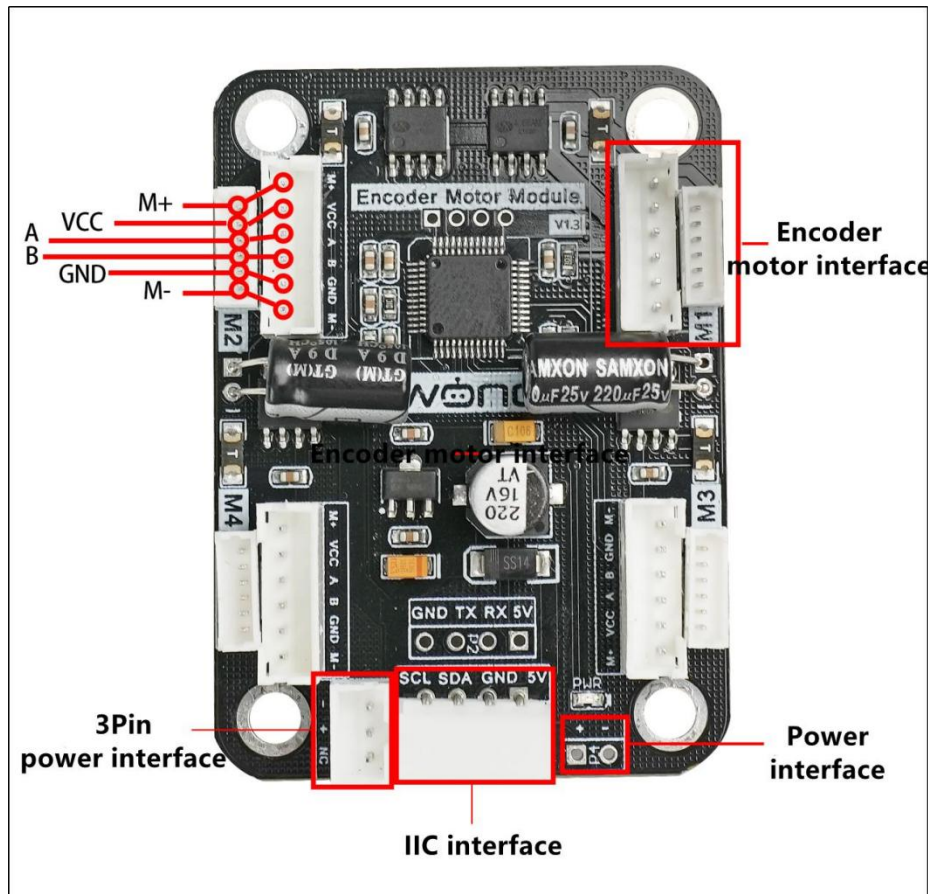


# Part 2 Encoder Motor Driver

## 1. Introduction to Module

This motor driver can work with microcontroller to drive TT motor and encoder motor. And its working voltage is DC 3V-12V. For interface layout, you can refer to the picture below.



The functions of the interface are listed below.

| Interface               | NO.                                                                                                                                                                                                                                           | Function                                |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|
| Encoder motor interface | GND                                                                                                                                                                                                                                           | negative pole of Hall power supply      |
|                         | A                                                                                                                                                                                                                                             | output terminal of A-phase pulse signal |
|                         | B                                                                                                                                                                                                                                             | output terminal of B-phase pulse signal |
|                         | VCC                                                                                                                                                                                                                                           | positive pole of Hall power supply      |
|                         | M+                                                                                                                                                                                                                                            | positive pole of motor power supply     |
|                         | M-                                                                                                                                                                                                                                            | negative pole of motor power supply     |
|                         | Note:                                                                                                                                                                                                                                         |                                         |
|                         | 1. The voltage between VCC and GND depends on the voltage of the power supply adopted by micro controller. In general, it is 3.3V or 5V.                                                                                                      |                                         |
|                         | 2. When the main shaft rotates clockwise, A terminal will output A-phase pulse signal first, and B terminal follows. When the main shaft rotates counterclockwise, B terminal will output A-phase pulse signal first, and A terminal follows. |                                         |
|                         | 3. The voltage between M+ and M- is determined by the voltage of the motor connected.                                                                                                                                                         |                                         |

|                      |     |                     |
|----------------------|-----|---------------------|
| IIC interface        | SCL | Clock line          |
|                      | SDA | dual-end data line  |
|                      | GND | ground line         |
|                      | 5V  | 5V DC output        |
| 3Pin power interface | -   | power negative pole |
|                      | +   | power positive pole |
|                      | nc  | none                |
| Power interface      | +   | power positive pole |
|                      | -   | power negative pole |

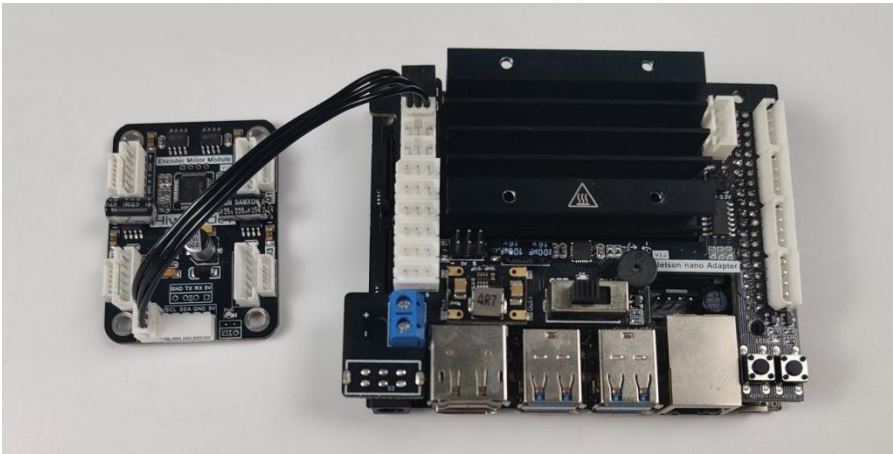
**Note:**

power supply through IIC interface will engender instability, therefore 6V-9V independent power supply is recommended.

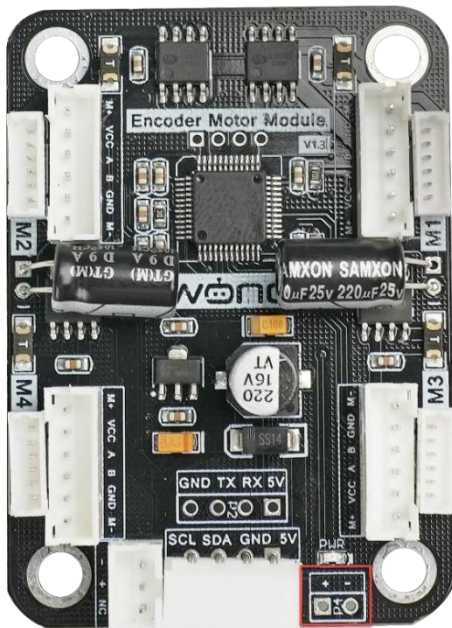
## 2. Wiring

### 2.1 Power Supply Wiring

1) Take Jetson Nano expansion board for example. Connect the expansion board to the driver module through 3pin wire.

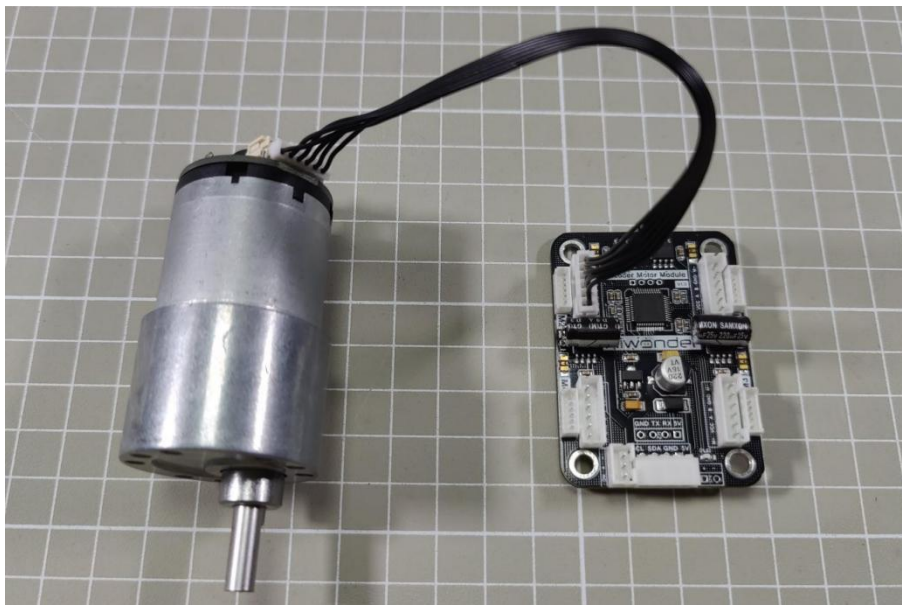


2) Power supply directly through the power interface. The pin header needs to be soldered on the module before power supply as marked in the picture, which is suitable for independent developers.



## 2.2 Motor Wiring

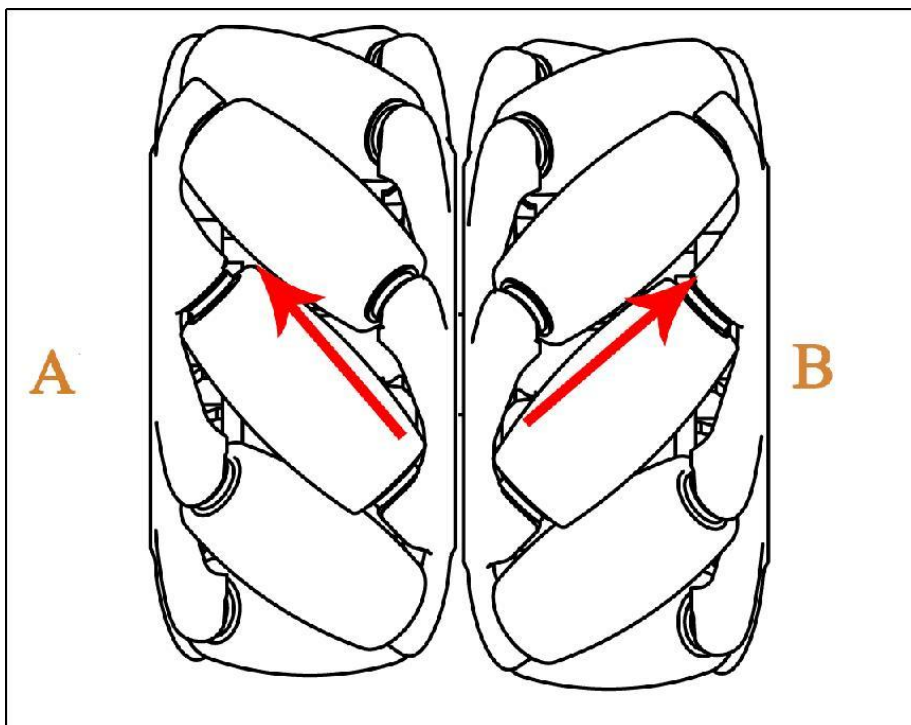
Take Hall encoder geared motor for example.



## Part 3 Working Logic of Mecanum Chassis

### 1. Preface

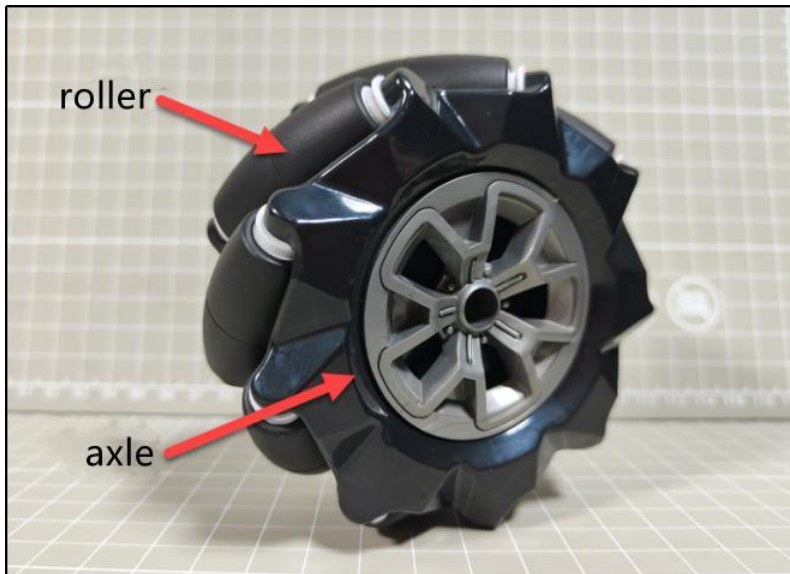
JetAuto adopts **omnidirectional movement mecanum wheels**. According to the included angle  $45^\circ$  between **the roller and axle** of the mecanum wheels, the wheels can be divided into **wheel A** and **wheel B** which are in mirror-image relationship with each other, as the figure shown below:



Featuring  $360^\circ$  movement, flexibility and stability, the mecanum wheel is an excellent omnidirectional wheel. The combination of four mecanum wheels can be more flexible to realize the omnidirectional movement.

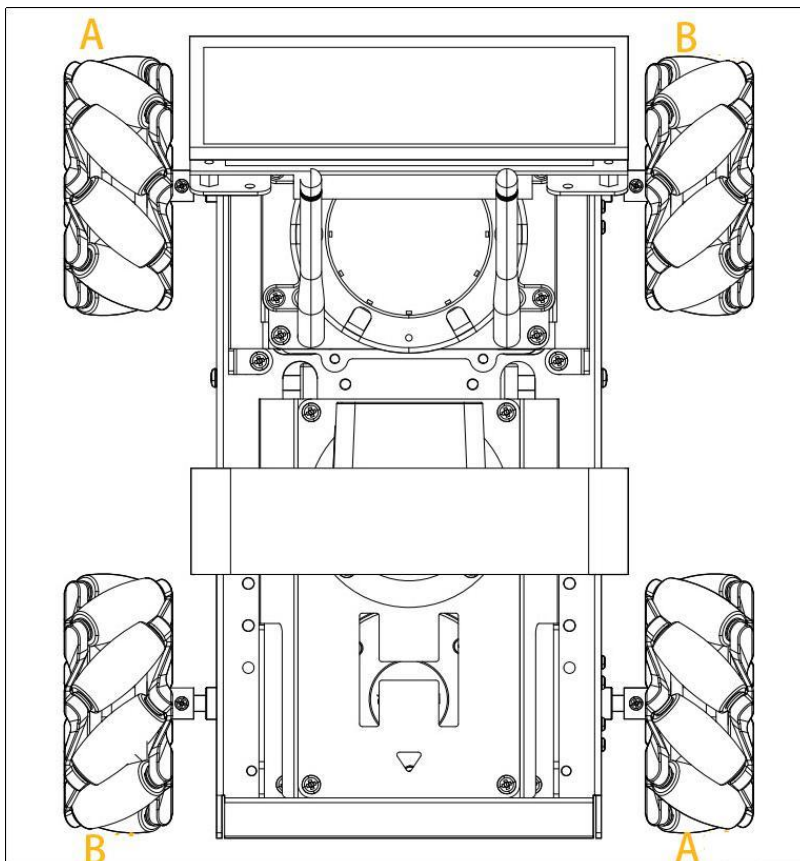
## 2. Working Principle

### 2.1 Mecanum Wheel Hardware Structure



The mecanum wheel is composed of rollers and axle. The axle serves as the main bracket of the whole wheel, and rollers are attached to the axle. The axle axis is at a 45-degree angle to the roller axis. In general, mecanum wheels work in a group of four, including two left wheels and two right wheels. A wheel and B wheel are symmetrical.

There are several combination of four mecanum wheels, such as AAAA, BBBB, AABB, ABAB, BABA. Not all combinations of the wheels enable robot car to move forward, backward, and sideways, etc. The combination of JetAuto's wheels are ABAB, which can realize omnidirectional movement.



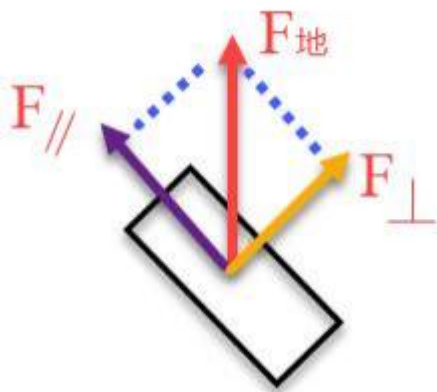


## 2.2 Mecanum Wheel Physical Characteristics

The omni-directional motion of the vehicle is achieved as the vector summation of propelling forces on the ground-engaging rollers can be in any direction by adjusting the wheel rotation direction and torque magnitude of the four wheels.

Due to the rollers at its rim oriented at an certain angle to the wheel circumference, the mecanum wheels can slip in sideways direction. The generatrix of small rollers are special. When the mecanum wheel revolves around its fixed axle, the envelope of each small roller is a cylindrical surface so that the wheel can roll forward continuously.

## 3. Motion Mode Analysis



Let us consider the example of a wheel turning forward, and let's focus on **the rollers that are in contact with the ground**. When the wheel turns forward, the roller that is in contact with the ground can be treated as a static point, and the ground will exert a **forward friction force** on the roller. We can decompose this friction force into two components, **one parallel to the roller** and the other **perpendicular to the roller**.

The component of **the force perpendicular to the roller will cause the roller to rotate**, resulting in rolling friction. However, this force is usually very small and will not significantly affect the movement of the wheel. We can **assume that this force will be dissipated by the rolling of the roller**.

However, **due to physical limitations**, the roller cannot roll in a direction parallel to its axis, resulting in sliding friction. **This force is the main contributor to the movement of the wheel**, and its direction is naturally the direction in which the wheel moves when rolling forward. In other words, **the wheel will move forward to the left, as shown in the figure**.

**Overall, the force parallel to the roller axis is the force that plays a critical role in the movement of the wheel, while the force perpendicular to the roller axis only results in rolling friction, which is negligible.**

# Part 4 Omnidirectional Movement Control

## 1. Program Logic

The four wheels of JetAuto work in **BABA mode** to move in all direction.

Through **motion control node, jetauto\_controller**, the program **publishes the message about the set linear velocity and angular velocity**. Then **analyze the linear velocity and angular velocity to control the motors on Mecanum wheel chassis to rotate** so as to make JetAuto move toward the corresponding direction.

The source code of this program is stored in

**/home/jetauto\_ws/src/jetauto\_driver/jetauto\_controller/scripts/jetauto\_controller\_main.py**

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import os # Provides functions for interacting with the operating system.
import sys # Provides functions for system-specific parameters and functions.
import math # Provides mathematical functions.
import rospy # ROS client library for Python.
from std_srvs.srv import Trigger # Service Message (Trigger)
from jetauto_sdk.mecanum import MecanumChassis # Class for controlling a
mecanum chassis from a specific JetAuto SDK (likely custom or third-party).
from geometry_msgs.msg import Twist, TransformStamped
# Class for representing a Twist message in ROS (linear and angular velocities).

class JetAutoController:
    def __init__(self):
        rospy.init_node('jetauto_controller', anonymous=False) # Initializes a
        ROS node with the name jetauto_controller.

        self.last_linear_x = 0
        self.last_linear_y = 0
        self.last_angular_z = 0
        # Sets initial values for last_linear_x, last_linear_y, and
        last_angular_z to keep track of previously received velocities.

        self.mecanum = MecanumChassis(a =103, b=97, wheel_diameter=96.5,
pulse_per_cycle=4320)#44.0 * 178.0
        # specifying robot parameters like wheel distance and encoder pulses
        per cycle

        self.machine_type = os.environ.get('MACHINE_TYPE') # get the value
        named 'MACHINE_TYPE' from the system's environment variable and assigns it to
        self.machine_type.
        self.go_factor = rospy.get_param('~go_factor', 1.00)
        self.turn_factor = rospy.get_param('~turn_factor', 1.00)
        # Get the values of the parameters 'go_factor' and 'turn_factor' from
        the ROS parameter server, or use the default value of 1.0 if the parameters
        don't exist, and store them in self.go_factor and self.turn_factor

        # Created two subscribers and a service
        rospy.Subscriber('jetauto_controller/cmd_vel', Twist,
self.cmd_vel_callback)
```

# This code line creates a subscriber named 'jetauto\_controller/cmd\_vel' that receives messages of type Twist and calls the self.cmd\_vel\_callback method to process the incoming message when it is received.

```
rospy.Subscriber('cmd_vel', Twist, self.app_cmd_vel_callback)
```

# The code line creates a subscriber named 'cmd\_vel' that also receives messages of type Twist, but calls the self.app\_cmd\_vel\_callback method to process the incoming message when it is received.

```
rospy.Service('jetauto_controller/load_calibrate_param', Trigger, self.load_calibrate_param)
```

# This line of code creates a service called 'jetauto\_controller/load\_calibrate\_param' that uses the Trigger type as the message type for requests and responses, and calls the self.load\_calibrate\_param method when a request is received. This means that when a node requests the service, the self.load\_calibrate\_param method will be called to process the request and return the response.

```
def load_calibrate_param(self, msg):
```

```
    self.go_factor = rospy.get_param('~go_factor', 1.00)
```

```
    self.turn_factor = rospy.get_param('~turn_factor', 1.00)
```

# Get the values of the parameters 'go\_factor' and 'turn\_factor' from the ROS parameter server, or use the default value of 1.0 if the parameters don't exist, and store them in self.go\_factor and self.turn\_factor

```
    rospy.loginfo('load_calibrate_param')
```

# This line of code is used to record a message in the ROS log indicating that parameter loading is complete.

```
    return [True, 'load_calibrate_param']
```

```
def app_cmd_vel_callback(self, msg): # A method called app_cmd_vel_callback is defined that takes a msg argument that is a Twist message containing information about the robot's linear and angular velocities.
```

```
    if msg.linear.x > 0.1:
```

```
        msg.linear.x = 0.1
```

# This line of code checks to see if the linear x-axis velocity (msg.linear.x) in the msg is greater than 0.1, and if it is, sets it to 0.1 to limit the maximum linear x-axis velocity.

```
    if msg.linear.x < -0.1:
```

```
        msg.linear.x = -0.1
```

# This line of code checks to see if the linear x-axis velocity (msg.linear.x) in the msg is less than -0.1, and if it is, sets it to -0.1 to limit the linear x-axis velocity to a minimum value.

```
    if msg.angular.z > 0.3:
```

```

    msg.angular.z = 0.3
    # This line of code checks to see if the angular velocity
(msg.angular.z) in msg is greater than 0.3, and if it is, sets it to 0.3 to
limit the angular velocity to a maximum value.

    if msg.angular.z < -0.3:
        msg.angular.z = -0.3
        # This line of code checks to see if the angular velocity
(msg.angular.z) in the msg is less than -0.3, and if so, sets it to -0.3 to
limit the angular velocity to a minimum value.
        self.cmd_vel_callback(msg)
        # Delivers the constrained message to the cmd_vel_callback method for
processing.

def cmd_vel_callback(self, msg):
    linear_x = self.go_factor*msg.linear.x
    linear_y = self.go_factor*msg.linear.y
    angular_z = self.turn_factor*msg.angular.z
    #These three lines of code calculate the linear x-axis velocity
(linear_x), the linear y-axis velocity (linear_y), and the angular velocity
(angular_z). These velocity values are scaled according to self.go_factor and
self.turn_factor to adjust the velocities to the actual robot.

    speed_up = False
    if abs(self.last_linear_x - linear_x) > 0.2 or abs(self.last_linear_y -
linear_y) > 0.2 or abs(self.last_angular_z - angular_z) > 1:
        speed_up = True
    # This code is used to detect if acceleration is required. It does this
by comparing the difference between the current speed and the last speed, and
if the difference exceeds a certain threshold, it sets the speed_up flag to
True, indicating that acceleration is required. This is done to respond faster
to speed changes.

    self.last_linear_x = linear_x
    self.last_linear_y = linear_y
    self.last_angular_z = angular_z

    linear_x_, linear_y_ = linear_x * 1000.0, linear_y * 1000.0 #mm to m

    speed = math.sqrt(linear_x_ ** 2 + linear_y_ ** 2) # Robot synthesis
speed "speed"
    direction = math.atan2(linear_y_, linear_x_) # Direction of movement
    direction = math.pi * 2 + direction if direction < 0 else direction
    self.mecanum.set_velocity(speed, direction, angular_z,
speed_up=speed_up)
    # This line of code calls the set_velocity() method of the chassis
controller mecanum, passing information about the synthesized velocity,

```



direction of motion, angular velocity, and whether or not acceleration is needed to control the robot's motion

```
if __name__ == "__main__":
    try:
        node = JetAutoController() # An instance of the JetAutoController class
        # is first created
        rospy.spin() # Here, the program will wait for the ROS node to exit.
    except Exception as e:
        rospy.logerr(str(e)) # If an exception occurs during execution, the
        # exception message is caught and logged to the ROS log via the rospy.logerr()
        # method.
    finally:
        node.mecanum.reset_motors() # Used to reset the motor status of the
        # robot chassis to ensure that the chassis is in a safe state at the end of the
        # program
        sys.exit() # ensure that even if an exception occurs, the cleanup
        # operation can be performed and the program can be exited correctly.
```

## 2. Operation Steps

**Note: the input command should be case sensitive, and the keywords can be complemented by “Tab” key.**

1) Start JetAuto, then connect it to NoMachine.

2) Double click  to open command line terminal.

3) Input command “**sudo systemctl stop start\_app\_node.service**” and press Enter to stop app service.

4) Input command “**roslaunch jetauto\_controller jetauto\_controller.launch**” to enable the motion control service.

5) Open a new terminal, and input command “**rostopic pub /jetauto\_controller/cmd\_vel geometry\_msgs/Twist "linear:"**”. Then press “**TAB**” key to complement the parameters of the command.

linear refers to the linear velocity. With the robot as the first person view, positive X-axis direction is the front of the robot, and Y-axis direction is its left side. And there is no Z-axis direction.

angular represents the angular velocity. When Z value is positive number, JetAuto will rotate counterclockwise. When Z value is negative number, the robot will rotate clockwise. There are no X-axis and Y-axis direction

Press **←** and **→** key to move to the parameter needing to be modified.

For example, change X value of linear velocity as “**0.3**”.

**Note: the unit of the parameter is m/s. And it is recommended to set the value within -0.6 - 0.6m/s**

6) If you want to exit this program, please press “**Ctrl+C**”. If it cannot be closed, please try again.

7) Input command “**sudo systemctl start start\_app\_node.service**” and press Enter to start app service.

**Note: If you do not start the APP self-start service, it will affect the normal realization of the APP corresponding gameplay. If you do not enter the command to start the APP, restarting the robot will also restart the APP startup service.**

### 3. Function Realization

After the game starts, JetAuto will move forward at 0.3m/s.

### 4. Program Analysis

#### ◆ Subscribe to Motion Control Node

Subscribe to the topic published by **jetauto\_controller** node to acquire the related motion data.

```
rospy.Subscriber('cmd_vel', Twist, self.app_cmd_vel_callback)
rospy.Service('jetauto_controller/load_calibrate_param', Trigger,
self.load_calibrate_param)
```

#### ◆ Load the Parameters

Load the linear velocity and angular velocity parameters in the command.

```
self.go_factor = rospy.get_param('~go_factor', 1.00)
self.turn_factor = rospy.get_param('~turn_factor', 1.00)

linear_x = self.go_factor*msg.linear.x
linear_y = self.go_factor*msg.linear.y
angular_z = self.turn_factor*msg.angular.z
```

Its linear velocity ranges from -0.7 to 0.7 m/s. With the robot as the first person view, positive X-axis direction is the front of the robot, and Y-axis direction is its left side. And there is no Z-axis direction.

Its angular velocity ranges from -3.5 to 3.5 rad/s. When Z value is positive number, JetAuto will rotate counterclockwise. When Z value is negative number, the robot will rotate clockwise. There are no X-axis and Y-axis direction

#### ◆ Control Motor Rotation

Analyze and calculate the linear velocity and angular velocity to get the parameter required by the motor to rotate. Then control the motor to rotate with **set\_velocity** function to let the robot move.

```
self.mecanum.set_velocity(speed, direction, angular_z, speed_up=speed_up)
```

The meaning of the parameters are as follow.

The first parameter “**speed**” refers to the motor speed ranging from -100 mm/s to 100 mm/s.

The second parameter “**direction**” refers to the moving direction of JetAuto ranging from 0 to 360 degree. When it is set as 90°, it will move forward. When it is set as 270°, it will move backward. When it is set as 0°, it will move right. When it is set as 180°, it will move left.

The third parameter “**angular\_z**” is offset rate whose unit is rad/s and ranges from -2 to 2. When it is set as positive number, the car will move counterclockwise. When it is negative number, it will move clockwise.

The fourth parameter “**speed\_up**” is used to increase the time required for acceleration and slow down the process of increasing speed when the speed increases rapidly.

## 5. Function Expansion

By changing X and Y values of linear velocity, the robot can move in all direction. By modifying Z value of angular velocity, the robot can rotate 360 degree. For example, make the robot move front left. When you execute step 5 in **2. Operation Steps**, set X as “**0.2**” and Y as “**0.2**”.

Then press Enter, and the robot will move front left.

## Part 5 Speed Control ✓

## Part 6 Introduction to PID Algorithm

### 1. PID Algorithm Description

PID algorithm is the most widely used autonomous controller which controls a process according to the **ratio of errors**, **Proportional (P)**, **Integral (I)**, and **Derivative (D)**. With simple logic, it is easy to realize and can be widely applicable. And its control parameters are separate and it is easy to select the parameter.

$$u(t) = K_p \left( e(t) + \frac{1}{T_i} \int_0^t e(t) dt + T_d \frac{de(t)}{dt} \right)$$

Simplify:

$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt}$$

And:

$u(t)$ ——the output of the PID controller;

$K_p$ ——ratio factor;

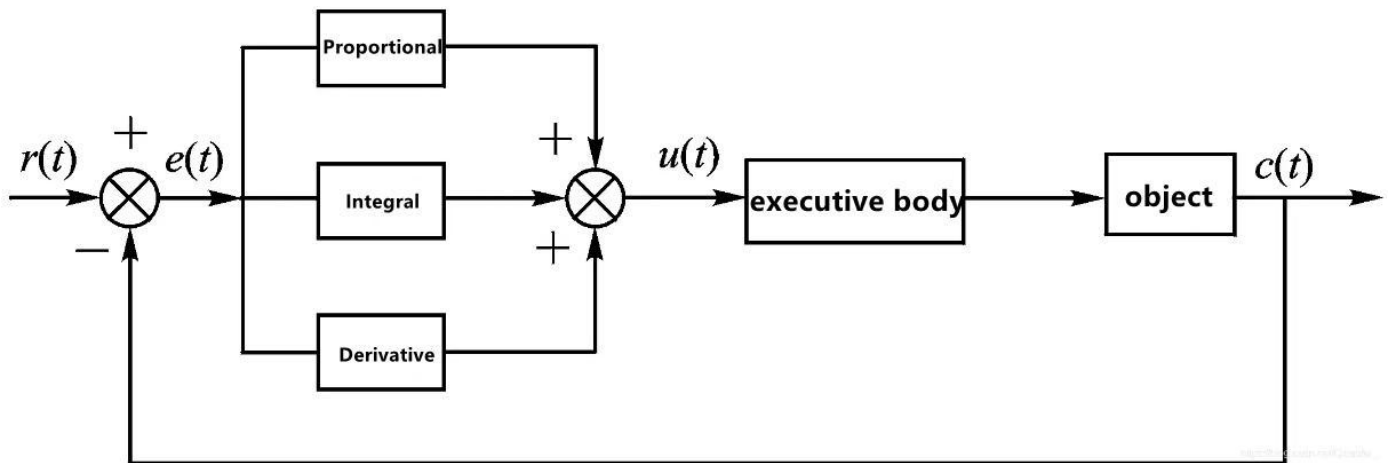
$K_i$ ——integral time constant;  $K_i = K T_{ip}$

$K_d$ ——derivative time constant;  $K_d = K_p \bullet T_d$

$e(t)$ —the difference between the given value and the measured value (error)

## 2. PID Algorithm Logic

In a closed-loop control system, **the output result of the object controlled will be returned to affect the output of the controller so as to generate one or multiple loop.** The closed-loop system has positive and negative feedback. If the feedback signal is opposite to the given signal, it is negative feedback. If they are the same, it is positive feedback.



### 2.1 Proportional

**Error is the difference between the given value (the set threshold  $r(t)$ ) and the measured output ( $c(t)$ ).** After the proportional is introduced, the output of the system is **proportional to the error**, which can **reduce the error quickly and stabilize the system**. When the system tends to stabilize, **steady-state error will still exist and cannot be eliminated**. The control effect of proportional depends on the value of  $K_p$ . The smaller the proportional coefficient  $K_p$ , the weaker the control and the slower the system response. On the contrary, the larger the proportional coefficient  $K_p$ , the stronger the control, and the faster the system response. However, **too large  $K_p$  will cause the system to generate large overshoot and oscillation**. The advantage of proportional control lies in its fast response, but the disadvantage is that there is a steady-state error.

**Take a bucket for example.** There is a hole at the bottom of the bucket, but you need to maintain the water in the bucket at 1m. And the current water level is 0.2m, and the water will leak 0.1m whenever you add the water. Here, proportional control can be adopted to control the water volume. The current water volume is 0.2m and the target is 1m. Set  $K_p=0.4$ . Then  $u=0.4 \cdot e$ , and  $e = \text{last water volume} - \text{current water volume}$

The first error:  $e(1)=1-0.2=0.8\text{m}$ . Therefore, the volume of the added water is 0.32m ( $K_p \cdot 0.8 = 0.4 \cdot 0.8 = 0.32\text{m}$ ), and the current water volume in the bucket is 0.42m ( $0.2+0.32-0.1=0.42\text{m}$ )

The second error:  $e(2) = 1-(0.42\text{m}) = 0.58\text{m}$ . Therefore, the volume of the added water is 0.192m ( $K_p \cdot 0.58=0.232\text{m}$ ), and the current water volume in the bucket is 0.552m ( $0.42+0.232-0.1=0.552$ )

| NO. | Current difference | Output water volumn | Final output water volumn | KP  |
|-----|--------------------|---------------------|---------------------------|-----|
| 1   |                    |                     | 0.2                       | 0.4 |
| 2   | 0.8                | 0.32                | 0.42                      |     |
| 3   | 0.58               | 0.232               | 0.552                     |     |
| 4   | 0.448              | 0.1792              | 0.6312                    |     |
| 5   | 0.3688             | 0.14752             | 0.67872                   |     |
| 6   | 0.32128            | 0.128512            | 0.707232                  |     |
| 7   | 0.292768           | 0.1171072           | 0.7243392                 |     |
| 8   | 0.2756608          | 0.11026432          | 0.73460352                |     |
| 9   | 0.26539648         | 0.10615859          | 0.74076211                |     |
| 10  | 0.259237888        | 0.10369516          | 0.74445727                |     |
| 11  | 0.255542733        | 0.10221709          | 0.74667436                |     |
| 12  | 0.25332564         | 0.10133026          | 0.74800462                |     |
| 13  | 0.251995384        | 0.10079815          | 0.74880277                |     |
| 14  | 0.25119723         | 0.10047889          | 0.74928166                |     |
| 15  | 0.250718338        | 0.10028734          | 0.749569                  |     |
| 16  | 0.250431003        | 0.1001724           | 0.7497414                 |     |
| 17  | 0.250258602        | 0.10010344          | 0.74984484                |     |
| 18  | 0.250155161        | 0.10006206          | 0.7499069                 |     |

Lastly, the water level stabilizes at 0.75m, and the error is 0.25m. When you add 0.1m water, the volume of the water added is equivalent to the volume of the water leaking. And this kind of error is called steady-state error.

**Steady-state error is defined as the difference between the output under the system tends to stabilize and the target output.**

## 2.2 Integral

**Integral aims at eliminating the steady-state error.** The integral of the error over time is continuously accumulated to keep the output changing so as to reduce the error. **If the integral time is sufficient, the steady-state error can be completely eliminated. The smaller  $T_i$  is, the stronger the function of integral is.** But too strong integral function will make the system overshoot and even oscillate. **The advantage of the integral is that the steady-state error can be completely eliminated, but the disadvantage is that the response speed is reduced and the overshoot will be too large.**

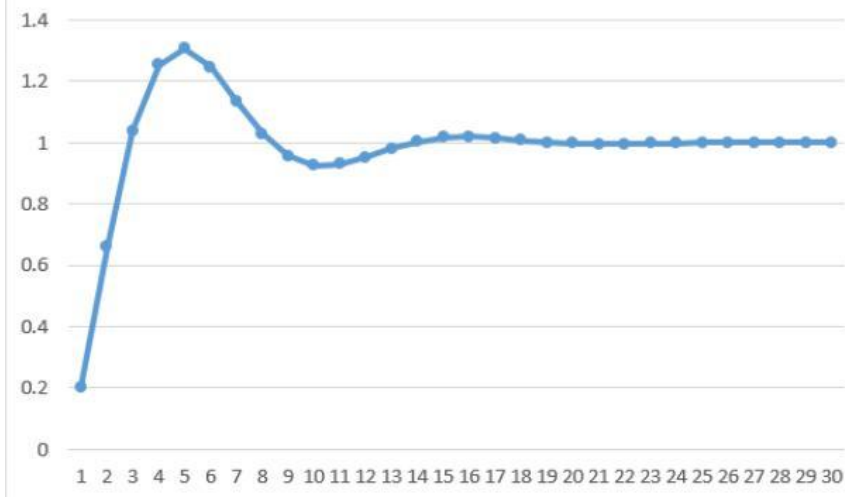
In the example given in “2.1 Proportional”, **the steady-state error still exists after proportional control is adopted.** To eliminate it completely, you can add integral control, and set the integral constant  $K_i$  as 0.3 ( $K_i=0.3$ ).

The first error:  $e(1)=1-0.2=0.8\text{m}$ . Therefore, the volume of the added water is:  $K_p \cdot 0.8 = 0.4 \cdot 0.8 = 0.32\text{m}$ ,  $K_i \cdot e(1) = 0.3 \cdot 0.8 = 0.24$ , and the final added water is:  $0.32\text{m} + 0.256 = 0.56$ . And the current water volume in the bucket is :  $0.2 + 0.276 - 0.1 = 0.66$

The second error:  $e(2)=1-0.66=0.34\text{m}$ . Therefore, the volume of the added water is:  $K_p \cdot 0.34 = 0.4 \cdot 0.34 = 0.136\text{m}$ ,  $K_i \cdot (e(2)+e(1)) = 0.3 \cdot 0.342 = 0.342$ , and the final added water is:  $0.136\text{m} + 0.342 = 0.478$ . And the current water volume in the bucket is :  $0.66 + 0.478 - 0.1 = 1.038$



| NO. | Current difference | KP output value | KI output value | PI output value | final water volumn | KP  | KI  |
|-----|--------------------|-----------------|-----------------|-----------------|--------------------|-----|-----|
| 1   |                    |                 |                 |                 | 0.2                |     |     |
| 2   | 0.8                | 0.32            | 0.24            | 0.56            | 0.66               | 0.4 | 0.3 |
| 3   | 0.34               | 0.136           | 0.342           | 0.478           | 1.038              |     |     |
| 4   | -0.038             | -0.0152         | 0.3306          | 0.3154          | 1.2534             |     |     |
| 5   | -0.2534            | -0.10136        | 0.25458         | 0.15322         | 1.30662            |     |     |
| 6   | -0.30662           | -0.122648       | 0.162594        | 0.039946        | 1.246566           |     |     |
| 7   | -0.246566          | -0.0986264      | 0.0886242       | -0.0100022      | 1.1365638          |     |     |
| 8   | -0.1365638         | -0.0546255      | 0.04765506      | -0.0069705      | 1.02959334         |     |     |
| 9   | -0.02959334        | -0.0118373      | 0.03877706      | 0.02693972      | 0.95653306         |     |     |
| 10  | 0.043466938        | 0.01738678      | 0.05181714      | 0.06920391      | 0.92573698         |     |     |
| 11  | 0.074263023        | 0.02970521      | 0.07409605      | 0.10380126      | 0.92953823         |     |     |
| 12  | 0.070461768        | 0.02818471      | 0.09523458      | 0.12341928      | 0.95295752         |     |     |
| 13  | 0.047042484        | 0.01881699      | 0.10934732      | 0.12816432      | 0.98112183         |     |     |
| 14  | 0.018878168        | 0.00755127      | 0.11501077      | 0.12256204      | 1.00368387         |     |     |
| 15  | -0.003683871       | -0.0014735      | 0.11390561      | 0.11243206      | 1.01611593         |     |     |
| 16  | -0.016115934       | -0.0064464      | 0.10907083      | 0.10262446      | 1.01874039         |     |     |
| 17  | -0.018740391       | -0.0074962      | 0.10344871      | 0.09595256      | 1.01469295         |     |     |
| 18  | -0.014692948       | -0.0058772      | 0.09904083      | 0.09316365      | 1.0078566          |     |     |



After the Integral control is adopted, the steady-state error is completely eliminated, but overshoot occurs.

## 2.3 Derivative

The slope error can be obtained by differentiating the error, and the slope reflects the change (rate of change) of the error signal, **so the error can be adjusted in advance in derivative part.** And it takes effect at the moment **when the error appears or changes, so as to avoid excessive overshoot and reduce the dynamic error and response time.** The larger the derivative coefficient  $T_d$ , the stronger the ability to suppress the error variation. The advantage of derivative is that it suppress overshoot and speed up the response. **The disadvantage is that the anti-interference ability is poor.** When the interference signal is strong, it is not recommended to add derivative.

In the above example, the integral effectively eliminates the steady-state error, but overshoot occurs. In order to reduce the overshoot, a derivative control is added.

### The first time:

$$K_p=0.4, K_I=0.3, K_D=0.3$$

$$e(1)=r(t)-c(1)=1-0.2=0.8m.$$

$$K_p \cdot e(1)=0.32$$

$$K_I \cdot e(1)=0.24$$

$$K_D \cdot [e(1)-e(1)]/\Delta t=0$$

$$\text{Added Water: } u(1)=K_p \cdot e(1)+K_I \cdot e(1)+K_D \cdot [e(1)-e(1)]/\Delta t=0.56$$

$$\text{Current Water: } c(1)+u(1)-0.1=0.66$$

### The second time:

$$e(2)=r(t)-c(2)=1-0.66=0.34m.$$

$$K_p \cdot e(2)=0.136$$

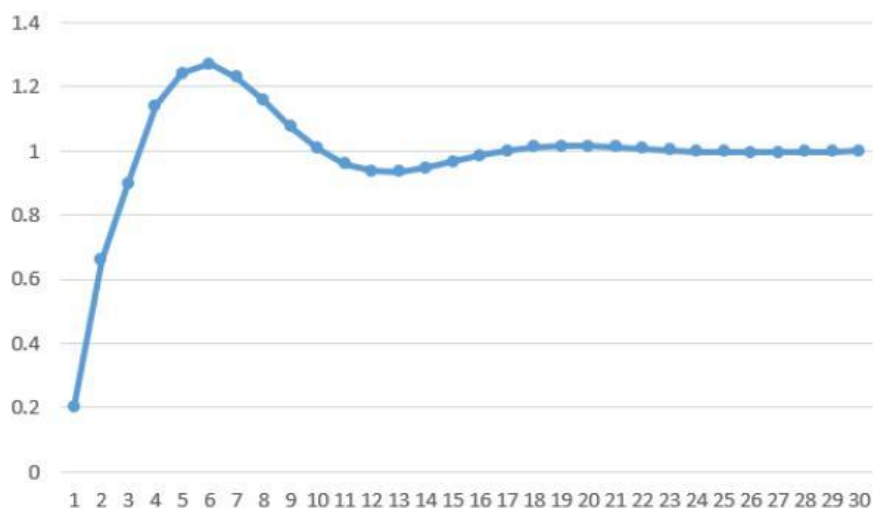
$$K_I \cdot [e(2)+e(1)]=0.342$$

$$K_D \cdot [e(2)-e(1)]/\Delta t=0.138$$

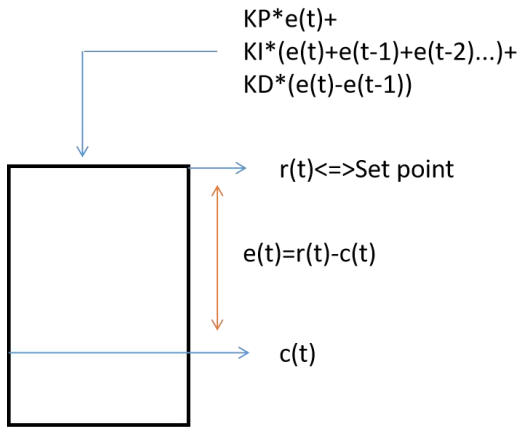
$$\text{Added Water: } u(2)=K_p \cdot e(2)+K_I \cdot [e(2)+e(1)]+K_D \cdot [e(2)-e(1)]/\Delta t=0.34$$

$$\text{Current Water: } c(2)+u(2)-0.1=0.9$$

| NO. | current difference | KP output value | KI output value | KD output  | PI output value | final water volumn | KP | KI  | KD  |
|-----|--------------------|-----------------|-----------------|------------|-----------------|--------------------|----|-----|-----|
| 1   |                    |                 |                 |            |                 | 0.2                |    | 0.4 | 0.3 |
| 2   | 0.8                | 0.32            | 0.24            | 0          | 0.56            | 0.66               |    |     |     |
| 3   | 0.34               | 0.136           | 0.342           | -0.138     | 0.34            | 0.9                |    |     |     |
| 4   | 0.1                | 0.04            | 0.372           | -0.072     | 0.34            | 1.14               |    |     |     |
| 5   | -0.14              | -0.056          | 0.33            | -0.072     | 0.202           | 1.242              |    |     |     |
| 6   | -0.242             | -0.0968         | 0.2574          | -0.0306    | 0.13            | 1.272              |    |     |     |
| 7   | -0.272             | -0.1088         | 0.1758          | -0.009     | 0.058           | 1.23               |    |     |     |
| 8   | -0.23              | -0.092          | 0.1068          | 0.0126     | 0.0274          | 1.1574             |    |     |     |
| 9   | -0.1574            | -0.06296        | 0.05958         | 0.02178    | 0.0184          | 1.0758             |    |     |     |
| 10  | -0.0758            | -0.03032        | 0.03684         | 0.02448    | 0.031           | 1.0068             |    |     |     |
| 11  | -0.0068            | -0.00272        | 0.0348          | 0.0207     | 0.05278         | 0.95958            |    |     |     |
| 12  | 0.04042            | 0.016168        | 0.046926        | 0.014166   | 0.07726         | 0.93684            |    |     |     |
| 13  | 0.06316            | 0.025264        | 0.065874        | 0.006822   | 0.09796         | 0.9348             |    |     |     |
| 14  | 0.0652             | 0.02608         | 0.085434        | 0.000612   | 0.112126        | 0.946926           |    |     |     |
| 15  | 0.053074           | 0.0212296       | 0.1013562       | -0.0036378 | 0.118948        | 0.965874           |    |     |     |
| 16  | 0.034126           | 0.0136504       | 0.111594        | -0.0056844 | 0.11956         | 0.985434           |    |     |     |
| 17  | 0.014566           | 0.0058264       | 0.1159638       | -0.005868  | 0.1159222       | 1.0013562          |    |     |     |
| 18  | -0.0013562         | -0.0005425      | 0.11555694      | -0.0047767 | 0.1102378       | 1.011594           |    |     |     |



Compared with proportional control, the overshoot is weakened.



数学公式过程:

$$e(t) = r(t) - c(t)$$

$$KP * e(t)$$

$$KI * (e(t) + e(t-1) + e(t-2) \dots)$$

$$KD * (e(t) - e(t-1)) / \Delta t$$

$$u(t) = KP * e(t) + KI * (e(t) + e(t-1) + e(t-2) \dots) + KD * (e(t) - e(t-1)) / \Delta t$$

$$c(t+1) = c(t) + u(t) - 0.1$$

数学公式过程to代码过程:

Step1:  $e(t) = r(t) - c(t)$  ✓

Step2:  $t$  ✓

Step3:  $\Delta t = t - (t-1)$  ✓

Step4:  $e(t) - e(t-1)$  ✓

Step5: #大于采样时间时进行以下自动控制模块  $\Delta t > \text{sample time}$

Step6:  $KP * e(t)$  ✓

Step7:  $KI * (e(t) + e(t-1) + e(t-2) \dots) \Rightarrow += e(t) \Delta t$  ✓

Step8: 控制积分超调量上限  $\Rightarrow \text{windup}$  ✓

#第一时刻无  $t-1, \Delta t = t - t = 0$

Step9.1:  $KD = 0$  ✓

#  $\Delta t > 0$ , 可微分

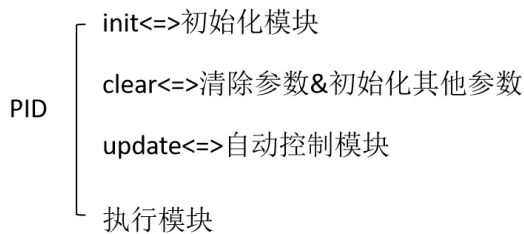
Step9.2:  $KD * (e(t) - e(t-1)) / \Delta t$  ✓

#为了下次循环, 记录  $t-1$  和  $e(t-1)$

Step10:  $t-1 \Leftarrow t$  ✓

Step11:  $e(t-1) \Leftarrow e(t)$  ✓

Step12:  $c(t+1) = KP * e(t) + KI * (e(t) + e(t-1) + e(t-2) \dots) + KD * (e(t) - e(t-1)) / \Delta t + c(t) - 0.1$  ✓



init<=>初始化模块

```

KP<=>self.Kp = P
KI<=>self.Ki = I
KD<=>self.Kd = D

t<=>self.current_time = time.time()

t-1<=>self.last_time = self.current_time

sample time=0

```

clear<=>清除参数&初始化其他参数

```

r(t)<=>self.SetPoint = 0.0

KP*e(t)<=>self.PTerm = 0.0

KI*(e(t)+e(t-1)+e(t-2)...)<=>self.ITerm = 0.0

KD*(e(t)-e(t-1))/Δt<=>self.DTerm = 0.0

e(t-1)<=>self.last_error = 0.0

windup<=>self.windup_guard = 20.0

c(t)<=>self.output = 0.0

```

update<=>自动控制模块

```

def update(self, feedback_value):
    error = self.SetPoint - feedback_value # Step1
    self.current_time = time.time() #Step2
    delta_time = self.current_time - self.last_time # Step3
    delta_error = error - self.last_error # Step4

    if (delta_time >= self.sample_time): # Step5
        self.PTerm = self.Kp * error #Step6
        self.ITerm += error * delta_time #Step7

        # Step8
        if (self.ITerm < -self.windup_guard):
            self.ITerm = -self.windup_guard
        elif (self.ITerm > self.windup_guard):
            self.ITerm = self.windup_guard

        # Step9.1
        self.DTerm = 0.0
        if delta_time > 0:
            # Step9.2
            self.DTerm = delta_error / delta_time

        self.last_time = self.current_time # Step10
        self.last_error = error # Step11

    # Step12
    self.output = self.PTerm + (self.Ki * self.ITerm) + (self.Kd * self.DTerm) + feedback_value -0.1

```

执行模块

```
if __name__ == '__main__':
    x_pid = PID(P=0.4, I=0.3, D=0.3)
    x_pid.SetPoint = 1
    x_pid.update(0.2)
    out = x_pid.output

    # Manually add a loop module that has achieved control
    while (out < x_pid.SetPoint):
        x_pid.update(out)
        out = x_pid.output

    print(out)
```

## CODE:

"""Ivmech PID Controller is simple implementation of a Proportional-Integral-Derivative (PID) Controller in the Python Programming Language.

More information about PID Controller:

[http://en.wikipedia.org/wiki/PID\\_controller](http://en.wikipedia.org/wiki/PID_controller)

"""

```
import sys
```

```
import time
```

```
# Positional PID Controller
```

```
class PID:
```

```
    """PID Controller
    """
```

```
    def __init__(self, P=0.0, I=0.0, D=0.0):
```

```
        self.Kp = P # Proportionality coefficients.
```

```
        self.Ki = I # Integration coefficients.
```

```
        self.Kd = D # Differential coefficients.
```

```
        self.sample_time = 0.00
```

```
        self.current_time = time.time()
```

```
        self.last_time = self.current_time
```

```
        #self.count = 0
```

```
        self.clear()
```

```
    def clear(self):
```

```
        """Clears PID computations and coefficients"""
```

```
        self.SetPoint = 0.0 # The desired value (setpoint) for the controlled
variable.
```



```

self.PTerm = 0.0
self.ITerm = 0.0
self.DTerm = 0.0
self.last_error = 0.0

# Windup Guard
self.int_error = 0.0
self.windup_guard = 20.0 # This limits the integral term to prevent
windup

self.output = 0.0 # The final output value calculated by the PID
controller.

def update(self, feedback_value):

    error = self.SetPoint - feedback_value # Step1

    self.current_time = time.time() #Step2
    delta_time = self.current_time - self.last_time # Step3
    delta_error = error - self.last_error # Step4

    if (delta_time >= self.sample_time): # Step5
        self.PTerm = self.Kp * error #Step6
        self.ITerm += error * delta_time #Step7
        #self.count +=1
        #print("NO",self.count,"ITerm:",self.ITerm)

        # Step8
        if (self.ITerm < -self.windup_guard):
            self.ITerm = -self.windup_guard
        elif (self.ITerm > self.windup_guard):
            self.ITerm = self.windup_guard

        # Step9.1
        self.DTerm = 0.0
        if delta_time > 0:
            # Step9.2
            self.DTerm = delta_error / delta_time

        # Remember last time and last error for next calculation
        self.last_time = self.current_time # Step10
        self.last_error = error # Step11

        # Step12
        self.output = self.PTerm + (self.Ki * self.ITerm) + (self.Kd *
self.DTerm) + feedback_value -0.1

def setKp(self, proportional_gain):

```

```

        """Determines how aggressively the PID reacts to the current error with
setting Proportional Gain"""
        self.Kp = proportional_gain

    def setKi(self, integral_gain):
        """Determines how aggressively the PID reacts to the current error with
setting Integral Gain"""
        self.Ki = integral_gain

    def setKd(self, derivative_gain):
        """Determines how aggressively the PID reacts to the current error with
setting Derivative Gain"""
        self.Kd = derivative_gain

    def setWindup(self, windup):
        """Integral windup, also known as integrator windup or reset windup,
refers to the situation in a PID feedback controller where
a large change in setpoint occurs (say a positive change)
and the integral terms accumulates a significant error
during the rise (windup), thus overshooting and continuing
to increase as this accumulated error is unwound
(offset by errors in the other direction).
The specific problem is the excess overshooting.
"""
        self.windup_guard = windup

    def setSampleTime(self, sample_time):
        """PID that should be updated at a regular interval.
Based on a pre-determined sampe time, the PID decides if it should
compute or return immediately.
"""
        self.sample_time = sample_time

if __name__ == '__main__':
    x_pid = PID(P=0.4, I=0.3, D=0.3)
    x_pid.SetPoint = 1
    x_pid.update(0.2)
    out = x_pid.output

    # Manually add a loop module that has achieved control
    while (out < x_pid.SetPoint):
        x_pid.update(out)
        out = x_pid.output

    print(out)

print(out)

```

# Part 7 IMU, Linear Velocity and Angular Velocity Calibration

Note:

1. Before the robot leaves the factory, relevant parameters are already set, so under normal circumstances, there is no need to perform calibration. Therefore, this section is only for understanding purposes. If you notice a significant deviation between the actual movement of the robot in the Rviz simulation software and the real robot's movement, you can make adjustments.

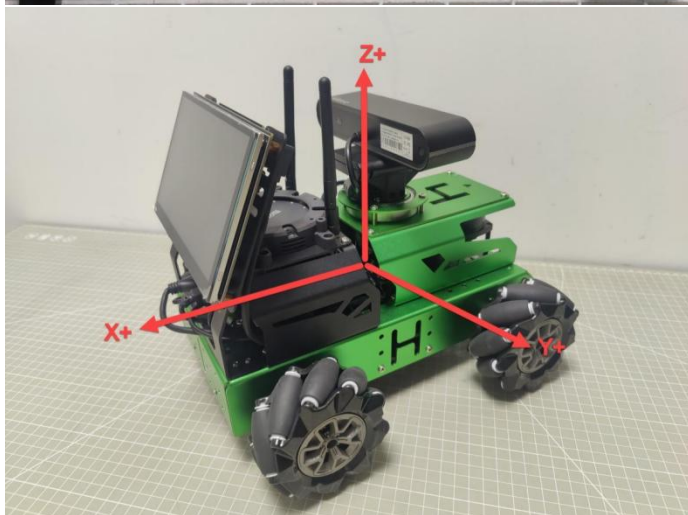
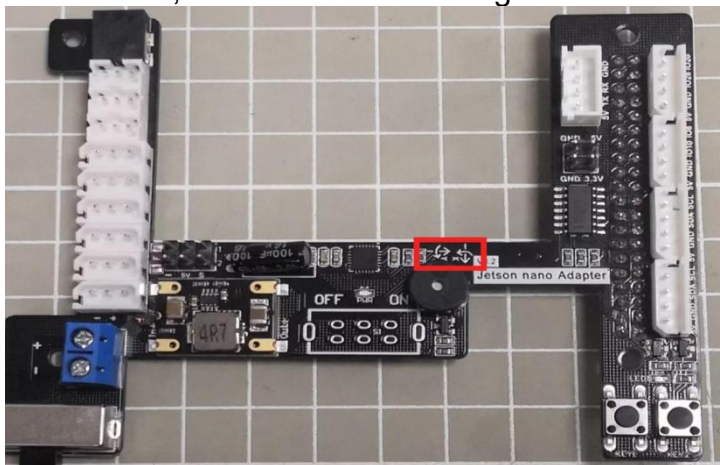
2. Calibration is only meant to reduce the deviation between the actual parameters of the hardware device and the ideal state parameters. There will always be some deviation in the actual hardware, so during calibration, you only need to adjust it to be relatively accurate according to your own requirements.

## 1. IMU Calibration

If the robot exhibits deviations during operation, it may require IMU calibration. Once the calibration process is completed, the robot can resume normal operation.

The **IMU (Inertial Measurement Unit)** is a device used to measure the three-axis attitude angle (angular rate) and acceleration of an object. It consists of the **gyroscope** and **accelerometer** as its main components, offering a total of **6 degrees of freedom to measure the angular velocity and acceleration of an object in three-dimensional space**.


Upon receiving the first IMU message, the node will prompt you to hold the IMU in a specific orientation and press Enter to record the measurement. It is important to note that the calibration orientation should align with the direction of the IMU on the expansion board. The XY axis diagram of the IMU on the expansion board, along with the XYZ coordinates when the robot is placed flat on the table, are illustrated in the figure below:



Among these axes, from the first perspective, the X+ direction represents the rear of the robot, while the X- direction corresponds to the front of the robot. The Y+ direction denotes the right side of the robot, while the Y- direction represents the left side. Although the expansion board does not indicate the direction of the Z axis, the Z+ direction is considered to be above the robot, while the Z- direction corresponds to below the robot.

Once you have completed the adjustment of parameters in all six directions, the node will save the calibration computation parameters into the designated YAML file. Please follow the detailed instructions provided below for further guidance.

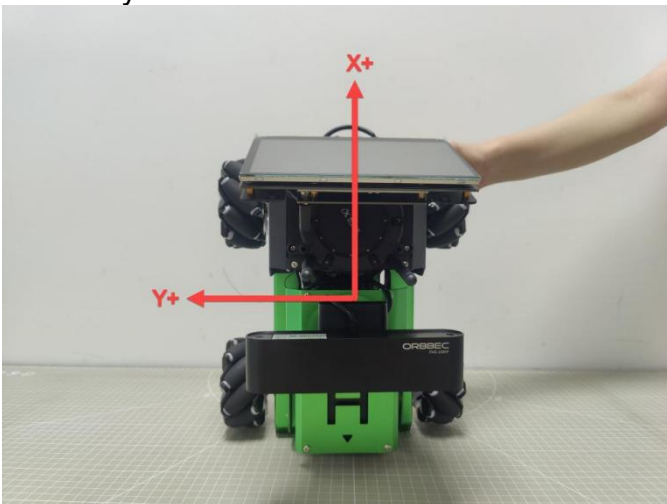
1) Start the robot, and access the robot system via NoMachine.

2) Click-on  to open the command-line terminal.

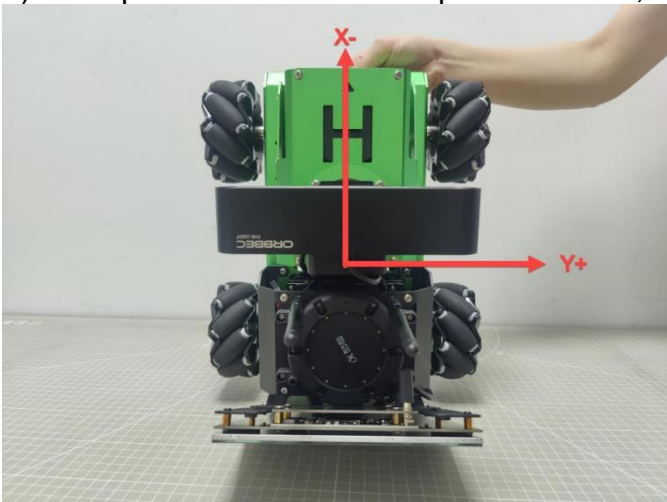
3) Run the command “**sudo systemctl stop start\_app\_node.service**” to disable the app auto-start service.

4) Execute the command “**roslaunch jetauto\_calibration calibrate\_imu.launch**” to start IMU calibration.

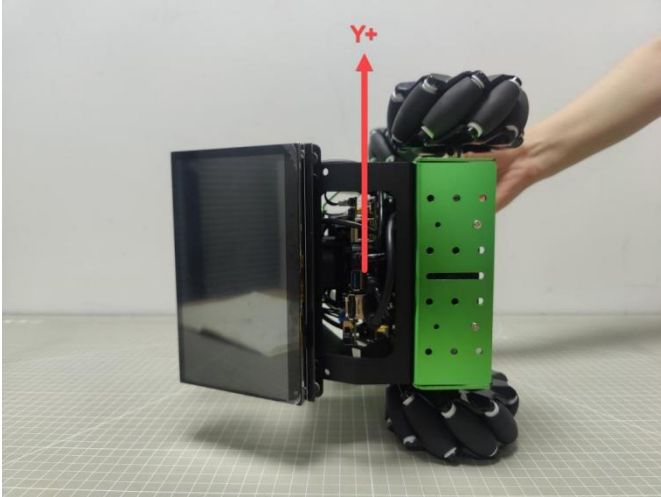
5) When you receive the following prompt, put the robot as the below picture shown, then press Enter key.



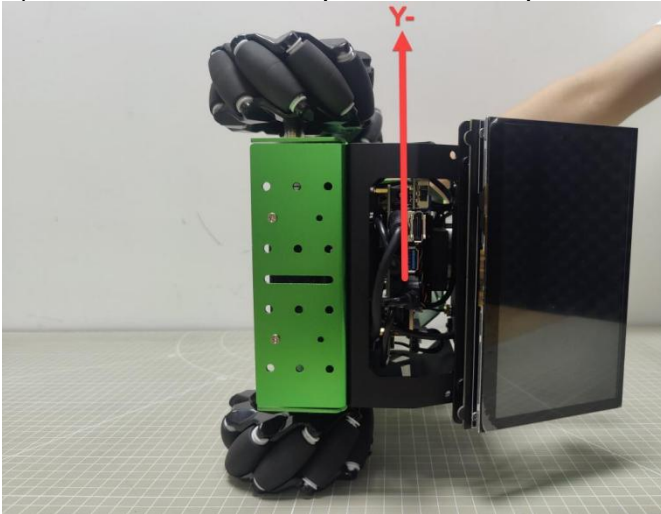
6) Then place the robot car as picture shown, and press Enter key.



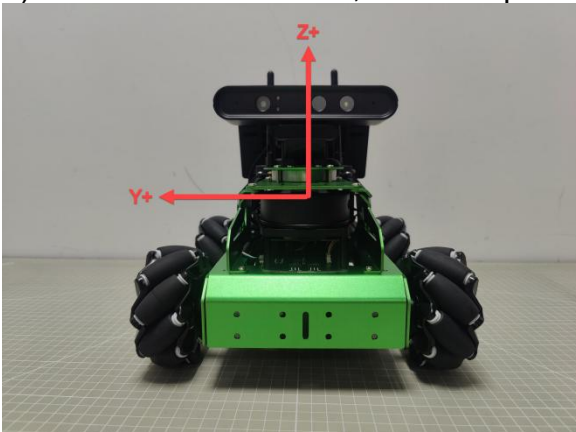
7) Place JetAuto as below, then press Enter.



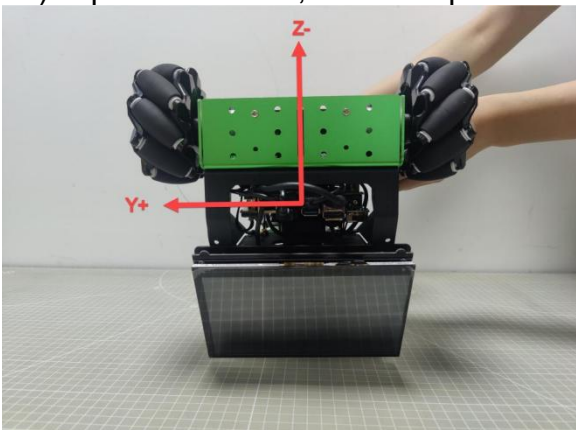
8) Position JetAuto as pictured, then press Enter key.



9) Place JetAuto like this, and then press Enter.



10) Flip over JetAuto, and then press Enter key.

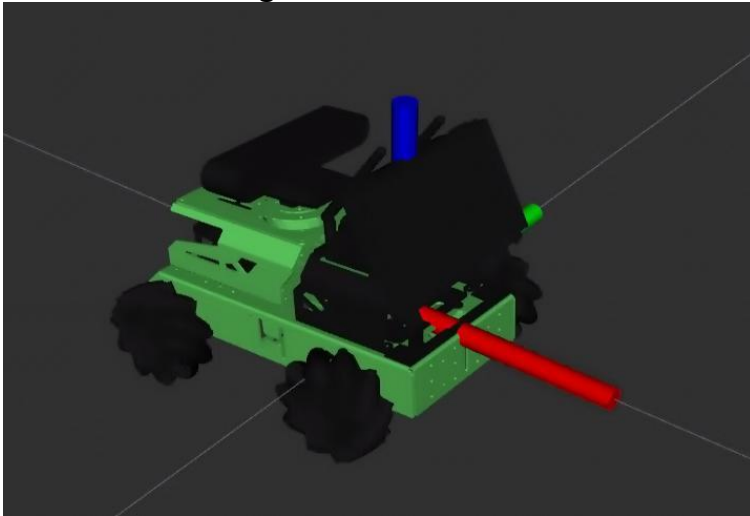




11) If the terminal prints the following message, it means the IMU calibration is completed.

```
Computing calibration parameters... Success!  
Saving calibration file... Success!
```

12) Once the calibration is done, enter this command "**roslaunch jetauto\_peripherals imu.launch debug:=true**" to check the model calibrated.




## 2. Angular Velocity Calibration

To calibrate the angular velocity, the robot needs to rotate in a full circle. Make sure to mark the robot's direction for easy observation. Follow these steps:

1) Place the robot on a flat surface and place a piece of tape or other marker in front of the robot's center.

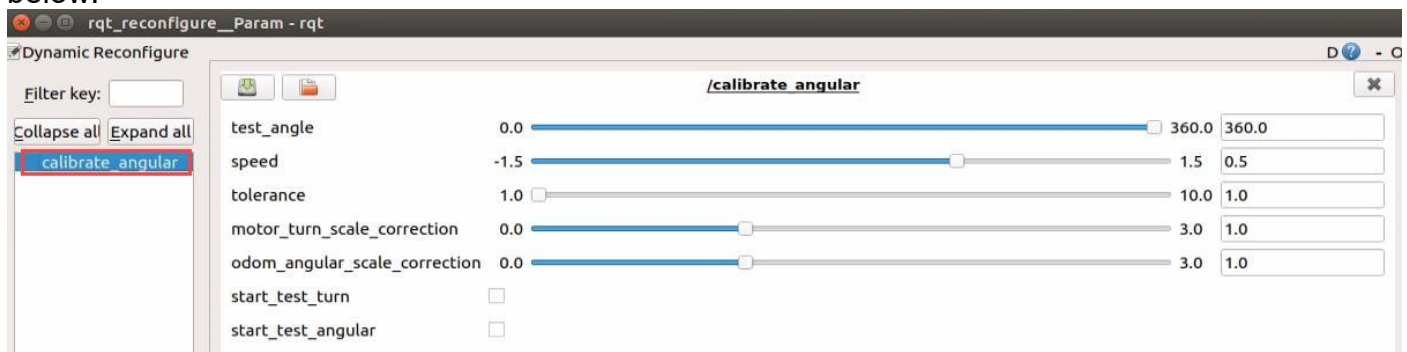
2) Start the robot and connect it to the remote control software NoMachine.

3) Click on  to open the command line terminal.

4) Enter the command "**sudo systemctl stop start\_app\_node.service**" and press Enter to stop the automatic startup service of the mobile app.

5) Enter the command "**roslaunch jetauto\_calibration calibrateAngular.launch turn:=true**" and press Enter to start adjusting the angular velocity using the "turn" parameter.

6) Click on "calibrateAngular" on the left side. The calibration interface will appear as shown below.



The parameters on the left side of the interface are defined as follows:

The first parameter, "test\_angle," represents the test rotation angle, with a default value of 360°.

The second parameter, "speed," represents the linear velocity with a default value of 0.15 meters per second.

The third parameter, "tolerance," represents the error value. A smaller error value results in more significant robot oscillations after reaching the target position.

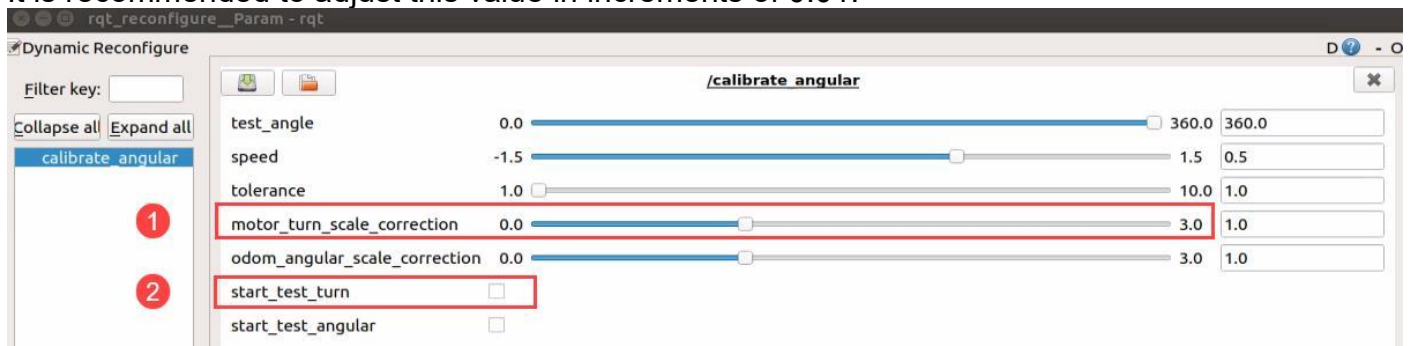
The fourth parameter, "motor\_turn\_scale\_correction," represents the motor rotation scale correction.

The fifth parameter, "odom\_angle\_scale\_correction," represents the odometry angle scale correction.

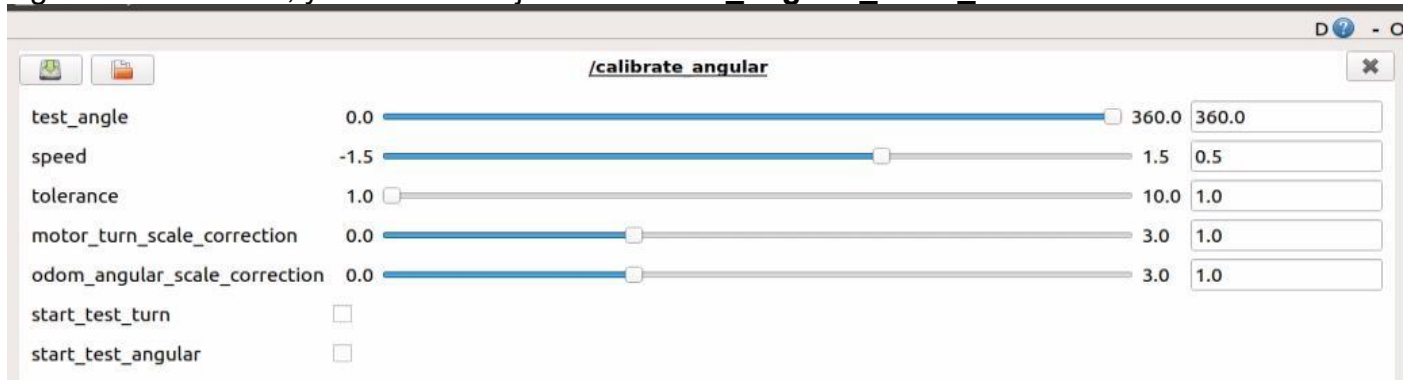
The sixth parameter, "start\_test\_turn," is the button to start testing the motor rotation scale correction.

The seventh parameter, "start\_testAngular," is the button to start testing the odometry angle scale correction.

Ensure the robot is properly aligned, with the marker placed in front of it. Check the "start\_test\_turn" option and the robot will rotate in place. If it fails to complete a full rotation, you need to adjust the "motor\_turn\_scale\_correction" value, which controls the motor's rotation scale. It is recommended to adjust this value in increments of 0.01.



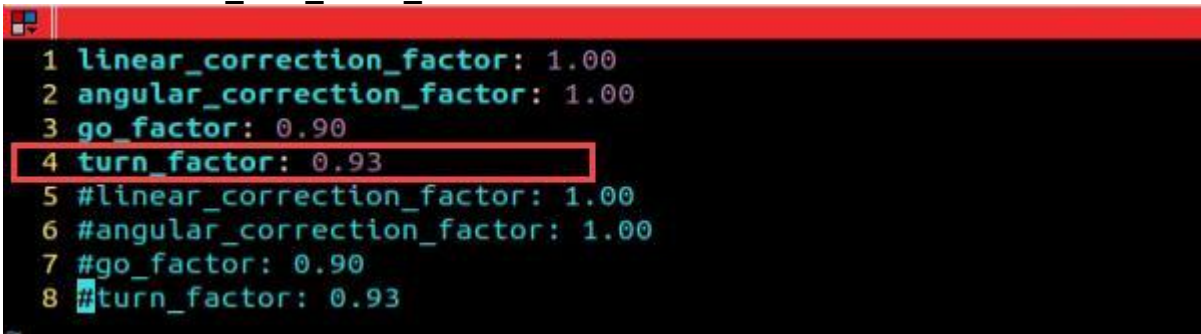
7) After the rotation, observe whether the marker is still in the center of the robot. If there is a significant deviation, you need to adjust the "odom\_angular\_scale\_correction" value.



8) Open a new command line terminal and enter the command: "**roscd jetauto\_controller/config**" to enter the calibration configuration file directory.

9) Enter the command "**vim calibrate\_params.yaml**" to open the configuration file.

10) Press the "I" key to enter edit mode and modify the value of **"turn\_factor"** to the adjusted value of **"motor\_turn\_scale\_correction."**

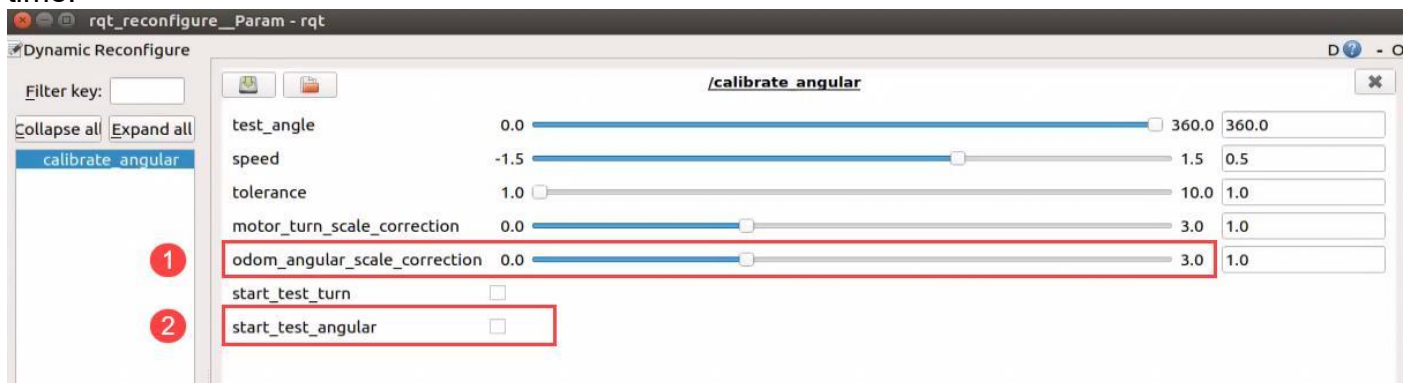
A terminal window with a black background and red title bar. It displays a list of parameters for a robot calibration. The parameters are: 1 linear\_correction\_factor: 1.00, 2 angular\_correction\_factor: 1.00, 3 go\_factor: 0.90, 4 turn\_factor: 0.93, 5 #linear\_correction\_factor: 1.00, 6 #angular\_correction\_factor: 1.00, 7 #go\_factor: 0.90, 8 #turn\_factor: 0.93. The line for 'turn\_factor: 0.93' is highlighted with a red rectangular box.

```
1 linear_correction_factor: 1.00
2 angular_correction_factor: 1.00
3 go_factor: 0.90
4 turn_factor: 0.93
5 #linear_correction_factor: 1.00
6 #angular_correction_factor: 1.00
7 #go_factor: 0.90
8 #turn_factor: 0.93
```

11) After the modification, press the "ESC" key, enter **":wq"** to exit and save the changes.

12) Enter the command **"roslaunch jetauto\_calibration calibrate\_angular.launch angular:=true"** and press Enter to start adjusting the angular velocity using the **"angular"** parameter.

Make sure the robot is properly aligned and the reference object is positioned directly in front and centered. Check the box next to **"start\_test\_angular"** to initiate the test. The robot will rotate in place by 360 degrees. After the rotation is complete, observe if the reference object remains in the center of the robot. If there is a significant deviation, you will need to adjust the value of **"odom\_angular\_scale\_correction,"** which is the scaling factor for the angle in the odometry. The normal value is 1. If the robot deviates to the left from the first-person perspective, decrease this value; if it deviates to the right, increase it. It is recommended to adjust the value by 0.01 each time.



13) Open a new command line terminal and enter the command **"roscd jetauto\_controller/config/"** to navigate to the calibration configuration file directory.

14) Enter the command **"vim calibrate\_params.yaml"** to open the configuration file.

15) Press the "I" key to enter the edit mode, and modify the value of **"angular\_correction\_factor"** to the adjusted value of **"odom\_angular\_scale\_correction."**


16) After making the modifications, press the "ESC" key, then enter **":wq"** to exit and save the changes.

17) If you need to close this procedure, you can press **"Ctrl+C"** in the terminal interface. If the closure fails, please try again.

### 3. Linear Velocity Calibration

1) Place JetAuto on a flat and open surface. Place a starting tape or any other starting marker in front of the robot, and place an endpoint tape or any other endpoint marker 1 meter in front of the robot.

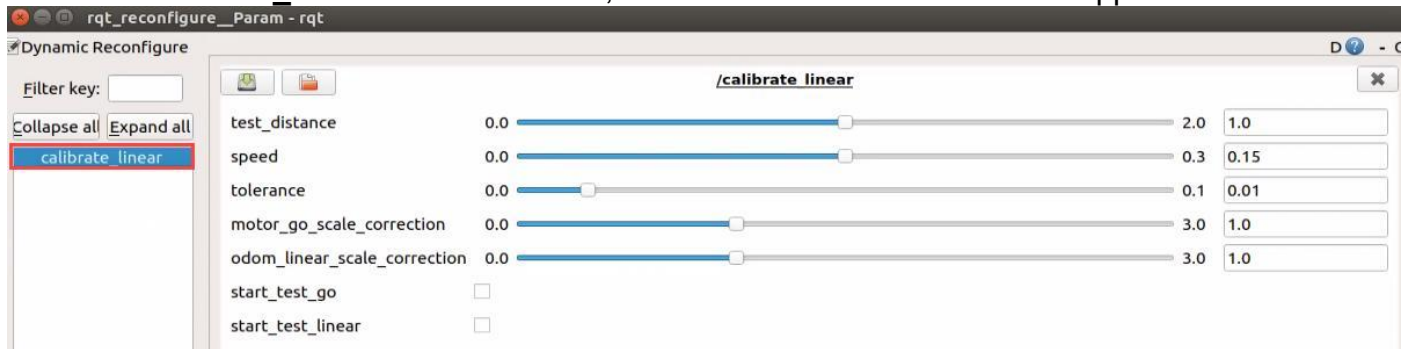
2) Start JetAuto and connect it to the remote control software NoMachine.

3) Double-click  to open the command line terminal.

4) Enter the command "**sudo systemctl stop start\_app\_node.service**" and press Enter to stop the automatic startup service of the mobile app.

5) Enter the command "**roslaunch jetauto\_calibration calibrate\_linear.launch go:=true**" and press Enter to activate the "go" parameter for adjusting the linear velocity.

Click on "**calibrate\_linear**" on the left side, and the calibration interface will appear as follows.



The parameters on the left side of the interface have the following meanings:

The first parameter "test\_distance" is the test distance, with a default value of 1 meter.

The second parameter "speed" is the linear velocity, with a default value of 0.15 meters per second.

The third parameter "tolerance" is the error value. The smaller the error value, the greater the robot's shaking amplitude after reaching the target position.

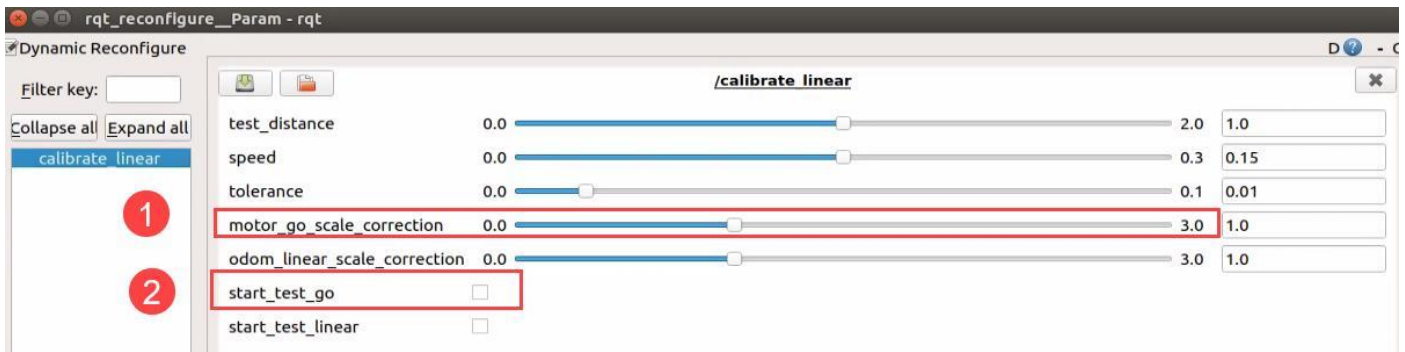
The fourth parameter "odom\_linear\_scale\_correction" is the odometer linear scaling factor.

The fifth parameter "motor\_go\_scale\_correction" is the motor forward scaling factor.

The sixth parameter "start\_test\_go" is the button to start testing the motor forward scaling factor.

The seventh parameter "start\_test\_linear" is the button to start testing the odometer linear scaling factor.

Ensure that the robot is aligned and positioned at the starting marker. Check the "start\_test\_go" option and the robot will move forward. Observe if the robot can travel in a straight line. If there is any deviation, you need to adjust the value of "motor\_go\_scale\_correction," which is the scaling factor for the motor's forward movement. It is recommended to adjust this value by 0.01 each time.



6) Open a new command line terminal and enter the command "**roscd jetauto\_controller/config/**" to navigate to the calibration configuration file directory.

7) Enter the command "**vim calibrate\_params.yaml**" to open the configuration file.

8) Press the "I" key to enter edit mode and modify the value of "go\_factor" to the adjusted value of "motor go scale correction."

```

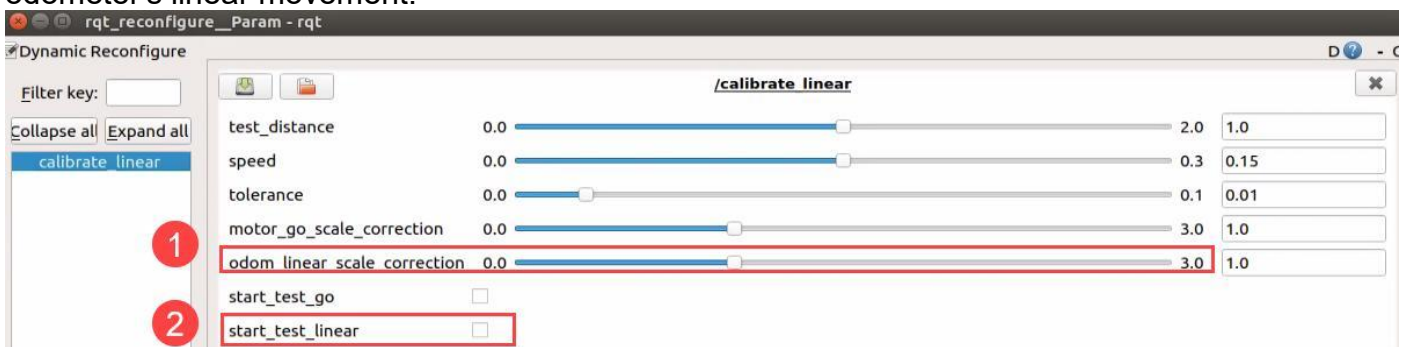
1 linear_correction_factor: 1.00
2 angular_correction_factor: 1.00
3 go_factor: 0.90
4 turn_factor: 0.93
5 #linear_correction_factor: 1.00
6 #angular_correction_factor: 1.00
7 #go_factor: 0.90
8 #turn_factor: 0.93

```

9) After making the modifications, press the "ESC" key, enter ":wq" to exit and save the changes.

10) Enter the command "**roslaunch jetauto\_calibration calibrate\_linear.launch linear:=true**" and press Enter to activate the "linear" parameter for adjusting the angular velocity.

11) Ensure that the robot is aligned and positioned at the starting marker. Check the "start\_test\_linear" option, and the robot will move a distance of 1 meter forward. After the movement, observe if the robot reaches the endpoint marker. If there is a significant deviation, you need to adjust the value of "odom\_linear\_scale\_correction," which is the scaling factor for the odometer's linear movement.



The normal value is 1. If the forward distance is less than 1 meter, decrease this value. If the forward distance is greater than 1 meter, increase this value. It is recommended to adjust this value by 0.01 each time.

12) Open a new command line terminal and enter the command "**roscd jetauto\_controller/config/**" to navigate to the calibration configuration file directory.

13) Enter the command "**vim calibrate\_params.yaml**" to open the configuration file.



14) Press the "I" key to enter edit mode and modify the value of "linear\_correction\_factor" to the adjusted value of "odom\_angular\_scale\_correction."

```
1 linear_correction_factor: 1.00
2 angular_correction_factor: 1.00
3 go_factor: 0.90
4 turn_factor: 0.93
5 #linear_correction_factor: 1.00
6 #angular_correction_factor: 1.00
7 #go_factor: 0.90
8 #turn_factor: 0.93
```

15) After making the modifications, press the "ESC" key, enter ":wq" to exit and save the changes.

16) To close this calibration process, press "Ctrl+C" in the terminal interface. If the closure fails, please try again.

After completing the calibration, you can enable the mobile app service by using a command or restarting the robot. If the mobile app service is not enabled, the related app functions will not work. (If the robot is restarted, the mobile app service will be automatically enabled.)

Enter the command "**sudo systemctl restart start\_app\_node.service**" to restart the mobile app service and wait for a beep sound to indicate that the service has started successfully.

## Part 8 Publish IMU and Odometer Data

IMU and odometer are crucial to robot, which can implement various functions. This Part discuss how to check data of IMU and odometer.

### 1. Publish IMU Data

#### 1.1 Enable Service

1) Start JetAuto, then connect it to NoMachine.

2) Double click  to open command line terminal.

3) Input command "**sudo systemctl stop start\_app\_node.service**" and press Enter to stop app autostart service.

4) Input command "**roslaunch jetauto\_peripherals imu.launch**" and press Enter to publish IMU data.

#### 1.2 View the Data

1) Open a new terminal, and input command "**rostopic list**" to check the current topic.

```
jetauto@jetauto-desktop:~$ rostopic list
/imu_corrected
/imu_data
/imu_raw
/rosout
/rosout_agg
/temperature
/tf
```

2) Input command “**rostopic info /imu\_data**” to check the type, publisher and subscriber of “/imu\_data” topic. And you also can change the topic.

```
jetauto@jetauto-desktop:~$ rostopic info /imu_data
Type: sensor_msgs/Imu

Publishers:
 * /imu_filter (http://192.168.149.1:45205/)

Subscribers: None
```

The type of this topic is “**sensor\_msgs/Imu**”, and its publisher is “/imu\_filter”, but it has no subscriber.

3) Input command “**rostopic echo /imu\_data**” to print the topic content, and you can change it to other topics.

```
orientation:
  x: -0.708871442233
  y: 0.0422049036073
  z: 0.151996089694
  w: -0.687471609026
orientation_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
angular_velocity:
  x: 0.00055786259542
  y: -0.000427694656489
  z: -0.000737175572519
angular_velocity_covariance: [1000000.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 1e-06]
linear_acceleration:
  x: -0.317466400531
  y: 2.31903699111
  z: -0.0249514822816
```

The data of three axis of IMU is displayed on the terminal.

## 2. Publish Odometer Data

### 2.1 Enable Service

1) Start JetAuto, then connect it to NoMachine.

2) Double click  to open command line terminal.

3) Input command “**sudo systemctl stop start\_app\_node.service**” and press Enter to stop app autostart service.

4) Input command “**roslaunch jetauto\_controller odom\_publish.launch**” and press Enter to publish odometer data.

### 2.2 View the Data

1) Open a new terminal, and input command “**rostopic list**” to check the current topic.

```
jetauto@jetauto-desktop:~$ rostopic list
/cmd_vel
/diagnostics
/imu
/imu_corrected
/imu_raw
/jetauto_controller/cmd_vel
/odom
/odom_raw
/rosout
/rosout_agg
/set_odom
/set_pose
/temperature
/tf
/tf_static
```

2) Input command “**rostopic info /odom\_raw**” to check the type, publisher and subscriber of “**/odom\_raw**”. And you can change the topic to be checked.

```
jetauto@jetauto-desktop:~$ rostopic info /odom_raw
Type: nav_msgs/Odometry

Publishers:
 * /jetauto_odom_publisher (http://192.168.149.1:34685/)

Subscribers:
 * /ekf_localization (http://192.168.149.1:38811/)
```

The type of this topic is “**nav\_msgs/Odometry**”, the publisher is “**/jetauto\_odom\_publisher**” and the subscriber is “**/ekf\_localization**”.

3) Input command “**rostopic echo /odom\_raw**” to print the topic content. And the topic can be changed.

```
pose:
  pose:
    position:
      x: 0.0
      y: 0.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
  covariance: [1e-09, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.001, 1e-09, 0.0, 0.0, 0.0,
0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000.0]
twist:
  twist:
    linear:
      x: 0.0
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
```

The pose and twist data acquired will be displayed on the terminal.

# Part 9 Handle Control

## 1. Preparation


- 1) Insert the handle receiver into any USB interface on JetAuto.
- 2) Please bring your own two AAA dry batteries. And insert them into the battery slot.

## 2. Device Connection

- 1) Start the robot. After robot starts successfully, turn on the handle switch. The two LEDs (red and green) on the handle will flash at the same time.
- 2) Please wait for a while. Then the robot will pair with the handle automatically. After successful pairing, the green light will keep lighting up.

**Note:** If the handle doesn't connect to the robot within 30s or there is no operation on the handle within 5 minutes after turning on, it will enter sleep mode. And you can press "START" to activate the handle.

## 3. Operation Steps

- 1) Start JetAuto, then connect it to NoMachine.
- 2) Double click  to open command line terminal.
- 3) Enter command "**systemctl stop start\_app\_node.service**" to stop auto-start service.
- 4) Input command "**roslaunch jetauto\_controller jetauto\_controller.launch**" to enable motion control service.
- 5) Input command "**roslaunch jetauto\_peripherals joystick\_control.launch**" to enable handle control service.
- 6) If you want to exit this program, press "**Ctrl+C**". If it cannot exit, please press the key again.

## 4. Program Outcome

After the game starts, you can press the handle, and the status of the buttons on the handle is printed on the terminal. And the functions of the buttons on the handle are listed below.

**Note:** you can slightly push the joystick to make JetAuto move at the low speed.

| Button/ Joystick | Operation                | Function                                                                                                                                      |
|------------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| START            | Press                    | Exit the sleep mode (If you are using JetAuto Pro, pressing "START" button enables JetAuto Pro's robot arm to return to the initial posture.) |
| Left joystick    | Push it in all direction | Control the robot to move in the corresponding direction                                                                                      |
| Right joystick   | Push it to left          | Control the robot to turn left                                                                                                                |
|                  | Push it to right         | Control the robot to turn right                                                                                                               |

## 5. Program Analysis

Controlling JetAuto with handle is to **connect handle control node with underlying driver node**. Then **the program will send the button message to make JetAuto perform different actions**.

The source code of this program is stored in  
`/home/jetauto_ws/src/jetauto_peripherals/scripts/joystick_control.py`

```
#!/usr/bin/env python3
# encoding: utf-8
import sys
import time
import math
import rospy
from enum import Enum
from std_srvs.srv import Trigger
import jetauto_sdk.misc as misc
import jetauto_sdk.buzzer as buzzer
import geometry_msgs.msg as geo_msg
import sensor_msgs.msg as sensor_msg
from hiwonder_servo_msgs.msg import CommandDuration, JointState
sys.path.append('/home/jetauto/jetauto_software/jetauto_arm_pc')
import action_group_controller as controller

AXES_MAP = 'lx', 'ly', 'rx', 'ry', 'r2', 'l2', 'hat_x', 'hat_y'
# Saves the name of each axis of the joystick.

BUTTON_MAP = 'cross', 'circle', '', 'square', 'triangle', '', 'l1', 'r1', 'l2',
'r2', 'select', 'start', '', 'l3', 'r3', '', 'hat_xl', 'hat_xr', 'hat_yu',
'hat_yd', ''
# Saves the names of the individual buttons on the joystick.

# Enumerated class: defines several possible values for the button state.
class ButtonState(Enum):
    Normal = 0
    Pressed = 1
    Holding = 2
    Released = 3

class JoystickController:
    def __init__(self):
        rospy.init_node('joystick_control', anonymous=True) # initializes the
        ROS node, which is named 'joystick_control'.
        self.max_linear = rospy.get_param('~max_linear', 0.4)
        self.max_angular = rospy.get_param('~max_angular', 2.0)
        # Get the values of the parameters keyed '~max_linear' and
        '~max_angular' from the ROS Parameter Server, respectively, or use the default
        values of 0.4 and 2.0, respectively, if these parameters are not set.
        # These parameters are used to control the maximum linear velocity and
        maximum angular velocity of the robot.

        self.machine = rospy.get_param('~machine', 'JetAuto')
```

```

        self.disable_servo_control = rospy.get_param('~disable_servo_control',
'true')
        # These parameters are used to determine the type of robot and whether
to disable servo control.

        cmd_vel = rospy.get_param('~cmd_vel', 'jetauto_controller/cmd_vel')
        # This parameter specifies the ROS topic used to publish robot motion
control commands

        self.joy_sub = rospy.Subscriber('joy', sensor_msg.Joy,
self.joy_callback)
        # A ROS subscriber is created, subscribing to a topic named 'joy' with
a message type of sensor_msg.Joy and a callback function of self.joy_callback.
        # This subscriber is used to receive input messages from the joystick.

        self.mecanum_pub = rospy.Publisher(cmd_vel, geo_msg.Twist, queue_size=1)
        # A ROS publisher is created, posting to the specified topic cmd_vel
with message type geo_msg.Twist.
        # This publisher is used to publish motion control commands for the
robot.

        self.last_axes = dict(zip(AXES_MAP, [0.0,] * len(AXES_MAP)))
        self.last_buttons = dict(zip(BUTTON_MAP, [0.0,] * len(BUTTON_MAP)))
        # These two lines of code create two separate dictionaries to store the
axis values and button states of the last received joystick.

        self.mode = 0
        # Initializes a variable mode to indicate the current mode, with a
default value of 0.

        self.joint1 = rospy.Publisher('joint1_controller/command_duration',
CommandDuration, queue_size=1)
        self.joint2 = rospy.Publisher('joint2_controller/command_duration',
CommandDuration, queue_size=1)
        self.joint3 = rospy.Publisher('joint3_controller/command_duration',
CommandDuration, queue_size=1)
        self.joint4 = rospy.Publisher('joint4_controller/command_duration',
CommandDuration, queue_size=1)
        self.jointr = rospy.Publisher('r_joint_controller/command_duration',
CommandDuration, queue_size=1)
        # The ROS publisher for publishing joint control commands is created.
        # Each joint has a corresponding publisher for publishing position
control commands for the joints

        self.joint1_state = 0
        self.joint2_state = 0
        self.joint3_state = 0
        self.joint4_state = 0

```



```

        self.joint5_state = 0
        self.jointr_state = 0
        # These statements initialize variables that store the current state of
each joint.
        # These variables are updated when a joint status message is received.

        rospy.Subscriber('joint1_controller/state', JointState, lambda msg:
setattr(self, 'joint1_state', msg.goal_pos))
        rospy.Subscriber('joint2_controller/state', JointState, lambda msg:
setattr(self, 'joint2_state', msg.goal_pos))
        rospy.Subscriber('joint3_controller/state', JointState, lambda msg:
setattr(self, 'joint3_state', msg.goal_pos))
        rospy.Subscriber('joint4_controller/state', JointState, lambda msg:
setattr(self, 'joint4_state', msg.goal_pos))
        rospy.Subscriber('jointr_controller/state', JointState, lambda msg:
setattr(self, 'jointr_state', msg.goal_pos))
        # Each subscriber stores the goal position (goal_pos ) in the received
joint state message into the corresponding self.jointX_state variable.

def axes_callback(self, axes):
    twist = geo_msg.Twist()
    # An empty Twist message object is created to represent the robot's
motion control commands
    if self.machine != 'JetMega':
        twist.linear.y = misc.val_map(axes['lx'], -1, 1, -self.max_linear,
self.max_linear)
        # Controls the motion of the robot in the y-axis direction by
mapping the horizontal axis value of the joystick to the linear velocity of the
robot in the y-axis direction and assigning that velocity value to the linear.y
property of the Twist message object.
        # The val_map function is typically used to map an input value in a
given range to an output value in another given range.
        """ Parameters include:
            axes['lx']: the value of the left horizontal axis of the joystick.
This value is typically in the range [-1, 1] and indicates the position of the
joystick on the horizontal axis.
            -1 and 1: Specifies the input range.
            -self.max_linear and self.max_linear: specifies the output range.
self.max_linear is the maximum linear speed of the robot set in the constructor.
        """
    else:
        twist.linear.y = 0
        twist.linear.x = misc.val_map(axes['ly'], -1, 1, -self.max_linear,
self.max_linear)

```

```

    # Based on the value axes['ly'] of the vertical axis on the left side
    of the joystick (usually the left joystick) , map it to another line speed
    control command for the robot twist.linear.x.
    twist.angular.z = misc.val_map(axes['rx'], -1, 1, -self.max_angular,
    self.max_angular)
    # Map the value axes['rx'] to the robot's angular velocity control
    command twist.angular.z, based on the value axes['rx' ] of the horizontal axis
    on the right side of the joystick (usually the right joystick)
    self.mecanum_pub.publish(twist)
    # Publish the motion control commands from the above calculations to
    the ROS topic self.mecanum_pub so that other nodes can receive and control the
    motion of the robot

    def select_callback(self, new_state): # Event for handling presses of the
    joystick's Select button.
        if new_state == ButtonState.Pressed: # Whether the button is pressed
            self.mode = 1 if self.mode == 0 else 0 # Switch the mode according
            to the current mode self.mode
            rospy.loginfo("SELECT="+ str(self.mode)) # Records the current mode
            information
            if self.mode == 0:
                buzzer.on()
                time.sleep(0.1)
                buzzer.off()
            elif self.mode == 1:
                for i in range(2):
                    buzzer.on()
                    time.sleep(0.1)
                    buzzer.off()
                    time.sleep(0.1)
            else:
                pass

    def l1_callback(self, new_state): # Event for handling the joystick's L1
    button
        if self.mode == 0 and not self.disable_servo_control: # Check that the
        current mode is 0 and that servo control is not disabled.
            p = self.joint4_state - math.radians(2) # calculates the target
            position of servo 4. It subtracts 2 radians from the current state of servo 4,
            self.joint4_state.
            self.joint4.publish(CommandDuration(data=p, duration=50)) # Posted
            a CommandDuration message to the ROS topic named
            'joint4_controller/command_duration' to control the position of servo 4.
            """ The CommandDuration message contains a target position data and
            a duration.
            data=p passes the calculated target position to the message.

```

```
        duration=50 specifies a duration of 50 milliseconds for the control
command. """
```

```
        self.joint4_state = p # Updated the state of servo 4
self.joint4_state, setting it to the newly calculated target position p.
```

```
    else:
```

```
        pass # No operation
```

```
    def l2_callback(self, new_state): # Event for handling the joystick's L2
button
```

```
        pass # No operation
```

```
    def r1_callback(self, new_state): # Event for handling the joystick's R1
button
```

```
        if self.mode == 0 and not self.disable_servo_control:
```

```
            p = self.joint4_state + math.radians(2) # calculates the target
position of servo 4. It adds 2 radians to the current state of servo 4,
self.joint4_state.
```

```
            self.joint4.publish(CommandDuration(data=p, duration=50)) # Posted
a CommandDuration message to the ROS topic named
'joint4_controller/command_duration' to control the position of servo 4.
```

```
            self.joint4_state = p # Updated the state of servo 4
```

```
self.joint4_state, setting it to the newly calculated target position p.
```

```
        else:
```

```
            pass # No operation
```

```
    def r2_callback(self, new_state): # Event for handling the joystick's L2
button
```

```
        pass # No operation
```

```
    def square_callback(self, new_state): # Event for handling the joystick's
Square button
```

```
        if self.mode == 0 and not self.disable_servo_control:
```

```
            p = self.jointr_state - math.radians(2) # calculates the target
position of servo r. It subtracts 2 radians from the current state of servo r,
self.jointr_state.
```

```
            self.jointr.publish(CommandDuration(data=p, duration=50)) # Posted
a CommandDuration message to the ROS topic named
'r_joint_controller/command_duration' to control the position of servo r.
```

```
            self.jointr_state = p # Updated the state of servo r
```

```
self.jointr_state, setting it to the newly calculated target position p.
```

```
        else:
```

```
            res = rospy.ServiceProxy('/set_row', Trigger)() # A ROS service
agent is created to invoke a service named /set_row.
```

```
            # Trigger is a ROS service message type that triggers an invocation
of the service
```

```
            if res.success:
```

```

        print('set row success')
    else:
        print('set row failed')

    def cross_callback(self, new_state): # Event for handling the joystick's
Cross button
        if self.mode == 0 and not self.disable_servo_control:
            p = self.joint3_state - math.radians(2) # calculates the target
position of servo 3. It subtracts 2 radians from the current state of servo 3,
self.joint3_state.
            self.joint3.publish(CommandDuration(data=p, duration=50)) # Posted
a CommandDuration message to the ROS topic named
'joint3_controller/command_duration' to control the position of servo 3.
            self.joint3_state = p # Updated the state of servo 3
self.joint3_state, setting it to the newly calculated target position p.
        else:
            pass # No operation

    def circle_callback(self, new_state): # Event for handling the joystick's
Circle button
        if self.mode == 0 and not self.disable_servo_control:
            p = self.jointr_state + math.radians(2) # calculates the target
position of servo r. It adds 2 radians to the current state of servo r,
self.jointr_state.
            self.jointr.publish(CommandDuration(data=p, duration=50)) # Posted
a CommandDuration message to the ROS topic named
'r_joint_controller/command_duration' to control the position of servo r.
            self.jointr_state = p # Updated the state of servo r
self.jointr_state, setting it to the newly calculated target position p.
        else:
            res = rospy.ServiceProxy('/set_column', Trigger)() # A ROS service
agent is created to invoke a service named /set_column.
            if res.success:
                print('set column success')
            else:
                print('set column failed')

    def triangle_callback(self, new_state): # Event for handling the joystick's
Triangle button
        if self.mode == 0 and not self.disable_servo_control:
            p = self.joint3_state + math.radians(2) # calculates the target
position of servo 3. It adds 2 radians to the current state of servo 3,
self.joint3_state.
            self.joint3.publish(CommandDuration(data=p, duration=50)) # Posted
a CommandDuration message to the ROS topic named
'joint3_controller/command_duration' to control the position of servo 3.

```

```

        self.joint3_state = p # Updated the state of servo 3
self.joint3_state, setting it to the newly calculated target position p.
    else:
        res = rospy.ServiceProxy('/set_triangle', Trigger)() # A ROS
service agent is created to invoke a service named /set_triangle.
        if res.success:
            print('set triangle success')
        else:
            print('set triangle failed')

    def start_callback(self, new_state): # Event for handling the joystick's
Start button
        if new_state == ButtonState.Pressed: # Whether the button is pressed
            buzzer.on()
            time.sleep(0.1)
            buzzer.off()
            if self.mode == 0:
                self.joint1.publish(CommandDuration(data=0, duration=2000)) #
Posted a CommandDuration message to a ROS topic named
'joint1_controller/command_duration' to control the position of servo joint1.
                self.joint2.publish(CommandDuration(data=math.radians(-40),
duration=2000)) # Posted a CommandDuration message to a ROS topic named
'joint2_controller/command_duration' to control the position of servo joint2.
                self.joint3.publish(CommandDuration(data=math.radians(110),
duration=2000)) # Posted a CommandDuration message to a ROS topic named
'joint3_controller/command_duration' to control the position of servo joint3.
                self.joint4.publish(CommandDuration(data=math.radians(70),
duration=2000)) # Posted a CommandDuration message to a ROS topic named
'joint4_controller/command_duration' to control the position of servo joint4.
                self.jointr.publish(CommandDuration(data=0, duration=2000)) #
Posted a CommandDuration message to a ROS topic named
'jointr_controller/command_duration' to control the position of the servo
jointr.

    def hat_xl_callback(self, new_state): # Event that handles the left stick
of the joystick moving to the left.
        if self.mode == 0 and not self.disable_servo_control:
            p = self.joint1_state - math.radians(2) # calculates the target
position of servo 1. It subtracts 2 radians from the current state of servo 1,
self.joint1_state.
            self.joint1.publish(CommandDuration(data=p, duration=50)) # Posted
a CommandDuration message to the ROS topic named
'joint1_controller/command_duration' to control the position of servo 1.
            self.joint1_state = p # Updated the state of servo 1
self.joint1_state, setting it to the newly calculated target position p.
        else:
            pass # No operation

```

```

def hat_xr_callback(self, new_state): # Event that handles the right stick
of the joystick moving to the right.
    if self.mode == 0 and not self.disable_servo_control:
        p = self.joint1_state + math.radians(2) # calculates the target
position of servo 1. It adds 2 radians to the current state of servo 1,
self.joint1_state.
        self.joint1.publish(CommandDuration(data=p, duration=50)) # Posted
a CommandDuration message to the ROS topic named
'joint1_controller/command_duration' to control the position of servo 1.
        self.joint1_state = p # Updated the state of servo 1
self.joint1_state, setting it to the newly calculated target position p.
    else:
        pass # No operation

def hat_yd_callback(self, new_state): # Event that handles the downward
movement of the joystick's left stick.
    if self.mode == 0 and not self.disable_servo_control:
        p = self.joint2_state - math.radians(2) # calculates the target
position of servo 2. It subtracts 2 radians from the current state of servo 2,
self.joint2_state.
        self.joint2.publish(CommandDuration(data=p, duration=50)) # Posted
a CommandDuration message to the ROS topic named
'joint2_controller/command_duration' to control the position of servo 2.
        self.joint2_state = p # Updated the state of servo 2
self.joint2_state, setting it to the newly calculated target position p.
    else:
        pass # No operation

def hat_yu_callback(self, new_state): # Event that handles the upward
movement of the joystick's left stick.
    if self.mode == 0 and not self.disable_servo_control:
        p = self.joint2_state + math.radians(2) # calculates the target
position of servo 2. It adds 2 radians to the current state of servo 2,
self.joint2_state.
        self.joint2.publish(CommandDuration(data=p, duration=50)) # Posted
a CommandDuration message to the ROS topic named
'joint2_controller/command_duration' to control the position of servo 2.
        self.joint2_state = p # Updated the state of servo 2
self.joint2_state, setting it to the newly calculated target position p.
    else:
        pass # No operation

def joy_callback(self, joy_msg: sensor_msgs.Joy): # For processing joystick
messages
    # joy_msg represents a joystick message, which is of type
sensor_msgs.Joy.

```



```

axes = dict(zip(AXES_MAP, joy_msg.axes))
axes_changed = False
hat_x, hat_y = axes['hat_x'], axes['hat_y'] # Gets the value of the
cross keys in the joystick message, hat_x is the horizontal value and hat_y is
the vertical value.
hat_xl, hat_xr = 1 if hat_x > 0.5 else 0, 1 if hat_x < -0.5 else 0 #
Determines whether the left stick moves left ( hat_xl ) or right ( hat_xr )
based on the cross key's lateral value.
hat_yu, hat_yd = 1 if hat_y > 0.5 else 0, 1 if hat_y < -0.5 else 0 #
Determines whether the left stick moves up ( hat_yu ) or down ( hat_yd ) based
on the vertical value of the cross keys.
buttons = list(joy_msg.buttons) # Gets the values of the buttons in the
joystick message and converts them to list form.
buttons.extend([hat_xl, hat_xr, hat_yu, hat_yd, 0]) # Add the movement
state of the cross key you just determined to the list of buttons, and add an
empty button value at the end.
buttons = dict(zip(BUTTON_MAP, buttons)) # A dictionary buttons is
constructed by pairing the name of a button with the corresponding button value,
where the key is the button name and the value is the corresponding button
value.
for key, value in axes.items(): # This is a for loop that iterates over
the joystick's dictionary of axis values, axes.
    if self.last_axes[key] != value: # Check if the value of the
current axis is different from the previous one
        axes_changed = True
if axes_changed:
    try:
        self.axes_callback(axes) # The axes_callback method is called,
passing the state of the current axis to this method for processing
    except Exception as e:
        rospy.logerr(str(e))
for key, value in buttons.items(): # This is a for loop that iterates
over the joystick's button state dictionary, buttons.
    new_state = ButtonState.Normal # Initialize the state of the button
to normal
    if value != self.last_buttons[key]: # Check if the current button's
state is different from the last one
        new_state = ButtonState.Pressed if value > 0 else
ButtonState.Released # If the value of the button is greater than 0, the button
state is set to pressed ( ButtonState.Pressed ), otherwise it is set to
released ( ButtonState.Released).
    else:
        new_state = ButtonState.Holding if value > 0 else
ButtonState.Normal

```

```

        callback = "".join([key, '_callback']) # Concatenates the name of
the current button with the _callback string, which is used to construct the
name of the corresponding callback method.
        if new_state != ButtonState.Normal:
            rospy.loginfo(key + ': ' + str(new_state)) # Record the current
button name and status to the ROS log for viewing during debugging and
monitoring.

            if hasattr(self, callback): # Use the hasattr() function to
check if the current object has an attribute called callback.
                try:
                    getattr(self, callback)(new_state) # Use the getattr()
function to dynamically retrieve an attribute named callback of the current
object and call the method referenced by this attribute, passing in the
parameter new_state, which calls the corresponding callback method to handle
the button event.
                except Exception as e:
                    rospy.logerr(str(e))

        self.last_buttons = buttons # Updated the last button status record
        self.last_axes = axes # Updated the last shaft status record.

if __name__ == "__main__":
    node = JoystickController() # Used to control and process input from the
Joystick and to initialize the ROS node
    try:
        rospy.spin() # Enter an infinite loop waiting to receive messages and
process them.
    except Exception as e:
        rospy.logerr(str(e))

```

## Initialize the Handle Control Node

Initialize the handle control node, and set the maximum speed to acquire the message of underlying driver topic.

```

rospy.init_node('joystick_control', anonymous=True)
self.max_linear = rospy.get_param('~max_linear', 0.4)
self.max_angular = rospy.get_param('~max_angular', 2.0)
self.machine = rospy.get_param('~machine', 'JetAuto')
self.disable_servo_control = rospy.get_param('~disable_servo_control',
'true')
cmd_vel = rospy.get_param('~cmd_vel', 'jetauto_controller/cmd_vel')

```

## Connect to the Underlying Driver Node

Subscribe to the topic of the handle control node, and connect it to the underlying driver node.

```

        self.joy_sub = rospy.Subscriber('joy', sensor_msg.Joy,
self.joy_callback)
        self.mecanum_pub = rospy.Publisher(cmd_vel, geo_msg.Twist, queue_size=1)

```

## Drive the Motor to Move

Convert the acquired handle control message into the linear velocity and angular velocity in the corresponding direction. Then publish the topic through the underlying driver node to push the motor to rotate.

```

def axes_callback(self, axes):
    twist = geo_msg.Twist()
    if self.machine != 'JetMega':
        twist.linear.y = misc.val_map(axes['lx'], -1, 1, -self.max_linear,
self.max_linear)
    else:
        twist.linear.y = 0
    twist.linear.x = misc.val_map(axes['ly'], -1, 1, -self.max_linear,
self.max_linear)
    twist.angular.z = misc.val_map(axes['rx'], -1, 1, -self.max_angular,
self.max_angular)
    self.mecanum_pub.publish(twist)

```

# Part 10 Keyboard Control

## 1. Operation Steps

1) Start JetAuto, then connect it to NoMachine.

2) Double click  to open command line terminal

3) Input command “**roslaunch jetauto\_controller jetauto\_controller.launch**” to enable the motion control service.

4) Input command “**roslaunch jetauto\_peripherals teleop\_key\_control.launch robot\_name:=“/”**” to enable motion control service.

5) Open a new terminal, then input command “**roslaunch jetauto\_peripherals teleop\_key\_control.launch robot\_name:=“/”**” to start keyboard control.

6) If you want to exit this program, press “**Ctrl+C**”. If it cannot exit, please press the key again.

## 2. Program Outcome

After the game starts, you can control the robot to move by pressing the specific key. The functions corresponding to the keys are listed below.

| Key | Function    | Note                                                                                                     |
|-----|-------------|----------------------------------------------------------------------------------------------------------|
| W   | Go forward  | When you short press the key, the robot will enter going forward mode. And it will keep going forward.   |
| S   | Go backward | When you short press the key, the robot will enter going backward mode. And it will keep going backward. |
| A   | Turn left   | When you long press the key, the robot will stop going forward or backward. And it will turn left.       |
| D   | Turn right  | When you long press the key, the robot will stop going forward or backward. And it will turn right.      |

### 3. Program Analysis

Use keyboard to control JetAuto to let keyboard control node to connect with the underlying driver node. Then through sending the keyboard action message to drive the motor.

The source code of this program is stored in  
**/home/jetauto\_ws/src/jetauto\_peripherals/scripts/teleop\_key\_control.py**

#### ◆Initialize the Keyboard Control Node

Initialize the keyboard control node and connect it with the underlying driver node.

```
rospy.init_node('jetauto_teleop')
mecanum_pub = rospy.Publisher('/jetauto_controller/cmd_vel', Twist,
queue_size=10)
```

#### ◆Acquire the Key Information

Read the key input through the system module.

```
if rlist:
    key = sys.stdin.read(1)
else:
    key = ''
```

#### ◆Set the Speed Change

According to the obtained key messages, set the speed change corresponding to each key.

```
if key == 'w':
    control_linear_vel = LIN_VEL
elif key == 'a':
    control_angular_vel = ANG_VEL
    control_linear_vel = 0
elif key == 'd':
    control_angular_vel = -ANG_VEL
    control_linear_vel = 0
elif key == 's':
    control_linear_vel = -LIN_VEL
elif key == '':
```

```

        #control_linear_vel = 0
        control_angular_vel = 0
    else:
        if (key == '\x03'):
            break

```

## ◆Drive the Motor to Move

Publish the topic with the obtained linear velocity X and angular velocity Z to drive the motor to move.

```

twist.linear.x = control_linear_vel
twist.linear.y = 0.0
twist.linear.z = 0.0

twist.angular.x = 0.0
twist.angular.y = 0.0
twist.angular.z = control_angular_vel

mecanum_pub.publish(twist)

```

## CODE

```

#!/usr/bin/env python3
# encoding: utf-8
import rospy
from geometry_msgs.msg import Twist

import sys, select, os
if os.name == 'nt':
    import msvcrt, time
else:
    import tty, termios

LIN_VEL = 0.2
ANG_VEL = 0.5

msg = """
Control Your Jetauto!
-----
Moving around:
    w
    a    s    d
CTRL-C to quit
"""

e = """
Communications Failed

```

```
"""
```

```
def getKey():
    if os.name == 'nt':
        timeout = 0.1
        startTime = rospy.get_time()
        while(1):
            if msvcrt.kbhit():
                if sys.version_info[0] >= 3:
                    return msvcrt.getch().decode()
                else:
                    return msvcrt.getch()
            elif time.time() - startTime > timeout:
                return ''

    tty.setraw(sys.stdin.fileno())
    rlist, _, _ = select.select([sys.stdin], [], [], 0.1)
    if rlist:
        key = sys.stdin.read(1)
    else:
        key = ''

    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key

if __name__ == "__main__":
    if os.name != 'nt':
        settings = termios.tcgetattr(sys.stdin)

    rospy.init_node('jetauto_teleop')
    mecanum_pub = rospy.Publisher('/jetauto_controller/cmd_vel', Twist,
    queue_size=10)

    status = 0
    target_linear_vel = 0.0
    target_angular_vel = 0.0
    control_linear_vel = 0.0
    control_angular_vel = 0.0

    try:
        print(msg)
        while not rospy.is_shutdown():
            key = getKey()
            if key == 'w':
                control_linear_vel = LIN_VEL
            elif key == 'a':
                control_angular_vel = ANG_VEL
                control_linear_vel = 0
```



```

elif key == 'd':
    control_angular_vel = -ANG_VEL
    control_linear_vel = 0
elif key == 's':
    control_linear_vel = -LIN_VEL
elif key == ' ':
    #control_linear_vel = 0
    control_angular_vel = 0
else:
    if (key == '\x03'):
        break

twist = Twist()

twist.linear.x = control_linear_vel
twist.linear.y = 0.0
twist.linear.z = 0.0

twist.angular.x = 0.0
twist.angular.y = 0.0
twist.angular.z = control_angular_vel

mecanum_pub.publish(twist)

except:
    print(e)

finally:
    twist = Twist()
    twist.linear.x = 0.0
    twist.linear.y = 0.0
    twist.linear.z = 0.0
    twist.angular.x = 0.0
    twist.angular.y = 0.0
    twist.angular.z = 0.0
    mecanum_pub.publish(twist)

if os.name != 'nt':
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)

```