Angular velocity calibration

This lesson uses the Raspberry Pi as an example. For Raspberry Pi and Jetson Nano motherboards, 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] -- [5.Enter Docker (For JETSON Nano and RPi 5)]**. For Orin motherboards, simply open a terminal and enter the commands mentioned in this lesson.

1. Program Description

Run the program and adjust the parameters in the dynamic parameter adjuster to calibrate the car's angular velocity. To visually demonstrate angular velocity calibration, instruct the car to move straight forward 1 meter and observe its actual distance to see if it is within the error range.

2. Starting the Program

2.1. Startup Commands

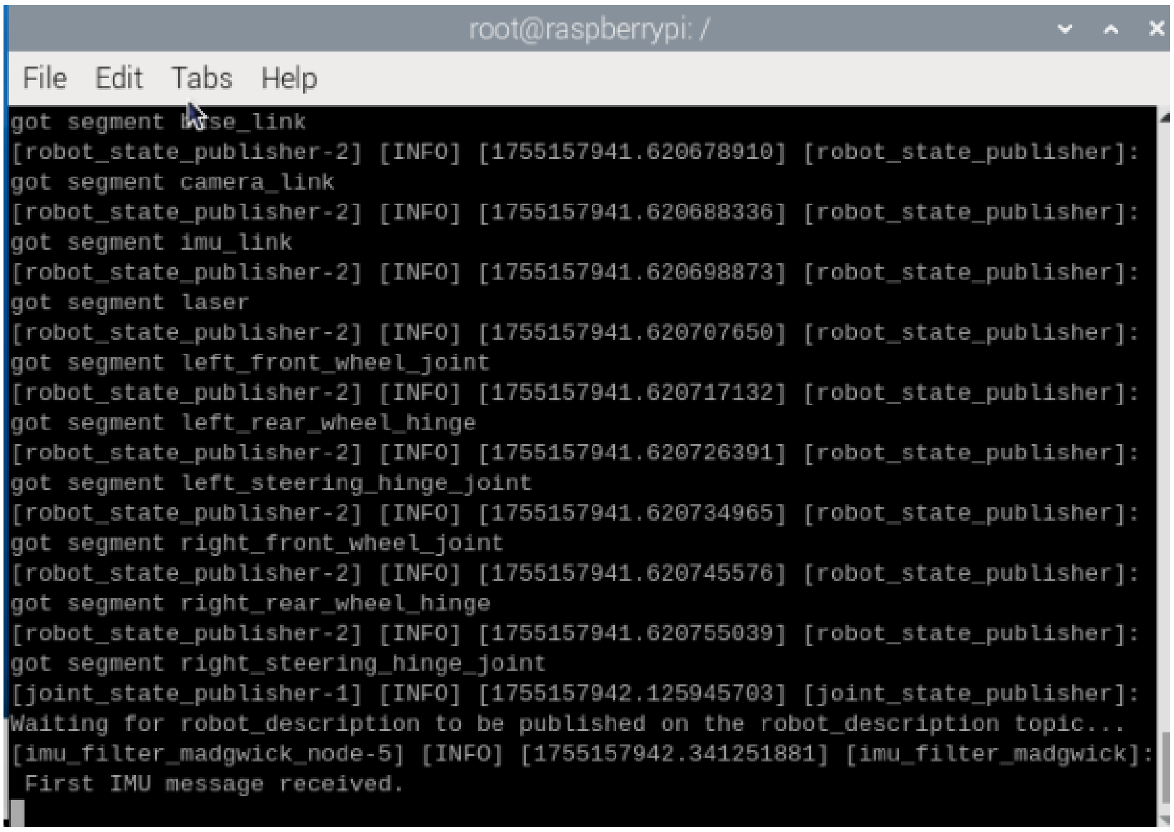
For the Raspberry Pi 5 controller, you must first enter the Docker container. For the Orin controller, this is not necessary.

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

All the following commands must be executed from the Docker terminal within the same Docker container.**(For steps, see [Docker Course] --- [3. Docker Submission and Multi-Terminal Access]).**

To start the chassis data, enter the terminal.

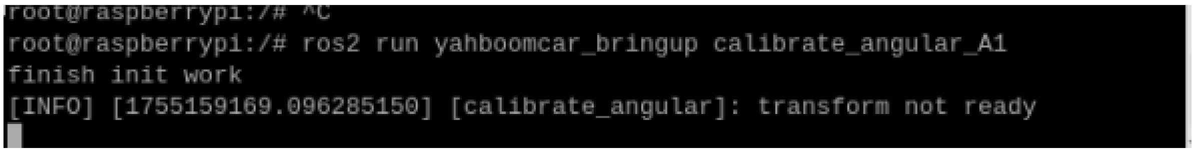
```
ros2 launch yahboomcar_bringup yahboomcar_bringup_A1_launch.py
```



```
got segment base_link
[robot_state_publisher-2] [INFO] [1755157941.620678910] [robot_state_publisher]:
got segment camera_link
[robot_state_publisher-2] [INFO] [1755157941.620688336] [robot_state_publisher]:
got segment imu_link
[robot_state_publisher-2] [INFO] [1755157941.620698873] [robot_state_publisher]:
got segment laser
[robot_state_publisher-2] [INFO] [1755157941.620707650] [robot_state_publisher]:
got segment left_front_wheel_joint
[robot_state_publisher-2] [INFO] [1755157941.620717132] [robot_state_publisher]:
got segment left_rear_wheel_hinge
[robot_state_publisher-2] [INFO] [1755157941.620726391] [robot_state_publisher]:
got segment left_steering_hinge_joint
[robot_state_publisher-2] [INFO] [1755157941.620734965] [robot_state_publisher]:
got segment right_front_wheel_joint
[robot_state_publisher-2] [INFO] [1755157941.620745576] [robot_state_publisher]:
got segment right_rear_wheel_hinge
[robot_state_publisher-2] [INFO] [1755157941.620755039] [robot_state_publisher]:
got segment right_steering_hinge_joint
[joint_state_publisher-1] [INFO] [1755157942.125945703] [joint_state_publisher]:
Waiting for robot_description to be published on the robot_description topic...
[imu_filter_madgwick_node-5] [INFO] [1755157942.341251881] [imu_filter_madgwick]:
 First IMU message received.
```

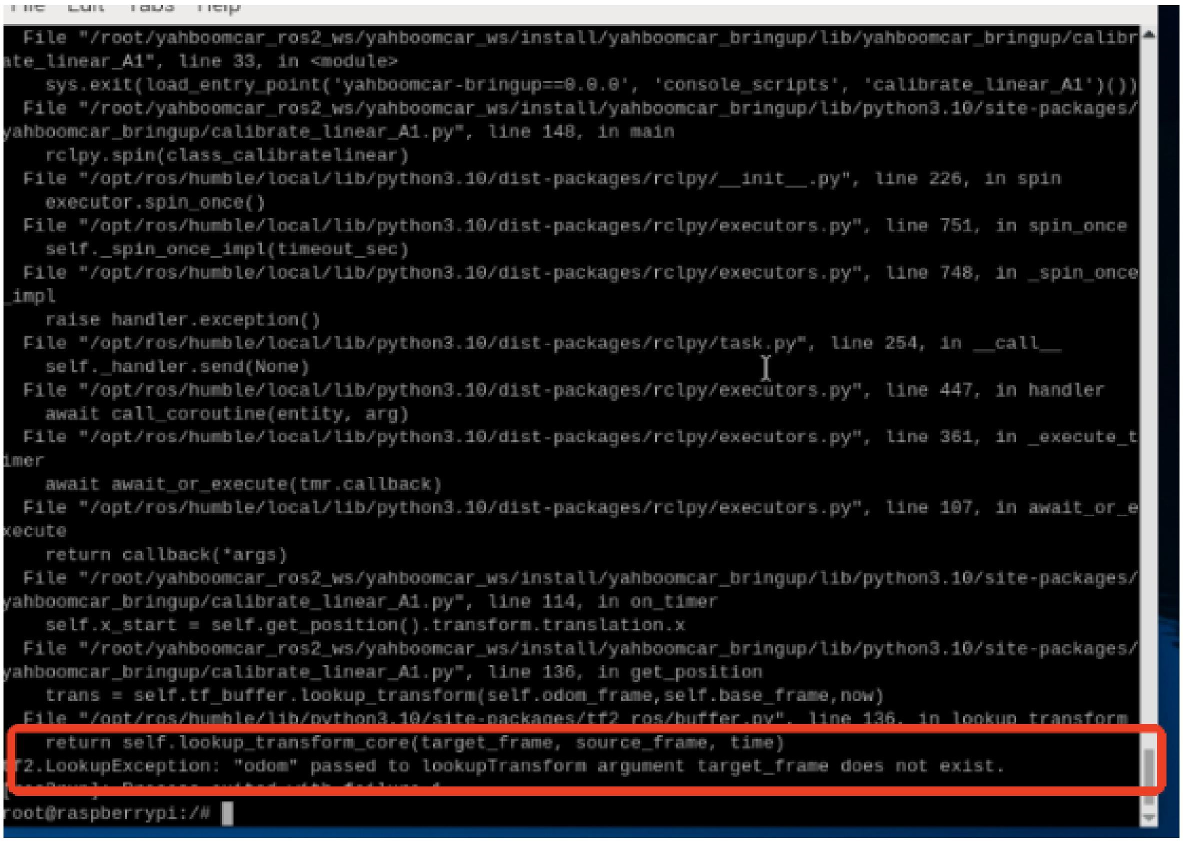
```
ros2 run yahboomcar_bringup calibrate_angular_A1
```

The following image is displayed successfully.



```
root@raspberrypi:/# ^C
root@raspberrypi:/# ros2 run yahboomcar_bringup calibrate_angular_A1
finish init work
[INFO] [1755159169.096285150] [calibrate_angular]: transform not ready
```

If the runtime error message indicates that no TF transformations are available, press **Ctrl+C** to exit the program and run it again.

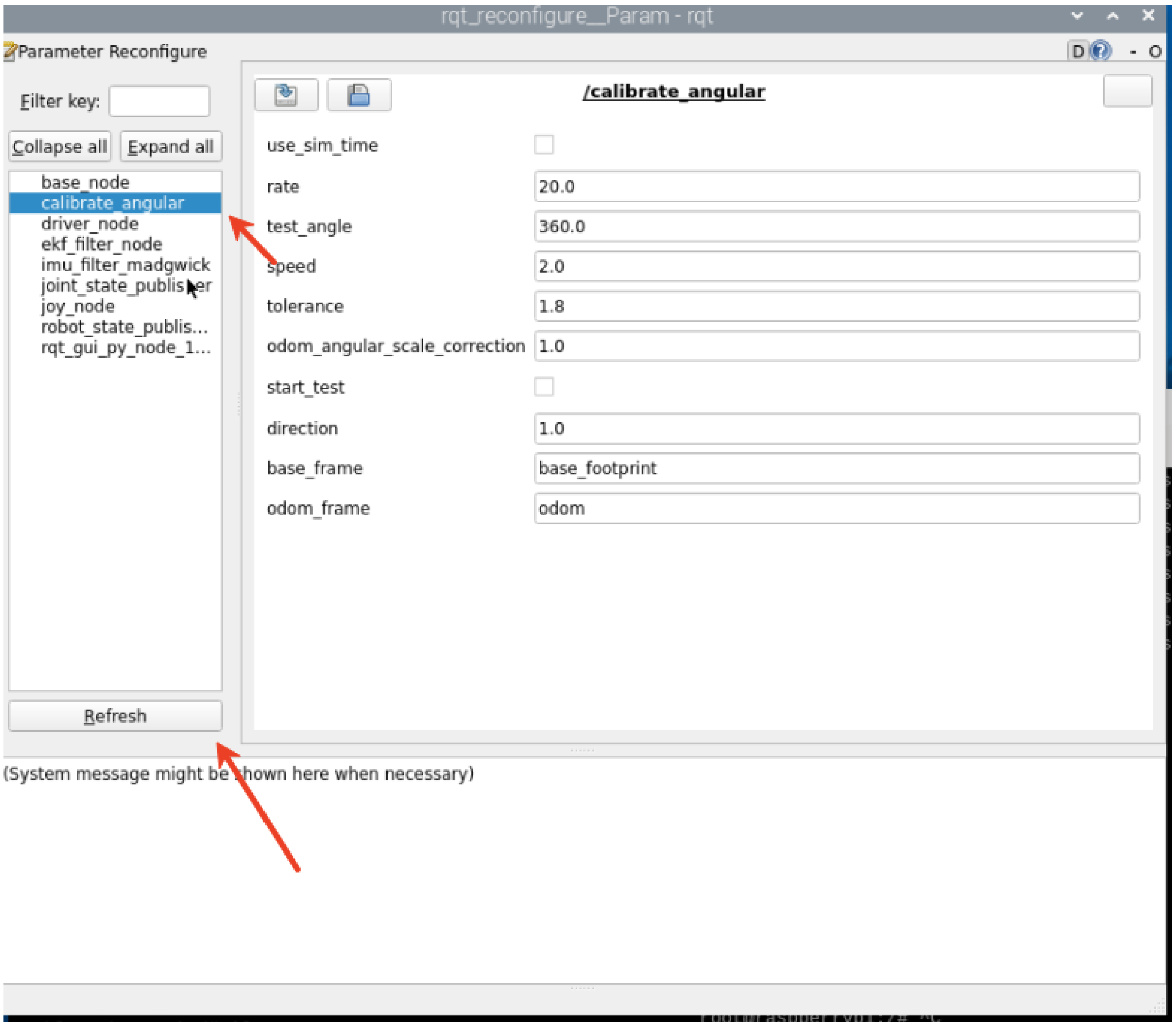


```
File  Edit  Tabs  Help
  File "/root/yahboomcar_ros2_ws/yahboomcar_ws/install/yahboomcar_bringup/lib/yahboomcar_bringup/calibr
ate_linear_A1", line 33, in <module>
    sys.exit(load_entry_point('yahboomcar-bringup==0.0.0', 'console_scripts', 'calibrate_linear_A1')())
  File "/root/yahboomcar_ros2_ws/yahboomcar_ws/install/yahboomcar_bringup/lib/python3.10/site-packages/
yahboomcar_bringup/calibrate_linear_A1.py", line 148, in main
    rclpy.spin(class_calibratelinear)
  File "/opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/__init__.py", line 226, in spin
    executor.spin_once()
  File "/opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py", line 751, in spin_once
    self._spin_once_impl(timeout_sec)
  File "/opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py", line 748, in _spin_once
_impl
    raise handler.exception()
  File "/opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/task.py", line 254, in __call__
    self._handler.send(None)
  File "/opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py", line 447, in handler
    await call_coroutine(entity, arg)
  File "/opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py", line 361, in _execute_t
imer
    await await_or_execute(tmr.callback)
  File "/opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py", line 107, in await_or_e
xecute
    return callback(*args)
  File "/root/yahboomcar_ros2_ws/yahboomcar_ws/install/yahboomcar_bringup/lib/python3.10/site-packages/
yahboomcar_bringup/calibrate_linear_A1.py", line 114, in on_timer
    self.x_start = self.get_position().transform.translation.x
  File "/root/yahboomcar_ros2_ws/yahboomcar_ws/install/yahboomcar_bringup/lib/python3.10/site-packages/
yahboomcar_bringup/calibrate_linear_A1.py", line 136, in get_position
    trans = self.tf_buffer.lookup_transform(self.odom_frame,self.base_frame,now)
  File "/opt/ros/humble/lib/python3.10/site-packages/tf2_ros/buffer.py", line 136, in lookup_transform
    return self.lookup_transform_core(target_frame, source_frame, time)
tf2.LookupException: "odom" passed to lookupTransform argument target_frame does not exist.
[ros2run]: Process exited with failure 1
root@raspberrypi:/#
```

Open the dynamic parameter adjuster and run the following command in the terminal:

```
ros2 run rqt_reconfigure rqt_reconfigure
```

Click the **calibrate_angular** node in the node options on the left:


Parameter Reconfigure

Filter key:

Collapse all | Expand all

base_node
calibrate_angular
driver_node
ekf_filter_node
imu_filter_madgwick
joint_state_publisher
joy_node
robot_state_publis...
rqt_gui_py_node_1...

Refresh

/calibrate_angular

use_sim_time	☐
rate	20.0
test_angle	360.0
speed	2.0
tolerance	1.8
odom_angular_scale_correction	1.0
start_test	☐
direction	1.0
base_frame	base_footprint
odom_frame	odom

(System message might be shown here when necessary)

Note: The above nodes may not appear when you first open the application. Click Refresh to see all nodes. The **calibrate_angular** node displayed is the node for calibrating angular velocity.

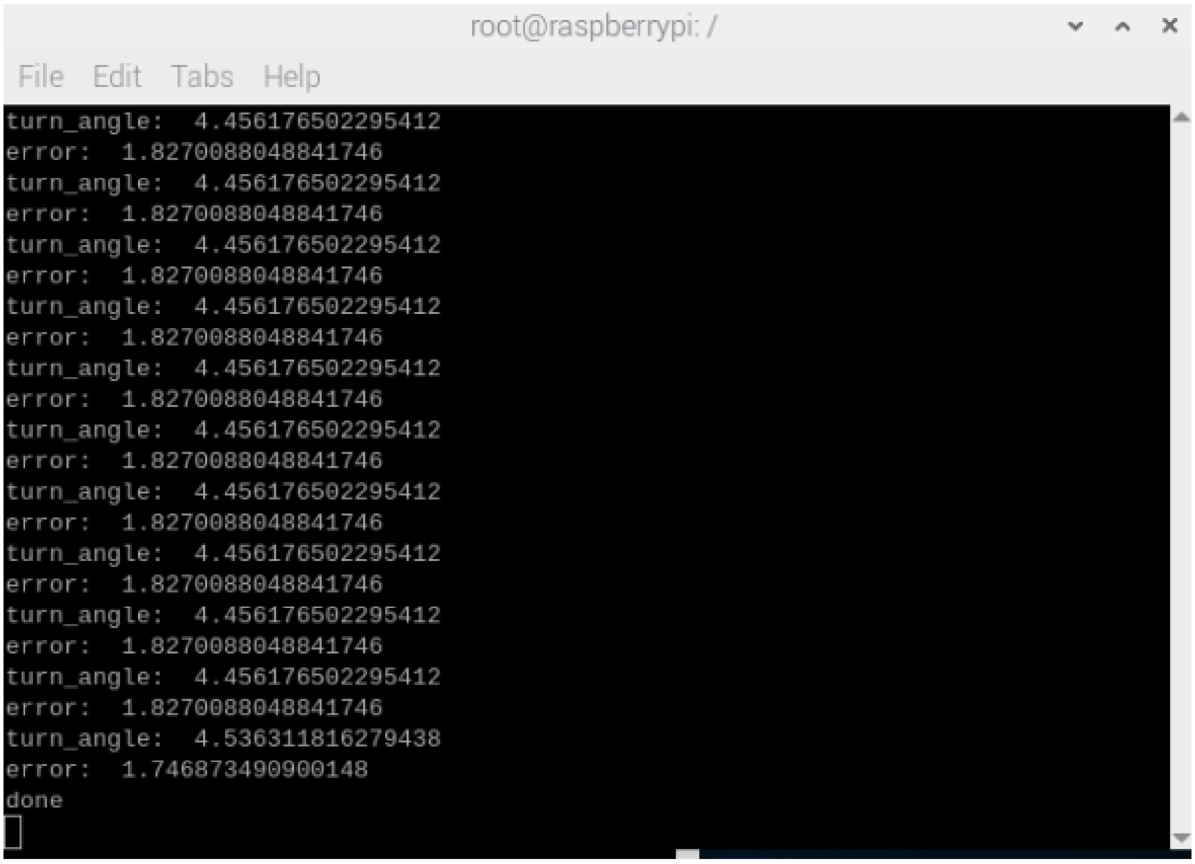
Other parameters in the rqt interface are described below:

- test_angle: The angle used for the calibration test. Here, a 360-degree rotation is tested.
- speed: The angular velocity.
- tolerance: The tolerance for error.
- odom_angular_scale_correction: The linear velocity scale factor. If the test results are unsatisfactory, adjust this value.
- start_test: Test on/off.
- base_frame: The name of the base coordinate system.
- odom_frame: The name of the odometry coordinate system.

2.2. Start Calibration

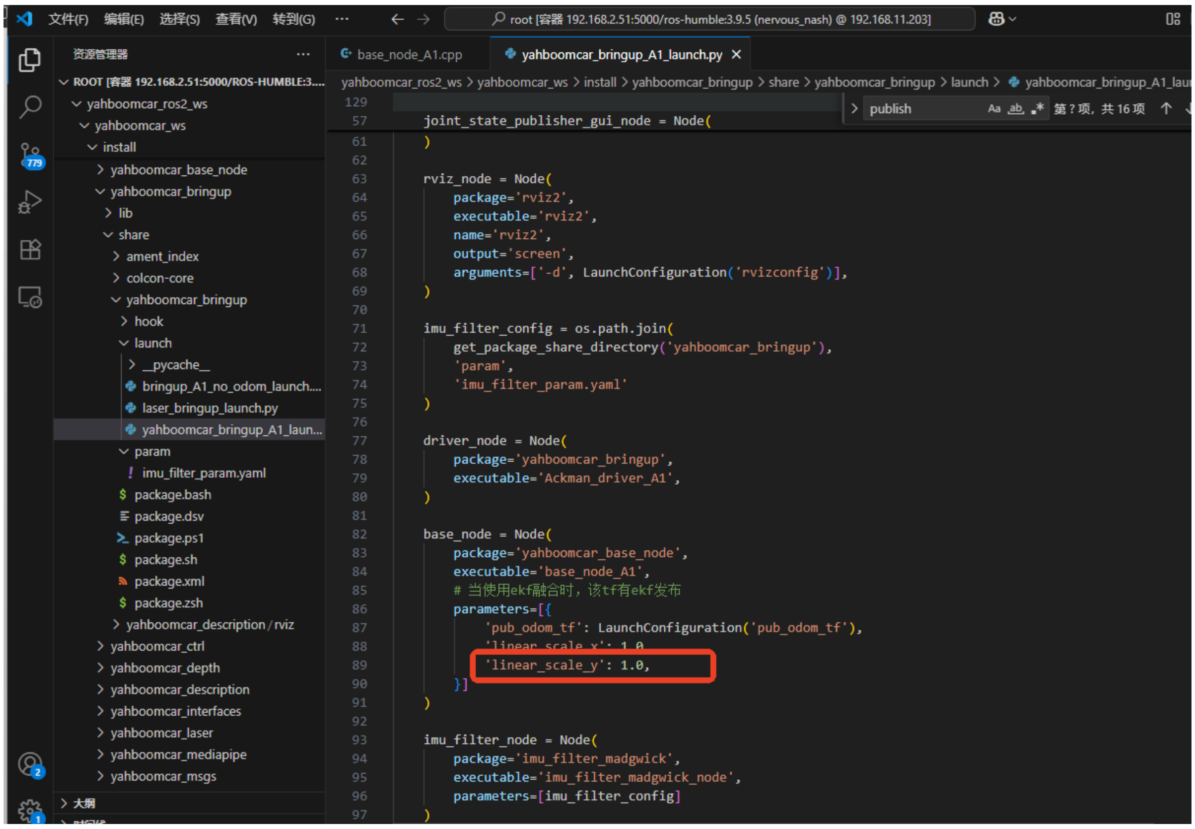
In the rqt_reconfigure interface, select the calibrate_angular node. Click the box to the right of the **start_test** node to begin calibration.

After clicking start_test, calibration begins. The car will monitor the TF transformations of base_footprint and odom, calculate the theoretical rotation angle, and issue a stop command when the error is less than the tolerance.



```
turn_angle:  4.456176502295412
error:  1.8270088048841746
turn_angle:  4.456176502295412
error:  1.8270088048841746
turn_angle:  4.456176502295412
error:  1.8270088048841746
turn_angle:  4.456176502295412
error:  1.8270088048841746
turn_angle:  4.456176502295412
error:  1.8270088048841746
turn_angle:  4.456176502295412
error:  1.8270088048841746
turn_angle:  4.456176502295412
error:  1.8270088048841746
turn_angle:  4.456176502295412
error:  1.8270088048841746
turn_angle:  4.456176502295412
error:  1.8270088048841746
turn_angle:  4.456176502295412
error:  1.8270088048841746
turn_angle:  4.536311816279438
error:  1.746873490900148
done
```

After the test is complete, remember the value of [odom_angular_scale_correction] and modify it to the linear_scale_y parameter in yahboomcar_bringup_A1_launch.py.

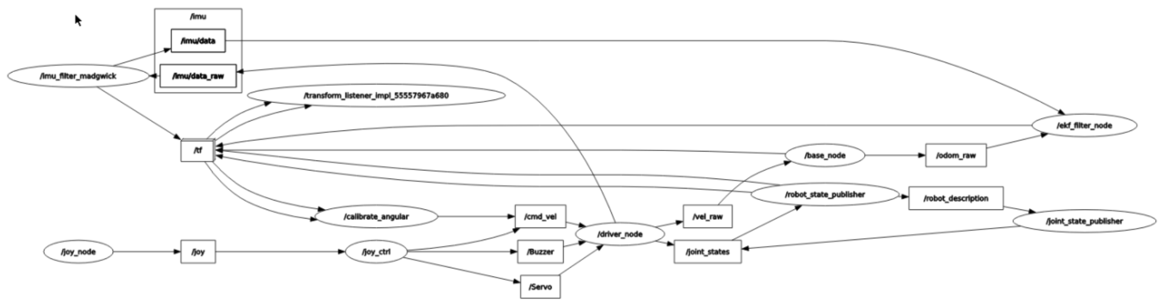


```
    joint_state_publisher_gui_node = Node(
    )

    rviz_node = Node(
        package='rviz2',
        executable='rviz2',
        name='rviz2',
        output='screen',
        arguments=['-d', LaunchConfiguration('rvizconfig')],
    )

    imu_filter_config = os.path.join(
        get_package_share_directory('yahboomcar_bringup'),
        'param',
        'imu_filter_param.yaml'
    )

    driver_node = Node(
        package='yahboomcar_bringup',
        executable='Ackman_driver_A1',
    )

    base_node = Node(
        package='yahboomcar_base_node',
        executable='base_node_A1',
        # 当使用ekf融合时，该tf有ekf发布
        parameters=[{
            'pub_odom_tf': LaunchConfiguration('pub_odom_tf'),
            'linear_scale_x': 1.0
            'linear_scale_y': 1.0,
        }]
    )

    imu_filter_node = Node(
        package='imu_filter_madgwick',
        executable='imu_filter_madgwick_node',
        parameters=[imu_filter_config]
    )
```

3. View the Node Relationship Graph

Open a terminal and enter the command:

```
ros2 run rqt_graph rqt_graph
```



In the above node relationship graph:

- The **imu_filter** node is responsible for filtering the chassis's raw IMU data **/imu/data** and publishing the filtered data **/imu/data**.
- The **/ekf_filter_node** node subscribes to the chassis's raw odometry data **/odom_raw** and filtered IMU data **/imu/data**, performs data fusion, and publishes the data to the **/odom** topic.
- The **calibrate_angular** node monitors the TF transformation between odom->base_footprint and publishes the /cmd_vel topic to control the robot's chassis movement.

4. Core Source Code Analysis

This program primarily utilizes TF to monitor coordinate transformations. By monitoring the coordinate transformation between base_footprint and odom, the robot can determine "how far I've walked" or "how many degrees I've turned."

Code path:

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_bringup/yahboomcar_bringup/cal
ibrate_angular_A1.py
```

Among them, the implementation of monitoring tf coordinate transformation is the get_odom_angle method in the Calibrateangular class:

```python
def get_odom_angle(self):
     try:
        now = rclpy.time.Time()
        rot = self.tf_buffer.lookup_transform(
            self.base_frame,
            self.odom_frame,
            now,
            timeout=rclpy.duration.Duration(seconds=1.0))
        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
```

The on_timer (timer callback function) method in the Calibrateangular class is used to determine the rotation angle of the robot chassis and control the chassis movement:

```
def on_timer(self):
    self.start_test =
self.get_parameter('start_test').get_parameter_value().bool_value
    self.odom_angular_scale_correction =
self.get_parameter('odom_angular_scale_correction').get_parameter_value().double
_value
    self.test_angle =
self.get_parameter('test_angle').get_parameter_value().double_value
    self.test_angle = radians(self.test_angle) #角度转成弧度
    self.speed = self.get_parameter('speed').get_parameter_value().double_value
    move_cmd = Twist()
    self.test_angle *= self.reverse
    #self.test_angle *= self.reverse
    #self.error = self.test_angle - self.turn_angle
    if self.start_test:
        self.error = self.turn_angle - self.test_angle
        if abs(self.error) > self.tolerance  :
            #move_cmd.linear.x = 0.2
            move_cmd.angular.z = copysign(self.speed, self.error)
            #print("angular: ",move_cmd.angular.z)
            self.cmd_vel.publish(move_cmd)
            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,flush=True)
            #self.error = self.test_angle - self.turn_angle
            print("error: ",self.error,flush=True)
            self.first_angle = self.odom_angle

            #print("first_angle: ",self.first_angle)
        else:
            self.error = 0.0
            self.turn_angle = 0.0
            print("done",flush=True)
            self.first_angle = 0
            self.reverse = -self.reverse
            self.start_test  =
rclpy.parameter.Parameter('start_test',rclpy.Parameter.Type.BOOL,False)
            all_new_parameters = [self.start_test]
            self.set_parameters(all_new_parameters)
    else:
        self.error = 0.0
        self.cmd_vel.publish(Twist())
        self.turn_angle = 0.0
        self.start_test  =
rclpy.parameter.Parameter('start_test',rclpy.Parameter.Type.BOOL,False)

        all_new_parameters = [self.start_test]
        self.set_parameters(all_new_parameters)
```