Gazebo pick and place world  
~/catkin\_ws/src$ git clone <https://github.com/dairal/ur5_simple_pick_and_place>

In the gazebo.launch file change line 9  
<**arg** name="world\_name" default="$(find ur5\_simple\_pick\_and\_place)/world/simple\_pick\_and\_place"/>

Line 18  
<node name="spawn\_gazebo\_model" pkg="gazebo\_ros" type="spawn\_model" args="-urdf -param robot\_description -model robot -x 0 -y 0 -z 1.21"

$ roslaunch ur5\_gripper\_moveit\_config demo\_gazebo.launch

Initializing the interface

Our C++ code is in a file called simple\_pick\_and\_place.cpp inside the src folder

#include <moveit/move\_group\_interface/move\_group\_interface.h>

#include <moveit/planning\_scene\_interface/planning\_scene\_interface.h>

**int** main(**int** argc, **char**\*\* argv)

{

  ros::init(argc, argv, "move\_group\_interface\_tutorial");

  ros::NodeHandle n;

  // ROS spinning must be running for the MoveGroupInterface to get information

  // about the robot's state. One way to do this is to start an AsyncSpinner

  // beforehand.

  ros::AsyncSpinner spinner(1);

  spinner.start();

  // MoveIt operates on sets of joints called "planning groups" and stores them in an object called

  // the `JointModelGroup`. Throughout MoveIt the terms "planning group" and "joint model group"

  // are used interchangably.

**static** **const** std::string PLANNING\_GROUP\_ARM = "ur5\_arm";

**static** **const** std::string PLANNING\_GROUP\_GRIPPER = "gripper";

  // The :planning\_interface:`MoveGroupInterface` class can be easily

  // setup using just the name of the planning group you would like to control and plan for.

  moveit::planning\_interface::MoveGroupInterface move\_group\_interface\_arm(PLANNING\_GROUP\_ARM);

  moveit::planning\_interface::MoveGroupInterface move\_group\_interface\_gripper(PLANNING\_GROUP\_GRIPPER);

Commanding the motions:

All available planning groups:

// We can get a list of all the groups in the robot:

ROS\_INFO\_NAMED("tutorial", "Available Planning Groups:");

std::copy(move\_group\_interface\_arm.getJointModelGroupNames().begin(),

        move\_group\_interface\_arm.getJointModelGroupNames().end(), std::ostream\_iterator<std::string>(std::cout, ", "));

Move to home position:

moveit::planning\_interface::MoveGroupInterface::Plan my\_plan\_arm;

// 1. Move to home position

move\_group\_interface\_arm.setJointValueTarget(move\_group\_interface\_arm.getNamedTargetValues("home"));

**bool** success = (move\_group\_interface\_arm.plan(my\_plan\_arm) == moveit::planning\_interface::MoveItErrorCode::SUCCESS);

ROS\_INFO\_NAMED("tutorial", "Visualizing plan 1 (pose goal) %s", success ? "" : "FAILED");

move\_group\_interface\_arm.move();

Place the TCP above the blue box:

// 2. Place the TCP (Tool Center Point, the tip of the robot) above the blue box

geometry\_msgs::PoseStamped current\_pose;

current\_pose = move\_group\_interface\_arm.getCurrentPose("ee\_link");

geometry\_msgs::Pose target\_pose1;

target\_pose1.orientation = current\_pose.pose.orientation;

target\_pose1.position.x = 0.3;

target\_pose1.position.y = 0.5;

target\_pose1.position.z = 0.2;

move\_group\_interface\_arm.setPoseTarget(target\_pose1);

success = (move\_group\_interface\_arm.plan(my\_plan) == moveit::planning\_interface::MoveItErrorCode::SUCCESS);

ROS\_INFO\_NAMED("tutorial", "Visualizing plan 1 (pose goal) %s", success ? "" : "FAILED");

move\_group\_interface\_arm.move();

Open the gripper:

moveit::planning\_interface::MoveGroupInterface::Plan my\_plan\_gripper;

// 3. Open the gripper

move\_group\_interface\_gripper.setJointValueTarget(move\_group\_interface\_gripper.getNamedTargetValues("open"));

Move the TCP close to the object:

// 4. Move the TCP close to the object

target\_pose1.position.z = target\_pose1.position.z - 0.2;

move\_group\_interface\_arm.setPoseTarget(target\_pose1);

Close the gripper:

// 5. Close the  gripper

move\_group\_interface\_gripper.setJointValueTarget(move\_group\_interface\_gripper.getNamedTargetValues("closed"));

Move the TCP above the plate:

// 6. Move the TCP above the plate

target\_pose1.position.z = target\_pose1.position.z + 0.2;

target\_pose1.position.x = target\_pose1.position.x - 0.6;

move\_group\_interface\_arm.setPoseTarget(target\_pose1);

Lower TCP and open the gripper:

// 7. Lower the TCP above the plate

target\_pose1.position.z = target\_pose1.position.z - 0.14;

move\_group\_interface\_arm.setPoseTarget(target\_pose1);

// 8. Open the gripper

move\_group\_interface\_gripper.setJointValueTarget(move\_group\_interface\_gripper.getNamedTargetValues("open"));

~/catkin\_ws$ catkin\_make

rosrun ur5\_simple\_pick\_and\_place pick\_and\_place

Collision avoidance with Moveit:

Change line 9 of the file gazebo.launch file:

<**arg** name="world\_name" default="$(find ur5\_simple\_pick\_and\_place)/world/simple\_pick\_and\_place\_collision"/>

roslaunch ur5\_gripper\_moveit\_config demo\_gazebo.launch

Adding collision objects to the moveit planning group:

moveit::planning\_interface::PlanningSceneInterface planning\_scene\_interface;

// Collision object

moveit\_msgs::CollisionObject collision\_object;

collision\_object.header.frame\_id = move\_group\_interface\_arm.getPlanningFrame();

collision\_object.id = "box1";

shape\_msgs::SolidPrimitive primitive;

primitive.type = primitive.BOX;

primitive.dimensions.resize(3);

primitive.dimensions[0] = 0.2;

primitive.dimensions[1] = 0.4;

primitive.dimensions[2] = 0.4;

geometry\_msgs::Pose box\_pose;

box\_pose.orientation.w = 1.0;

box\_pose.position.x = 0.0;

box\_pose.position.y = 0.5;

box\_pose.position.z = 1.23 - 1.21;

collision\_object.primitives.push\_back(primitive);

collision\_object.primitive\_poses.push\_back(box\_pose);

collision\_object.operation = collision\_object.ADD;

std::vector<moveit\_msgs::CollisionObject> collision\_objects;

collision\_objects.push\_back(collision\_object);

planning\_scene\_interface.applyCollisionObjects(collision\_objects);

ROS\_INFO\_NAMED("tutorial", "Add an object into the world");

ros::Duration(2.0).sleep();

ROS\_INFO\_NAMED("tutorial", "Remove the object from the world");

std::vector<std::string> object\_ids;

object\_ids.push\_back(collision\_object.id);

planning\_scene\_interface.removeCollisionObjects(object\_ids);

rosrun simple\_pick\_and\_place pick\_and\_place\_collision