



KU
KASETSART
UNIVERSITY

BY robotcitizens



Robot Arm

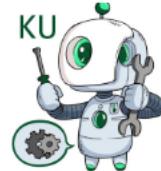
OVERVIEW

Robot Manipulator Concept

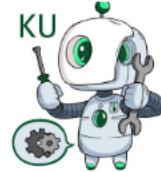
- Manipulation Concept

MoveIt!

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



Robot Manipulator Concept





Robot Manipulator Concept

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

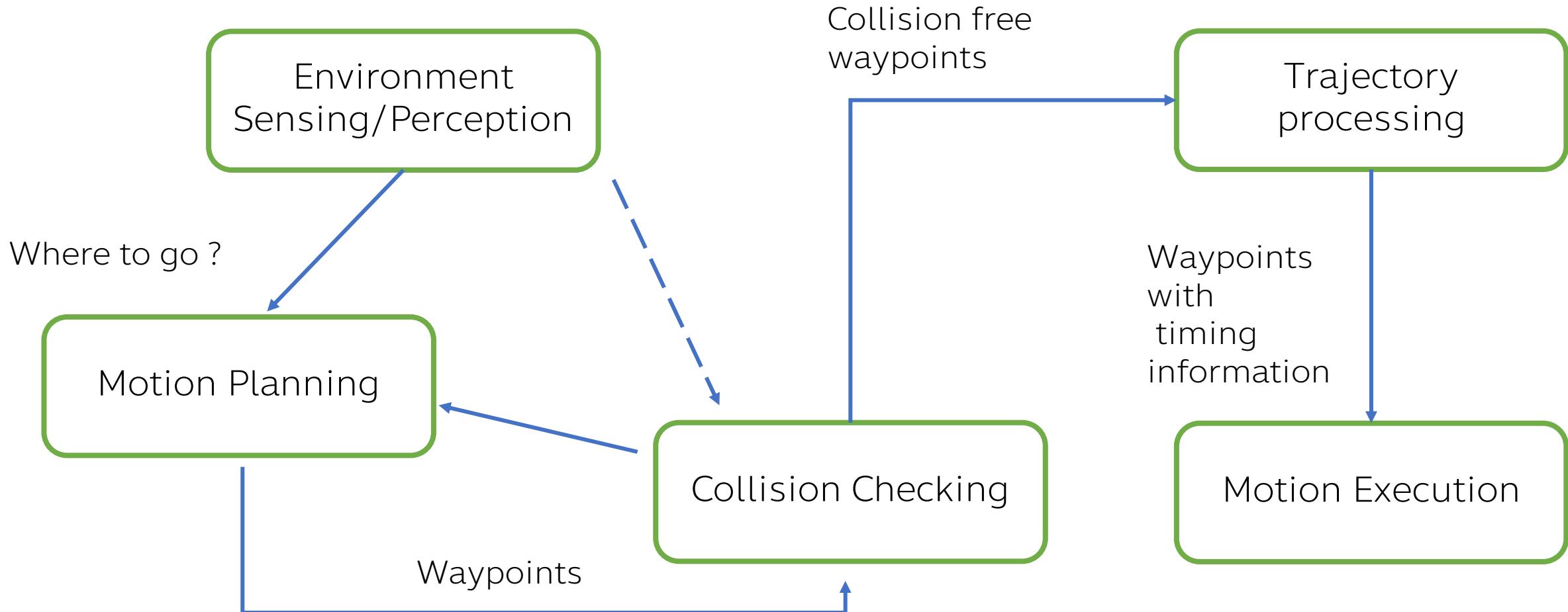
- Pick and place object
- CNC milling, wire cut
- Fasten two parts of car





Robot Manipulator Concept

Manipulation – function modules





Robot Manipulator Concept

- Kinect
- Lidar

Environment
Sensing/Perception

3D sensor, Camera, Force sensor

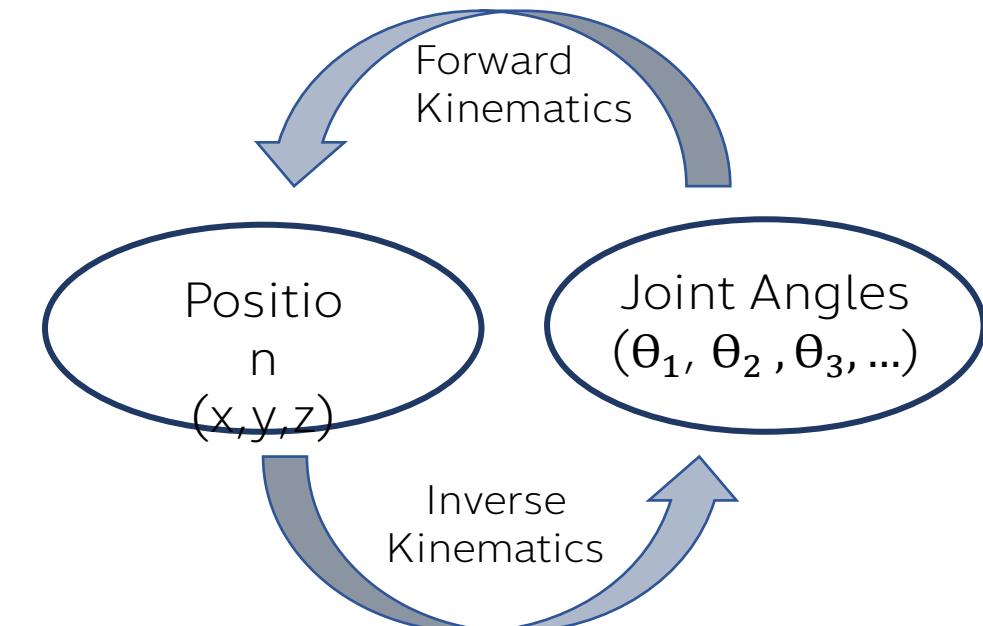
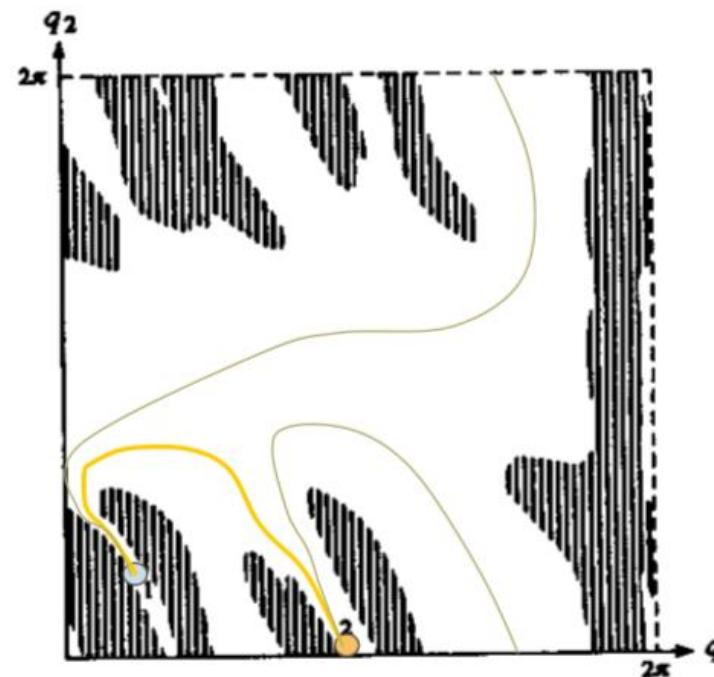
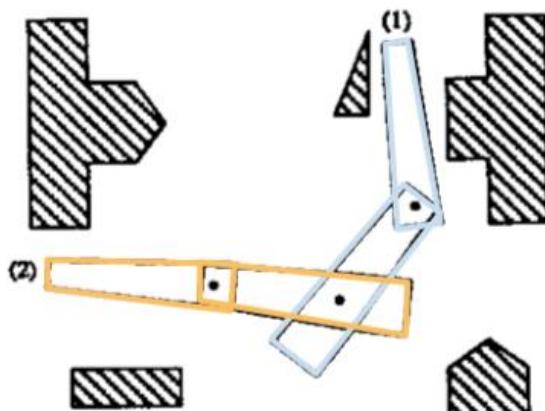


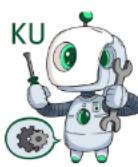


Robot Manipulator Concept

- OMPL library
- CHOMP library

Motion Planning

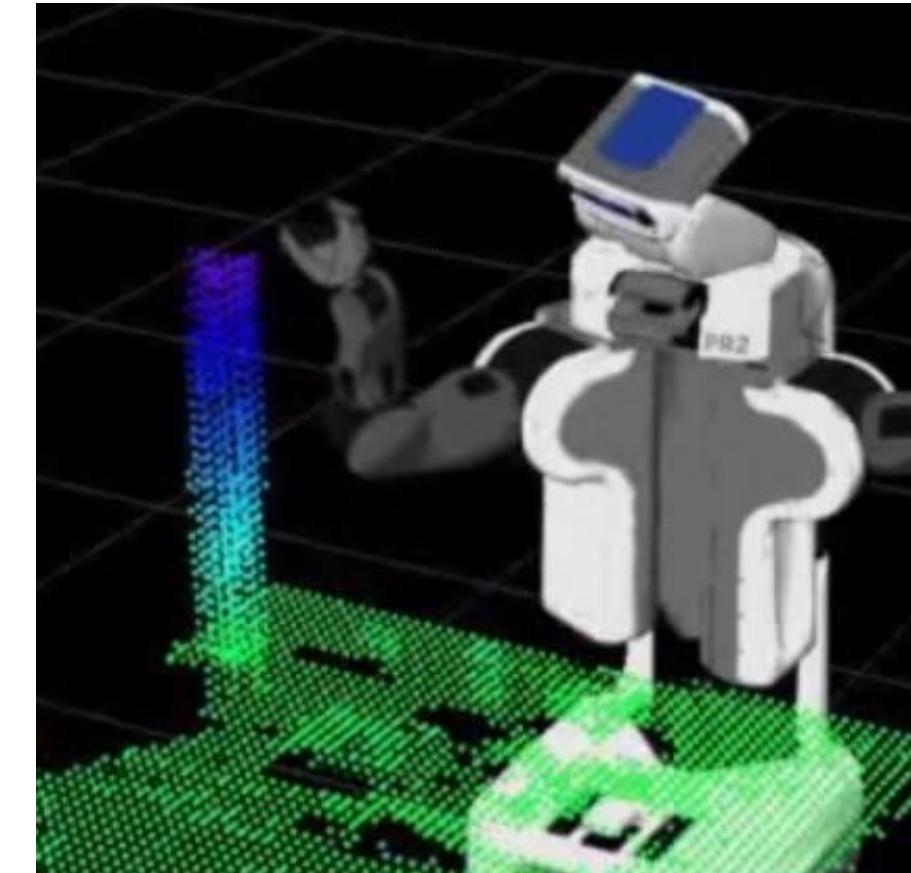
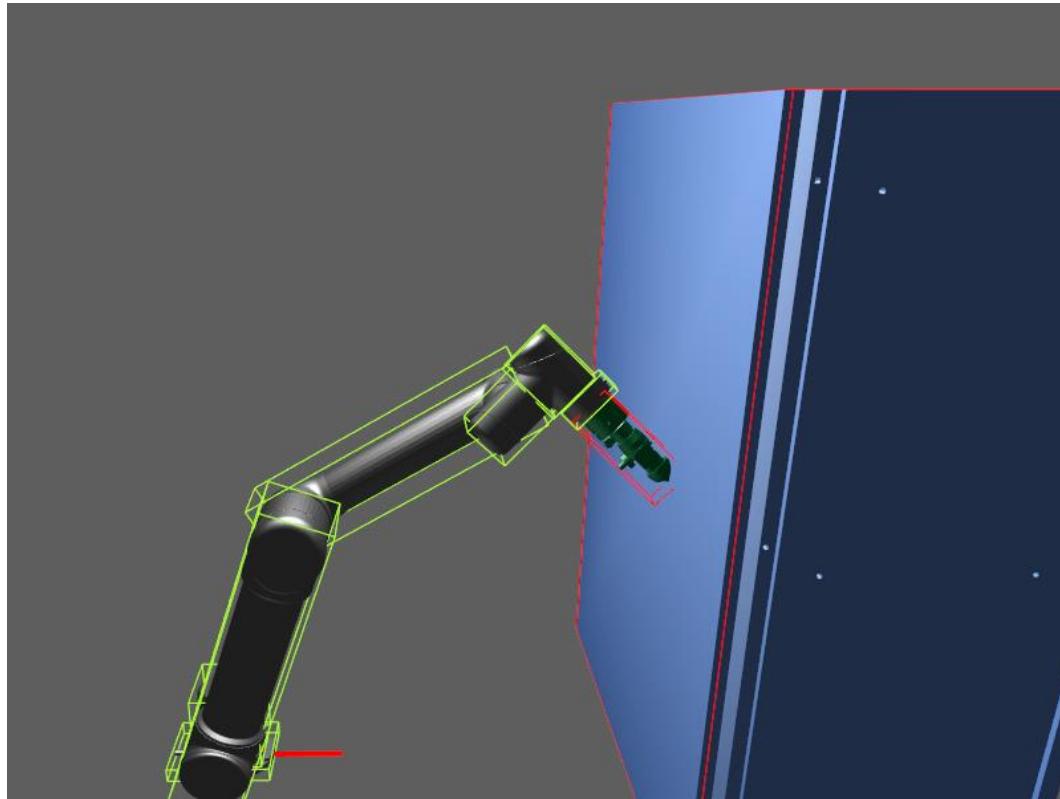




Robot Manipulator Concept

- FCL library
- Bullet library

Collision Checking

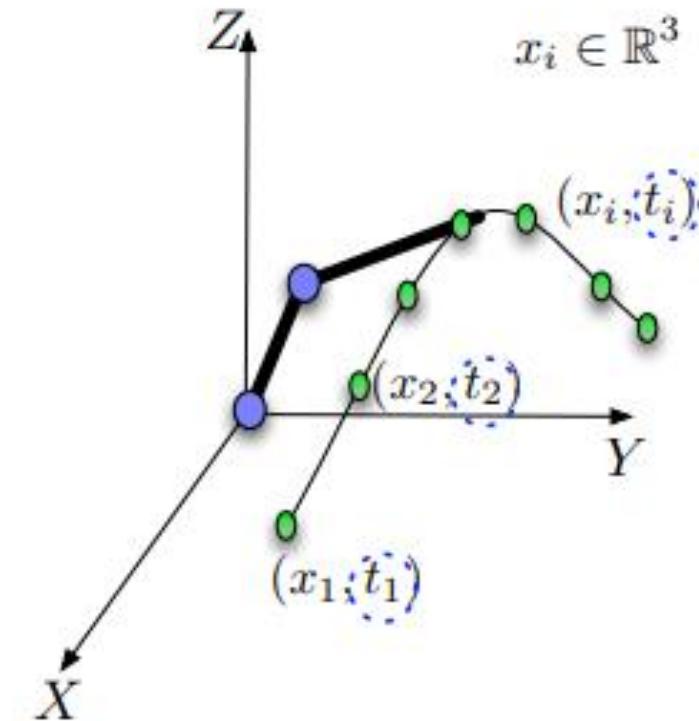
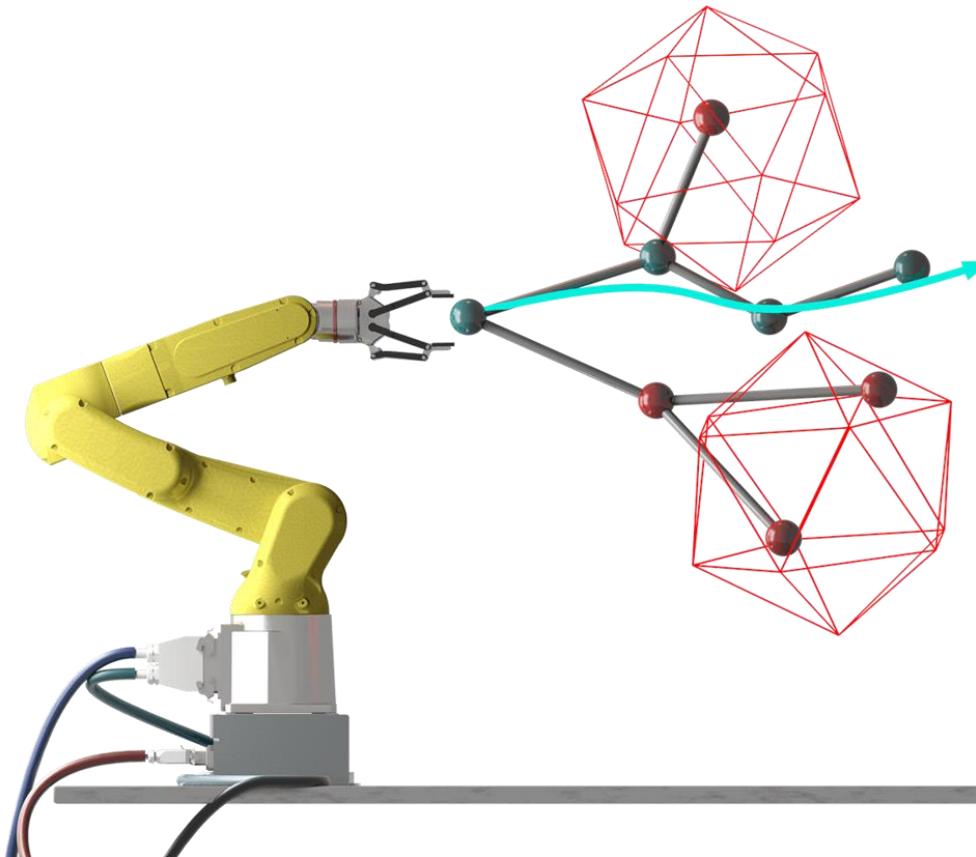


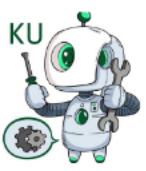


Robot Manipulator Concept

- IPTP library
- TOPP library

Trajectory
processing

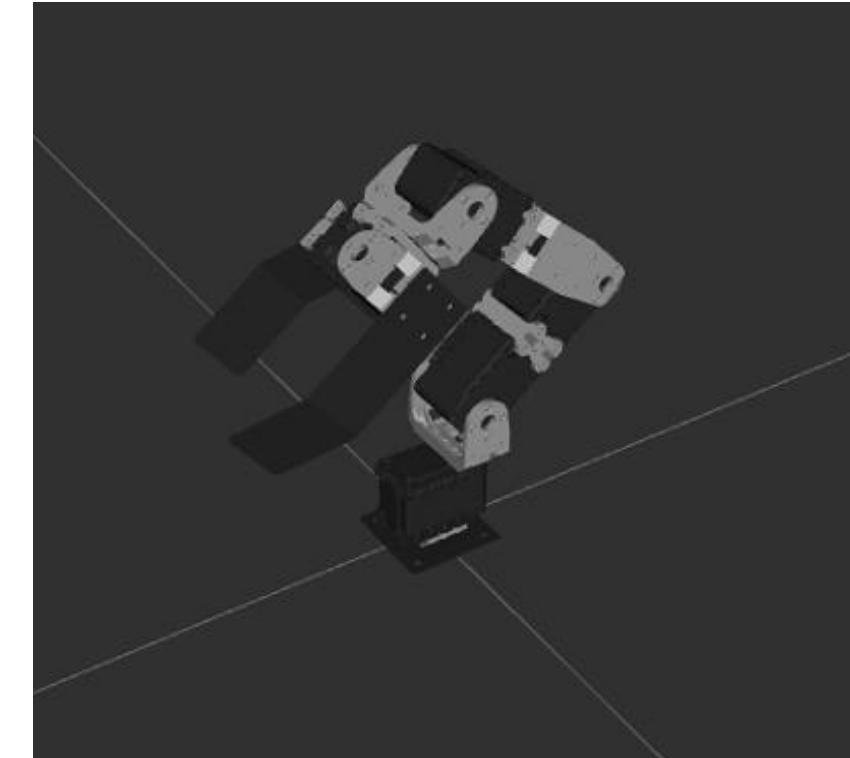




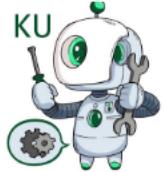
Robot Manipulator Concept

- Real Robot
(Dynamixel Controller)
- Simulation

Motion Execution



MoveIt!

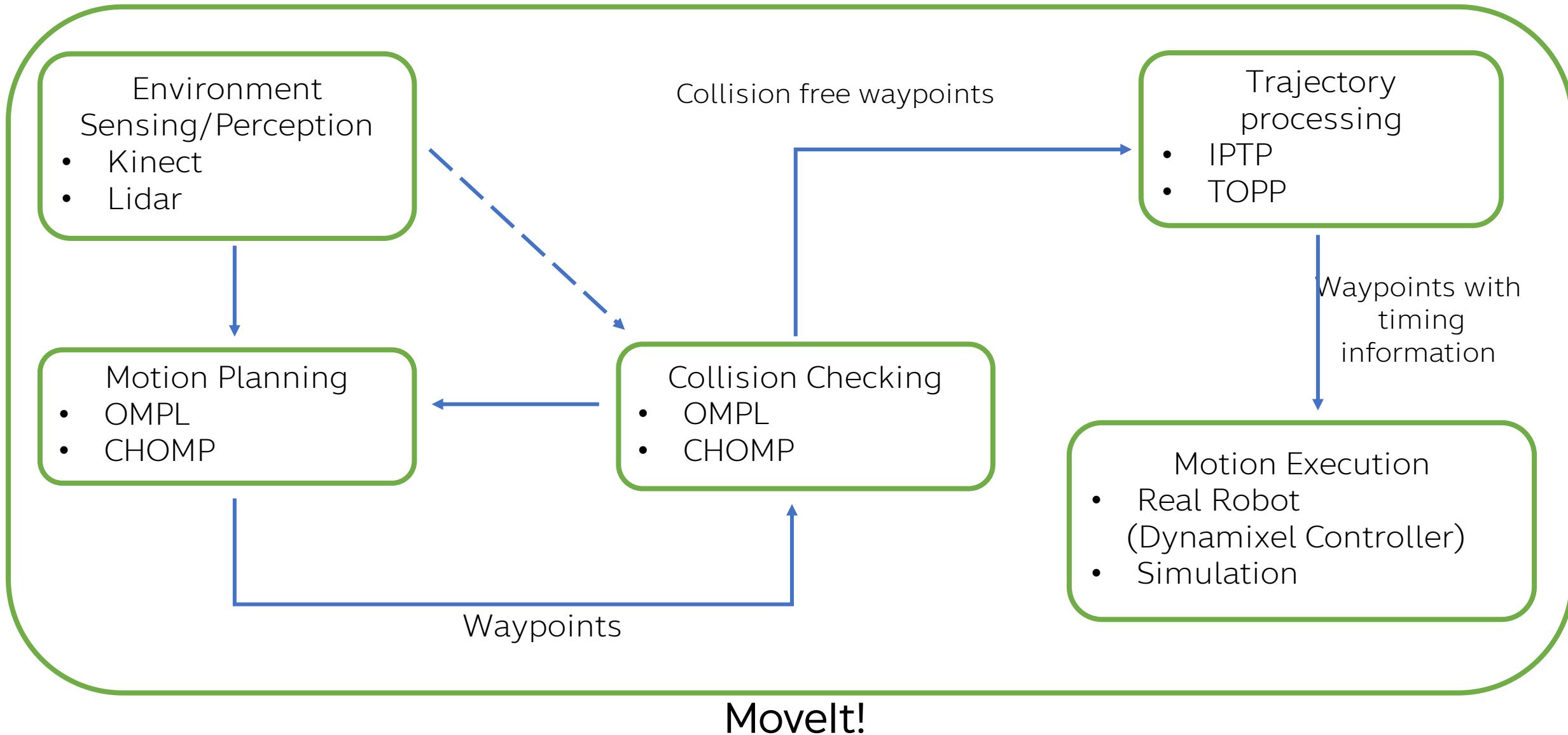


KU

KASETSART
UNIVERSITY



Movel!





Movel!



Install MoveIt! for ROS kinetic

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



Install Dynamixel controller, msg, driver

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

Movelt!

Go to source directory in your workspace

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

Download URDF

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

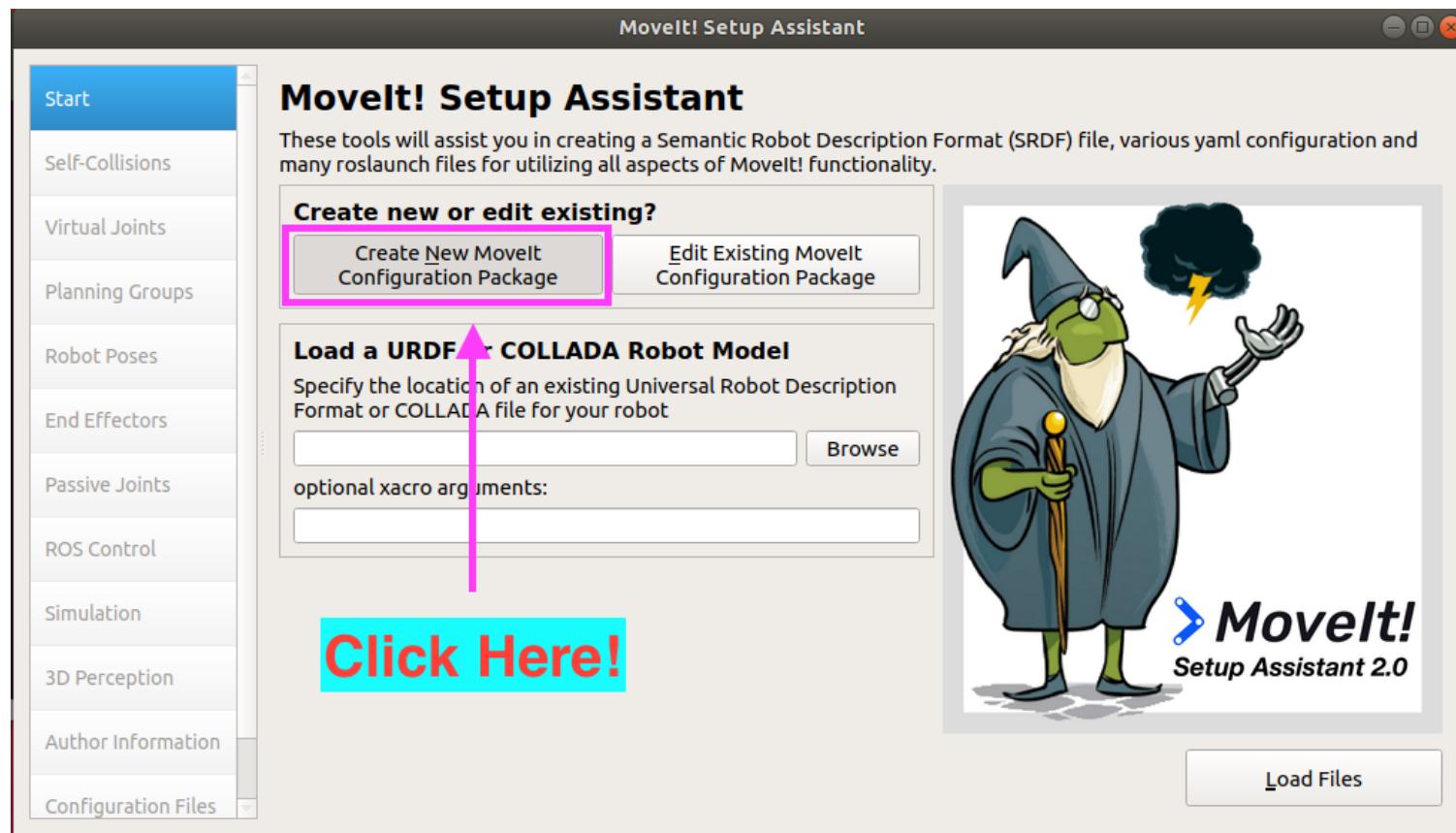
Then go to your workspace and run

```
$ cd ..
```

```
$ catkin_make
```

MoveIt!

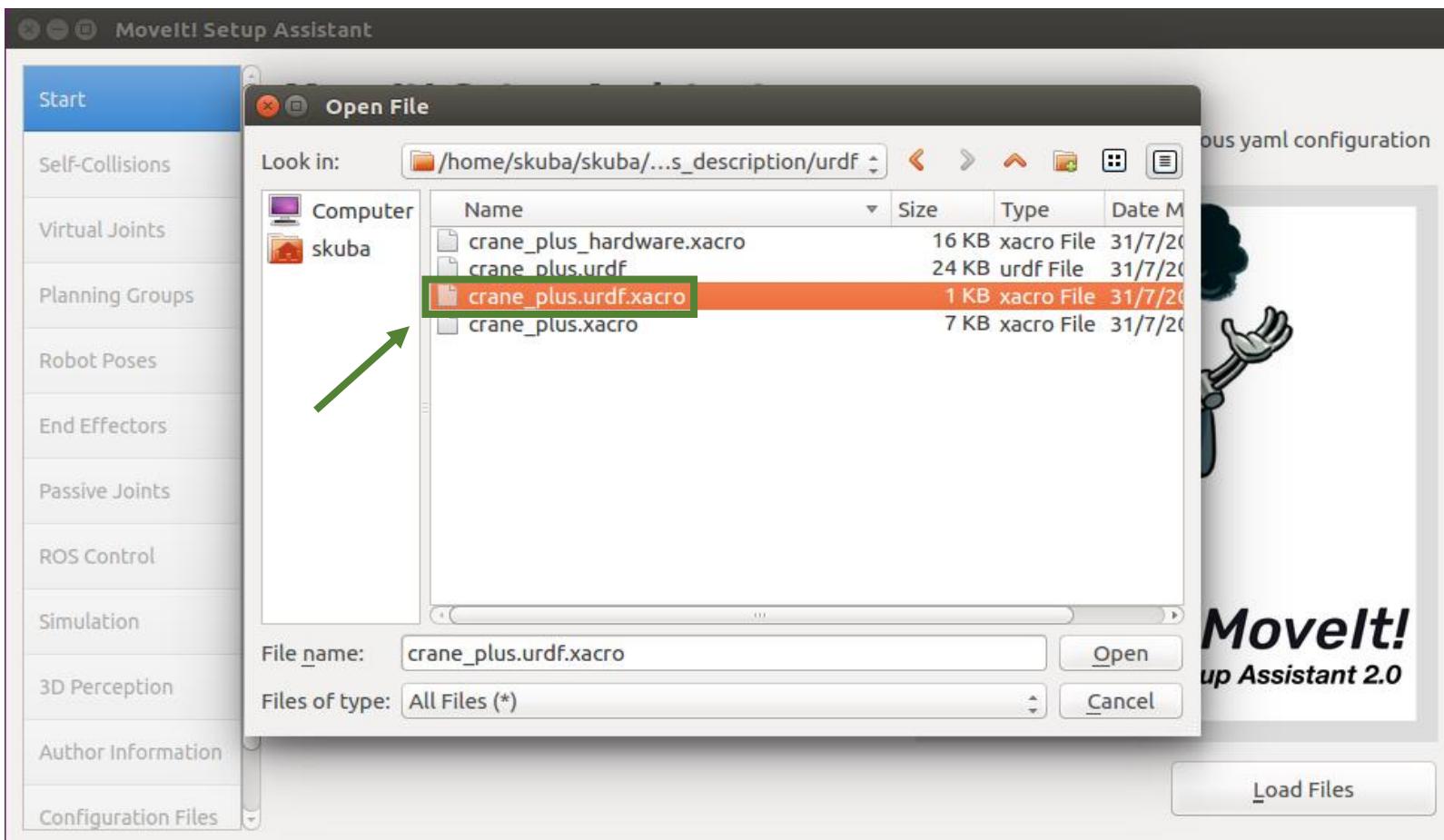
```
$ roslaunch moveit_setup_assistant setup_assistant.launch
```

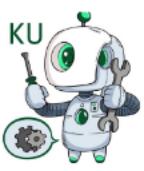


Movel!

Select URDF File

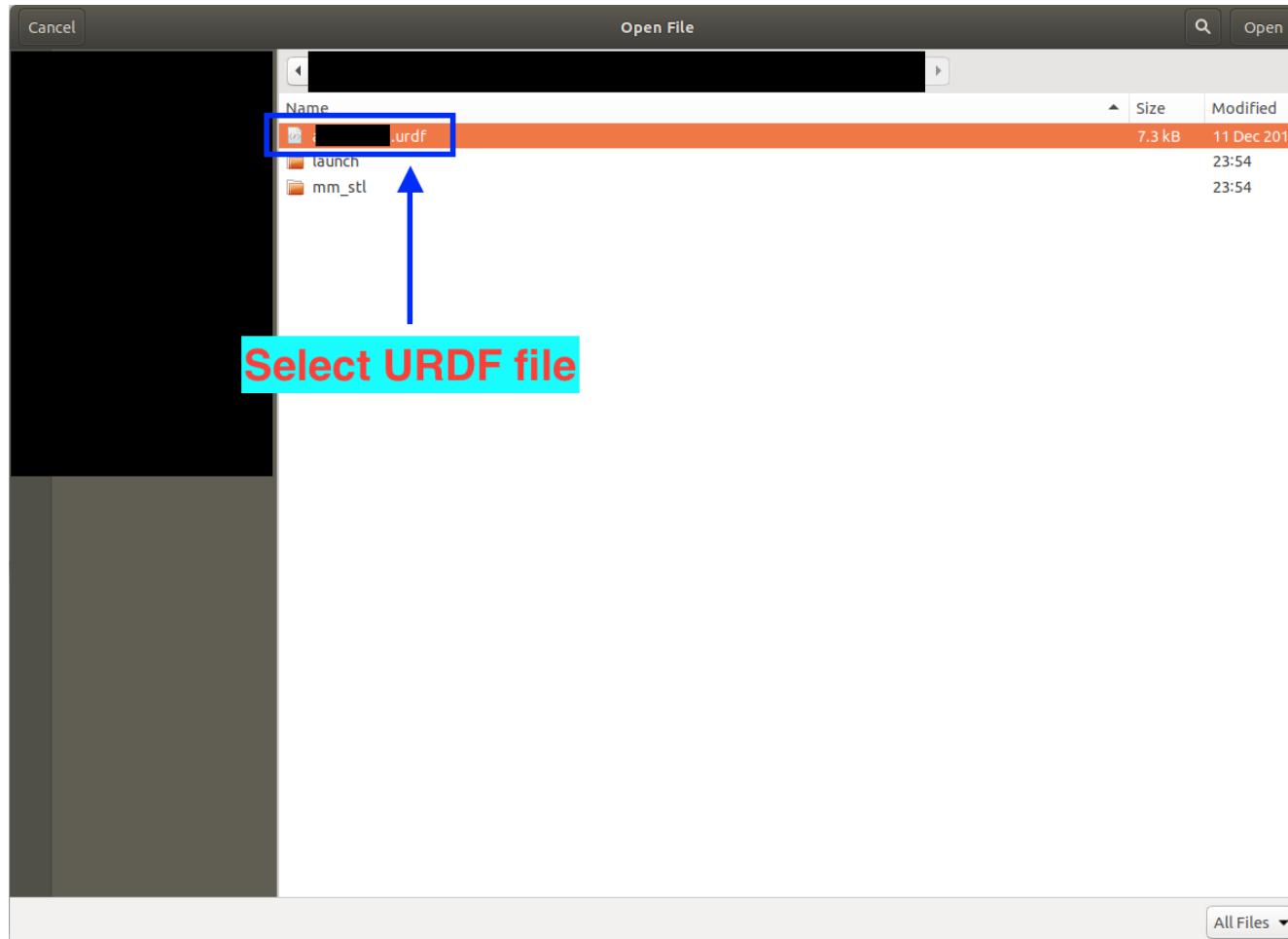
```
~/ros_workshop_ws/src/crane_plus_arm/crane_plus_description/urdf/crane_plus.urdf.xacro
```





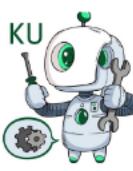
Movel!

For another URDF file

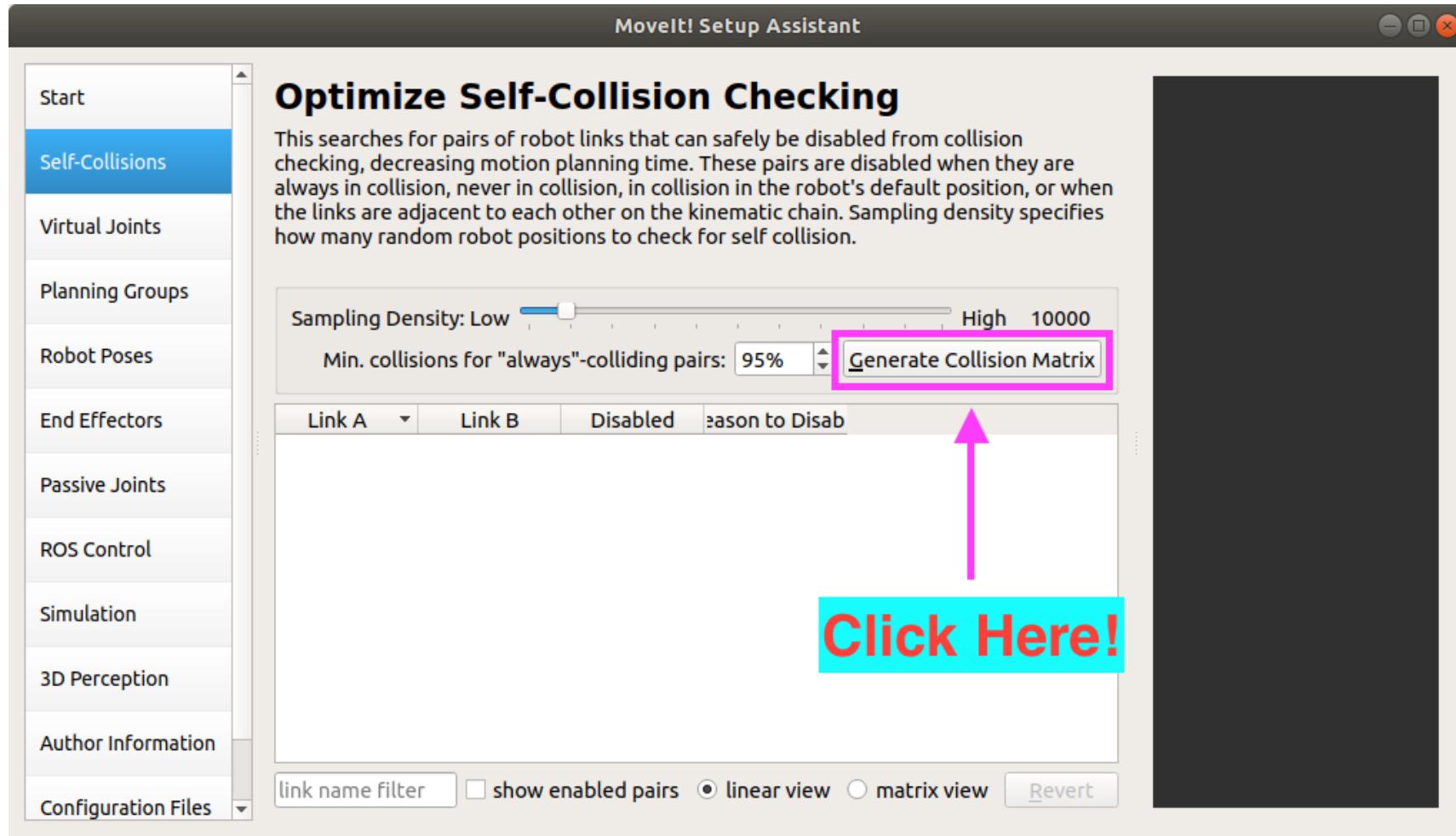


You can export your CAD file
To the URDF File by using
the open source library

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



Movelt!



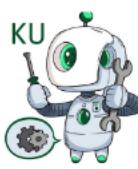
Click Here!



Movelt!

Add Planning Groups





Movelt!

Movelt! Setup Assistant

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:

Kin. Search Resolution:

Kin. Search Timeout (sec):

Kin. Solver Attempts:

OMPL Planning

Group Default Planner:

Next, Add Components To Group:

Recommended:

Advanced Options:



Movelt!

The screenshot shows the MoveIt! Setup Assistant application window titled "Define Planning Groups". The left sidebar lists various setup options: Start, Self-Collisions, Virtual Joints, Planning Groups (which is selected), Robot Poses, End Effectors, Passive Joints, ROS Control, Simulation, 3D Perception, Author Information, and Configuration Files.

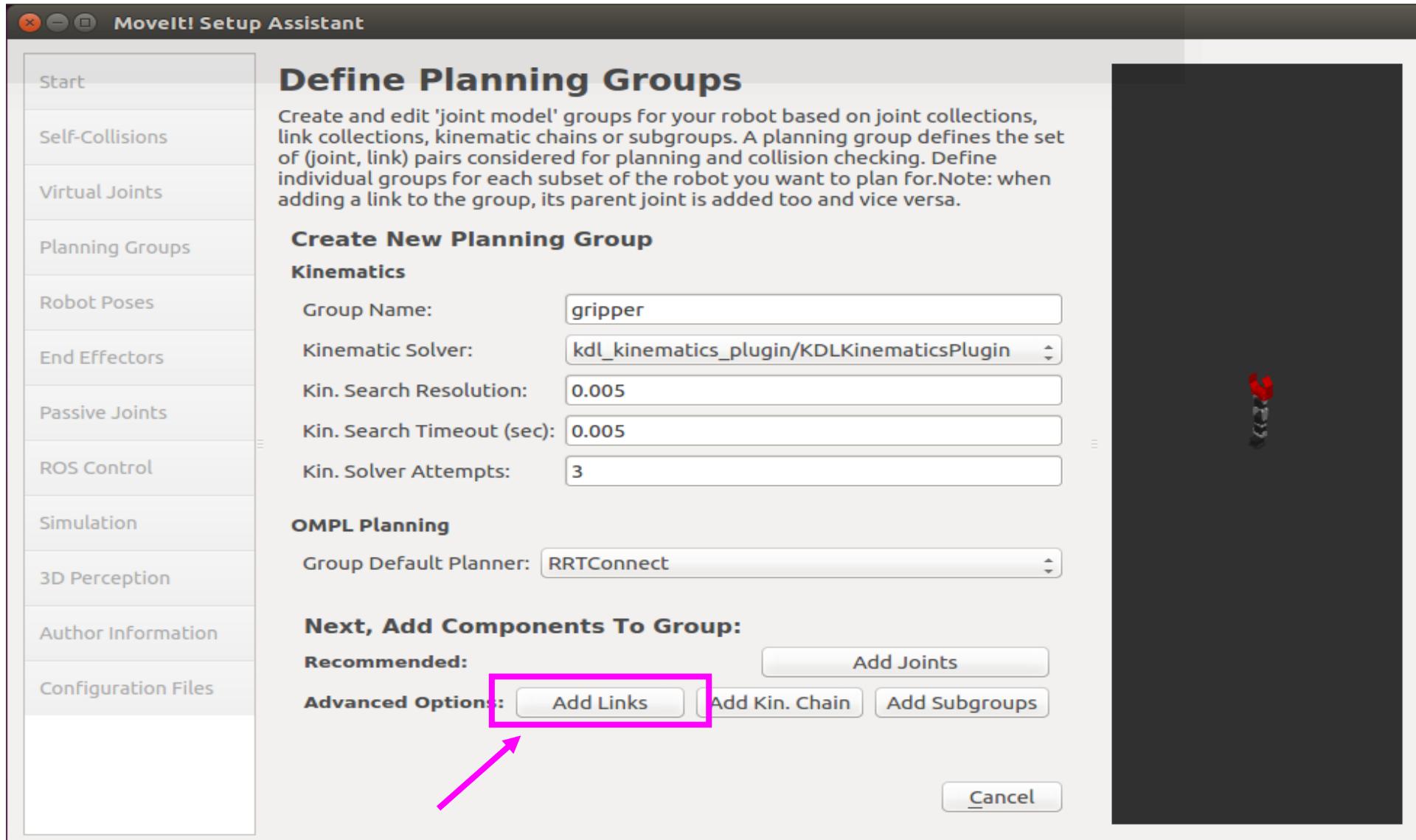
The main content area is titled "Edit 'arm' Kinematic Chain". It displays a hierarchical tree of robot links under the heading "Robot Links". The tree starts with "base_link" and branches down through "crane_plus_mounting_link", "crane_plus_internal_mounting_plate_link", "crane_plus_internal_base_bracket_link", "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", and finally "crane_plus_gripper_link".

At the bottom of the interface, there are two input fields with "Choose Selected" buttons, both of which are highlighted with a red box:

- Base Link: crane_plus_mounting_link
- Tip Link: crane_plus_gripper_link

On the right side of the window, there is a small 3D rendering of a red robotic arm with a gripper at the end.

Movelt!





Movelt!

Movelt! Setup Assistant

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

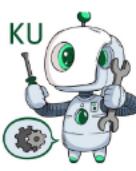
	Link Name
1	base_link
2	crane_plus_mounting_link
3	crane_plus_internal_mounting
4	crane_plus_internal_base_revolute
5	crane_plus_internal_shoulder_revolute
6	crane_plus_shoulder_revolute
7	crane_plus_shoulder_flex_link
8	crane_plus_internal_upper_arm
9	crane_plus_internal_upper_elbow
10	crane_plus_internal_lower_arm
11	crane_plus_elbow_link

Selected Links

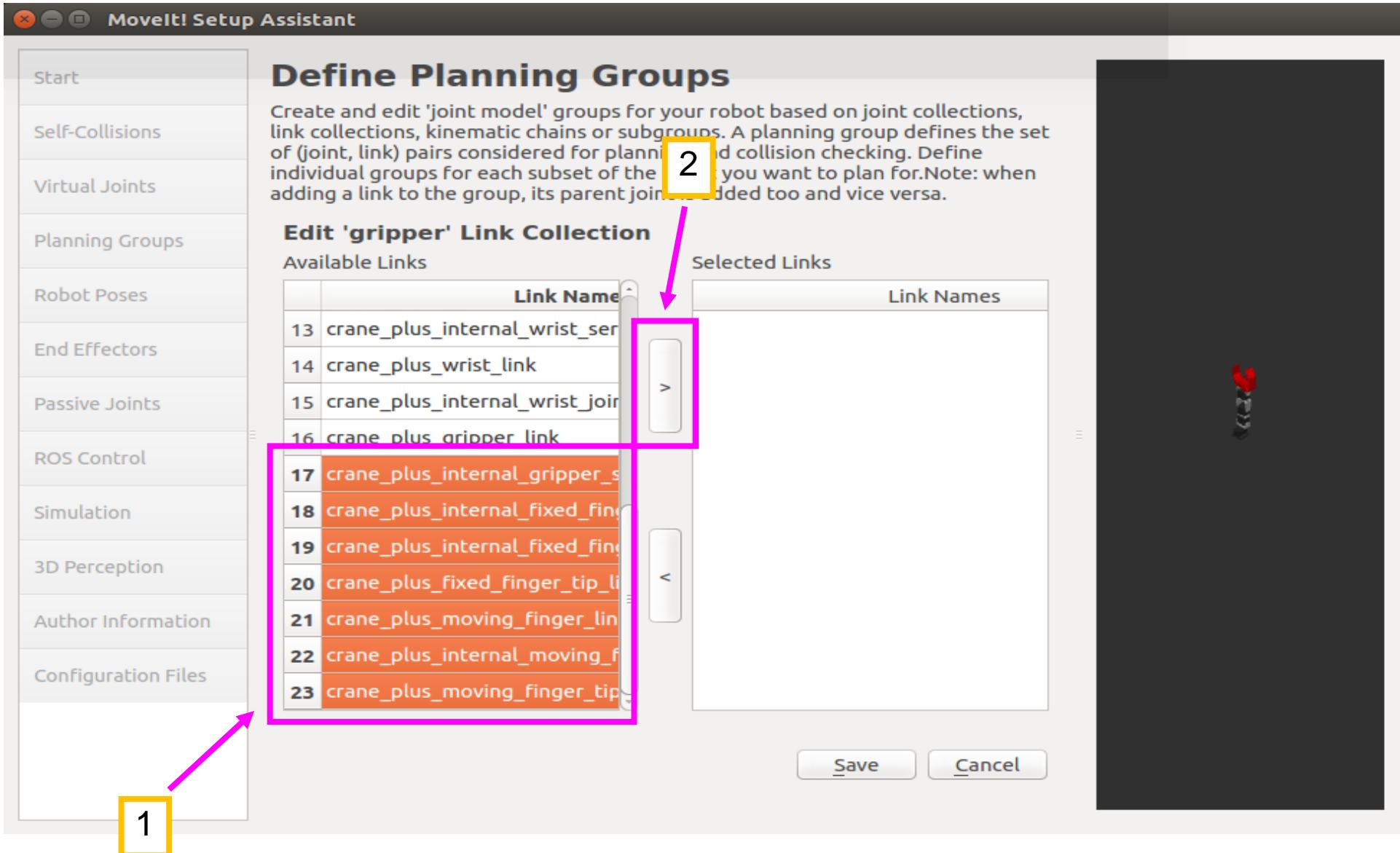
	Link Names
--	------------

> <

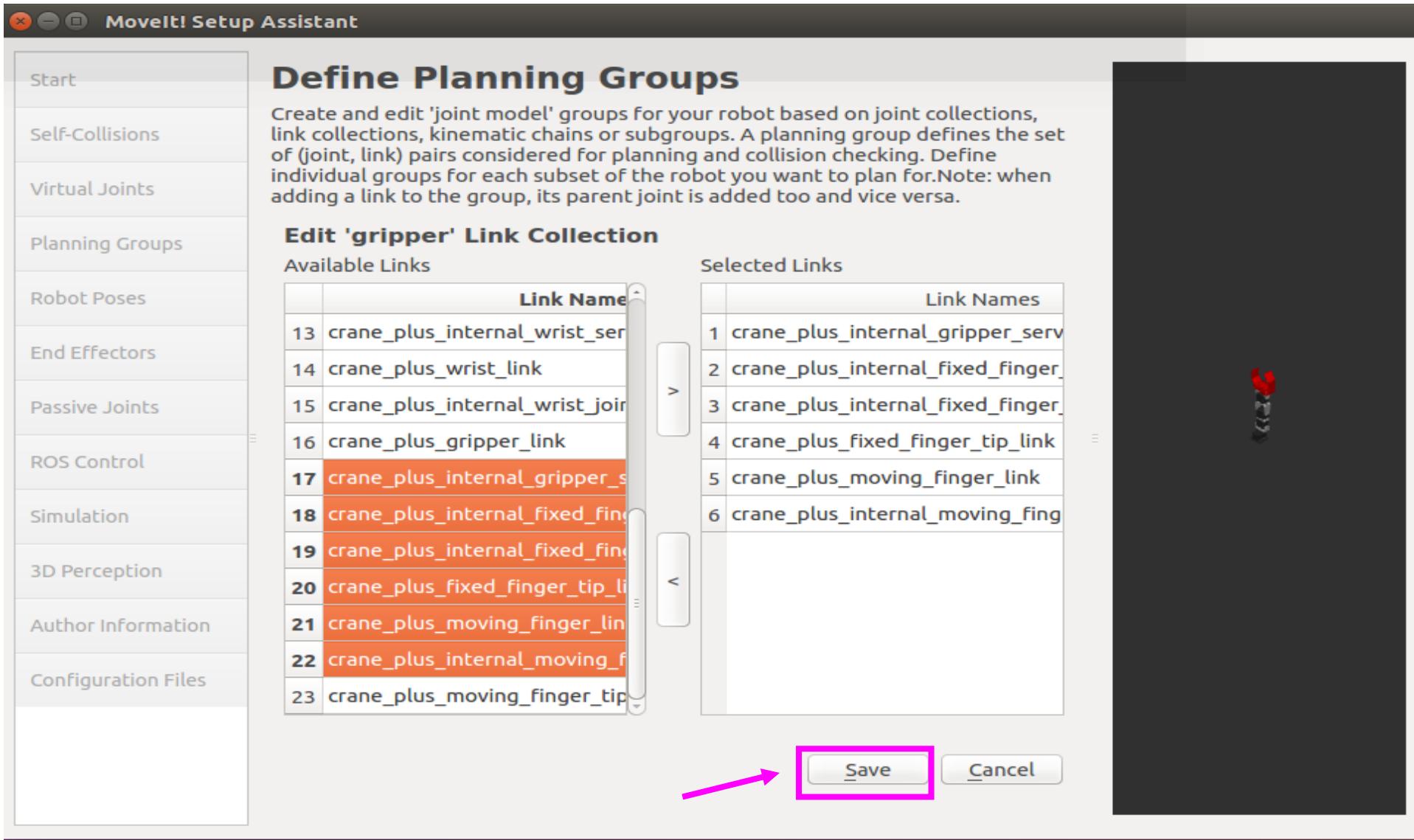
Save **Cancel**



Movelt!

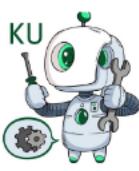


Movelt!



Movelt!





Movelt!

Add Pose

Setup Assistant

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:

Planning Group:

crane_plus_shoulder_revolute_joint

crane_plus_shoulder_flex_joint

crane_plus_elbow_joint

crane_plus_wrist_joint

crane_plus_gripper_joint



Save **Cancel**

Movelt!

Add Pose

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:

resting

Planning Group:

arm

crane_plus_shoulder_revolute_joint

0.0000

crane_plus_shoulder_flex_joint

0.8840

crane_plus_elbow_joint

1.7787

crane_plus_wrist_joint

1.3373

crane_plus_gripper_joint

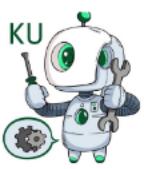
0.0000

	Pose Name	Group Name
1	vertical	arm
2	resting	arm

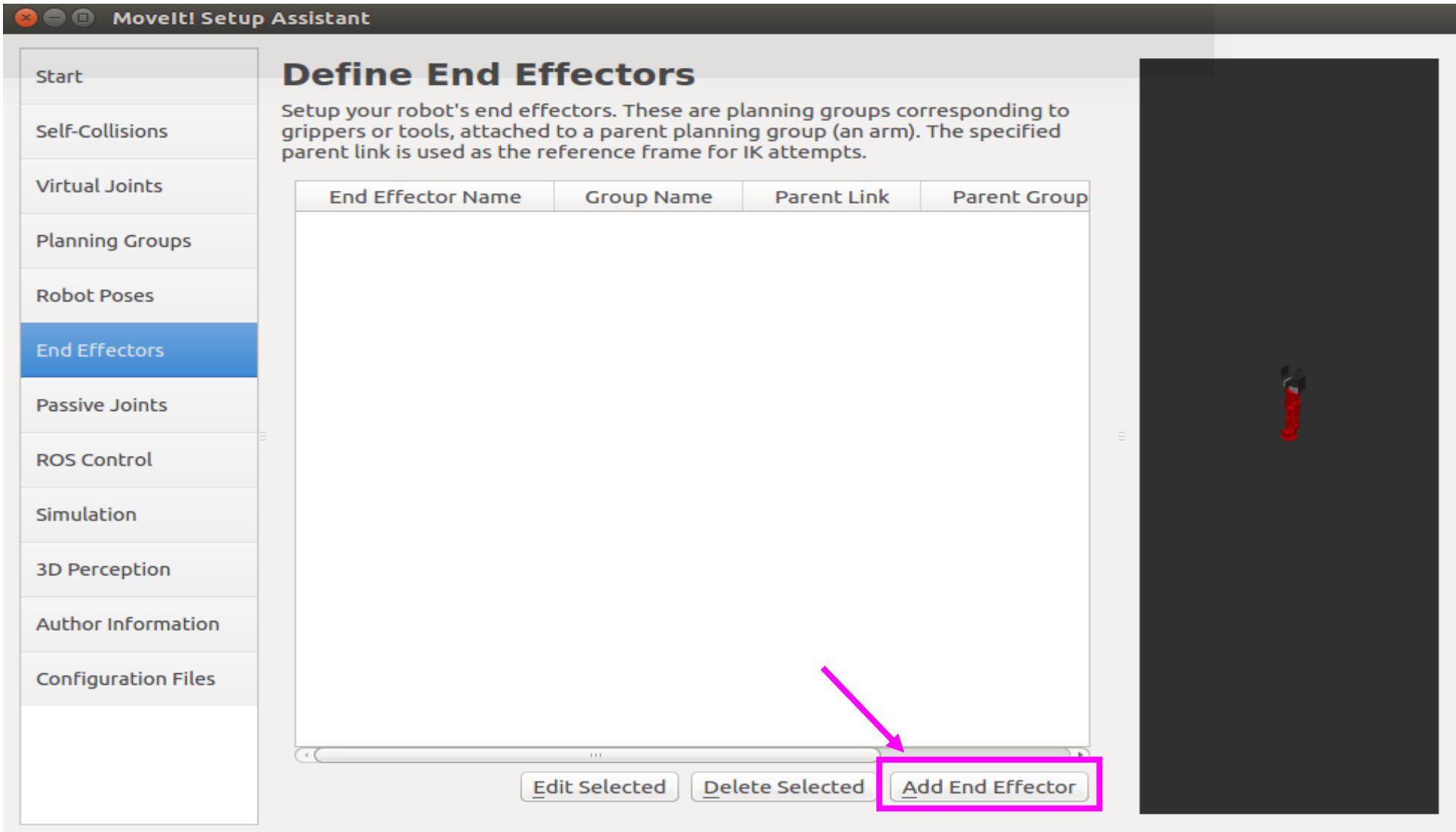
Save

Cancel





Movelt!





Movel!

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.

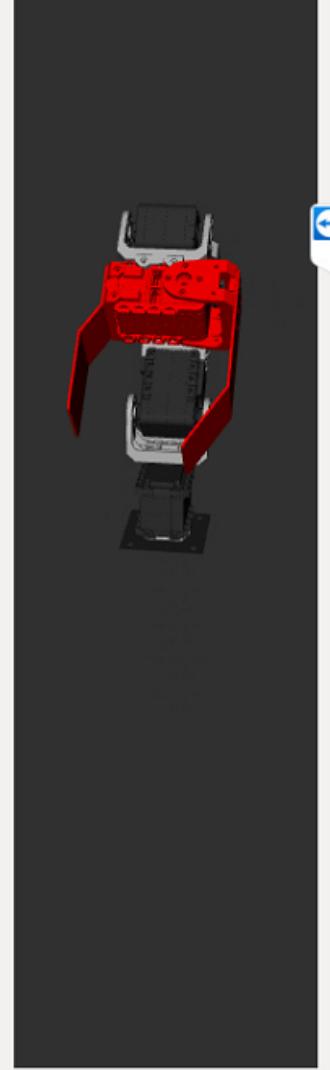
End Effector Name:

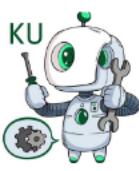
End Effector Group:

Parent Link (usually part of the arm):

Parent Group (optional):

Save **Cancel**





Movel!

Movel! Setup Assistant

Define Passive Joints
Specify the set of passive joints (not actuated). Joint state is not expected to be published for these joints.

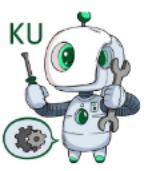
Active Joints

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

> <



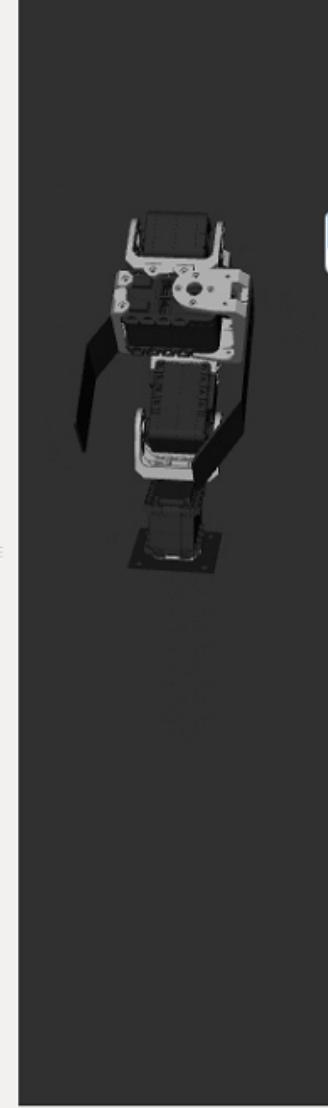
Setup ROS Controllers

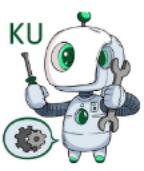
Configure MoveIt! to work with ROS Control to control the robot's physical hardware

[Auto Add FollowJointTrajectory 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](#)





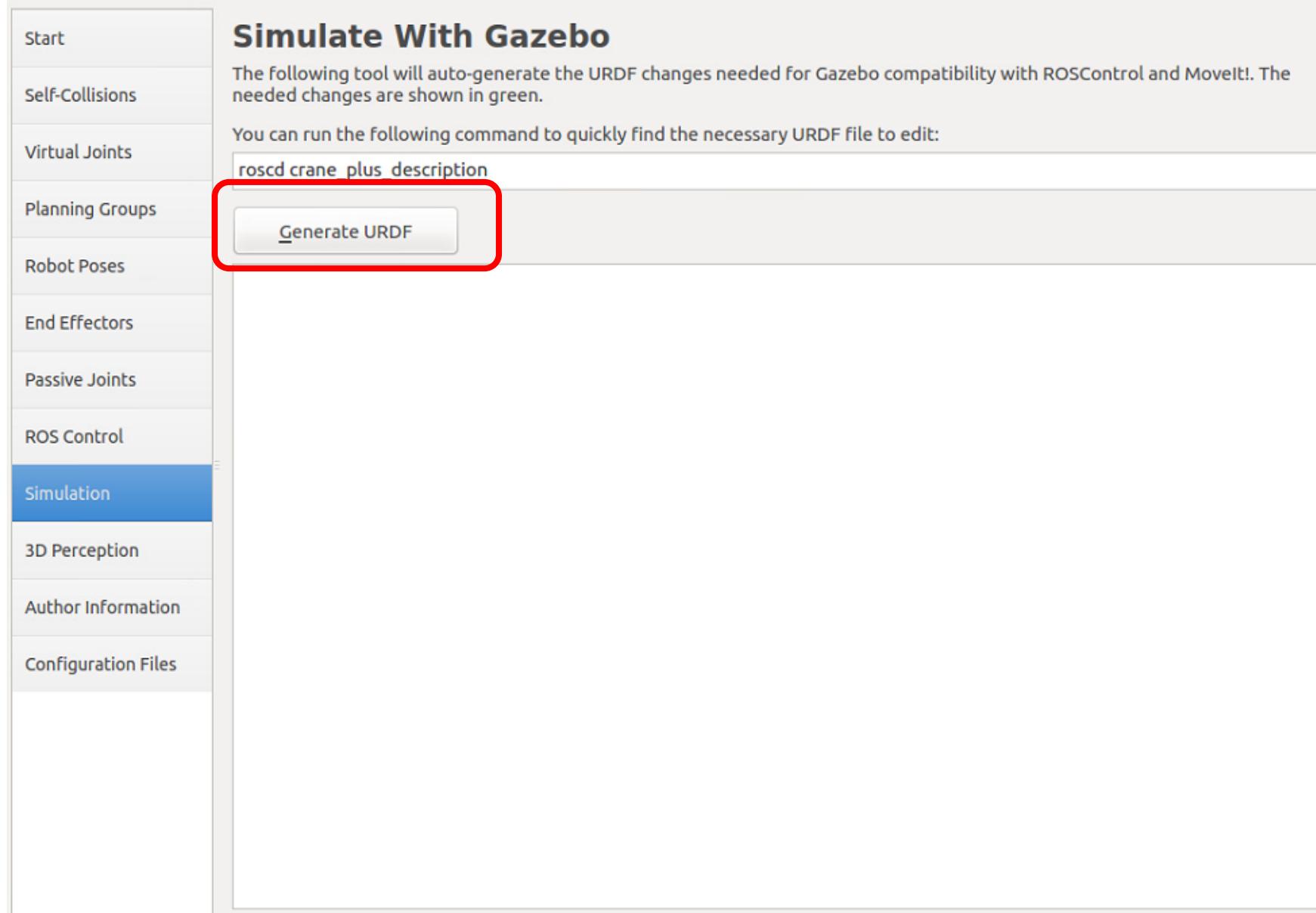
Simulate With Gazebo

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

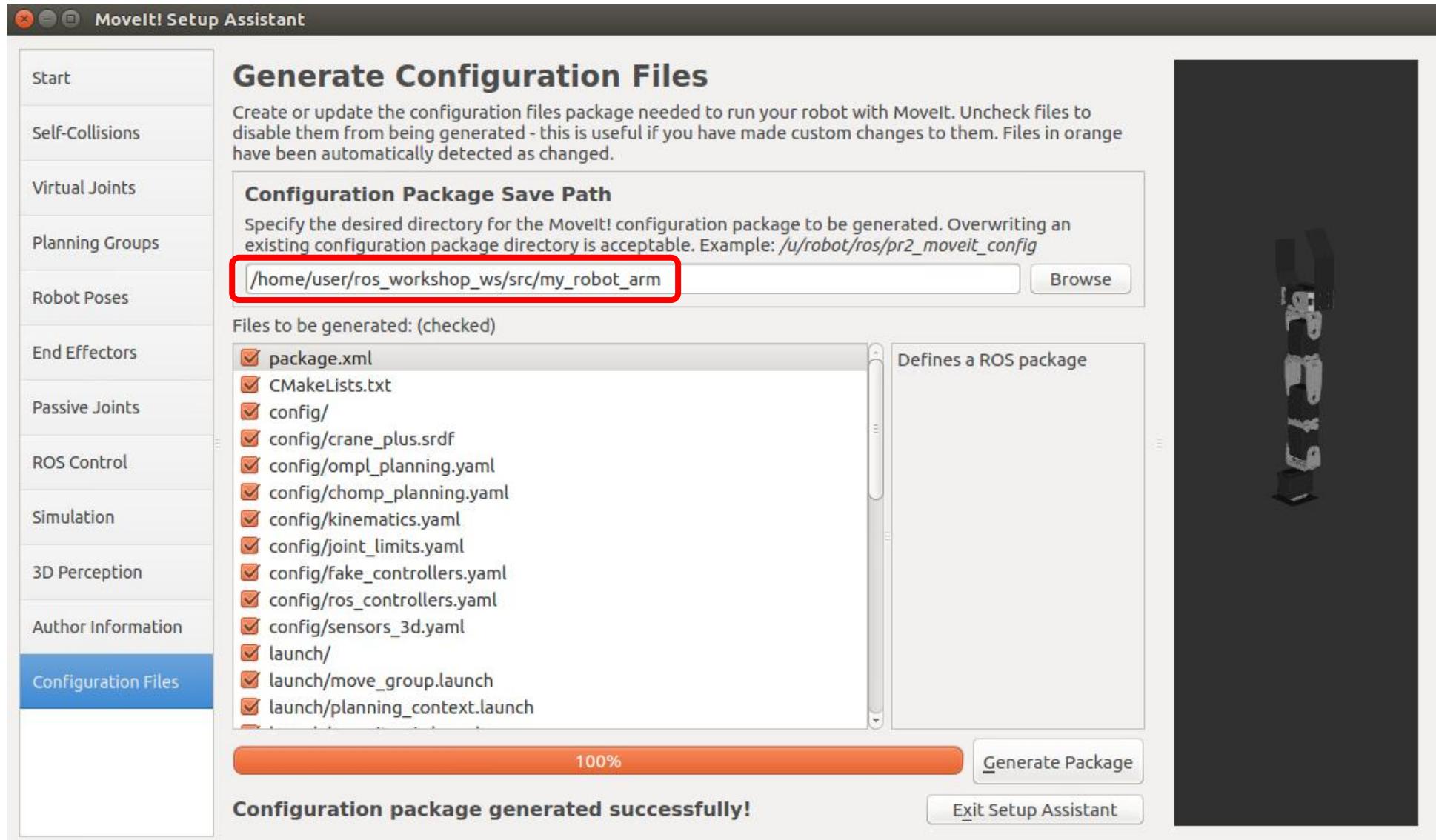
You can run the following command to quickly find the necessary URDF file to edit:

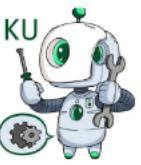
```
roscd crane_plus_description
```

Generate URDF

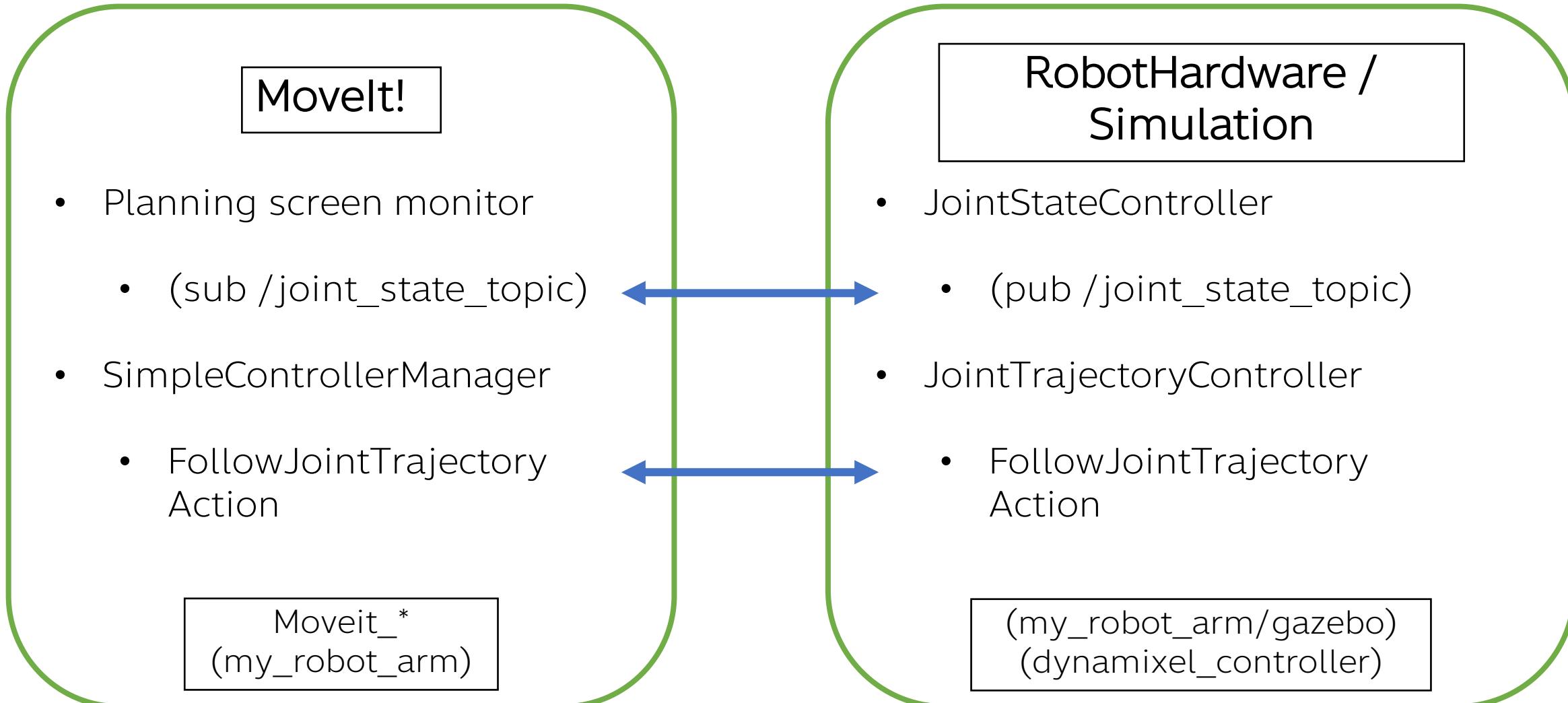


Movelt!





Movelt!



MoveIt!

Running Moveit! Demo

Go to workspace and do catkin_make

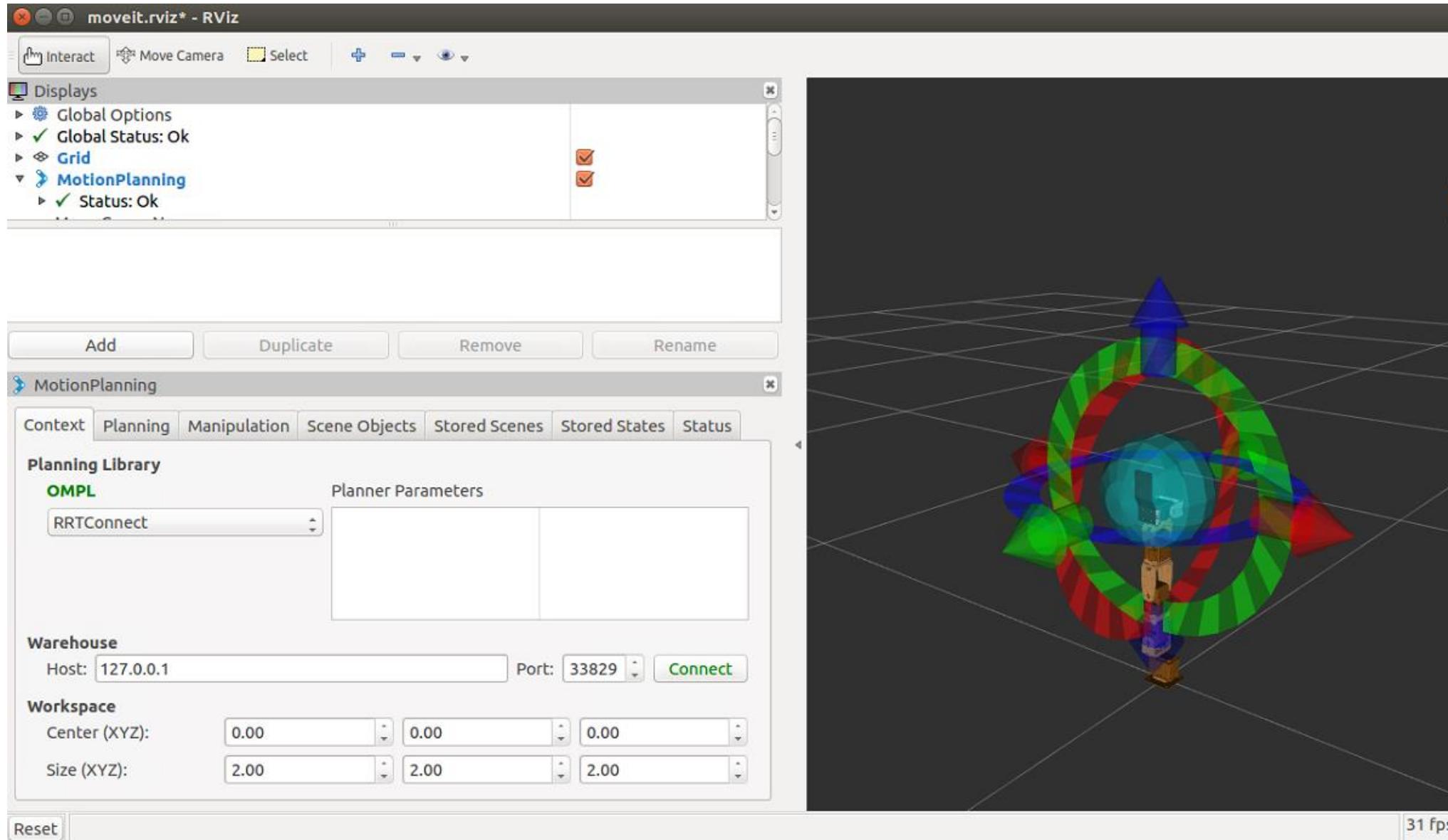
```
$ cd ~/ros_workshop_ws  
$ catkin_make
```

Go to workspace and do catkin_make

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



Movel!



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>
```

Movelt!

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!

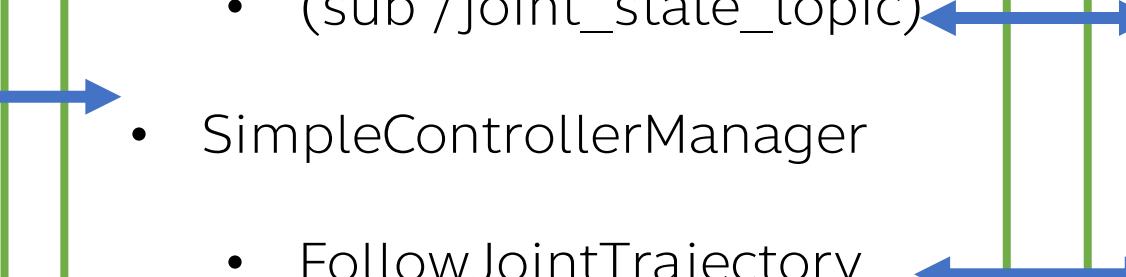
- Planning scene monitor
 - (sub /joint_state_topic)
- SimpleControllerManager
- FollowJointTrajectory Action

MoveIt_*
(my_robot_arm)

**RobotHardware /
Simulation**

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

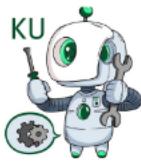
(my_robot_arm/gazebo)
(dynamixel_controller)



Movelt!

- 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>
```



Movelt!

- Create script folder in `my_robot_arm` package

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

- Create python code

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

MoveGroup
Interface



Movelt!

```
#!/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()
```

Movelt!

- 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
```

Movelt!

- 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
```

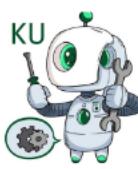
Movelt!

- 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()
```



Movelit!

Code

<https://bit.ly/3csSqax>

MoveIt!

- Run Robot simulation

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

- Run python program

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

MovIt!

Bring robot arm from simulation to real world!

```
$ rosrun my_robot_arm
```

```
$ mkdir config && cd config
```

Movel!

```
$ gedit servo_controller_manager.yaml
```

```
namespace: dynamixel_controller_manager

serial_ports:

    dxl_tty1:

        port_name: "/dev/ttyUSBO"
        baud_rate: 1000000
        min_motor_id: 1
        max_motor_id: 5
        update_rate: 10
```

Movel!

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`).

Movelt!

Explain the code

```
port_name: "/dev/ttyUSBO"
```

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

```
baud_rate: 1000000
```

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

Movelt!

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 ****

Movelt!

Explain the code

```
update_rate: 10
```

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

Movel!

```
$ gedit servo_controller.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
```

Movelt!

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”.

Movel!

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).

Movelt!

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”.

Movelt!

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).

Movel!

```
$ 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
```

Movelt!

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”.

Movelt!

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).

Movelt!

```
$ 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 ***

Movel!

Create node file!

```
$ rosdep my_robot_arm
```

```
$ cd src
```

Movel!

```
$ gedit my_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
```

Movel!

```
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 message(`msg`) from `DynamixelJointState` to `JointState` and publish to `js_publisher`

Movelt!

```
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 message and publish to `js_publisher`

Movelt!

```
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`

Movelt!

```
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)
```

Movel!

```
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))
```

Movelt!

```
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())
```

Movel!

Making launch file!

```
<launch>
  <node name="dynamixel_manager"
        pkg="dynamixel_controllers"
        type="controller_manager.py"
        required="true"
        output="screen" >
    <rosparam file="$(find your_package)/config/servo_controller_manager.yaml"
              command="load" />
  </node>
```

Movel!

Making launch file!

```
<rosparam file="$(find your_package)/config/servo_controllers.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"/>
```

Movelt!

Making launch file!

```
<rosparam file="$(find your_package)/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" />
```

Movelt!

Making launch file!

```
<node name="crane_plus_joint_state_publisher"
      pkg="your_package"
      type="my_state_publisher.py">
  <rosparam
    file="$(find your_package)/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>
```

THANK YOU

