1. PID algorithm control

Note: The PID has been debugged before the product leaves the factory. It is not recommended to adjust it yourself, as it may cause problems with subsequent functions. But you can learn debugging methods and operating procedures.

Function package path: ~/transbot_ws/src/transbot_bringup

1.1. Debugging method

First check the source code transbot_driver.py

```
    def dynamic_reconfigure_callback(self, config, level):
        # self.bot.set_pid_param(config['Kp'], config['Ki'], config['Kd'])
        ... ...
```

Find the self.bot.set_pid_param() function in the above two functions. The default PID is not adjustable. If you need to adjust it, remove the [#] in front and just correct the format.

To facilitate debugging, use the remote control method for debugging. Either jetson nano or a virtual machine can be used as the host, but ros Master must be started by the host. Take jetson nano/Raspberry Pi 4B and Raspberry Pi 5 as hosts as an example.

jetson nano terminal/Raspberry Pi 4B

```
roscore
rosrun transbot_bringup transbot_driver.py
```

Raspberry Pi 5

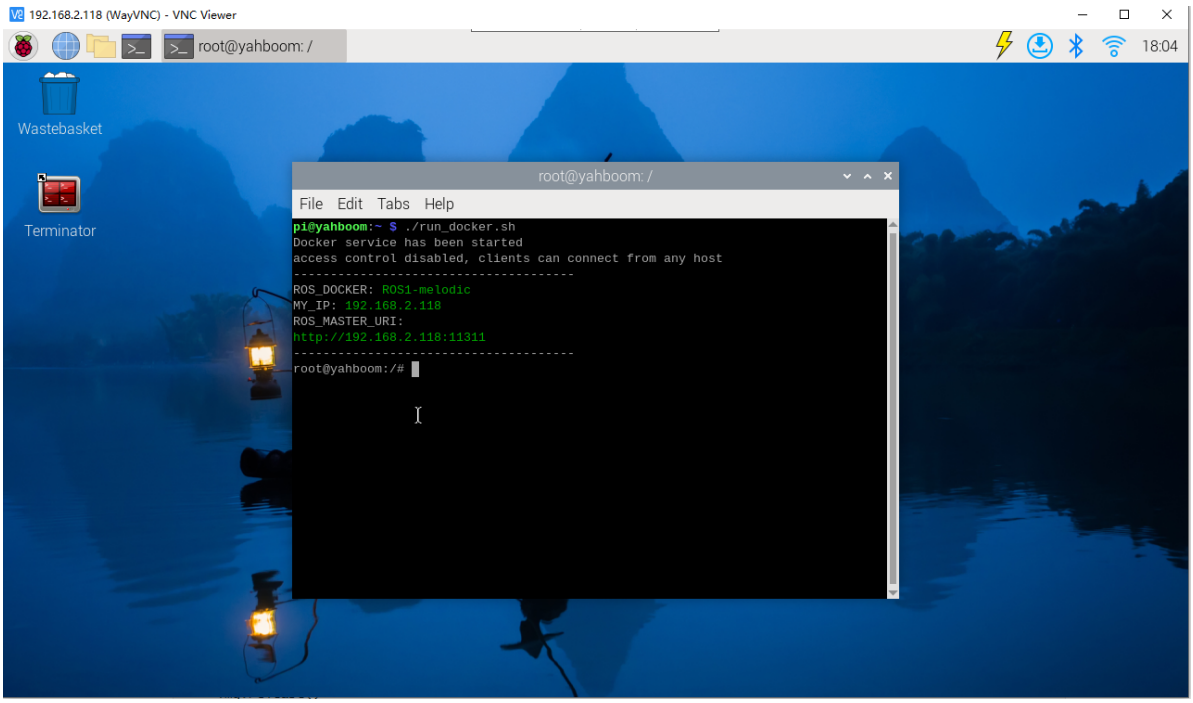
Before running, please confirm that the large program has been permanently closed

Enter docker

Note: If there is a terminal that automatically starts docker, or there is a docker terminal that has been opened, you can directly enter the docker terminal to run the command, and there is no need to manually start docker

Start docker manually

```
./run_docker.sh
```




```
roscore
```

Enter the same docker from multiple terminals

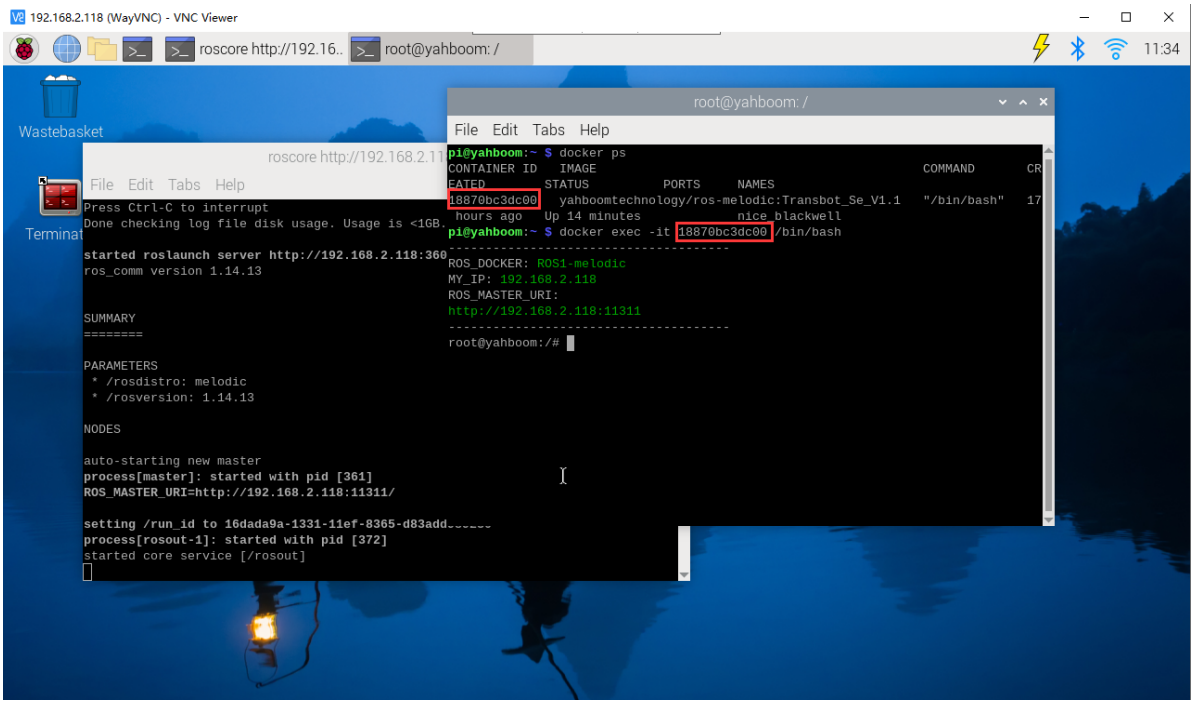
Keep the program of the previous docker terminal running and open a new terminal

Enter the following command

```
docker ps
```

Enter the same docker and use the following 18870bc3dc00 to modify the ID displayed on the actual terminal.

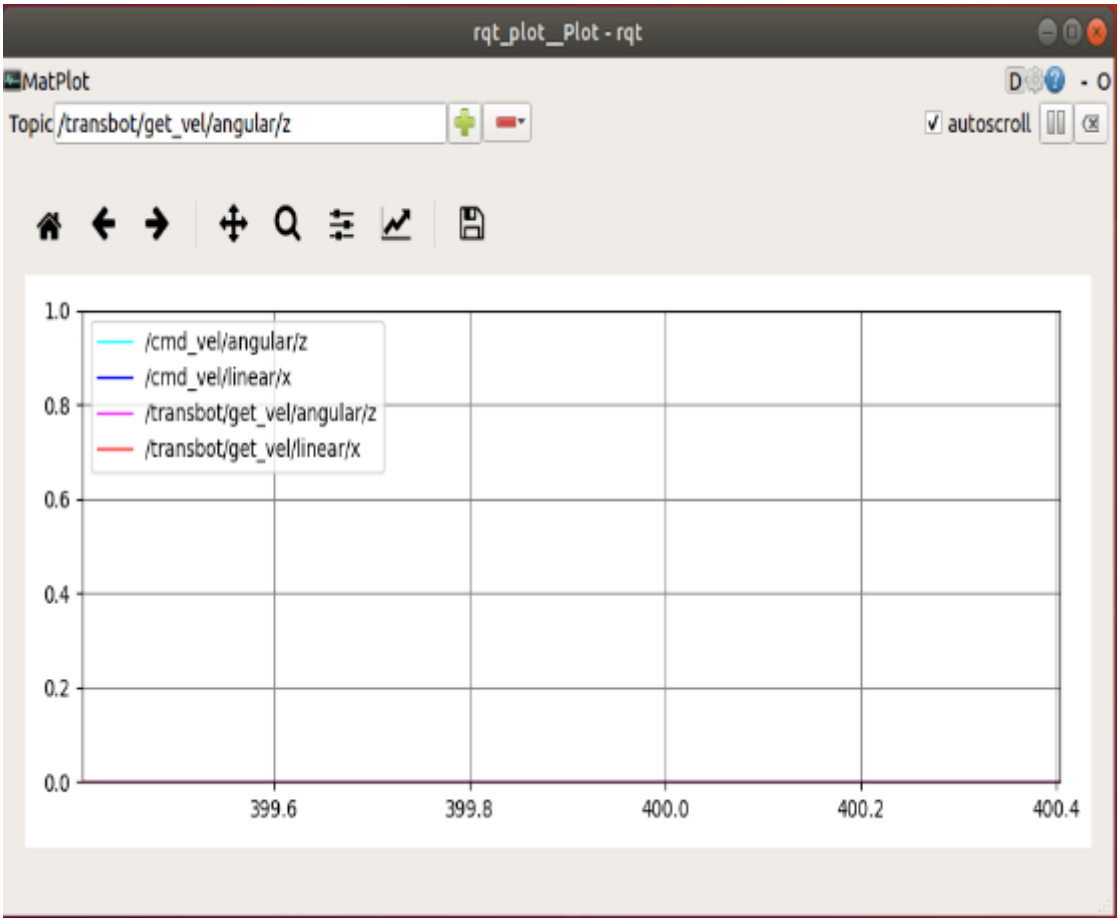
```
docker exec -it 18870bc3dc00 /bin/bash
```



```
rosrun transbot_bringup transbot_driver.py
```

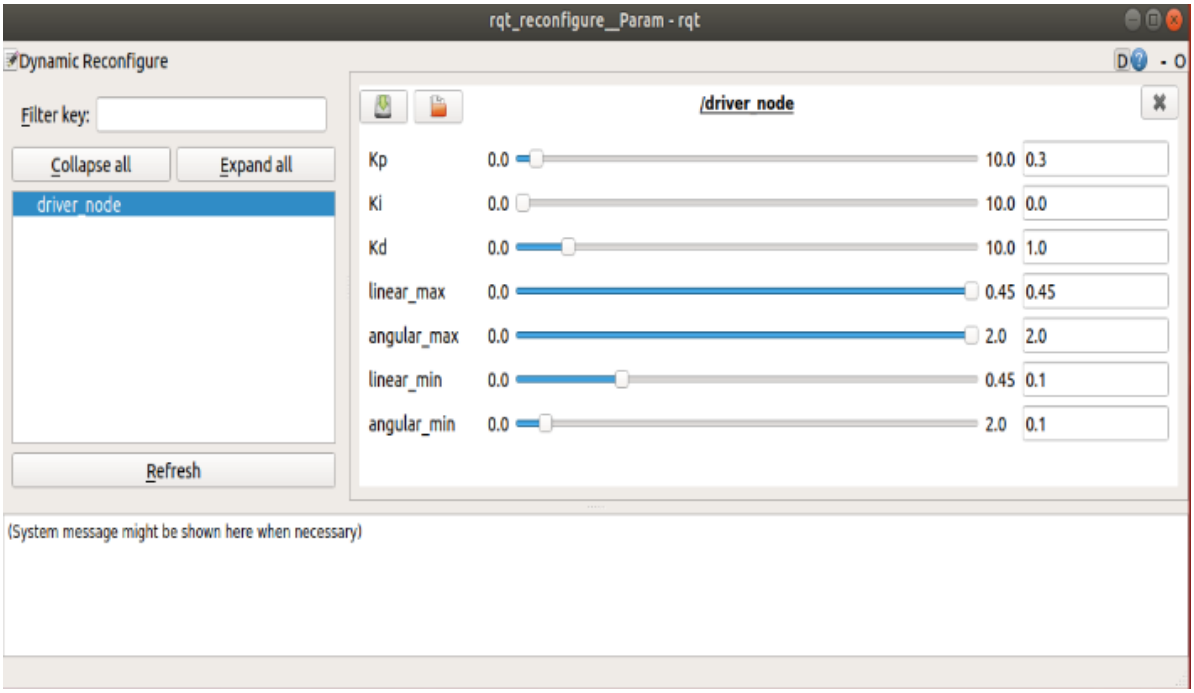
virtual machine

```
rqt_plot # rqt visualization tool
rosrun rqt_reconfigure rqt_reconfigure # Parameter adjuster
rosrun teleop_twist_keyboard teleop_twist_keyboard.py # Keyboard control node
```



First select the debugging parameters. After entering the name of each parameter as shown in the figure, click [+] to add, [-] to delete.

- /cmd_vel/angular/z: target angular velocity
- /cmd_vel/linear/x: target linear speed
- /transbot/get_vel/angular/z: Current angular velocity
- /transbot/get_vel/linear/x: current linear speed



As shown in the figure, select the [driver_node] node. We only need to adjust the three parameters [Kp], [Ki], and [Kd], and do not adjust the others.

Debugging steps:

- Use the keyboard keys on the keyboard control terminal to drive the car to move [forward], [backward], [turn left], and [turn right]
- Observe the changes in the [rqt_plot] window image so that the current angular velocity is close to the target angular velocity, and the current linear velocity is close to the target linear velocity. It is impossible to overlap. The target speed is in an ideal state without any interference, and the speed can be increased instantly.
- Use the keyboard [q], [z], [w], [x], [e], [c] to increase or decrease the speed and test the status of different speeds.
- Observe the changes of [rqt_plot], adjust [Kp], [Ki], and [Kd] data through [rqt_reconfigure], test multiple times, and select the optimal data.

Keyboard control instructions

```
anything else: stop
q/z: increase/decrease max speeds by 10%
w/x: increase/decrease only linear speed by 10%
e/c: increase/decrease only angular speed by 10%
```

Button	Car [linear, angular]	Button	Car [linear, angular]
[i] or [I] (forward)	[linear, 0]	[u] or [U]	[linear, angular]
[,] (back)	[-linear, 0]	[o] or [O]	[linear, - angular]
[j] or [J] (turn left)	[0, angular]	[m] or [M]	[- linear, - angular]
[l] or [L] (turn right)	[0, - angular]	[.]	[- linear, angular]

After debugging is completed, the PID is automatically stored in the PCB. Just comment out the PID setting part according to the way you originally viewed the source code.

Note: Other gameplay requires PID debugging, the theory is the same, theoretical reference.

1.2、PID algorithm theory

PID is to perform proportional integral derivative calculation on the input deviation, and the superimposed result of the calculation is used to control the actuator. The formula is as follows:

$u(t)=K_p[e(t)+\frac{1}{T_i}\int_{0}^{t}e(t)dt+Td\frac{de(t)}{dt}]$

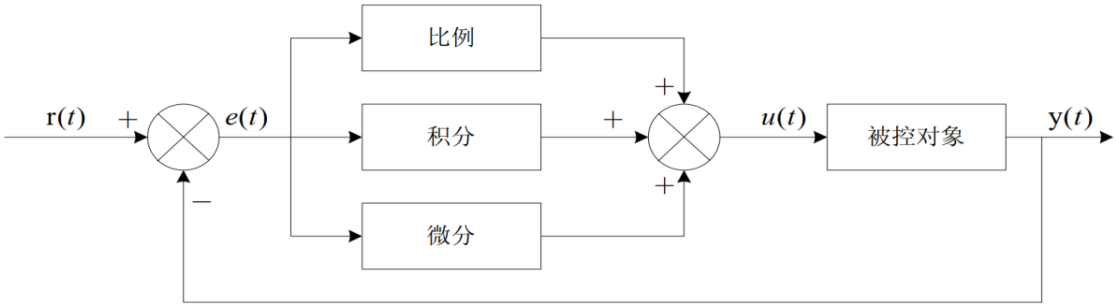
It consists of three parts:

-P is the ratio, that is, the input deviation is multiplied by a coefficient;

-I is the integral, which is the integral operation of the input deviation;

-D is differential, which performs differential operation on the input deviation.

The following figure shows a basic PID controller:



1.2.1, proportion part

The mathematical expression of the proportional part is: $K_p*e(t)$

1.2.2, integral part

The mathematical expression of the integral part is: $\frac{K_p}{T_i}\int_{0}^{t}e(t)dt$

1.2.3, differential part

The mathematical expression of the differential part is: $K_p*Td\frac{de(t)}{dt}$

1.3、PID algorithm selection

Digital PID control algorithms can be divided into positional PID and incremental PID control algorithms. Then we should first understand its principle before deciding which PID algorithm to use:

1.3.1、Positional PID algorithm

$u(k)=K_pe(k)+K_I\sum_{i=o}e(i)+K_D[e(k)-e(k-1)]$

e(k): The value set by the user (target value)-the current state value of the control object

Proportion P: e(k)

Integral I: the accumulation of ∑e(i) errors

Differential D: e(k)-e(k-1) this time error-last time error

That is to say, the positional PID is the actual position of the current system, and the deviation from the expected position you want to achieve is used for PID control.

1.3.2、Incremental PID algorithm

$\Delta u(k)=u(k)-u(k-1)=K_P[e(k)-e(k-1)]+K_ie(k)+K_D[e(k)-2e(k-1)+e(k-2)]$

Proportion P: $e(k)-e(k-1)$ this time error-last time error

Integral I: $e(k)$ error

Differential D: $e(k)-2e(k-1)+e(k-2)$ This time error-2*Last error+Last last error