# 5. Voice-controlled robotic arm garbage identification and sorting

## 5.1. Function description

Through interaction with the voice module, rosmaster can identify and classify the junk pictures on the prescription blocks and broadcast the recognition and classification results by voice; then the robot arm will pick up the identified blocks, navigate to the set area autonomously, and put them down. , and finally returns to the set origin.

## 5.2. Start

```
#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
```

**Note: Due to the processing speed of nano master, loading model data and image processing will be a little stuck but it can still run successfully. NX master will be better.**

### 5.2.1. Function package path

```
~/yahboomcar_ws/src/garbage_identify_yolov5/
```

### 5.2.2. Start

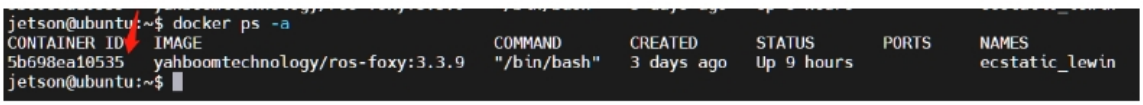
Note: **Multi-machine communication configuration** needs to be implemented between rosmaster X3Plus and the virtual machine. You can view the content of **"yahboomcar Course\06, Linux Operating System\04. Multi-machine Communication Configuration"** for configuration.

```
roslaunch yahboomcar_voice_ctrl voice_transport_base.launch
```

**<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
```



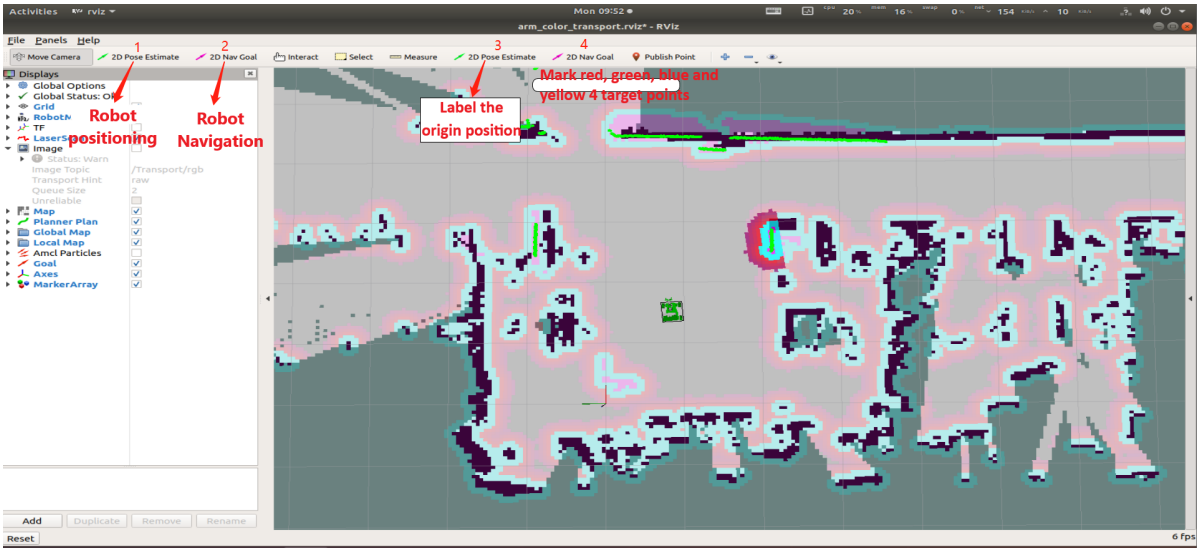

2. Now enter the docker container in the newly opened terminal:

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


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

```
#(Other master)rosmaster X3Plus startup
python3 ~/yahboomcar_ws/src/garbage_identify_yolov5/garbage_identify_yolov5.py
#(pi5)rosmaster X3Plus starts
python3.8 ~/yahboomcar_ws/src/garbage_identify_yolov5/garbage_identify_yolov5.py
#Virtual machine startup
roslaunch arm_color_transport transport_rviz.launch
```

Each tool annotation on the virtual machine rviz is as shown in the figure below,




- step 1

  In rviz, use tool 1 to calibrate and position the rosmaster;

- Step 2

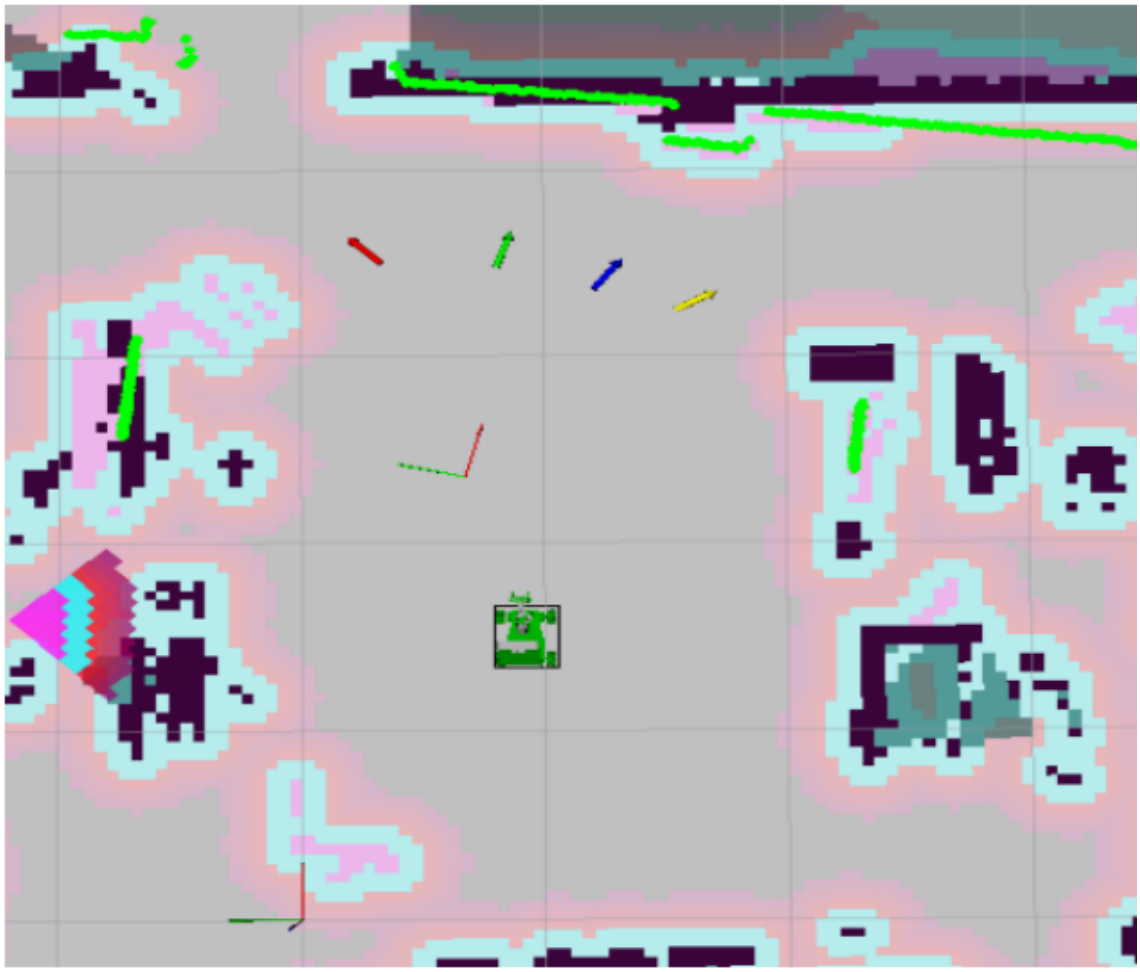
  In rviz, use the 3 tool to mark the origin of the rosmaster. This origin is the position where the car automatically returns after putting down the color block;

- Step 3

  In rviz, use 4 tools to mark red, green, blue, and yellow destinations on the map in sequence. After the rosmaster's robotic arm gripper picks up the color blocks,

  It will navigate to the destination corresponding to the color of the color block. It is best to mark the four points in a row, and the distance between the front and back should not be too far apart.

  Of course, when planning a path, you may hit color blocks.



Note: **Hazardous garbage** corresponds to the **red** marking point; **Wet garbage** corresponds to the **green** marking point; **Recyclable garbage** corresponds to the **blue** Marking point; **Dry garbage** corresponding

The **yellow** marked point.

- Step 4

  Say "Hello, Xiaoya" to rosmaster. After waking up the voice module, place the color block about 20cm away from the camera on the robotic arm. Wait for about 8 seconds until the object appears stably in the picture, and then say to it "This is What kind of garbage?" It will answer the name and type of garbage it recognizes, and after the second beep, hand the block to the gripper of the robotic arm, and the gripper will clamp the block. After rosmaster moves back to adjust its attitude, it will autonomously navigate to the previously set location. After arriving, it will put down the block with its paws and announce "Placement completed". Then, after moving back and adjusting the position, it will navigate back to the set origin and wait for the next recognition.

**Note: After running the program, this screen will appear. This is because the program is loading model data, which takes a while. When the screen turns into color, it means the model is loaded successfully.**


**5.2.3, Core code garbage_identify_yolov5.py**

- code path

  ```
  ~/yahboomcar_ws/src/garbage_identify_yolov5
  ```

- Core code analysis

  1). Import the corresponding library file

  ```
  from garbage_identify import garbage_identify #Garbage identification library
  from Speech_Lib import Speech #Speech recognition library
  from garbage_library import GarbageTransport #Transportation navigation
  library
  ```

  2), create objects

  ```
  self.target = garbage_identify() #Create garbage identification object
  spe = Speech() #Create speech recognition object
  self.garbage_transbot = GarbageTransport() #Create a transportation
  navigation object
  ```

  3). Garbage identification library garbage_identify.py

```
Model path:/home/jetson/software/yolov5-5.0/model0.pt
#pi5 /root/software/yolov5-5.0/model0.pt
Main function: garbage_run (img): executes the garbage identification
function, passes in the original image, returns the identified image, and
identifies the name
get_pos(), obtain identification information and return name
```

The program terminal will also print out the recognition results. When garbage is recognized, the recognition screen will be stuck because other programs are executed at this time, and the program is blocked. After other programs are executed, it will return to normal. Recognition screen.

4. Main processing function of garbage_identify_yolov5.py: process

In this function, it mainly judges the return result obtained by self.target.garbage_run(img), which is the name of the garbage. Then, it enters the self.garbage_transbot.process() function. This function takes parameters. The parameters The representative meaning is to go to the point marked on rviz.

5), garbage_transbot.process() function

After entering this function, the current mode state will be determined first. If it is the default Grip state, the passed parameter value will be determined. **1 represents the red mark point, 2 represents the green mark point, 3 represents the blue mark point, and 4 Indicates a yellow marker**. For other statuses, you can view the source code of the program. It can be seen that every time the program in this state is executed, the mode state will be changed to the next state expected to be executed.

### 5.3. Program flow chart