Multi-car synchronous control

1. Multi-car synchronous control

1.1. Program function description

Connect the handle to the virtual machine, set the virtual machine to connect to the handle, and after the program on the virtual machine and the car is started, the joystick of the handle can be used to control the two cars at the same time, and the movements of the two cars are synchronized.

1.2, Program reference path

After SSH connects to the car, the location of the function source code is,

```
/home/sunrise/yahboomcar_ws/src/yahboomcar_multi/launch/yahboomcar_bringup_multi_
ctrl.launch.xml
```

The virtual machine control code is located at,

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_multi/launch/multi_ctrl.launch.xml
```

1.3, Program startup

Note: Here you need to set up multi-machine communication in advance, that is, the virtual machine and the car (taking two cars as an example) need to be in the **same LAN** and the **ROS_DOMAIN_ID** must be the same, in order to be able to perform distributed communication.

Input in the virtual machine terminal,

```
ros2 launch yahboomcar_multi multi_ctrl.launch.xml
```

After SSH connects to car 1, input in the terminal,

```
ros2 launch yahboomcar_multi yahboomcar_bringup_multi_ctrl.launch.xml
robot_name:=robot1
```

After SSH connects to car 2, input in the terminal,

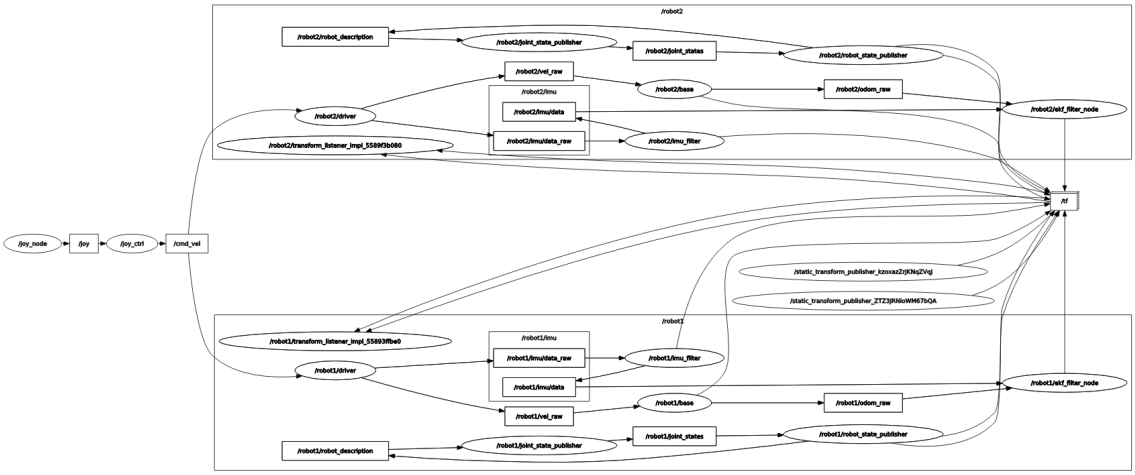
```
ros2 launch yahboomcar_multi yahboomcar_bringup_multi_ctrl.launch.xml
robot_name:=robot2
```

After the program starts, press L1 to enter control.

1.4. View the node communication graph

Virtual machine terminal input,

```
ros2 run rqt_graph rqt_graph
```



It can be seen that the virtual machine published the /cmd_vel topic speed, and the chassis of robot1 and robot2 subscribed to the topic, so when the virtual machine publishes the topic data, both cars will receive it and then control the movement of their respective chassis.

1.5、Core code analysis

The code on the virtual machine side is the same as the handle control code on the car side, so I won't go into details here. Let's mainly look at the content on the car side and the startup file **yahboomcar_bringup_multi_ctrl.launch.xml**,

```
<launch>
<arg name="robot_name" default="robot1"/>
<group>
<push-ros-namespace namespace="$(var robot_name)"/>
<!--driver_node-->
<node name="driver" pkg="yahboomcar_bringup" exec="Mcnamu_driver"
output="screen">
<param name="imu_link" value="$(var robot_name)/imu_link"/>
<remap from="cmd_vel" to="/cmd_vel"/>
</node>
<!--base_node-->
<node name="base" pkg="yahboomcar_base_node" exec="base_node" output="screen">
<param name="odom_frame" value="$(var robot_name)/odom"/> <param
name="base_footprint_frame" value="$(var robot_name)/base_footprint"/> </node>
<!--imu_filter_node--> <node name="imu_filter" pkg="imu_filter_madgwick"
exec="imu_filter_madgwick_node" output="screen"> <param name="fixed_frame"
value="$(var robot_name)/base_link"/> <param name="use_mag" value="false"/>
<param name="publish_tf" value="false"/> <param name="world_frame" value="$(var
robot_name)/enu"/> <param name="orientation_stddev" value="0.05"/> </node> <!--
ekf_node--> <node name="ekf_filter_node" pkg="robot_localization"
exec="ekf_node"> <param name="odom_frame" value="$(var robot_name)/odom"/> <param
name="base_link_frame" value="$(var robot_name)/base_footprint"/> <param
name="world_frame" value="$(var robot_name)/odom"/> <param from="$(find-pkg-share
yahboomcar_multi)/param/ekf_$(var robot_name).yaml"/> <remap
from="odometry/filtered" to="odom"/> <remap from="/odom_raw" to="odom_raw"/>
</node> </group> <include file="$(find-pkg-share
yahboomcar_description)/launch/description_multi_$(var robot_name).launch.py"/>
</launch>
```

Here we use the XML format to write the launch file, so that we can add namespaces in front of multiple nodes. Adding namespaces is to solve the conflict caused by the same node name. We have two cars here. Take the chassis program as an example. If both chassis nodes are named driver, then after establishing multi-machine communication, this is not allowed. So we add the namespace **namespace** in front of the node name. In this way, the chassis node name of car 1 is /robot1/driver, and the chassis node program of car 2 is /robot2/driver. In the launch file in XML format, a group group is used to specify that in this group, both the node name and the topic name need to be added with **namespace**.

```
<group>
<push-ros-namespace namespace="$(var robot_name)"/>
</group>
```

The parameters defined in the launch file in XML format are **$(var robot_name)**. The parameter passed in here is robot_name. When entering the command, you can enter it in the command line.

```
ros2 launch yahboomcar_multi yahboomcar_bringup_multi_ctrl.launch.xml
robot_name:=robot1
```

One thing to note here is that after adding the namespace, we remap the topic name of /robot1/cmd_vel to /cmd_vel, which is to receive the topic message sent by the virtual machine.

```
<node name="driver" pkg="yahboomcar_bringup" exec="Mcnamu_driver"
output="screen">
<param name="imu_link" value="$(var robot_name)/imu_link"/>
<remap from="cmd_vel" to="/cmd_vel"/>
</node>
```

Regarding the parameter input, plus the namespace, it is necessary to explain it in the launch file, such as the following parameters,

```
<node name="base" pkg="yahboomcar_base_node" exec="base_node" output="screen">
<param name="odom_frame" value="$(var robot_name)/odom"/>
<param name="base_footprint_frame" value="$(var robot_name)/base_footprint"/>
</node>
```

It can be seen that the odom_frame parameter is assigned to $(varrobot_name)/odom, thebase_footprint_frameparameterisassignedto$(var robot_name)/base_footprint.

If some parameters do not need to be added with a namespace, then they are passed in as a parameter table, for example,

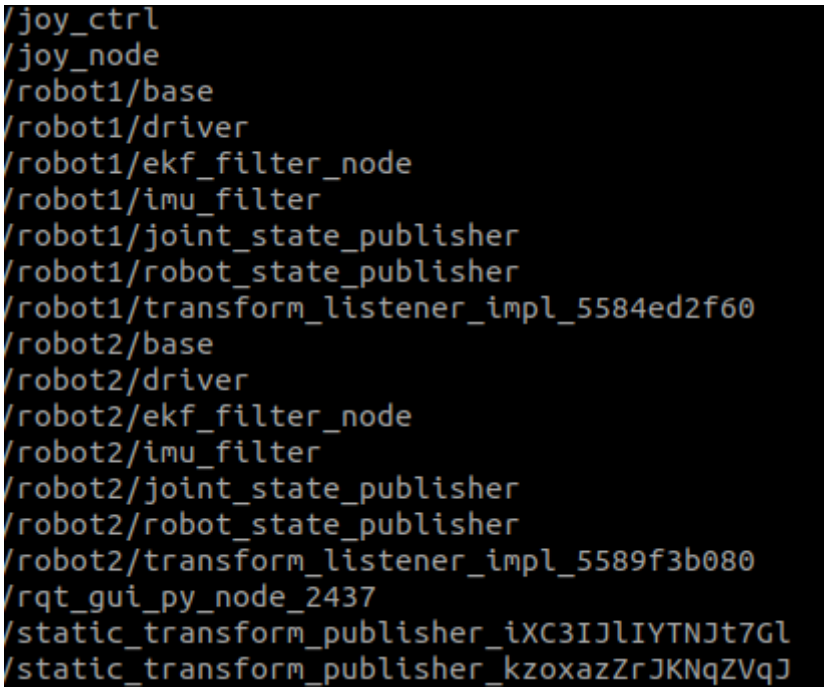
```
<param from="$(find-pkg-share yahboomcar_multi)/param/ekf_$(var
robot_name).yaml"/>
```

It should be noted that this parameter table needs to add the namespace to be able to find it accurately. If we start robot1, the node started becomes robot1/ekf_filter_node, so the beginning of the parameter file should be,

```
### ekf config file ###
robot1/ekf_filter_node:
ros__parameters:
```

View the node list, enter in the virtual machine terminal,

```
ros2 node list
```



To view the topic list, enter the virtual machine terminal,

```
ros2 topic list
```

```
yahboom@VM:~/Desktop$ ros2 topic list
/Buzzer
/JoyState
/RGBLight
/cmd_vel
/diagnostics
/joy
/joy/set_feedback
/parameter_events
/robot1/Buzzer
/robot1/RGBLight
/robot1/edition
/robot1/imu/data
/robot1/imu/data_raw
/robot1/imu/mag
/robot1/joint_states
/robot1/odom
/robot1/odom_raw
/robot1/robot_description
/robot1/set_pose
/robot1/vel_raw
/robot1/voltage
/robot2/Buzzer
/robot2/RGBLight
/robot2/edition
/robot2/imu/data
/robot2/imu/data_raw
/robot2/imu/mag
/robot2/joint_states
/robot2/odom
/robot2/odom_raw
/robot2/robot_description
/robot2/set_pose
/robot2/vel_raw
/robot2/voltage
/rosout
/tf
/tf_static
```