

Angular velocity calibration

Note: The ROS_DOMAIN_ID of the Raspberry Pi and the microROS control board need to be consistent. You can check [MicroROS Control Board Parameter Configuration] to set the microROS control board ROS_DOMAIN_ID. Check the tutorial [Connect MicroROS Agent] to determine whether the IDs are consistent.

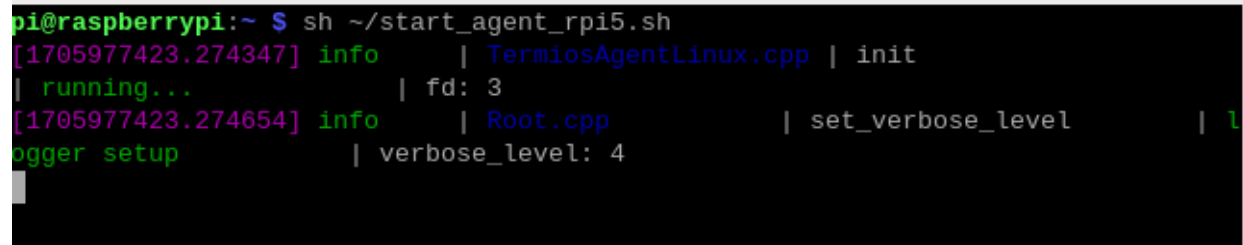
1. Program function description

The car connects to the agent, runs the program, and adjusts the parameters here through the dynamic parameter adjuster to calibrate the car's angular velocity. The intuitive reflection of the calibrated angular velocity is to give the car a command to rotate 360 degrees (one revolution) to see how many degrees it actually rotates and whether it is within the error range.

2. Start and connect to the agent

After successfully starting the Raspberry Pi, enter the following command to start the agent.

```
sh ~/start_agent_rpi5.sh
```



```
pi@raspberrypi:~ $ sh ~/start_agent_rpi5.sh
[1705977423.274347] info      | TermiosAgentLinux.cpp | init
| running...              | fd: 3
[1705977423.274654] info      | Root.cpp               | set_verbose_level      | 1
logger setup              | verbose_level: 4
```

Then, turn on the car switch and wait for the car to connect to the agent. The connection is successful, as shown in the figure below.

```

subscriber created      | client_key: 0x57E5DE1D, subscriber_id: 0x001(4), partici
pant_id: 0x000(1)
[1705977460.524081] info    | ProxyClient.cpp      | create_datareader      | d
atareader created      | client_key: 0x57E5DE1D, datareader_id: 0x001(6), subscri
ber_id: 0x001(4)
[1705977460.530288] info    | ProxyClient.cpp      | create_topic           | t
opic created           | client_key: 0x57E5DE1D, topic_id: 0x005(2), participant_
id: 0x000(1)
[1705977460.533849] info    | ProxyClient.cpp      | create_subscriber      | s
ubscriber created      | client_key: 0x57E5DE1D, subscriber_id: 0x002(4), partici
pant_id: 0x000(1)
[1705977460.540811] info    | ProxyClient.cpp      | create_datareader      | d
atareader created      | client_key: 0x57E5DE1D, datareader_id: 0x002(6), subscri
ber_id: 0x002(4)
[1705977460.548331] info    | ProxyClient.cpp      | create_topic           | t
opic created           | client_key: 0x57E5DE1D, topic_id: 0x006(2), participant_
id: 0x000(1)
[1705977460.553389] info    | ProxyClient.cpp      | create_subscriber      | s
ubscriber created      | client_key: 0x57E5DE1D, subscriber_id: 0x003(4), partici
pant_id: 0x000(1)
[1705977460.556688] info    | ProxyClient.cpp      | create_datareader      | d
atareader created      | client_key: 0x57E5DE1D, datareader_id: 0x003(6), subscri
ber_id: 0x003(4)

```

3. Enter the car system docker

Open another terminal and enter the following command to enter docker.

```
sh ros2_humble.sh
```

When the following interface appears, you have successfully entered docker. Now you can control the car through commands.

```

pi@raspberrypi:~ $ ./ros2_humble.sh
access control disabled, clients can connect from any host
Successful
MY_DOMAIN_ID: 20
root@raspberrypi:/#

```

4. Start the program

First, start the car's underlying data processing program. The program will release the TF transformation of odom->base_footprint. With this TF change, you can calculate "how many degrees the car has turned." Enter the following command at the terminal.

```
ros2 launch yahboomcar_bringup yahboomcar_bringup_launch.py
```

```

-----robot_type = x3-----
[INFO] [imu_filter_madgwick_node-1]: process started with pid [431]
[INFO] [ekf_node-2]: process started with pid [433]
[INFO] [static_transform_publisher-3]: process started with pid [435]
[INFO] [joint_state_publisher-4]: process started with pid [437]
[INFO] [robot_state_publisher-5]: process started with pid [439]
[INFO] [static_transform_publisher-6]: process started with pid [441]
[static_transform_publisher-6] [WARN] [1705990344.817767742] []: Old-style arguments are deprecated; see --help for new-style arguments
[static_transform_publisher-3] [WARN] [1705990344.820799310] []: Old-style arguments are deprecated; see --help for new-style arguments
[static_transform_publisher-6] [INFO] [1705990344.867824585] [static_transform_publisher_Vw23cBm mv8AsTkAi]: Spinning until stopped - publishing transform
[static_transform_publisher-6] translation: ('0.000000', '0.000000', '0.050000')
[static_transform_publisher-6] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-6] from 'base_footprint' to 'base_link'
[static_transform_publisher-3] [INFO] [1705990346.845764293] [base_link_to_base_imu]: Spinning until stopped - publishing transform
[static_transform_publisher-3] translation: ('-0.002999', '-0.003000', '0.031701')
[static_transform_publisher-3] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-3] from 'base_link' to 'imu_frame'
[robot_state_publisher-5] [WARN] [1705990346.899959298] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.
[robot_state_publisher-5] [INFO] [1705990346.900111131] [robot_state_publisher]: got segment base_link
[robot_state_publisher-5] [INFO] [1705990346.900207686] [robot_state_publisher]: got segment imu_Link
[robot_state_publisher-5] [INFO] [1705990346.900221723] [robot_state_publisher]: got segment jq1_Link
[robot_state_publisher-5] [INFO] [1705990346.900233427] [robot_state_publisher]: got segment jq2_Link
[robot_state_publisher-5] [INFO] [1705990346.900243334] [robot_state_publisher]: got segment radar_Link
[robot_state_publisher-5] [INFO] [1705990346.900253279] [robot_state_publisher]: got segment yh_Link
[robot_state_publisher-5] [INFO] [1705990346.900262908] [robot_state_publisher]: got segment yq_Link
[robot_state_publisher-5] [INFO] [1705990346.900272834] [robot_state_publisher]: got segment zh_Link
[robot_state_publisher-5] [INFO] [1705990346.900283705] [robot_state_publisher]: got segment zq_Link
[imu_filter_madgwick_node-1] [INFO] [1705990347.009413102] [imu_filter]: Starting ImuFilter
[imu_filter_madgwick_node-1] [INFO] [1705990347.010925118] [imu_filter]: Using dt computed from message headers
[imu_filter_madgwick_node-1] [INFO] [1705990347.010983044] [imu_filter]: The gravity vector is kept in the IMU message.
[imu_filter_madgwick_node-1] [INFO] [1705990347.011764412] [imu_filter]: Imu filter gain set to 0.100000
[imu_filter_madgwick_node-1] [INFO] [1705990347.011816635] [imu_filter]: Gyro drift bias set to 0.000000
[imu_filter_madgwick_node-1] [INFO] [1705990347.011844116] [imu_filter]: Magnetometer bias values: 0.000000 0.000000 0.000000
[imu_filter_madgwick_node-1] [INFO] [1705990347.079544893] [imu_filter]: First IMU message received.
[joint_state_publisher-4] [INFO] [1705990347.120567333] [joint_state_publisher]: Waiting for robot_description to be published on the robot_description topic...

```

Then, start the car angular velocity calibration program and enter in the terminal.

```
ros2 run yahboomcar_bringup calibrate_angular
```

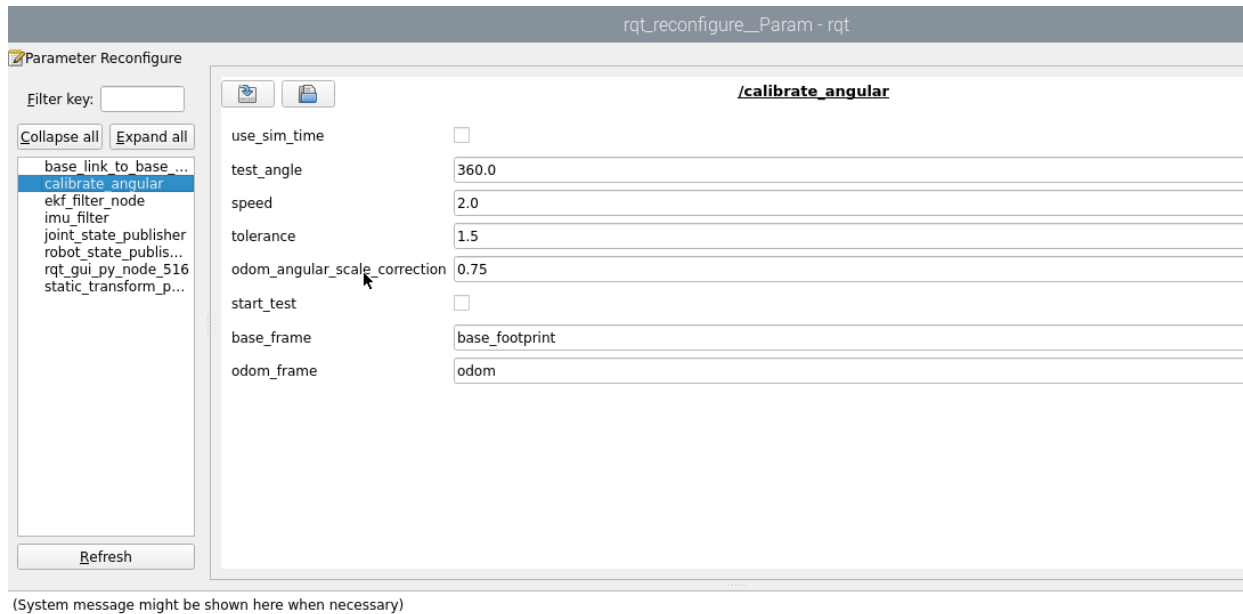
```

root@raspberrypi:~# ros2 run yahboomcar_bringup calibrate_angular
finish init work

```

Finally, open the dynamic parameter adjuster and enter the following command in the terminal,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



Note: There may not be the above nodes when you first open it. You can see all the nodes after clicking Refresh. The **calibrate_angular** node shown is the node for calibrating angular velocity.

5. Start calibration

- In the rqt_reconfigure interface, select the calibrate_angular node. There is **start_test** below and click the box to the right to start calibration. Other parameters in the rqt interface are described as follows:
- test_angle: Calibrate the test angle, here the test rotates 360 degrees;
- speed: Angular velocity magnitude;
- tolerance: tolerance for error;
- odom_angular_scale_correction: Linear speed proportional coefficient, if the test result is not ideal, just modify this value;
- start_test: test switch;
- base_frame: The name of the base coordinate system;
- odom_frame: The name of the odometer coordinate system

Click start_test to start calibration. The car will monitor the TF transformation of base_footprint and odom, calculate the theoretical distance traveled by the car, wait until the error is less than the tolerance, and issue a parking instruction.

```

error: 6.3616122915058435
turn_angle: -0.07866185524485375
error: 6.36184716242444
turn_angle: -0.07866185524485375
error: 6.36184716242444
turn_angle: -0.07915014372615123
error: 6.3623354509057375
turn_angle: -0.07915014372615123
error: 6.3623354509057375
turn_angle: -0.07915014372615123
error: 6.3623354509057375
turn_angle: -0.07915014372615123
error: 6.3623354509057375
turn_angle: -0.07915014372615123
error: 6.3623354509057375
turn_angle: -0.0784502712803612
error: 6.361635578459947
turn_angle: -0.0784502712803612
error: 6.361635578459947
turn_angle: -0.0784502712803612
error: 6.361635578459947

```

The turn_angle here is in radians. If the actual angle of the car's rotation is not 360 degrees, then modify the odom_angular_scale_correction parameter in rqt. After modification, click on the blank space, click start_test again, reset start_test, and then click start_test again. Calibration. The same goes for modifying other parameters. You need to click on the blank space to write the modified parameters.

6. Source code reference path,

Source code reference path,

```
/root/yahboomcar_ws/src/yahboomcar_bringup/yahboomcar_bringup
```

calibrate_angular.py, The core code is as follows,

```

#Monitor the TF transformation of base_footprint and odom
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)
        cac1_rot = PyKDL.Rotation.Quaternion(rot.transform.rotation.x,
        rot.transform.rotation.y, rot.transform.rotation.z, rot.transform.rotation.w)
        #print("cac1_rot: ", cac1_rot)
        angle_rot = cac1_rot.GetRPY()[2]
        #print("angle_rot: ", angle_rot)
        return angle_rot

    except (LookupException, ConnectivityException, ExtrapolationException):
        self.get_logger().info('transform not ready')
        raise
    return

#Calculate the angle and error 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)  
#print("delta_angle: ",self.delta_angle)  
self.turn_angle += self.delta_angle  
print("turn_angle: ",self.turn_angle)  
self.error = self.test_angle - self.turn_angle
```