AprilTag machine code tracking

1. Program Functionality

> After the program starts, it automatically detects the AprilTag code on the screen. The servo gimbal will lock onto the detected AprilTag code, keeping it at the center of the screen. Pressing the `q/Q key` exits the program.

2. Program Code Reference Path

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/apriltagTracker.py
```

- apriltagTracker.py
  Mainly performs AprilTag code detection, calculates the servo's desired rotation angle based on the detected AprilTag code's center coordinates, and sends the servo's angle control data to the car.

3. Program Startup

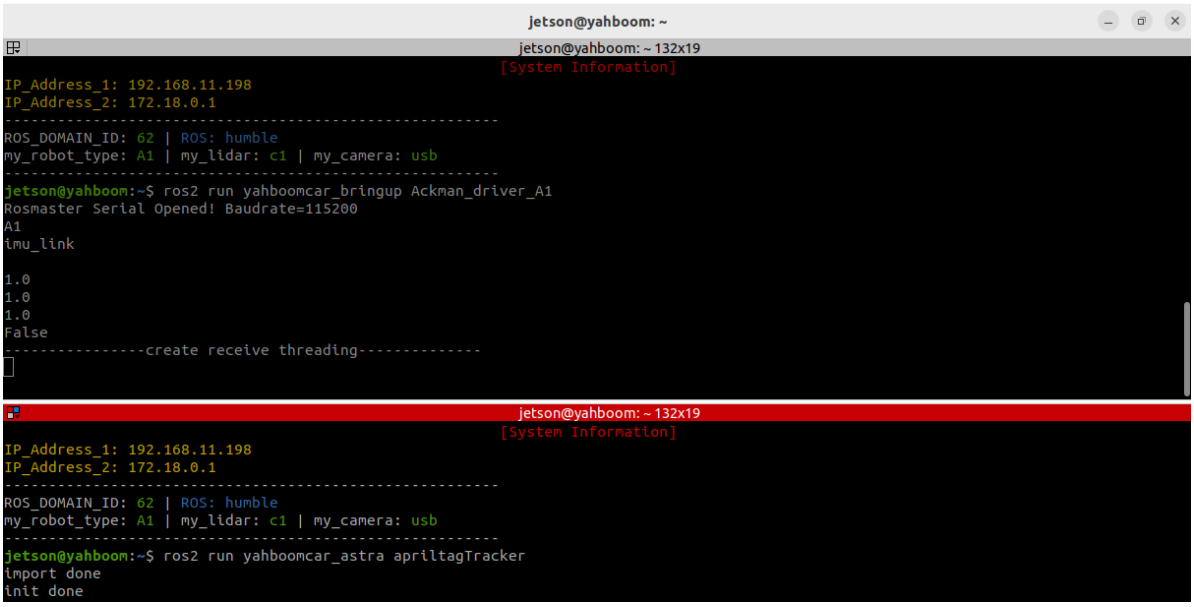
3.1. Startup Command

For the Raspberry Pi 5 controller, you must first enter the Docker container. This is not necessary for the Orin board.

Enter the Docker container (for steps, see the Docker section [3. Docker Submission and Multi-Terminal Access]).

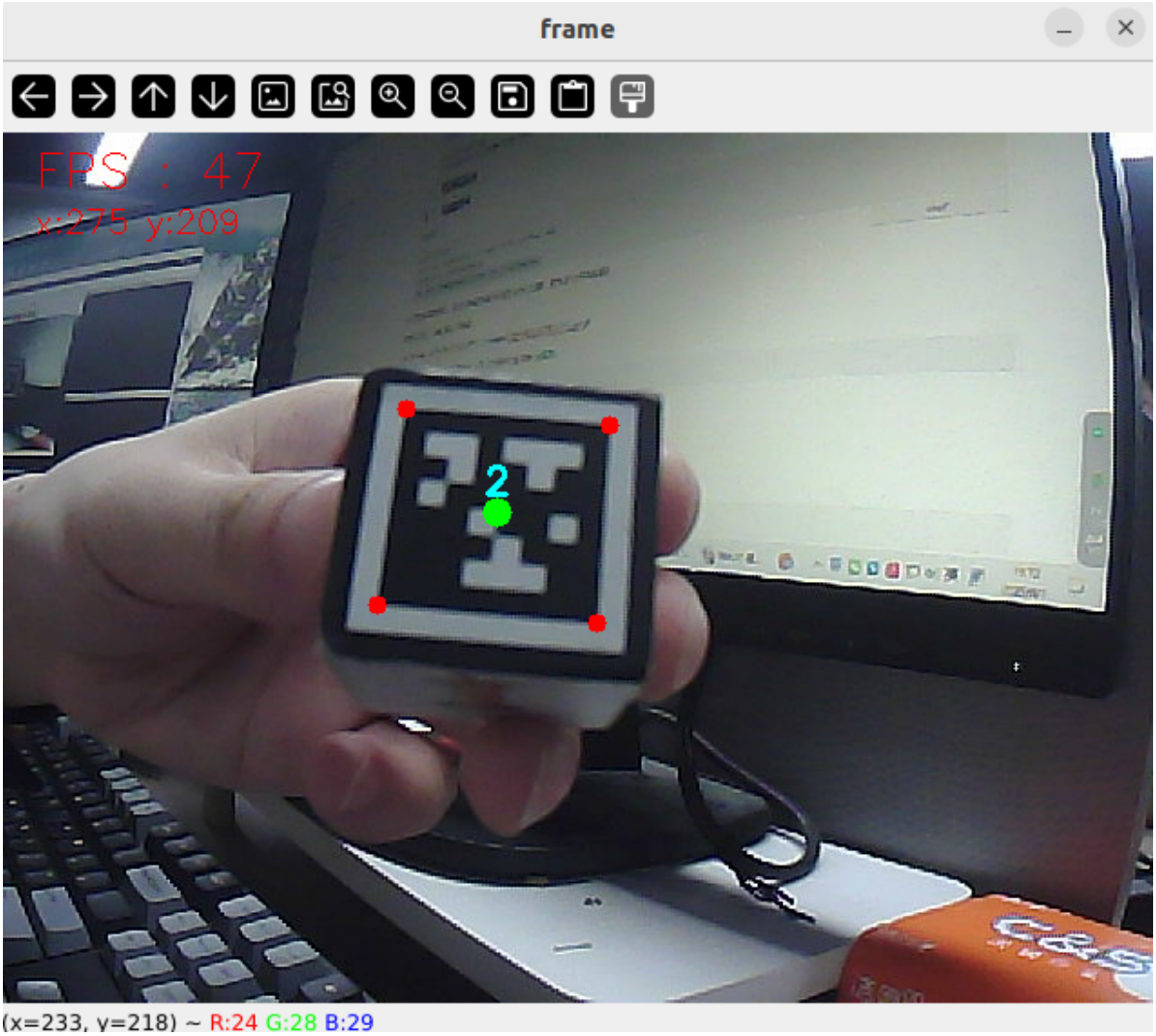
Enter the terminal.

```
# Start the car chassis
ros2 run yahboomcar_bringup Ackman_driver_A1
# Start the AprilTag machine code tracking program
ros2 run yahboomcar_astra apriltagTracker
```



When the machine code is detected, it will automatically be outlined and its center position will be displayed. The servo gimbal will begin tracking.

After the program starts, the following screen will appear.



In addition, we can enter the following command to print information about the target center coordinates:

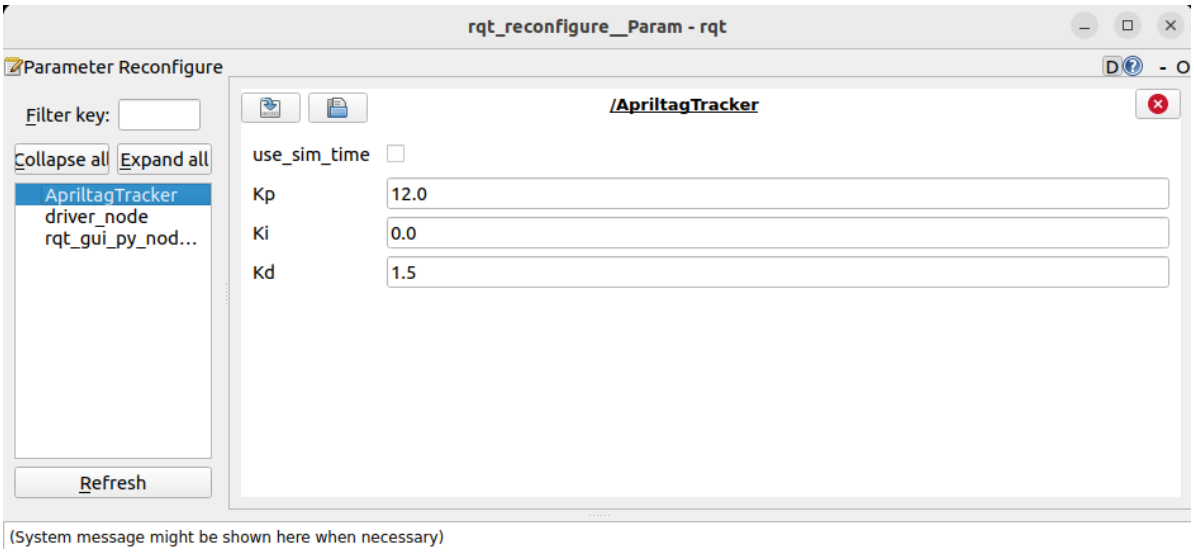
```
ros2 topic echo /Current_point
```

```
jetson@yahboom:~$ ros2 topic echo /Current_point
anglex: 42.0
angley: 220.0
distance: 0.0
---
anglex: 49.0
angley: 222.0
distance: 0.0
---
```

3.2 Dynamic Parameter Adjustment

In the terminal, enter:

```
ros2 run rqt_reconfigure rqt_reconfigure
```



☑ After modifying the parameters, click a blank area in the GUI to write the parameter values. Note that the values will only take effect during the current boot. To permanently implement them, you need to modify the parameters in the source code.

As shown in the figure above,

- apriltagTracker is primarily responsible for servo gimbal movement, adjusting PID-related parameters to achieve optimal gimbal motion.

🛠 Parameter Analysis:

[Kp], [Ki], [Kd]: PID control of the servo gimbal speed during tracking.

4. Core Code

4.1. apriltagTracker.py

This program has the following main functions:

- Initializes the AprilTag detector
- Opens the camera and captures an image
- Detects the AprilTag tag in the image

- Calculates the AprilTag center coordinates and publishes them
- Uses the PID algorithm to calculate the servo angle and issue control commands

Some of the core code is as follows:

```
# Create a publisher to publish the center coordinates of the tracked object
self.pub_position = self.create_publisher(Position, "/Current_point", 10)
# Defines the servo data publisher
self.pub_Servo = self.create_publisher(ServoControl, 'Servo', 10)
...
# Initialize the AprilTag detector
self.at_detector = Detector(
    searchpath=['apriltags'],
    families='tag36h11',
    nthreads=8,
    quad_decimate=2.0,
    quad_sigma=0.0,
    refine_edges=1,
    decode_sharpening=0.25,
    debug=0
)
...
# Detect AprilTag
tags = self.at_detector.detect(cv.cvtColor(frame, cv.COLOR_BGR2GRAY))
if len(tags) > 0:
    tag = tags[0]
    self.Center_x, self.Center_y = tag.center
    self.execute(self.Center_x, self.Center_y)
...
# Calculate the servo angle using the PID algorithm based on the x and y values
def execute(self, point_x, point_y):
    position = Position()
    position.anglex = point_x * 1.0
    position.angley = point_y * 1.0
    # Publish the center coordinates message
    self.pub_position.publish(position)
    ...
    # Limit PID polarity and maximum value
    [x_Pid, y_Pid] = self.PID_controller.update([point_x - 320, point_y - 240])
    x_Pid = x_Pid * (abs(x_Pid) <= self.Kp/2.4)
    y_Pid = y_Pid * (abs(y_Pid) <= self.Kp/2.4)
    ...
    # Publish the calculated servo angle
    self.pub_Servo.publish(self.servo_angle)
```