

Keyboard Control

Keyboard Control

1. Course Content
2. Preparation
 - 2.1 Content Description
 - 2.2 Starting the Agent
3. Running the Example
 - 3.1 Starting Keyboard Control
 - 3.2 Key Control Instructions
 - 3.2.1 Direction Control
 - 3.2.2 Speed Control
4. Source Code Analysis
 - 4.1 View the Node Relationship Graph
 - 4.2 Viewing Topic Messages and Message Types
 - 4.3 Program Flowchart
 - 4.4 Source Code Analysis
 - 4.41 Published Topic: cmd_vel
 - 4.42 Movement Dictionary and Speed Dictionary

1. Course Content

Learn how to control robot movement using the keyboard and its principles.

After running the program, use keyboard keys to publish speed topics to control the robot chassis' movement.

2. Preparation

2.1 Content Description

This course uses the Jetson Orin NX as an example. For Raspberry Pi and Jetson Nano boards, you need to open a terminal and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this course in the terminal. For instructions on entering the Docker container, refer to the product tutorial **[Configuration and Operation Guide] - [Entering the Docker (Jetson Nano and Raspberry Pi 5 users see here)]**. For Orin and NX boards, simply open a terminal and enter the commands mentioned in this course.

2.2 Starting the Agent

Note: The Docker agent must be started before testing all examples. If it's already started, you don't need to restart it.

Enter the command in the vehicle terminal:

```
sh start_agent.sh
```

The terminal prints the following message, indicating a successful connection.

```
Jetson@yahboom: ~
Jetson@yahboom: ~ 177x26
Jetson@yahboom:~$ ./open_agent.sh
[1749538760.063253] Info | TermiosAgentLinux.cpp | Init | running... | fd: 3
[1749538760.064012] Info | Root.cpp | set_verbose_level | logger setup | verbose_level: 4
[1749538760.941420] Info | Root.cpp | create_client | create | client_key: 0x0DA64EFC, session_id: 0x81
[1749538760.941537] Info | SessionManager.hpp | establish_session | session established | client_key: 0x0DA64EFC, address: 0
[1749538760.979971] Info | ProxyClient.cpp | create_participant | participant created | client_key: 0x0DA64EFC, participant_id: 0x000(1)
[1749538760.983835] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x000(2), participant_id: 0x000(1)
[1749538760.988244] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x000(3), participant_id: 0x000(1)
[1749538760.993117] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x000(5), publisher_id: 0x000(3)
[1749538760.997675] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x001(2), participant_id: 0x000(1)
[1749538761.001121] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x001(3), participant_id: 0x000(1)
[1749538761.007277] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x001(5), publisher_id: 0x001(3)
[1749538761.010634] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x002(2), participant_id: 0x000(1)
[1749538761.014296] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x002(3), participant_id: 0x000(1)
[1749538761.020171] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x002(5), publisher_id: 0x002(3)
[1749538761.023939] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x003(2), participant_id: 0x000(1)
[1749538761.029173] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x003(3), participant_id: 0x000(1)
[1749538761.034377] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x003(5), publisher_id: 0x003(3)
[1749538761.038946] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x004(2), participant_id: 0x000(1)
[1749538761.042215] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x004(3), participant_id: 0x000(1)
[1749538761.048422] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x004(5), publisher_id: 0x004(3)
[1749538761.051660] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x005(2), participant_id: 0x000(1)
[1749538761.057494] Info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x0DA64EFC, publisher_id: 0x005(3), participant_id: 0x000(1)
[1749538761.062183] Info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x0DA64EFC, datawriter_id: 0x005(5), publisher_id: 0x005(3)
[1749538761.066842] Info | ProxyClient.cpp | create_topic | topic created | client_key: 0x0DA64EFC, topic_id: 0x006(2), participant_id: 0x000(1)
[1749538761.071217] Info | ProxyClient.cpp | create_subscriber | subscriber created | client_key: 0x0DA64EFC, subscriber_id: 0x000(4), participant_id: 0x000(1)
```

3. Running the Example

3.1 Starting Keyboard Control

Note:

- The Jetson Nano and Raspberry Pi series controllers must first enter the Docker container (for steps, see the [Docker Course Section - Entering the Robot's Docker Container]).

Run the keyboard control node on the vehicle terminal or in the virtual machine:

```
ros2 run yahboomcar_ctrl yahboom_keyboard
```

```
Jetson@yahboom: ~
Jetson@yahboom:~$ ros2 run yahboomcar_ctrl yahboom_keyboard

Control Your SLAM-Bot!
-----
Moving around:
  u   i   o
  j   k   l
  m   ,   .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
t/T : x and y speed switch
s/S : stop keyboard control
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2      turn 1.0
```

3.2 Key Control Instructions

3.2.1 Direction Control

[i] or [I]	[linear, 0]	[u] or [U]	[linear, angular]
[.]	[-linear, 0]	[o] or [O]	[linear, -angular]
[j] or [J]	[0, angular]	[m] or [M]	[-linear, -angular]
[l] or [L]	[0, -angular]	[.]	[-linear, angular]

3.2.2 Speed Control

Key	Speed Change	Key	Speed Change
[q]	Increase both linear and angular velocities by 10%	[z]	Decrease both linear and angular velocities by 10%
[w]	Increase only linear velocity by 10%	[x]	Decrease only linear velocity by 10%
[e]	Increase only angular velocity by 10%	[c]	Decrease only angular velocity by 10%
[t]	Switch linear velocity between X-axis and Y-axis	[s]	Stop keyboard control

4. Source Code Analysis

Source code path:

jetson orin nano, jetson orin NX:

```
/home/jetson/yahboomcar_ws/src/yahboomcar_ctrl/yahboomcar_ctrl/yahboom_keyboard.py
```

Jetson Orin Nano, Raspberry Pi:

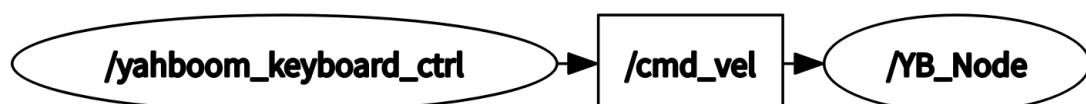
You need to enter Docker first.

```
root/yahboomcar_ws/src/yahboomcar_ctrl/yahboomcar_ctrl/yahboom_keyboard.py
```

4.1 View the Node Relationship Graph

Open a terminal and enter the command:

```
ros2 run rqt_graph rqt_graph
```



From the node relationship diagram, we can see:

yahboom_keyboard_ctrl: Controls the robot chassis by publishing the **/cmd_vel** topic

/YB_Node: The robot chassis node subscribes to the **/cmd_vel** topic and uses the inverse kinematic solution to calculate the speed of each wheel, thereby controlling the robot's movement.

4.2 Viewing Topic Messages and Message Types

Open a terminal and enter the command:

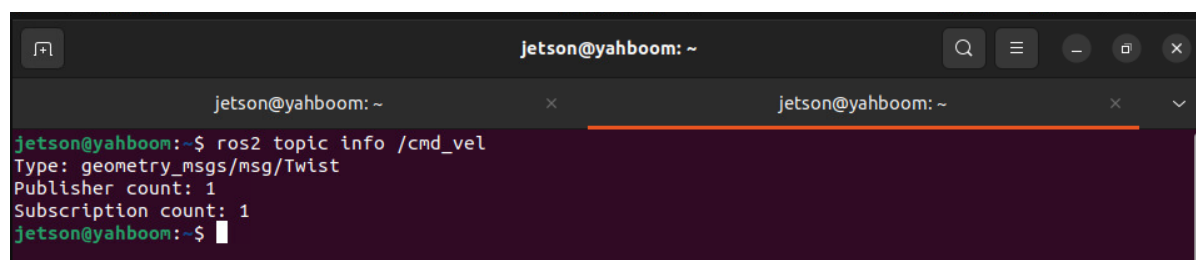
```
ros2 topic echo /cmd_vel
```

When controlling the robot chassis' movements using the keyboard, data is published to the **/cmd_vel** topic by printing messages.

```
x: 0.2
y: 0.0
z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 0.2
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

Enter the following command to view the message type of the **/cmd_vel** topic:

```
ros2 topic info /cmd_vel
```



```
jetson@yahboom: ~
jetson@yahboom: ~
jetson@yahboom:~$ ros2 topic info /cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 1
jetson@yahboom:~$
```

The Type column indicates that the message type of the **/cmd_vel** topic is **geometry_msgs/msg/Twist**. Enter the following command to view the composition of the **geometry_msgs/msg/Twist** message type:

```
ros2 interface show geometry_msgs/msg/Twist
```

```
jetson@yahboom:~$ ros2 interface show geometry_msgs/msg/Twist
# This expresses velocity in free space broken into its linear and angular parts.

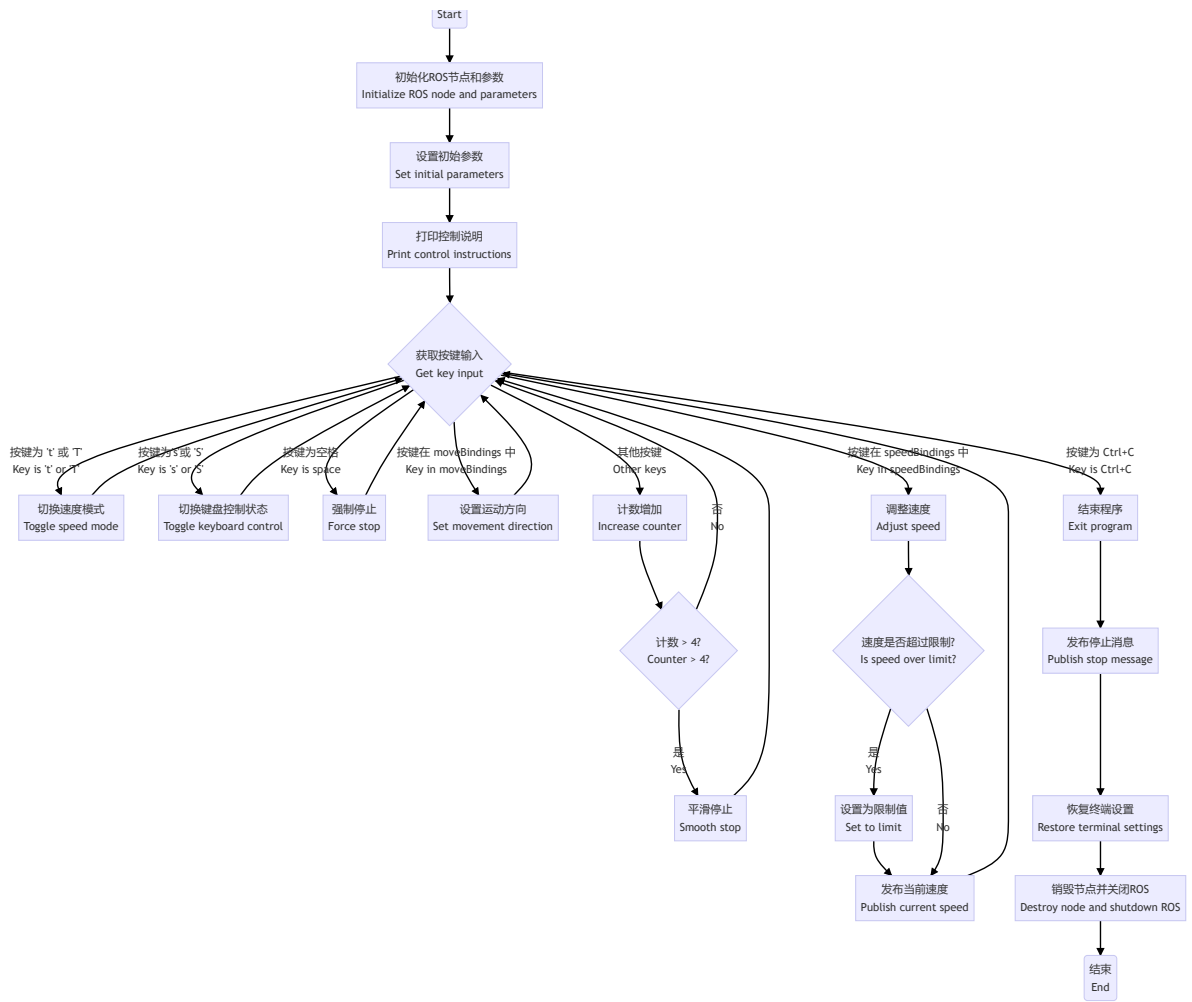
Vector3  linear
  float64 x
  float64 y
  float64 z
Vector3  angular
  float64 x
  float64 y
  float64 z
jetson@yahboom:~$
```

From the composition of the above message types, we can see that the robot chassis movement is controlled by two vector groups: linear (linear velocity) and angular (angular velocity). Each data element is a float64 floating-point number. The following explains the meaning of each data element.

- linear
 - float64 x: x-axis velocity
 - float64 y: y-axis velocity
 - float64 z: z-axis velocity
- angular
 - float64 x: x-axis angular velocity
 - float64 y: y-axis angular velocity
 - float64 z: z-axis velocity : z-axis angular velocity

Because the robot chassis can only move within a two-dimensional plane, only linear-x (x-axis velocity), linear-y (y-axis velocity), and angular-z (z-axis angular velocity) are published when controlling the robot via the keyboard.

4.3 Program Flowchart



4.4 Source Code Analysis

Source Code Path:

Jetson Orin Nano, Jetson Orin NX:

```
/home/jetson/yahboomcar_ws/src/yahboomcar_ctrl/yahboomcar_ctrl/yahboom_keyboard.py
```

Jetson Orin Nano, Raspberry Pi:

You need to first enter Docker.

```
root/yahboomcar_ws/src/yahboomcar_ctrl/yahboomcar_ctrl/yahboom_keyboard.py
```

4.41 Published Topic: cmd_vel

```
pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
```

Just package the speed and publish it via `pub.publish(twist)`. The chassis' speed subscriber will receive the speed data and then drive the car.

4.42 Movement Dictionary and Speed Dictionary

- The movement dictionary mainly stores characters related to direction control

```
moveBindings = {  
    'i': (1, 0),  
    'o': (1, -1),  
    'j': (0, 1),  
    'l': (0, -1),  
    'u': (1, 1),  
    ',': (-1, 0),  
    '.': (-1, 1),  
    'm': (-1, -1),  
    'I': (1, 0),  
    'O': (1, -1),  
    'J': (0, 1),  
    'L': (0, -1),  
    'U': (1, 1),  
    'M': (-1, -1),  
}
```

- The speed dictionary mainly stores the characters related to speed control

```
speedBindings = {  
    'Q': (1.1, 1.1),  
    'Z': (.9, .9),  
    'W': (1.1, 1),  
    'X': (.9, 1),  
    'E': (1, 1.1),  
    'C': (1, .9),  
    'q': (1.1, 1.1),  
    'z': (.9, .9),  
    'w': (1.1, 1),  
}
```

```

    'x': (.9, 1),
    'e': (1, 1.1),
    'c': (1, .9),
}

```

4.43 Get the current key information

```

def getKey(self):
    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, self.settings)
    return key

```

4.44 Determine the key value and publish the /cmd_vel speed topic

```

while (1):
    key = yahboom_keyboard.getKey()
    if key=="t" or key == "T": xspeed_switch = not xspeed_switch
    elif key == "s" or key == "S":
        print ("stop keyboard control: {}".format(not stop))
        stop = not stop
    if key in moveBindings.keys():
        x = moveBindings[key][0]
        th = moveBindings[key][1]
        count = 0
    elif key in speedBindings.keys():
        speed = speed * speedBindings[key][0]
        turn = turn * speedBindings[key][1]
        count = 0
        if speed > yahboom_keyboard.linear_speed_limit:
            speed = yahboom_keyboard.linear_speed_limit
            print("Linear speed limit reached!")
        if turn > yahboom_keyboard.angular_speed_limit:
            turn = yahboom_keyboard.angular_speed_limit
            print("Angular speed limit reached!")
        print(yahboom_keyboard.vels(speed, turn))
        if (status == 14): print(msg)
        status = (status + 1) % 15
    elif key == ' ': (x, th) = (0, 0)
    else:
        count = count + 1
        if count > 4: (x, th) = (0, 0)
        if (key == '\x03'): break
    if xspeed_switch: twist.linear.x = speed * x
    else: twist.linear.y = speed * x
    twist.angular.z = turn * th
    if not stop: yahboom_keyboard.pub.publish(twist)
    if stop: yahboom_keyboard.pub.publish(Twist())

```