

1. Multi-machine handle control

1. Multi-machine handle control

1.1. Multi-machine configuration

1.1.1. Multi-machine communication settings

1.1.2. Multi-machine time synchronization

1.2. Use

1.2.1. Start the robot

1.2.2. Turn on handle control

1.3. Handle control analysis

1.3.1. Control multiple robots

1.3.2. Control a robot

1.1. Multi-machine configuration

When using multi-machine handle control, you first need to ensure that the robots are under the same LAN and configured with the same [ROS_MASTER_URI]; multiple robots can only have one host to control their movements. In the case of this section, the virtual machine is set as the host machine, and other robots are slave machines. There are several slave machines. Of course, you can also set a certain robot as the master machine and the others as slave machines.

1.1.1. Multi-machine communication settings

Check the IP of the virtual machine

```
ifconfig
```

```
yahboom@VM:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.106 netmask 255.255.255.0 broadcast 192.168.2.255
    inet6 fe80::f932:2481:31f4:a257 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:bc:8a:06 txqueuelen 1000 (Ethernet)
    RX packets 412408 bytes 87569068 (87.5 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 146320 bytes 190425290 (190.4 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 44762 bytes 30920047 (30.9 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 44762 bytes 30920047 (30.9 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
#Raspberry Pi 5 master needs to enter docker first, please perform this step
#If running the script into docker fails, please refer to ROS/07, Docker tutorial
~/run_docker.sh
```

Next, you only need to modify the .bashrc file of the slave machine (Robot side). There are several slave machines and several configurations.

```
sudo vim ~/.bashrc
```

Find the following line of content

```
export ROS_MASTER_URI=http://IP:11311
```

The [IP] here uses the IP (virtual machine side).

```
export ROS_MASTER_URI=http://192.168.2.106:11311
```

After setting the IP, it is best to refresh the environment variables.

```
source ~/.bashrc
```

1.1.2. Multi-machine time synchronization

Install

```
sudo apt install ntp ntpdate
```

time view command

```
date -R
```

When there is a network, the system will automatically synchronize the network system time without setting. If there is no network, the time needs to be set when the time between robots is different.

- Internet available

Automatically synchronize network time command

```
ntpd pool ntp.ubuntu.com
```

- No network

The test case is using two robots.

Command to manually set time

```
sudo date -s "2020-01-1 01:01:01"
```

1), Server-side (host-side) configuration

Open the [ntp.conf] file

```
sudo vim /etc/ntp.conf
```

Add at the end of the file

```
restrict 192.168.2.0 mask 255.255.255.0 nomodify notrap  
server 127.127.1.0 # local clock  
fudge 127.127.1.0 stratum 10
```

The first line is to enable the machine on the 192.168.2.xxx network segment to synchronize time with this machine (specifically, it depends on whether your IP is 192.168.2.xxx. If it is different, change it to your actual format)

The second and third lines are for time synchronization between the local hardware time and the local ntp service.

After making the changes, restart the ntp service

```
sudo /etc/init.d/ntp restart
```

The server-side settings are completed, let's start the client-side settings.

2. Client (slave side) configuration

Open the [ntp.conf] file

```
sudo vim /etc/ntp.conf
```

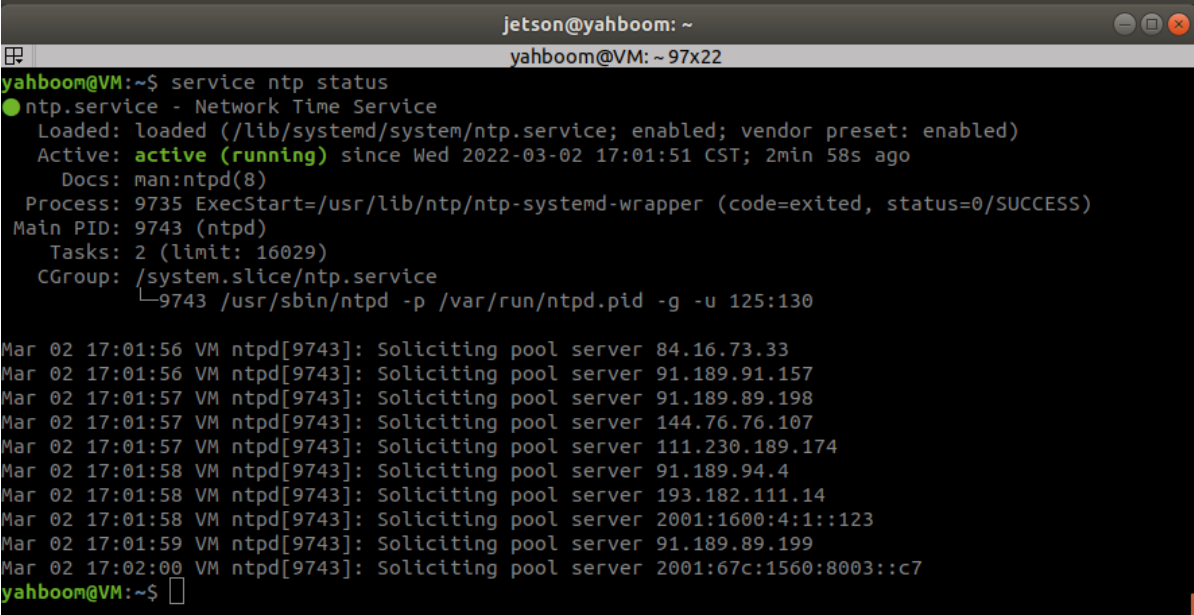
Add [server]+[IP] at the end of the file

```
server 192.168.2.106
```

3), time synchronization

On the [server side], execute the following code to check whether the ntp service of the [server side] is running.

```
service ntp status
```



```
jetson@yahboom: ~
yahboom@VM: ~ 97x22
yahboom@VM:~$ service ntp status
ntp.service - Network Time Service
   Loaded: loaded (/lib/systemd/system/ntp.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2022-03-02 17:01:51 CST; 2min 58s ago
     Docs: man:ntpd(8)
  Process: 9735 ExecStart=/usr/lib/ntp/ntp-systemd-wrapper (code=exited, status=0/SUCCESS)
 Main PID: 9743 (ntpd)
    Tasks: 2 (limit: 16029)
   CGroup: /system.slice/ntp.service
           └─9743 /usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 125:130

Mar 02 17:01:56 VM ntpd[9743]: Soliciting pool server 84.16.73.33
Mar 02 17:01:56 VM ntpd[9743]: Soliciting pool server 91.189.91.157
Mar 02 17:01:57 VM ntpd[9743]: Soliciting pool server 91.189.89.198
Mar 02 17:01:57 VM ntpd[9743]: Soliciting pool server 144.76.76.107
Mar 02 17:01:57 VM ntpd[9743]: Soliciting pool server 111.230.189.174
Mar 02 17:01:58 VM ntpd[9743]: Soliciting pool server 91.189.94.4
Mar 02 17:01:58 VM ntpd[9743]: Soliciting pool server 193.182.111.14
Mar 02 17:01:58 VM ntpd[9743]: Soliciting pool server 2001:1600:4:1::123
Mar 02 17:01:59 VM ntpd[9743]: Soliciting pool server 91.189.89.199
Mar 02 17:02:00 VM ntpd[9743]: Soliciting pool server 2001:67c:1560:8003::c7
yahboom@VM:~$
```

If it is not started, restart the ntp service.

Network updates (required for all devices)

```
sudo /etc/init.d/networking restart
```

Time synchronization

```
sudo ntpdate 192.168.2.106
```

If the client's NTP is also enabled, you need to close the NTP service first and then perform time synchronization.

```
sudo /etc/init.d/ntp stop
```

1.2. Use

Take the virtual machine as the host machine and the two robots as slave machines as an example.

1.2.1. Start the robot

Virtual machine side

```
roscore
```

Start the command (robot1 side). For the convenience of operation, this section takes [mono + laser + yahboomcar] as an example.

```
roslaunch yahboomcar_multi laser_bringup_multi.launch ns:=robot1 # laser +  
yahboomcar  
roslaunch yahboomcar_multi laser_usb_bringup_multi.launch ns:=robot1 # mono +  
laser + yahboomcar  
roslaunch yahboomcar_multi laser_astapro_bringup_multi.launch ns:=robot1 # Astra  
+ laser + yahboomcar
```

Start the command (robot2 side). For the convenience of operation, this section takes [mono + laser + yahboomcar] as an example.

```
roslaunch yahboomcar_multi laser_bringup_multi.launch ns:=robot2 # laser +  
yahboomcar  
roslaunch yahboomcar_multi laser_usb_bringup_multi.launch ns:=robot2 # mono +  
laser + yahboomcar  
roslaunch yahboomcar_multi laser_astapro_bringup_multi.launch ns:=robot2 # Astra  
+ laser + yahboomcar
```

More robots and so on.

1.2.2. Turn on handle control

Method 1: One controller controls multiple robots at the same time

Connect the controller receiver to the USB port of the virtual machine and start the command

```
roslaunch yahboomcar_multi joy_multi.launch
```

Note: The [launch] file defaults to three robots.

joy_multi.launch

```
<launch>
  <arg name="first_robot1" default="robot1"/>
  <arg name="second_robot2" default="robot2"/>
  <arg name="third_robot3" default="robot3"/>
  <param name="use_sim_time" value="false"/>
  <node name="joy_node" pkg="joy" type="joy_node" output="screen"
respawn="false"/>
  <!-- ##### first_robot1 #####
##### -->
  <include file="$(find yahboomcar_multi)/launch/library/joy_base.launch">
    <arg name="ns" default="$(arg first_robot1)"/>
  </include>
  <!-- ##### second_robot2 #####
##### -->
  <include file="$(find yahboomcar_multi)/launch/library/joy_base.launch">
    <arg name="ns" default="$(arg second_robot2)"/>
  </include>
  <!-- ##### third_robot3 #####
##### -->
  <include file="$(find yahboomcar_multi)/launch/library/joy_base.launch">
    <arg name="ns" default="$(arg third_robot3)"/>
  </include>
</launch>
```

If there are 2 robots, just comment out the part of the third robot. If you don't comment it out, it will not have any effect. If there are 4 robots or more, you can add it according to the samples of the first 3 robots.

Method 2: One handle controls one robot alone

<PI5 needs to open another terminal to enter the same docker container

1. In the above steps, a docker container has been opened. You can open another terminal on the host (car) to view:

```
docker ps -a
```

```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS        NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9   "/bin/bash"            3 days ago    Up 9 hours                   ecstatic_lewin
jetson@ubuntu:~$
```

2. Now enter the docker container in the newly opened terminal:

```
docker exec -it 5b698ea10535 /bin/bash
```

```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS        NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9   "/bin/bash"            3 days ago    Up 9 hours                   ecstatic_lewin
jetson@ubuntu:~$ docker exec -it 5b698ea10535 /bin/bash
-----
my_robot_type: x3 | my_lidar: a1 | my_camera: astrapro
-----
root@ubuntu:/#
```

After successfully entering the container, you can open countless terminals to enter the container.

Connect the handle receiver to the corresponding robot, and start the command on the corresponding robot side (take robot1 as an example)

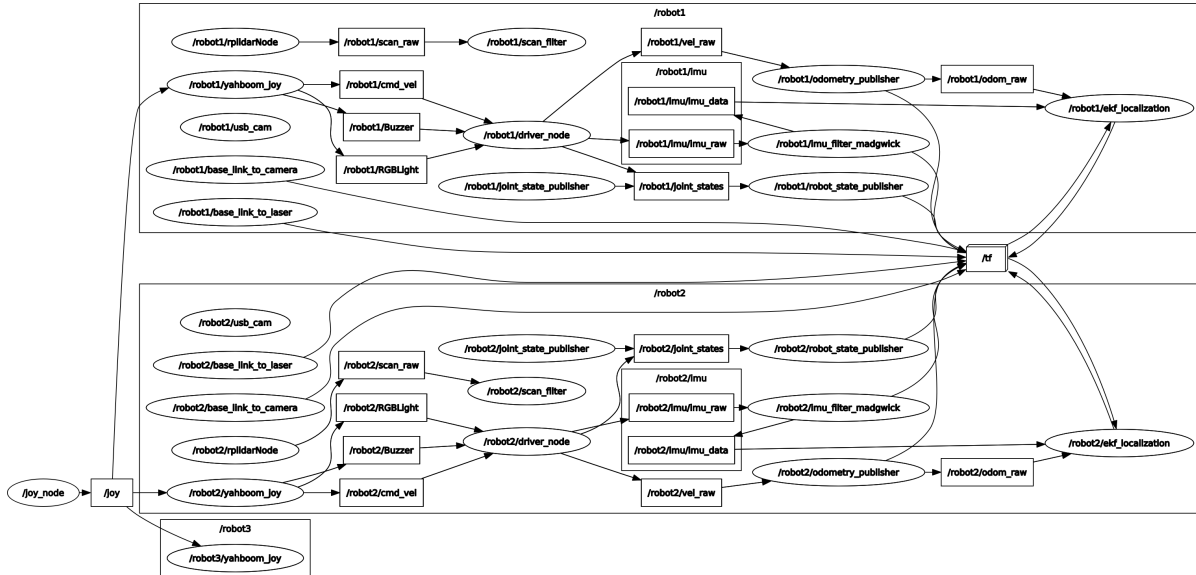
```
roslaunch yahboomcar_multi joy_each.launch ns:=robot1
```

1.3. Handle control analysis

Node view

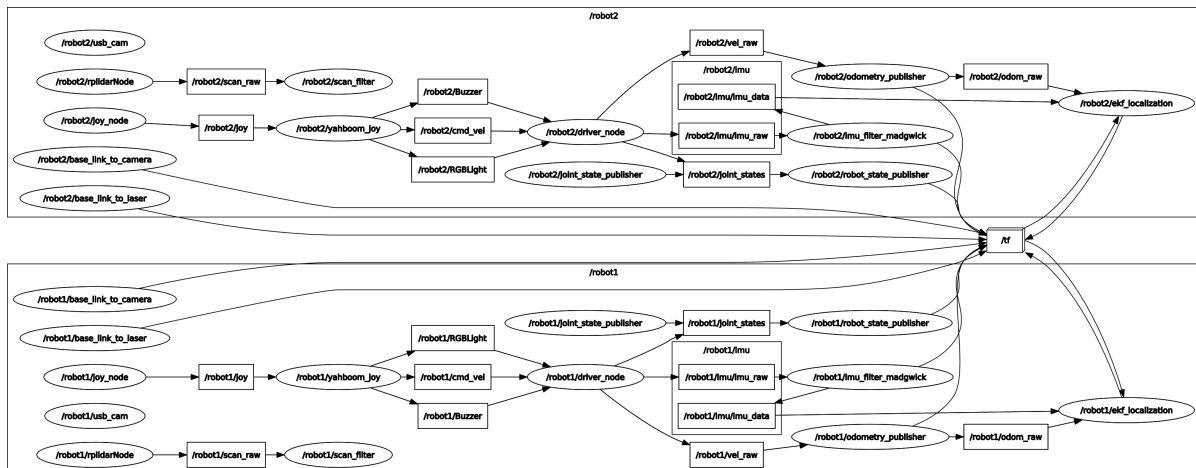
rqt_graph

1.3.1. Control multiple robots



It can be seen from the figure that when one controller controls multiple robots at the same time, only one [joy_node] node is needed to receive the controller signal, and different [yahboom_joy] nodes are set for different robots to control the corresponding robots.

1.3.2. Control a robot



It can be seen from the figure that when a handle controls a robot, each robot must be set and controlled separately.