

Robotic arm ROS control

Before starting this function, you need to close the process of the big program and APP. If you need to start the big program and APP again later, start the terminal,

```
bash ~/dofbot_pro/APP_DOFBOT_PRO/start_app.sh
```

1. Function description

After the program is started, you can control the posture of the robot arm and the value of a single servo through ros. Here, the command line tool of ros2 topic is used to change the posture of the robot arm and the value of a single servo. In the subsequent program, the robot arm is also controlled through the topic.

2. Start and operate

2.1. Start command

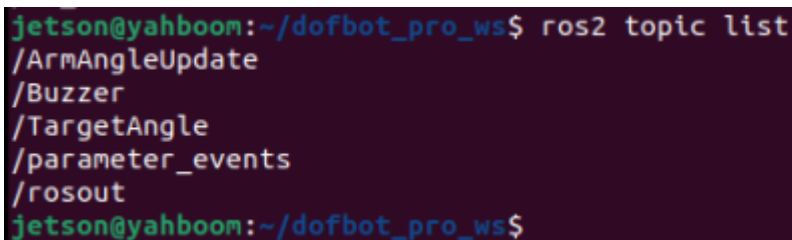
After opening the terminal, enter the following command

```
#Start the bottom layer  
ros2 run dofbot_pro_driver arm_driver
```

2.2. Operation steps

Start the terminal and enter the following command,

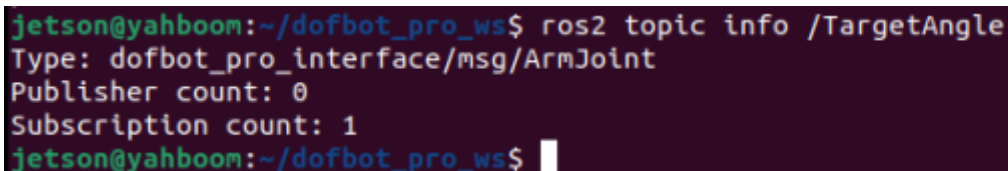
```
ros2 topic list
```



```
jetson@yahboom:~/dofbot_pro_ws$ ros2 topic list  
/ArmAngleUpdate  
/Buzzer  
/TargetAngle  
/parameter_events  
/rosout  
jetson@yahboom:~/dofbot_pro_ws$
```

As shown in the figure above, the displayed /TargetAngle is the topic of controlling the robotic arm. We use the rostopic tool to publish the message of this topic. First, look at the relevant information of this topic. Enter in the terminal,

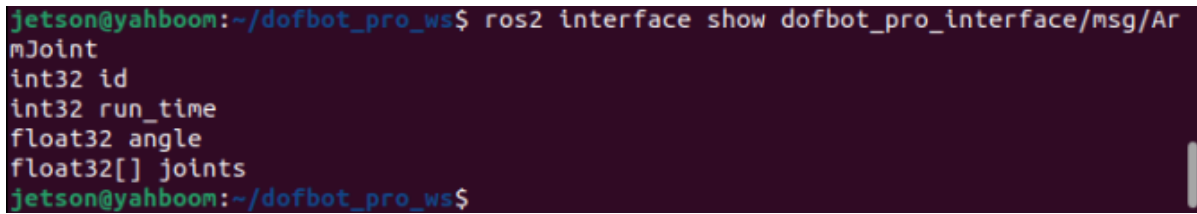
```
ros2 topic info /TargetAngle
```



```
jetson@yahboom:~/dofbot_pro_ws$ ros2 topic info /TargetAngle  
Type: dofbot_pro_interface/msg/ArmJoint  
Publisher count: 0  
Subscription count: 1  
jetson@yahboom:~/dofbot_pro_ws$
```

As shown in the figure above, the data type name of the topic is dofbot_pro_info/ArmJoint. This is our customized message data interface. You can view the specific content of the data type through the following command. Enter the terminal,

```
ros2 interface show dofbot_pro_interface/msg/ArmJoint
```



```
jetson@yahboom:~/dofbot_pro_ws$ ros2 interface show dofbot_pro_interface/msg/ArmJoint
int32 id
int32 run_time
float32 angle
float32[] joints
jetson@yahboom:~/dofbot_pro_ws$
```

As shown in the figure above, the specific content of the dofbot_pro_interface/ArmJoint type data is displayed. The properties of each variable are as follows:

- id: The ID of the corresponding servo. When controlling a single servo, you need to specify this value. The value range is [1-6];
- run_time: The running time of the servo, which means that the robot arm needs to run to the specified posture or the time for a single servo to run to the specified angle within this time. The unit is ms
- angle: The control angle. When controlling a single servo, you need to set the angle value after running. The value range is [0,180]
- joints: An array of length 6. When controlling the values of 6 servos, write the values of 6 servos into this data. The array subscript corresponds to the id-1 of each servo. The control range of each servo angle is [0,180]

We can test it with the following command,

Control a single servo. Take the control of servo No. 1 as an example. Enter in the terminal,

```
ros2 topic pub /TargetAngle dofbot_pro_interface/ArmJoint "id: 1
run_time: 1000
angle: 120.0
joints: []"
```

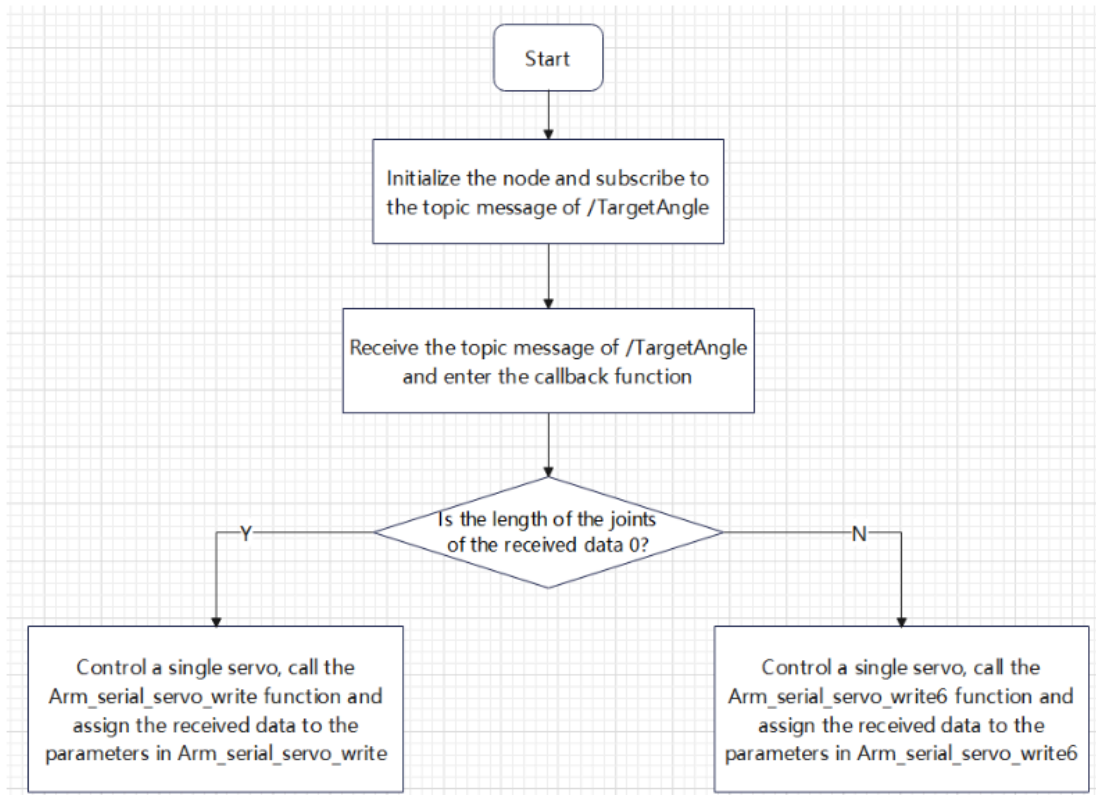
It is recommended to use Tab on the single-touch keyboard to complete the topic name and the content of the following data, and then modify the corresponding value. After entering, press Enter to send the topic data. The No. 1 servo of the robotic arm will rotate to a state of 120°.

To control multiple servos, take the robotic arm to present an upward straight posture as an example. Enter in the terminal

```
ros2 topic pub /TargetAngle dofbot_pro_interface/ArmJoint "id: 1
run_time: 1000
angle: 120.0
joints: [90,90,90,90,90,90]"
```

Similarly, we recommend using Tab on the single-touch keyboard to complete the topic name and the content of the following data, and then modify the corresponding value. After entering, press Enter to send the topic data. The robotic arm will present an upward straight posture.

3. Program flow chart



4. Core code analysis

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_driver/dofbot_pro_driver/arm_driver.py
```

Import library files

```
# Bottom-level driver library files
from Arm_Lib import Arm_Device
# Custom message data
from dofbot_pro_interface.msg import *
# Import rospy library
import rospy
from rospy.node import Node
```

Define a subscriber and subscribe to the topic of TargetAngle,

```
self.sub_Arm = self.create_subscription(ArmJoint, "TargetAngle", self.Armcallback,
queue_size=1000)
```

Write a callback function to process subscribed messages and send data to the underlying control.

```
def Armcallback(self, msg):
    if not isinstance(msg, ArmJoint):
        print("-----")
        return
    arm_joint = ArmJoint()
    print("msg.joints: ", msg.joints)
```

```

print("msg.joints: ",msg.run_time)
if len(msg.joints) != 0:
    arm_joint.joints = self.cur_joints
    for i in range(2):
        print("-----")
        def Armcallback(self,msg):
            if not isinstance(msg, ArmJoint):
                print("-----")
                return
            arm_joint = ArmJoint()
            print("msg.joints: ",msg.joints)
            print("msg.joints: ",msg.run_time)
            if len(msg.joints) != 0:
                arm_joint.joints = self.cur_joints
                for i in range(2):
                    print("-----")
                    self.Arm.Arm_serial_servo_write6(msg.joints[0],
msg.joints[1],msg.joints[2],msg.joints[3],msg.joints[4],msg.joints[5],time=msg.ru
n_time)

                    self.cur_joints = list(msg.joints)
                    self.ArmPubUpdate.publish(arm_joint)
                #time.sleep(0.01)
            else:
                arm_joint.id = msg.id
                arm_joint.angle = msg.angle
                for i in range(2):
                    print("msg.id: ",msg.id)
                    self.Arm.Arm_serial_servo_write(msg.id, msg.angle, msg.run_time)
                    self.cur_joints[msg.id - 1] = msg.angle
                    self.ArmPubUpdate.publish(arm_joint)
        self.Arm.Arm_serial_servo_write6(msg.joints[0],
msg.joints[1],msg.joints[2],msg.joints[3],msg.joints[4],msg.joints[5],time=msg.ru
n_time)
        self.cur_joints = list(msg.joints)
        self.ArmPubUpdate.publish(arm_joint)
        #time.sleep(0.01)
    else:
        arm_joint.id = msg.id
        arm_joint.angle = msg.angle
        for i in range(2):
            print("msg.id: ",msg.id)
            self.Arm.Arm_serial_servo_write(msg.id, msg.angle, msg.run_time)
            self.cur_joints[msg.id - 1] = msg.angle
            self.ArmPubUpdate.publish(arm_joint)

```

The code shows the length of the joints received by the subscription, that is, `len(msg.joints)` in the code. The length of the joint determines whether to control 6 servos or a single servo. If it is not 0, it means **controlling 6 servos**. When we publish data to the bottom layer, we need to **use the `Arm_serial_servo_write6` function to send data**; on the contrary, if it is 0, it means **controlling a single servo**, and we need to **use the `Arm_serial_servo_write` function to send data**.

`Arm_serial_servo_write6` function is provided by the library `Arm_Lib`, which is a custom library for controlling the bottom-level driver. `Arm_serial_servo_write6` has 7 parameters, which are the values of the 6 servos and the runtime. After we receive the topic data, we can enter the corresponding values of the joints and runtime of the 6 servos.

Arm_serial_servo_write function is also provided by the library Arm_Lib. There are three parameters that need to be input, namely the ID of the servo to be controlled, the angle of the servo to be controlled, and the runtime. After we receive the topic data, we can enter the ID value, the control angle value angle, and the runtime value accordingly.