5. Robot calibration

1. Program function description

After the program runs, through the dynamic parameter regulator, here to adjust the parameters to calibrate the linear speed and angular speed of the cart. Take X3 model as an example, the intuitive embodiment of calibrating linear speed is to let the cart walk 1 meter straight forward to see how far it actually runs and whether it is within the error range; the intuitive embodiment of calibrating angular speed is to let the cart rotate 360 degrees to see whether the angle of rotation of the cart is within the error range.

2. Program code reference path

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

```
#标定线速度源码 # Calibrate line speed source code
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_bringup/yahboomcar_bringup
/calibrate_linear_R2.py
#标定角速度源码 # Calibrated angular velocity source code
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_bringup/yahboomcar_bringup
/calibrate_angular_R2.py
```

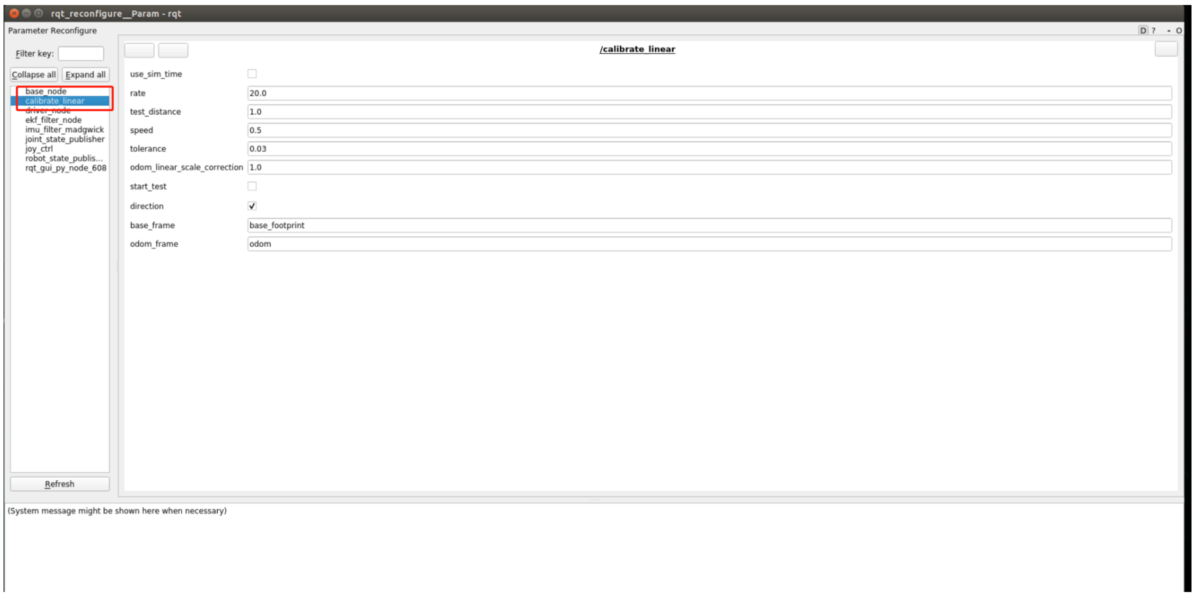
3. Program startup

After entering the docker container, depending on the actual model, take R2 as an example, terminal input, the

```
#底盘驱动 # Chassis drive
ros2 launch yahboomcar_bringup yahboomcar_bringup_R2_launch.py

#线速度和角速度分开运行
#Linear velocity and angular velocity run separately
#标定线速度 # Calibrated line speed
ros2 run yahboomcar_bringup calibrate_linear_R2
#标定角速度 # Calibrated angular velocity
ros2 run yahboomcar_bringup calibrate_angular_R2

#动态参数调节 #Dynamic parameter adjustment
ros2 run rqt_reconfigure rqt_reconfigure
```



/calibrate_linear

use_sim_time	☐
rate	20.0
test_distance	1.0
speed	0.5
tolerance	0.03
odom_linear_scale_correction	1.0
start_test	☐
direction	✓
base_frame	base_footprint
odom_frame	odom

(System message might be shown here when necessary)

Take calibration line speed as an example, click "start_test" to calibrate the line speed of the car in x-direction, and observe whether the car moves the distance of test_distance, the default setting here is 1m, you can customize the distance of the test before calibrating, it must be a decimal number, click the blank space after setting, the program will write it automatically. The program will write it automatically after setting and clicking on the blank. If the trolley moves more than the acceptable error range (value of tolerance variable), then set the value of odom_linear_scale_correction. The following are the meanings of the individual parameters.

parameters	meaning
rate	Frequency of publication (can be left unchanged)
test_distance	Distance for testing line speed
speed	The magnitude of the linear velocity
tolerance	Acceptable error values
odom_linear_scale_correction	scale factor
start_test	start testing
base_frame	Listen to the parent coordinates of the TF transformation
odom_frame	Listening to sub-coordinates of TF transformations

The variable settings for testing angular velocity are more or less the same, but test_distance has been changed to test_angle and speed has been changed to angular velocity.

4. Program core source code analysis

This program mainly uses TF to listen to the transformation of coordinates, by listening to the transformation of coordinates between base_footprint and odom to let the robot know "how far I have gone/how many degrees I have turned".

Take calibrate_linear_R2.py as an example, the core code is as follows.

```
#监听TF变换 #Listening to TF transformations
def get_position(self):
    try:
```

```
    now = rclpy.time.Time()
    trans = self.tf_buffer.lookup_transform(self.odom_frame,self.base_frame,now)
    return trans
    except (LookupException, ConnectivityException, ExtrapolationException):
    self.get_logger().info('transform not ready')
    raise
    return
#获取当前的xy坐标，根据之前的xy坐标，计算距离
# Get the current xy coordinates and calculate the distance based on the previous
xy coordinates.
self.position.x = self.get_position().transform.translation.x
self.position.y = self.get_position().transform.translation.y
print("self.position.x: ",self.position.x)
print("self.position.y: ",self.position.y)
distance = sqrt(pow((self.position.x - self.x_start), 2) +
                pow((self.position.y - self.y_start), 2))
distance *= self.odom_linear_scale_correction
```

The core calibrate_angular_R2 code is as follows.

```
#这里同样是监听了TF变换，获取到了当前位姿信息，只不过这里还做了转换，把四元数转了欧拉角的转换，
然后再返回
#Here again, the TF transformation is listened to, and the current bit position
information is obtained, except that here the conversion is also done, and the
quaternion is converted to an Eulerian angle conversion, and then returned
def get_odom_angle(self):
    try:
    now = rclpy.time.Time()
    rot = self.tf_buffer.lookup_transform(self.odom_frame,self.base_frame,now)
    #print("oring_rot: ",rot.transform.rotation)
    cacl_rot = PyKDL.Rotation.Quaternion(rot.transform.rotation.x,
rot.transform.rotation.y, rot.transform.rotation.z, rot.transform.rotation.w)
    #print("cacl_rot: ",cacl_rot)
    angle_rot = cacl_rot.GetRPY()[2]
    #print("angle_rot: ",angle_rot)
    except (LookupException, ConnectivityException, ExtrapolationException):
    self.get_logger().info('transform not ready')
    return
#计算旋转角度 # Calculate the angle of rotation
self.odom_angle = self.get_odom_angle()
self.delta_angle = self.odom_angular_scale_correction *
self.normalize_angle(self.odom_angle - self.first_angle)
```

The published TF transform is published at the node base_node, and the code path is.

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_base_node/src/base_node_R2
```

This node will receive the data from /vel_raw and publish the odom data through mathematical computation, along with the TF transformation, the core code is as follows.

```
#计算xy坐标以及xyzw四元数的值，xy两点坐标表示位置，xyzw四元数表示姿态
# Calculate the xy coordinates and the value of the xyzw quaternion, where the xy
two-point coordinates represent the position and the xyzw quaternion represents
the attitude
double steer_angle = linear_velocity_y_;
double MI_PI = 3.1416;
double R = wheelbase_ / tan(steer_angle/180.0*MI_PI);
double angular_velocity_z_ = linear_velocity_x_/R;
double delta_heading = angular_velocity_z_ * vel_dt_; //radians
double delta_x = (linear_velocity_x_ * cos(heading_)) * vel_dt_; //m
double delta_y = (linear_velocity_x_ * sin(heading_)) * vel_dt_; //m
x_pos_ += delta_x;
y_pos_ += delta_y;
heading_ += delta_heading;
tf2::Quaternion myQuaternion;
geometry_msgs::msg::Quaternion odom_quat ;
myQuaternion.setRPY(0.00,0.00,heading_ );
#发布TF变换 #Release TF transformations
geometry_msgs::msg::TransformStamped t;
rclcpp::Time now = this->get_clock()->now();
t.header.stamp = now;
t.header.frame_id = "odom";
t.child_frame_id = "base_footprint";
t.transform.translation.x = x_pos_;
t.transform.translation.y = y_pos_;
t.transform.translation.z = 0.0;
t.transform.rotation.x = myQuaternion.x();
t.transform.rotation.y = myQuaternion.y();
t.transform.rotation.z = myQuaternion.z();
t.transform.rotation.w = myQuaternion.w();
tf_broadcaster_->sendTransform(t);
```