Songling Robot Product **Agile** Composite Mobile Robot Development Kit User Guide

AGILE
SCOUT

1 Product Introduction

Agile compound mobile robot series development kit is an advanced development kit specially developed by Songling Robot for industrial scientific research and education application customers. In the face of scientific research and education, the robot products are becoming more and more flexible. Based on the Songling Robot ROS ecosystem, this package integrates high-performance tools. control, high-precision LiDAR, multi-sensors, multi-DOF robotic arms, and visual perception. Kit contains multi-line lidar autonomous Navigation, robotic arm moveIt motion control planning, visual recognition, robotic arm autonomous grasping and other functions are integrated into one, which solves the problem for customers in the face of mobile machines. Robots are characterized by high technical complexity and high integration difficulty in complex application scenarios, bringing customers the ultimate user experience. Product includes The complete technical manual and supporting technical documents and codes are all open source, which reduces the difficulty of application and learning for customers. Product kits can It is widely used in agriculture, intelligent manufacturing, education and training, scientific research and exploration.

2 Main configuration list and parameter description

2.1 Main configuration

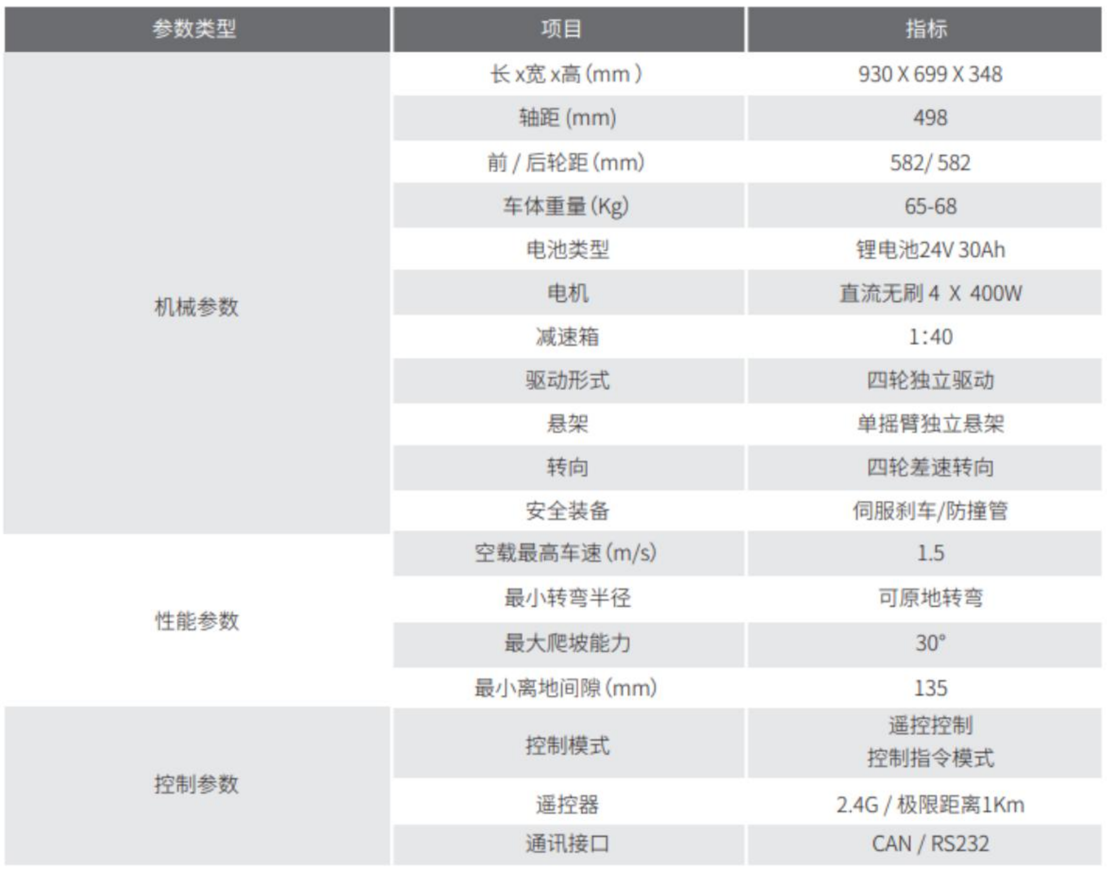
Kit version	**Cobot Kit**
Chassis mobile platform	SCOUT 2.0
robotic arm	Xarm 5
Robot Arm Clamp	Dahuan AG 95
IPC	x-7010 (i7 16G 256SSD)
Vision sensor	RealSense D435
lidar sensor	VLP 16
Screen	HD display
router	B316 4G Router

2.2 Introduction of main accessories

- **SCOUT** product introduction

  SCOUT2.0 is an all-round industry application UGV (UNMANNED GROUND VEHICLE). It is a module that adopts The multi-functional and modular industrial application mobile robot development platform with the concept of intelligent and intelligent design has strong load capacity and strong It has a wide range of application areas for the power system. Stereo cameras, lidars, GPS, IMU, mechanical hands and other equipment can be optionally added. Install SCOUT2.0 as an extension application. SCOUT2.0 can be applied to unmanned inspection, security, scientific research, exploration, logistics and other fields area.

AGILE X
SCOUT

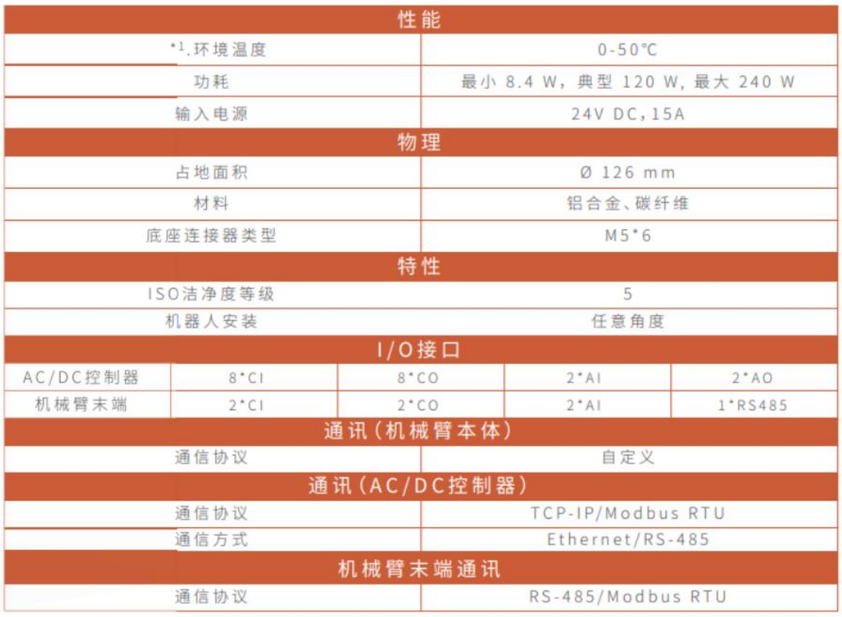


参数类型	项目	指标
机械参数	长 x宽 x高 (mm)	930 X 699 X 348
	轴距 (mm)	498
	前 / 后轮距 (mm)	582/ 582
	车体重量 (Kg)	65-68
	电池类型	锂电池24V 30Ah
	电机	直流无刷 4 X 400W
	减速箱	1:40
	驱动形式	四轮独立驱动
	悬架	单摇臂独立悬架
	转向	四轮差速转向
	安全装备	伺服刹车/防撞管
性能参数	空载最高车速 (m/s)	1.5
	最小转弯半径	可原地转弯
	最大爬坡能力	30°
	最小离地间隙 (mm)	135
控制参数	控制模式	遥控控制 控制指令模式
	遥控器	2.4G / 极限距离1Km
	通讯接口	CAN / RS232

- Introduction of Xarm Robotic Arm

The Xarm series robot is an industrial-grade lightweight collaborative robot product carefully selected by Songling Robot according to the scientific research and education industry. The product adopts the joint modular design and uses the robot for developers. Through this interface, users can observe the running status of the robot in real time, make many control settings for the robot, and can also perform offline simulation offline, which greatly improves the work efficiency of practical applications. At the same time, according to the characteristics of the scientific research and education industry, we adapt ROS and support open source solutions such as Moveit.

Machine Translated by Google



性能				
*1.环境温度	0-50℃			
功耗	最小 8.4 W，典型 120 W，最大 240 W			
输入电源	24V DC，15A			
物理				
占地面积	Ø 126 mm			
材料	铝合金、碳纤维			
底座连接器类型	M5*6			
特性				
ISO洁净度等级	5			
机器人安装	任意角度			
I/O接口				
AC/DC控制器	8*CI	8*CO	2*AI	2*AO
机械臂末端	2*CI	2*CO	2*AI	1*RS485
通讯（机械臂本体）				
通信协议	自定义			
通讯（AC/DC控制器）				
通信协议	TCP-IP/Modbus RTU			
通信方式	Ethernet/RS-485			
机械臂末端通讯				
通信协议	RS-485/Modbus RTU			

*1. 机器人的工作温度为0-50℃。当关节在高速连续工作时，请尝试降低环境温度。

- **Introduction of X-7010 Industrial Computer**

X-7010 is a modular combination of high-performance ultra-small industrial computer, an industrial industrial control computer customized for the characteristics of high computing power and high environmental requirements in the mobile robot industry. It adopts Intel platform and supports 8/9th 35W high-speed processor. It adopts the large-area aluminum fins of high-efficiency heat pipes and the active/passive dual heat dissipation design of PWM fans. It is suitable for areas with high computing requirements such as intelligent robots, unmanned driving, machine vision, and smart cities. Pre-installed Ubuntu 18.04, ROS Melodic (Full Desktop) version, and some development environments commonly used in robot development, enabling instant use.

- Has a palm-sized compact and ultra-small body;
- Support Intel 8th desktop high-performance CPU;
- Active and passive heat dissipation equipped with heat pipes and smart fans;
- Supports various acceleration card expansion schemes such as miniPCIE and NVME;
- Multi-channel ultra-high-speed dedicated serial ports, suitable for a variety of radar applications;
- High-speed communication with multi-channel USB3.1 Gen2 and dual Gigabit network;
- Sturdy die-formed aluminum alloy machine, compliant with heavy-duty vibration and shock;
- -20~60ÿ wide temperature operation;

- ## Vision Sensor RealSense D435

Binocular vision sensors have a wide range of application scenarios and needs in the robot industry, such as robot vision measurement and visual navigation. At present, we have selected vision sensors that are commonly used in the scientific research and education industry. Equipped with a global image snapshot and wide field of view, the Intel RealSense Depth Camera D435 efficiently captures and streams depth data from moving objects, providing highly accurate depth perception for mobile prototypes.

	model	**Intel Realsense D435**
Basic Features	Application scenarios	outside/indoor
Measuring distance	about 10 meters	
Deepin Quick Type	Global Express/3um X 3um	
Whether to support IMU	support	
depth camera	deep tech	Active infrared
FOV	86° x 57° (±3°)	
Minimum depth distance	0.105m	
depth resolution	1280x720	
Maximum measurement distance	about 10 meters	
depth frame rate	90fps	
RGB	Resolution	1280x800
FOV	69.4° × 42.5° (±3°)	
frame rate	30fps	
other information	size	90mm x 25mm x 25mm
Interface Type	USB-C 3.1	

- ## radar sensor

The RS-16 is a small and advanced version of the RoboSense lidar. RS-16 is cost-effective compared to similarly priced sensors higher, and retains some of the main features of RoboSense's breakthrough in lidar, such as real-time, 360°, three Dimensional coordinates and distances, reflectance measurements with calibration.

RS-16 has a measurement range of up to 100 m, low power consumption (about 8 W), light weight (about 830 g), and small size (Ø103mm x 72mm) with dual return features that make it ideal for backpack measurements, drone mounts and other mobile devices choose.

item	parameter
Maximum ranging	100m
Ranging accuracy	±3cm
scan rate	Single echo 300,000 points/second Double echo 600,000 points/second
vertical view	-15°~+15°
Vertical resolution	2°
scanning frequency	5Hz~20Hz
Security Level	Class 1
weight	830g
Power consumption	8W
Voltage	9V~18V
working temperature	-10ÿ~+60ÿ

- mechanical clamp

Dahuan Robot AG 95 Electric Gripper has two self-adaptive parallel mechanical joint fingers (rear joint fingers and each joint finger). It consists of several linkages and a spring, as shown in Figure 1.1. Articulated fingers can make up to 5 points of contact with an object s contact. The joint fingers are driven by a tactile drive control method, so that the number of motors is less than the total number of joints. This design simplifies grasping , so that the articulated fingers can automatically adapt to the shape of the object they grasp.

Maximum recommended load	**3-5kg***
Finger opening and closing stroke (programming adjustable)	0-95mm
Grip force (programmable adjustable)	45-160N
Fastest finger opening and closing speed	190mm/s
own weight	1kg
Finger repeatability	0.03mm
Protocol	TCP/IP, USB2.0, RS485, I/O, CAN2.0A, EtherCAT (optional)
working voltage	24V DC±10%
Operating temperature range	0~50ÿ

3 Example development

3.1 Preparation before development

3.1.1 Download Remote Desktop Tools

A remote desktop tool is installed on the industrial computer in the robot. The user needs to install the corresponding tool on their own laptop or computer, and remotely control the computer on the robot through the router on the robot. Industrial computer.

(1) Download the installation package

https://www.nomachine.com/download

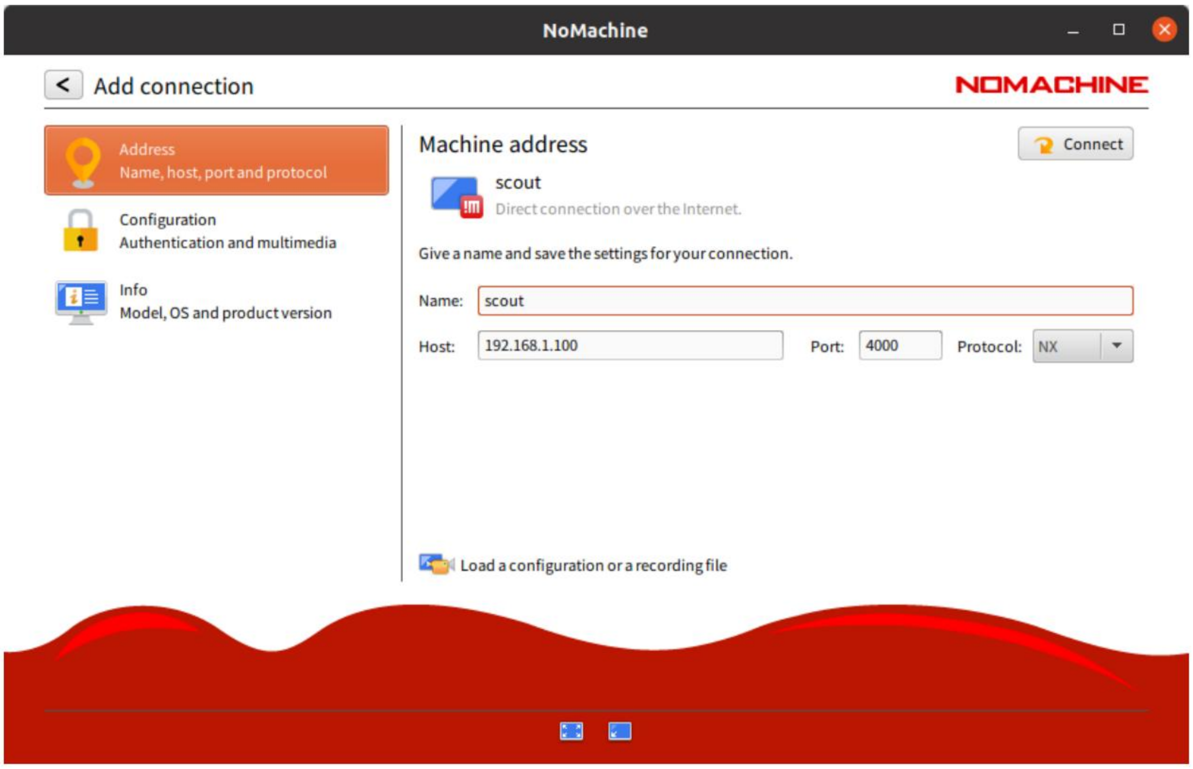
Select the operating system corresponding to your laptop or computer to download and install.

(2) How to use

The user on the industrial computer: scout, the computer's power-on password: agx. Router name: HAIWEI_B316_BD96, router password and login password are the same as 12345678. Router ip: 192.168.1.1, industrial computer ip: 192.168.1.100.

First, use your own computer or laptop to find the wifi on the robot, and enter the password to connect.

Open the downloaded remote connection tool NoMachine and add a remote connection:


NoMachine
Add connection
NOMACHINE
Address
Name, host, port and protocol
Configuration
Authentication and multimedia
Info
Model, OS and product version
Connect
Load a configuration or a recording file

Machine address

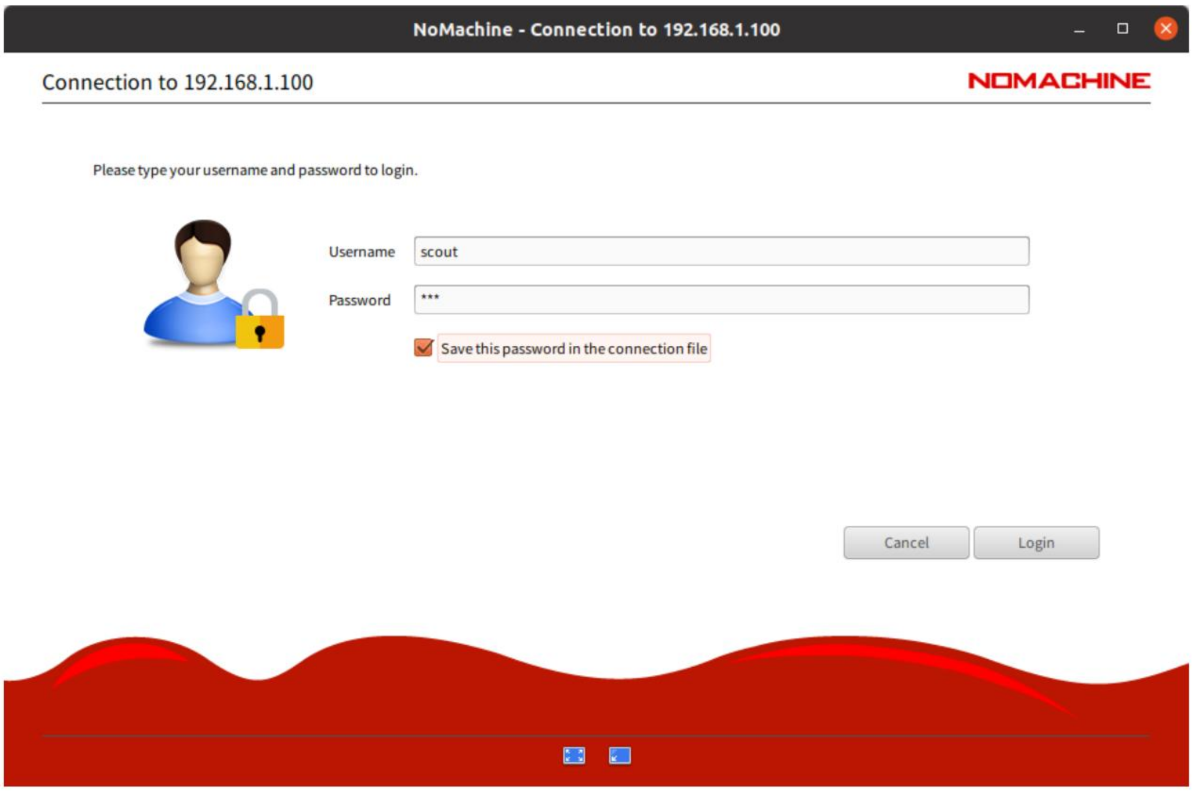
scout
Direct connection over the Internet.

Give a name and save the settings for your connection.

Name: scout
Host: 192.168.1.100 Port: 4000 Protocol: NX

name: scout, host: 192.168.1.100

Enter username: scout and password: agx, check Remember password.


NoMachine - Connection to 192.168.1.100

Connection to 192.168.1.100

Please type your username and password to login.

Username: scout

Password: ***

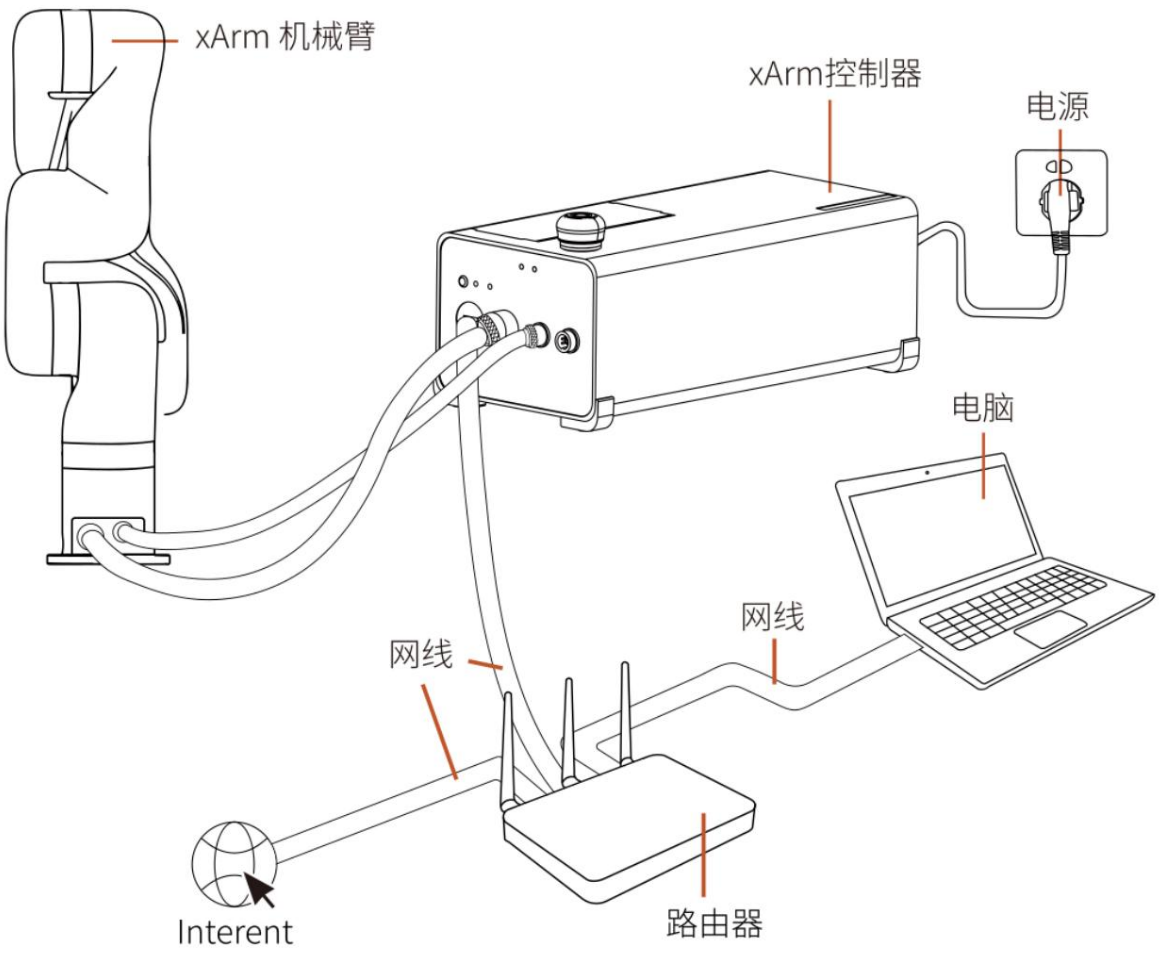
☑ Save this password in the connection file

It can be connected to the industrial computer on the robot.

3.1.2 Connect to control the movement of the robotic arm

The manipulator mounted on the robot is UFACTORY's xArm 6-axis manipulator, manipulator body, manipulator controller, and robot chassis

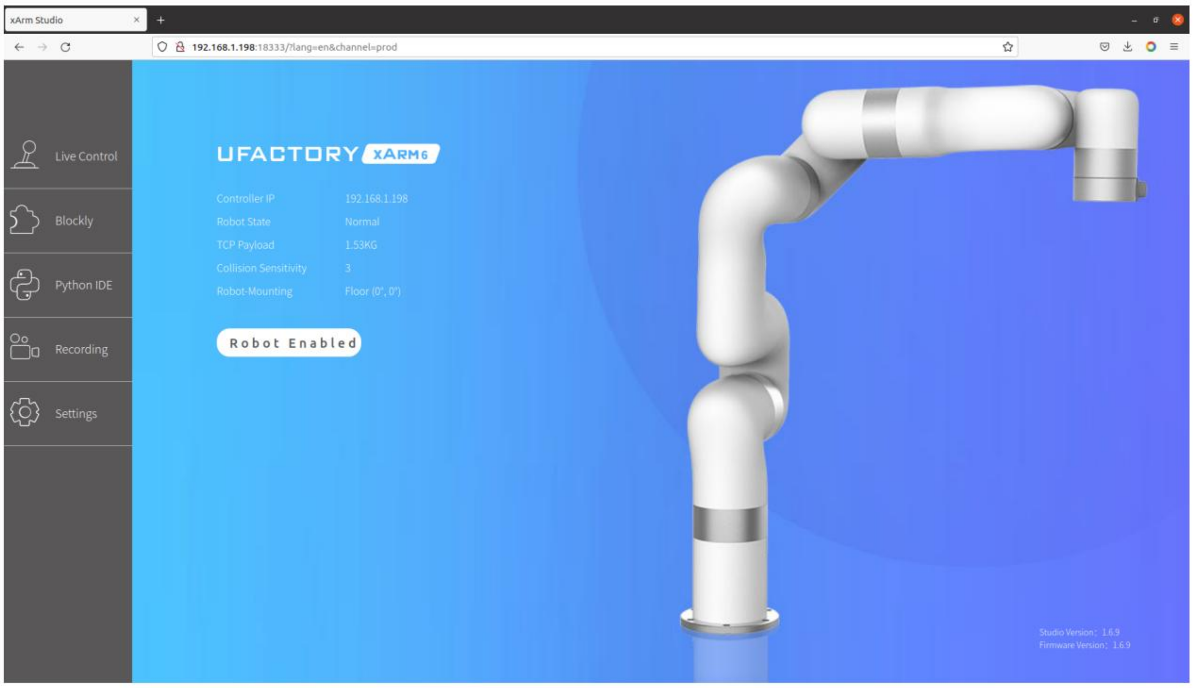
The connection between controllers (industrial personal computers) is shown in the following figure:



The computer in the picture represents the robot chassis controller. The network communication between the robot arm controller and the chassis controller is carried out through the router. The network segment of the local area network is 192.168.1, and the default IP address of the robot arm is 192.168.1.198.

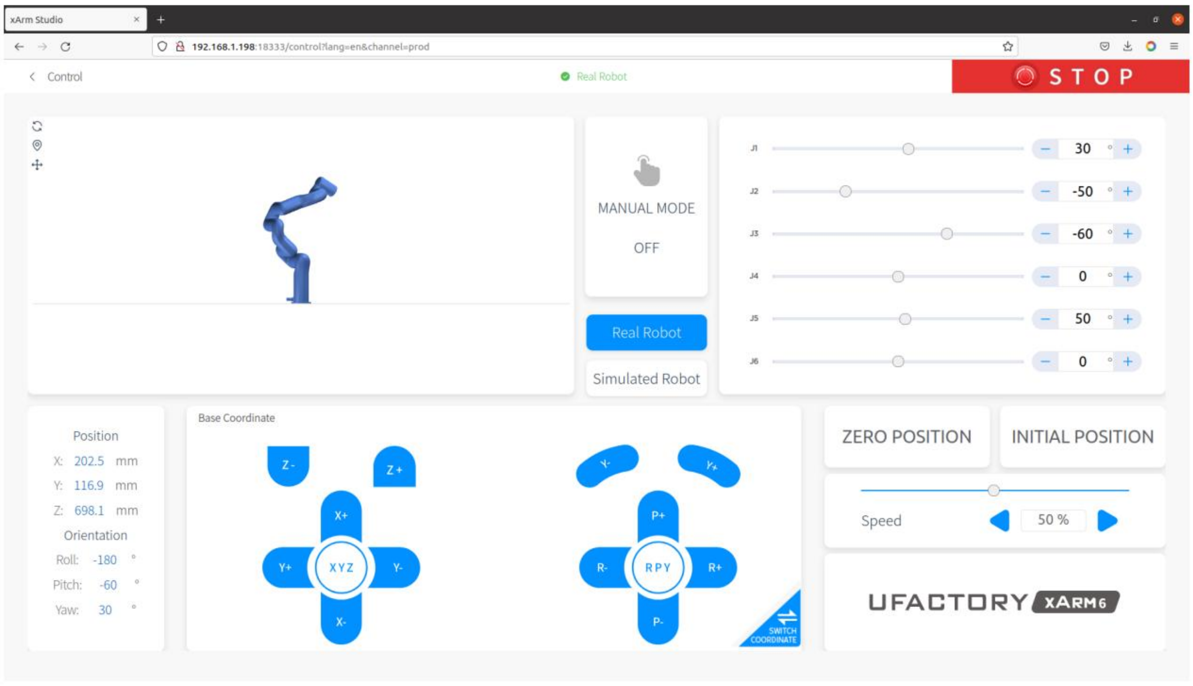
- interface control

After turning on the power of the robot, we open the browser, enter 192.168.1.203:18333 in the address bar, and go back to the robot arm web interface:



Controller IP 192.168.1.198
Robot State Normal
TCP Payload 1.53KG
Collision Sensitivity 3
Robot-Mounting Floor (0°, 0°)

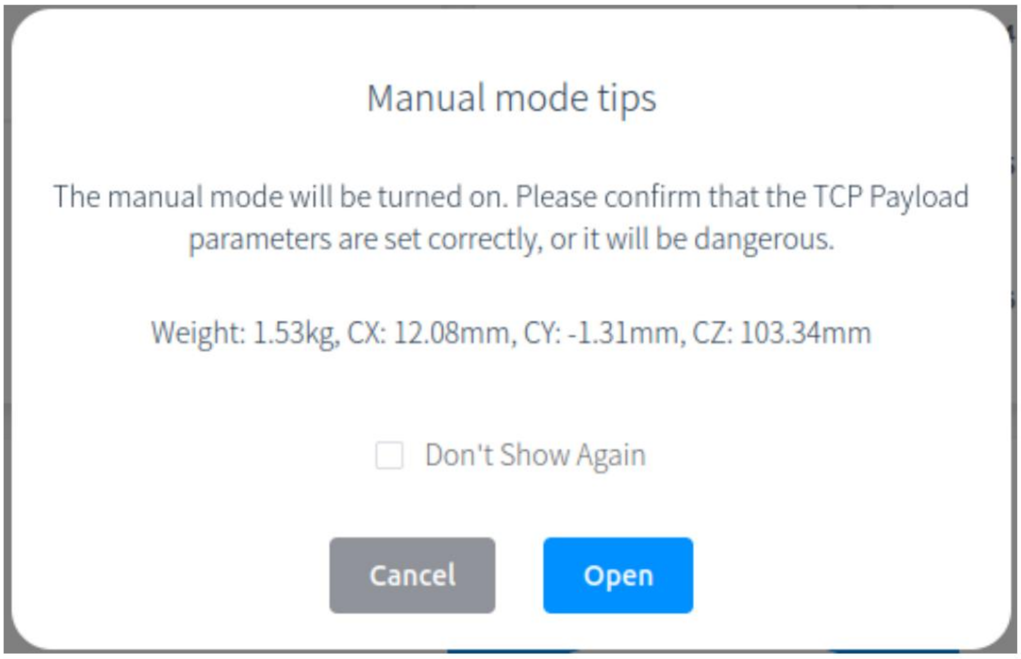
Click Live Control on the left to enter the real-time control interface of the robotic arm as shown in the figure below. Here we can see the state configuration of the robotic arm, the end position and posture of the robotic arm, and the angle data of each joint. At the same time, we can also control each joint of the robotic arm to move through the corresponding buttons, or directly Control the pose of the end of the robotic arm in the base coordinate system.



J1 30 °
J2 -50 °
J3 -60 °
J4 0 °
J5 50 °
J6 0 °

Position
X: 202.5 mm
Y: 116.9 mm
Z: 698.1 mm
Orientation
Roll: -180 °
Pitch: -60 °
Yaw: 30 °

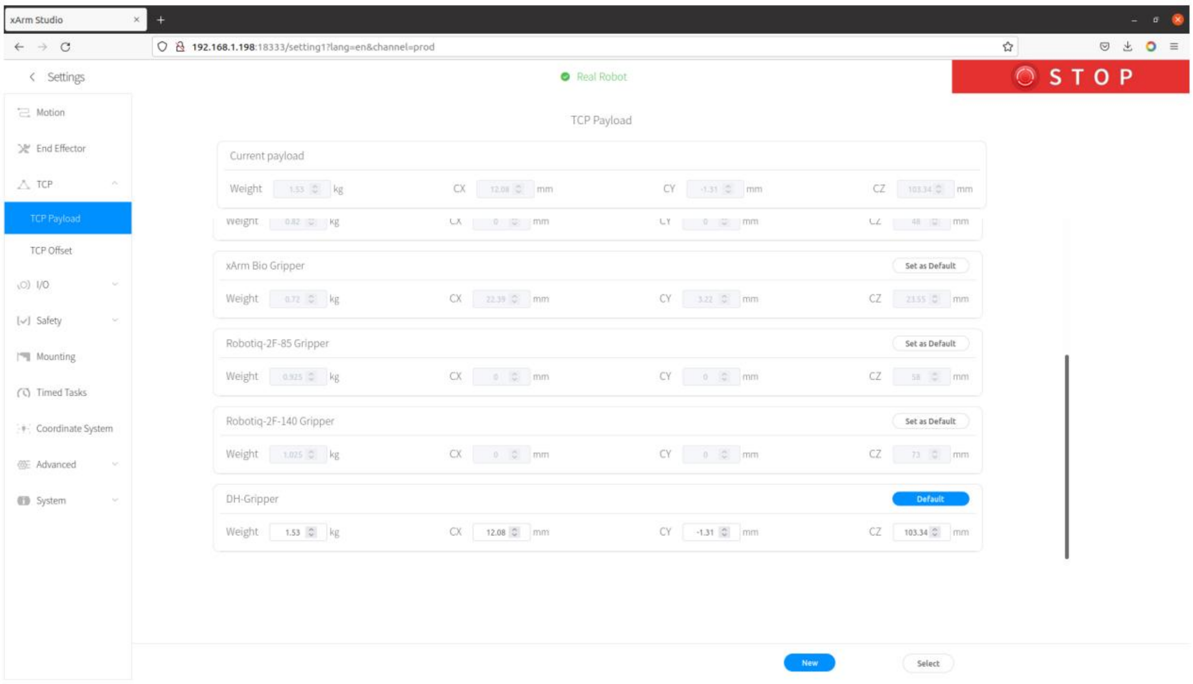
In the interface, we can click the MANUAL MODE button to turn it ON to start the manual mode, and then we can drag the robotic arm to move. Note: Before turning on the manual mode, you need to configure the load parameters of the end tool of the robot arm, which are already configured by default, and the user can also perform automatic parameter identification on the setting page.



Manual mode tips

The manual mode will be turned on. Please confirm that the TCP Payload parameters are set correctly, or it will be dangerous.

Weight: 1.53kg, CX: 12.08mm, CY: -1.31mm, CZ: 103.34mm



192.168.1.198:18333/setting1?lang=en&channel=prod

Real Robot

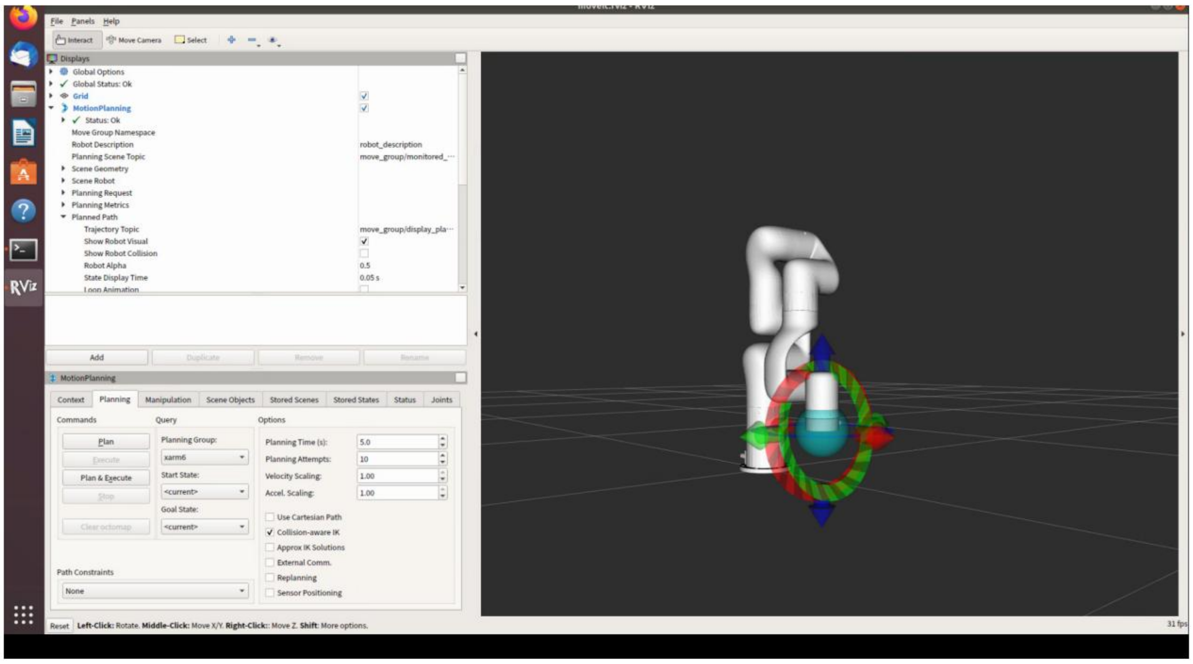
TCP Payload

	Weight (kg)	CX (mm)	CY (mm)	CZ (mm)
Current payload	1.53	12.08	-1.31	103.34
	0.82	0	0	48
xArm Bio Gripper (Set as Default)	0.72	22.39	3.22	23.55
Robotiq-2F-85 Gripper (Set as Default)	0.925	0	0	58
Robotiq-2F-140 Gripper (Set as Default)	1.025	0	0	73
DH-Gripper (Default)	1.53	12.08	-1.31	103.34

The DH Gripper parameter in the TCP payload above is an adaptive identification parameter, which includes the Dahuan robotic hand and RealSense camera installed at the end of the robotic arm.

- **MoveIt! Control**

Open a terminal in the robot and run the command `roslaunch xarm6_moveit_config realMove_exec.launch robot_ip:=192.168.1.198` , start MoveIt! robot arm planning control.


moveit.rviz - RViz
File Panels Help
Interact Move Camera Select
Displays
Add Duplicate Remove Rename
MotionPlanning
Context Planning Manipulation Scene Objects Stored Scenes Stored States Status Joints
Plan Execute Plan & Execute Stop Clear octomap
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Move Z. Shift: More options.
31 fps

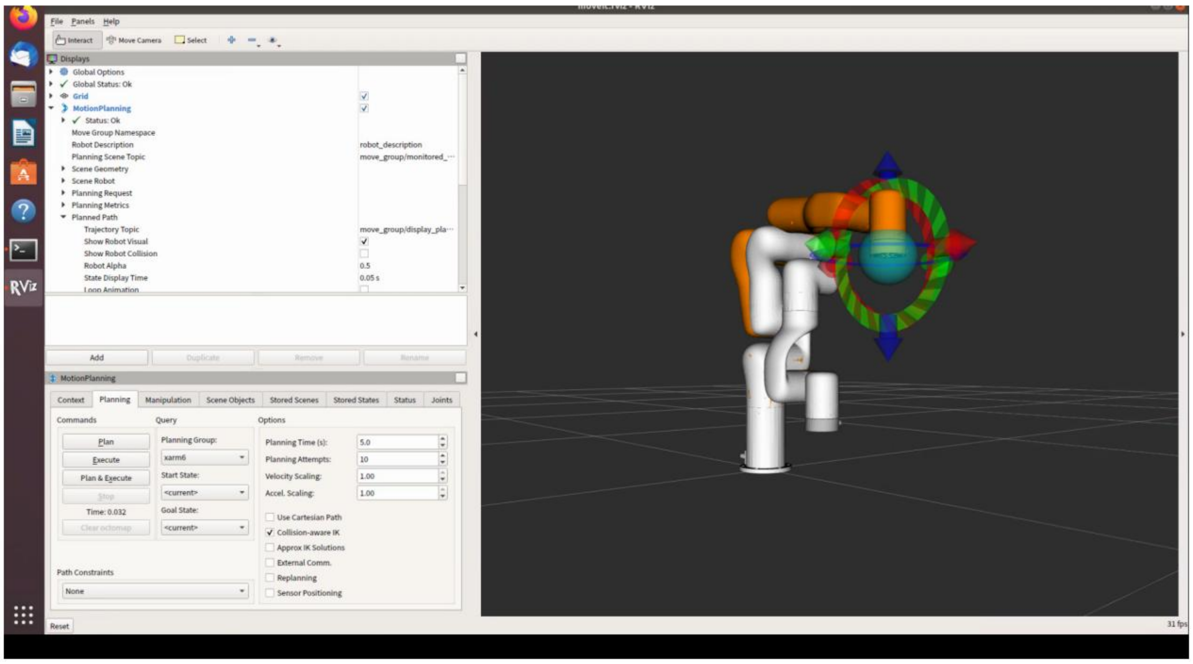
Global Options
Global Status: Ok
Grid ✓
MotionPlanning ✓
Status: Ok
Move Group Namespace
Robot Description robot_description
Planning Scene Topic move_group/monitored_...
Scene Geometry
Scene Robot
Planning Request
Planning Metrics
Planned Path
Trajectory Topic move_group/display_pla...
Show Robot Visual ✓
Show Robot Collision
Robot Alpha 0.5
State Display Time 0.05 s
Loop Animation

Commands / Query / Options
Planning Group: xarm6
Start State: <current>
Goal State: <current>
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
Use Cartesian Path
✓ Collision-aware IK
Approx IK Solutions
External Comm.
Replanning
Sensor Positioning
Path Constraints: None

We can use the mouse to drag the end of the robotic arm in RViz to another pose, and then click Plan in the lower left corner of MotionPlanning

Click the button to see if MoveIt can plan from the current pose to the target pose. If the path planning is successful and there are no other obstacles in the path, click the

Plan button to control the robotic arm to move to the target pose.


moveit.rviz - RViz
File Panels Help
Interact Move Camera Select
Displays
Add Duplicate Remove Rename
MotionPlanning
Context Planning Manipulation Scene Objects Stored Scenes Stored States Status Joints
Plan Execute Plan & Execute Stop Clear octomap
Reset
31 fps

Global Options
Global Status: Ok
Grid ✓
MotionPlanning ✓
Status: Ok
Move Group Namespace
Robot Description robot_description
Planning Scene Topic move_group/monitored_...
Scene Geometry
Scene Robot
Planning Request
Planning Metrics
Planned Path
Trajectory Topic move_group/display_pla...
Show Robot Visual ✓
Show Robot Collision
Robot Alpha 0.5
State Display Time 0.05 s
Loop Animation

Commands / Query / Options
Time: 0.032
Planning Group: xarm6
Start State: <current>
Goal State: <current>
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
Use Cartesian Path
✓ Collision-aware IK
Approx IK Solutions
External Comm.
Replanning
Sensor Positioning
Path Constraints: None

- ROS command control

There are three commonly used control modes for the robotic arm, which are:

Mode 0 : Position mode based on xArm controller planning;

ÿ ***Mode 1 :*** Position mode based on external trajectory planner; ***Mode 2 :*** Free

drag (zero gravity) mode.

MoveIt! uses Mode1 for control. Before using ROS command control, you need to switch the mode to Mode 0:

```
error occurs $ rosservice call /xarm/clear_err $ rosservice call /x#Clear motion_ctrl 8 1 #Enable
the joints, run when the joints are not enabled $ rosservice call /xarm/set_mode 0
                                                                   #Set the desired
$ rosservice call /xarm/set_state 0                                operating mode #set state to 0 (Ready state)
```

After the mode switching is completed, the robot arm can be controlled to move in the joint space through the following commands:

Machine Translated by Google

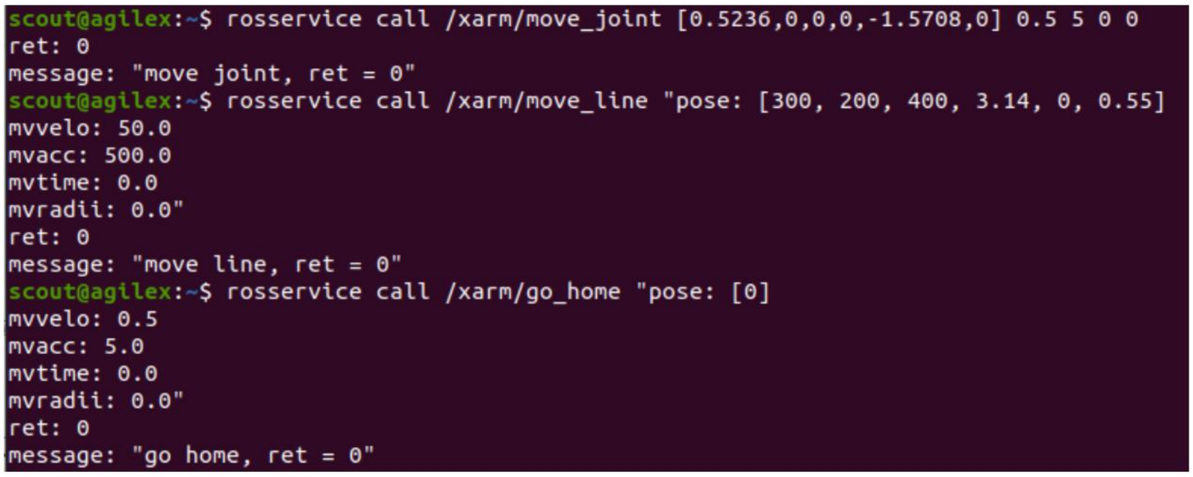
```
$ rosservice call /xarm/move_joint [0.5236,0,0,0,-1.5708,0] 0.5 5 0 0
```

The robot arm is controlled to move in Cartesian space by the following commands:

```
$ rosservice call /xarm/move_line "pose: [300, 200, 300, 3.14, 0, 0.55] mvvelo: 50.0

mvacc: 500.0
mvtime: 0.0
mvradii: 0.0"
```

Zero the robotic arm with the following command:

```
$ rosservice call /xarm/go_home "pose: [0] mvvelo: 0.5

mvacc: 5.0
mvtime: 0.0
mvradii: 0.0"
```



```
scout@agilex:~$ rosservice call /xarm/move_joint [0.5236,0,0,0,-1.5708,0] 0.5 5 0 0
ret: 0
message: "move joint, ret = 0"
scout@agilex:~$ rosservice call /xarm/move_line "pose: [300, 200, 400, 3.14, 0, 0.55]
mvvelo: 50.0
mvacc: 500.0
mvtime: 0.0
mvradii: 0.0"
ret: 0
message: "move line, ret = 0"
scout@agilex:~$ rosservice call /xarm/go_home "pose: [0]
mvvelo: 0.5
mvacc: 5.0
mvtime: 0.0
mvradii: 0.0"
ret: 0
message: "go home, ret = 0"
```

3.1.3 Use ar_track_alvar to achieve visual positioning tracking

3.1.3.1 Function Introduction

This package is ROS PACKAGE for Alvar, an open source AR tag tracking library.

(1) ar_track_alvar has 4 main functions: Generate AR tags of different sizes, resolutions and data/ID encodings

(2) Identify and track the pose of a single AR tag, optionally integrating depth data from a depth camera for better pose estimation.

(3) Recognize and track combined poses consisting of multiple labels. This allows for more stable pose estimation, robustness to occlusion, and tracking of polygonal objects.

(4) Automatically calculate the spatial relationship between tags in the bundle using camera images, so that the user does not have to manually measure and input in the XML file.

Enter the tag location to use the bundling feature (currently not working).

Alvar is newer and more advanced than ARToolkit, which has been the basis for several other ROS AR tagging packages. Alvar features adaptive thresholding to handle various lighting conditions, optical flow-based tracking for more stable pose estimation, and an improved label recognition method that does not slow down significantly as the number of labels increases .

3.1.3.2 Instructions for use

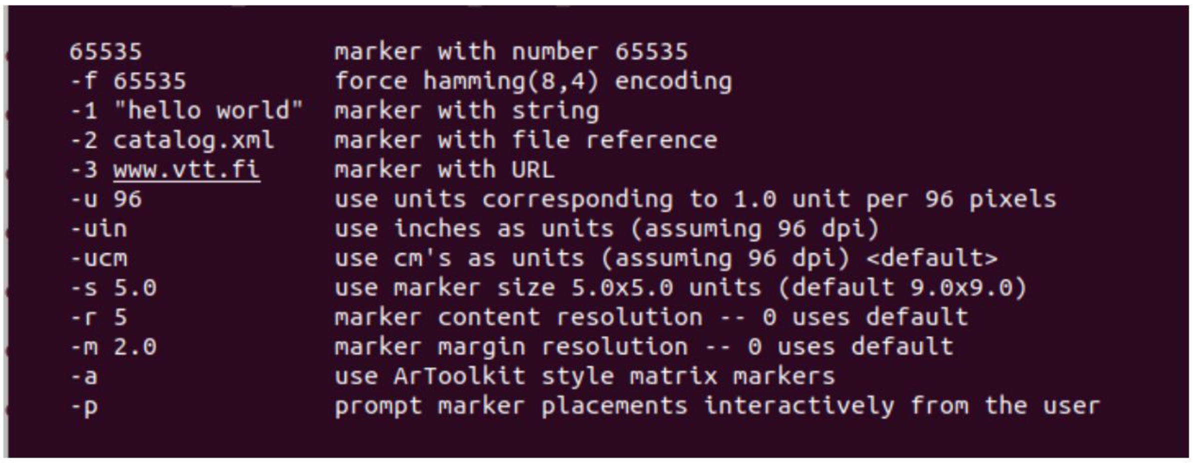
(1) Generate labels

Machine Translated by Google

```
$ source ~/catkint_workspace/devel/setup.bash $ rosrun
ar_track_alvar createMarker 0 -s 3.0
```

> The number behind represents the id number of the generated tag, and tags with different numbers can be generated as needed

The detailed parameter settings are as follows:



```
65535              marker with number 65535
-f 65535           force hamming(8,4) encoding
-1 "hello world"   marker with string
-2 catalog.xml     marker with file reference
-3 www.vtt.fi      marker with URL
-u 96              use units corresponding to 1.0 unit per 96 pixels
-uin               use inches as units (assuming 96 dpi)
-ucm               use cm's as units (assuming 96 dpi) <default>
-s 5.0             use marker size 5.0x5.0 units (default 9.0x9.0)
-r 5               marker content resolution -- 0 uses default
-m 2.0             marker margin resolution -- 0 uses default
-a                 use ArToolkit style matrix markers
-p                 prompt marker placements interactively from the user
```

(2) Printing labels

Pay attention to the size when printing labels, the default is to use 3x3 cm labels, if the printing size is different from the default size, please modify the label size in the ~/

your_workspace/src/open_manipulator_perceptions/open_manipulator_ar_markers/launch/agx _ar_pose.launch file .

```
<arg name="user_marker_size"              default="3"/>
```

> In the process of use, the default label 0 is attached to the object, and the label 2 is attached to the object placement platform.

(3) Operation recognition function

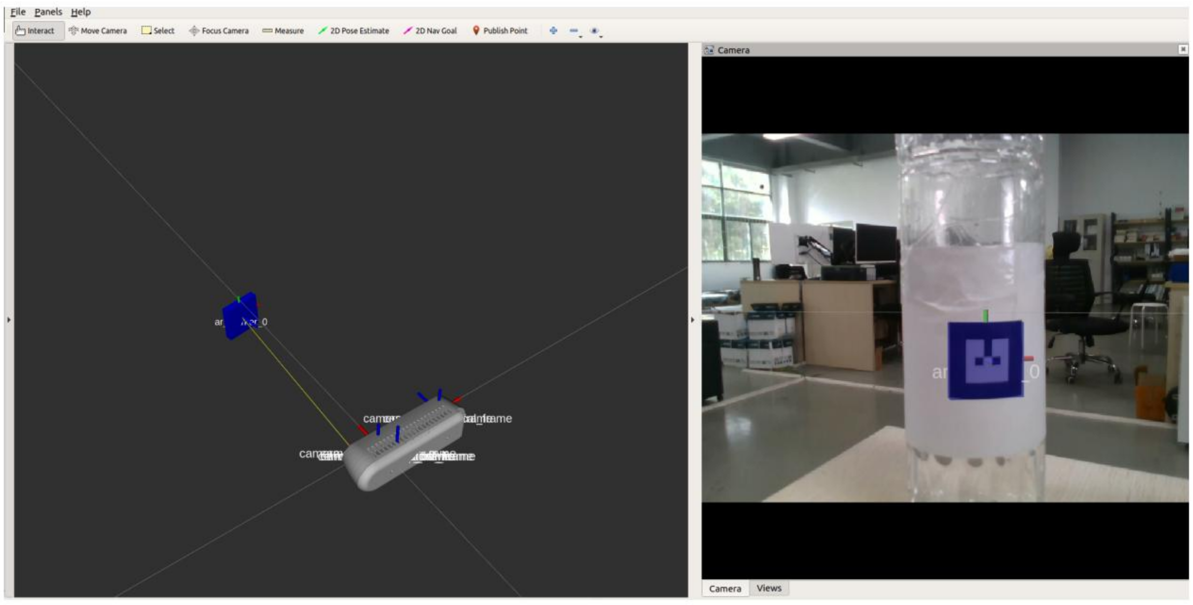
Connect the camera to the computer with a usb3.0 cable, and run the following command:

```
$ source ~/catkin_workspace/devel/setup.bash $ roslaunch
agx_xarm_bringup agx_ar_pose.launch
```

The structure of the agx_ar_pose.launch file is shown in the following figure, and the detailed parameters are shown in the following table:

```xml
<?xml version="1.0" ?>
<launch>
  <arg name="x"            default="0"/>
  <arg name="y"            default="0"/>
  <arg name="z"            default="0"/>
  <arg name="roll"            default="0"/>
  <arg name="pitch"            default="0"/>
  <arg name="yaw"            default="0"/>
  <arg name="r_x"            default="0"/>
  <arg name="r_y"            default="0"/>
  <arg name="r_z"            default="0"/>
  <arg name="r_w"            default="0"/>
  <arg name="parent_link"            default="wrist3_Link"/>
  <arg name="serial_no"            default=""/>
  <arg name="marker_frame_id"      default="_color_frame"/>
  <arg name="user_marker_size"     default="3"/>
  <arg name="use_quaternion" default="false"/>
  <arg name="camera_model" default="realsense_d435" doc="model type [astra_pro, realsense_d435, raspicam]"/>
  <arg name="camera_namespace" default="camera"/>

  <group if="$(eval camera_model == 'realsense_d435')">
    <include file="$(find realsense2_camera)/launch/rs_camera.launch">
      <arg name="camera"                value="$(arg camera_namespace)"/>
      <arg name="enable_pointcloud"     value="false" />
      <arg name="serial_no"             value="$(arg serial_no)"/>
    </include>

    <node unless="$(arg use_quaternion)" pkg="tf" type="static_transform_publisher" name="$(arg camera_namespace)_to_realsense_frame"
      args="$(arg x) $(arg y) $(arg z) $(arg yaw) $(arg pitch) $(arg roll) $(arg parent_link) $(arg camera_namespace)_link 10" />

    <node if="$(arg use_quaternion)" pkg="tf" type="static_transform_publisher" name="$(arg camera_namespace)_to_realsense_frame"
      args="$(arg x) $(arg y) $(arg z) $(arg r_x) $(arg r_y) $(arg r_z) $(arg r_w) $(arg parent_link) $(arg camera_namespace)_link 10" />

    <include file="$(find ar_track_alvar)/launch/pr2_indiv_no_kinect.launch">
      <arg name="marker_size" value="$(arg user_marker_size)" />
      <arg name="max_new_marker_error" value="0.08" />
      <arg name="max_track_error" value="0.2" />
      <arg name="cam_image_topic" value="$(arg camera_namespace)/color/image_raw" />
      <arg name="cam_info_topic" value="$(arg camera_namespace)/color/camera_info" />
      <arg name="output_frame" value="$(arg camera_namespace)$(arg marker_frame_id)" />
      <arg name="node_name" value="$(arg camera_namespace)"/>
    </include>
  </group>

</launch>
```

parameter	Defaults	Function
x,y,z	0	The camera is fixed in a static position in the robotic arm
roll, pitch, yaw	0	Static pose Euler role representation with camera fixed in robotic arm
r_w,r_x,r_y,r_z	0	Quaternion representation of static pose with camera fixed in robotic arm
parent_link	wrist3_Link	The camera publishes the relative coordinate system of tf
serial_no	null	Serial number to start the camera
user_marker_size	3	The size of the label (cm)
use_quaternion	false	Whether to use quaternion to represent attitude
camera_model	realsense_d435	camera model
camera_namespace	camera	camera namespace



3.1.4 Use Gmapping to build a map

3.1.4.1 Function introduction

The gmapping function package subscribes to the depth information, IMU information, and odometry information of the robot, and completes the configuration of some necessary parameters to create and output a probability-based two-dimensional grid map. The gmapping function package is based on the open source SLAM algorithm of the openslam community.

3.1.4.2 Function operation

```
$ roslaunch agilexpro open_lidar.launch $ roslaunch
agilexpro gmapping.launch
```

After building the map, save the map in the ~/catkin_workspace/src/agilexpro/maps directory

```
$ roscd agilexpro/maps $
rosrun map_server map_saver -f map
```

> The last map is the name of the generated map. When navigating, the default is to load the map file named with the map. If you choose another name when naming, you need to modify the name of the loaded map. Change the name of the map below in the ~catkin_workspace/src/agilexpro/launch/navigation_4wd.launch file to your name.
>
> ```
> <node name="map_server" pkg="map_server" type="map_server" args="$(find agilexpro)/maps/
> map.yaml" output="screen">
> ```

3.1.5 Navigating with move_base

3.1.5.1 Function introduction

The move_base package provides an action implementation (see the actionlib package) that, given a goal in the world, will attempt to use the move Move the base to make it happen. The move_base node links the global and local planners together to accomplish its global navigation task. The move_base node also maintains two costmaps, one for the global planner and one for the local planner (see the costmap_2d package), which is used for navigation tasks.

3.1.5.2 Function operation

```
$ roslaunch agilexpro open_lidar.launch $ roslaunch
agilexpro navigation.launch
```

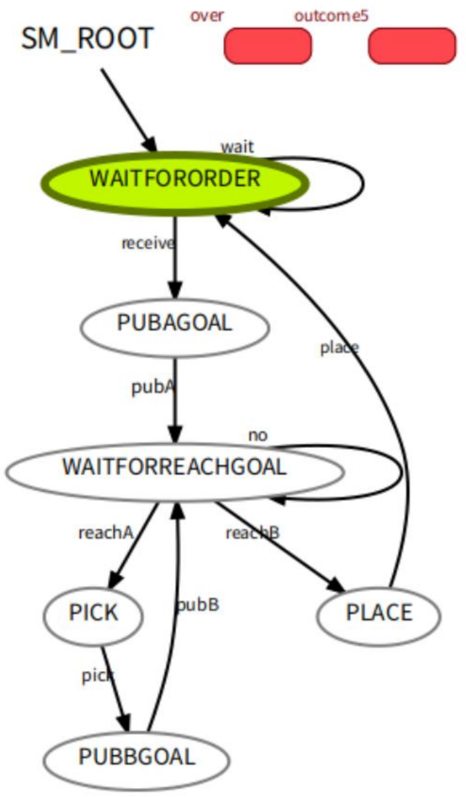
3.1.6 Use the smach library to implement task state switching

3.1.6.1 Function introduction

All possible states and state transitions are clearly described in SMACH, which is useful when the robot executes some complex plans. this base This essentially eliminates the conflict of combining different modules, allowing systems such as mobile robot manipulation to do more interesting things

3.1.6.2 Function operation

```
$ source ~/catkin_workspace/devel/setup.bash $ rosrun
agx_xarm_smach smach_demo.py $ rosrun smach_viewer
smach_viewer.py
```



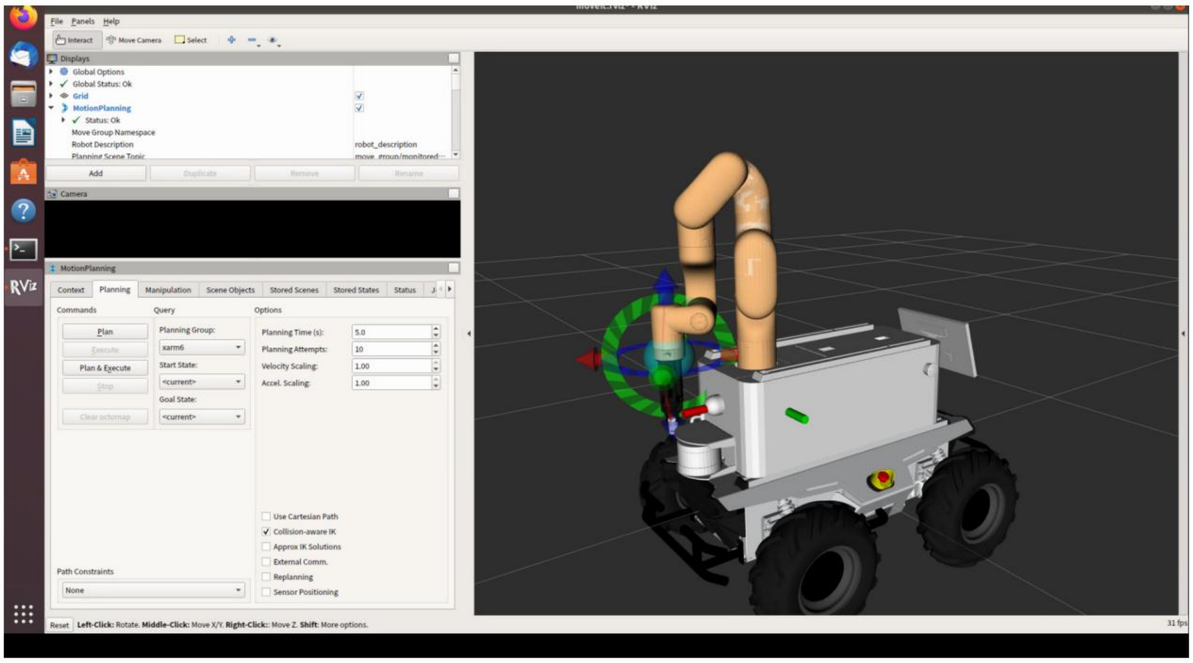
3.1.7 Use MoveIt to plan the path of the robotic arm

3.1.7.1 Function introduction

Use MoveIt's mature motion planner to perform path planning, collision detection and path execution for the motion of the robotic arm. Thus, an optimal path can be calculated when a target point is given.

3.1.7.2 Function operation

```
$ roslaunch agx_xarm_bringup setup_arm.launch
```



Robot Description: robot_description
Planning Scene Topic: move_group/monitored···
Planning Group: xarm6
Start State: <current>
Goal State: <current>
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
☐ Use Cartesian Path
☑ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning
Path Constraints: None
Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Move Z. **Shift:** More options.
31 fps

This is a complete combination model of the chassis and the robotic arm. We can drag the end of the robotic arm to perform path planning, and the robotic arm will not collide with the chassis at this time.

3.2 Function usage

3.2.1 Start the robotic arm

```
$ roslaunch agx_xarm_bringup setup_arm.launch
```

> Turn on the power of the robotic arm control cabinet and unlock the insurance of each joint of the robotic arm. Long press the power button on the teaching display screen connected to the control cabinet of the robotic arm, click Save->Start in the pop-up small window, and the electrical switch of the robotic arm is turned on.

In this launch file, one parameter can be modified, robot_ip. The incoming ip is the default ip of the robotic arm control cabinet: 192.168.1.100.

```
<launch>
        <arg name="robot_ip" default="192.168.1.198" /> <include file="$
        (find scout_xarm_moveit_config)/launch/realMove_exec.launch"> <arg name="robot_ip" value="$(arg
                robot_ip)" />
        </include>
        <node pkg="dh_gripper_driver" type="dh_gripper_joint_state"
name="dh_gripper_state" />
</launch>
```

3.2.2 Start the camera

```
$ roslaunch agx_xarm_bringup open_camera.launch
```

> Note that you need to connect the camera and the industrial computer with a USB3.0 data cable before starting.

The parameters that the launch file supports to modify are as follows:

```
        <arg name="camera_namespace" value="cam1"/> <arg
        name="serial_no" value="035422071503"/> <arg
        name="node_name" value="cam1"/> <arg
        name="use_quaternion" value ="true"/> <!--<arg name="roll"
        value="0"/> <arg name="pitch" value="-1.57"/> <arg
        name="yaw" value="- 1.57"/> <arg name="x" value="0.05"/>
        <arg name="y" value="0.03"/> <arg name="z" value="0.0"/>--
        > < arg name="x" value="0.0446414171704"/> <arg name="y"
        value="0.053996778351"/> <arg name="z"
        value="0.0600140964856"/> <arg name="r_x" value=
        "-0.000283027265062"/> <arg name="r_y"
        value="-0.0682970373998"/> <arg name="r_z"
        value="0.984025866108"/> <arg name="r_w"
        value="0.164403556559"/>
```

- camera_namespace: The namespace of the camera node. When starting multiple cameras, you may need to modify the namespace of each camera.
- serial_no: The serial number S/N code of the camera. When multiple cameras need to be started, the serial number of the camera can be passed in to specify a camera to open.
- node_name: Generally the same as the namespace.
- use_quaternion: true means use quaternion, false means use Euler role.

- x, y, z: the position of the camera from the robotic arm.
- roll, pitch, yaw: camera pose Euler.
- r_x, r_y, r_z, r_w: camera pose quaternion.

When the launch file is started, the image recognition node is also started, and the spatial coordinate information of the object and the object placement platform will be updated in real time. You can obtain these two points of information according to the following interface. The service name is /pick_point. The service type is agx_pick_msg/AgxPickSrv, and the detailed message format is as follows:

```
int32 item                # 0--bottle 1--pick 2--place # The direction of the
int32 handpose int32      gripping object 0--front 1--upside # Preparing grip point. 0--The
Prepose                   real position of the object; # 1--The direction of the object
                          according to the handpose Offset one clip position forward or up
---
#Return
value geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
geometry_msgs/Quaternion orientation float64 x

    float64 y
    float64 z
    float64 w
```

When q is 0, the coordinate information of the object is returned; when q is 2, the position information of the object is returned.

3.2.3 Start the machine

```
$ roslaunch agx_xarm_bringup open_gripper.launch
```

The topic interface for controlling the machine is /gripper/ctrl, the message type is dh_gripper_msgs/GripperCtrl, and the message format is as follows:

```
bool initialize                 #Whether to
initialize float32 position #The distance between the two clips #The
float32 force                   strength of the clipping object
float32 speed                 # Two-finger clamp This parameter is invalid, and the clamping speed is positively correlated with force
```

There is a demo file in the agx_xarm_pick function package to input the position of the fingertip to control the mechanism. Use the following instructions:

```
$ rosrun agx_xarm_pick control_gripper #q exit
```

3.2.4 Start the agx_xarm_pick node

```
$ roslaunch agx_xarm_bringup agx_xarm_bring.launch
```

The node needs to obtain the taskid passed in by the state machine to control the robot to perform different tasks respectively. The service interface of the task is /send_task, the service type is agx_pick_msg/TaskCmd, and the message format is as follows:

```
int32 taskID
geometry_msgs/PoseStamped A_goal
    std_msgs/Header header
        uint32 seq
        time stamp
        string frame_id
    geometry_msgs/Pose pose
        geometry_msgs/Point position float64 x

            float64 y
            float64 z
        geometry_msgs/Quaternion orientation float64 x

            float64 y
            float64 z
            float64 w
geometry_msgs/PoseStamped B_goal std_msgs/
    Header header
        uint32 seq time
        stamp string
        frame_id geometry_msgs/
    Pose pose geometry_msgs/Point
        position
            float64x _
            float64 y
            float64 z
        geometry_msgs/Quaternion orientation
            float64x _
            float64 y
            float64 z
            float64 w
---
bool result
```

3.2.5 Start the smach state machine node

```
$ rosrun agx_xarm_smach smach_demo.py
```

The following state transitions are defined:

```
smach.StateMachine.add('WAITFORORDER', WaitFororder(),
              transitions={'receive':'PUBAGOAL',
      'wait':'WAITFORORDER'})

smach.StateMachine.add('PUBAGOAL', PubAGoal(),
              transitions={'pubA':'WAITFORREACHGOAL'})

smach.StateMachine.add('WAITFORREACHGOAL', WaitForReachGoal(),
        transitions={'no':'WAITFORREACHGOAL', 'reachA':'PICK', 'reachB':'PLACE'})



smach.StateMachine.add('PICK', Pick(),
              transitions={'pick':'PUBBGOAL'})

smach.StateMachine.add('PUBBGOAL', PubBGoal(),
              transitions={'pubB':'WAITFORREACHGOAL'})

smach.StateMachine.add('PLACE', Place(),
transitions={'place':'WAITFORORDER'})
```

The above is the state transition logic of the entire system, and corresponding states can be added as needed. Taking the first state as an example, a "WAITFORORDER" state is defined here. The class of this state is WaitFororder, and transitions represent state transitions. If the output result of WaitFororder is receive, it will jump to "PUBAGOAL", and if the output result of the status is wait, it will jump to the "WAITFORORDER" state.

3.2.6 Start mapping

```
$ roslaunch agilexpro open_lidar.launch $ roslaunch
agilexpro gmapping.launch
```

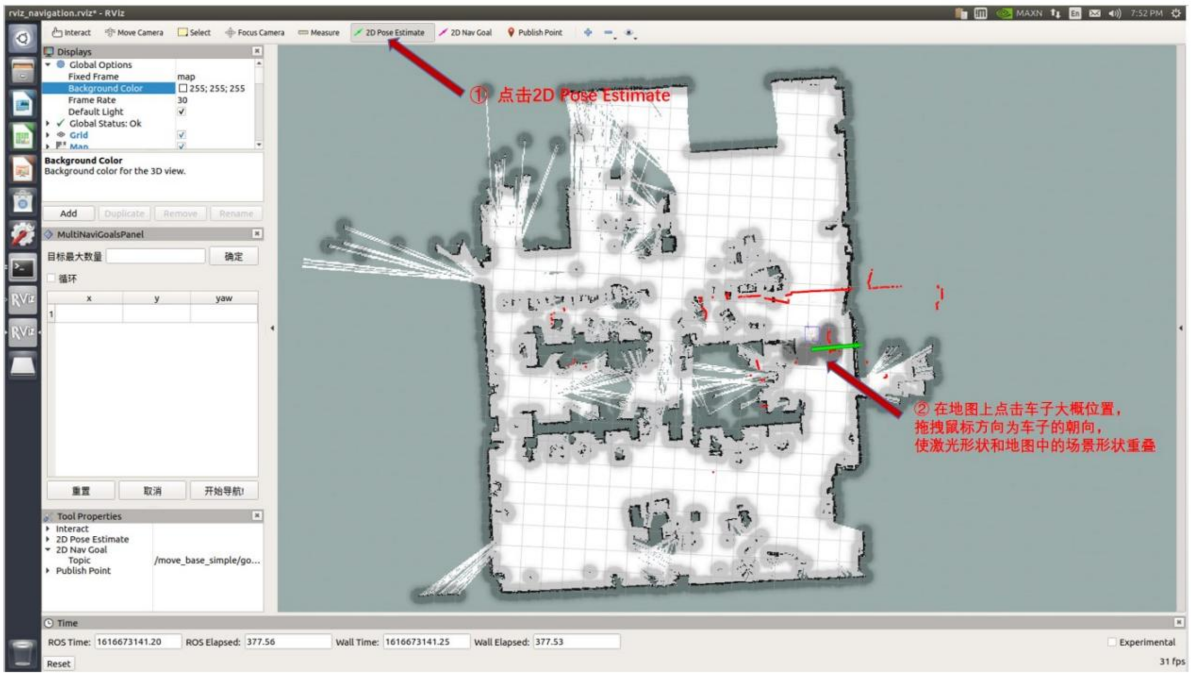
Use the remote control to remotely control the robot to create a map in the scene, and execute the following commands to save the map

```
$ roscd agilexpro/maps $
rosrun map_server map_saver -f map
```

3.2.7 Start Navigation

```
$ roslaunch agilexpro open_lidar.launch $ rosrun
scout_bringup setup_can2usb.bash #Initialize chassis communication $ roslaunch
agilexpro navigation_4wd.launch
```

When it is just started, the program defaults to the position where the map is started, and it needs to be corrected by publishing an approximate position and rotating the chassis with the handle. Correction is done when the laser shape and the scene shape in the map overlap.

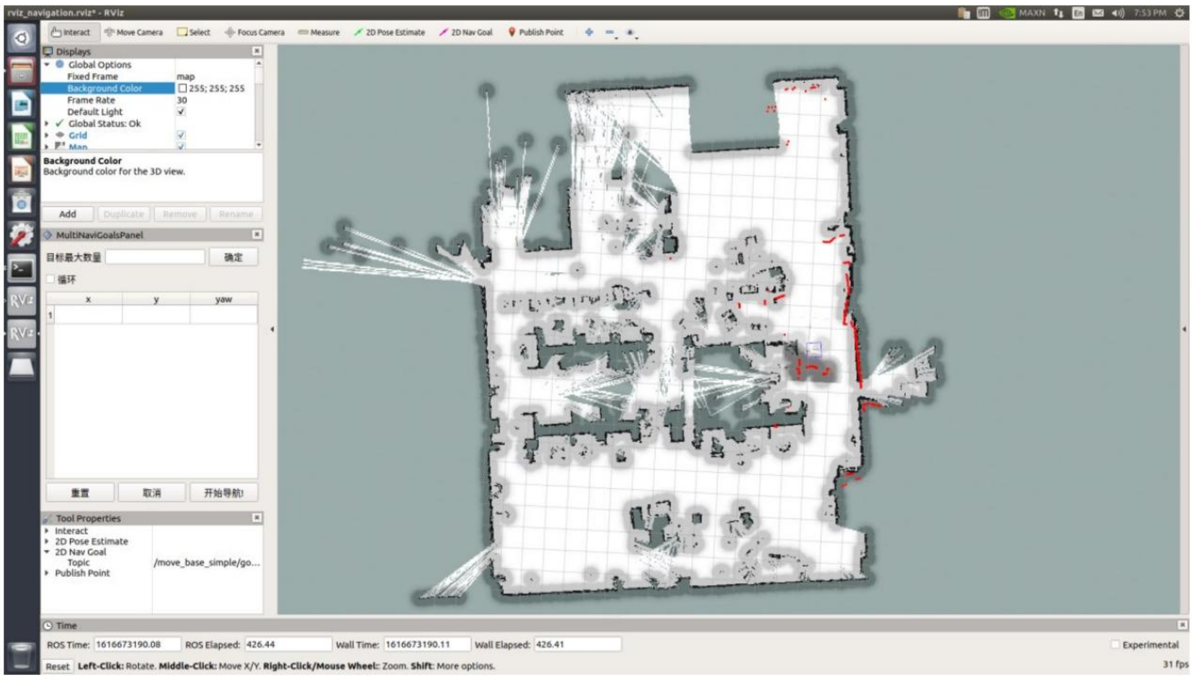


Global Options
Fixed Frame map
Background Color 255; 255; 255
Frame Rate 30
Default Light ✓
Global Status: Ok
Grid ✓
Map ✓

Background Color
Background color for the 3D view.

Topic /move_base_simple/go...

ROS Time: 1616673141.20 ROS Elapsed: 377.56 Wall Time: 1616673141.25 Wall Elapsed: 377.53



Global Options
Fixed Frame: map
Background Color: 255; 255; 255
Frame Rate: 30
Default Light: ✓
Global Status: Ok
Grid: ✓
Map: ✓

Background Color
Background color for the 3D view.

2D Nav Goal Topic: /move_base_simple/go...

ROS Time: 1616673190.08 ROS Elapsed: 426.44 Wall Time: 1616673190.11 Wall Elapsed: 426.41

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

The topic interface of a certain point of navigation is /move_base_simple/goal, the message type is geometry_msgs/PoseStamped, and the message format is as follows:

```
std_msgs/Header header
    uint32 seq time
    stamp string
    frame_id geometry_msgs/
Pose pose geometry_msgs/Point
    position
        float64x _
        float64 y
        float64 z
    geometry_msgs/Quaternion orientation float64 x

        float64 y
        float64 z
        float64 w
```

The feedback information after reaching a certain point can be obtained from the /move_base/result topic, the message type is base_msgs/MoveBaseActionResult, and the message format is as follows:

```
std_msgs/Header header
    uint32 seq time
    stamp string
    frame_id
actionlib_msgs/GoalStatus status
    uint8 PENDING=0
    uint8 ACTIVE=1
    uint8 PREEMPTED=2
    uint8 SUCCEEDED=3
    uint8 ABORTED=4
    uint8 REJECTED=5
    uint8 PREEMPTING=6
    uint8 RECALLING=7
    uint8 RECALLED=8
    uint8 LOST=9
    actionlib_msgs/GoalID goal_id
        time stamp
        string id
    uint8 status
    string text
move_base_msgs/MoveBaseResult result
```

When the robot reaches the target point, you can check that the status is 3.

3.2.8 Record the coordinate position of two points

```
$ roscd agx_xarm_pick/config/
$ rosrun agx_xarm_pick record
```

It needs to cooperate with step 7, and move the small control to the specified position, such as the position in front of the workpiece, about 1m. As prompted, press

Record the positions of the two points next time.

3.2.9 Mobile grab demo

Set two locations in the scene as a grip point and a placement point. After recording the coordinates of the two locations in step 8, turn on the

order.

```
$ roslaunch agx_xarm_bringup setup_arm.launch $ roslaunch                #Start the robotic arm
agx_xarm_bringup open_camera.launch $ roslaunch agx_xarm_bringup         #start camera
open_gripper.launch $ roslaunch agx_xarm_bringup                         #start clip
agx_xarm_bring.launch $ rosrun agx_xarm_smach smach_demo.py $            #Start moving the gripper node
roslaunch agilexpro open_lidar.launch $ roslaunch agilexpro              #Start state machine
navigation_4wd.launch $ roslaunch agx_xarm_pick test.launch              #start radar
                                                                         #Start navigation
                                                                         #Start executing the task
```

Switch the remote control to command control mode, and the composite mobile robot starts to navigate to the first location to grab something.

3.2.10 In- situ Grab and Place Demonstration

Label the grabbed object and place it in a suitable position for the robot. A demo that can be grabbed and placed by executing the following commands

```
$ roslaunch agx_xarm_bringup set_setup_arm.launch
$ roslaunch agx_xarm_bringup open_camera.launch
$ roslaunch agx_xarm_bringup open_gripper.launch
$ rosrun agx_xarm_pick demo_pick $ rosrun                  #grab the object
agx_xarm_pick demo_place                                   # place objects
```

4 Handling and Answers to Frequently Asked Questions

5 Other instructions

1. xArm User
Manual 2. xArm GitHub
3. SCOUT User
Manual 4. AgileX
GitHub 5. RoboSense-LiDAR