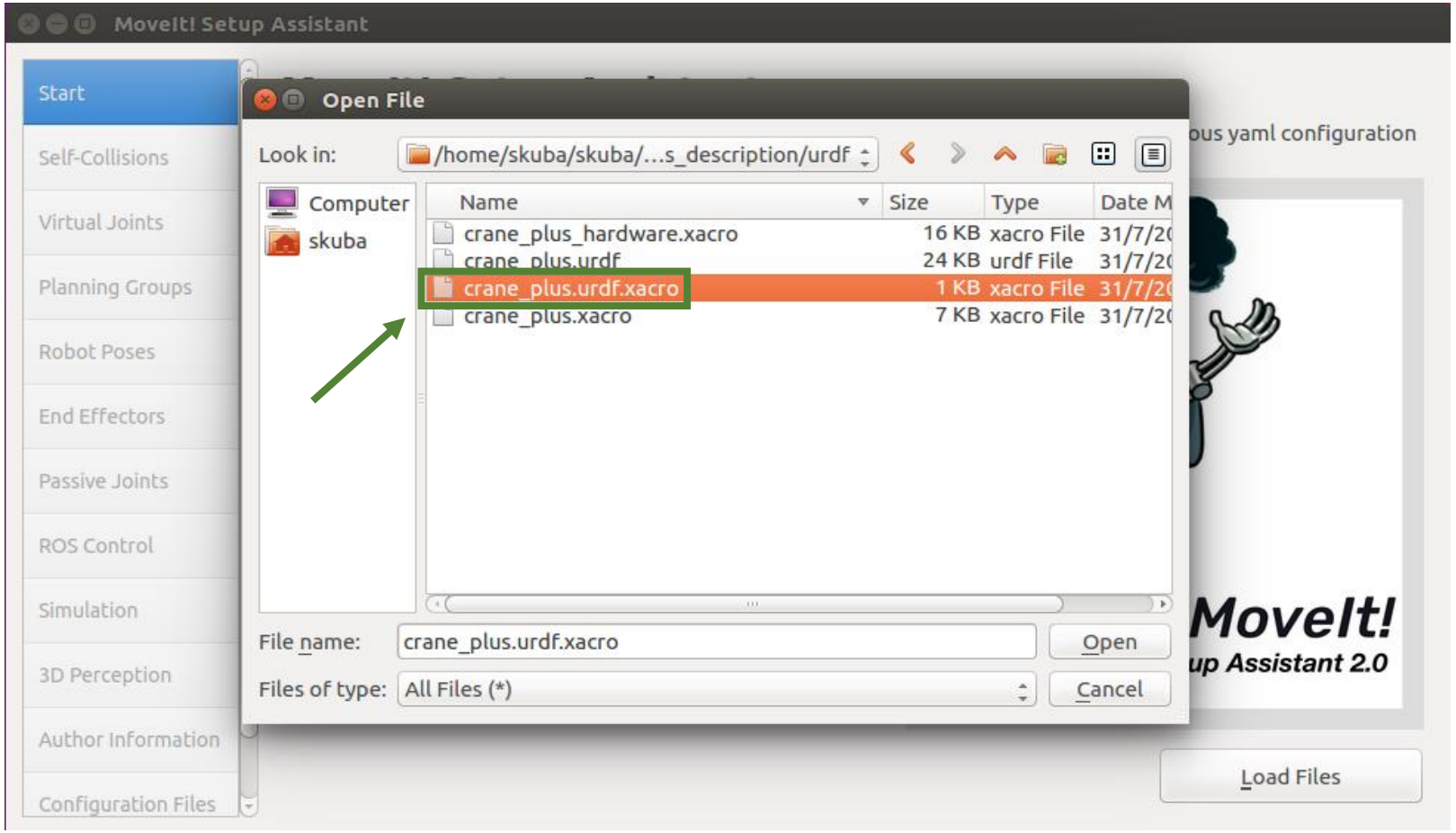
```
$ roslaunch moveit_setup_assistant setup_assistant.launch
```



MoveIt!

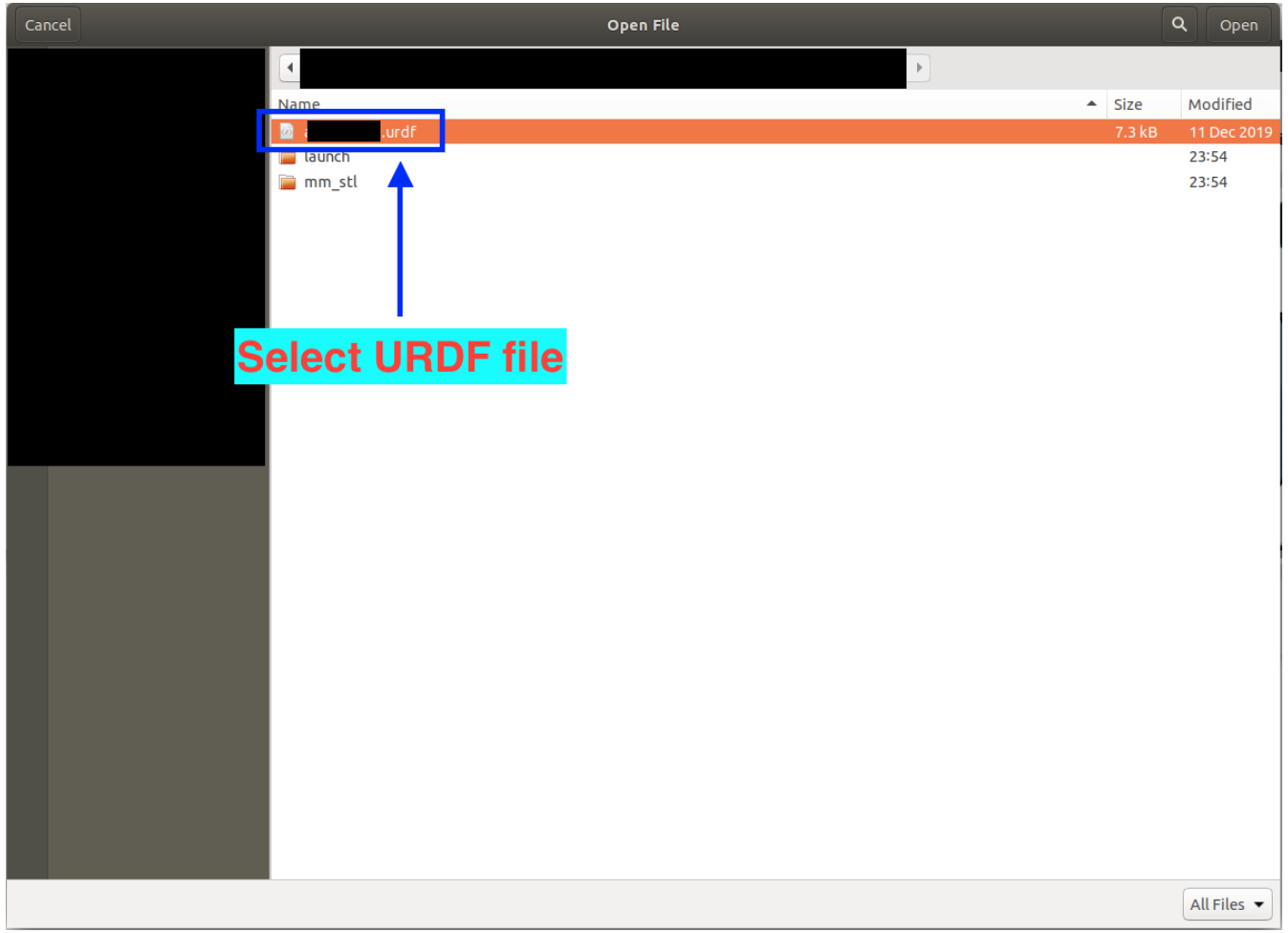
Select URDF File

~/ros_workshop_ws/src/crane_plus_arm/crane_plus_description/urdf/crane_plus.urdf.xacro



MoveIt!

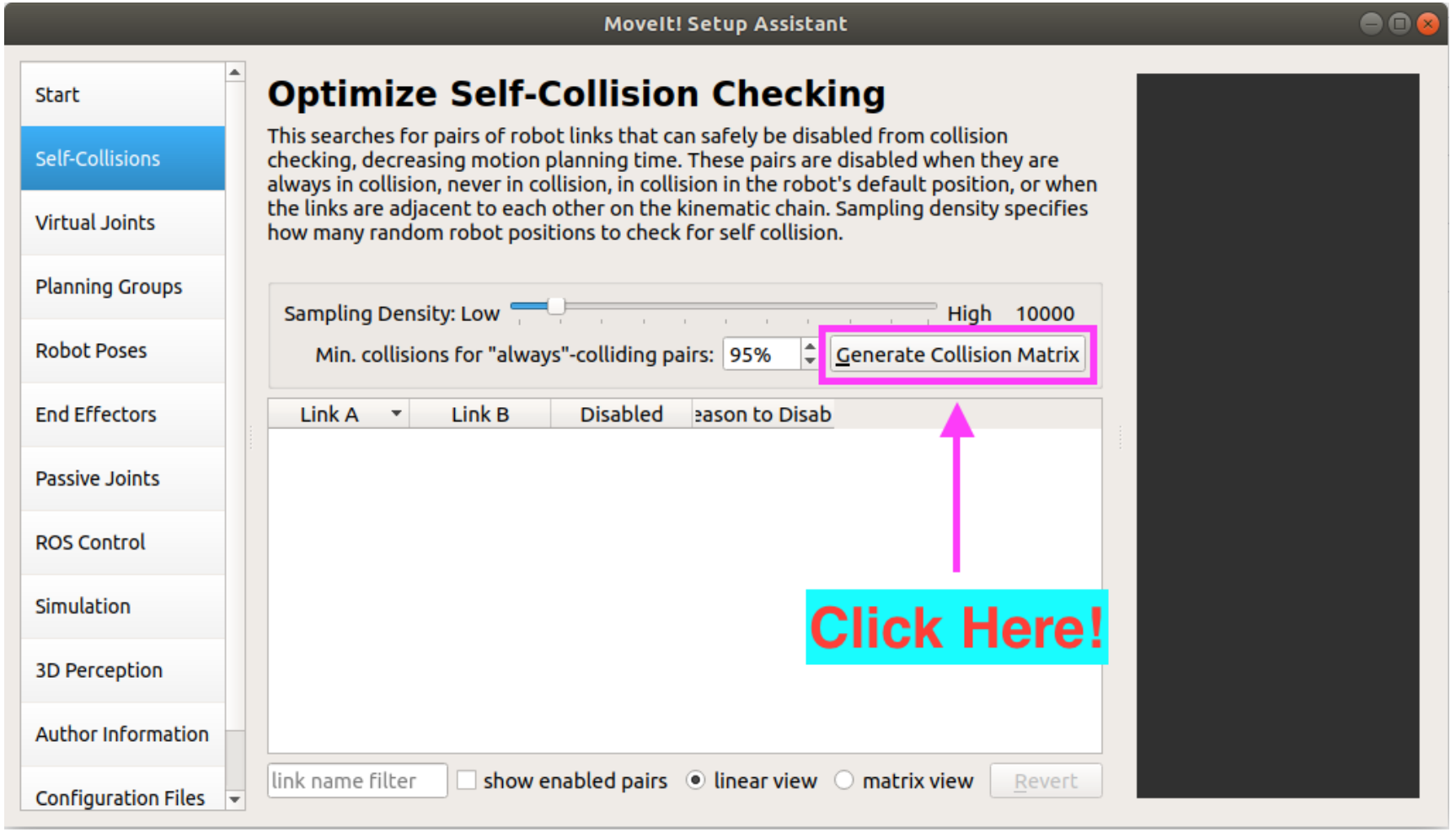
For another URDF file



You can export your CAD file
To the URDF File by using
the open source library

- For Fusion 360
  - https://github.com/syuntoku14/fusion2urdf/tree/master
- For Solidworks
  - http://wiki.ros.org/sw_urdf_exporter

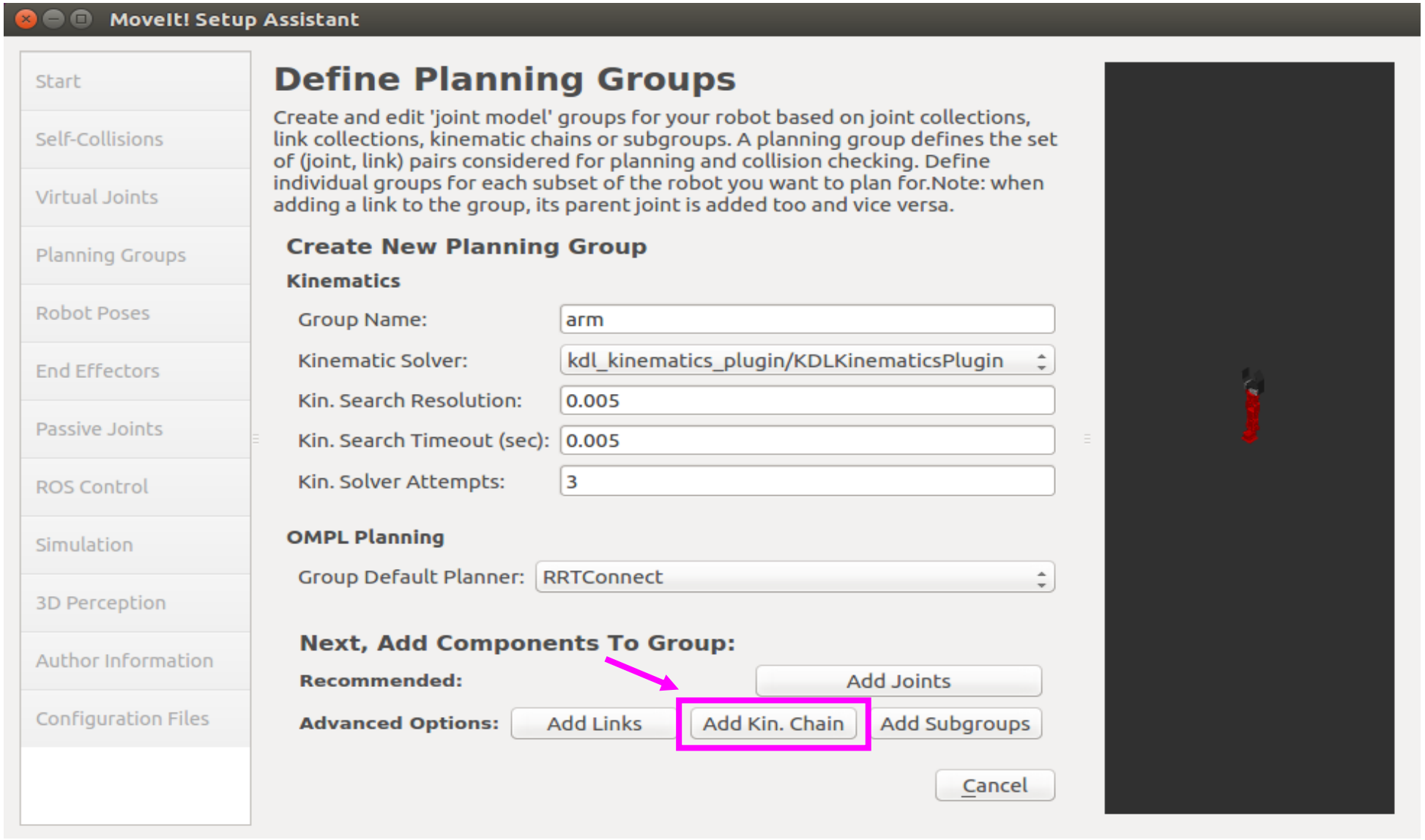
MoveIt!


MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
Simulation
3D Perception
Author Information
Configuration Files
Optimize Self-Collision Checking
This searches for pairs of robot links that can safely be disabled from collision checking, decreasing motion planning time. These pairs are disabled when they are always in collision, never in collision, in collision in the robot's default position, or when the links are adjacent to each other on the kinematic chain. Sampling density specifies how many random robot positions to check for self collision.
Sampling Density: Low
High
10000
Min. collisions for "always"-colliding pairs:
95%
Generate Collision Matrix
Link A
Link B
Disabled
eason to Disab
Click Here!
link name filter
show enabled pairs
linear view
matrix view
Revert

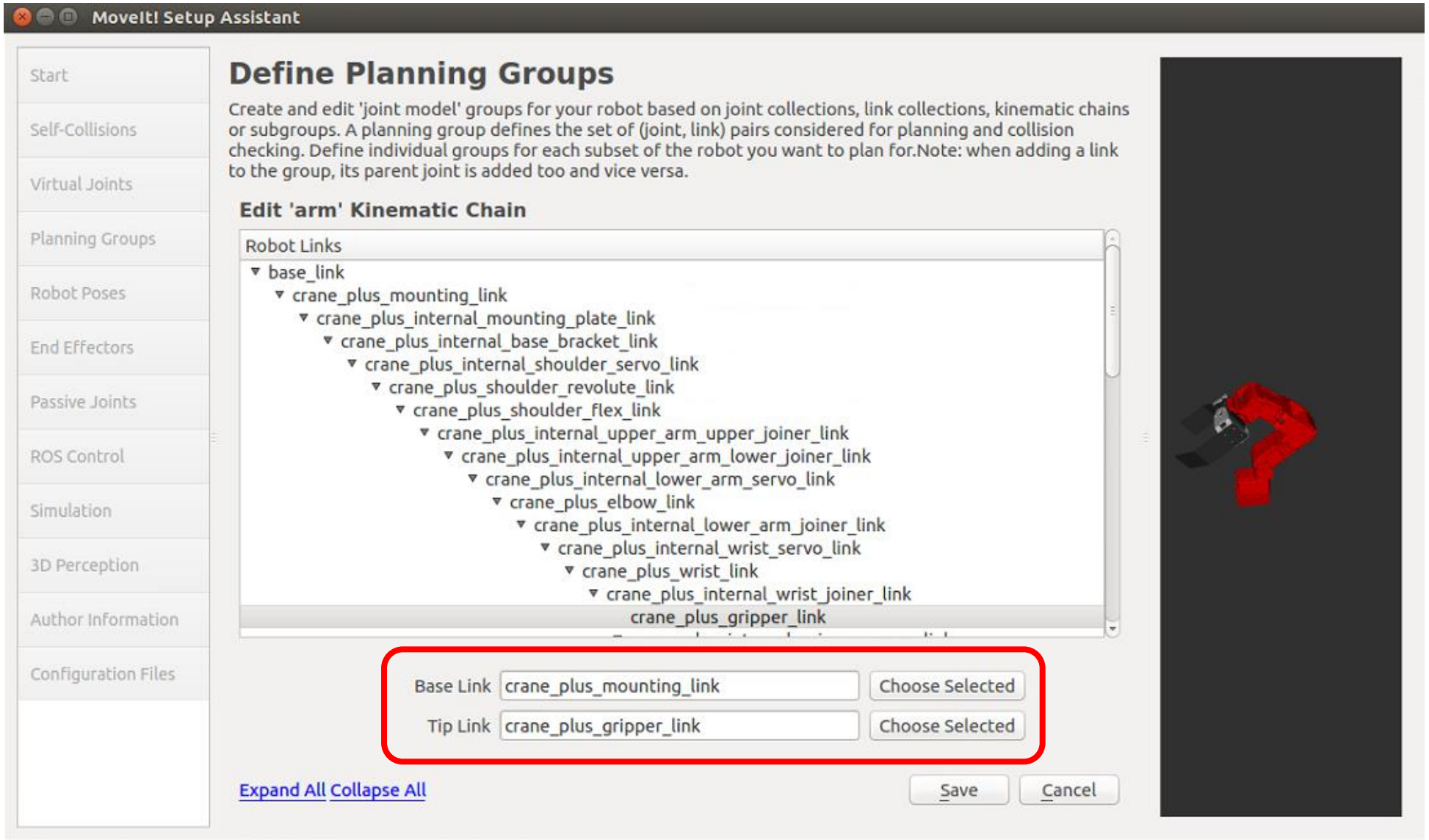
MoveIt!

Add Planning Groups

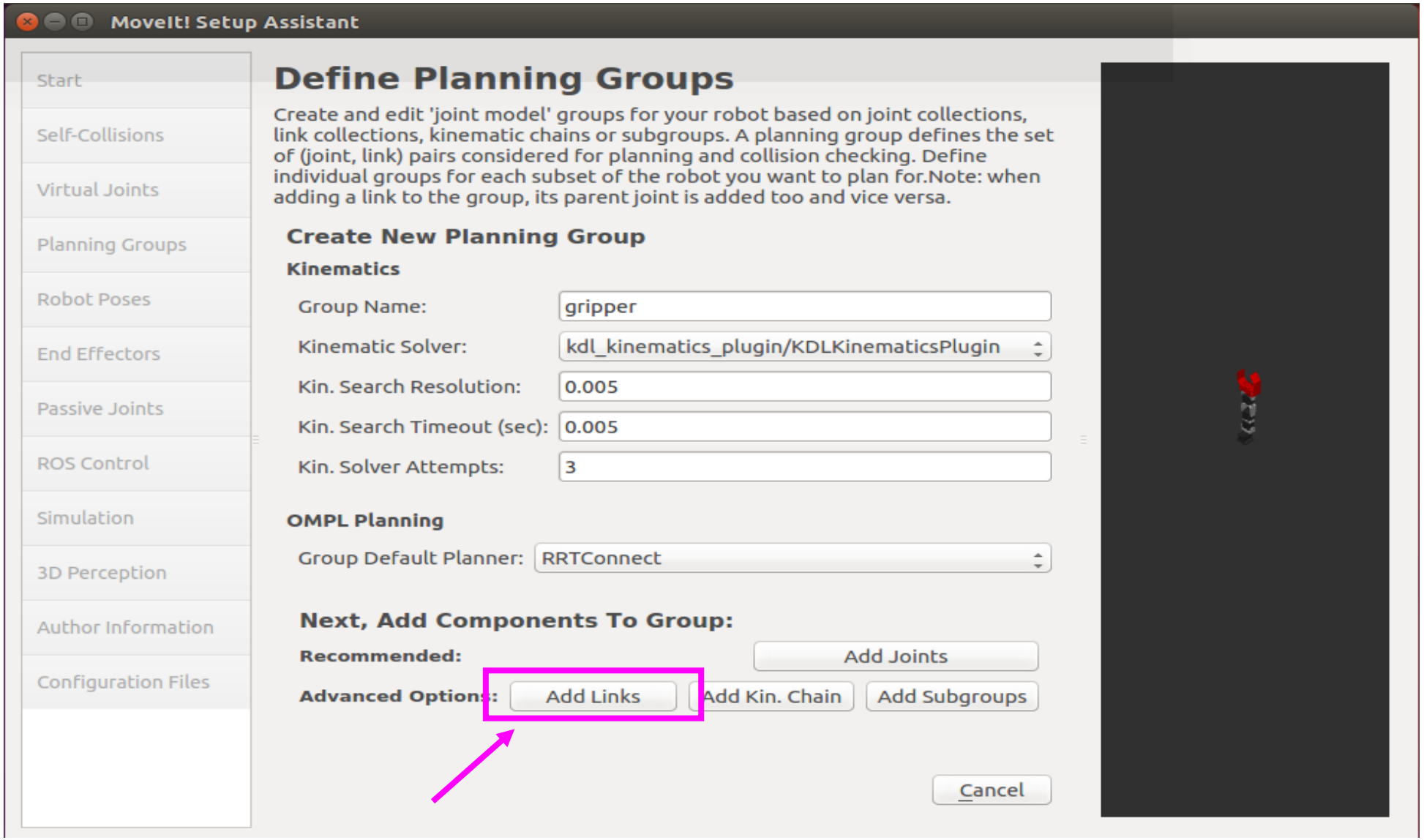
MoveIt!


MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
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: arm
Kinematic Solver: kdl_kinematics_plugin/KDLKinematicsPlugin
Kin. Search Resolution: 0.005
Kin. Search Timeout (sec): 0.005
Kin. Solver Attempts: 3
OMPL Planning
Group Default Planner: RRTConnect
Next, Add Components To Group:
Recommended:
Add Joints
Advanced Options:
Add Links
Add Kin. Chain
Add Subgroups
Cancel

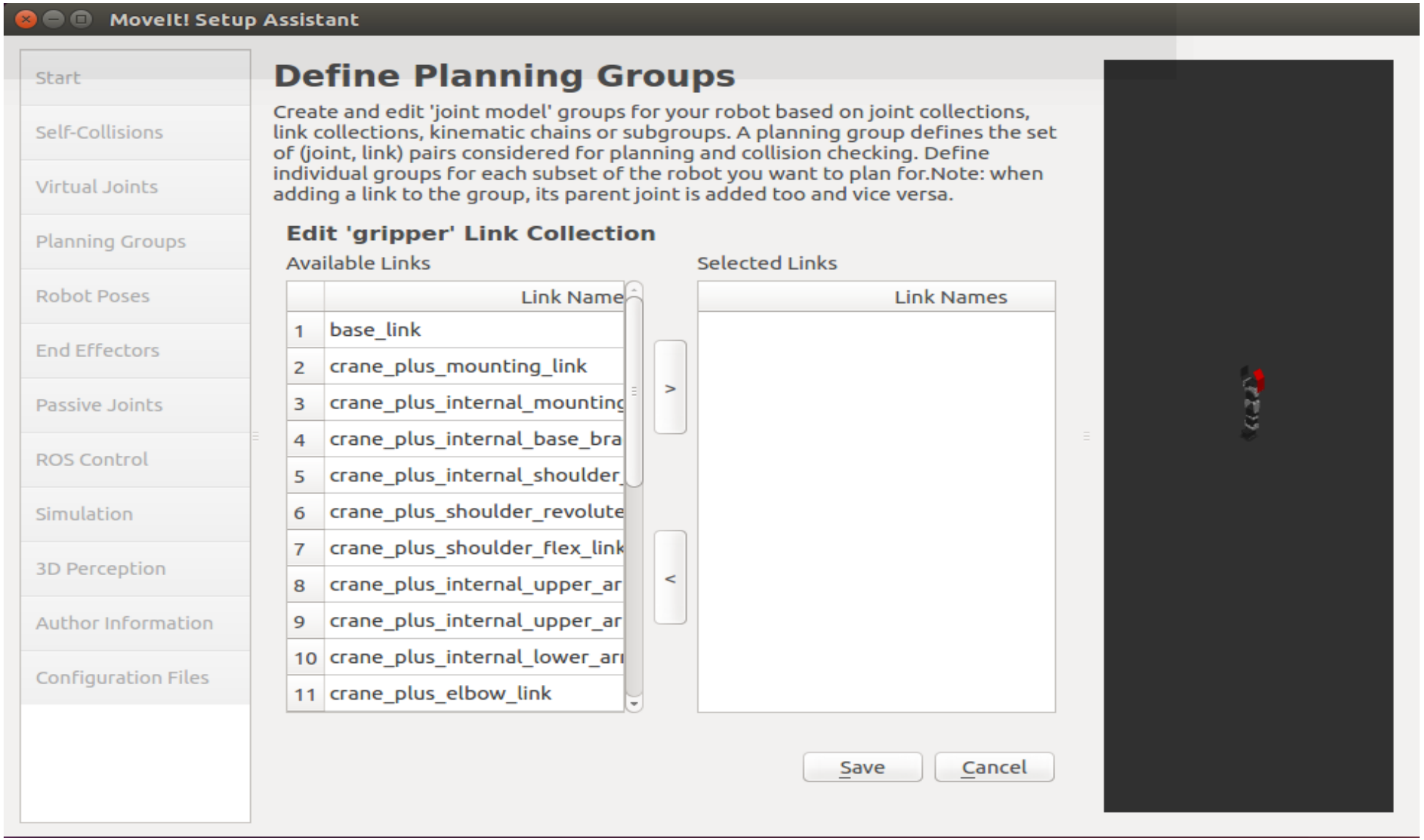
MoveIt!


MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
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.
Edit 'arm' Kinematic Chain
Robot Links
base_link
crane_plus_mounting_link
crane_plus_internal_mounting_plate_link
crane_plus_internal_base_bracket_link
crane_plus_internal_shoulder_servo_link
crane_plus_shoulder_revolute_link
crane_plus_shoulder_flex_link
crane_plus_internal_upper_arm_upper_joiner_link
crane_plus_internal_upper_arm_lower_joiner_link
crane_plus_internal_lower_arm_servo_link
crane_plus_elbow_link
crane_plus_internal_lower_arm_joiner_link
crane_plus_internal_wrist_servo_link
crane_plus_wrist_link
crane_plus_internal_wrist_joiner_link
crane_plus_gripper_link
Base Link
crane_plus_mounting_link
Choose Selected
Tip Link
crane_plus_gripper_link
Choose Selected
Expand All
Collapse All
Save
Cancel

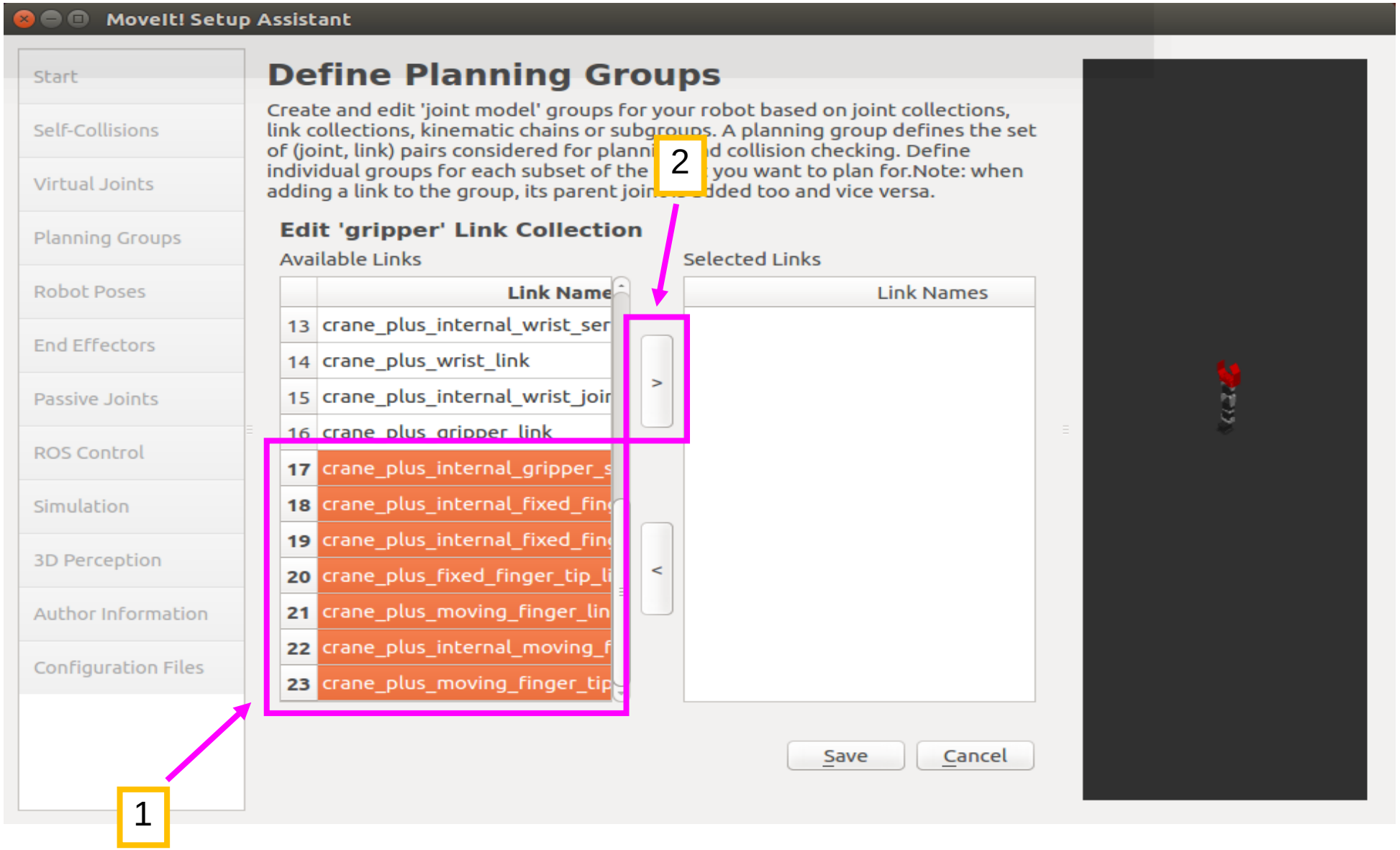
MoveIt!


MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
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: gripper
Kinematic Solver: kdl_kinematics_plugin/KDLKinematicsPlugin
Kin. Search Resolution: 0.005
Kin. Search Timeout (sec): 0.005
Kin. Solver Attempts: 3
OMPL Planning
Group Default Planner: RRTConnect
Next, Add Components To Group:
Recommended:
Add Joints
Advanced Options:
Add Links
Add Kin. Chain
Add Subgroups
Cancel

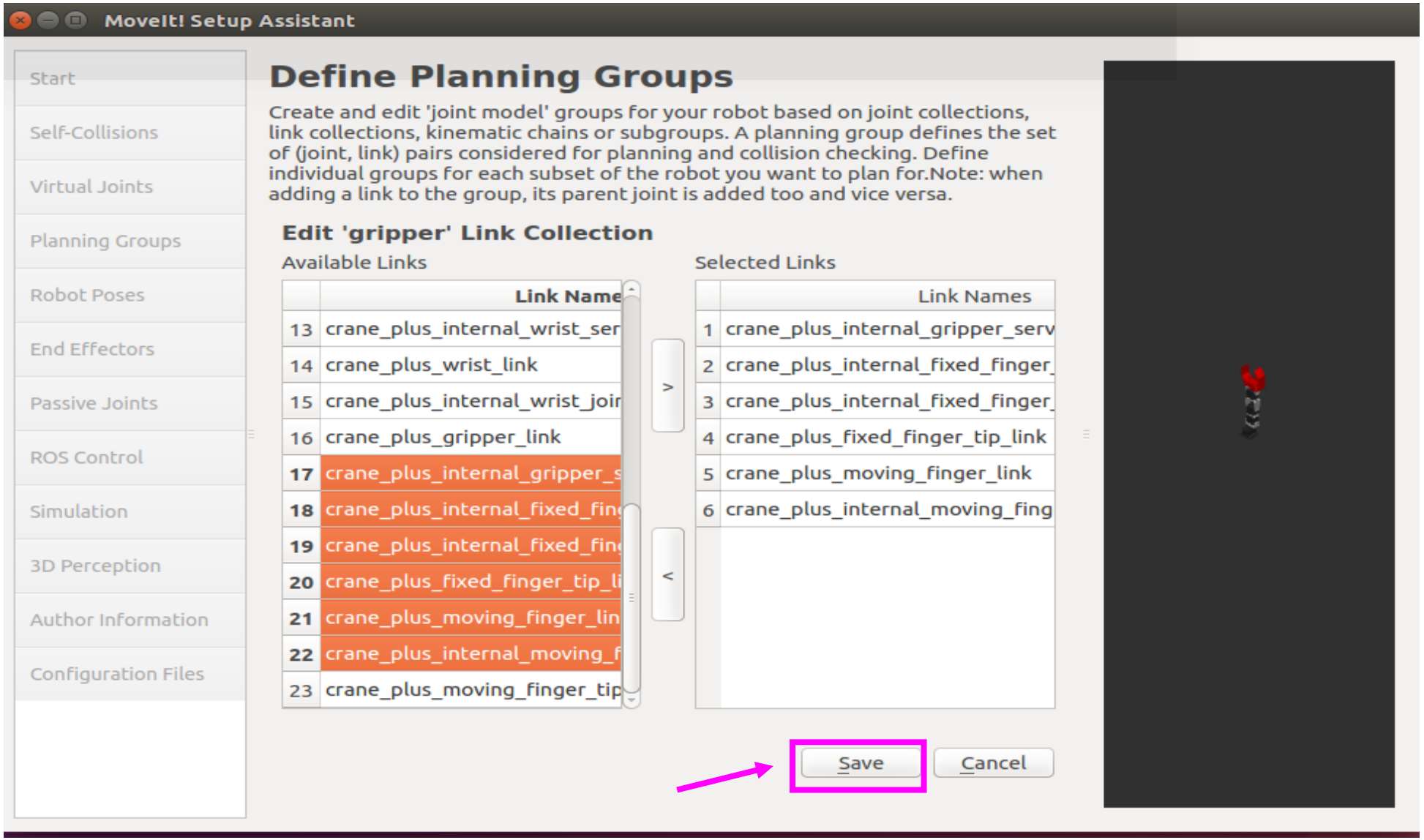
MoveIt!


MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
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.
Edit 'gripper' Link Collection
Available Links
Link Name
1 base_link
2 crane_plus_mounting_link
3 crane_plus_internal_mounting
4 crane_plus_internal_base_bra
5 crane_plus_internal_shoulder_
6 crane_plus_shoulder_revolute
7 crane_plus_shoulder_flex_link
8 crane_plus_internal_upper_ar
9 crane_plus_internal_upper_ar
10 crane_plus_internal_lower_arı
11 crane_plus_elbow_link
>
<
Selected Links
Link Names
Save
Cancel

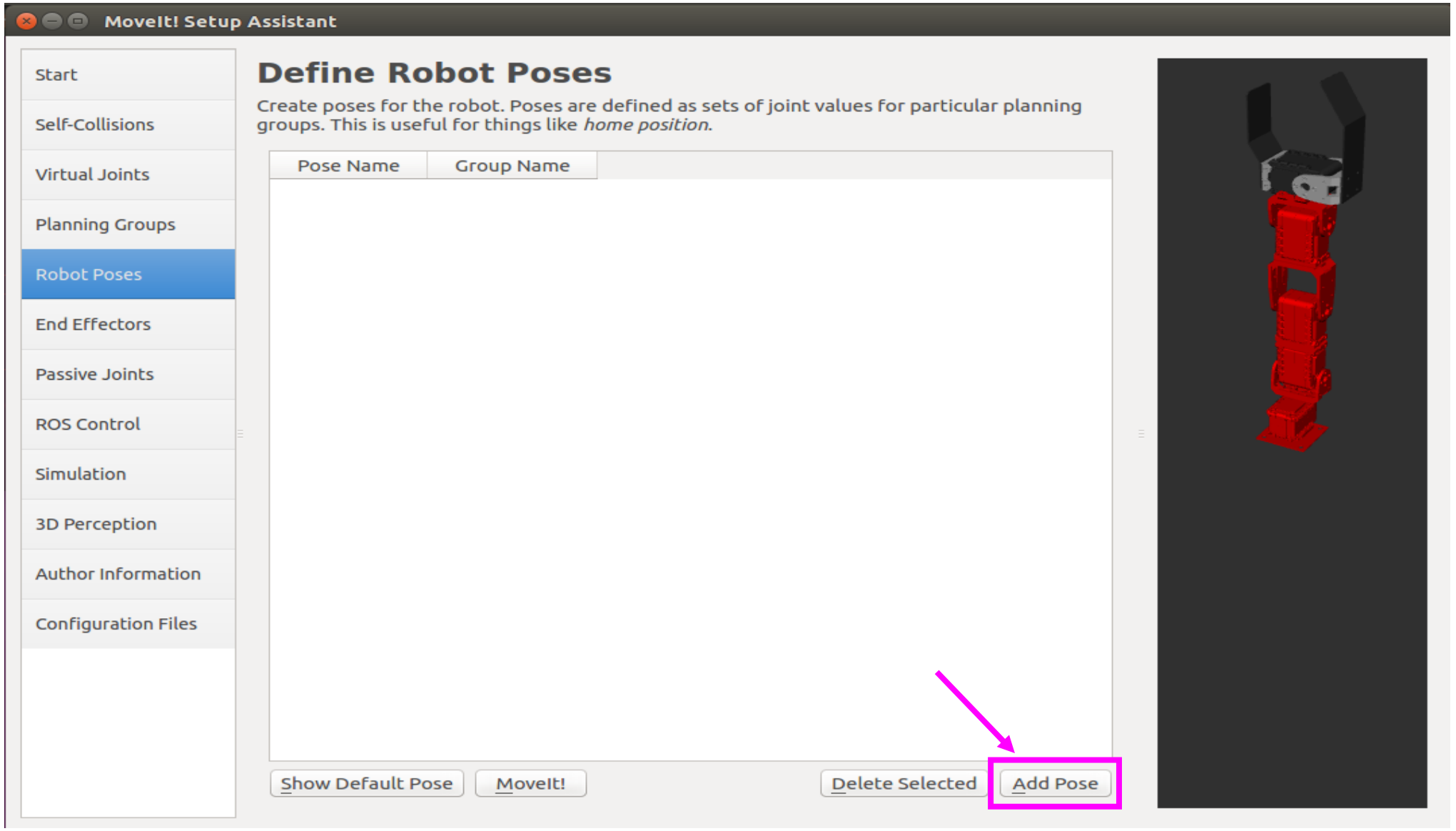
MoveIt!


MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
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 planni d collision checking. Define individual groups for each subset of the you want to plan for.Note: when adding a link to the group, its parent join dded too and vice versa.
Edit 'gripper' Link Collection
Available Links
Link Name
13 crane_plus_internal_wrist_ser
14 crane_plus_wrist_link
15 crane_plus_internal_wrist_joi
16 crane_plus_gripper_link
17 crane_plus_internal_gripper_s
18 crane_plus_internal_fixed_fin
19 crane_plus_internal_fixed_fin
20 crane_plus_fixed_finger_tip_li
21 crane_plus_moving_finger_lin
22 crane_plus_internal_moving_f
23 crane_plus_moving_finger_tip
>
<
Selected Links
Link Names
Save
Cancel
1
2

MoveIt!

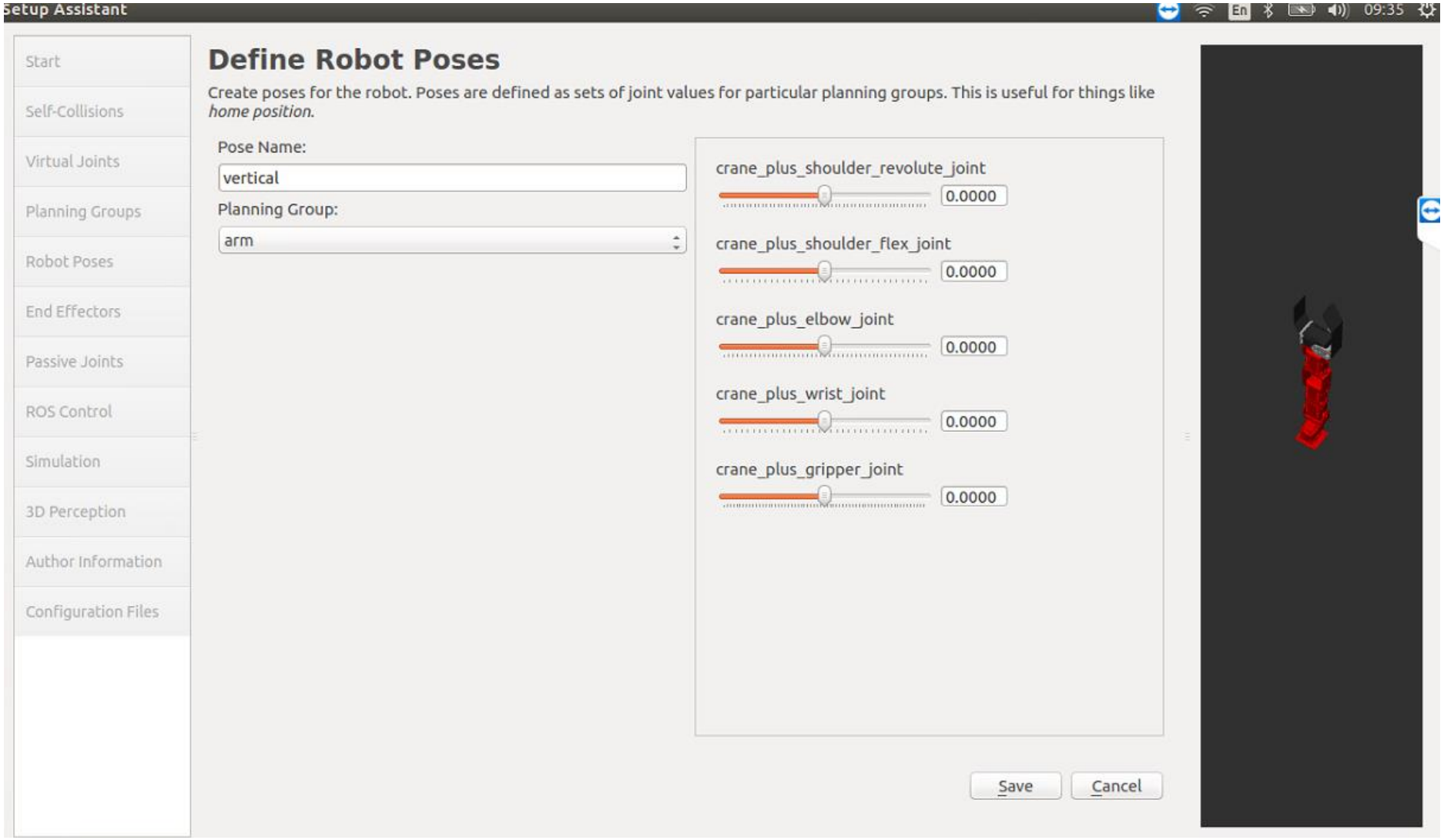

MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
Simulation
3D Perception
Author Information
Configuration Files
Define Planning Groups
Create and edit 'joint model' groups for your robot based on joint collections, link 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.
Edit 'gripper' Link Collection
Available Links
Link Name
13 crane_plus_internal_wrist_ser
14 crane_plus_wrist_link
15 crane_plus_internal_wrist_joir
16 crane_plus_gripper_link
17 crane_plus_internal_gripper_s
18 crane_plus_internal_fixed_fin
19 crane_plus_internal_fixed_fin
20 crane_plus_fixed_finger_tip_li
21 crane_plus_moving_finger_lin
22 crane_plus_internal_moving_f
23 crane_plus_moving_finger_tip
>
<
Selected Links
Link Names
1 crane_plus_internal_gripper_serv
2 crane_plus_internal_fixed_finger_
3 crane_plus_internal_fixed_finger_
4 crane_plus_fixed_finger_tip_link
5 crane_plus_moving_finger_link
6 crane_plus_internal_moving_fing
Save
Cancel

MoveIt!


MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
Simulation
3D Perception
Author Information
Configuration Files
Define Robot Poses
Create poses for the robot. Poses are defined as sets of joint values for particular planning groups. This is useful for things like home position.
Pose Name
Group Name
Show Default Pose
MoveIt!
Delete Selected
Add Pose

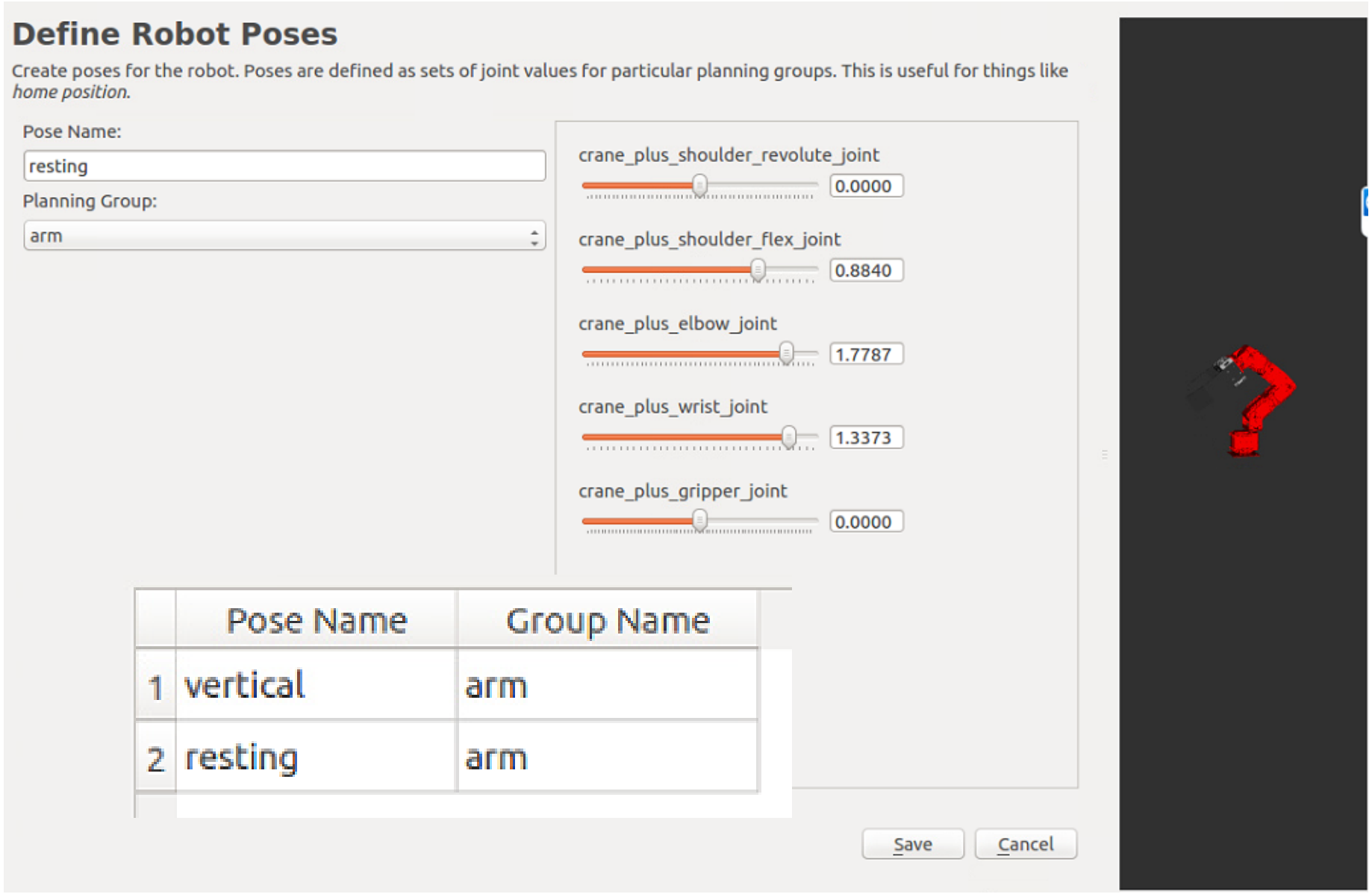
MoveIt!

Add Pose

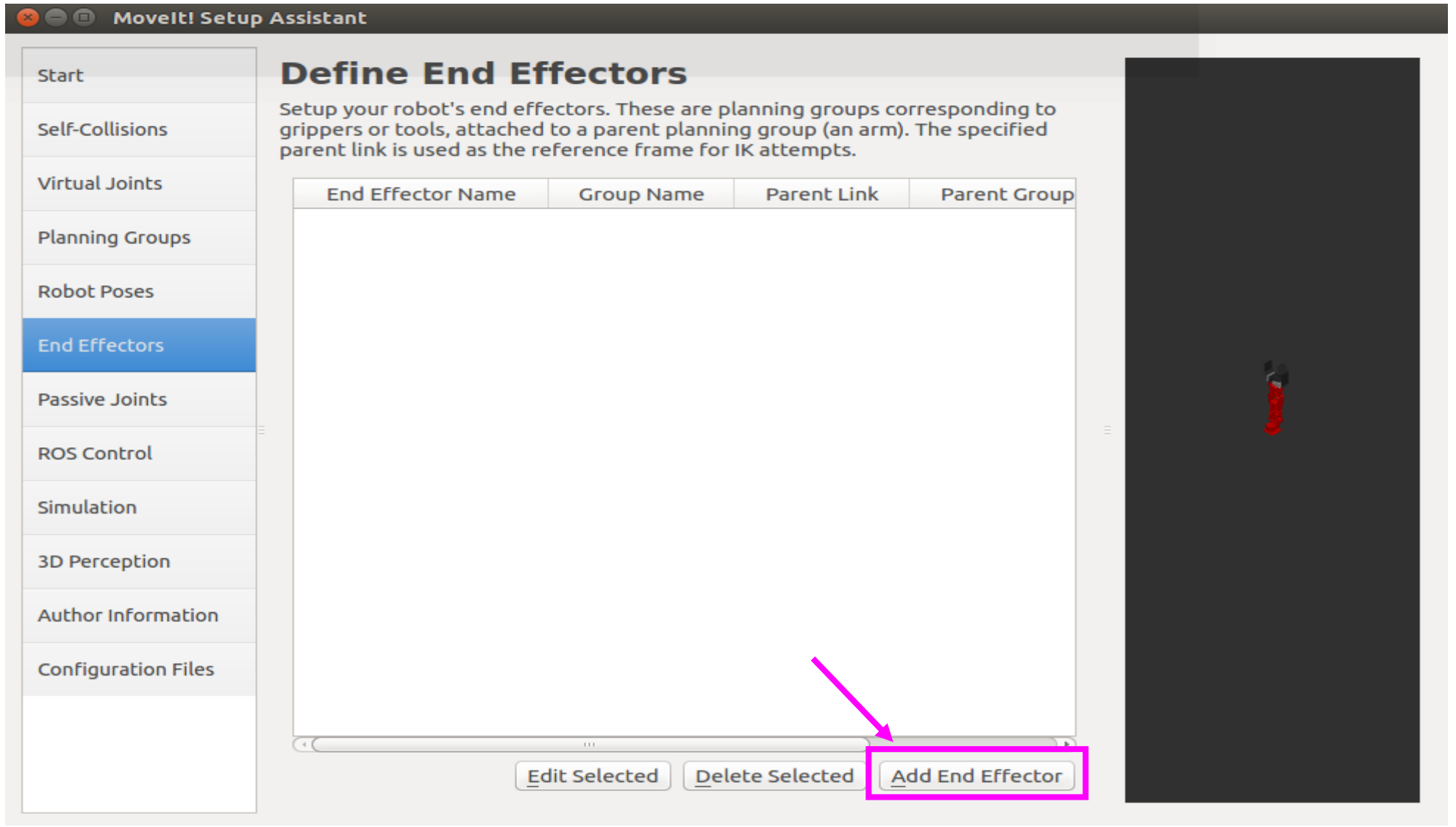


MoveIt!

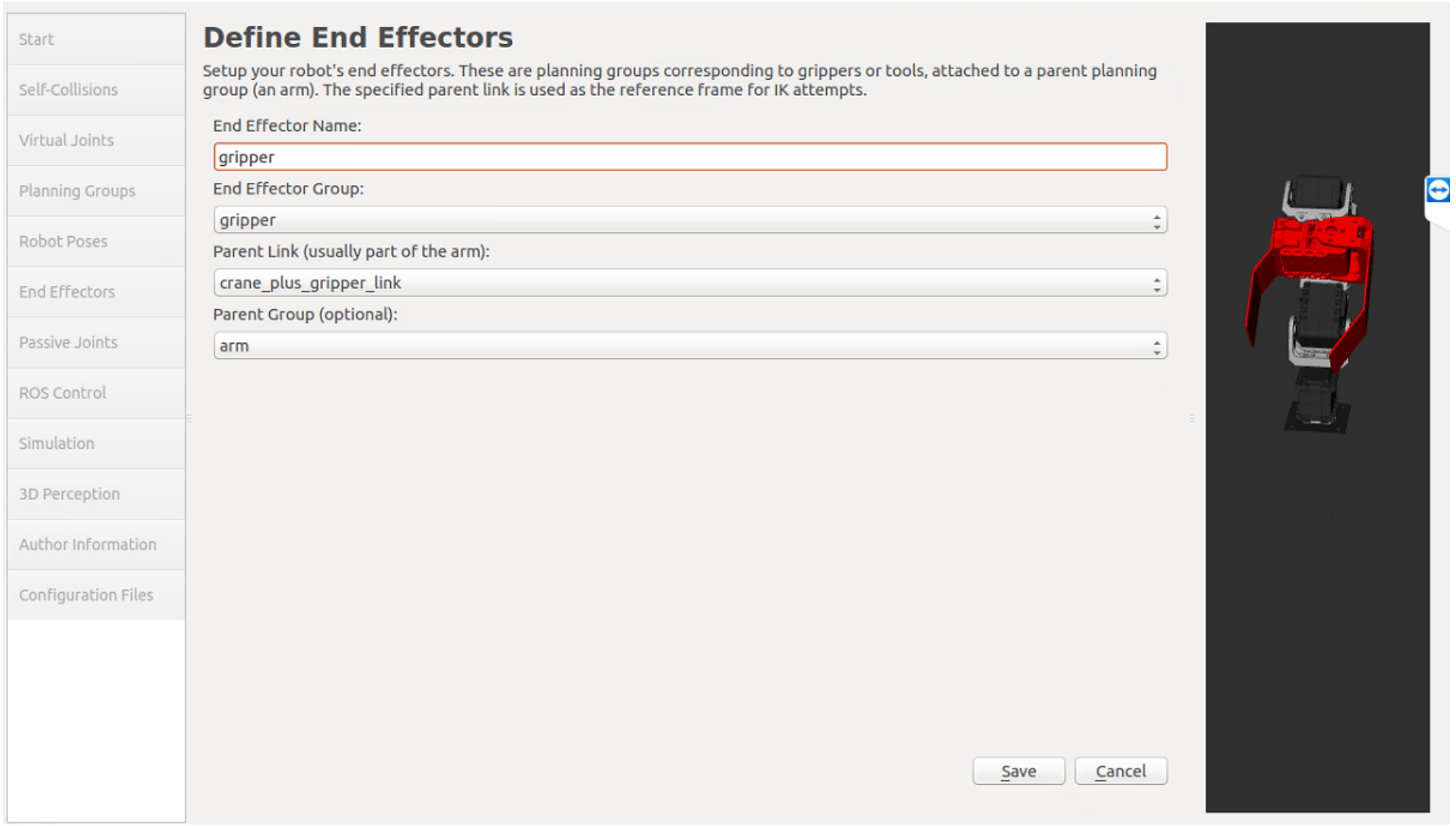
Add Pose



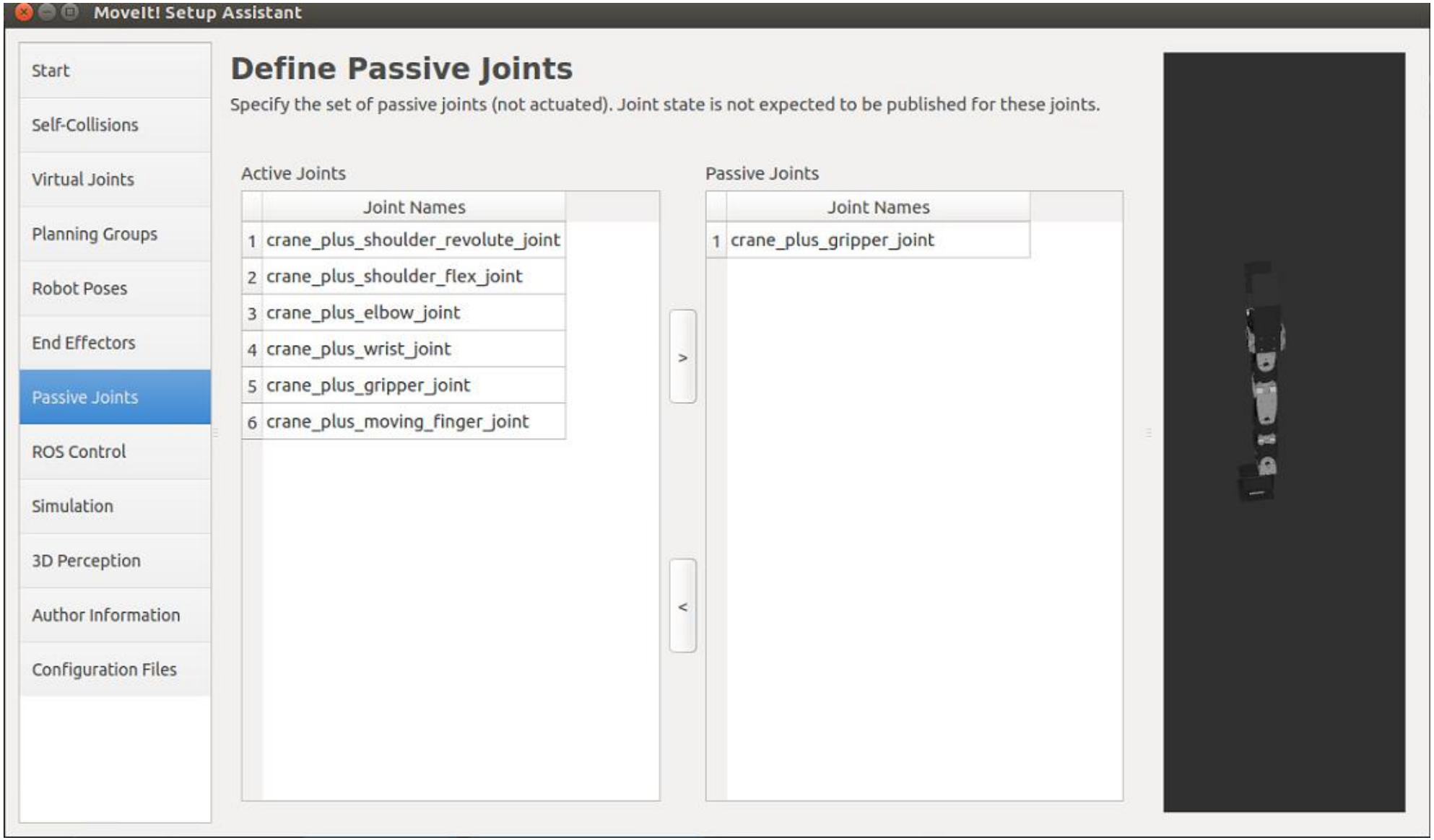
MoveIt!

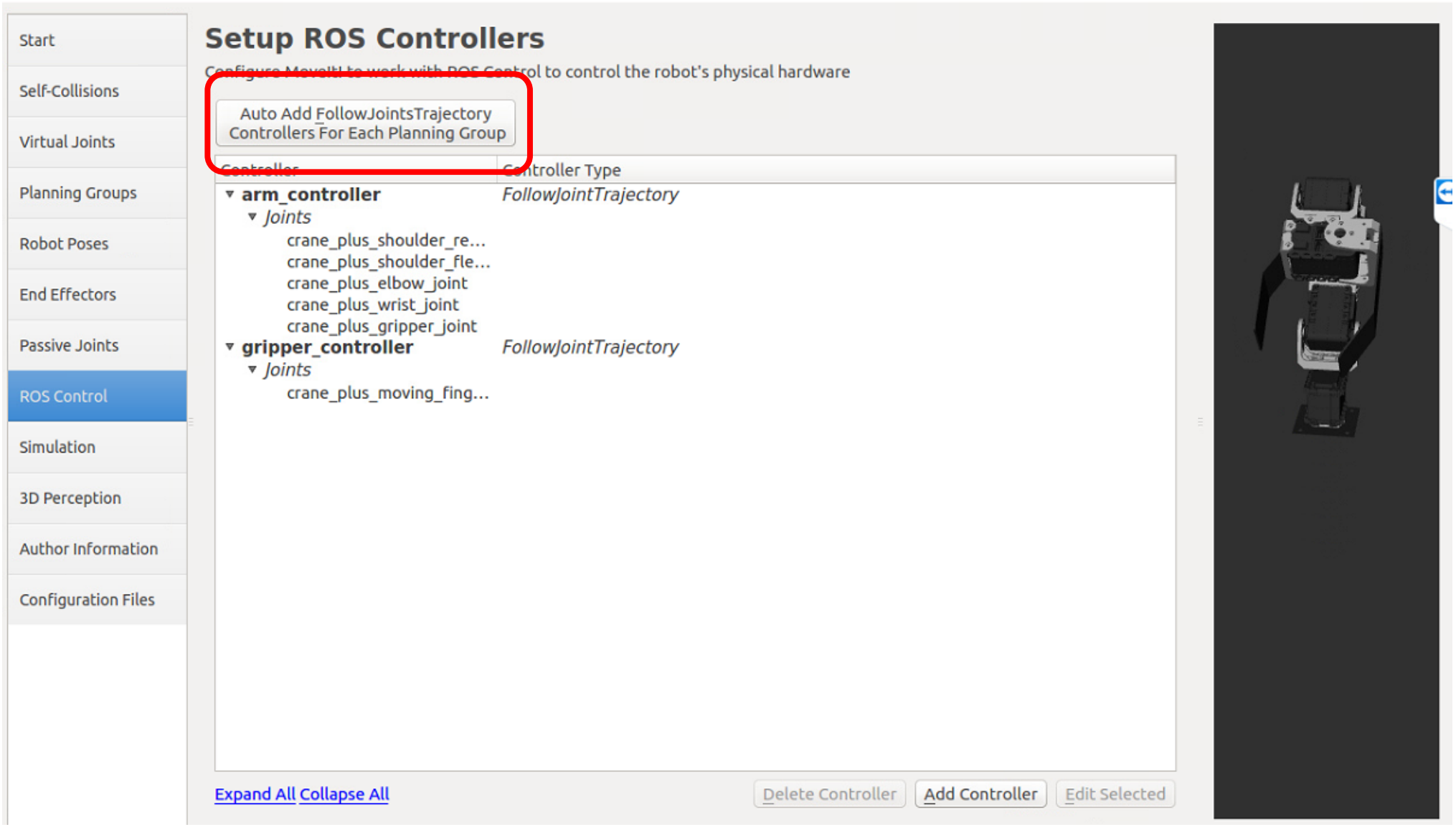

MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
Simulation
3D Perception
Author Information
Configuration Files
Define End Effectors
Setup your robot's end effectors. These are planning groups corresponding to grippers or tools, attached to a parent planning group (an arm). The specified parent link is used as the reference frame for IK attempts.
End Effector Name
Group Name
Parent Link
Parent Group
Edit Selected
Delete Selected
Add End Effector

MoveIt!


Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
Simulation
3D Perception
Author Information
Configuration Files
Define End Effectors
Setup your robot's end effectors. These are planning groups corresponding to grippers or tools, attached to a parent planning group (an arm). The specified parent link is used as the reference frame for IK attempts.
End Effector Name:
gripper
End Effector Group:
gripper
Parent Link (usually part of the arm):
crane_plus_gripper_link
Parent Group (optional):
arm
Save
Cancel

MoveIt!


MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
Simulation
3D Perception
Author Information
Configuration Files
Define Passive Joints
Specify the set of passive joints (not actuated). Joint state is not expected to be published for these joints.
Active Joints
Joint Names
1 crane_plus_shoulder_revolute_joint
2 crane_plus_shoulder_flex_joint
3 crane_plus_elbow_joint
4 crane_plus_wrist_joint
5 crane_plus_gripper_joint
6 crane_plus_moving_finger_joint
>
<
Passive Joints
Joint Names
1 crane_plus_gripper_joint


Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
Simulation
3D Perception
Author Information
Configuration Files
Setup ROS Controllers
Configure MoveIt! to work with ROS Control to control the robot's physical hardware
Auto Add FollowJointsTrajectory Controllers For Each Planning Group
Controller
Controller Type
arm_controller
FollowJointTrajectory
Joints
crane_plus_shoulder_re...
crane_plus_shoulder_fle...
crane_plus_elbow_joint
crane_plus_wrist_joint
crane_plus_gripper_joint
gripper_controller
FollowJointTrajectory
Joints
crane_plus_moving_fing...
Expand All
Collapse All
Delete Controller
Add Controller
Edit Selected

KU KASETSART UNIVERSITY

- Start
- Self-Collisions
- Virtual Joints
- Planning Groups
- Robot Poses
- End Effectors
- Passive Joints
- ROS Control
- Simulation
- 3D Perception
- Author Information
- Configuration Files

Simulate With Gazebo

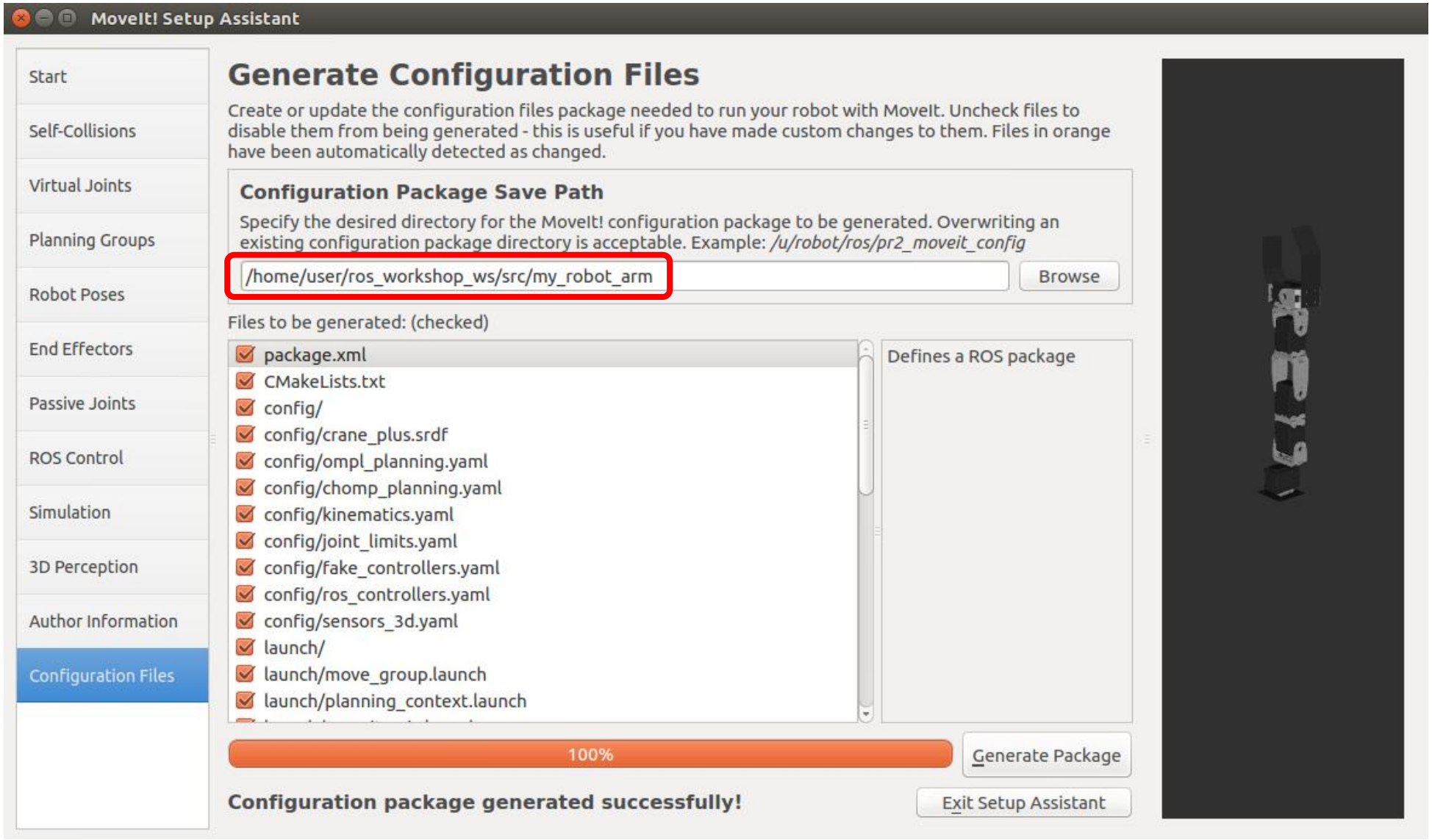
The following tool will auto-generate the URDF changes needed for Gazebo compatibility with ROSControl and MoveIt!. The needed changes are shown in green.

You can run the following command to quickly find the necessary URDF file to edit:

```
roscd crane_plus_description
```

Generate URDF

MoveIt!


MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
ROS Control
Simulation
3D Perception
Author Information
Configuration Files
Generate Configuration Files
Create or update the configuration files package needed to run your robot with MoveIt. Uncheck files to disable them from being generated - this is useful if you have made custom changes to them. Files in orange have been automatically detected as changed.
Configuration Package Save Path
Specify the desired directory for the MoveIt! configuration package to be generated. Overwriting an existing configuration package directory is acceptable. Example: /u/robot/ros/pr2_moveit_config
/home/user/ros_workshop_ws/src/my_robot_arm
Browse
Files to be generated: (checked)
package.xml
CMakeLists.txt
config/
config/crane_plus.srdf
config/ompl_planning.yaml
config/chomp_planning.yaml
config/kinematics.yaml
config/joint_limits.yaml
config/fake_controllers.yaml
config/ros_controllers.yaml
config/sensors_3d.yaml
launch/
launch/move_group.launch
launch/planning_context.launch
Defines a ROS package
100%
Generate Package
Configuration package generated successfully!
Exit Setup Assistant

MoveIt!

MoveIt!

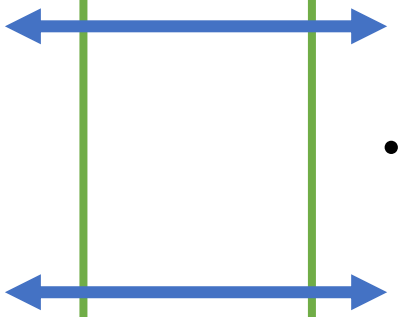
- Planning screen monitor
  - (sub /joint_state_topic)
- SimpleControllerManager
  - FollowJointTrajectory Action

Moveit_* (my_robot_arm)

RobotHardware / Simulation

- JointStateController
  - (pub /joint_state_topic)
- JointTrajectoryController
  - FollowJointTrajectory Action

(my_robot_arm/gazebo) (dynamixel_controller)



MoveIt!

Running Moveit! Demo

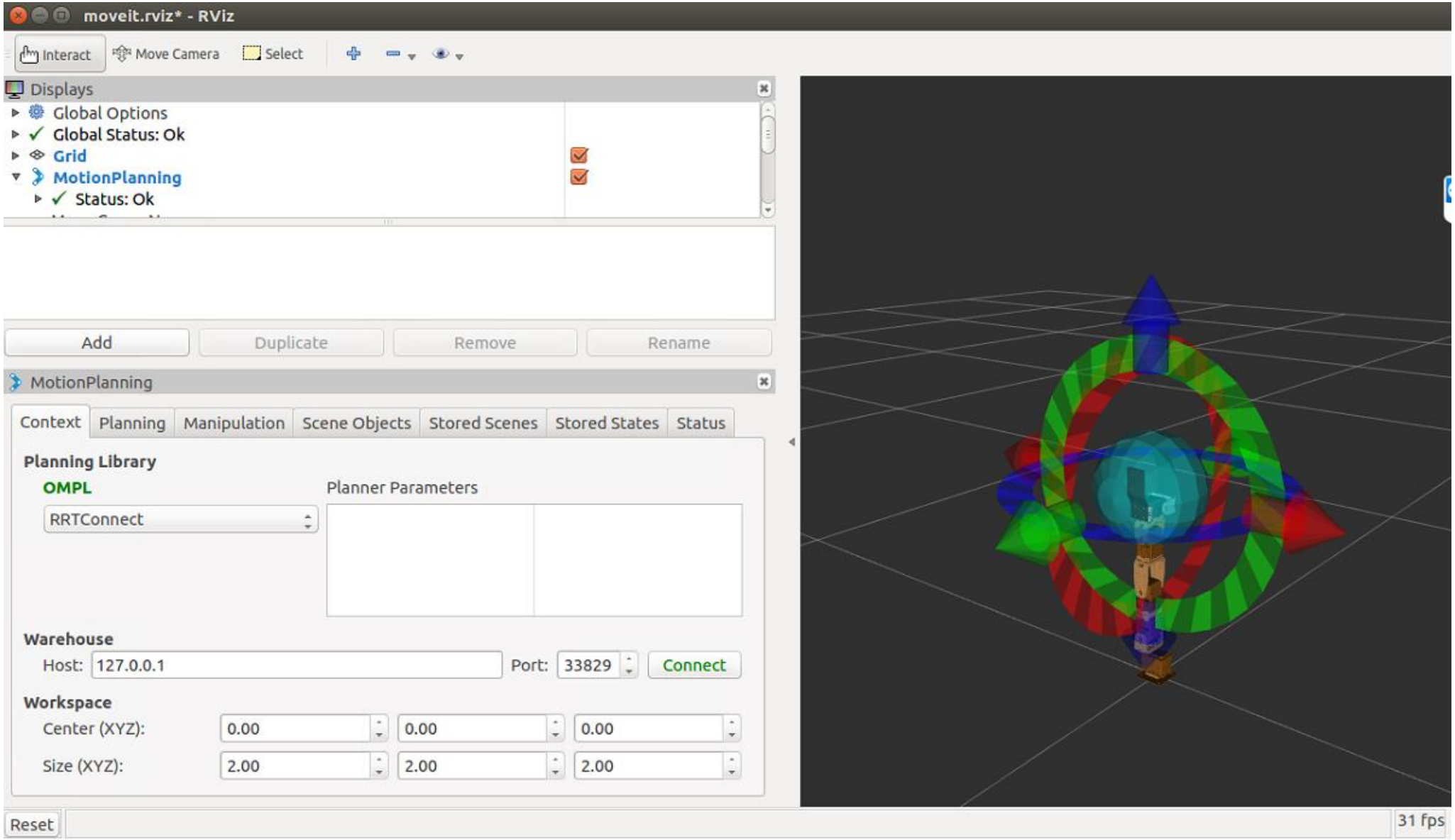
Go to workspace and do catkin_make

```
$ cd ~/ros_workshop_ws
$ catkin_make
```

Go to workspace and do catkin_make

```
$ roslaunch my_robot_arm demo.launch
```

MoveIt!


moveit.rviz* - RViz
Interact
Move Camera
Select
Displays
Global Options
Global Status: Ok
Grid
MotionPlanning
Status: Ok
Add
Duplicate
Remove
Rename
MotionPlanning
Context
Planning
Manipulation
Scene Objects
Stored Scenes
Stored States
Status
Planning Library
OMPL
Planner Parameters
RRTConnect
Warehouse
Host: 127.0.0.1
Port: 33829
Connect
Workspace
Center (XYZ): 0.00 0.00 0.00
Size (XYZ): 2.00 2.00 2.00
Reset
31 fps

MoveIt!

Moveit Commander (CLI)

Moveit! Command line tool to send motion command to robot arm.

- Started with the following command

```
$ roslaunch my_robot_arm demo.launch
```

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

- List all usable command

```
$ help
```

- Select move group to use

```
$ use arm
```

MoveIt!

Moveit Commander (CLI)

- Plan and execute motion to named joint value targets from srdf

```
$ go <named_target>
```

```
<group_state name="vertical" group="arm">
    <joint name="crane_plus_elbow_joint" value="0" />
    <joint name="crane_plus_gripper_joint" value="0" />
    <joint name="crane_plus_shoulder_flex_joint" value="0" />
    <joint name="crane_plus_shoulder_revolute_joint" value="0" />
    <joint name="crane_plus_wrist_joint" value="0" />
</group_state>
<group_state name="resting" group="arm">
    <joint name="crane_plus_elbow_joint" value="1.7787" />
    <joint name="crane_plus_gripper_joint" value="0" />
    <joint name="crane_plus_shoulder_flex_joint" value="0.884" />
    <joint name="crane_plus_shoulder_revolute_joint" value="0" />
    <joint name="crane_plus_wrist_joint" value="1.3373" />
</group_state>
```

MoveIt!

Moveit Commander (CLI)

- Plan and execute linear motions for robot end effector

```
$ go <up | down | left | right | forward | backward> <distance_in_m>
```

- Get current joint state and pose of end effector of active group

```
$ current
```

- Execute multiple commands from a script

```
$ load <path_to_script_file/script_file_name>
```

MoveIt!

Moveit Commander (CLI)

- Create file **moveit_cli_test** at home directory

```
use arm
go vertical
go resting
go forward 0.1
go backward 0.1
go vertical
```

```
$ roslaunch my_robot_arm demo.launch
```

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

```
$ load /home/<username>/moveit_cli_test
```

MoveIt!

MoveGroup Interface

Moveit Commander (CLI)

Moveit!

- Planning screne monitor
  - (sub /joint_state_topic)
- SimpleControllerManager
  - FollowJointTrajectory Action

Moveit_* (my_robot_arm)

RobotHardware / Simulation

- JointStateController
  - (pub /joint_state_topic)
- JointTrajectoryController
  - FollowJointTrajectory Action

(my_robot_arm/gazebo) (dynamixel_controller)

MoveIt!

- Add the group_state in <package_name>/config/ crane_plus.srdf file

```
<group_state name="ready_to_pick" group="arm">
    <joint name="crane_plus_elbow_joint" value="1.6176" />
    <joint name="crane_plus_gripper_joint" value="0" />
    <joint name="crane_plus_shoulder_flex_joint" value="-0.0747" />
    <joint name="crane_plus_shoulder_revolute_joint" value="-0.951" />
    <joint name="crane_plus_wrist_joint" value="1.4198" />
</group_state>
<group_state name="ready_to_place" group="arm">
    <joint name="crane_plus_elbow_joint" value="1.6176" />
    <joint name="crane_plus_gripper_joint" value="0" />
    <joint name="crane_plus_shoulder_flex_joint" value= "-0.747" />
    <joint name="crane_plus_shoulder_revolute_joint" value="1.01243" />
    <joint name="crane_plus_wrist_joint" value= "1.4198" />
</group_state>
```

MoveIt!

- Create script folder in **my_robot_arm** package

```
$ roscd my_robot_arm
$ mkdir script
```

- Create python code

```
$ cd script
$ gedit simple_arm_command.py
```

MoveIt!

```
#!/usr/bin/env python
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import actionlib
import geometry_msgs
def simple_arm_command():
    ## First initialize moveit_commander and rospy.
    moveit_commander.roscpp_initialize(sys.argv)
    rospy.init_node('simple_arm_command',anonymous=True)
    ## Instantiate a MoveGroupCommander object.  This object is an interface
    ## to one group of joints.  In this case the group refers to the joints of
    ## arm. This interface can be used to plan and execute motions on arm.
    arm_group = moveit_commander.MoveGroupCommander("arm")
    ## Action clients to the ExecuteTrajectory action server.
    arm_client = actionlib.SimpleActionClient('execute_trajectory',
    moveit_msgs.msg.ExecuteTrajectoryAction)
    arm_client.wait_for_server()
    rospy.loginfo('Execute Trajectory server is available for arm')
```

MoveIt!

- Move to named point

```
## Set a named joint configuration as the goal to plan for a move group.
## Named joint configurations are the robot poses defined
## via MoveIt! Setup Assistant.
arm_group.set_named_target("vertical")
## Plan to the desired joint-space goal using the default planner (RRTConnect).
arm_plan_home = arm_group.plan()
## Create a goal message object for the action server.
arm_goal = moveit_msgs.msg.ExecuteTrajectoryGoal()
## Update the trajectory in the goal message.
arm_goal.trajectory = arm_plan_home
## Send the goal to the action server.
arm_client.send_goal(arm_goal)
arm_client.wait_for_result()
```

MoveIt!

- Move with Cartesian path

```
## Cartesian Paths
## ^^^^^^^^^^^^^^^
## You can plan a cartesian path directly by specifying a list of waypoints
## for the end-effector to go through.
waypoints = []
# start with the current pose
current_pose = arm_group.get_current_pose()
rospy.sleep(0.5)
current_pose = arm_group.get_current_pose()
## create linear offsets to the current pose
new_eef_pose = geometry_msgs.msg.Pose()
# Manual offsets because we don't have a camera to detect objects yet.
new_eef_pose.position.x = current_pose.pose.position.x
new_eef_pose.position.y = current_pose.pose.position.y
new_eef_pose.position.z = current_pose.pose.position.z + 0.05
```

MoveIt!

- Move with Cartesian path

```
# Retain orientation of the current pose.
new_eef_pose.orientation = copy.deepcopy(current_pose.pose.orientation)
waypoints.append(new_eef_pose)
waypoints.append(current_pose.pose)
## 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 specify the jump threshold as 0.0, effectively
## disabling it.
fraction = 0.0
for count_cartesian_path in range(0,3):
  if fraction < 1.0:
    (plan_cartesian, fraction) = arm_group.compute_cartesian_path(
                                 waypoints,   # waypoints to follow
                                 0.01,        # eef_step
                                 0.0)         # jump_threshold

  else:
    break
```

MoveIt!

- Move with Cartesian path

```
arm_goal = moveit_msgs.msg.ExecuteTrajectoryGoal()
arm_goal.trajectory = plan_cartesian
arm_client.send_goal(arm_goal)
arm_client.wait_for_result()
```

- Stop moveit_commander

```
## When finished shut down moveit_commander.
moveit_commander.roscpp_shutdown()
```

MoveIt!

Code

https://bit.ly/3csSqax

MoveIt!

- Run Robot simulation

```
$ roslaunch my_robot_arm demo.launch
```

- Run python program

```
$ rosrun my_robot_arm simple_arm_command.py
```

MoveIt!

Bring robot arm from simulation to real world!

```
$ roscd my_robot_arm
```

```
$ cd config
```

MoveIt!

Configuration files

- servo_controller_manager.yaml
- servo_controllers.yaml
- arm_joint_trajectory_controller.yaml
- servo_controller_names.yaml

MoveIt!

```
$ gedit servo_controller_manager.yaml
```

```
namespace: dynamixel_controller_manager
serial_ports:
    dxl_tty1:
        port_name: "/dev/ttyUSB0"
        baud_rate: 1000000
        min_motor_id: 1
        max_motor_id: 5
        update_rate: 10
```

MoveIt!

Explain the code

```
namespace: dynamixel_controller_manager
```

- Define your namespace (Show in example as dynamixel_controller_manager).

```
serial_ports:

    dxl_tty1:
```

- Define your serial ports name (Show in example as dxl_tty1).

MoveIt!

Explain the code

```
port_name: "/dev/ttyUSB0"
```

- Define port name to your device path.
- You can find this path at /dev (Show in example as /dev/ttyUSB0).

```
baud_rate: 1000000
```

- Define your servo baud rate (Show in example as 1000000).

MoveIt!

Explain the code

```
min_motor_id: 1
```

- Define your first motor ID (Show in example as 1).

**** Motor ID must be in order ****

```
max_motor_id: 5
```

- Define your last motor ID (Show in example as 5).

**** Motor ID must be in order ****

MoveIt!

Explain the code

```
update_rate: 10
```

- Set your motor update rate (Show in example as 10).

MoveIt!

Configuration files

- servo_controller_manager.yaml
- servo_controllers.yaml
- arm_joint_trajectory_controller.yaml
- servo_controller_names.yaml

MoveIt!

```
$ gedit servo_controllers.yaml
```

```
shoulder_revolute_servo_controller:
  controller:
    package: dynamixel_controllers
    module: joint_position_controller
    type: JointPositionController
  joint_name: crane_plus_shoulder_revolute_joint
  joint_speed: 1.17
  motor:
    id: 1
    init: 512
    min: 0
    max: 1023
```

MoveIt!

Explain the code

```
shoulder_revolute_servo_controller:
```

- Define your joint name (Show in example as shoulder_revolute_servo_controller).

```
controller:
    package: dynamixel_controllers
    module: joint_position_controller
    type: JointPositionController
```

- Define your controller package if use dynamixel servo use “dynamixel_controllers”.
- Define your controller module if use dynamixel servo use “joint_position_controller”.
- Define your controller type if use dynamixel servo use “JointPositionController”.

MoveIt!

Explain the code

```
joint_name: crane_plus_shoulder_revolute_joint
```

- Define your joint name (Show in example as crane_plus_shoulder_revolute_joint).

****** Must define joint name same as name in urdf ******

```
joint_speed: 1.17
```

- Define your joint speed (Show in example as 1.17).

MoveIt!

Explain the code

```
shoulder_revolute_servo_controller:
```

- Define your joint name (Show in example as shoulder_revolute_servo_controller).

```
controller:
    package: dynamixel_controllers
    module: joint_position_controller
    type: JointPositionController
```

- Define your controller package if use dynamixel servo use “dynamixel_controllers”.
- Define your controller module if use dynamixel servo use “joint_position_controller”.
- Define your controller type if use dynamixel servo use “JointPositionController”.

MoveIt!

Explain the code

```
motor:
  id: 1
  init: 512
  min: 0
  max: 1023
```

- Define your motor ID (Show in example as 1).

  ** This id will in range of ID in previous file **
- Define servo initialize position (Show in example as 512).
- Define servo minimum position (Show in example as 0).
- Define servo maximum position (Show in example as 1023).

MoveIt!

Configuration files

- servo_controller_manager.yaml
- servo_controllers.yaml
- arm_joint_trajectory_controller.yaml
- servo_controller_names.yaml

MoveIt!

```
$ gedit arm_joint_trajectory_controller.yaml
```

```
crane_plus:
  controller:
    package: dynamixel_controllers
    module: joint_trajectory_action_controller
    type: JointTrajectoryActionController
  joint_trajectory_action_node:
    min_velocity: 0.1
    constraints:
      goal_time: 0.25
```

MoveIt!

Explain the code

```
crane_plus:
```

- Define your joint trajectory controller name (Show in example as crane_plus).

```
controller:
    package: dynamixel_controllers
    module: joint_trajectory_action_controller
    type: JointTrajectoryActionController
```

- Define your controller package.
  If use dynamixel servo use “dynamixel_controllers”.
- Define your controller module.
  If use dynamixel servo use “joint_trajectory_action_controller”.
- Define your controller type.
  If use dynamixel servo use “JointTrajectoryActionController”.

MoveIt!

Explain the code

```
joint_trajectory_action_node:
    min_velocity: 0.1
    constraints:
      goal_time: 0.25
```

- Define your minimum servo speed (Show in example as 0.1).
- Define your servo goal time constraints (Show in example as 0.25).

MoveIt!

Configuration files

- servo_controller_manager.yaml
- servo_controllers.yaml
- arm_joint_trajectory_controller.yaml
- servo_controller_names.yaml

MoveIt!

```
$ gedit servo_controller_names.yaml
```

```
servo_controller_names:
  - shoulder_revolute_servo_controller
  - shoulder_flex_servo_controller
  - elbow_servo_controller
  - wrist_servo_controller
  - finger_servo_controller
```

- Define your controller name.

  *** Same as controller name in servo_controller.yaml ***

MoveIt!

Python file

- crane_plus_joint_state_publisher.py

MoveIt!

Create node file!

```
$ roscd my_robot_arm
```

```
$ mkdir src && cd src
```

MoveIt!

```
$ gedit crane_plus_joint_state_publisher.py
```

```
#!/usr/bin/env python

import sys

import rospy

from sensor_msgs.msg import JointState
from dynamixel_msgs.msg import JointState as DynamixelJointState
```

MoveIt!

```
def joint_state_cb(msg, js_publisher):
    js = JointState()
    js.header.stamp = msg.header.stamp
    js.name = [msg.name]
    js.position = [msg.current_pos]
    js.velocity = [msg.velocity]
    js.effort = [msg.load]
    js_publisher.publish(js)
```

- This function is a callback function from subscriber. It convert massage(msg) from DynamixelJointState to JointState and publish to js_publisher

MoveIt!

```
def subscribe_to_servo_controller(joint, namespace, js_publisher):
        topic_name = '/'.join([namespace, joint, 'state'])
        return rospy.Subscriber(topic_name,
                DynamixelJointState, callback=joint_state_cb,
                callback_args=js_publisher)
```

- This function is a subscriber function that subscribe from topic_name and call joint_state_cb as callback function for convert massage and publish to js_publisher

MoveIt!

```
def publish_virtual_joint_cb_maker(joint_name, pub):

    def publisher(event):
        js = JointState()
        js.header.stamp = rospy.Time.now()
        js.name = [joint_name]
        js.position = [0]
        js.velocity = [0]
        js.effort = [0]
        pub.publish(js)
    return publisher
```

- This function is for send zero jointstate to joint_name by topic pub

MoveIt!

```
def main():
        rospy.init_node('crane_plus_joint_state_publisher')
        try:
                servo_controllers = rospy.get_param('~servo_controller_names')
        except KeyError:
                rospy.logerr('Configuration error: Could not find list of servo'
                        'controllers on the parameter server')
                return 1
        namespace = rospy.get_param('namespace', '')
        publish_virtual_joint = rospy.get_param('~publish_virtual_joint', True)
```

MoveIt!

```
if not servo_controllers:
        rospy.logerr('No servo controller names specified; nothing to publish')
        return 1
js_pub = rospy.Publisher('joint_states', JointState, queue_size=1)
servo_subs = []
for s in servo_controllers:
        servo_subs.append(subscribe_to_servo_controller(s, namespace, js_pub))
```

MoveIt!

```
        if publish_virtual_joint:
                rospy.Timer(rospy.Duration(1),
                        publish_virtual_joint_cb_maker(publish_virtual_joint, js_pub))
        rospy.spin()
        return 0


if __name__ == '__main__':
   sys.exit(main())
```

MoveIt!

Launch file

- start_arm.launch

MoveIt!

Making launch file!

```
$ roscd my_robot_arm/launch && gedit start_arm.launch
```

```
<launch>
        <param name="robot_description"
               command="$(find xacro)/xacro --inorder
               '$(find crane_plus_description)/urdf/crane_plus.urdf.xacro'"
          />
        <node name="dynamixel_manager"
               pkg="dynamixel_controllers"
               type="controller_manager.py"
               required="true"
               output="screen" >
        <rosparam  file="$(find my_robot_arm)/config/servo_controller_manager.yaml“
               command="load” />
        </node>
```

MoveIt!

Making launch file!

```
<rosparam file="$(find my_robot_arm)/config/servo_controller.yaml"
          command="load"/>
<node name="servo_controller_spawner"
          pkg="dynamixel_controllers"
          type="controller_spawner.py"
          args="--manager=dynamixel_controller_manager --port dxl_tty1
                    shoulder_revolute_servo_controller
                    shoulder_flex_servo_controller
                    elbow_servo_controller
                    wrist_servo_controller
                    finger_servo_controller“
          output="screen"/>
```

MoveIt!

Making launch file!

```
<rosparam file="$(find my_robot_arm)/config/arm_joint_trajectory_controller.yaml"
          command="load"/>
<node name="joint_trajectory_controller_spawner"
          pkg="dynamixel_controllers"
          type="controller_spawner.py"
          args="--manager=dynamixel_controller_manager --type=meta
                    crane_plus
                    shoulder_revolute_servo_controller
                    shoulder_flex_servo_controller
                    elbow_servo_controller
                    wrist_servo_controller"
          output="screen" />
```

MoveIt!

Making launch file!

```
        <node name="crane_plus_joint_state_publisher"
                pkg="my_robot_arm"
                type="my_state_publisher.py">
                <rosparam
                        file="$(find my_robot_arm)/config/servo_controller_names.yaml"
                        command="load" />
                <param name="publish_virtual_joint" value="crane_plus_gripper_joint"/>
        </node>
        <node name="robot_state_publisher"
                pkg="robot_state_publisher"
                type="robot_state_publisher" />
</launch>
```

MoveIt!

Copy file!

```
$ cd
```

```
$ git clone https://github.com/RobotCitizens/ros-workshop.git && git checkout Day4
```

```
$ cp ~/ros-workshop/Day4/Code-example/config/*   ~/ros_workshop_ws/src/my_robot_arm/config
```

```
$ cp ~/ros-workshop/Day4/Code-example/src/*   ~/ros_workshop_ws/src/my_robot_arm/src
```

```
$ cp ~/ros-workshop/Day4/Code-example/launch/*   ~/ros_workshop_ws/src/my_robot_arm/launch
```

MoveIt!

Modify code!

```
$ roscd my_robot_arm/launch
```

```
$ gedit demo.launch
```

```
<arg name="fake_execution" value="true"/>  →  <arg name="fake_execution" value="false"/>
```

```
$ roscd my_robot_arm/config
```

```
$ gedit ros_controllers.yaml
```

```
controller_list:
        - name: arm_controller  →  crane_plus
```

MoveIt!

Run code!

```
$ roslaunch my_robot_arm demo.launch
```

```
$ roslaunch my_robot_arm start_arm_standalone.launch
```

THANK YOU