Linear velocity calibration

This lesson uses the Raspberry Pi as an example. For Raspberry Pi and Jetson Nano , 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 , 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 linear speed. To visually demonstrate the linear speed, instruct the car to move straight forward for 1 meter and observe its actual distance to see if it is within the error range.

2. Starting the Program

2.1. Startup Commands

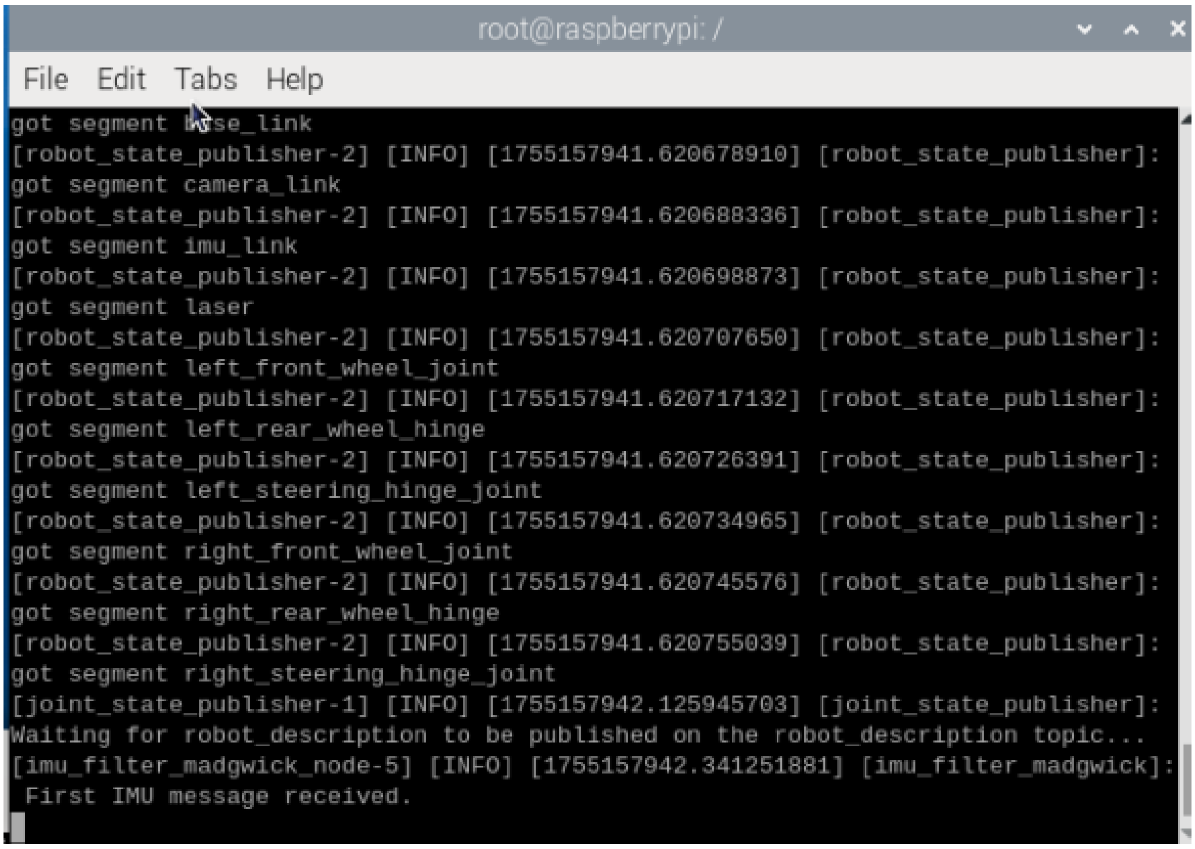
For the Raspberry Pi 5 controller, you must first enter the Docker container. For the Orin controller, this is not necessary.

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.

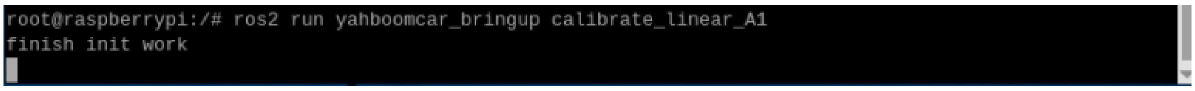
```
ros2 launch yahboomcar_bringup yahboomcar_bringup_A1_launch.py
```



```
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.
```

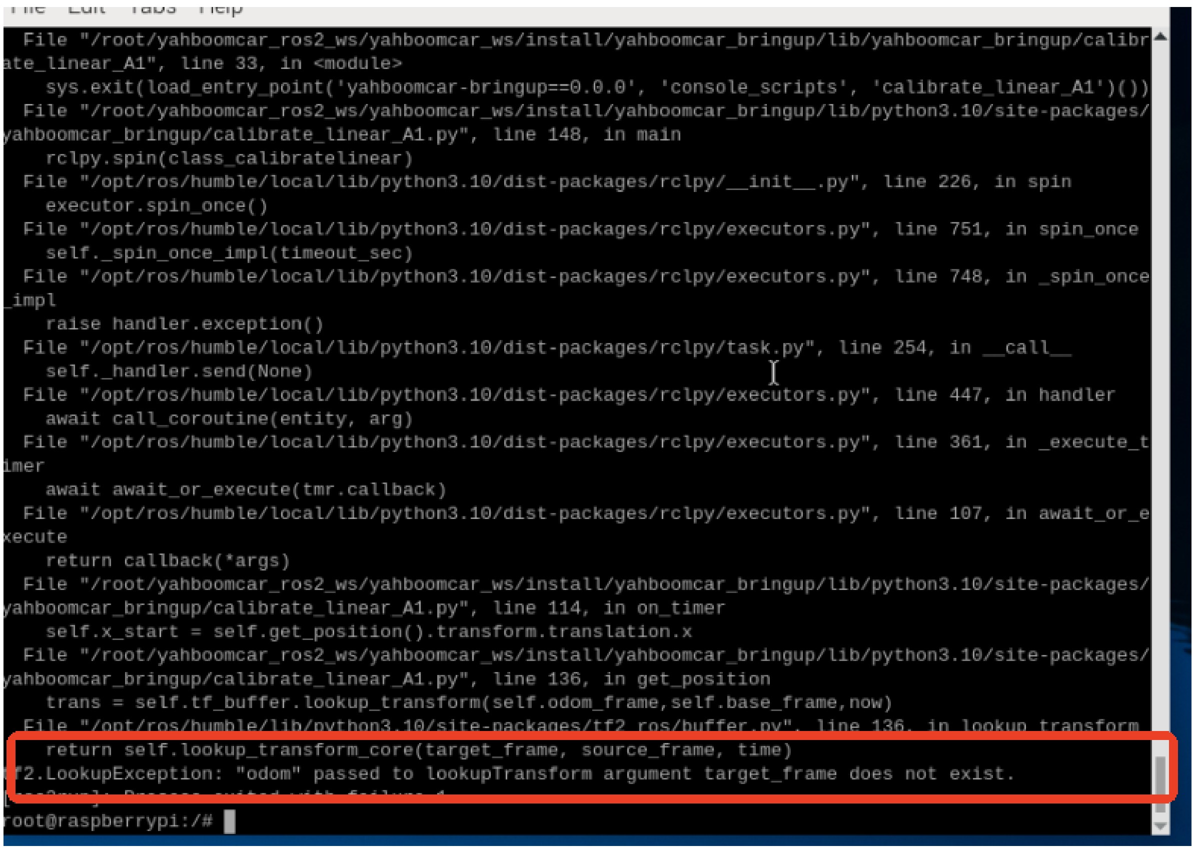
```
ros2 run yahboomcar_bringup calibrate_linear_A1
```

The following image is displayed successfully.



```
root@raspberrypi:/# ros2 run yahboomcar_bringup calibrate_linear_A1
finish init work
```

If the following error message appears during runtime indicating that no TF transformations were found, press **Ctrl+C** to exit the program and run it again.

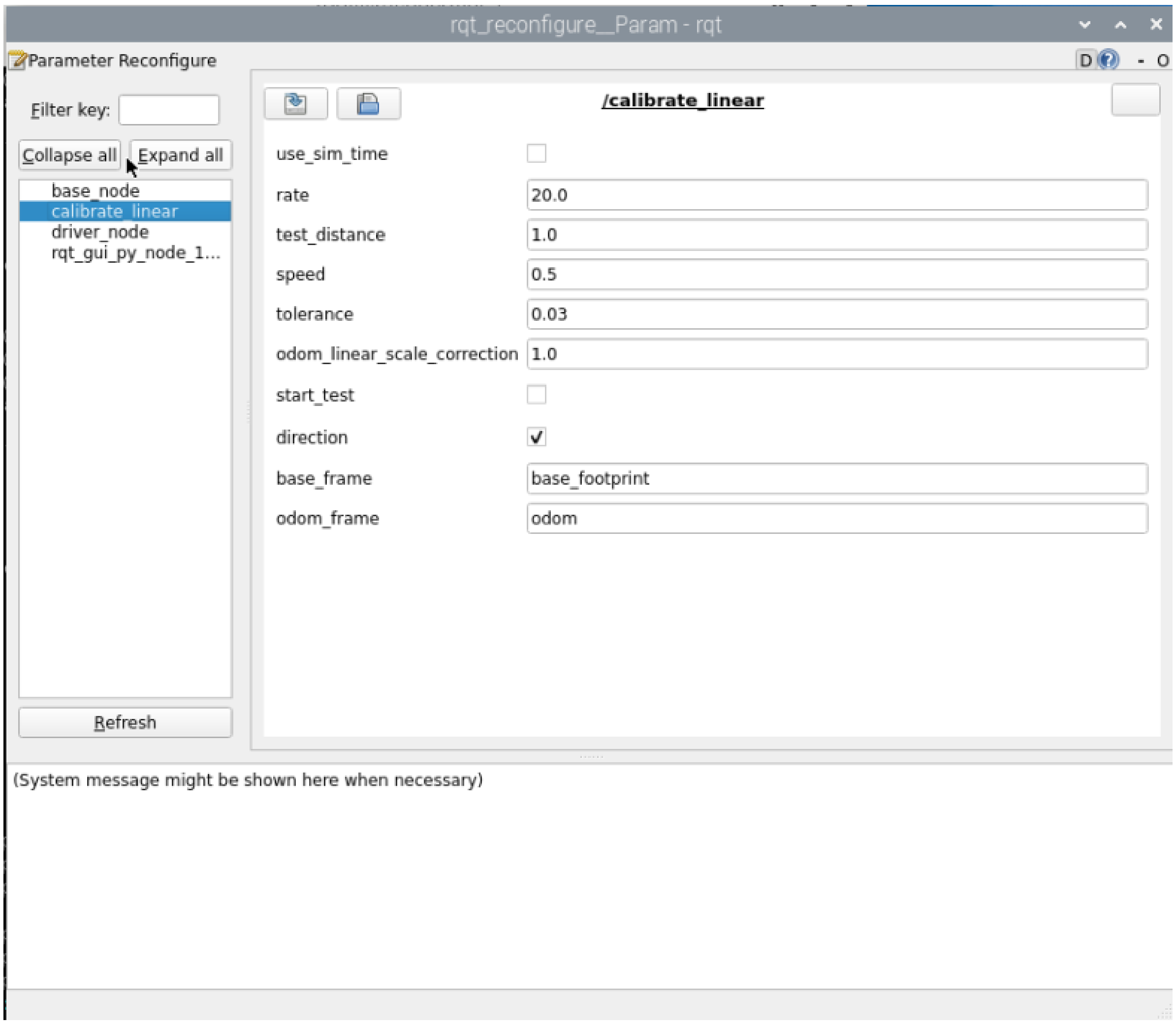


```
  File "/root/yahboomcar_ros2_ws/yahboomcar_ws/install/yahboomcar_bringup/lib/yahboomcar_bringup/calibr
ate_linear_A1", line 33, in <module>
    sys.exit(load_entry_point('yahboomcar-bringup==0.0.0', 'console_scripts', 'calibrate_linear_A1')())
  File "/root/yahboomcar_ros2_ws/yahboomcar_ws/install/yahboomcar_bringup/lib/python3.10/site-packages/
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.
root@raspberrypi:/# 
```

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

```
ros2 run rqt_reconfigure rqt_reconfigure
```

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



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)

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

The rqt interface parameters are described as follows:

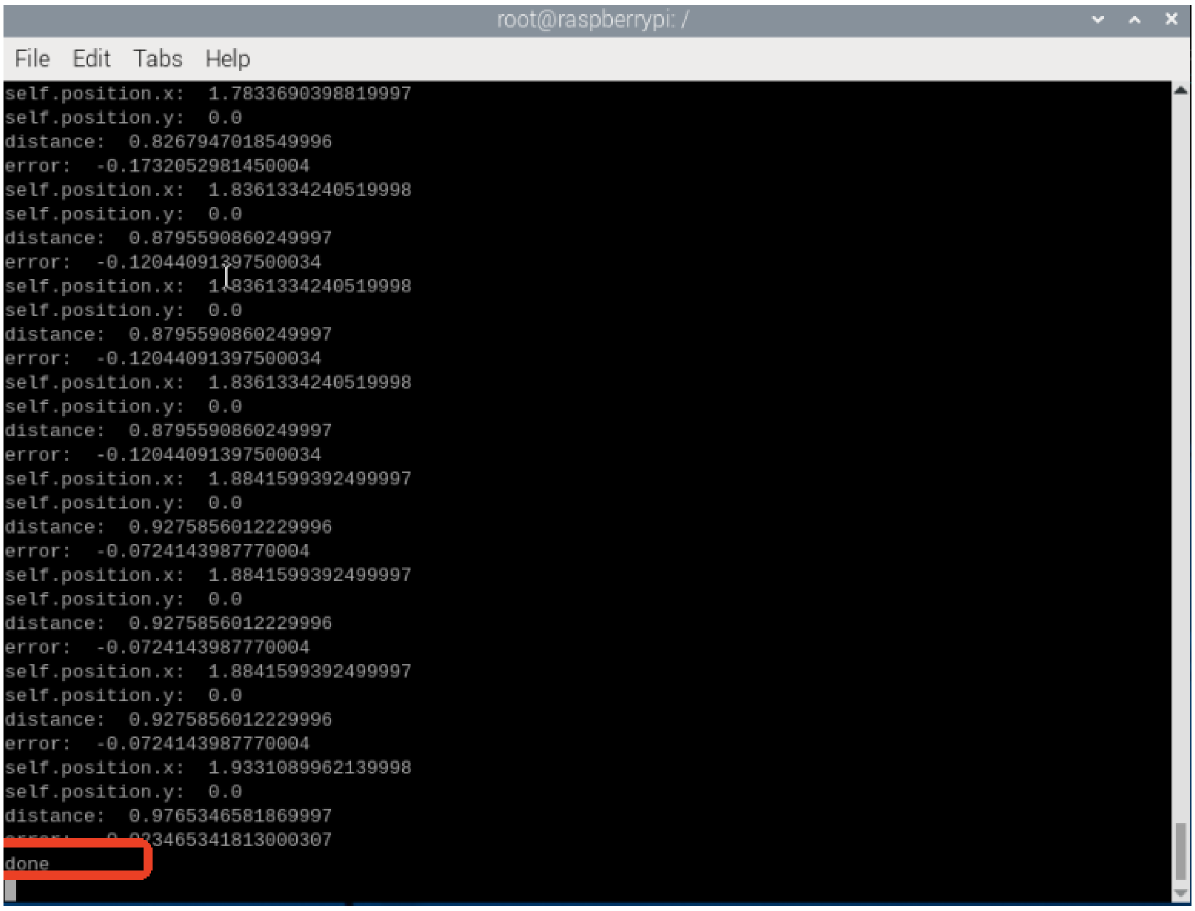
- test_distance: Calibration test distance. Here, the test is a 1-meter forward movement.
- speed: Linear speed.
- tolerance: Error tolerance.
- odom_linear_scale_correction: Linear speed scaling factor. If the test results are unsatisfactory, adjust this value.
- start_test: Test switch.
- direction: Can be ignored. This value is used for the McLennan robot. Modifying it allows you to calibrate the linear speed of left and right movements.
- 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_linear node (click **Refresh** if it doesn't appear).

Select a reference of known length on the ground (a tape measure, tile, etc.). Change **test_distance** to the actual test distance (1 meter is used as an example). Click the **start_test** box to begin calibration.

After clicking start_test, calibration begins. The robot will monitor the TF transformations of base_footprint and odom, calculate the theoretical distance traveled, and when the error is less than tolerance, issue a stop command and the terminal will print "done." If the actual distance traveled is less than 1 meter, increase the **odom_linear_scale_correction** parameter appropriately. After making these changes, click a blank space, click start_test again, reset the start_test value, and click start_test again to complete the calibration. Modifying other parameters is similar; click a blank space to write the modified parameters. Record the final calibrated **odom_linear_scale_correction** parameter.

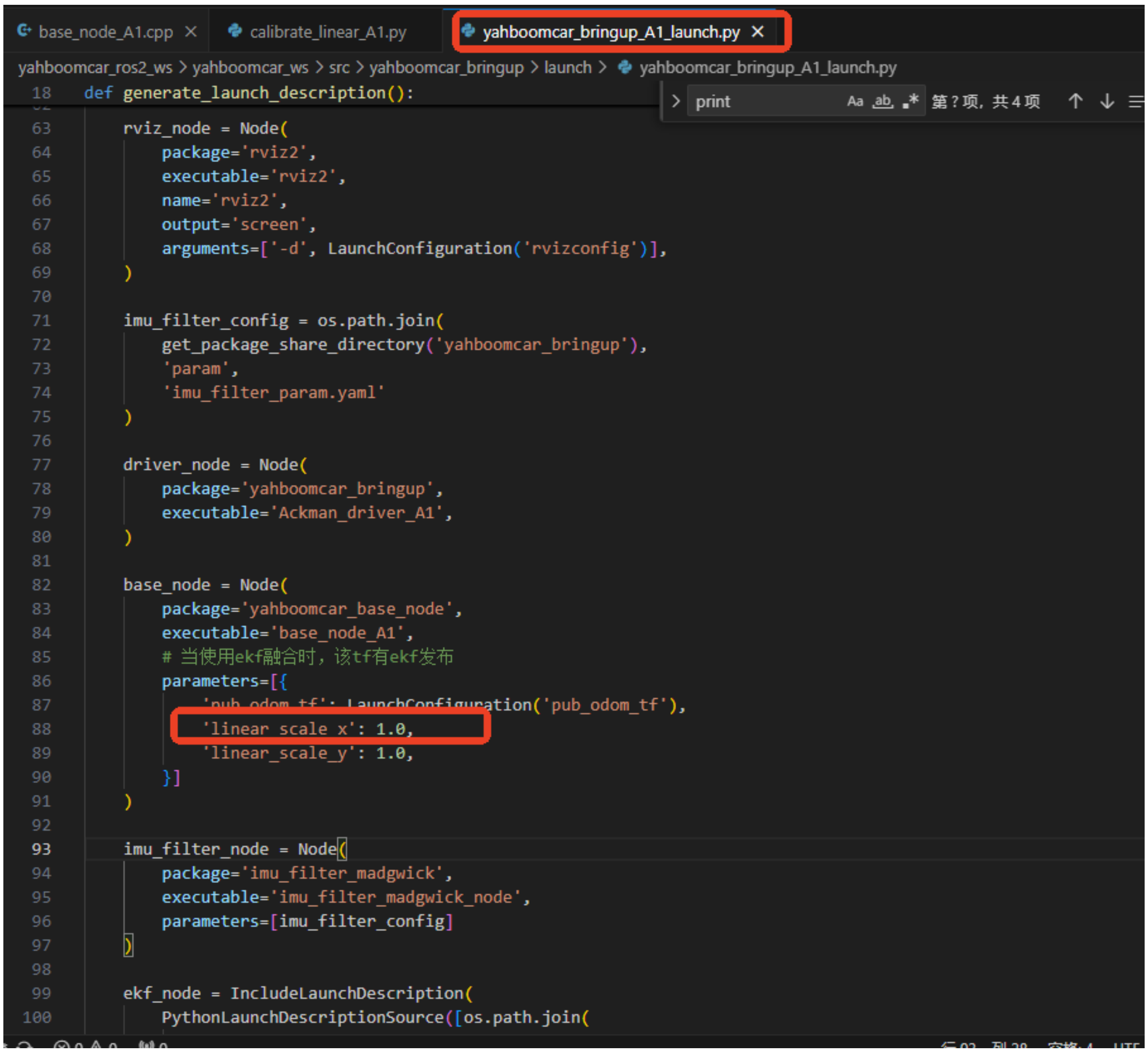


```
self.position.x:  1.7833690398819997
self.position.y:  0.0
distance:  0.8267947018549996
error:  -0.1732052981450004
self.position.x:  1.8361334240519998
self.position.y:  0.0
distance:  0.8795590860249997
error:  -0.12044091397500034
self.position.x:  1.8361334240519998
self.position.y:  0.0
distance:  0.8795590860249997
error:  -0.12044091397500034
self.position.x:  1.8361334240519998
self.position.y:  0.0
distance:  0.8795590860249997
error:  -0.12044091397500034
self.position.x:  1.8841599392499997
self.position.y:  0.0
distance:  0.9275856012229996
error:  -0.0724143987770004
self.position.x:  1.8841599392499997
self.position.y:  0.0
distance:  0.9275856012229996
error:  -0.0724143987770004
self.position.x:  1.8841599392499997
self.position.y:  0.0
distance:  0.9275856012229996
error:  -0.0724143987770004
self.position.x:  1.9331089962139998
self.position.y:  0.0
distance:  0.9765346581869997
error:  -0.023465341813000307
done
```

After testing, remember the value of odom_linear_scale_correction and modify it to the linear_scale_x parameter in yahboomcar_bringup_A1_launch.py.

Path:

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_bringup/launch/yahboomcar_bringup_A1_launch.py
```

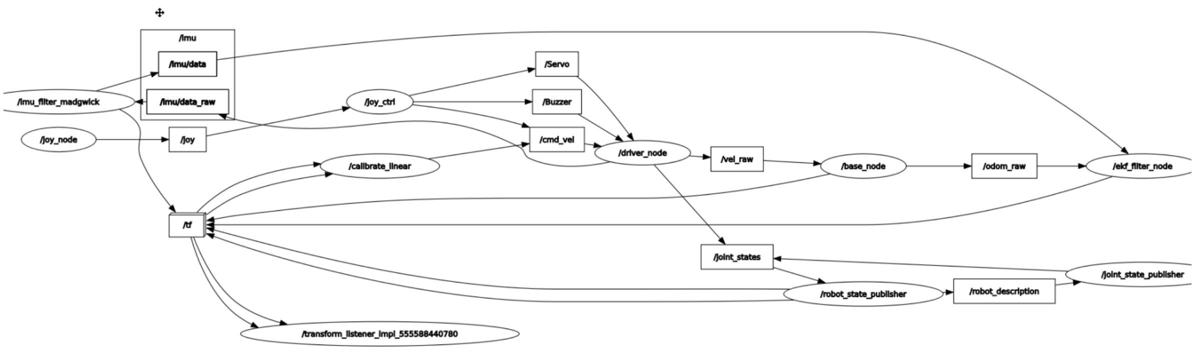

base_node_A1.cpp
calibrate_linear_A1.py
yahboomcar_bringup_A1_launch.py
yahboomcar_ros2_ws > yahboomcar_ws > src > yahboomcar_bringup > launch > yahboomcar_bringup_A1_launch.py
print
第？项，共4项

```
def generate_launch_description():
    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]
    )

    ekf_node = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([os.path.join(
```

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 filters the chassis' raw IMU data (/imu/data) and publishes the filtered data (/imu/data).
- The ekf_filter_node node subscribes to the chassis' raw odometry data (/odom_raw) and filtered IMU data (/imu/data), fuses the data, and publishes it to the topic (/odom).

- The calibrate_linear node monitors the TF transformation from odom to base_footprint and publishes to the topic (/cmd_vel) to control the robot's chassis movement.

4. Core Source Code Analysis

This program primarily utilizes TF monitoring of coordinate transformations. By monitoring the coordinate transformation between base_footprint and odom, the robot can determine "how far I've traveled" 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_position method in the CalibrateLinear class:

```
def get_position(self):
     try:
        now = rclpy.time.Time()
        transform = self.tf_buffer.lookup_transform(
            self.base_frame,
            self.odom_frame,
            now,
            timeout=rclpy.duration.Duration(seconds=1.0))
        return transform

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

The on_timer method (timer callback function) in the CalibrateLinear class is used to determine the displacement of the robot chassis and control its movement:

```
def on_timer(self):
    move_cmd = Twist()
    #self.get_param()
    self.start_test =
self.get_parameter('start_test').get_parameter_value().bool_value
    self.odom_linear_scale_correction =
self.get_parameter('odom_linear_scale_correction').get_parameter_value().double_
value
    self.rate = self.get_parameter('rate').get_parameter_value().double_value
    self.test_distance =
self.get_parameter('test_distance').get_parameter_value().double_value
    self.direction =
self.get_parameter('direction').get_parameter_value().double_value
    self.tolerance =
self.get_parameter('tolerance').get_parameter_value().double_value
    self.speed = self.get_parameter('speed').get_parameter_value().double_value
    if self.start_test:
        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
        print("distance: ",distance)
        error = distance - self.test_distance
        print("error: ",error)
        #start = time()
        if not self.start_test or abs(error) < self.tolerance:
            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)

            print("done")
        else:
            move_cmd.linear.x = copysign(self.speed, -1 * error)
            '''if self.direction:
                print("x")
                move_cmd.linear.x = copysign(self.speed, -1 * error)
            else:
                move_cmd.linear.y = copysign(self.speed, -1 * error)
                print("y")'''
        self.cmd_vel.publish(move_cmd)

    else:
        self.x_start = self.get_position().transform.translation.x
        self.y_start = self.get_position().transform.translation.y

        self.cmd_vel.publish(Twist())
```