

# Part 1 MoveIt Configuration

Note: MoveIt has already been configured before delivery, and you can experience MoveIt simulation directly.

## 1. MoveIt introduction

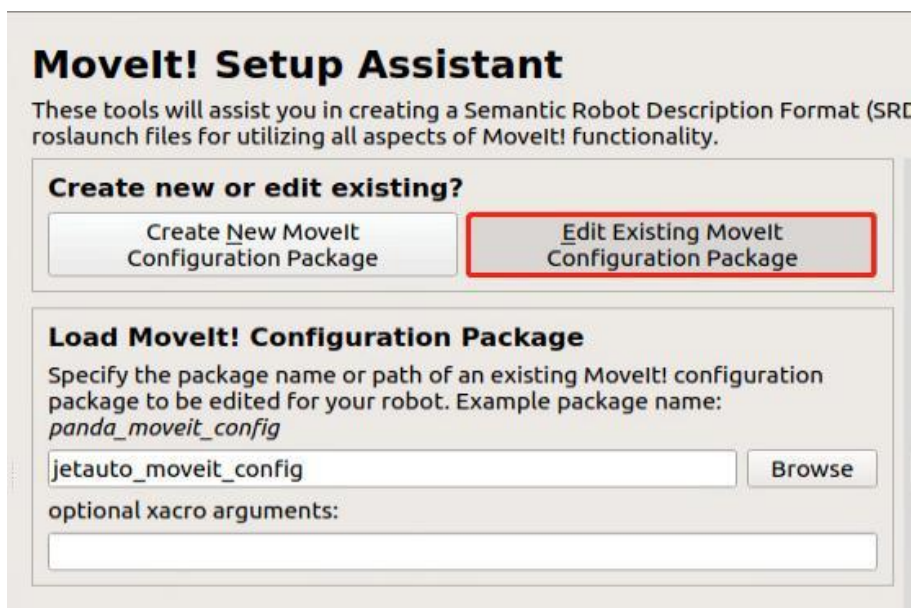
MoveIt is composed of manipulation function packs of ROS. By incorporating the latest advances in [motion planning](#), [manipulation](#), [3D perception](#), [kinematics](#), [collision detection](#), MoveIt is state of the art software for mobile manipulation

It is a user-friendly platform used to **develop advanced robot applications** and **assess new robot design**. Besides, it is an open-source software widely applied in industrial, commercial, R&D and other fields.

In addition, MoveIt provides **series of mature plugins and tools** to **achieve rapid robotic arm control configuration**. Large amount of API are encapsulated facilitating your secondary development and creative application development.

## 2. Run Configuration Program

- 1) Start JetAuto, and connect it to NoMachine.
- 2) Open terminal.
- 3) Input command “**sudo systemctl stop start\_app\_node.service**” and press Enter to stop app service.
- 4) Input command “**roslaunch jetauto\_moveit\_config setup\_assistant.launch**” to open MoveIt setup assistant.
- 5) Click “**Edit Existing MoveIt Configuration Package**” to edit the existed configuration pack



- 6) Click “**Browse > home/jetautp\_ws/src/jetauto\_simulations/ > jetauto\_moveit\_config > open**” as pictured.

## Movelt! Setup Assistant

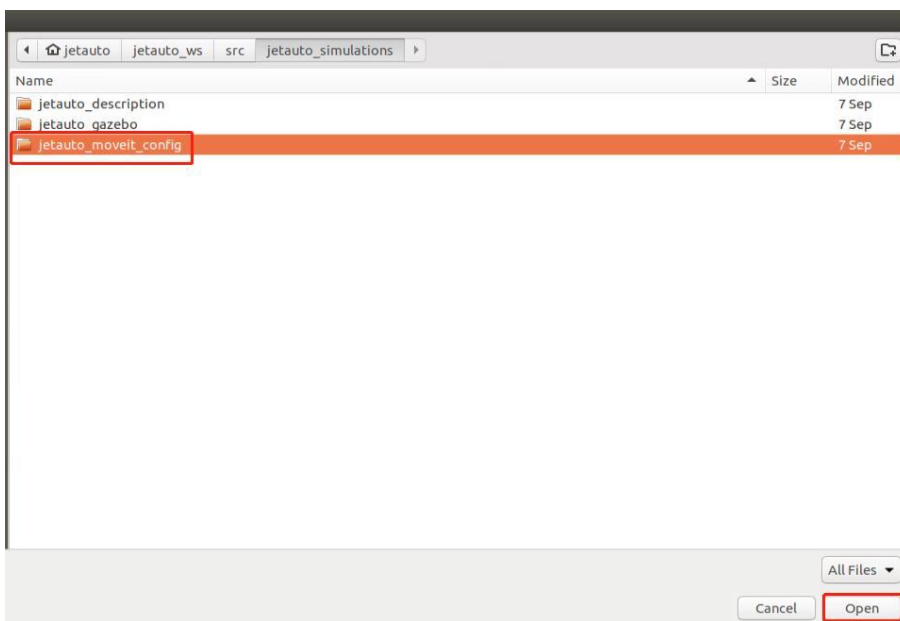
These tools will assist you in creating a Semantic Robot Description Format (SRDF) roslaunch files for utilizing all aspects of Movelt! functionality.

**Create new or edit existing?**

**Load Movelt! Configuration Package**

Specify the package name or path of an existing Movelt! configuration package to be edited for your robot. Example package name: *panda\_moveit\_config*

optional xacro arguments:



7) Click “**Load Files**” and wait for the file to finish loading.

## Movelt! Setup Assistant

These tools will assist you in creating a Semantic Robot Description Format (SRDF) file, various yaml configuration and many roslaunch files for utilizing all aspects of Movelt! functionality.

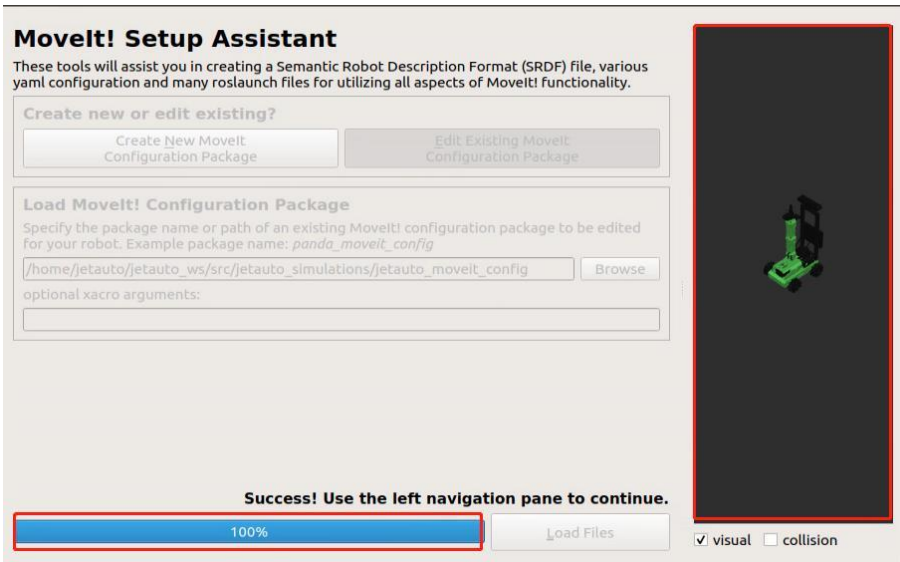
**Create new or edit existing?**

**Load Movelt! Configuration Package**

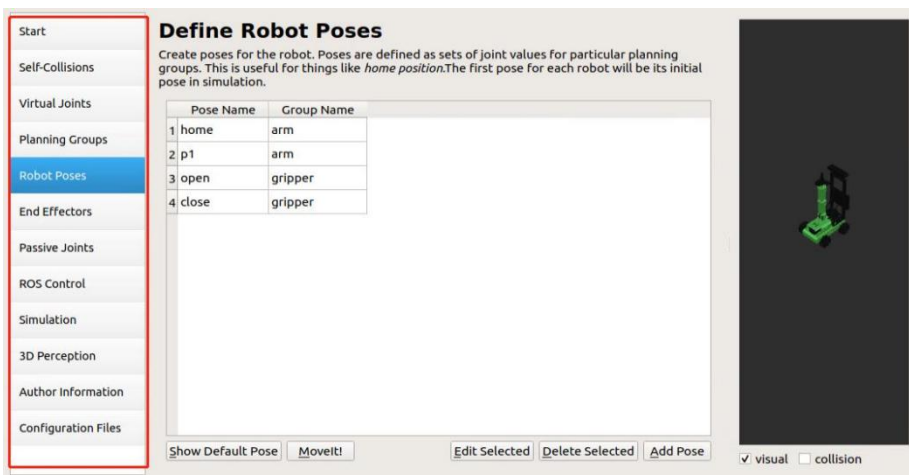
Specify the package name or path of an existing Movelt! configuration package to be edited for your robot. Example package name: *panda\_moveit\_config*

optional xacro arguments:

When it is loaded to 100%, the robot model will appear, which means that the file is loaded successfully.



8) Configure the contents of “**Virtual joints**” and “**Robot pose**” at left side.

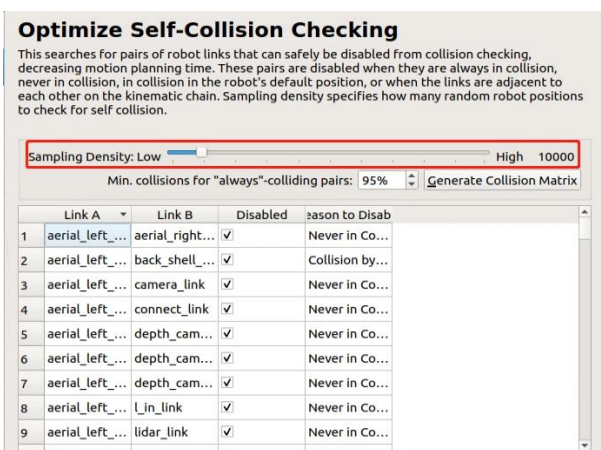


## 3. Configuration Content

### ◆Self-Collisions

**Generate custom collision matrix.** Default collision matrix generator will search all robot joints, and it can stop detection safely to reduce time spent on motion planning.

**Sampling density** refers to how many joints are randomly sampled to detect collision. Higher density requires longer computation time, and default sampling density is 10000 collision detection.



## ◆Virtual joints

Create virtual joints to connect robot to simulation program. A virtual robot joint is defined between base\_footprint and world\_frame in this step.

### Define Virtual Joints

Create a virtual joint between a robot link and an external frame of reference (considered fixed with respect to the robot).

	Virtual Joint Name	Child Link	Parent Frame	Type
1	world	base_footprint	world_frame	fixed

## ◆Planning Groups

Create 'joint model' group which is used to describe different joints composing the robot.

### Define Planning Groups

Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for. Note: when adding a link to the group, its parent joint is added too and vice versa.

Current Groups	
▼ arm	
▼ Joints	
joint1 - Revolute	
joint2 - Revolute	
joint3 - Revolute	
joint4 - Revolute	
end_effector_joint - Fixed	
Links	
Chain	
Subgroups	
▼ gripper	
▼ Joints	
l_in_joint - Revolute	
l_out_joint - Revolute	
l_joint - Revolute	
r_in_joint - Revolute	
r_out_joint - Revolute	
r_joint - Revolute	
Links	
Chain	
Subgroups	

## ◆Robot Poses

Define robot poses, and customize robot pose name and 'joint model' groups required by this pose.

### Define Robot Poses

Create poses for the robot. Poses are defined as sets of joint values for particular planning groups. This is useful for things like *home position*. The first pose for each robot will be its initial pose in simulation.

	Pose Name	Group Name
1	home	arm
2	p1	arm
3	open	gripper
4	close	gripper

## ◆Passive Joints

Define different joints. Set the joints that can be used and cannot be used.

## Define Passive Joints

Specify the set of passive joints (not actuated). Joint state is not expected to be published for these joints.

### Active Joints

	Joint Names
1	joint1
2	joint2
3	joint3
4	joint4
5	l_in_joint
6	l_out_joint
7	l_joint
8	r_in_joint
9	r_out_joint
10	r_joint

### Passive Joints

Joint Names
-------------

## ◆ROS Control

Add analog controllers to joints via ROS Control panel, then you can use MoveIt to simulate the robotic arm motion.

## Setup ROS Controllers

Configure MoveIt! to work with ROS Control to control the robot's physical hardware

Auto Add FollowJointsTrajectory  
Controllers For Each Planning Group

Controller	Controller Type
▼ <b>gripper_controller</b>	<i>position_controllers/JointTrajectoryController</i>
▼ <i>joints</i>	
r_joint	
▼ <b>arm_controller</b>	<i>position_controllers/JointTrajectoryController</i>
▼ <i>joints</i>	
joint1	
joint2	
joint3	
joint4	
▼ <b>arm_controller</b>	<i>FollowJointTrajectory</i>
▼ <i>joints</i>	
joint1	
joint2	
joint3	
joint4	
▼ <b>gripper_controller</b>	<i>FollowJointTrajectory</i>
▼ <i>joints</i>	
r_joint	

## ◆Simulation

Set Gazebo simulation and URDF files required by simulation.

## Simulate With Gazebo

The following tool will auto-generate the URDF changes needed for Gazebo compatibility with ROSControl and MoveIt!. The needed changes are shown in green.

You can run the following command to quickly find the necessary URDF file to edit:

roscd jetauto\_description

Generate URDF



## ◆Author Information

### Add maintainer information

**Specify Author Information**

Input contact information of the author and initial maintainer of the generated package. catkin requires valid details in the package's package.xml

Name of the maintainer this MoveIt! configuration:

Email of the maintainer of this MoveIt! configuration:

## ◆Configuration Files

**Generate configuration files.** After selecting the save path, click “Generate Package”.

**Generate Configuration Files**

Create or update the configuration files package needed to run your robot with MoveIt. Uncheck files to disable them from being generated - this is useful if you have made custom changes to them. Files in orange have been automatically detected as changed.

**Configuration Package Save Path**

Specify the desired directory for the MoveIt! configuration package to be generated. Overwriting an existing configuration package directory is acceptable. Example: `/u/robot/ros/nanda_moveit_config`

Files to be generated: (checked)

<input type="checkbox"/>	package.xml	Defines a ROS package
<input type="checkbox"/>	CMakeLists.txt	
<input checked="" type="checkbox"/>	config/	
<input type="checkbox"/>	config/jetauto.srdf	
<input type="checkbox"/>	config/ompl_planning.yaml	
<input type="checkbox"/>	config/chomp_planning.yaml	
<input type="checkbox"/>	config/kinematics.yaml	
<input type="checkbox"/>	config/joint_limits.yaml	
<input type="checkbox"/>	config/cartesian_limits.yaml	
<input type="checkbox"/>	config/fake_controllers.yaml	
<input type="checkbox"/>	config/ros_controllers.yaml	
<input type="checkbox"/>	config/sensors_3d.yaml	
<input checked="" type="checkbox"/>	launch/	

0%

# Part 2 MoveIt Usage Precaution and Control

## 1. Usage Precaution

Place the robot in a wide area to **ensure enough motion space**. Keep a certain distance from the robot to avoid hurt when robot is working. **Bend the antenna slightly backward to avoid robotic arm from hitting the antenna.**

In following operations, MoveIt is used to **plan the path** so that **control simulation model** and **real robot to move to specific position**.

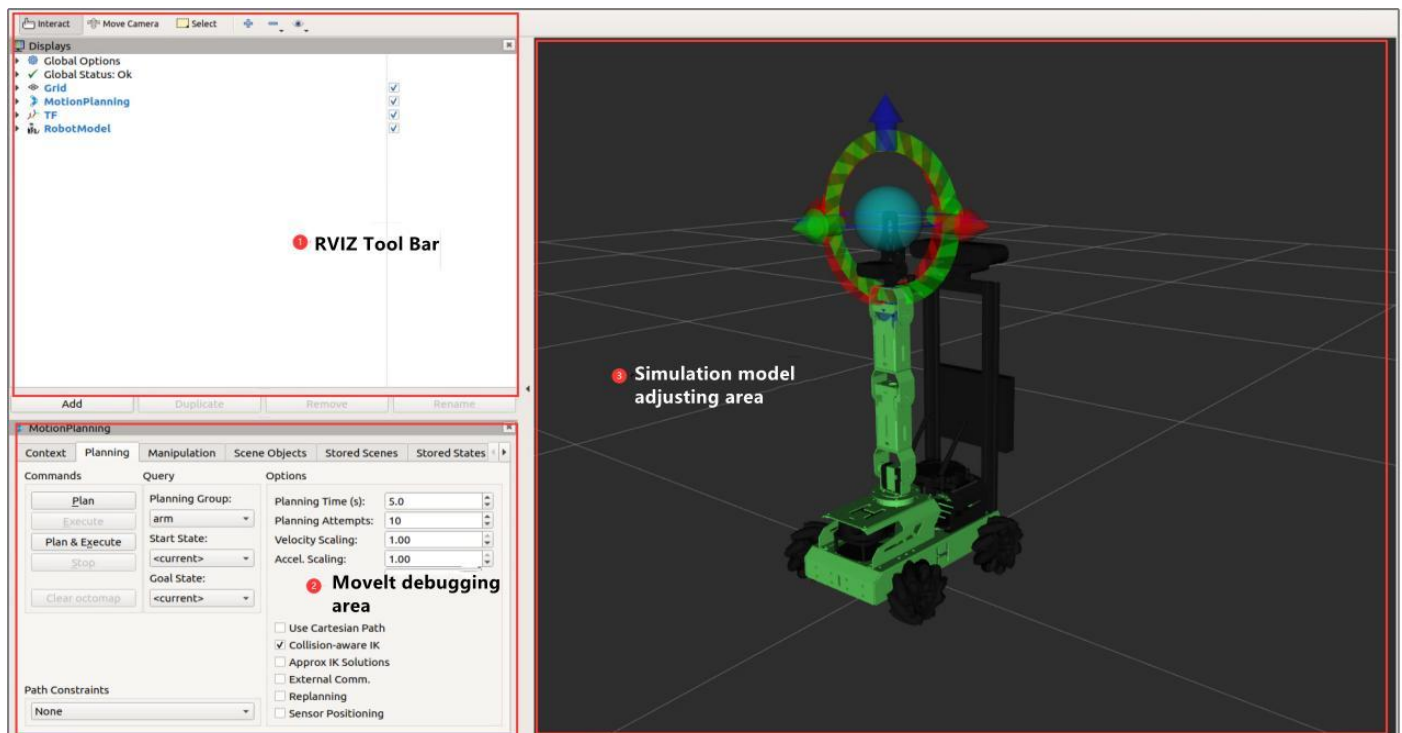
## 2. Enable Related Service

- 1) Start JetAuto, and connect it to NoMachine.
- 2) Open terminal.
- 3) Input command “**sudo systemctl stop start\_app\_node.service**” and press Enter to stop app service.

4) Input command “**roslaunch hiwonder\_servo\_controllers start.launch**” to enable servo control service.

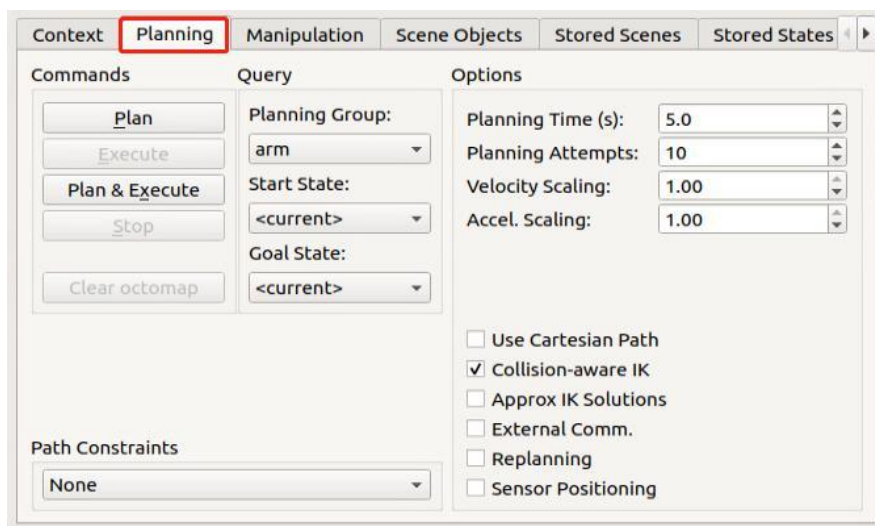
5) Open a new terminal, and input command “**roslaunch jetauto\_moveit\_config demo.launch fake\_execution:=false**” to enable MoveIt control service.

The program interface is as pictured.



### 3. Control Instruction

1) Find and click “**Planning**” in MoveIt debugging area.

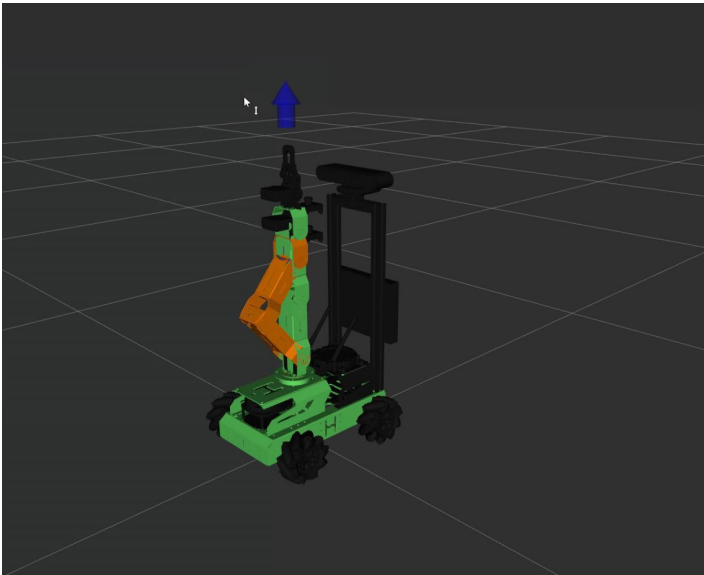


2) **The arrows** in simulation model adjusting area **can be dragged to adjust robotic arm pose**.

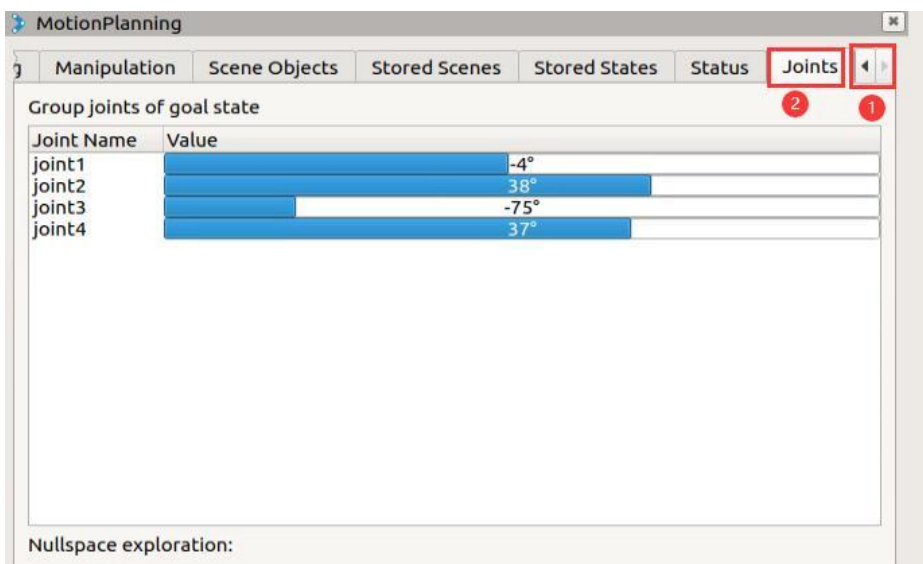
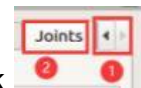
With robot as the first person view, **red arrow is used to control Y-axis movement**, and positive Y-axis direction corresponds to robot left.

**Green arrow controls X-axis movement**, and positive X-axis direction corresponds to robot front.

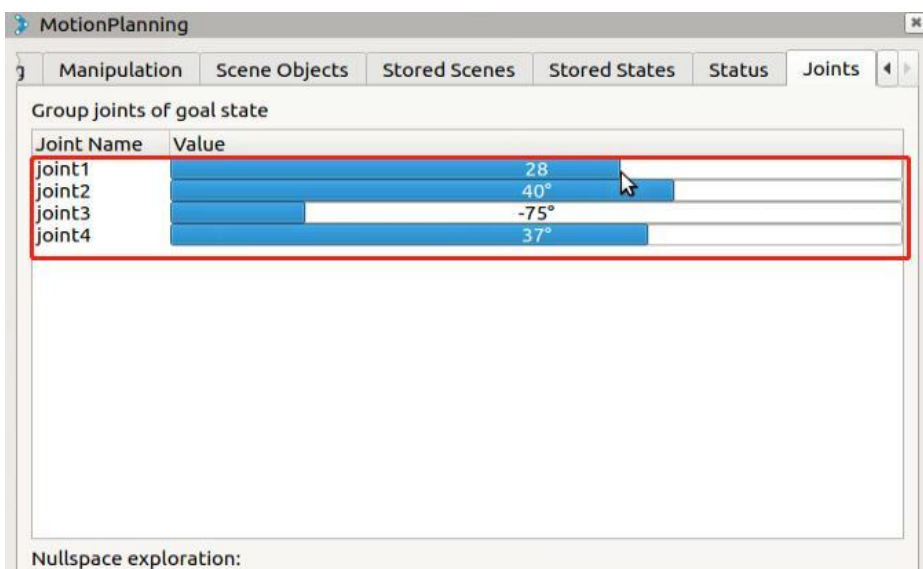
Blue arrow controls Z-axis movement, and positive Z-axis direction corresponds to robot top.



3) Besides using arrows to adjust pose, you can directly adjust individual joint. Click



4) Drag slider to adjust joint angle.

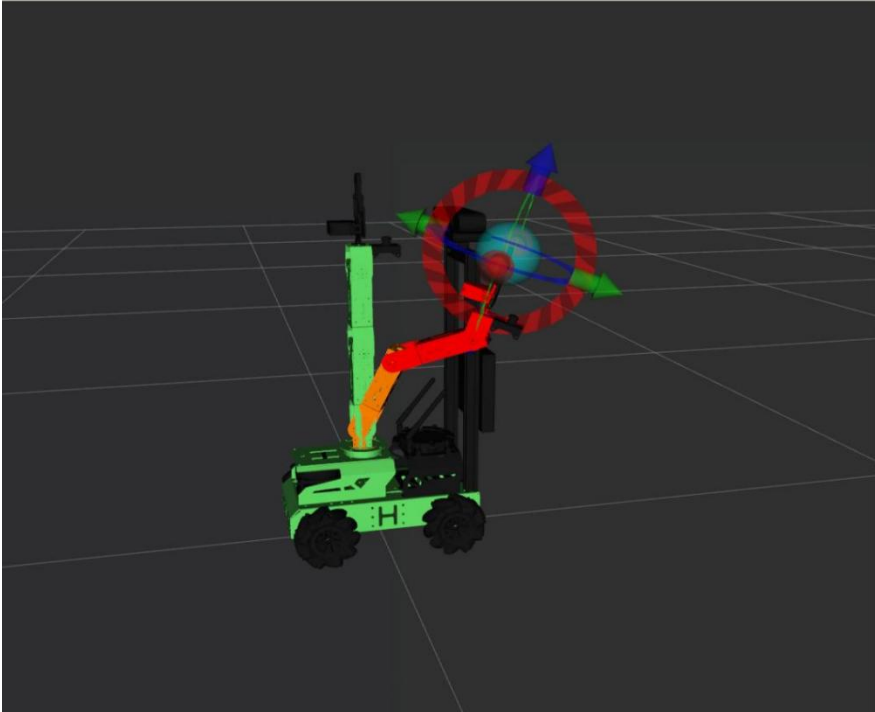




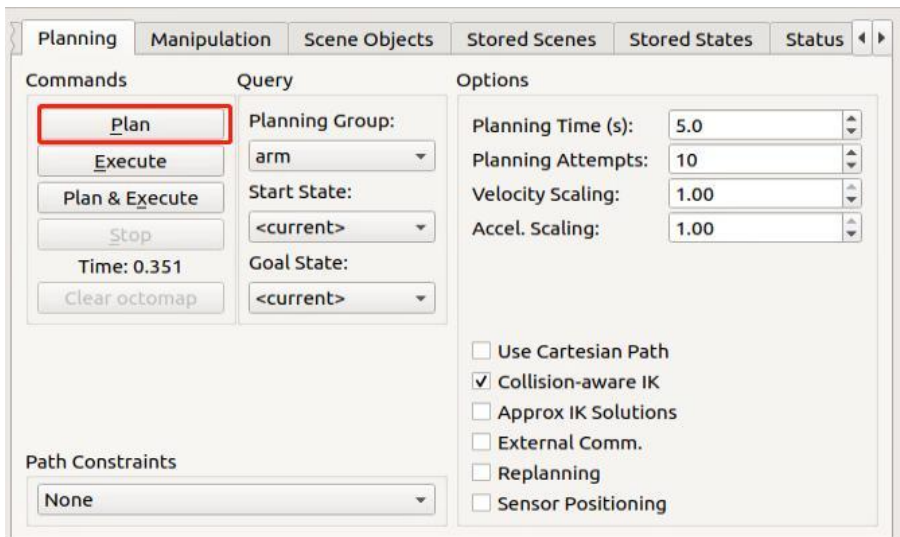
5) After robotic arm motion path is planned successfully, **new position is marked in orange** as pictured.



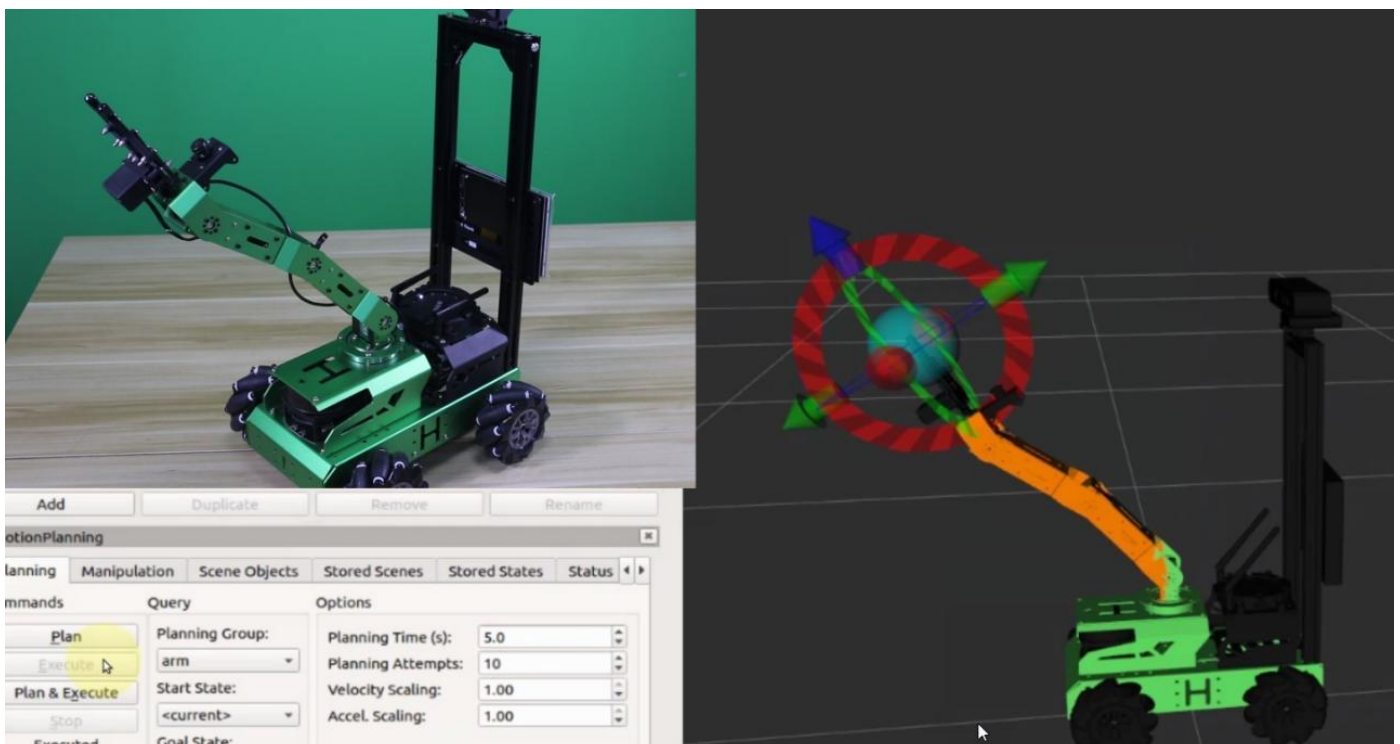
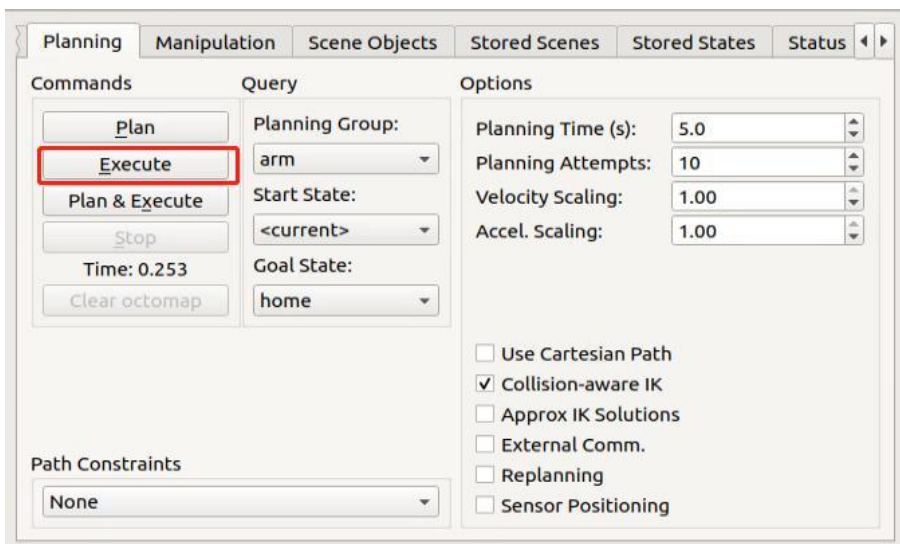
If robotic arm hit other stuffs, like depth camera, **it will be marked in red**. You need to adjust the robotic arm pose again until there is no collision, otherwise robotic arm will not perform this action.



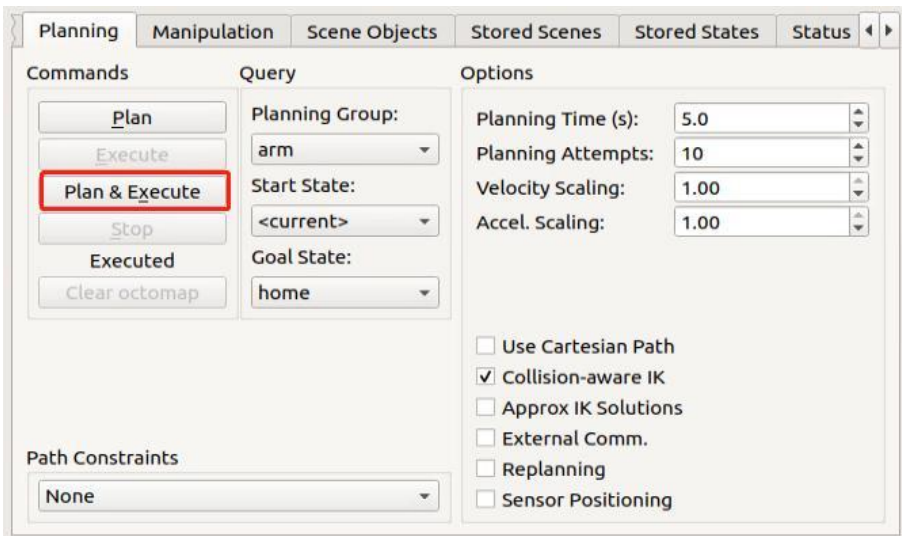
6) Having planned the path, return back to “**Planning**” panel. Click “**Plan**”, then **the animation that simulated robotic arm model moves to the specific position will be displayed**.



7) Click “**Execute**” to **let both the simulation model and robotic arm execute the planned movement**. Note: robotic arm recover vertical pose first, then execute the planned movement.



8) Click “**Plan & Execute**”. Real robotic arm will take first to execute the action, then the planned motion path is demonstrated on model adjusting area.



## Part 3 MoveIt Random Motion

### 1. Usage Precaution

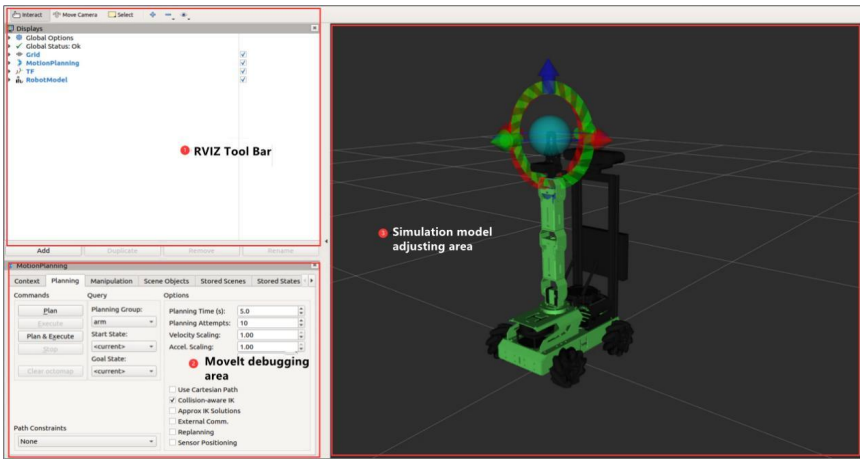
Place the robot in a wide area to ensure enough motion space. Keep a certain distance from the robot to avoid hurt when robot is working. Bend the antenna slightly backward to avoid robotic arm from hitting the antenna.

In following operations, MoveIt is used to plan the path so that control simulation model and real robot to move to random position.

### 2. Enable Related Service

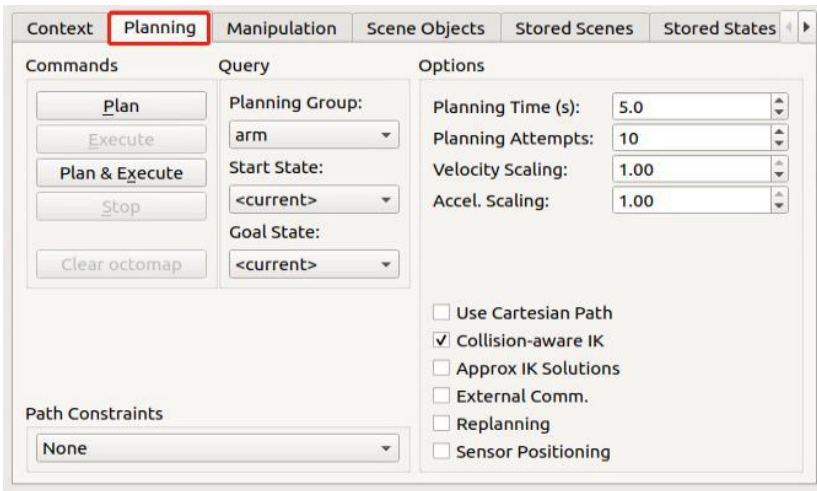
- 1) Start JetAuto, and connect it to NoMachine.
- 2) Open terminal.
- 3) Input command “**sudo systemctl stop start\_app\_node.service**” and press Enter to stop app service.
- 4) Open a new terminal, and input command “**roslaunch jetauto\_moveit\_config demo.launch fake\_execution:=false**” to enable MoveIt control service.

The program interface is as pictured.



### 3. Random Motion

1) Find and click “Planning” in Movelt debugging area.



2) The arrows in simulation model adjusting area can be dragged to adjust robotic arm pose.

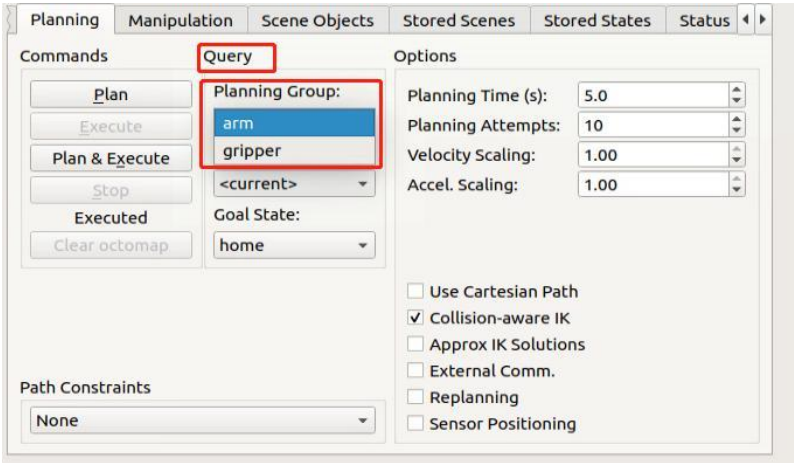
With robot as the first person view, red arrow is used to control Y-axis movement, and positive Y-axis direction corresponds to robot left.

Green arrow controls X-axis movement, and positive X-axis direction corresponds to robot front.

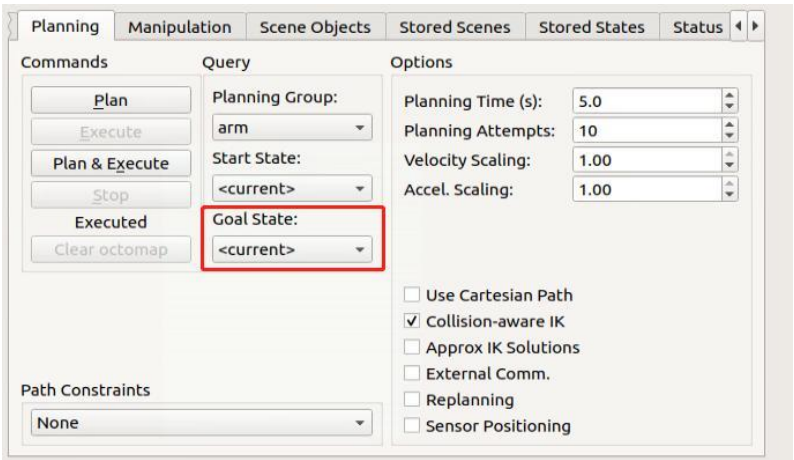
Blue arrow controls Z-axis movement, and positive Z-axis direction corresponds to robot top.



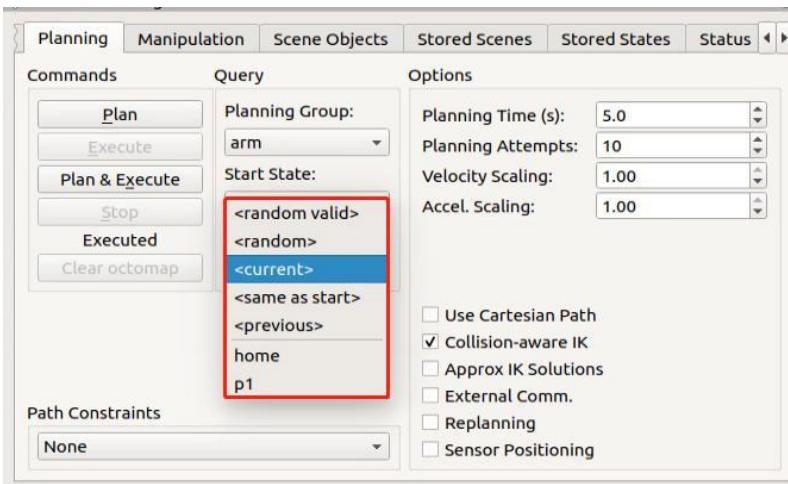
3) Find “Query”. Click “Planning Group” and select joint model group, for example arm.



4) Unfold drop-down manual of “Goal State”, and select target position.







The meaning of these parameters is as follow.

**random valid:** valid random position. No collision will happen.

**random:** random position. Collision may occur.

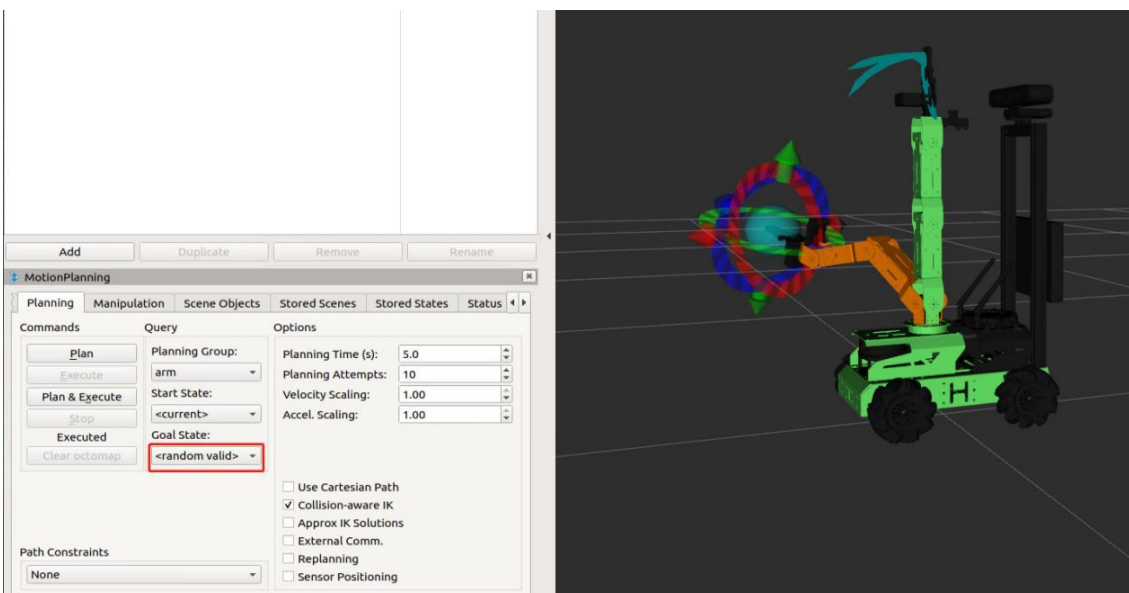
**current:** current position

**same as start:** position that is same as the start

**previous:** previous target position

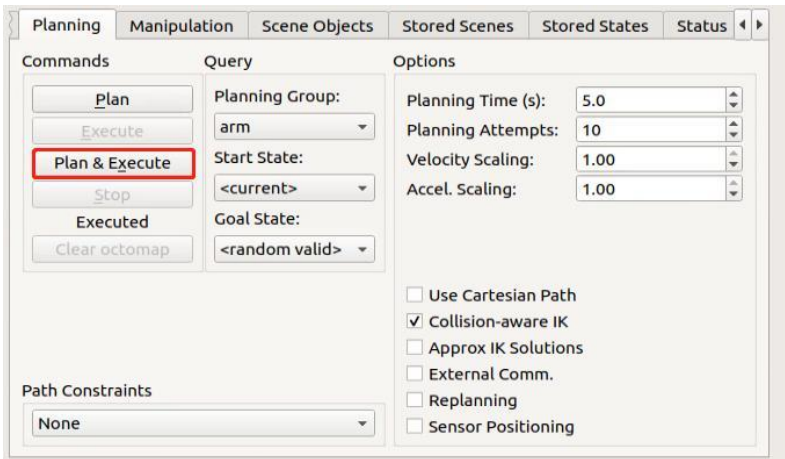
**home/ p1:** default position set by the program

5) To eliminate collision, select “**random valid**” to pick a target position randomly, and this target position will displayed on the simulation model.



6) Click “**Plan & Execute**” to let both the simulation model and real robotic arm execute action at the same time. Simulation model demonstrates the planned motion path, and real robotic arm execute the action.

**Warm tips:** the software will react more slowly than the robotic arm, which is normal.



# Part 4 Part 4 MoveIt Kinematic

## 1. Kinematic Introduction

**Kinematics** is a sub field of physics, developed in classical mechanics, that **describes and researches the motion of objects without considering the forces that cause them to move.** **Forward** and **inverse kinematics** are two computation method of robot motion.

**Forward kinematics** refers to **process of obtaining position and velocity of end effector, given the known joint angles and angular velocities.**

**Inverse kinematics** refers to **process of obtaining robot joint angles according to the given position and pose of end effector that is calculate backward the angle that servos need to rotate based on the final position and pose of robot.**

## 2. Forward Kinematic Analysis

### 2.1 DH Parameter Introduction

The **DH parameter** is a mechanical arm mathematical model and coordinate system determination system that uses **four parameters** to express the **position and angle relationship between two pairs of joint links.** As we will see below, it artificially **reduces two degrees of freedom by limiting the position of the origin and the direction of the X axis**, so it only needs four parameters to define a coordinate system with **six degrees of freedom.**

The four parameters selected by DH have very clear physical meanings, as follows:

- 1) **link length**( $a_{i-1}, a_i$ ) : The length of the common normal between the axes of the two joints (Rotation axis of rotation joint, translation axis of translation joint)
- 2) **link twist**( $\alpha_{i-1}$ ): The angle at which the axis of one joint rotates around their common normal relative to the axis of the other joint
- 3) **link offset**( $d_i$ ): The common normal of one joint and the next joint and the distance between the common normal of one joint and the previous joint along this joint axis
- 4) **joint angle**( $\theta_i$ ): The common normal of one joint and the next joint and the angle of rotation around the joint axis with the common normal of the previous joint

The above definition is very complicated, but it will be much clearer when combined with the coordinate system.

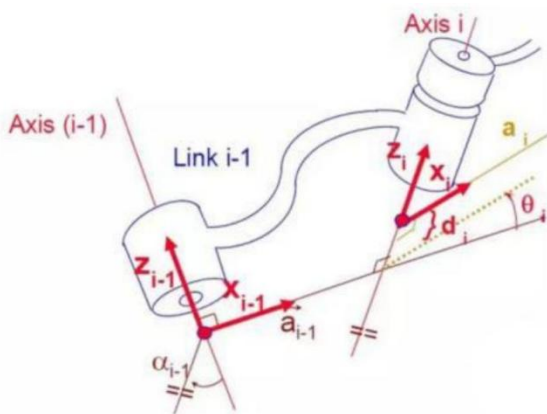
First of all you should pay attention to the two most important "lines": **the joint axis**, and **the common normal** between the axis joint and the adjacent joint.

In the DH parameter system, we **set axis as the z axis**; **common normal as the x axis**, and **the direction of the x axis is: from this joint to the next joint**.

Of course, these two rules alone are not enough to completely determine the coordinate system of each joint. Let's talk about the steps to determine the coordinate system in detail below.

In applications such as the simulation of the robotic arm, we often adopt other methods to establish the coordinate system, but mastering the methods mentioned here is necessary for you to understand the mathematical expression of the robotic arm and understand our subsequent analysis.

The figure below shows two typical robot joints. Although such joints and links are not necessarily similar to the joints and links of any actual robot, they are very common and can easily represent any joint of the actual robot.



### 3. Determine the Coordinate System

To determine the coordinate system, there are generally the following steps:

In order to **model the robot with DH notation**, the **first thing** is to **specify a local ground reference coordinate system for each joint**, so a **Z axis** and an **X axis** must be specified for each joint.

**Specify the Z axis.**

If the joint is rotating, the Z axis is in the direction of rotation according to the right-hand rule.

The rotation angle around the Z axis is a variable of the joint;

if the joint is a sliding joint, the Z axis is the direction of movement along a straight line.

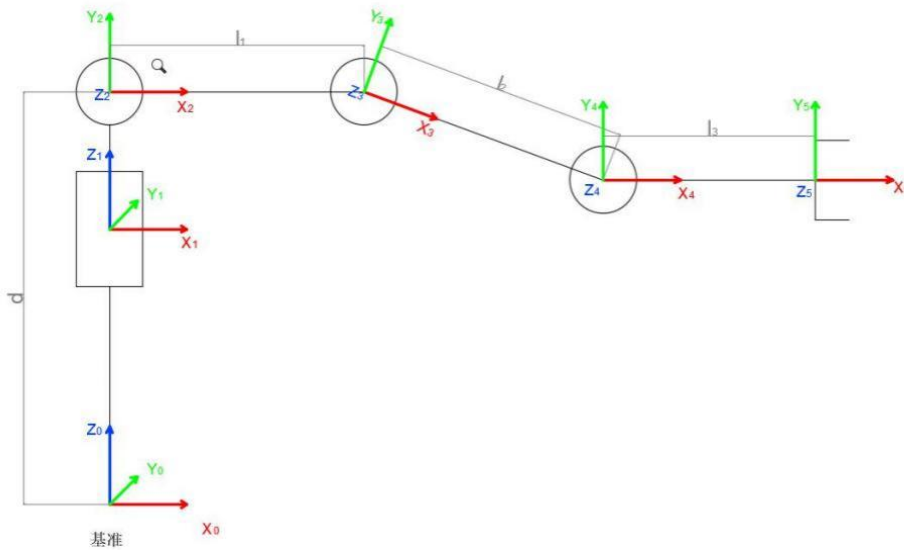
The link length  $d$  along the Z axis is the joint variable.

**Specify the X axis.** When the two joints are not parallel or intersect, the Z axis is usually a diagonal line, but there is always a common vertical line with the shortest distance, which is orthogonal to any two diagonal lines. Define the X axis of the local reference coordinate system in the direction of the common perpendicular. If  $a_n$  represents the common perpendicular between  $Z_{n-1}$ , the direction of  $X_n$  will be along  $a_n$ .

Of course there are special circumstances. When the Z axes of the two joints are parallel, there will be countless common perpendiculars. At this time, you can select the one that is collinear with the common perpendicular of the previous joint, which can simplify the model; if two joints intersect, there is no common perpendicular between them. In this case, the line perpendicular to

the plane formed by the two axes can be defined as X Shaft can simplify the model.

**After attaching the corresponding coordinate system to each joint, as shown in the following figure:**



After determining the coordinate system, we can express the above four parameters in a more concise way:

**link length  $\alpha_{i-1}$** : the distance from  $Z_{i-1}$  to  $Z_i$  along  $X_{i-1}$

**link twist  $\alpha_{i-1}$** : the angle of  $Z_i$  relative to  $Z_{i-1}$  to rotate around  $X_{i-1}$

**link offset  $d_i$** : the distance from  $X_{i-1}$  to  $X_i$  along  $Z_i$

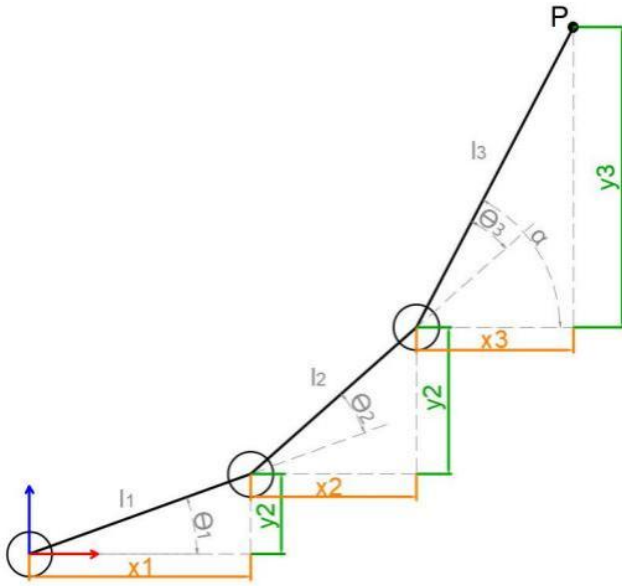
**joint angle  $\theta_i$** :  $X_i$  relative to  $X_{i-1}$  around  $Z_i$

Next we can write the DH parameter table of the robotic arm:

## 4. Inverse Kinematic Analysis

For the robot arm, the position and orientation of the gripper are given to obtain the rotation angle of each joint. The three-dimensional motion of the robotic arm is more complicated. In order to simplify the model, we remove the rotation joint of the station so that the kinematics analysis can be performed on a two-dimensional plane.

**Inverse kinematics analysis generally requires a large number of matrix operations**, and the process is complex and computationally expensive, so it is difficult to implement. In order to better meet our needs, we **use geometric methods to analyze the robotic arm**.



We simplify the model of the robotic arm, remove the base pan/tilt, and the actuator part to get the main body of the robotic arm. From the figure above, you can see **the coordinates (x, y) of the end point P of the robotic arm**, which ultimately **consists of three parts (x1+x2+x3, y1+y2+y3)**.

Among them **theta1, theta2, theta3** in the above figure are **the angles of the servo that we need to solve**, and **alpha is the angle between the paw and the horizontal plane**. From the figure, it is obvious that the top angle of the claw **alpha=theta1+theta2+theta3**, based on which we can formulate the following formula:

$$x = l_1 \cos \theta_1 + l_2 \cos (\theta_1 + \theta_2) + l_3 \cos (\theta_1 + \theta_2 + \theta_3)$$

$$y = l_1 \sin \theta_1 + l_2 \sin (\theta_1 + \theta_2) + l_3 \sin (\theta_1 + \theta_2 + \theta_3)$$

Among them, x and y are given by the user, and l1, l2, and l3 are the inherent properties of the mechanical structure of the robotic arm.

In order to facilitate the calculation, we will deal with the known part and consider the whole:

$$m = l_3 \cos(\alpha) - x$$

$$n = l_3 \sin(\alpha) - y$$

Substituting m and n into the existing equation, and then simplifying can get:

$$l_2 = (l_1 \cos \theta_1 + m)^2 + (l_1 \sin \theta_1 + n)^2$$

Through calculation:

$$\sin \theta_1 = \frac{(-b \pm \sqrt{b^2 - 4ac})}{2a}$$

We see that the above formula is the root-finding formula of a quadratic equation in one variable:



$$a = m^2 + n^2$$

$$b = \frac{-n(l_1^2 - l_2^2 - m^2 - n^2)}{l_2}$$

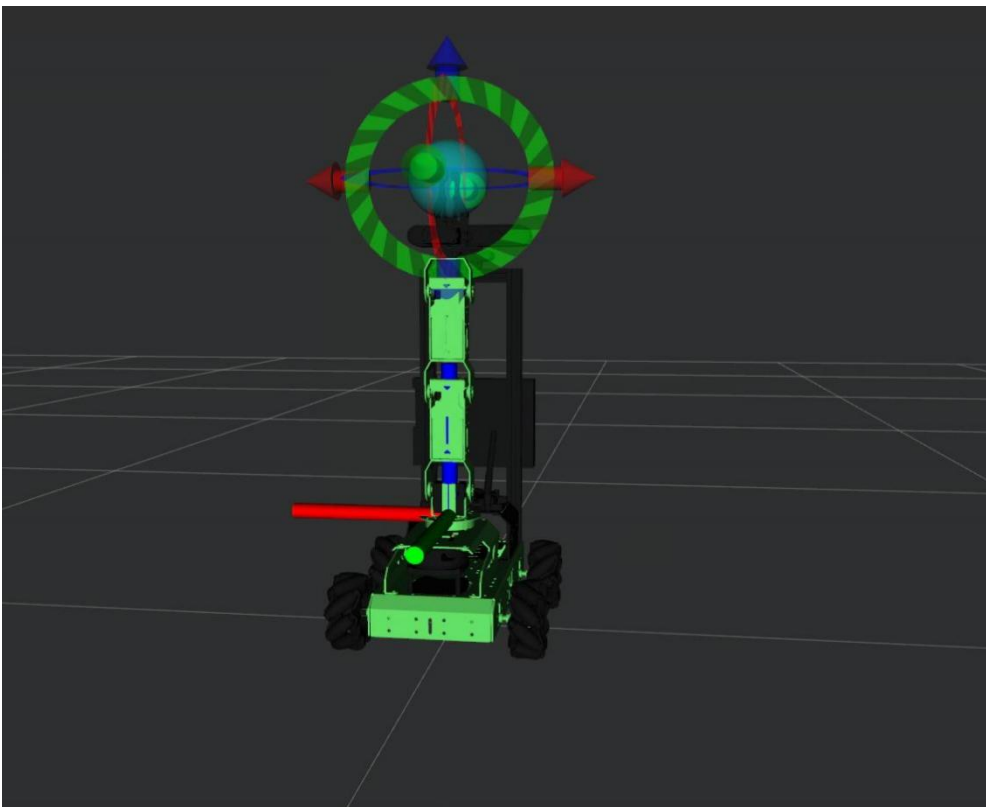
$$c = \left( \frac{l_1^2 - l_2^2 - m^2 - n^2}{2l_2} \right)^2$$

Based on this, **we can find the angle of  $\theta_1$** , and similarly we can also find  $\theta_2$ . From this we can obtain the angles of the three steering gears, and then control the steering gears according to the angles to realize the control of the coordinate position.

## 5. Robotic Arm Coordinate System Introduction

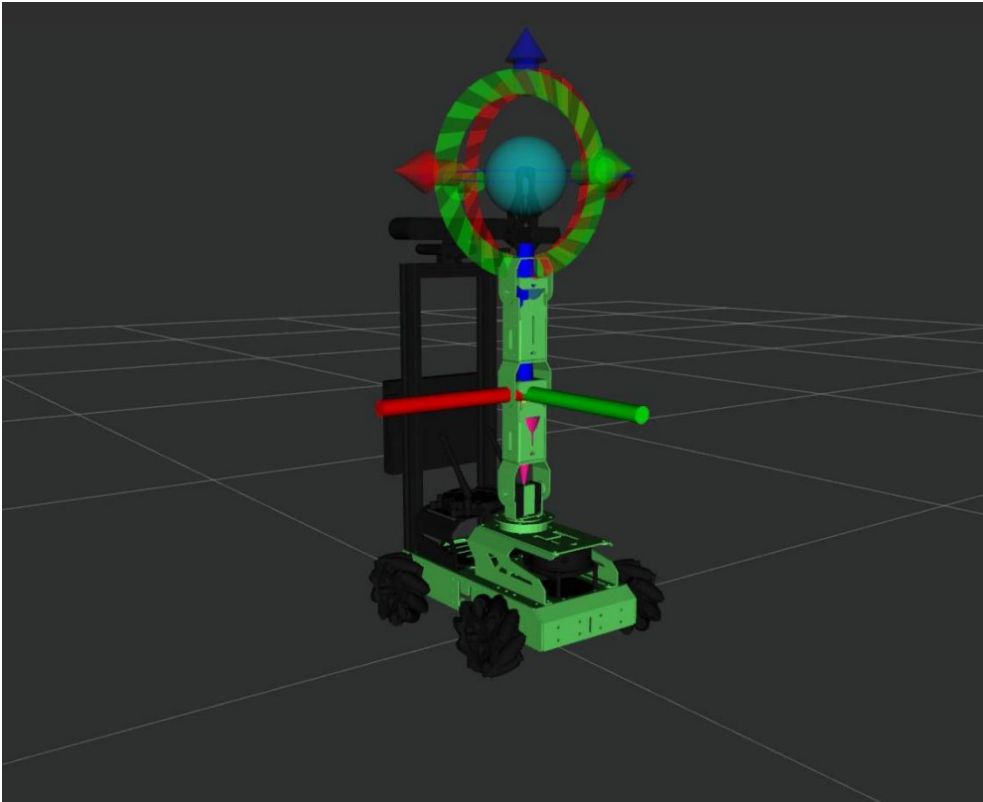
### ◆Base Coordinate System

**With bottom servo as origin, red axis is Y axis, green axis is Y axis, and blue axis is Z axis.**



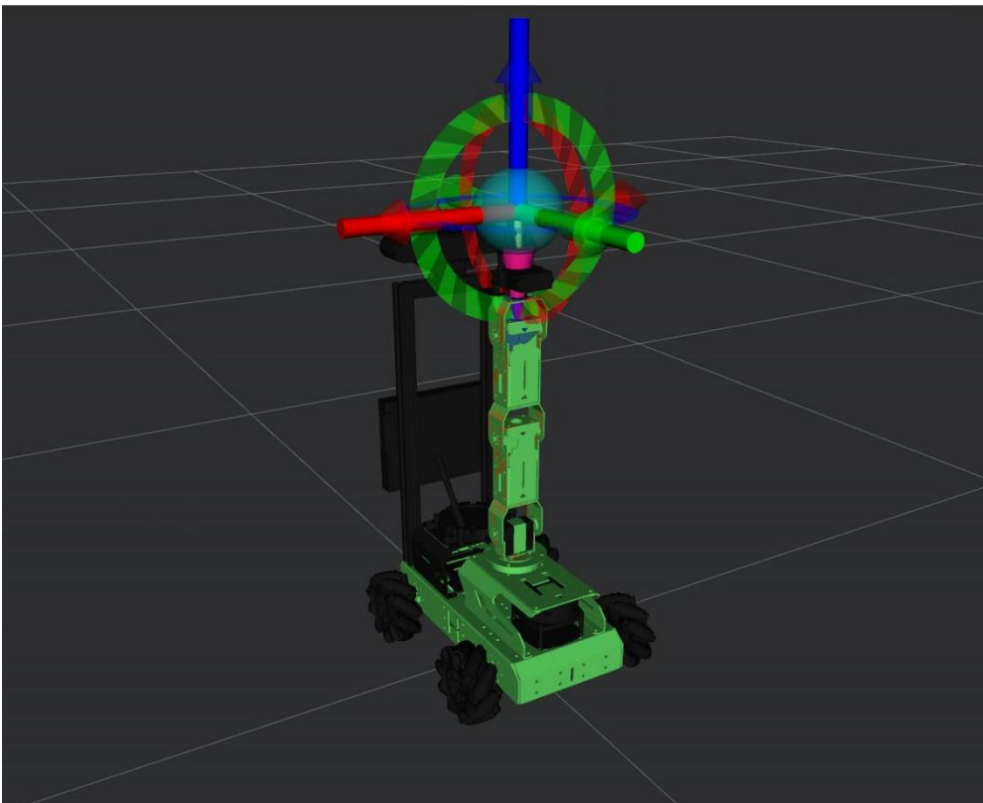
### ◆Joint Coordinate System

**Robotic arm is composed of joints and links. With pink cone as origin, red axis is Y axis, green axis is X axis, and blue axis is Z axis.**



#### ◆Tool Coordinate System

Tool is usually installed on the end of robotic arm to implement different tasks, and JetAuto is equipped with gripper. Gripper coordinate system takes pink cone as origin. Red axis is Y axis, green axis is X axis and blue axis is Z axis.



# Part 5 MoveIt Cartesian Path

## 1. Cartesian Path Introduction

### 1.1 Cartesian Coordinate System

**Cartesian coordinate system** refers to **rectangular coordinate system** and **oblique coordinate system**. Affine coordinate system on a plane is composed of two numerical axis that intersect at origin. If unit of measure on both numerical axis are equivalent, this affine coordinate system is defined as Cartesian Coordinate System. Cartesian coordinate system whose two numerical axis are perpendicular is rectangular coordinate system, otherwise oblique coordinate system.

Cartesian coordinate system is generally used to describe position, velocity and acceleration. Coordinates are usually right-handed. For right-handed coordinates the right thumb points along the z axis in the positive direction and the curl of the fingers represents a motion from the first or x axis to the second or y axis. When viewed from the top or z axis the system is counter-clockwise.



### 1.2 Cartesian Path Planning

Cartesian path planning can be divided into two types, namely point-to-point path planning and continuous path planning. It is a computation process where robot joint motion trajectory is computed with kinematic according to the given robot target point or path, so as to track the desired path.

### 1.3 MoveIt Cartesian Path Analysis

Each axis of robot can be controlled through joint space which is comprised of joint vectors. Each axis moves by interpolation, and they don't care how other axis move. The motion trajectory of robot axis end between two points is random curve.

In some special cases, the motion trajectory of robotic arm end is required to be straight line or arc. Therefore, Cartesian path planning is used to restrict the shape of motion trajectory.

**Note:** Place the robot in a wide area to ensure enough motion space. Keep a certain distance from the robot to avoid hurt when robot is working. Bend the antenna slightly backward to avoid robotic arm from hitting the antenna.

## 2. Operation Steps

Add Cartesian path planning to restrict motion of simulation model and real robotic arm.

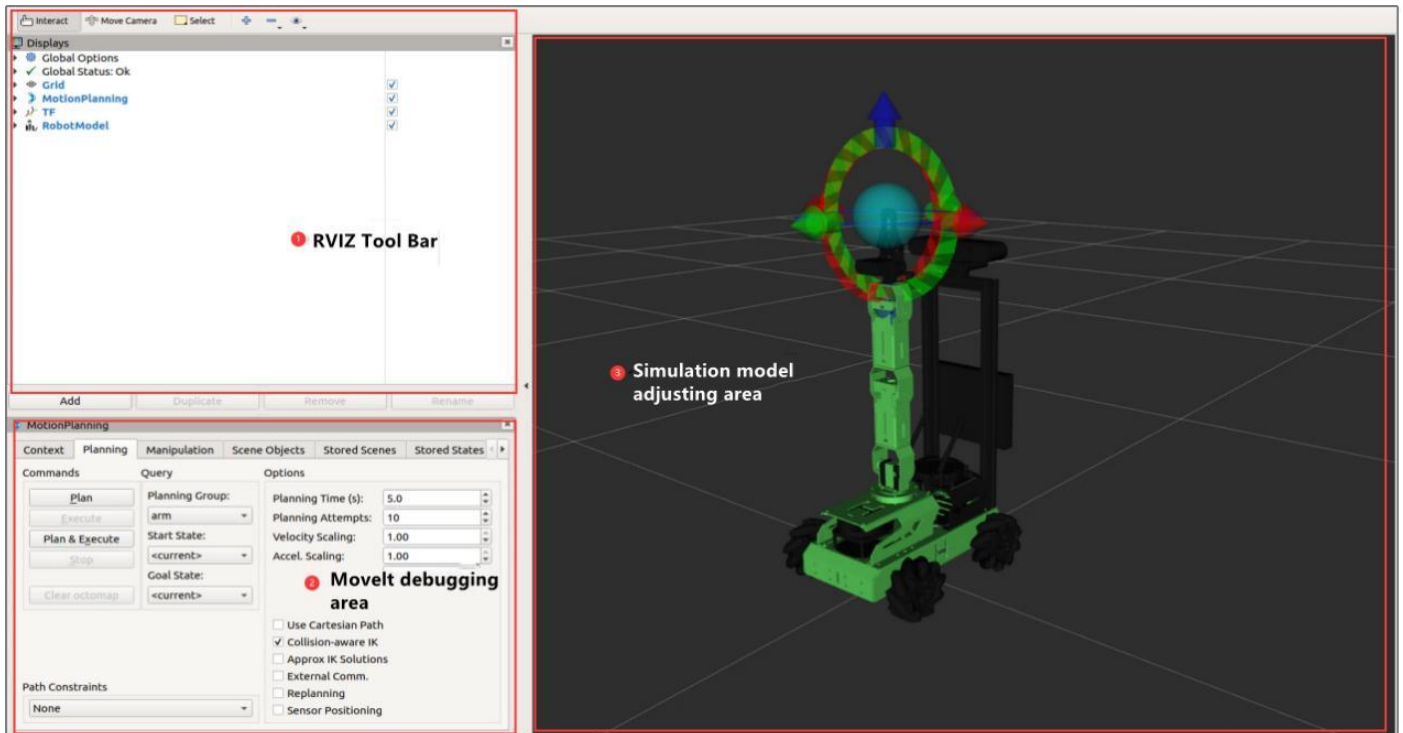
1) Start JetAuto, and connect it to NoMachine.

2) Open terminal.

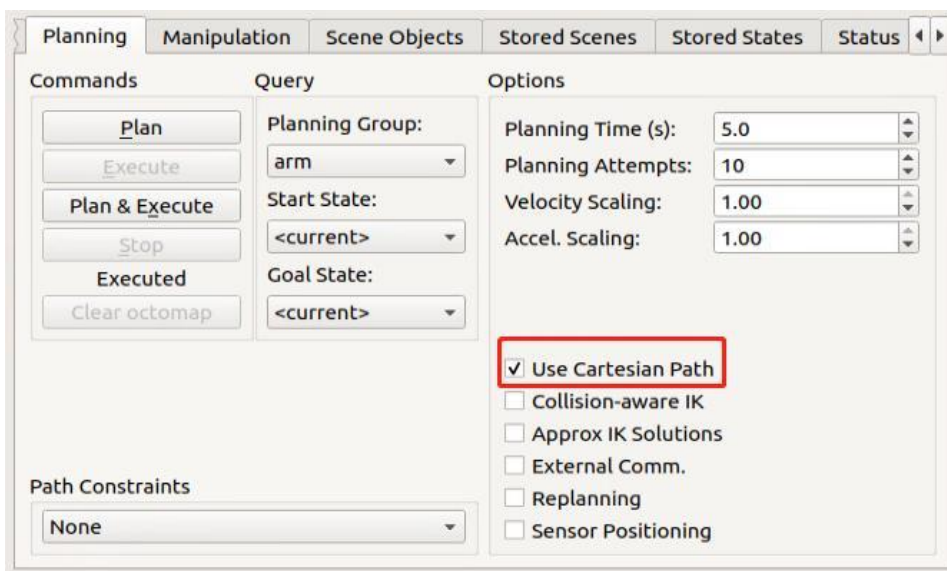
3) Input command “**sudo systemctl stop start\_app\_node.service**” and press Enter to stop app service.

4) Open a new terminal, and input command “**roslaunch jetauto\_moveit\_config demo.launch fake\_execution:=false**” to enable MoveIt control service.

The program interface is as pictured.



5) Click “**Planning > Use Cartesian Path**” to start Cartesian path planning.

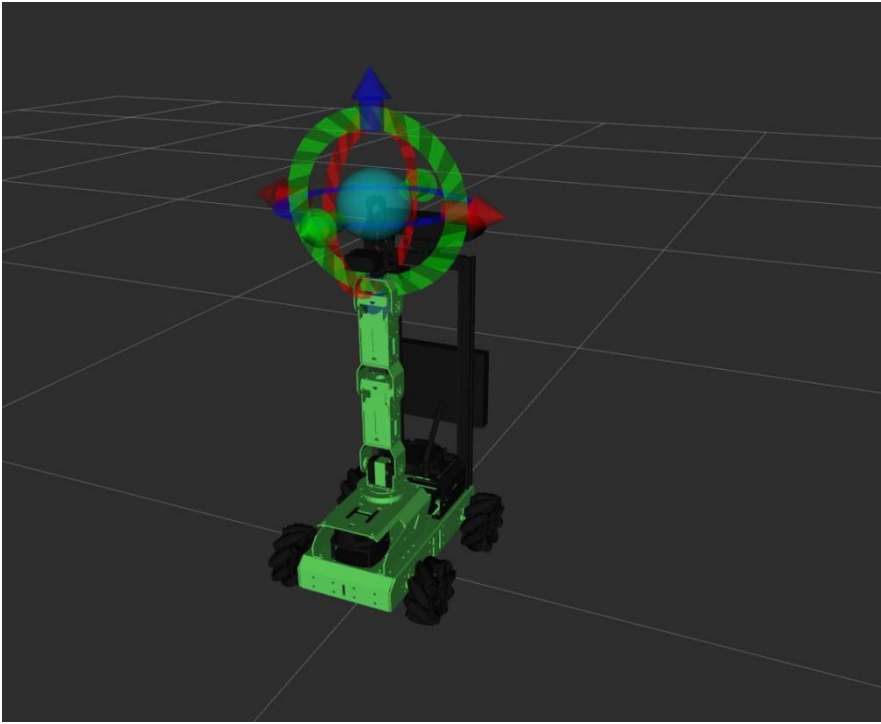


Then, drag arrows in “simulation model adjust area” to plan path for robotic arm.

With robot as the first person view, red arrow is used to control Y-axis movement, and positive Y-axis direction corresponds to robot left.

Green arrow controls X-axis movement, and positive X-axis direction corresponds to robot front.

Blue arrow controls Z-axis movement, and positive Z-axis direction corresponds to robot top.



6) Finishing planing, click “**Plan & Execute**”, then the simulation model will execute this action and linearly move its end effector in Cartesian space. If this action cannot be implemented resulting from Cartesian path restriction, this path planning ends in failure.

## Part 6 MoveIt Collision Detection

### 1. MoveIt Collision Detection Introduction

#### 1.1 Collision Detection Configuration

Collision detection in MoveIt2 is configured by CollisionWorld object in planning scene. Collision detection is mainly executed by FCL software pack which is a major CC library for MoveIt2.

#### 1.2 Collision Object

MoveIt2 welcomes different types of objects to detect collision, including:

- 1) Meshes
- 2) Basic shape: box, cylinder, cone, sphere and plane.
- 3) Octomap—can be directly used in collision detection

#### 1.3 Allowed Collision Matrix (ACM)

Allowed Collision Matrix (ACM) will encode a binary value to sign whether to detect collision between objects.

If values of these two objects are set as 1, collision detection will not be executed, otherwise collision detection will continue as schedule.



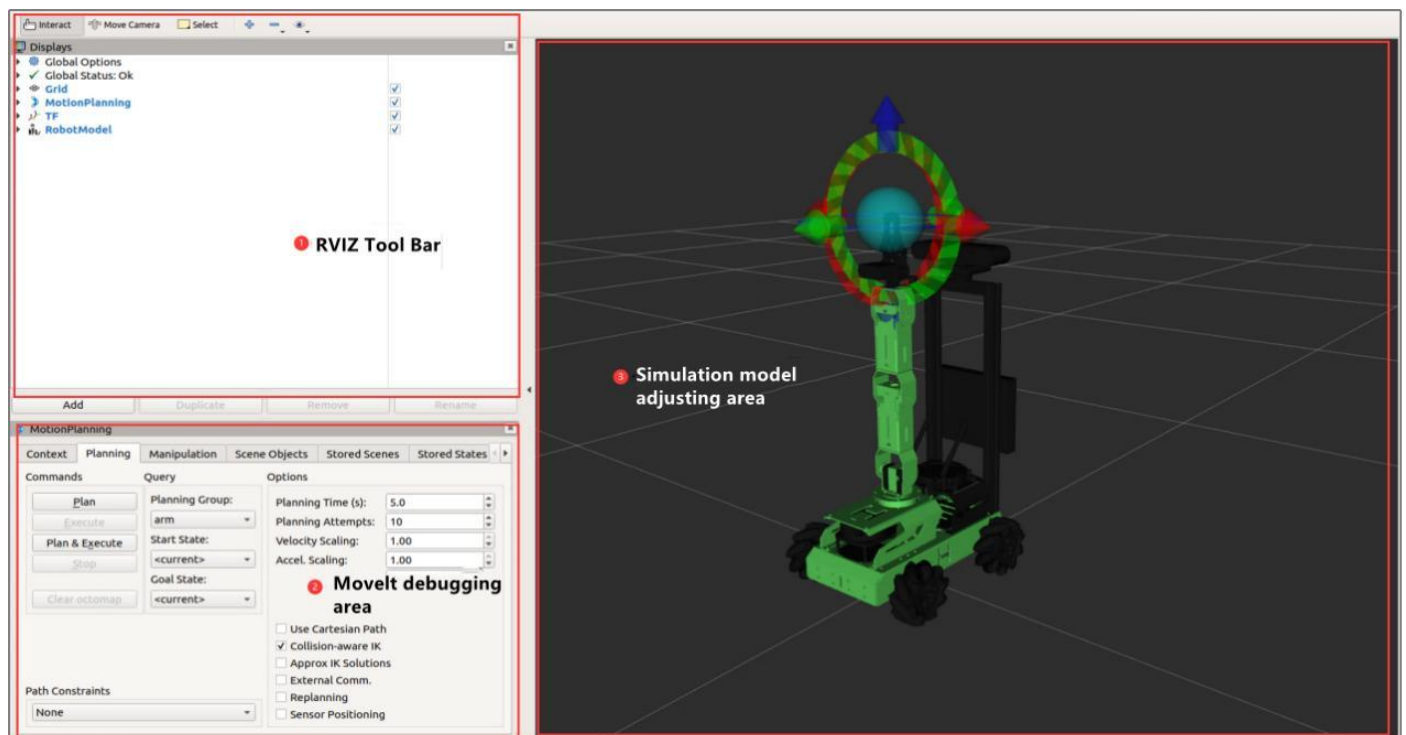
**Note:** Place the robot in a wide area to ensure enough motion space. Keep a certain distance from the robot to avoid hurt when robot is working. Bend the antenna slightly backward to avoid robotic arm from hitting the antenna.

## 2. Operation Steps

Add a collision model to demonstrate the collision detection effect of simulation model and real robotic arm.

- 1) Start JetAuto, and connect it to NoMachine.
- 2) Open terminal.
- 3) Input command “**sudo systemctl stop start\_app\_node.service**” and press Enter to stop app service.
- 4) Open a new terminal, and input command “**roslaunch jetauto\_moveit\_config demo.launch fake\_execution:=false**” to enable MoveIt control service.

The program interface is as pictured.



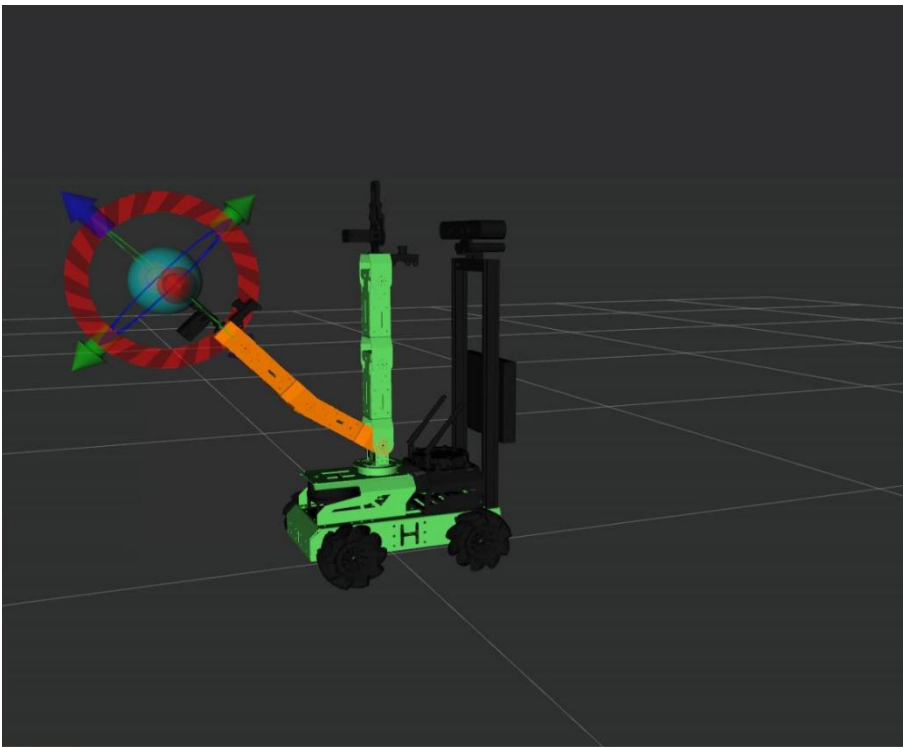
- 5) Then, drag arrows in “simulation model adjust area” to plan path for robotic arm.

With robot as the first person view, red arrow is used to control Y-axis movement, and positive Y-axis direction corresponds to robot left.

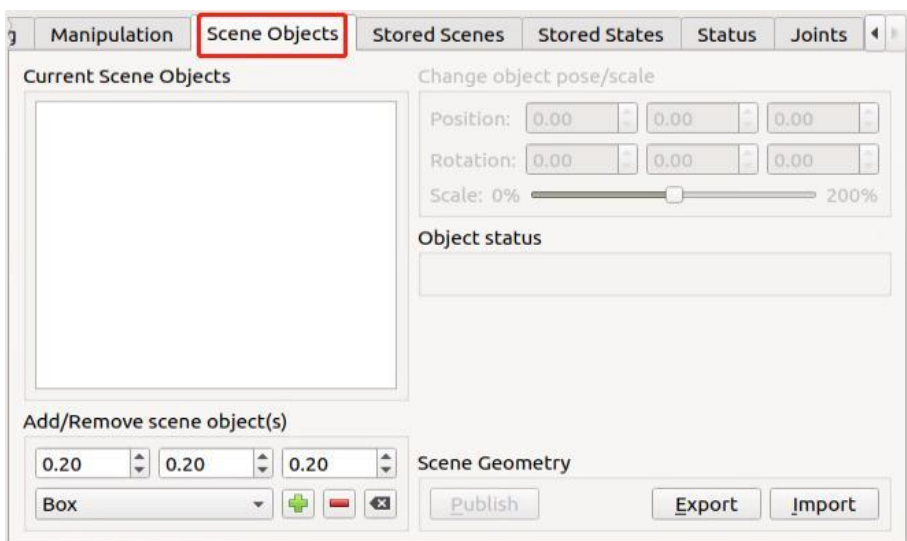
Green arrow controls X-axis movement, and positive X-axis direction corresponds to robot front.

Blue arrow controls Z-axis movement, and positive Z-axis direction corresponds to robot top.

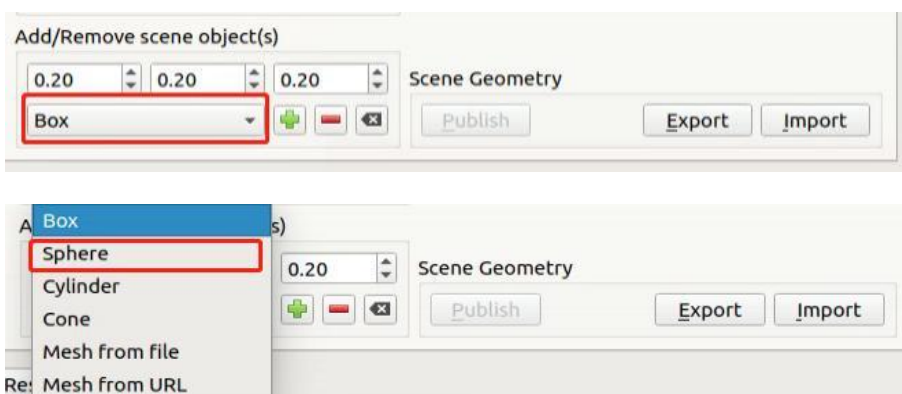
New position of robot is marked in orange as pictured.



6) After planning the path for robotic arm, click “**Scene Objects**” to add collision model.



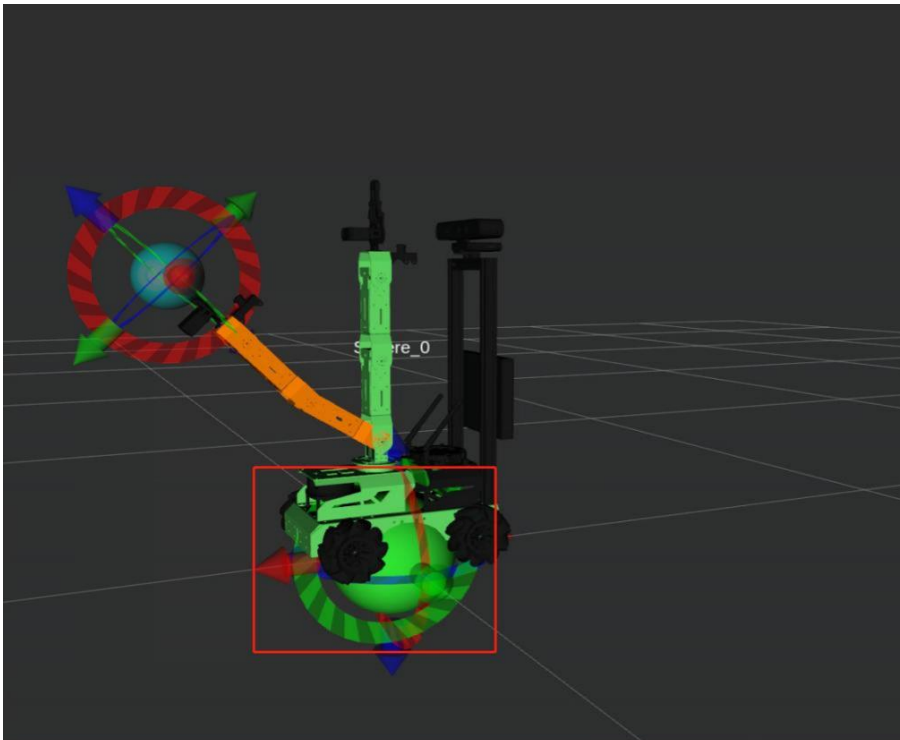
7) Unfold “**Box**” drop-down menu, and select one collision model, for example “**Sphere**”.



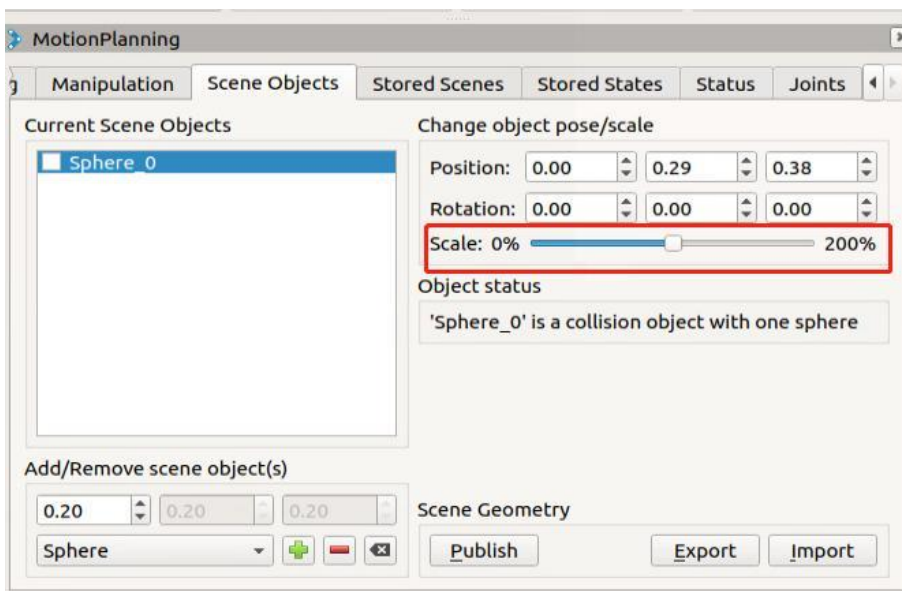
8) Click “+” to add this collision model



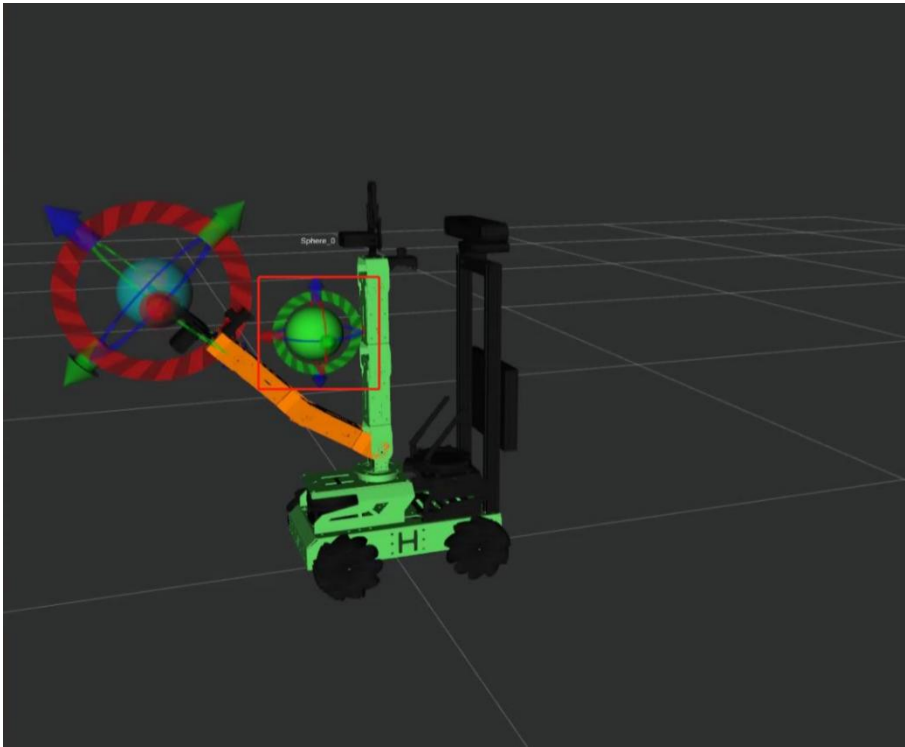
This model is generated at the bottom of the robot.



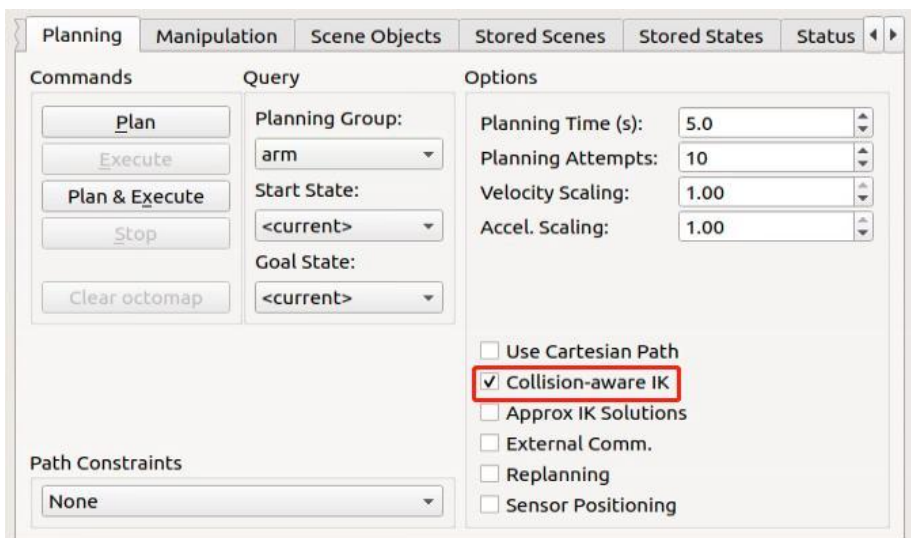
9) Drag the slider to change model size. 50% is recommended.



10) Drag 3D arrows of the sphere to move it between the initial position and target position.



11) Click **“Planning > Collision-aware IK”** to start model collision detection.



12) Click **“Plan & Execute”** to start planned motion. If the following dialog box pops up, click **“Yes”**.



13) After that, the robotic arm will execute the planned path while avoiding the collision model to prevent collision.

# Part 7 MoveIt Scene Design

## 1. Rviz Plugin Introduction

Rviz is visualization widget provided by ROS and also a plugin of MoveIt, which allows you to visualize external information and send message to control the monitored object.

With Rviz, you can set virtual scene, set robot's initial and target statuses, test motion planning algorithms and output 3D model.

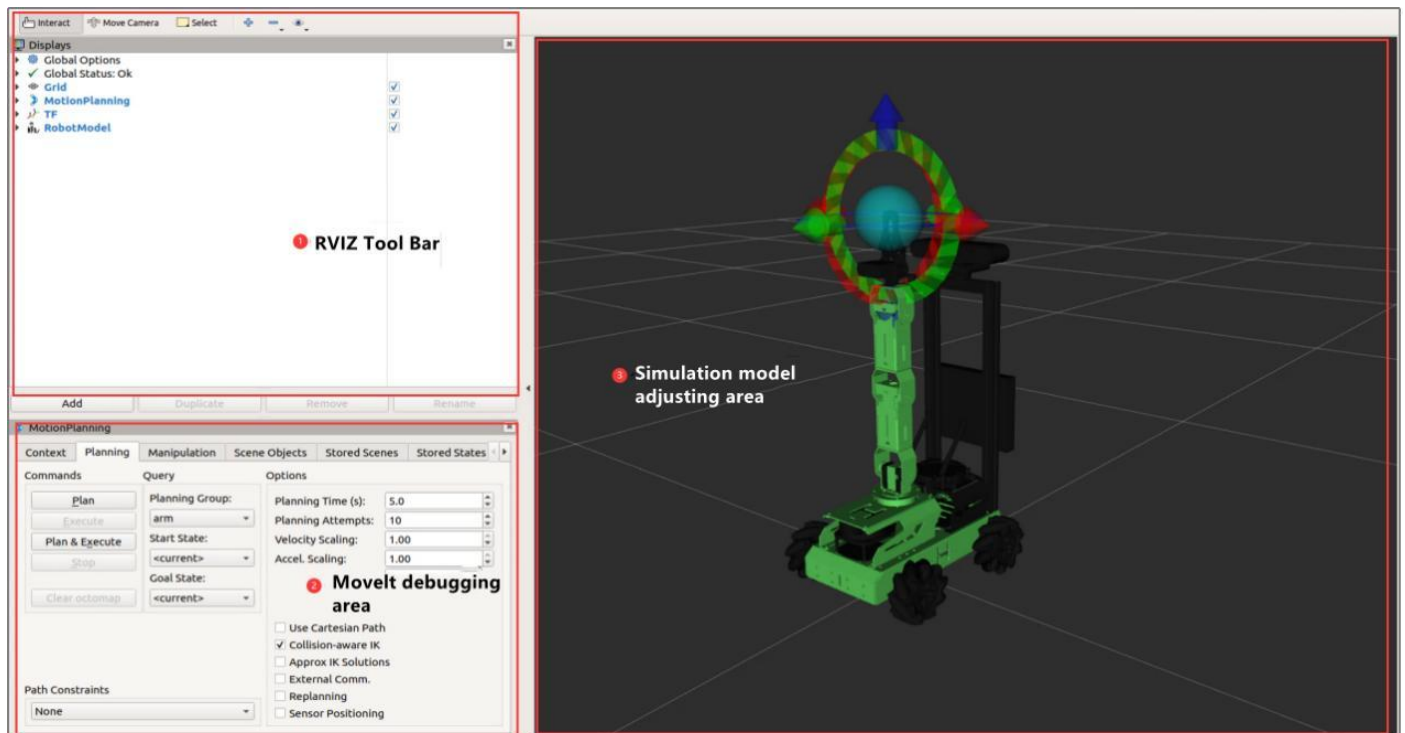
**Note:** Place the robot in a wide area to ensure enough motion space. Keep a certain distance from the robot to avoid hurt when robot is working. Bend the antenna slightly backward to avoid robotic arm from hitting the antenna.

## 2. Operation Steps

Add object model in simulation scene.

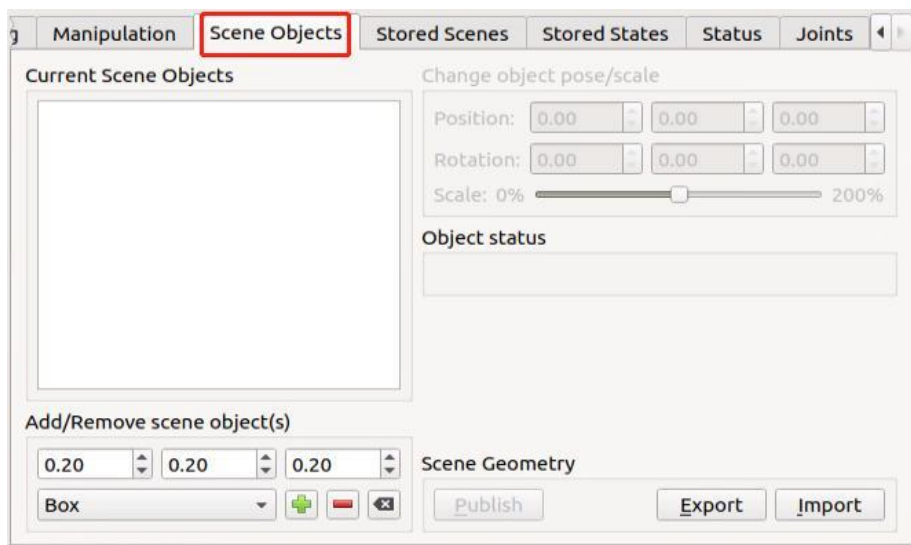
- 1) Start JetAuto, and connect it to NoMachine.
- 2) Open terminal.
- 3) Input command “**sudo systemctl stop start\_app\_node.service**” and press Enter to stop app service.
- 4) Open a new terminal, and input command “**roslaunch jetauto\_moveit\_config demo.launch fake\_execution:=false**” to enable MoveIt control service.

The program interface is as pictured.

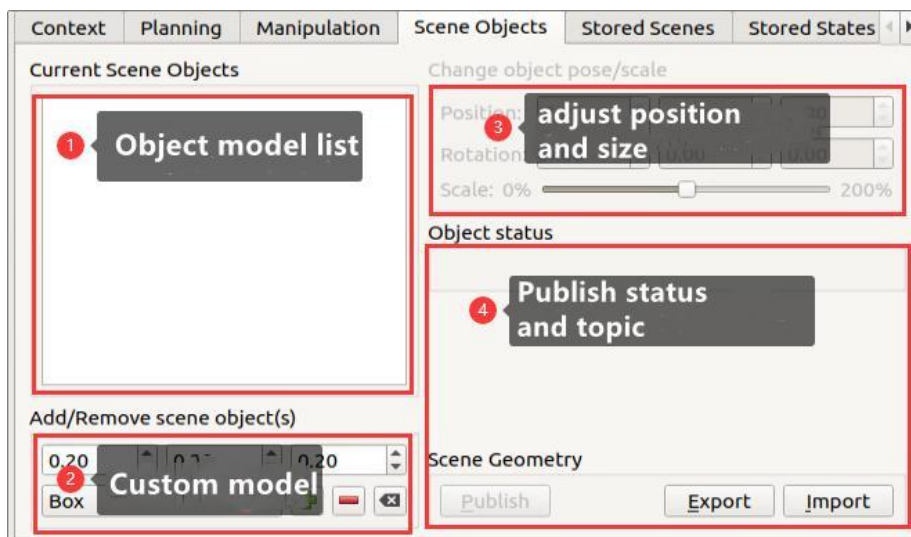




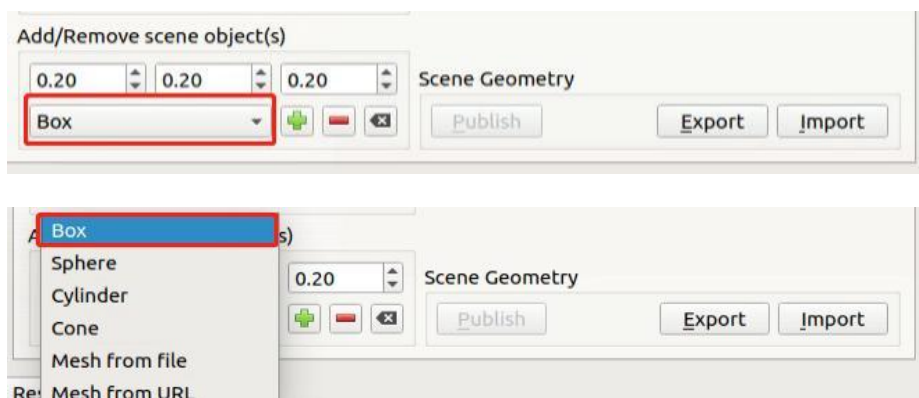
5) Find “**Scene Objects**” in **Movelt debugging area** to add scene object model.



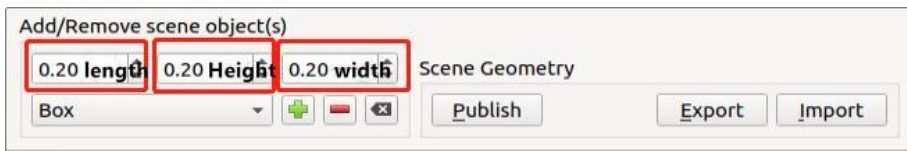
6) This section is divided into 4 parts as pictured



7) Add basic model, for example Box



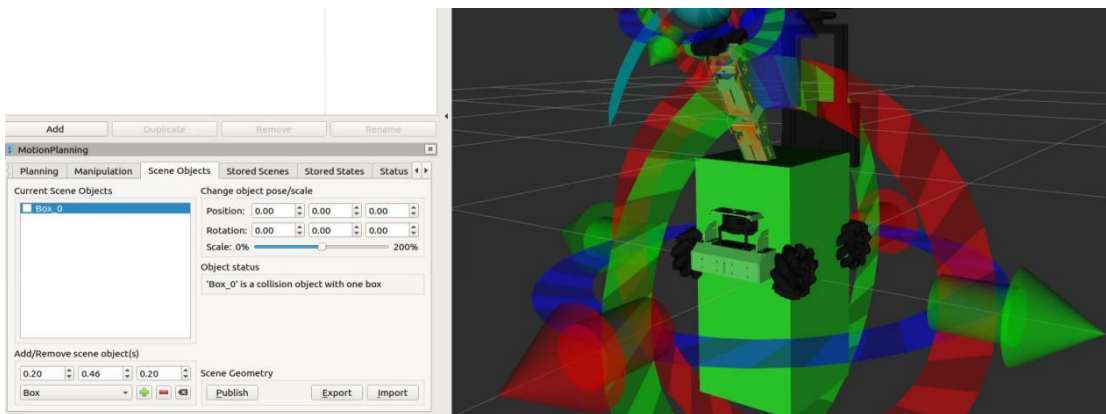
8) Adjust the size of object model in meter.



9) After adjustment, click “+” to add the selected object model to the scene.



10) The model list will be updated after you add the model, and the model is generated at the center of the scene i.e. center of robot



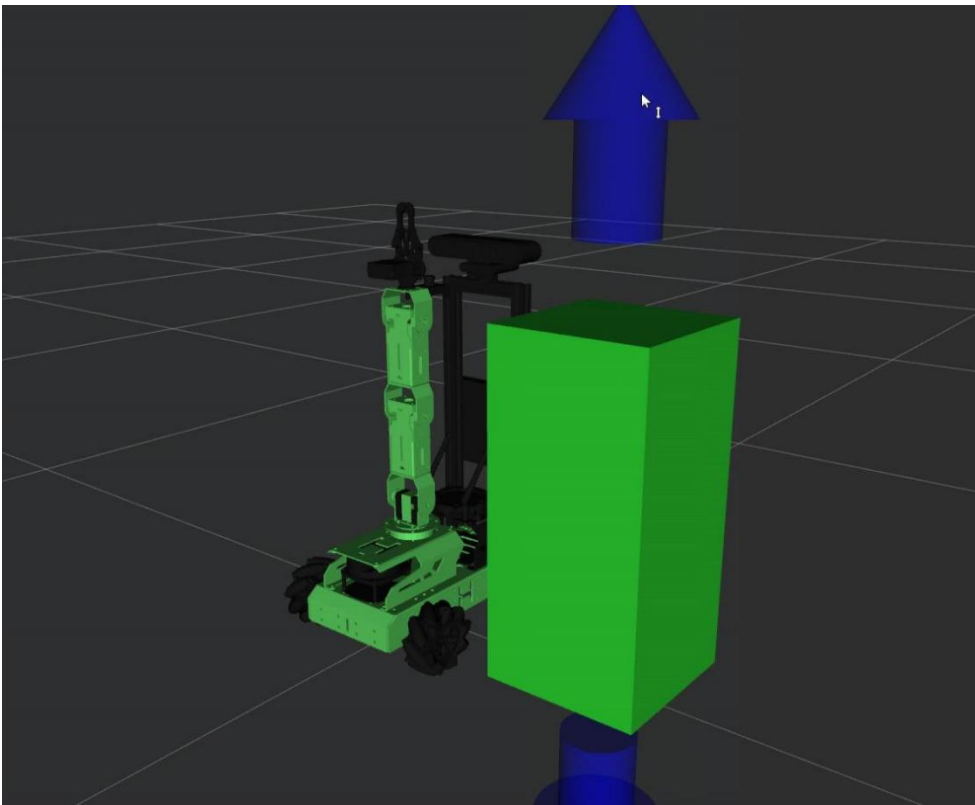
11) Drag arrows in “**simulation model adjust area**” to adjust robotic arm pose.

With robot as the first person view, red arrow is used to control Y-axis movement, and positive Y-axis direction corresponds to robot left.

Green arrow controls X-axis movement, and positive X-axis direction corresponds to robot front.

Blue arrow controls Z-axis movement, and positive Z-axis direction corresponds to robot top.

New position of robot is marked in orange as pictured



12) Besides using arrows to adjust pose, you can adjust its pose in the area as pictured.

Change object pose/scale

Position:	0.00	0.29	0.25
Rotation:	0.00	0.00	0.00
Scale:	0% <input type="range"/> 200%		

**"Position"** is to adjust position of object. The boxes from left to right are used to adjust X, Y and Z axis.

**"Rotation"** is to adjust object angle. The boxes from left to right are used to adjust X, Y and Z axis.

**"Scale"** is to adjust object size.

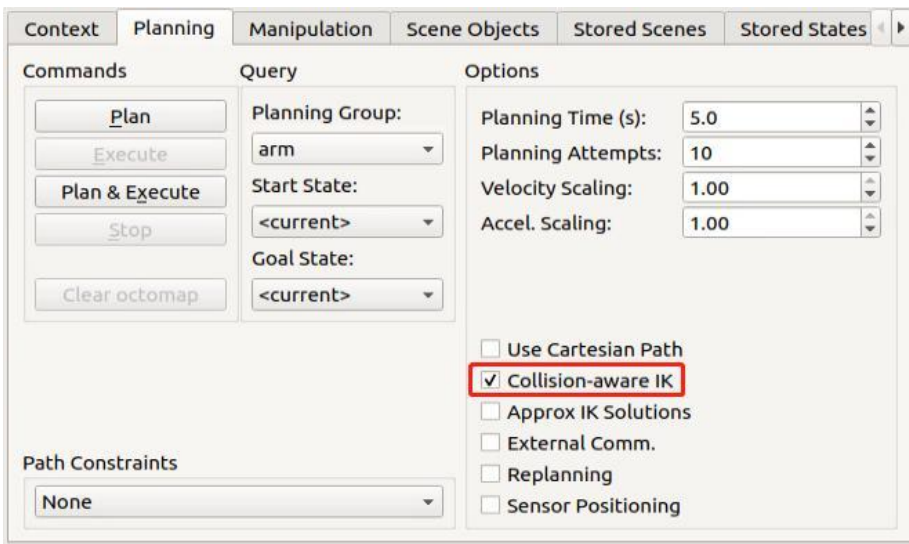
13) Click **"Publish"** after adjustment to publish topic of this model. MoveIt will subscribe to this topic automatically.

Add/Remove scene object(s)

0.20	0.46	0.20
Box	<input type="button" value="+"/>	<input type="button" value="-"/>

Scene Geometry

14) To protect object model from collision, tick **"Collision-aware IK"** to start model collision detection.



## Part 8 MoveIt Trajectory Planning

### 1. Motion Planner Introduction

#### 1.1 Open Motion Planner Library (OMPL)

OMPL is an open robotic motion planner library based on sampling method. Most of algorithms in this library derive from RRT and RPM, for example RRTStar and RRT-Connect.

By virtue of modular design, front-end GUI support and stable update, OMPL is the most mainstream motion planner software. It is default motion planner software of ROS.

A planner that is based on sampling need not take into account dimension of the planning object. That is to say no dimensional explosion will happen. A key reason why it can be used to control robotic arm is that it can effectively tackle path planning problems in high-dimensional space and complex constraints.

For motion planning of N DOF robotic arm, OMPL can plan a trajectory for end effector in robotic arm joint space and create M arrays (trajectory is composed of M control points). Dimension of each array is N (joint sequence of each control point). It can ensure robotic arm will not collide with surrounding obstacles when executing this trajectory.

#### 1.2 Industrial Motion Planner (Pilz)

Pilz industrial motion planner is a deterministic generator for circular and linear motions. In addition, it also supports integrating multiple motion segments with MoveIt2.

#### 1.3 Stochastic Trajectory Optimization for Motion Planning (STOMP)

STOMP is a motion planner based on optimization and  $PI^2$  algorithm. This planner can plan smooth trajectories, avoid obstacles and optimize constraints for the robotic arm. Arbitrary terms in the cost function can be optimized since algorithm does not require gradients.

#### 1.4 Search-based Planning Library (SBPL)

It is a generic set of motion planners using search-based planning to discrete space

#### 1.5 Covariant Hamiltonian Optimization for Motion Planning (CHOMP)

CHOMP is an innovative trajectory optimization based on gradient that makes ordinary motion planning simpler and more trainable.

Most high-dimensional motion planners divide the generating process of trajectory into two stages, including planning and optimization. At the stage of optimization, this algorithm employs covariant and functional gradients to design motion planning algorithm that is totally based on trajectory optimization.

Given a infeasible initial trajectory, CHOMP will make quick response to the surroundings so as to make trajectory collision-free, while optimize joint speed and acceleration.

**Note:** Place the robot in a wide area to ensure enough motion space. Keep a certain distance from the robot to avoid hurt when robot is working. Bend the antenna slightly backward to avoid robotic arm from hitting the antenna.

The program used in this Part integrates OMPL and CHOMP planner, and the default planner is OMPL.

## 2. Operation Steps

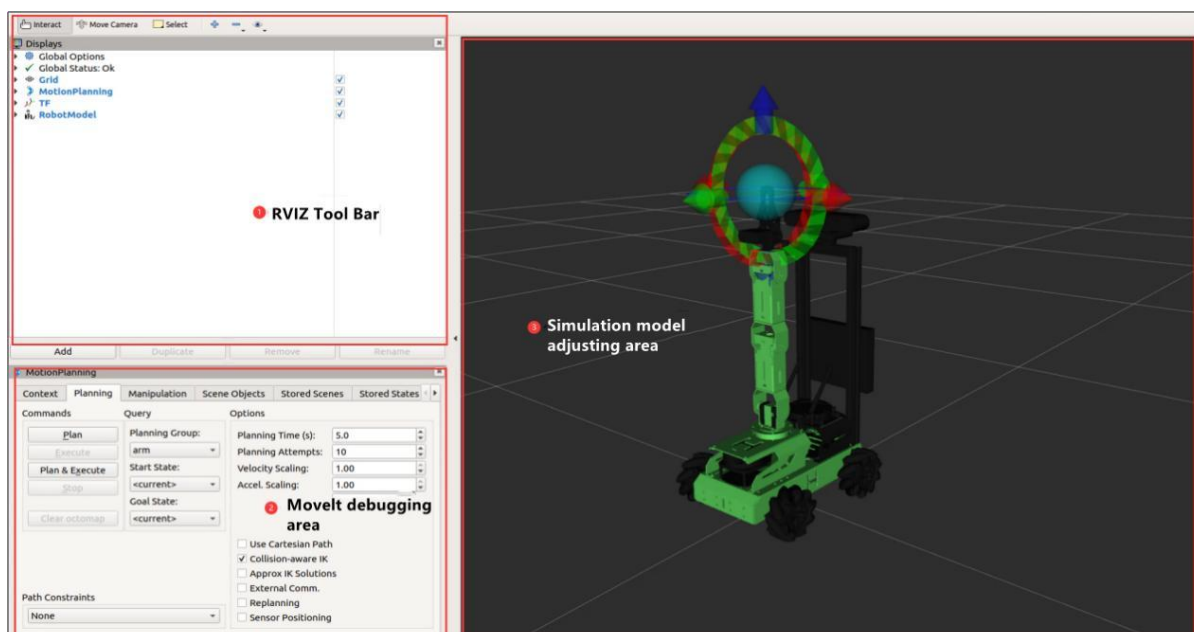
1) Start JetAuto, and connect it to NoMachine.

2) Open terminal.

3) Input command “**sudo systemctl stop start\_app\_node.service**” and press Enter to stop app service.

4) Open a new terminal. Input command “**roslaunch jetauto\_moveit\_config demo.launch fake\_execution:=false pipeline:=chomp**” to enable control service. “**pipeline**” refers to the designated trajectory planner, and the default planner is OMPL. You can change it as “**chomp**” to start CHOMP motion trajectory planning.

The program interface is as pictured.



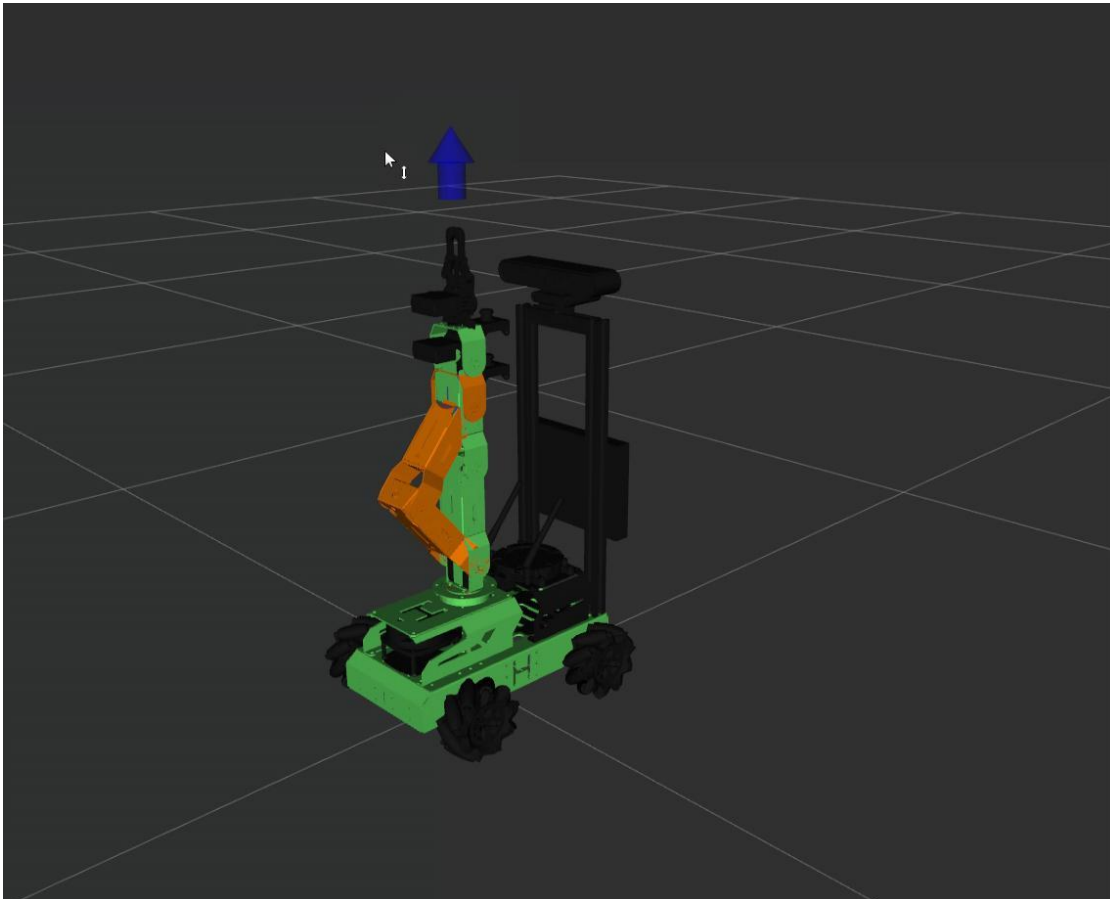
5) Drag arrows in “**simulation model adjusting area**” to adjust robotic arm pose.



With robot as the first person view, red arrow is used to control Y-axis movement, and positive Y-axis direction corresponds to robot left.

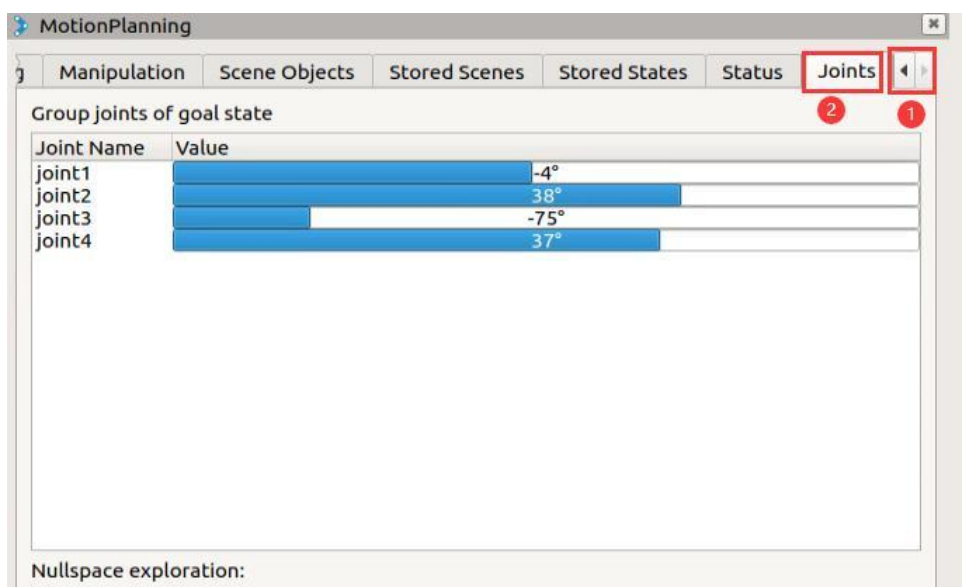
Green arrow controls X-axis movement, and positive X-axis direction corresponds to robot front.

Blue arrow controls Z-axis movement, and positive Z-axis direction corresponds to robot top.

New position of robot is marked in orange as pictured

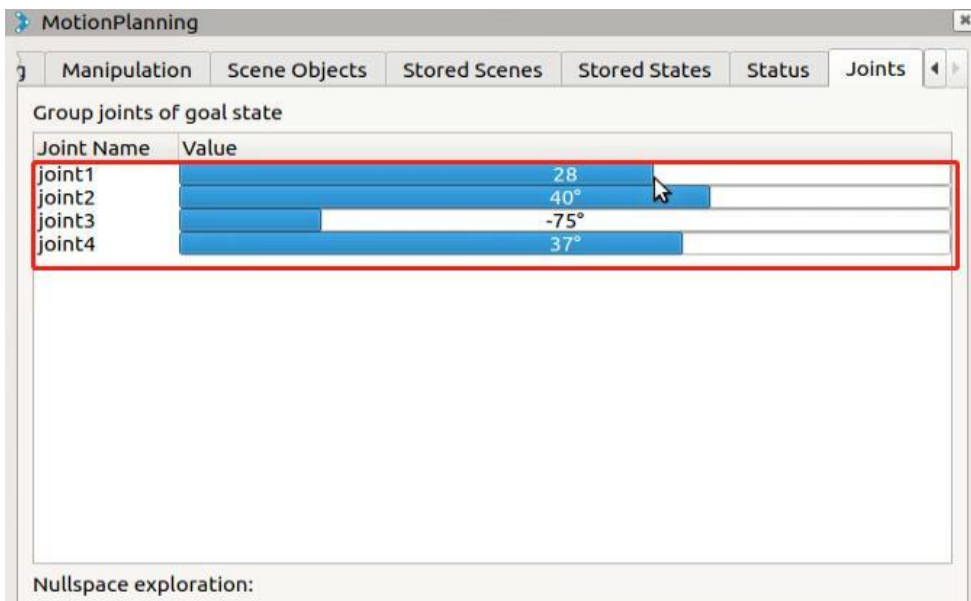


6) Besides using arrows to adjust pose, you can click   to adjust individual joint.

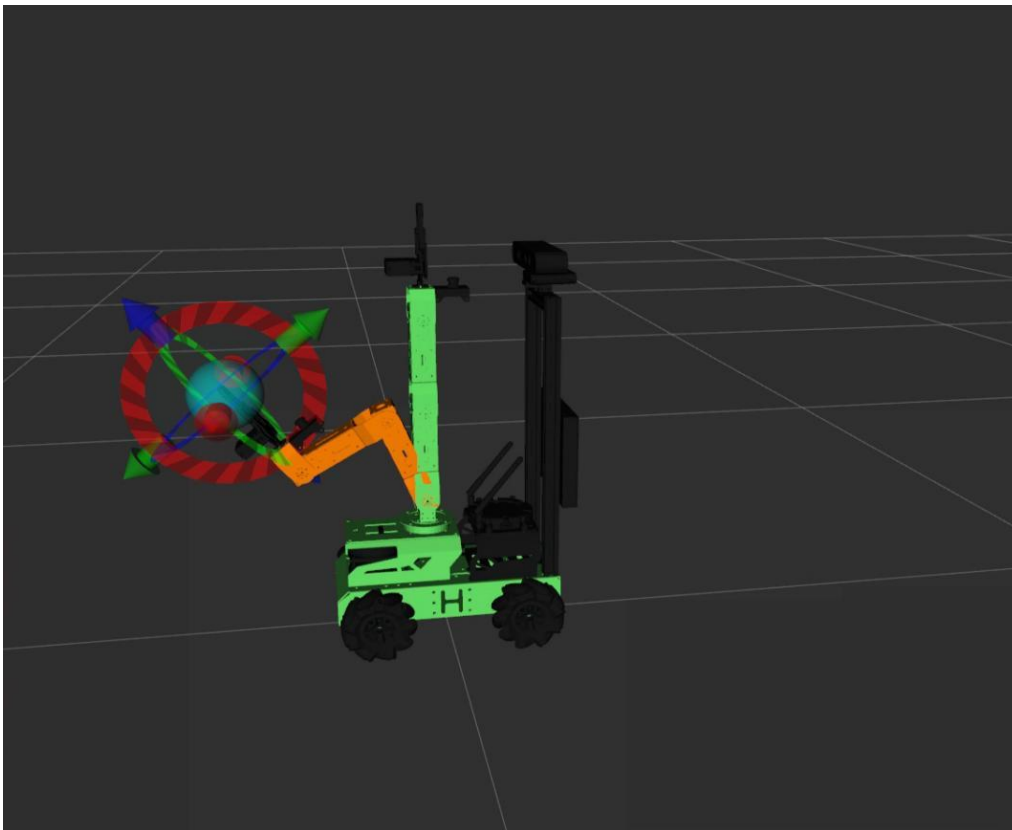




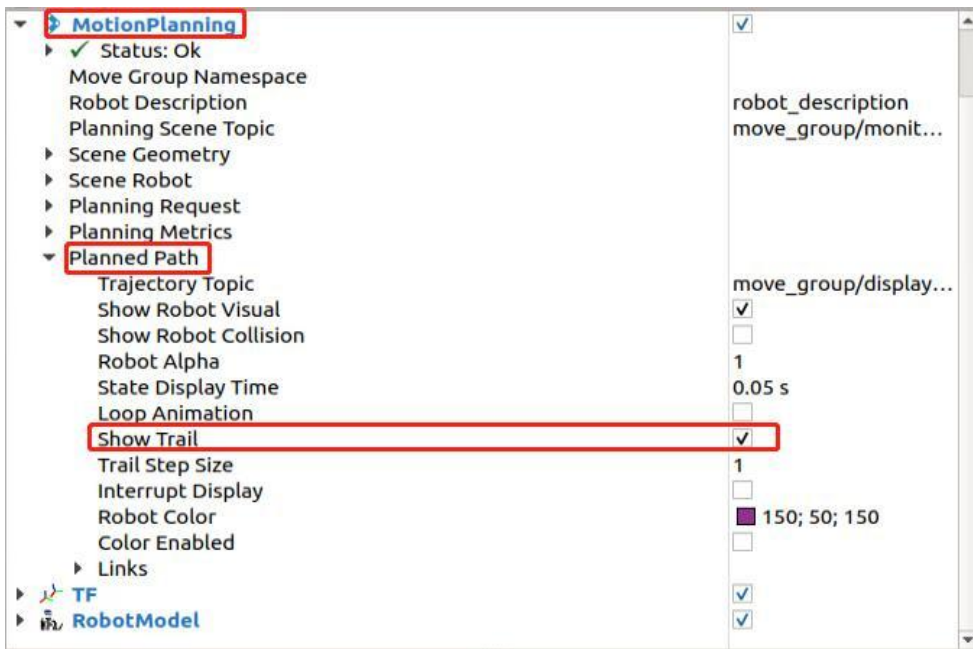
7) Drag sliders to adjust joint angle.



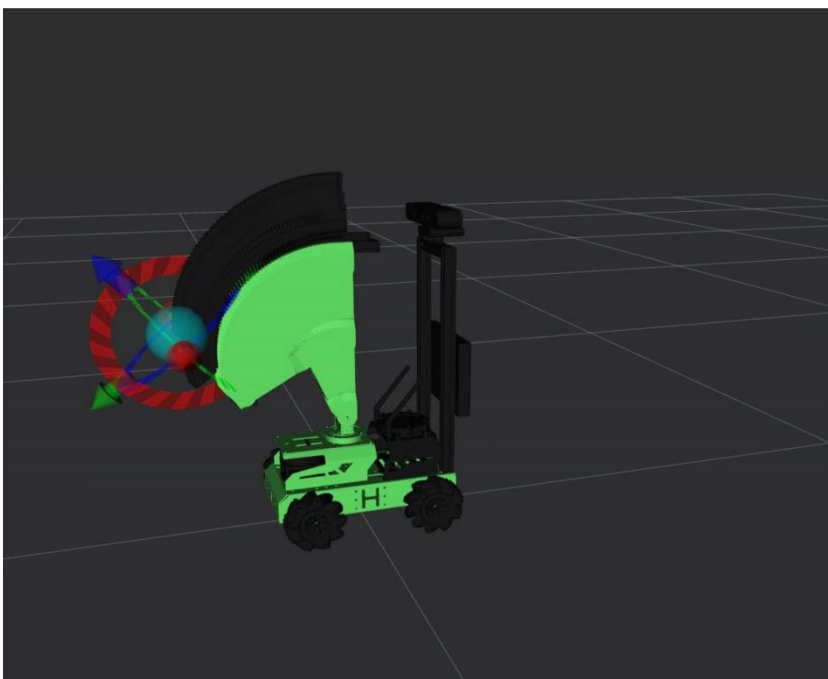
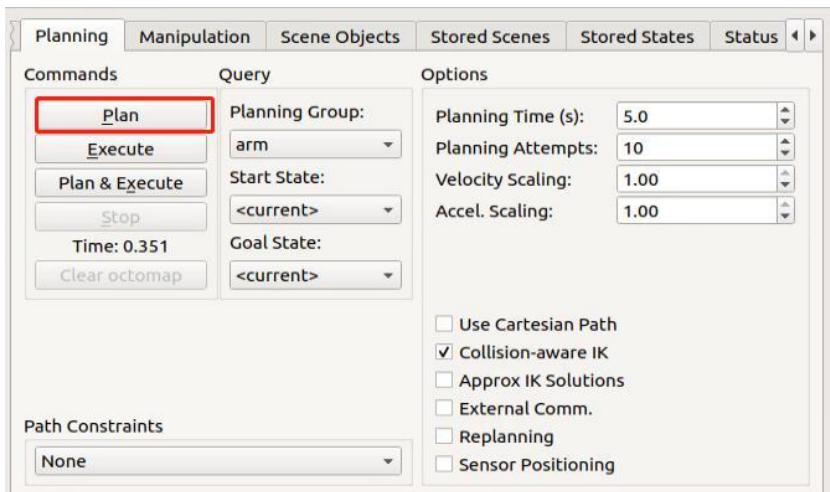
8) After robotic arm motion path is planned successfully, new position is marked in orange as pictured. If robotic arm hit other stuffs, like depth camera, it will be marked in red. You need to adjust the robotic arm pose again until there is no collision, otherwise robotic arm will not perform this action.



9) Move to RVIZ tool bar, click “**Motion Planning > Planned Path**”, then tick “**Show Trail**”. The simulation model will demonstrate each frame of robotic arm motion.



10) Return back to “**Planning**” section. Click “**Plan**” to make simulation model move to the planning position.



11) After that, cancel “**Show Trail**”, then click “**Execute**”. Simulation model and robot will execute the planning action simultaneously.

The screenshot shows a software interface with a 'Planning' tab selected. The interface is divided into three main sections: 'Commands', 'Query', and 'Options'. In the 'Commands' section, the 'Execute' button is highlighted with a red rectangular box. Other buttons in this section include 'Plan', 'Plan & Execute', 'Stop', and 'Clear octomap'. The 'Query' section contains dropdown menus for 'Planning Group' (set to 'arm'), 'Start State' (set to '<current>'), and 'Goal State' (set to 'home'). The 'Options' section includes numerical input fields for 'Planning Time (s)' (5.0), 'Planning Attempts' (10), 'Velocity Scaling' (1.00), and 'Accel. Scaling' (1.00). Below these are several checkboxes: 'Use Cartesian Path' (unchecked), 'Collision-aware IK' (checked), 'Approx IK Solutions' (unchecked), 'External Comm.' (unchecked), 'Replanning' (unchecked), and 'Sensor Positioning' (unchecked). At the bottom left, there is a 'Path Constraints' dropdown menu set to 'None'. The top of the window has a tab bar with 'Planning', 'Manipulation', 'Scene Objects', 'Stored Scenes', 'Stored States', and 'Status'.