Machine code following

This lesson uses the Raspberry Pi as an example. For Raspberry Pi and Jetson-Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this lesson in the terminal. For instructions on entering the Docker container from the host computer, refer to **[01. Robot Configuration and Operation Guide] -- [4.Enter Docker (For JETSON Nano and RPi 5)]**. For RDKX5 and Orin boards, simply open a terminal and enter the commands mentioned in this lesson.

1. Program Functionality

> After the program starts, the robot will recognize the machine code and select it. The robot will then continuously follow the machine code, remaining in the center of the screen and maintaining a 0.4-meter distance from the recognized machine code. Press the `q/Q key` to exit the program.

2. Program Code Reference Path

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_depth/yahboomcar_depth/Advanced/apriltagFollow.py
```

- apriltagFollow.py

  This function mainly detects the AprilTag machine code. Based on the detected AprilTag machine code center coordinates and the distance between the machine code and the vehicle, it calculates the vehicle's forward distance and turning angle, and then publishes the forward and turning data to the vehicle.

3. Program Startup

3.1. Startup Commands

For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

Enter the Docker container (see [Docker course] --- [4. Docker Startup Script] for steps).

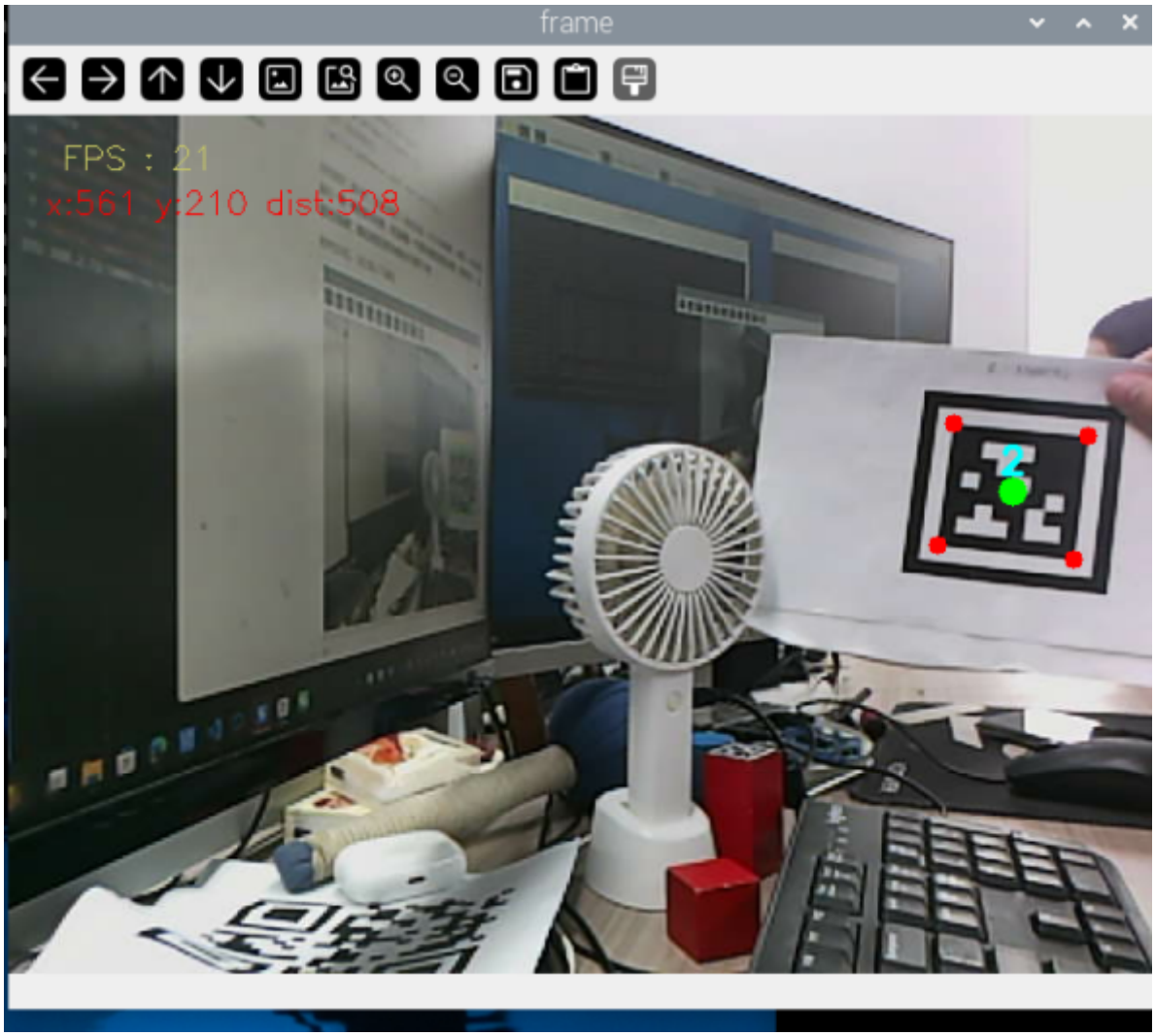
All the following commands must be executed within the same Docker container **(see [Docker course] --- [3. Docker Submission and Multi-Terminal Access] for steps).**

Enter the terminal.

```
# Start the depth camera data
ros2 launch ascamera hp60c.launch.py
# Start the car chassis
ros2 launch yahboomcar_bringup yahboomcar_bringup_M1_launch.py
# Start the QR code following program
ros2 run yahboomcar_depth depth_apriltagFollow
```

When a machine code is detected, the machine code area will be automatically outlined, and the car will begin following. **(Note: If the robot is running intermittently and the performance is poor, you can replace the machine code image with one that better captures depth information. This is because the robot will stop if the machine code depth data is 0.)**

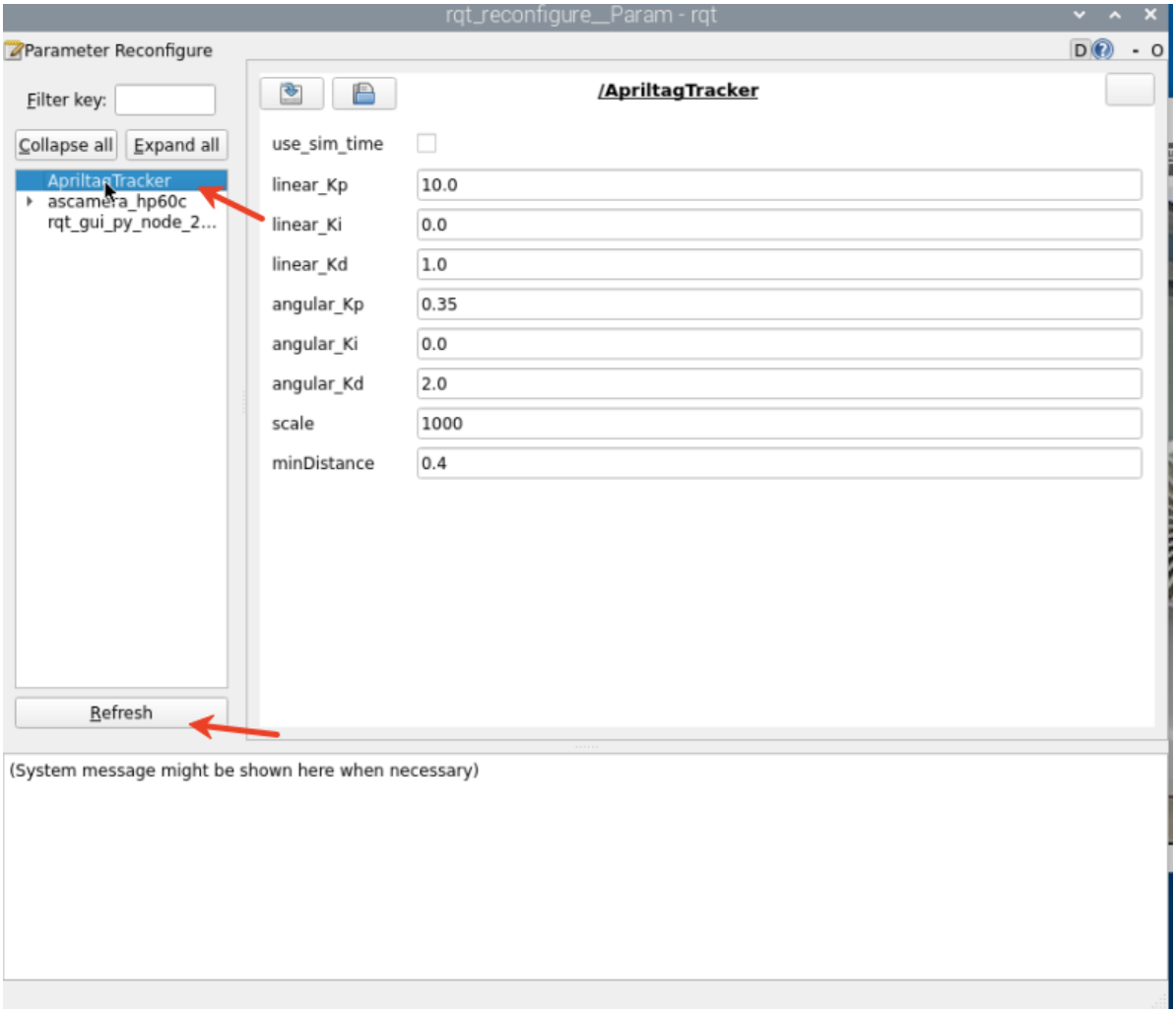
After starting the program, the following screen will appear.



3.2 Dynamic Parameter Adjustment

Enter in the terminal:

```
ros2 run rqt_reconfigure rqt_reconfigure
```



☑ After modifying the parameters, click a blank space in the GUI to write the parameter values. Note that this will only take effect for the current startup. To permanently change the parameters, you need to modify them in the source code.

Parameter Analysis:

[linear_Kp], [linear_Ki], [linear_Kd]: PID control of linear velocity during the car's following process.

[angular_Kp], [angular_Ki], [angular_Kd]: PID control of angular velocity during the car's following process.

[minDistance]: Following distance, which is maintained at this distance.

[scale]: PID scaling.

4. Core Code

4.1. apriltagFollow.py

This program has the following main functions:

- Subscribes to the depth camera topic and obtains topic images;
- Uses the dt_apriltags library to detect the recognized machine tags;
- Processes the image to obtain the center coordinates of the machine tag image and depth distance information.
- Calculates the distance PID and center coordinate PID to obtain the car's forward speed, distance, and turning angle.

Some of the core code is as follows.

```
#AprilTag检测与处理#AprilTag detection and processing
tags = self.at_detector.detect(
    cv.cvtColor(result_image, cv.COLOR_BGR2GRAY),  # 转为灰度 #Convert to
grayscale
    False,
    None,
    0.025)  # 解码阈值参数 #Decoding threshold parameter

tags = sorted(tags, key=lambda tag: tag.tag_id)  # 按ID排序 #Sort by ID
# 绘制标签：红色角点，绿色中心点 Draw labels: red corner point, green center point
result_image = draw_tags(result_image, tags, corners_color=(0,0,255),
center_color=(0,255,0))
#机器码检测处理 #Machine code detection and processing
if len(tags) > 0:
    tag = tags[0]  # 只处理第一个检测到的标签 #Only process the first detected label
    self.Center_x, self.Center_y = tag.center  # 获取标签中心 Get tag center
    corners = tag.corners.reshape((-1, 2))
    # 计算边界框 Calculate bounding boxes
    x, y, w, h = cv.boundingRect(corners.astype(np.int32))
    self.Center_r = w * h // 100  # 面积作为半径指标 Area as a radius indicator
#距离计算 #Distance calculation
points = [
    (int(self.Center_y), int(self.Center_x)),
    (int(self.Center_y+1), int(self.Center_x+1)),
    (int(self.Center_y-1), int(self.Center_x-1))
]

valid_depths = []
for y, x in points:
    # 获取深度值（注意: OpenCV是row,col即y,x） Get depth value (note: OpenCV is row,
col is y, x)
    depth_val = depth_image_info[y][x]
    if depth_val != 0:  # 过滤无效深度 Filter invalid depth
        valid_depths.append(depth_val)

dist = int(sum(valid_depths)/len(valid_depths)) if valid_depths else 0
self.dist = dist  # 存储当前距离 #Store current distance
#控制执行 #Control execution
if dist > 0.2:  # 有效距离阈值(0.2米) Effective distance threshold (0.2 meters)
    self.execute(self.Center_x, dist)  # 执行控制 Execution control

# 根据x值、y值，使用PID算法，计算运动速度，转弯角度
# Calculate the movement speed and turning angle using the PID algorithm based on
the x and y values
def execute(self, rgb_img, action):
    #PID计算偏差 PID calculation deviation
    linear_x = self.linear_pid.compute(dist, self.minDist)
    angular_z = self.angular_pid.compute(point_x,320)
    # 小车停车区间，速度为0 The car is in the parking area and the speed is 0
    if abs(dist - self.minDist) < 30: linear_x = 0
    if abs(point_x - 320.0) < 30: angular_z = 0
    twist = Twist()
    ...
    # 将计算后的速度信息发布 Publish the calculated speed information
    self.pub_cmdVel.publish(twist)
```