Collision detection

Preface: Raspberry Pi 5 and Jetson Nano run ROS in Docker, so the performance of running MoveIt2 is generally poor. Users of Raspberry Pi 5 and Jetson Nano boards are advised to run MoveIt2 examples in a virtual machine. Orin motherboards run ROS directly on the motherboard, so users of Orin boards can run MoveIt2 examples directly on the motherboard, using the same instructions as running in a virtual machine. This section uses running in a virtual machine as an example.

1. Content Description

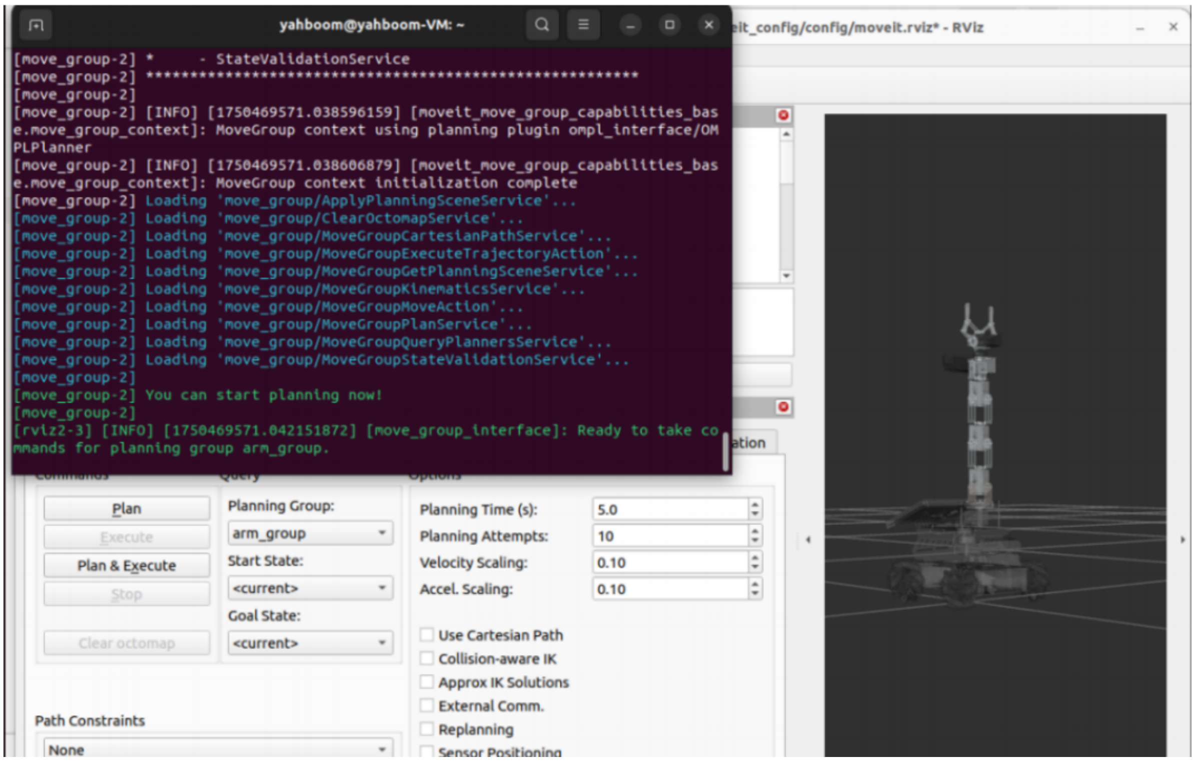
This lesson explains how to use the MoveIt library to add obstacles to rviz and implement obstacle avoidance when MoveIt plans motion.

2. Start

Open a terminal in the virtual machine and enter the following command to start MoveIt2.

```
ros2 launch test_moveit_config demo.launch.py
```

After the program is started, when the terminal displays **"You can start planning now!"** , it indicates that the program has been successfully started, as shown in the figure below.



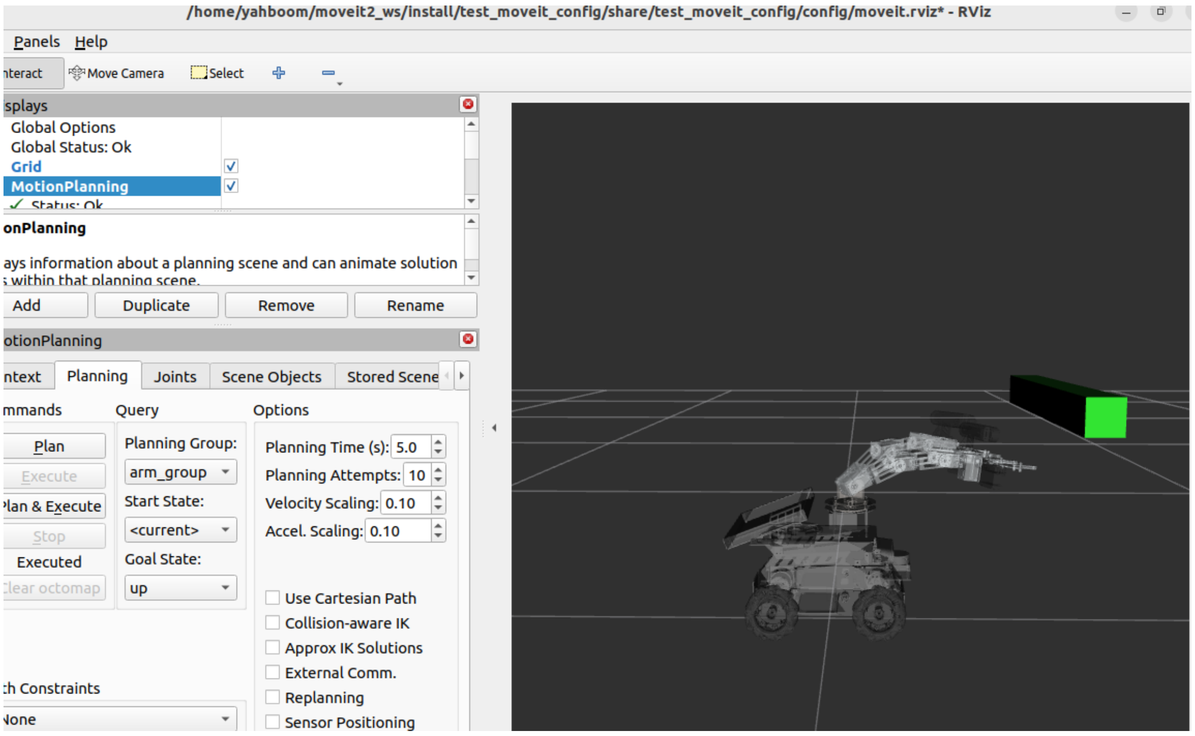
```
[move_group-2] *     - StateValidationService
[move_group-2] **********************************************************
[move_group-2]
[move_group-2] [INFO] [1750469571.038596159] [moveit_move_group_capabilities_bas
e.move_group_context]: MoveGroup context using planning plugin ompl_interface/OM
PLPlanner
[move_group-2] [INFO] [1750469571.038606879] [moveit_move_group_capabilities_bas
e.move_group_context]: MoveGroup context initialization complete
[move_group-2] Loading 'move_group/ApplyPlanningSceneService'...
[move_group-2] Loading 'move_group/ClearOctomapService'...
[move_group-2] Loading 'move_group/MoveGroupCartesianPathService'...
[move_group-2] Loading 'move_group/MoveGroupExecuteTrajectoryAction'...
[move_group-2] Loading 'move_group/MoveGroupGetPlanningSceneService'...
[move_group-2] Loading 'move_group/MoveGroupKinematicsService'...
[move_group-2] Loading 'move_group/MoveGroupMoveAction'...
[move_group-2] Loading 'move_group/MoveGroupPlanService'...
[move_group-2] Loading 'move_group/MoveGroupQueryPlannersService'...
[move_group-2] Loading 'move_group/MoveGroupStateValidationService'...
[move_group-2]
[move_group-2] You can start planning now!
[move_group-2]
[rviz2-3] [INFO] [1750469571.042151872] [move_group_interface]: Ready to take co
mmands for planning group arm_group.
```

Planning Group: arm_group
Start State: <current>
Goal State: <current>
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
Path Constraints: None

Then, we enter the following command in the virtual machine terminal to start the collision detection program,

```
ros2 run MoveIt_demo obstacle_avoidance
```

After the program is run, a green block will be added to rviz. The robot arm will first move to the up position, then plan a movement to the down position, avoiding obstacles during the process; finally, it will plan a movement from the down position to the up position, also avoiding obstacles during the process. As shown in the figure below,



Planning Group: arm_group
Start State: <current>
Goal State: up
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
Use Cartesian Path
Collision-aware IK
Approx IK Solutions
External Comm.
Replanning
Sensor Positioning
None

3. Core code analysis

The code path in the virtual machine is: `/home/yahboom/moveit2_ws/src/MoveIt_demo/src/obstacle_avoidance.cpp`

```
#include <rclcpp/rclcpp.hpp>
#include <moveit/move_group_interface/move_group_interface.h>
#include <moveit/planning_scene_interface/planning_scene_interface.h>
#include <geometry_msgs/msg/pose.hpp>


class  Avoidance  : public  rclcpp::Node
{
public :
  Avoidance ()
    : Node ( "random_target_move" )
  {
    // Initialize other content
    RCLCPP_INFO ( this -> get_logger (), "Initializing AvoidanceMoveIt2Control."
);
  }

  void  initialize ()
  {

    int  max_attempts  =  5 ;   // Maximum number of planning attempts
    int  attempt_count  =  0 ;   // Current number of attempts
    // Initialize move_group_interface_ in this function and create a planning
group named arm_group
```

```
    move_group_interface_  =  std::make_shared <
moveit::planning_interface::MoveGroupInterface > ( shared_from_this (),
"arm_group" );
    //Create an interface for managing planning scenes to add collision objects
(obstacles) to the scene
    planning_scene_interface_  =  std::make_shared <
moveit::planning_interface::PlanningSceneInterface > ();

    move_group_interface _-> setNumPlanningAttempts ( 10 );   // Set the maximum
number of planning attempts to 10
    move_group_interface _-> setPlanningTime ( 5.0 );          // Set the
maximum time for each planning to 5 seconds

    //Define a collision object
    moveit_msgs::msg::CollisionObject  collision_object ;
    collision_object . header . frame_id  =  move_group_interface_ ->
getPlanningFrame ();
    collision_object . id  =  "box1" ;

    // Create a simple collision body or visualization object to describe the
robot's environment
    shape_msgs::msg::SolidPrimitive  primitive ;
    //Set the type of geometric object to box
    primitive . type  =  primitive . BOX ;
    primitive . dimensions . resize ( 3 );
    //Set the size of the geometric object box in meters
    primitive . dimensions [ primitive . BOX_X ] =  0.05 ;
    primitive . dimensions [ primitive . BOX_Y ] =  0.05 ;
    primitive . dimensions [ primitive . BOX_Z ] =  0.5 ;

    //Create an object that describes the box pose of the collection object and
assign the data in the object
    geometry_msgs::msg::Pose  box_pose ;
    box_pose . orientation . w  =  0.7071 ;
    box_pose . orientation . x  =  0.7071 ;
    box_pose . position . x  =  0.35 ;
    box_pose . position . y  =  0.0 ;
    box_pose . position . z  =  0.35 ;

    //Add the shape of the collision object, the primitive just defined
    collision_object . primitives . push_back ( primitive );
    //Add the pose of the collision object, the box_pose just defined
    collision_object . primitive_poses . push_back ( box_pose );
    //Add collision objects to the environment
    collision_object . operation  =  collision_object . ADD ;

    std::vector < moveit_msgs::msg::CollisionObject >  collision_objects ;
      //Add a collision object to the pending list
    collision_objects . push_back ( collision_object );

    RCLCPP_INFO ( this -> get_logger (), "Add an object into the world" );
    //Add collision objects to rviz for display
    planning_scene_interface_ -> addCollisionObjects ( collision_objects );

    // Plan the path
    moveit::planning_interface::MoveGroupInterface::Plan  my_plan ;

    while ( attempt_count  <  max_attempts )
    {
        attempt_count ++ ;
        // Set the predefined target position
        move_group_interface_ -> setNamedTarget ( "up" );
        // Create a plan and execute it
        moveit::planning_interface::MoveGroupInterface::Plan  my_plan ;
        bool  success  = ( move_group_interface_ -> plan ( my_plan ) ==
 moveit::core::MoveItErrorCode::SUCCESS );
        //If the plan is successful, then execute the plan
        if ( success )
        {
            RCLCPP_INFO ( this -> get_logger (), "Planning succeeded, moving the
arm." );
            move_group_interface_ -> execute ( my_plan );
            attempt_count  =  0 ;
            break ;
        }
        else
        {
            RCLCPP_INFO ( this -> get_logger (), "Planning failed!" );
        }
    }


    while ( attempt_count  <  max_attempts )
    {
        attempt_count ++ ;
        // Set the predefined target position
        move_group_interface_ -> setNamedTarget ( "down" );
        // Create a plan and execute it
        moveit::planning_interface::MoveGroupInterface::Plan  my_plan ;
         bool  success  = ( move_group_interface_ -> plan ( my_plan ) ==
 moveit::core::MoveItErrorCode::SUCCESS );
        //If the plan is successful, then execute the plan
        if ( success )
        {
            RCLCPP_INFO ( this -> get_logger (), "Planning succeeded, moving the
arm." );
            move_group_interface_ -> execute ( my_plan );
            attempt_count  =  0 ;
            break ;
        }
        else
        {
            RCLCPP_INFO ( this -> get_logger (), "Planning failed!" );
        }
    }


    while ( attempt_count  <  max_attempts )
    {
        attempt_count ++ ;
        // Set the predefined target position
        move_group_interface_ -> setNamedTarget ( "up" );
        // Create a plan and execute it
        moveit::planning_interface::MoveGroupInterface::Plan  my_plan ;
        bool  success  = ( move_group_interface_ -> plan ( my_plan ) ==
 moveit::core::MoveItErrorCode::SUCCESS );
```

```
        //If the plan is successful, then execute the plan
        if ( success )
        {
            RCLCPP_INFO ( this -> get_logger (), "Planning succeeded, moving the
arm." );
            move_group_interface_ -> execute ( my_plan );
            attempt_count  =  0 ;
            break ;
        }
        else
        {
            RCLCPP_INFO ( this -> get_logger (), "Planning failed!" );
        }
    }

  }

private :
  std::shared_ptr < moveit::planning_interface::MoveGroupInterface >
 move_group_interface_ ;
  std::shared_ptr < moveit::planning_interface::PlanningSceneInterface >
 planning_scene_interface_ ;
};

int  main ( int  argc , char  ** argv )
{
  rclcpp::init ( argc , argv );
  auto  node  =  std::make_shared < Avoidance > ();

  // Initialization
  node- > initialize ();

  rclcpp::spin ( node );
  rclcpp::shutdown ();
  return  0 ;
}
```