

Robotic Arm ROS Control

Before starting this function, you need to close the large program and APP processes. If you need to restart the large program and APP later, start them from the terminal:

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

1. Function Description

After the program starts, you can control the robotic arm's pose and individual servo values through ROS. Here we use the ros2 topic command-line tools to change the robotic arm's pose and individual servo values. In subsequent programs, the robotic arm is also controlled through topics.

2. Startup and Operation

2.1. Startup Commands

After opening the terminal, enter the following commands:

```
#Start underlying layer  
ros2 run dofbot_pro_driver arm_driver
```

2.2. Operation Steps

Enter the following command in the terminal:

```
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 for controlling the robotic arm. We use the rostopic tool to publish messages to this topic. First, let's 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 this topic is dofbot_pro_info/ArmJoint. This is our custom message data interface. You can view the specific content of this data type through the following command. Enter in 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, it displays the specific content in the dofbot_pro_interface/ArmJoint type data. The attributes of each variable are as follows:

- id: Corresponds to the servo id. When controlling a single servo, you need to specify this value. The value range is [1-6];
- run_time: Servo running time, indicating the time within which the robotic arm needs to run to the specified pose or the time for a single servo to run to the specified angle, unit is ms
- angle: 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 the 6 servos into this data. The array index corresponds to each servo's id-1. The control range of each servo angle is [0,180]

We can test through the following commands:

Control a single servo, taking controlling servo 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: []"
```

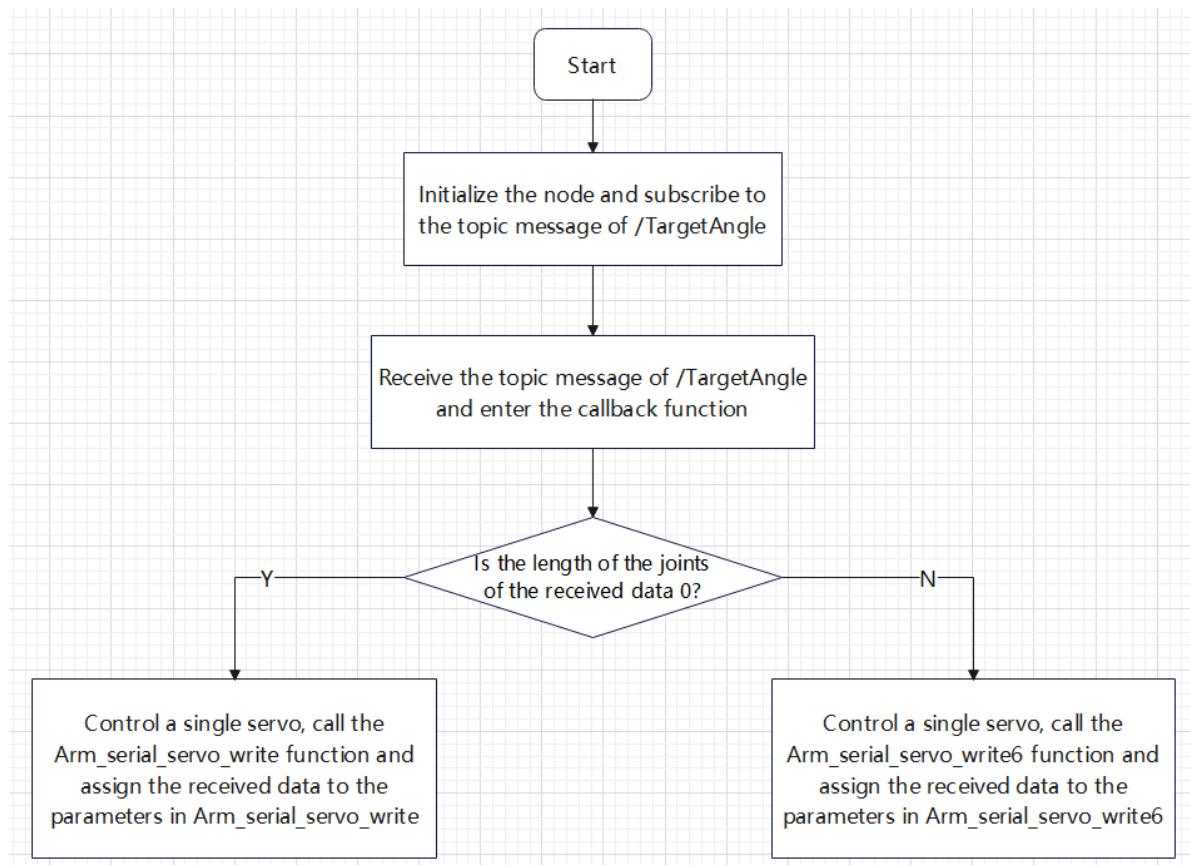
Here it is recommended to use Tab on the keyboard to complete the topic name and the following data content, then modify the corresponding values. After entering, press Enter to send the topic data. Servo 1 of the robotic arm will rotate to a 120° state.

Control multiple servos, taking controlling the robotic arm to present an upward straight pose 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 keyboard to complete the topic name and the following data content, then modify the corresponding values. After entering, press Enter to send the topic data. The robotic arm will present an upward straight pose.

3. Program Flowchart



4. Core Code Analysis

Code path:

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

Import library files

```
#Underlying driver library file
from Arm_Lib import Arm_Device
#Custom message data
from dofbot_pro_interface.msg import *
#import rosplay
import rclpy
from rclpy.node import Node
```

Define a subscriber to subscribe to the TargetAngle topic:

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

Write callback function to process subscribed messages and send data to 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("-----")
        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.run_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 explains the length of the subscribed received joints, which is `len(msg.joints)` in the code. The length of the joint is used to determine whether to control 6 servos or a single servo. If it is not 0, it means **control 6 servos**. When we publish data to the underlying layer, we need to **use the Arm_serial_servo_write6 function to send data**; conversely, if it is 0, it means **control a single servo**, which requires **using the Arm_serial_servo_write function to send data**.

The **Arm_serial_servo_write6** function is provided by the `Arm_Lib` library. The `Arm_Lib` library is a custom library we created for controlling underlying drivers. `Arm_serial_servo_write6` has 7 parameters, which are the values of 6 servos and the running time. After receiving the topic data, we can input the 6 servo values joints and runtime values accordingly.

The **Arm_serial_servo_write** function is also provided by the `Arm_Lib` library. Here there are three parameters that need to be input: the id of the servo to be controlled, the angle of the servo to be controlled, and the running time. After receiving the topic data, we can input the id value, control angle value angle, and runtime value accordingly.