Project Intelligent Robotics
Assignment #4

The previous assignments introduced the basic ROS framework and its interfaces. The next assignment requires you to utilize 3rd-party packages built in ROS to control the actions of a robot arm. The major project to do that in the context of ROS is called “MoveIt!”. Most of its documentation can be found at `http://moveit.ros.org`.

Task 4.1 Install the description and configuration packages for TAMS’ UR5 setup:

Whereas the MoveIt! packages are already installed on the lab computers, you have to download and build the packages specific to the UR5 robot setup yourself.

The most easy way to checkout all of them uses ROS’ workspace-management tool `wstool`:

- Navigate to `http://github.com/TAMS-Group/rosinstalls` and download the file `kinetic-tams-ur5-setup+pick-place.rosinstall`.
- Move it into your workspace’s `src` folder and rename it to `.rosinstall`.
- Now you can use `wstool` (with all its subcommands) to manage the workspace. Most notably you can call `wstool update` to download and setup all repositories listed in the file.
- Build your workspace

Task 4.2 Geometric Simulation, Kinematics and Path Planning:

4.2.1: Now you should be able to launch the geometric simulation, i.e. “demo mode”, for the setup:

```
roslaunch tams_ur5_setup_moveit_config demo.launch
```

This command launches the MoveIt! demo mode with TAMS’ configuration together with the RViz based visualization for the framework. You will need to have this launch file running for all further tasks on this assignment sheet.

4.2.2: You can move the colored interactive marker in RViz to select a goal configuration the arm should reach. The visualization automatically computes *inverse kinematics (IK)* for you. Thus, you only specify the pose of the *end-effector* and the orange robot model visualizes the chosen *robot pose* (the joint configuration) to reach that pose.

4.2.3: Choose the “Planning” tab in the lower left panel. On the left side you can request MoveIt! to plan (and execute) a collision-free path from the current state of the arm to your goal state. You can choose the planner used to compute the path in the “Context” tab. Assess the quality and success of different planners given different start/goal states and planning time.

Task 4.3 Code Interfaces:

4.3.1: Implement a ROS node that moves the arm:

- Create a ROS package that depends on the packages
  - `roscpp`
  - `moveit_ros_planning_interface`
- Create a C++ ROS node in this package
- Create an instance of the class `moveit::planning_interface::MoveGroupInterface` to interface with MoveIt. You can find API documentation on the class on the website.

  The name of the planning group for the UR5 arm (without the attached gripper) is "`arm`".
- Use the `move()` method of your instance to plan and execute a motion to a previously set goal state
- There are many ways to define a goal before calling `move()`. Use at least two of the following:
  - Use `setNamedTarget(const std::string&)` to set the goal to one of the pre-specified robot poses available for the setup. You can find a list in the "Goal State"-field in RViz.
  - The current joint states of the robot are published to the topic `/joint_states`. Use `rostopic echo /joint_states` to find the current values of all ur5-related joints. Set them as the goal using one of the `setJointValueTarget` methods.
  - Use the `setJointValueTarget(const geometry_msgs::PoseStamped& p)` method to request a target pose `p` for the end-effector.

    Set `p.header.frame_id` to "`table_top`" in order to specify the pose with respect to the ROS frame `table_top`. To visualize this frame in RViz, you can add an "Axis" display and set the frame as its reference.

    For now, don't change the orientation of your target pose. Instead specify `p.pose.orientation.w` as `1.0` and leave the other three components at `0.0`.

4.3.2: Write another node that adds a collision object, i.e. an obstacle the arm should avoid, to the planning scene in MoveIt.

- Create an instance of the class `moveit::planning_interface::PlanningSceneInterface`. Again, the API documentation can be found on the website.
- Use the method `applyCollisionObject(const moveit_msgs::CollisionObject&)` to spawn the obstacle. To do so, create a `moveit_msgs::CollisionObject` message, fill in the fields to specify a tall object on top of the table and pass it to the method.
- Once more, plan and execute paths and notice that the arm should not collide with the specified collision object.