

Robotic Arm ROS Control

Before starting this function, you need to close the process of the big program and APP. Enter the following program in the terminal to close the process of the big program and APP.

```
sh ~/app_Arm/kill_YahboomArm.sh
sh ~/app_Arm/stop_app.sh
```

If you need to start the big program and APP again later, start the terminal.

```
sudo systemctl start yahboom_arm.service
sudo systemctl start yahboom_app.service
```

1. Functional description

After the program is started, the posture of the robot arm and the value of a single servo can be controlled through ros. Here, the rostopic command line tool is used to change the posture of the robot arm and the value of a single servo. In subsequent programs, the robot arm is also controlled through topics.

2. Start and operate

2.1. Start command

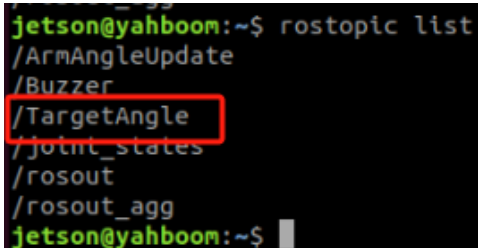
After opening the terminal, enter the following command

```
#Start roscore. After launching the launch file, you do not need to start
roscore
roscore
#Start the bottom layer
roslaunch dofbot_pro_info arm_driver.py
```

2.2. Operation steps

Start the terminal and enter the following command,

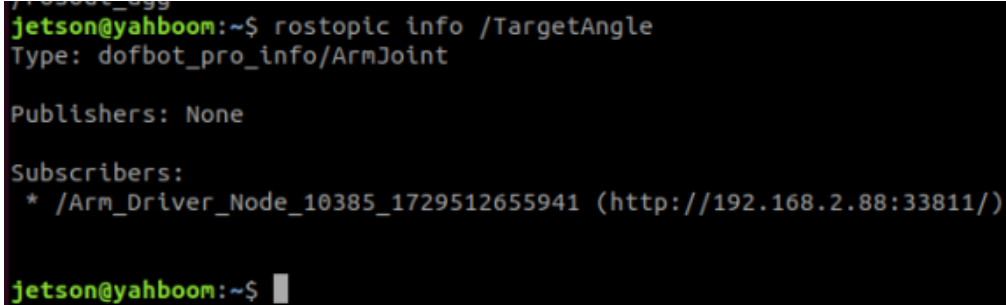
```
rostopic list
```



```
jetson@yahboom:~$ rostopic list
/ArmAngleUpdate
/Buzzer
/TargetAngle
/joint_states
/rosout
/rosout_agg
jetson@yahboom:~$
```

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

```
rostopic info /TargetAngle
```



```
jetson@yahboom:~$ rostopic info /TargetAngle
Type: dofbot_pro_info/ArmJoint

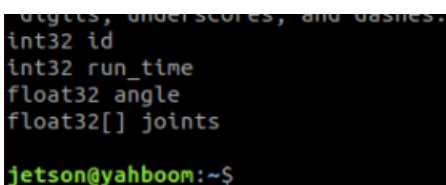
Publishers: None

Subscribers:
* /Arm_Driver_Node_10385_1729512655941 (http://192.168.2.88:33811/)

jetson@yahboom:~$
```

As shown in the figure above, the data type name of the 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,

```
rosmmsg show dofbot_pro_info/ArmJoint
```



```
digits, underscores, and dashes.
int32 id
int32 run_time
float32 angle
float32[] joints

jetson@yahboom:~$
```

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

- id: corresponds to the id of the servo. When controlling a single servo, you need to specify this value. The value range is [1-6];
- run_time: servo running time, which means that within this time, the robot arm needs to run to the specified posture or the time for a single servo to run to the specified angle. The 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 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 through the following instructions.

To control a single servo, take the control of servo No. 1 as an example. Enter the terminal,

```
rostopic pub /TargetAngle dofbot_pro_info/ArmJoint "id: 1
run_time: 1000
angle: 120.0
joints:
- 0"
```

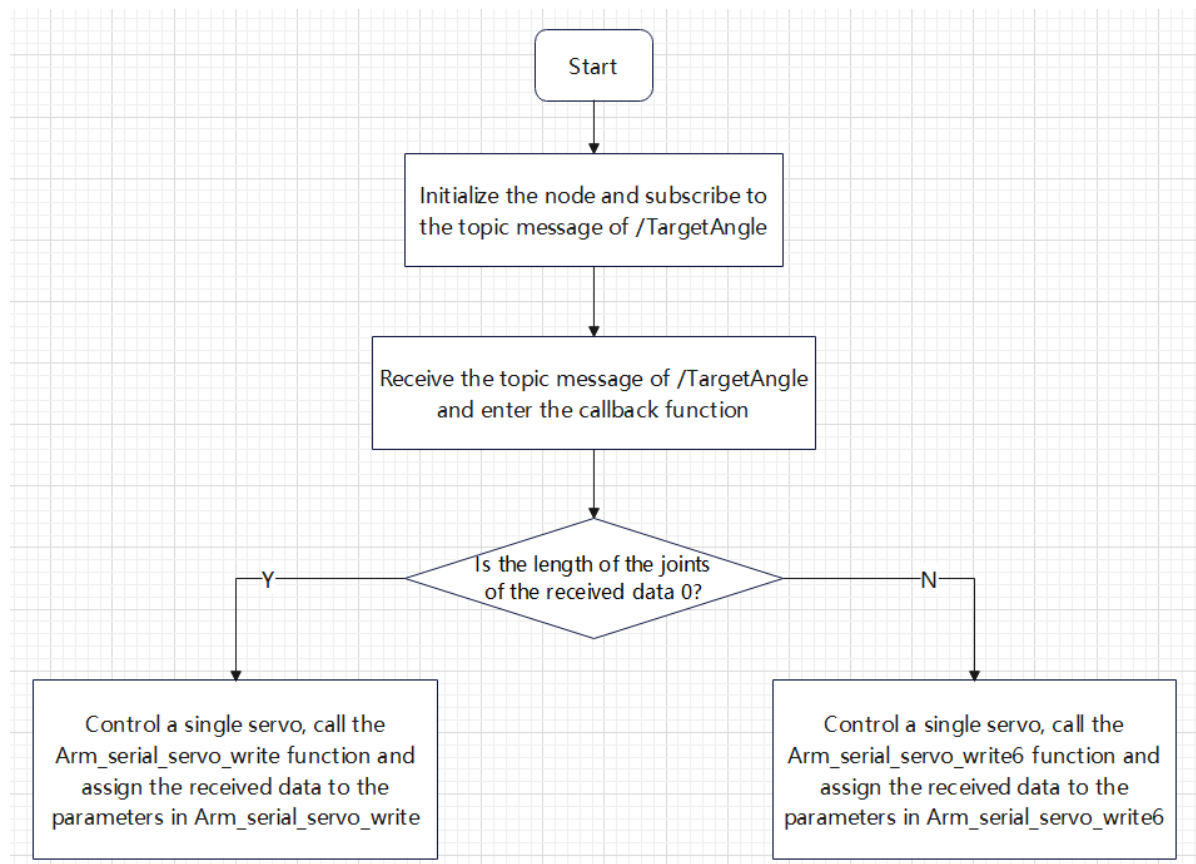
It is recommended to use Tab on the single-tap keyboard to complete the topic name and the content of the following data, and then modify the corresponding value. After entering, press Enter to send the topic data. The No. 1 servo of the robot arm will rotate to a state of 120°.

To control multiple servos, take the robot arm to show an upward straight posture as an example, enter the terminal

```
rostopic pub /TargetAngle dofbot_pro_info/ArmJoint "id: 1
run_time: 1000
angle: 120.0
joints: [90,90,90,90,90,90]"
```

Similarly, we recommend using Tab on the single-tap 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, and 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_info/scripts/arm_driver.py

Import library file

```
#Low-level driver library file
from Arm_Lib import Arm_Device
#Customized message data
from dofbot_pro_info.msg import *
#Import rospy library
import rospy
```

Define a subscriber to subscribe to the TargetAngle topic,

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

Write a callback function to process the subscribed message and send the 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.run_time: ",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.r
un_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, which 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**.

The **`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.

The **`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.