In this lecture, we will continue with composing a simple pick and place pipeline with different MoveGroup APIs.

Important notes:

1. This requires the instructions for the MoveIt Setup Assistant to be completed. (Units 4.31 to 4.3.4)
   If you haven't followed them, please go to this previous units, and pay special attention to the additional steps at the end of unit 4.3.3.
2. In the new version of ROS, the plan() function returns a tuple. Therefore, the old code:
   `plan = robot1_group.plan(),` could no longer be used.
   It has been replaced by the following code: `_, plan, _, _ = robot1_group.plan()`

Pick & Place: Part 2

0:00 / 0:00

1.0x

Testing our pick and place pipeline:

Start the factory simulation.

```
$ roslaunch hrwros_gazebo hrwros_environment.launch
```

In a new terminal source our setup files.

```
$ source $HOME/hrwros_ws/devel/setup.bash
$ roscd hrwros_week4/scripts/
```

Uncomment the following function in `.../hrwros_week4/scripts/simple_pick_place.py` (#L84):

```
robot1_client.wait_for_result()
```

Launch the node

```
$ roslaunch hrwros_week4 hrwros_simple_pick_place.launch
```

Now let's test the non-blocking execution. Open up `.../hrwros_week4/scripts/simple_pick_place.py`:

Remove the following function and relaunch the program (#L84).

```
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

```
current_pose = robot1_group.get_current_pose()
rospy.sleep(0.5)
current_pose = robot1_group.get_current_pose()
```

- Waypoints should only consist of pose messages
- Linear offsets with `geometry_msgs.msg.Pose()`.

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`

Finally, we can add the newly created waypoint and the current pose.

Question 1

1 point possible (ungraded)

We tried testing the non-blocking functionality by sending two goals one after the other from robot1_client. And we were told that this would also not work if we send a goal at the same time using the robot2_client for the second robot. What will actually happen if we do so assuming the planning is always successful?

- ◯ Robot1 will not move at all, and only robot2 will move.
- ◯ Robot1 will start moving and stop when the second goal is sent by robot2 to the same execute_trajectory action server. The second robot will complete executing the trajectory towards the specified named target, if we wait for the result of execution.
- ◯ Both Robots will reach their respective goals.
- ◯ Nothing will happen. Both robots will not move..

Submit

Question 2

1 point possible (ungraded)

The waypoints to generate linear motion paths for the end-effector of the robot arms are a list of elements with geometry_msgs/PoseStamped message type.

- ◯ True
- ◯ False

Submit