Manipulation with MoveIt

MSc. Robotics

Simon Bøgh

Associate Professor

Faculty of Engineering, Department of Materials and Production

AALBORG UNIVERSITY
DENMARK

Outline

- Manipulation basic concepts
- Manipulation with MoveIt
- MoveIt Setup Assistant
- MoveIt Commander
- Move Group interface
- Motion planning
  - Pick and place behaviors using industrial robots with ROS MoveIt!

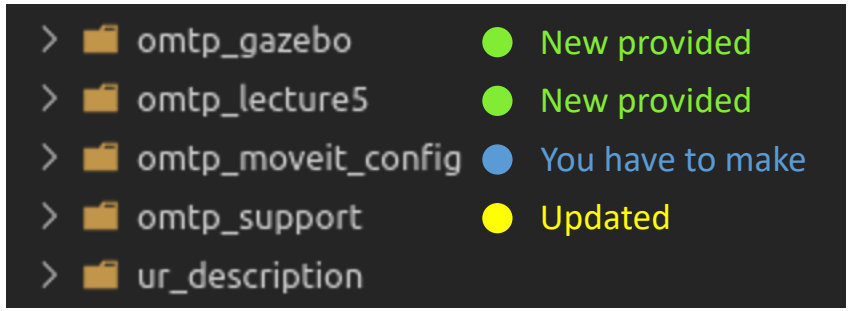
Manipulation overview

- **Goal**
  - Enable our robots in the OMTP factory to do object manipulation

- **MoveIt** – a ROS package for manipulation
- **MoveIt setup assistant** – configure robot setups to use MoveIt
- Plan and execute motions – the "**move_group**" ROS node
- **Implement a simple pick and place pipeline**

OMTP Factory after last time

```
$ roslaunch omtp_support visualize_omtp_factory.launch
```



Manipulation

Basic concepts

Basic concepts

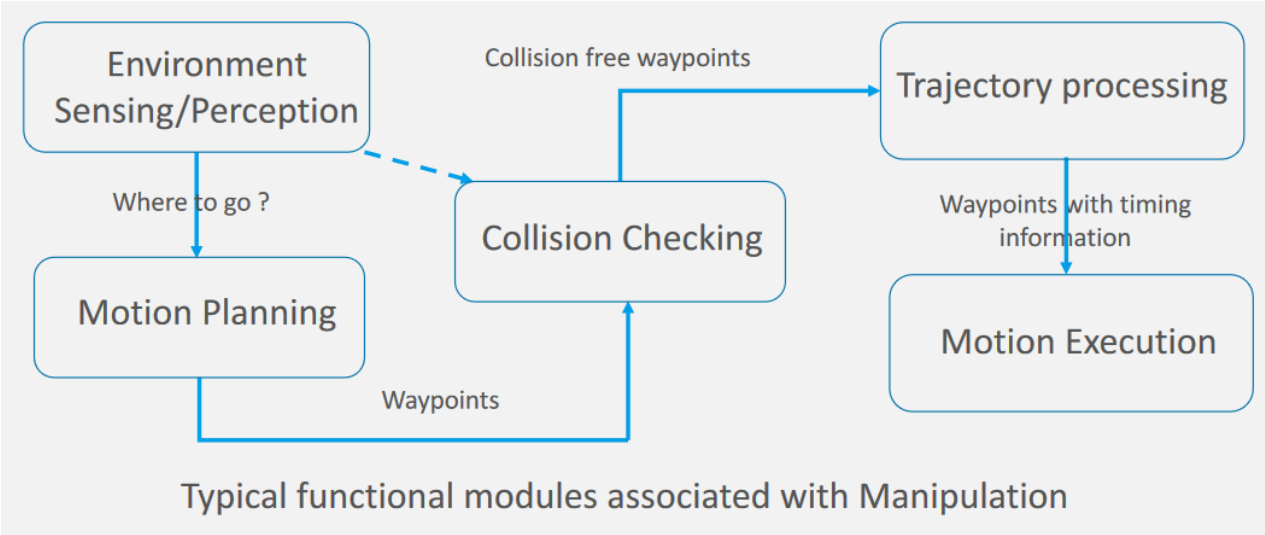
- Interact with the environment and modify it using a robot to perform useful tasks e.g.
  - Pick and place an object
  - Fasten two parts of a car chassis in a car manufacturing environment

- An integral part of automation

Manipulation functional modules

- Manipulation consists of multiple successive steps
- These steps are also known as functional modules
  - We need to **sense the environment** somehow
  - We need to **plan the motion** we wish to perform
  - We need to **check we won't collide** with anything while moving
  - We need to **calculate our trajectory and how fast** we want to go
  - We need to **execute the motion**

Manipulation functional modules



Typical functional modules associated with Manipulation

Mondragon University, Spain

Manipulation with MoveIt!

What is MoveIt?

- Open source software library for manipulation using robot arms
  - Easy integration with ROS

```
$ sudo apt-get install ros-<distro>-moveit
```

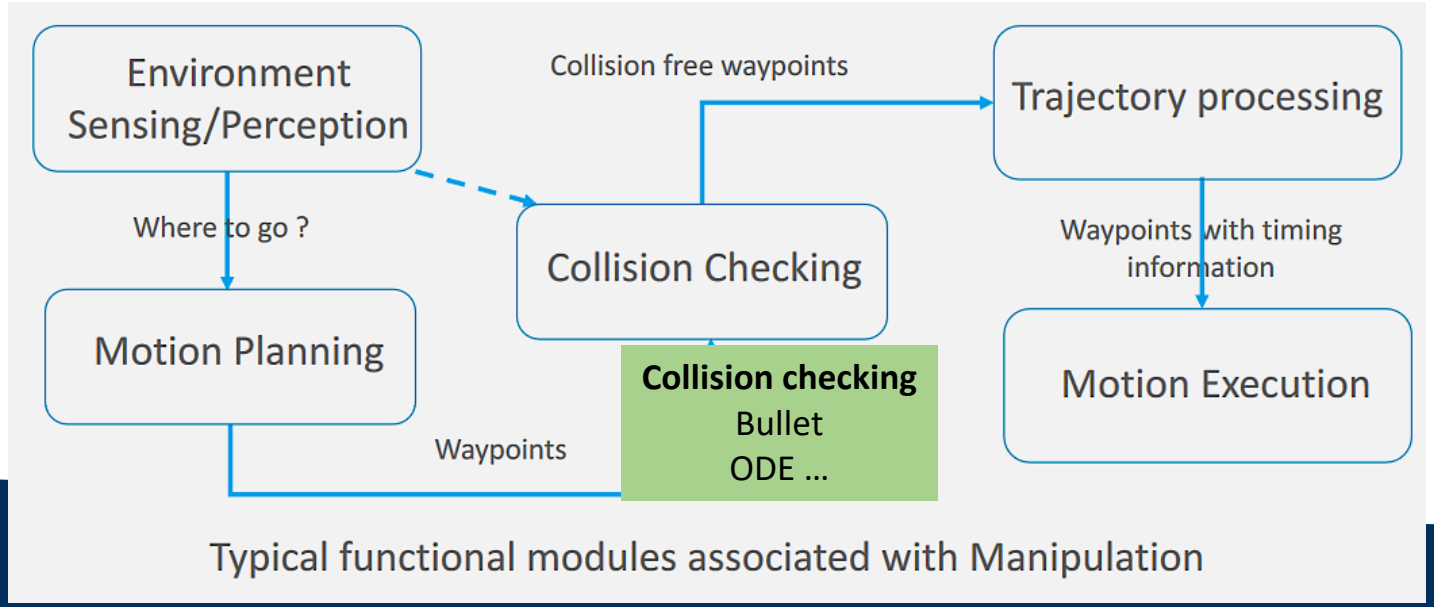
- Often used for serial link manipulators (plan and execute motions)
  - Can also integrate information from 2D and 3D sensors
- Platform to configure and use functionalities associated with manipulation

Functional modules setup

- Functional modules of manipulation will be configured with MoveIt
  1) Sense the environment
  2) Plan the motion
  3) Check for collision
  4) Calculate trajectory and how fast to move
  5) Execute the motion
- **MoveIt glues it all together**

Sensors
Intel RealSense
…

Planning libraries
OMPL
CHOMP …



Typical functional modules associated with Manipulation

Trajectory processing
IPTP
TOPP …

Motion execution
Real Robot (ROS-Industrial driver)
Simulation

MoveIt takes care of many things for us

- **Maintain information consistency** so the modules can work together
- **Integrate robot kinematic** information with planning
- **Report and request alternative motion plans** in case of collisions
- Account for any **hardware limitations** such as joint limits
- **Keep track of the current state of the robot** and its environment while performing a manipulation task
- **Communicate with the robot hardware/simulation** and notify the ROS application once a desired manipulation task is complete

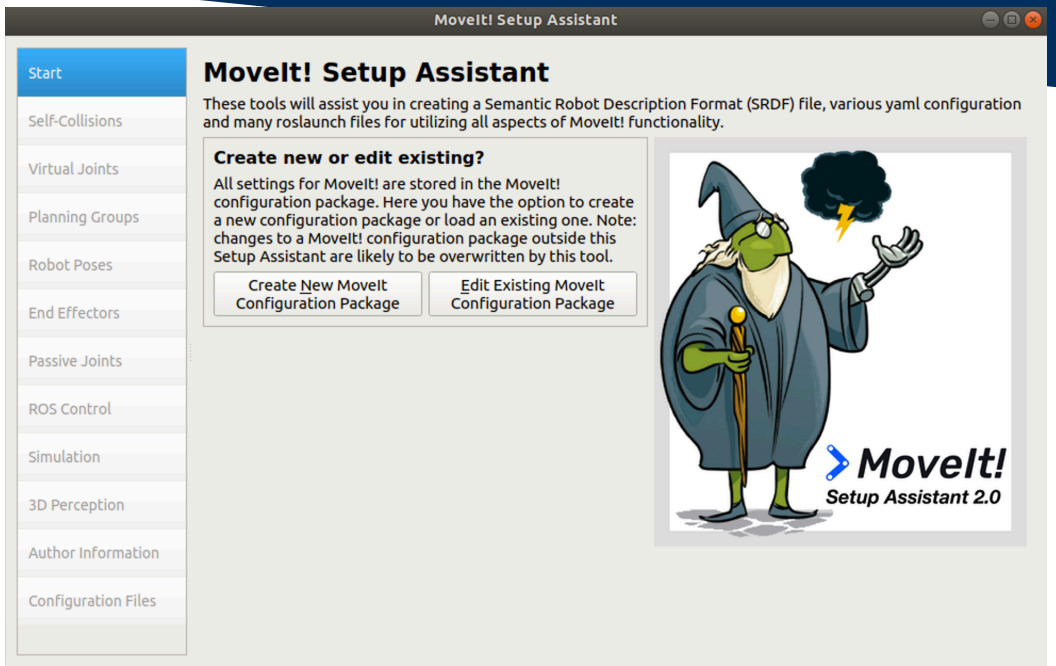
MoveIt - from the user's perspective

- The user only sees the **move_group** ROS node
  - Contains several ROS services and actions

- Configuring the move_group node
  - Robot description (URDF/XACRO) (we built in earlier lecture)
  - Robot semantic description (SRDF) (which we will see soon)
  - Joint limits, planners, etc.

- We set all of this up with the **MoveIt! Setup Assistant**

MoveIt! Setup Assistant



MoveIt! Setup Assistant

- Install MoveIt Setup Assistant

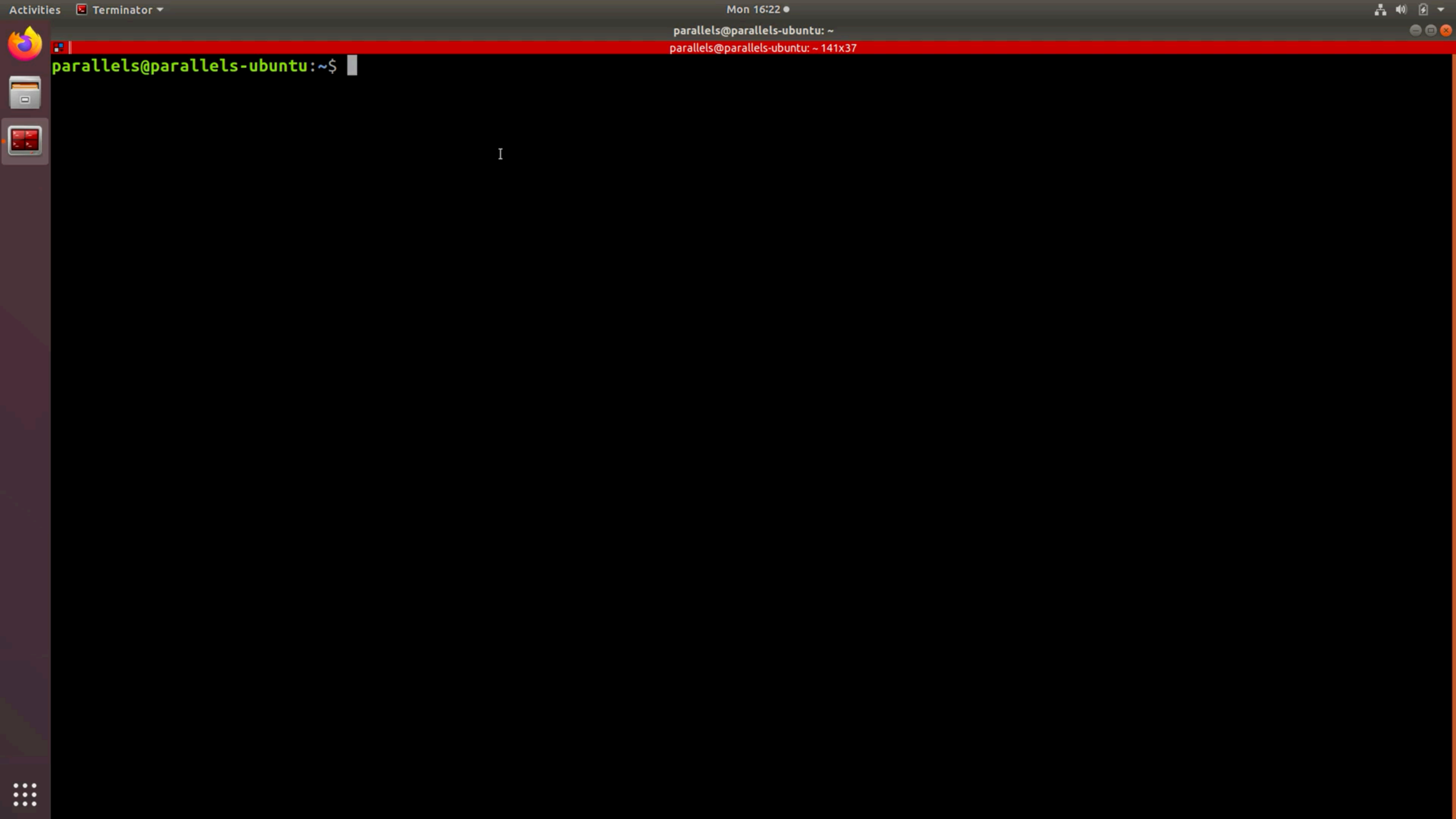
```
$ sudo apt install ros-melodic-moveit-setup-assistant
```

- Source catkin workspace

```
$ source ~/omtp_course_ws/devel/setup.bash
```

- Launch MoveIt Setup Assistant

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


Activities
Terminator
Mon 16:22
parallels@parallels-ubuntu: ~
parallels@parallels-ubuntu: ~ 141x37
parallels@parallels-ubuntu:~$

Learn to use the MoveIt Setup Assistant

- Step 1
  - Import URDF
  - Self-collision
  - Virtual Joints
  - Planning Groups

Learn to use the MoveIt Setup Assistant

- Step 2
  - Robot Poses
  - End Effectors
  - Passive Joints
  - ROS Control: Control the robot's physical hardware
    - ROS Control is a set of packages that include controller interfaces, controller managers, transmissions and hardware_interfaces, for more details please look at ros_control documentation
    - ROS Control tab can be used to auto generate simulated controllers to actuate the joints of your robot. This will allow us to provide the correct ROS interfaces MoveIt!.
  - Simulation
  - 3D Perception
  - Author Information
  - Configuration Files

MoveIt Commander

Testing our MoveIt package

MoveIt Commander

- Command line tool to send motion commands to the robot
- Starting command:

```
$ roslaunch omtp_moveit_config demo.launch

$ rosrun moveit_commander moveit_commander_cmdline.py
```

- Useful to test configuration

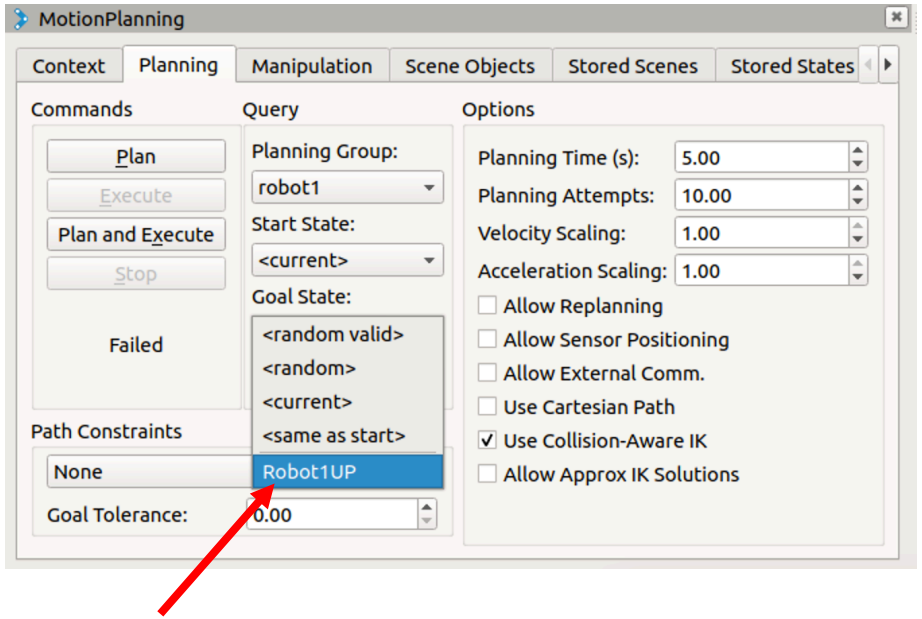
```
> Use robot1
OK

robot1> go Robot1UP
Moved to Robot1UP

robot1> go down 0.2
Moved down by 0.2 m
```

go commands

- backward
- forward
- up
- down
- left
- right
- random



MoveIt Commander

- Load command
  - Create the required script file
  - Name the script and type commands outside the CCS in a regular terminal.

```
$ touch moveit_commander_test
$ gedit moveit_commander_test
```

- For example, you can add the following commands in the script:

```
> use robot1
> go Robot1UP
> go down 0.1
```

- Then, switch back to the terminal where you started MoveIt! Commander and use the following command:

```
> load moveit_commander_test
```

  - The robot corresponding to the planning group you have used in the script should start moving now

Move Group interface

Creating a MoveIt-based ROS application

Move Group Interface

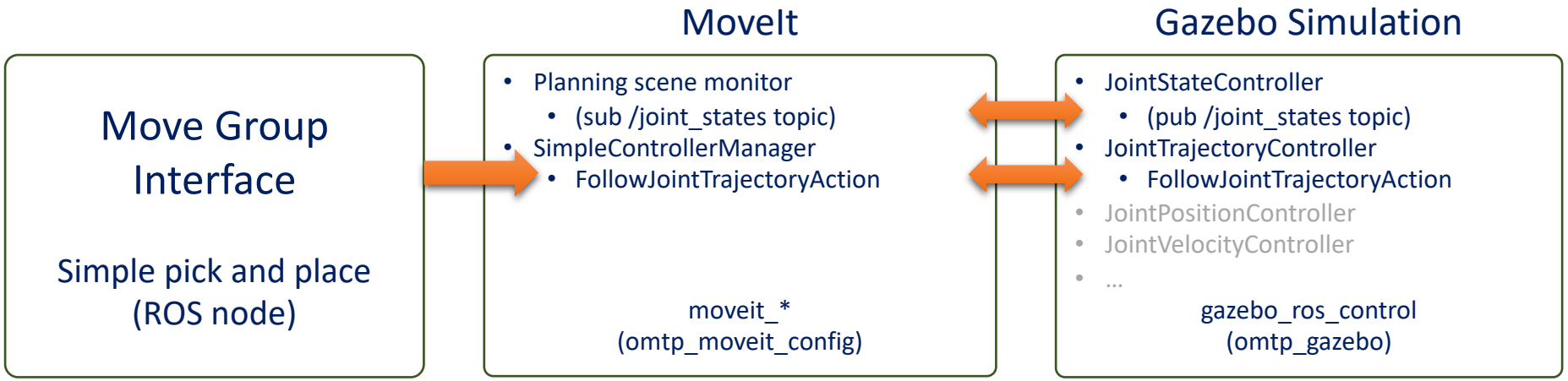
- Goal
  - Enter commands in our own ROS node using the Move Group Interface and create a simple pick and place pipeline

- Move Group interface
  - A collection of APIs to access capabilities of **move_group** ROS node
  - Create MoveIt!-based ROS applications
  - Setup a simple pick and place pipeline
    - Using ROS action clients to talk to MoveIt

Conceptual overview

- **MoveIt** takes care of trajectory execution (with Gazebo or hardware)
- **Move Group Interface** sends these trajectories (MoveIt! Commander uses this implicitly)

Move Group Interface



Move Group Interface sends these trajectories (MoveIt! Commander uses this implicitly)

MoveIt takes care of trajectory execution (with Gazebo or hardware)

Move Group Interface

- MoveIt commander scripts
  - Program sequence of commands to achieve desired motions
- Why learn Move Group Interface to do the same thing?
  - Access more capabilities of **move_group** node
  - Process intermediate results e.g. post-process the trajectory generated
  - Gives overall more flexibility
- Automated behavior in robotic applications
  - Motion module is just one component
  - User input ideally should just be the press of one button

Simple pick and place

Move Group Python Interface

Move Group Python Interface

- We will compose a simple pick and place pipeline with different Move Group APIs

- Move Group Interface APIs
  - **set_named_target** (<"robot joint configuration">)
  - **plan()** - plans a motion to the goal
  - **get_current_pose()** - get pose of the end effector and joint configuration
  - **compute_cartesian_path** (<waypoints>, <resolution>, <jump_threshold>, <collision_checking=True>)

http://docs.ros.org/melodic/api/moveit_tutorials/html/doc/move_group_python_interface/move_group_python_interface_tutorial.html

Joint configuration paths

- Named joint configuration
  - Defined in MoveIt Setup Assistant

```
71  ## Set a named joint configuration as the goal to plan for a move group.
72  ## Named joint configurations are the robot poses defined via MoveIt! Setup Assistant.
73  robot1_group.set_named_target("R1Home")
74
75  ## Plan to the desired joint-space goal using the default planner (RRTConnect).
76  plan = robot1_group.plan()
77  ## Create a goal message object for the action server.
78  robot1_goal = moveit_msgs.msg.ExecuteTrajectoryGoal()
79  ## Update the trajectory in the goal message.
80  robot1_goal.trajectory = plan
81
82  ## Send the goal to the action server.
83  robot1_client.send_goal(robot1_goal)
84  robot1_client.wait_for_result()
85
86  robot1_group.set_named_target("R1PreGrasp")
87
88  plan = robot1_group.plan()
89  robot1_goal = moveit_msgs.msg.ExecuteTrajectoryGoal()
90  robot1_goal.trajectory = plan
91
92  robot1_client.send_goal(robot1_goal)
93  robot1_client.wait_for_result()
```

First goal

Second goal

Cartesian paths

- Computing linear motion
- Specify a list of waypoints for the end-effector to go through

```
95   #  ## Cartesian Paths
96   # ## ^^^^^^^^^^^^^^^^
97   # ## You can plan a cartesian path directly by specifying a list of waypoints
98   # ## for the end-effector to go through.
99   waypoints = []
100  # start with the current pose
101  current_pose = robot1_group.get_current_pose()
102  rospy.sleep(0.5)
103  current_pose = robot1_group.get_current_pose()
104
105  ## create linear offsets to the current pose
106  new_eef_pose = geometry_msgs.msg.Pose()
107
108  # Manual offsets because we don't have a camera to detect objects yet.
109  new_eef_pose.position.x = current_pose.pose.position.x + 0.10
110  new_eef_pose.position.y = current_pose.pose.position.y - 0.20
111  new_eef_pose.position.z = current_pose.pose.position.z - 0.20
112
113  # Retain orientation of the current pose.
114  new_eef_pose.orientation = copy.deepcopy(current_pose.pose.orientation)
115
116  waypoints.append(new_eef_pose)
117  waypoints.append(current_pose.pose)
```

Get current Pose

PoseStamped -> Pose

Add offset

Cartesian paths

- PoseStamped vs. Pose

  `$ rosmsg show geometry_msgs/PoseStamped`

- Pose stamped messages consists of timing and reference frame information along with the pose message type
- The extraction of the pose message can be done as following:

  `current_pose.pose.position.x +0.10`

```
$ rosmsg show geometry_msgs/PoseStamped
std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
geometry_msgs/Pose pose
      geometry_msgs/Point position
            float64 x
            float64 y
            float64 z
       geometry_msgs/Quaternion orientation
            float64 x
            float64 y
            float64 z
            float64 w
```

Cartesian paths

- Move Group API for Cartesian Paths

```
(plan_cartesian, fraction) = robot1_group.compute_cartesian_path(…)
```

```
119  ## We want the cartesian path to be interpolated at a resolution of 1 cm
120  ## which is why we will specify 0.01 as the eef_step in cartesian
121  ## translation.  We will specify the jump threshold as 0.0, effectively
122  ## disabling it.
123  fraction = 0.0
124  for count_cartesian_path in range(0,3):
125    if fraction < 1.0:
126      (plan_cartesian, fraction) = robot1_group.compute_cartesian_path(
127                                   waypoints,   # waypoints to follow
128                                   0.01,        # eef_step
129                                   0.0)         # jump_threshold
130    else:
131      break
132
133  robot1_goal = moveit_msgs.msg.ExecuteTrajectoryGoal()
134  robot1_goal.trajectory = plan_cartesian
135
136  robot1_client.send_goal(robot1_goal)
137  robot1_client.wait_for_result()
```

Cartesian path interpolation

Prepare the msg

Send the msg and wait

Intro to Move Group Python Interface

- Part 1
  - Python script, go through the **simple_pick_place.py** script
  - The required specific required modules are:
    - moveit_commander – tell python we work with MoveIt!
    - moveit_msgs.msg
    - actionlib – for the movement with actionlib.SimpleActionClient()
    - geometry_msgs – for planning lineair or carthesian spaced motions
- We will use
  - **set_named_target** – set a goal configuration
  - **plan()** – plans a motion to the goal
  - **.send_goal()** – send the goal to the action server

Intro to Move Group Python Interface

- Part 2
  - Test our pick and place pipeline
    - chmod +x simple_pick_place.py
    - rosrun
  - Test the nonblocking execution
    - Remove the following function and restart the script robot1_client.wait_for_result()
  - Results of first robot:
    - Preemption of the first goal
    - Only the second was executed
  - Results second robot:
    - Two separate clients which send information to the same server
    - Blocking
- Waypoints:
  - Poses of robot end-effector
  - Timing synchronisation results in an incorrect pose
  - Issues with get_current_pose() API
    - Fixed with delay: rospu.sleep(0.5)
  - Waypoints should only consist of pose messages
  - Linear offsets with geometry_msgs.msg.Pose()

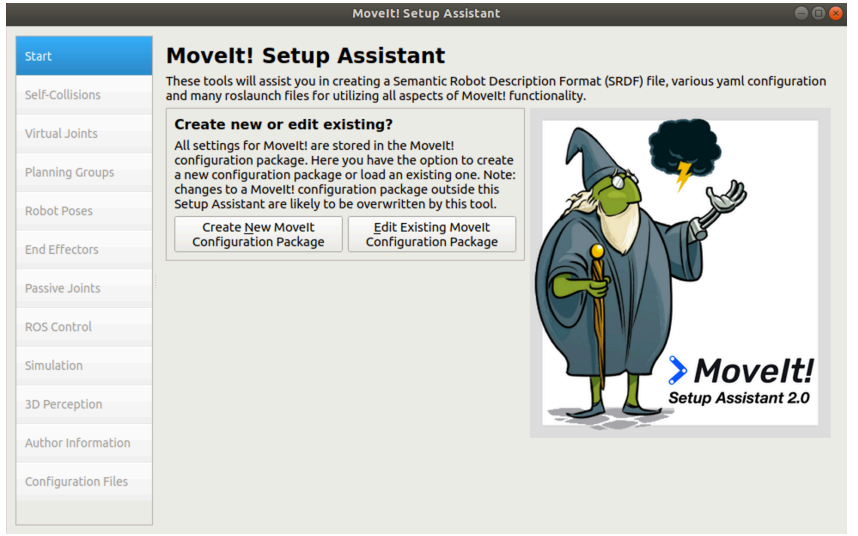
Move Group Python Interface

- Documentation
  - http://docs.ros.org/melodic/api/moveit_tutorials/html/doc/move_group_python_interface/move_group_python_interface_tutorial.html

Assignments

1. Create a MoveIt package using MoveIt Setup Assistant

- Configure a Move Group for each robot
- Create the following poses:
  - Robot 1: R1Up, R1Home, R1PreGrasp, R1Place
  - Robot 2: R2Up, R2Home, R2PreGrasp, R2Place
- Save the package as *omtp_moveit_config*
- NOTE: edit *ros_controllers.yaml*, add namespace
  - name: **robot1/**robot1_controller
  - name: **robot2/**robot2_controller



Assignments

2. Use MoveIt Commander to execute a script of motions

- Create a script to execute the following sequence
  - R1Home → R1PreGrasp → move down 0.1m → R1Place → move up 0.2m
- Start script from MoveIt Commander: *> load your_script_name*

Assignments

3. Complete lecture5_assignment3.py

- Only change code where it says ***<write your code here>***
- Make the Python script executable ***chmod +x lecture5_assignment3.py***
- Start the factory simulation in gazebo: ***$ roslaunch omtp_lecture5 omtp_lecture5_environment.launch***
- Launch your script: ***$ roslaunch omtp_lecture5 lecture5_assignment3.launch***
- You should see the robot do simple pick and place behavior

Submit assignment to git repo

- Submit assignments and post on MS Teams

Extras

Notes for reference

ROS controllers

- $ sudo apt install ros-melodic-ros-control

OMTP MoveIt config package

- MoveIt Setup Assistant
  - Create *omtp_moveit_config* package from *omtp_support/urdf/omtp_factory.xacro*

- Edit ros_controllers.yaml, add namespace
  - - name: **robot1/**robot1_controller
  - - name: **robot2/**robot2_controller

- Unpause Gazebo

```
30  controller_list:
31    - name: robot1/robot1_controller
32      action_ns: follow_joint_trajectory
33      default: True
34      type: FollowJointTrajectory
35      joints:
36        - robot1_shoulder_pan_joint
37        - robot1_shoulder_lift_joint
38        - robot1_elbow_joint
39        - robot1_wrist_1_joint
40        - robot1_wrist_2_joint
41        - robot1_wrist_3_joint
42    - name: robot2/robot2_controller
43      action_ns: follow_joint_trajectory
44      default: True
```

How to run

- $ roslaunch omtp_lecture5 omtp_lecture5_environment.launch
- $ rosrun moveit_commander moveit_commander_cmdline.py

Gazebo tips

- Start Gazebo without GUI (gzclient) in your launch-file
  - gui:=false
- You can always start the GUI from terminal
  - $ gzclient
- Unpause Gazebo sim with rosservice
  - $ rosservice call /gazebo/unpause_physics "{}"
- Increase Gazebo GUI FPS
  - Change view from Perspective to Orthographic