

## 5. Control Muto color tracking

Before running this program, you need to bind the port number of the voice board and the port number of the ROS expansion board on the host machine. You can refer to the previous chapter for binding;

When entering the docker container, you need to mount the voice board to recognize the voice board in the docker container.

### 1. Program function description

After the program is started, say "Hello, yahboom" to the module. The module replies "Yes" to wake up the voice board, and then you can tell it to start tracking any color of red/green/blue/yellow. After receiving the command, the program recognizes the color, loads the processed image, and then presses the R2 button on the handle to start the tracking program. Muto will track the recognized color. When the recognized color moves slowly, Muto will also track the movement.

### 2. Program code reference path

After entering the docker container, the source code of this function is located at

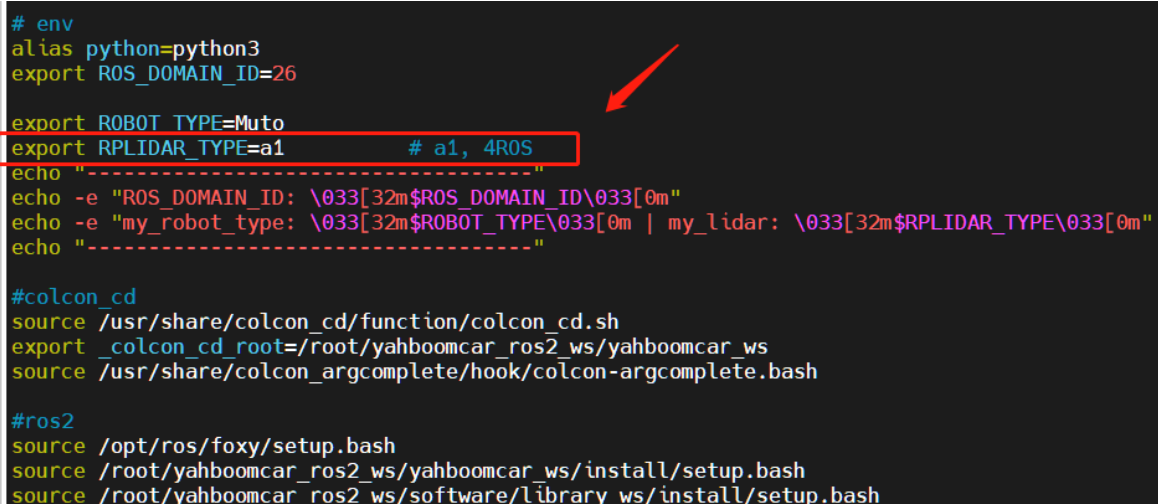
```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_voice_ctrl/yahboomcar_voice_ctrl/Voice_Ctrl_colorTracker.py  
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_voice_ctrl/yahboomcar_voice_ctrl/Voice_Ctrl_colorHSV.py
```

### 3. Configuration before use

Note: Since the Muto series robots are equipped with multiple lidar devices, the factory system has been configured with routines for multiple devices. However, since the product cannot be automatically recognized, the lidar model needs to be manually set.

After entering the container: Make the following modifications according to the lidar type:

```
root@ubuntu:/# cd  
root@ubuntu:~# vim .bashrc
```



```
# env  
alias python=python3  
export ROS_DOMAIN_ID=26  
  
export ROBOT_TYPE=Muto  
export RPLIDAR_TYPE=a1 # a1, 4ROS  
echo "-----"  
echo -e "ROS_DOMAIN_ID: \033[32m$ROS_DOMAIN_ID\033[0m"  
echo -e "my_robot_type: \033[32m$ROBOT_TYPE\033[0m | my_lidar: \033[32m$RPLIDAR_TYPE\033[0m"  
echo "-----"  
  
#colcon_cd  
source /usr/share/colcon_cd/function/colcon_cd.sh  
export _colcon_cd_root=/root/yahboomcar_ros2_ws/yahboomcar_ws  
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash  
  
#ros2  
source /opt/ros/foxy/setup.bash  
source /root/yahboomcar_ros2_ws/yahboomcar_ws/install/setup.bash  
source /root/yahboomcar_ros2_ws/software/library_ws/install/setup.bash
```

After the modification is completed, save and exit vim, and then execute:

```
root@jetson-desktop:~# source .bashrc
-----
ROS_DOMAIN_ID: 26
my_robot_type: Muto | my_lidar: a1
-----
root@jetson-desktop:~#
```

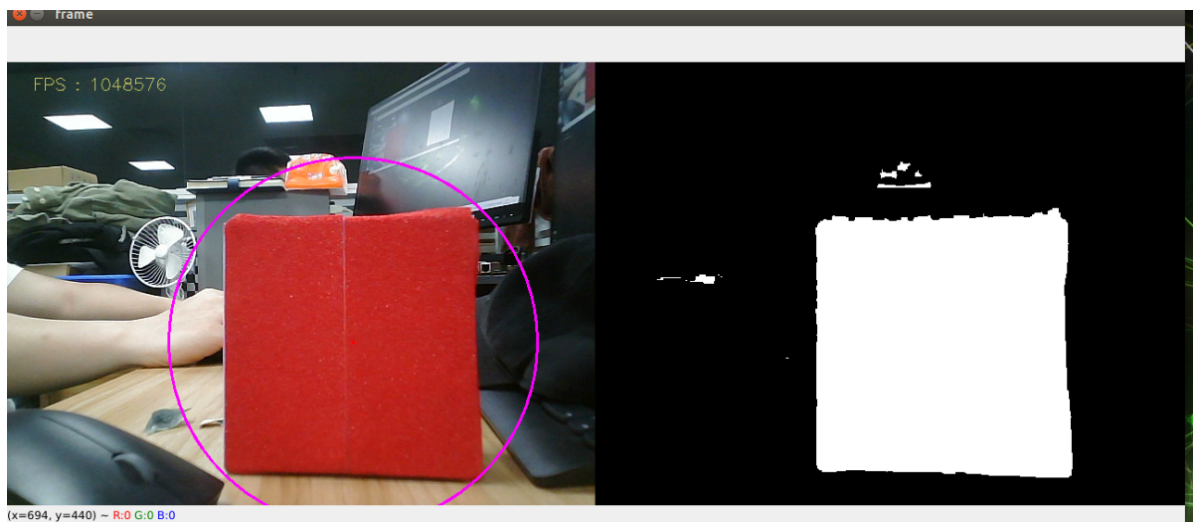
You can see the current modified lidar type.

## 4. Program startup

### 4.1. Start command

After entering the docker container, enter in the terminal

```
#Start the Voice Control Color Tracking Node
ros2 launch yahboomcar_voice_ctrl voice_ctrl_color_tracker_launch.py
#Start the depth camera and obtain depth images
ros2 launch astra_camera astra.launch.xml
```

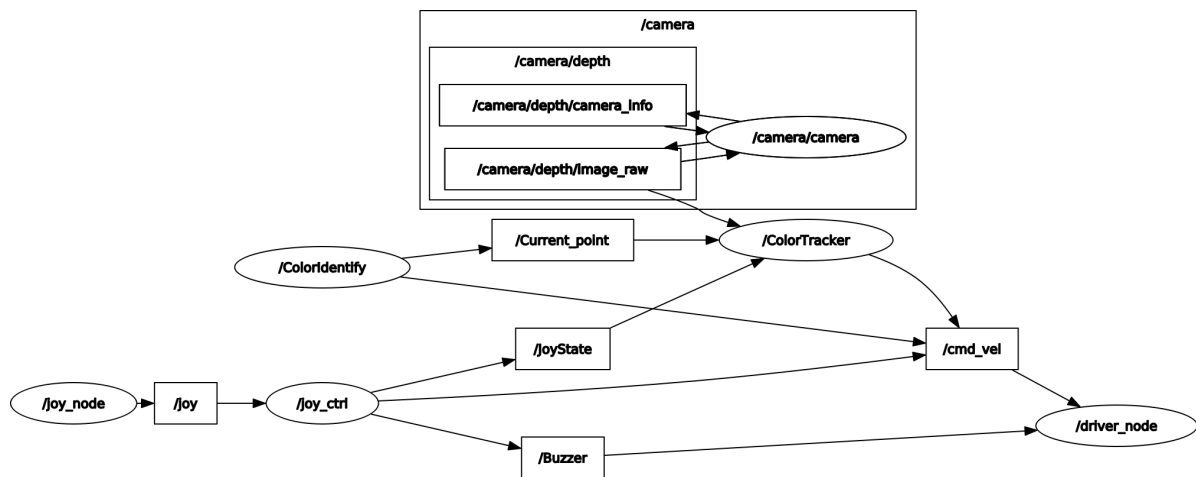


Take tracking red as an example. After waking up the module, tell it "start tracking red" and press the R2 button on the remote control. After the program receives the command, it starts processing the image. Then calculate the center coordinates of the red object, publish the center coordinates of the object; combine the depth information provided by the depth camera, calculate the speed, and finally publish it to drive Muto.

### 4.2. Node topic communication diagram

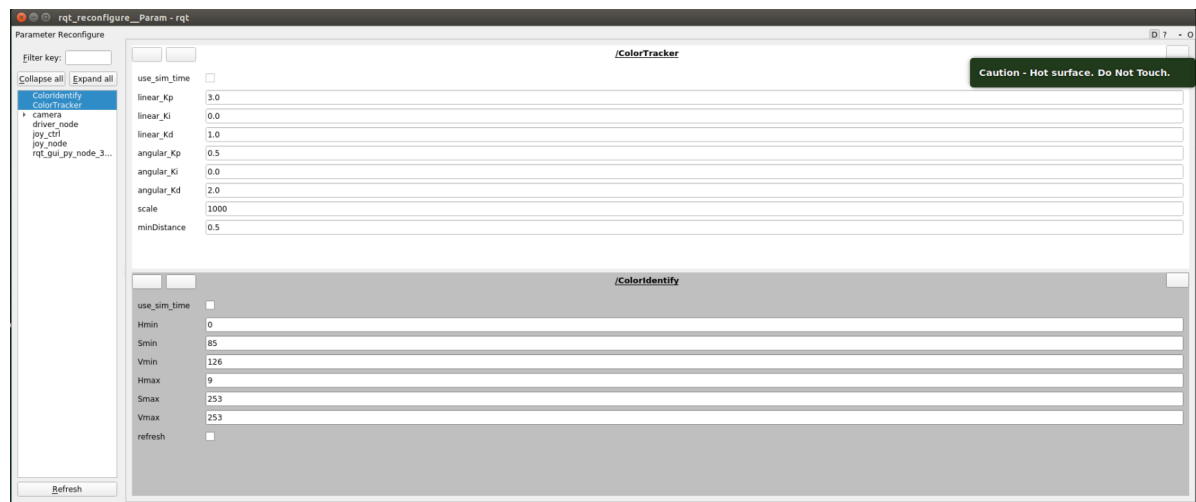
docker terminal input,

```
ros2 run rqt_graph rqt_graph
```



You can also use dynamic parameter adjuster, modify parameters, and docker terminal input,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



After modifying the parameters, click on a blank space in the GUI to write the parameter value. As can be seen from the above figure,

- colorHSV is mainly responsible for image processing and can adjust the value of HSV;
- colorTracker is mainly responsible for calculating speed and can adjust speed and distance related parameters.

The meaning of each parameter is as follows:

colorHSV

Parameter name	Parameter meaning
Hmin	H minimum value
Smin	S minimum value
Vmin	V minimum value
Hmax	H maximum value
Smax	S maximum value
Vmax	V maximum value
refresh	Refresh data into the program

colorTracker

Parameter name	Parameter meaning
linear_Kp	Linear speed P value
linear_Ki	Linear speed i value
linear_Kd	Linear speed d value
angular_Kp	Angular speed P value
angular_Kp	Angular speed i value
angular_Kp	Angular speed value
scale	Proportional coefficient
minDistance	Tracking distance

## 5. Core code

### 5.1.colorHSV

This part mainly analyzes voice commands, image processing, and finally releases the center coordinates.

```
#Define a publisher to publish the center coordinates of the detected object
self.pub_position = self.create_publisher(Position, "/Current_point", 10)
#Import speech driver library
from Speech_Lib import Speech
#Create voice control objects
self.spe = Speech()
#The following is to obtain the instruction, then judge the recognition result,
and load the corresponding HSV value.
command_result = self.spe.speech_read()
self.spe.void_write(command_result)
if command_result == 73 :
self.model = "color_follow_line"
print("tracker red")
```

```

self.hsv_range = [(0, 175, 149), (180, 253, 255)]
#Process the image, calculate the center coordinates of the detected object,
enter the execute function, and publish the center coordinates
rgb_img, binary, self.circle = self.color.object_follow(rgb_img, self.hsv_range)
if self.circle[2] != 0: threading.Thread(
    target=self.execute, args=(self.circle[0], self.circle[1],
self.circle[2])).start()
if self.point_pose[0] != 0 and self.point_pose[1] != 0: threading.Thread(
    target=self.execute, args=(self.point_pose[0],
self.point_pose[1], self.point_pose[2])).start()

```

## 5.2.colorTracker

This part receives the topic data and depth data of the center coordinates, then calculates the speed and publishes it to the chassis.

```

#Define a subscriber to subscribe to depth information
self.sub_depth = self.create_subscription(Image, "/camera/depth/image_raw",
self.depth_img_Callback, 1)
#Define a subscriber to subscribe to center coordinate information
self.sub_position =
self.create_subscription(Position, "/Current_point", self.positionCallback, 1)
#callback function
def positionCallback(self, msg) #Get the center coordinate value
def depth_img_Callback(self, msg) #Get depth information
#Pass in the X value of the center coordinates and the depth information,
calculate the speed and publish it to the chassis
def execute(self, point_x, dist)

```

Combined with the node communication diagram of 3.2, understanding the source code will be clearer.