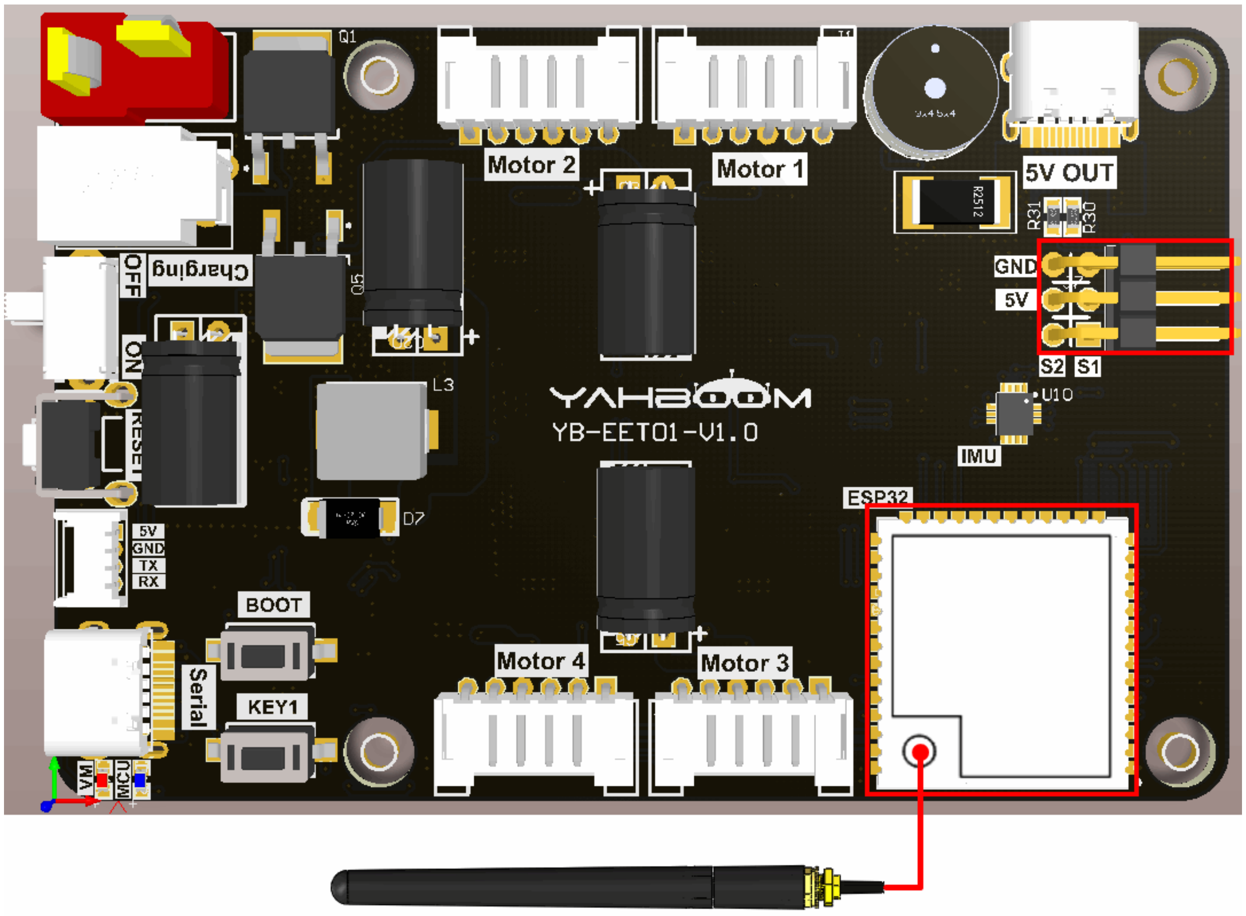
Subscribe PWM servo topics

1. Experimental purpose

Learn ESP32-microROS components, access the ROS2 environment, and subscribe to the topic of controlling the angle of the PWM servo.

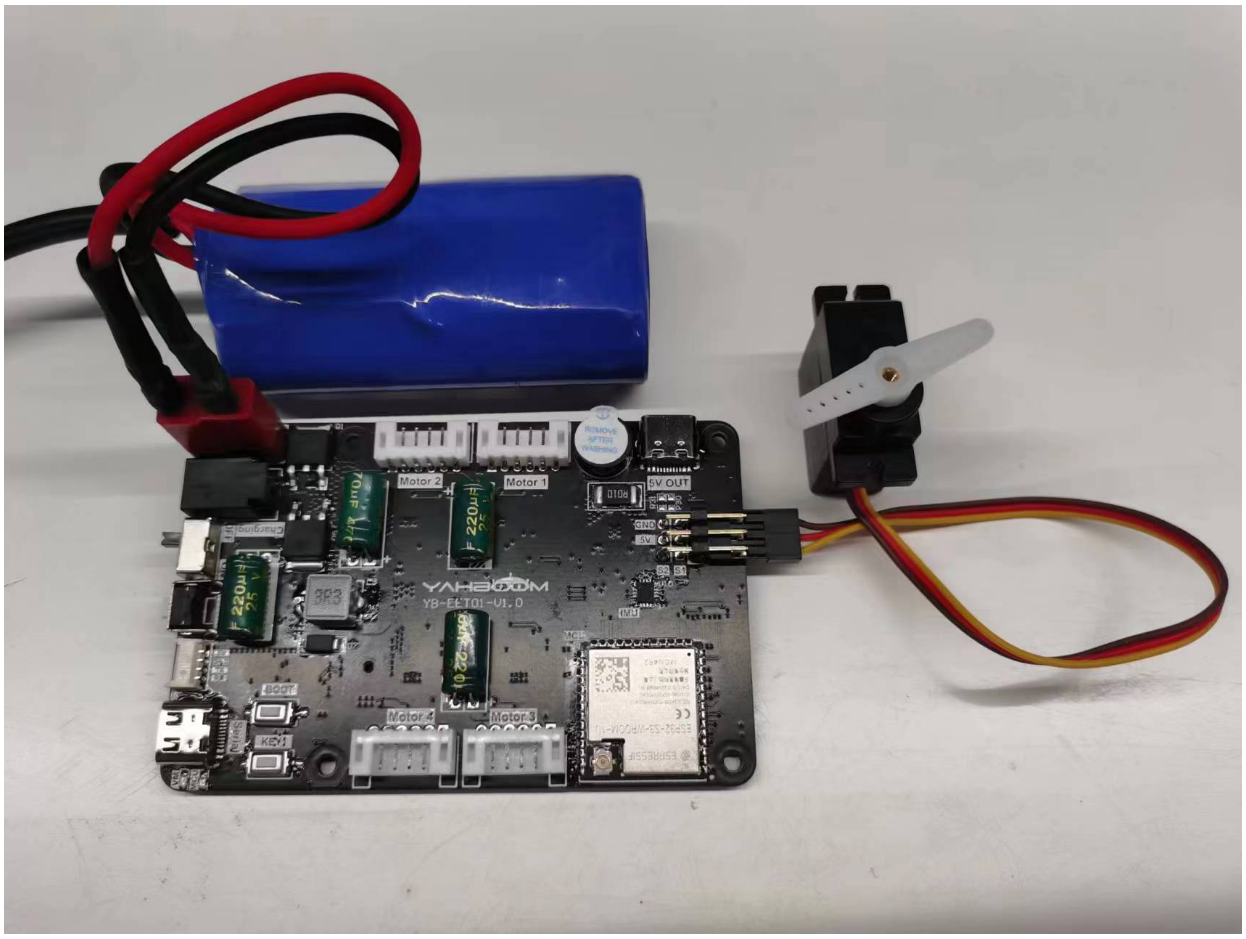
2. Hardware connection

As shown in the figure below, the microROS control board integrates the ESP32-S3-WROOM core module, which has its own wireless WiFi function. The ESP32-S3 core module needs to be connected to an antenna and has two PWM servo interfaces. An additional PWM servo is required. You can see the effect only after connecting it. You also need to connect the type-C data cable to the computer and the microROS control board for the firmware burning function.



The bottom of the PWM servo port is S1, and the top is S2. Please connect according to the silk screen. The brown wire is connected to GND, the red wire is connected to 5V, and the yellow wire is connected to S1/S2.

PWM servo connection diagram

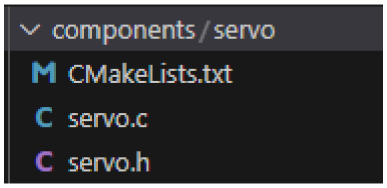


3. Core code analysis

The virtual machine path corresponding to the program source code is as follows

```
~/esp/Samples/microros_samples/servo_subscriber
```

Since the PWM servo is used this time, and the components of the PWM servo have been made in the previous routine, the components of the PWM servo need to be copied to the components directory of the project. Call Servo_Init at the beginning of the program to initialize the PWM servo.



```
components / servo
  CMakeLists.txt
  servo.c
  servo.h
```

Get the WiFi name and password to connect from the IDF configuration tool.

```
#define ESP_WIFI_SSID      CONFIG_ESP_WIFI_SSID
#define ESP_WIFI_PASS      CONFIG_ESP_WIFI_PASSWORD
#define ESP_MAXIMUM_RETRY  CONFIG_ESP_MAXIMUM_RETRY
```

The uros_network_interface_initialize function will connect to WiFi hotspots based on the WiFi configuration in IDF.

```
ESP_ERROR_CHECK(uros_network_interface_initialize());
```

Then obtain ROS_NAMESPACE, ROS_DOMAIN_ID, ROS_AGENT_IP and ROS_AGENT_PORT from the IDF configuration tool.

```
#define ROS_NAMESPACE      CONFIG_MICRO_ROS_NAMESPACE
#define ROS_DOMAIN_ID      CONFIG_MICRO_ROS_DOMAIN_ID
#define ROS_AGENT_IP       CONFIG_MICRO_ROS_AGENT_IP
#define ROS_AGENT_PORT     CONFIG_MICRO_ROS_AGENT_PORT
```

Initialize the configuration of microROS, in which ROS_DOMAIN_ID, ROS_AGENT_IP and ROS_AGENT_PORT are modified in the IDF configuration tool according to actual needs.

```
    rcl_allocator_t allocator = rcl_get_default_allocator();
    rclc_support_t support;

    // 创建rcl初始化选项
    // Create init_options.
    rcl_init_options_t init_options = rcl_get_zero_initialized_init_options();
    RCCHECK(rcl_init_options_init(&init_options, allocator));
    // 修改ROS域ID
    // change ros domain id
    RCCHECK(rcl_init_options_set_domain_id(&init_options, ROS_DOMAIN_ID));

    // 初始化rmw选项
    // Initialize the rmw options
    rmw_init_options_t *rmw_options =
rcl_init_options_get_rmw_init_options(&init_options);

    // 设置静态代理IP和端口
    // Setup static agent IP and port
    RCCHECK(rmw_uros_options_set_udp_address(ROS_AGENT_IP, ROS_AGENT_PORT,
rmw_options));
```

Try to connect to the proxy. If the connection is successful, go to the next step. If the connection to the proxy is unsuccessful, you will always be in the connected state.

```
    while (1)
    {
        ESP_LOGI(TAG, "Connecting agent: %s:%s", ROS_AGENT_IP, ROS_AGENT_PORT);
        state_agent = rclc_support_init_with_options(&support, 0, NULL,
&init_options, &allocator);
        if (state_agent == ESP_OK)
        {
            ESP_LOGI(TAG, "Connected agent: %s:%s", ROS_AGENT_IP, ROS_AGENT_PORT);
            break;
        }
        vTaskDelay(pdMS_TO_TICKS(500));
    }
```

Create the node "servo_subscriber", in which ROS_NAMESPACE is empty by default and can be modified in the IDF configuration tool according to actual conditions.

```
    rcl_node_t node;
    RCCHECK(rclc_node_init_default(&node, "servo_subscriber", ROS_NAMESPACE,
&support));
```

To create subscribers "servo_s1" and "servo_s2", you need to specify the ROS topic information as std_msgs/msg/Int32 type.

```
    RCCHECK(rclc_subscription_init_default(
        &subscriber_s1,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "servo_s1"));
    RCCHECK(rclc_subscription_init_default(
        &subscriber_s2,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "servo_s2"));
```

Add subscribers to the executor, where the handle_num parameter of the executor represents the number added to the executor.

```
    rclc_executor_t executor;
    int handle_num = 2;
    RCCHECK(rclc_executor_init(&executor, &support.context, handle_num,
&allocator));

    RCCHECK(rclc_executor_add_subscription(&executor, &subscriber_s1, &msg_s1,
&servo_s1_callback, ON_NEW_DATA));
    RCCHECK(rclc_executor_add_subscription(&executor, &subscriber_s2, &msg_s2,
&servo_s2_callback, ON_NEW_DATA));
```

When the microros subscriber receives the topic data, the servo S1 rotation angle is controlled in the servo_s1_callback callback function, and the servo S2 rotation angle is controlled in the servo_s2_callback callback function. The angle control range of servo S1 is -90~90, and the angle control range of servo S2 is -90~20.

```
void servo_s1_callback(const void * msgin)
{
    const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msgin;
    int8_t angle = (msg->data) & 0xFF;
    printf("Servo S1: %d\n",  angle);
    Servo_Set_Angle(SERVO_ID_S1, angle);
}
void servo_s2_callback(const void * msgin)
{
    const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msgin;
    int8_t angle = (msg->data) & 0xFF;
    printf("Servo S2: %d\n",  angle);
    Servo_Set_Angle(SERVO_ID_S2, angle);
}
```

Note: In order to adapt to the servo gimbal, the control angle of the gimbal has been restricted. If you want to cancel the restriction, please modify the content of the servo.h file to configure the servo angle.

```
#define SERVO_S1_DEF_ANGLE (0)   // Default angle of S1
#define SERVO_S1_MIN_ANGLE (-90) // Limit the minimum angle of S1
#define SERVO_S1_MAX_ANGLE (90)  // Limit the maximum angle of S1

#define SERVO_S2_DEF_ANGLE (-60) // Default angle of S2
#define SERVO_S2_MIN_ANGLE (-90) // Limit the minimum angle of S2
#define SERVO_S2_MAX_ANGLE (20)  // Limit the maximum angle of S2
```

Call rclc_executor_spin_some in the loop to make microros work normally.

```
    while (1)
    {
        rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));
        usleep(1000);
    }
```

4. 4. Compile, download and flash firmware

Use a Type-C data cable to connect the virtual machine/computer and the microROS control board. If the system pops up, choose to connect to the virtual machine.

Activate the ESP-IDF development environment. Note that every time you open a new terminal, you need to activate the ESP-IDF development environment before compiling the firmware.

```
source ~/esp/esp-idf/export.sh
```

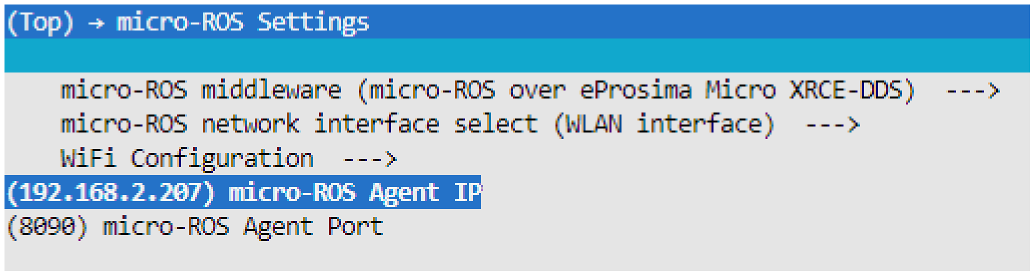
Enter the project directory

```
cd ~/esp/Samples/microros_samples/servo_subscriber
```

Open the ESP-IDF configuration tool.

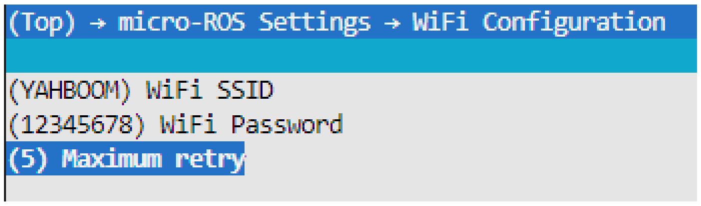
```
idf.py menuconfig
```

Open micro-ROS Settings, fill in the IP address of the agent host in micro-ROS Agent IP, and fill in the port number of the agent host in micro-ROS Agent Port.



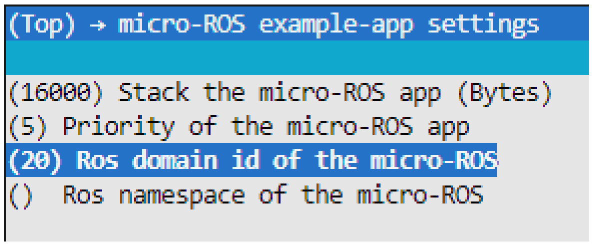
```
(Top) → micro-ROS Settings

    micro-ROS middleware (micro-ROS over eProsima Micro XRCE-DDS)  --->
    micro-ROS network interface select (WLAN interface)  --->
    WiFi Configuration  --->
(192.168.2.207) micro-ROS Agent IP
(8090) micro-ROS Agent Port
```

Open micro-ROS Settings->WiFi Configuration in sequence, and fill in your own WiFi name and password in the WiFi SSID and WiFi Password fields.



```
(Top) → micro-ROS Settings → WiFi Configuration

(YAHBOOM) WiFi SSID
(12345678) WiFi Password
(5) Maximum retry
```

Open the micro-ROS example-app settings. The Ros domain id of the micro-ROS defaults to 20. If multiple users are using it at the same time in the LAN, the parameters can be modified to avoid conflicts. Ros namespace of the micro-ROS is empty by default and does not need to be modified under normal circumstances. If non-empty characters (within 10 characters) are modified, the namespace parameter will be added before the node and topic.



```
(Top) → micro-ROS example-app settings

(16000) Stack the micro-ROS app (Bytes)
(5) Priority of the micro-ROS app
(20) Ros domain id of the micro-ROS
()  Ros namespace of the micro-ROS
```

After modification, press S to save, and then press Q to exit the configuration tool.

Compile, burn, and open the serial port simulator.

```
idf.py build flash monitor
```

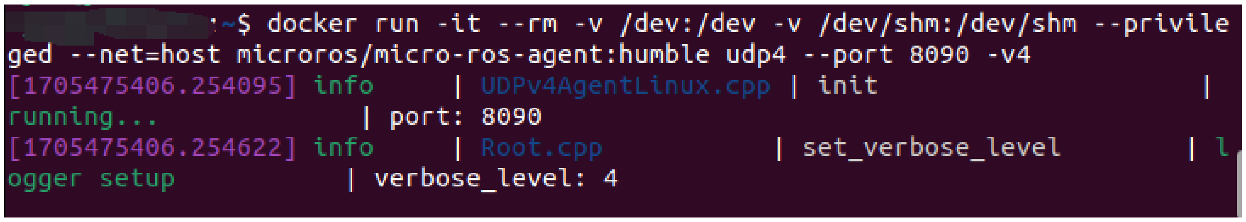
If you need to exit the serial port simulator, press **Ctrl+]**.

5. Experimental results

After powering on, ESP32 tries to connect to the WiFi hotspot, and then tries to connect to the proxy IP and port.

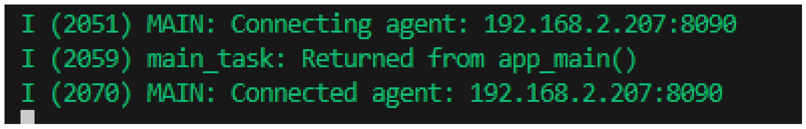
If the agent is not turned on in the virtual machine/computer terminal, please enter the following command to turn on the agent. If the agent is already started, there is no need to start the agent again.

```
docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --net=host
microros/micro-ros-agent:humble udp4 --port 8090 -v4
```



```
[redacted]:~$ docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privile
ged --net=host microros/micro-ros-agent:humble udp4 --port 8090 -v4
[1705475406.254095] info     | UDPv4AgentLinux.cpp | init                     |
running...             | port: 8090
[1705475406.254622] info     | Root.cpp            | set_verbose_level        | l
ogger setup            | verbose_level: 4
```

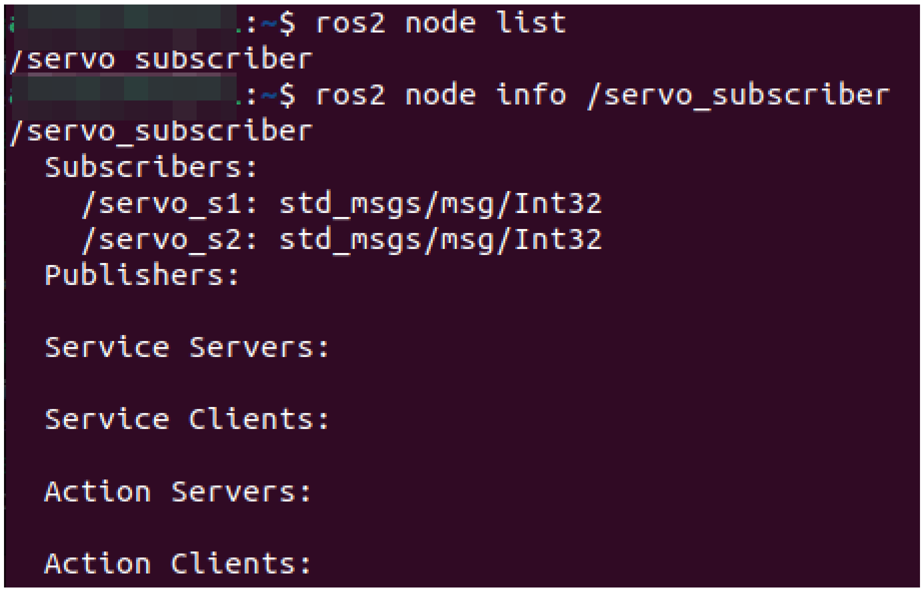
After the connection is successful, a node and two subscribers are created.



```
I (2051) MAIN: Connecting agent: 192.168.2.207:8090
I (2059) main_task: Returned from app_main()
I (2070) MAIN: Connected agent: 192.168.2.207:8090
```

At this time, you can open another terminal on the virtual machine/computer and view the /servo_subscriber node.

```
ros2 node list
ros2 node info /servo_subscriber
```



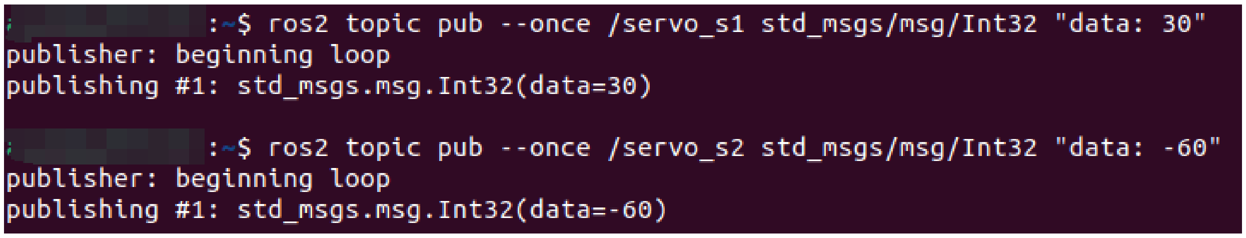
```
[redacted]:~$ ros2 node list
/servo_subscriber
[redacted]:~$ ros2 node info /servo_subscriber
/servo_subscriber
  Subscribers:
    /servo_s1: std_msgs/msg/Int32
    /servo_s2: std_msgs/msg/Int32
  Publishers:

  Service Servers:

  Service Clients:

  Action Servers:

  Action Clients:
```

Publish data to the /servo_s1 topic and control the servo S1 to rotate to 30 degrees.

```
ros2 topic pub --once /servo_s1 std_msgs/msg/Int32 "data: 30"
```

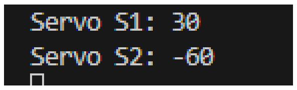
Publish data to the /servo_s2 topic and control the servo S2 to rotate to -60 degrees.

```
ros2 topic pub --once /servo_s2 std_msgs/msg/Int32 "data: -60"
```



```
[redacted]:~$ ros2 topic pub --once /servo_s1 std_msgs/msg/Int32 "data: 30"
publisher: beginning loop
publishing #1: std_msgs.msg.Int32(data=30)

[redacted]:~$ ros2 topic pub --once /servo_s2 std_msgs/msg/Int32 "data: -60"
publisher: beginning loop
publishing #1: std_msgs.msg.Int32(data=-60)
```

You can see the servo angle information printed out on the serial port simulator, indicating that the subscription is successful.



```
Servo S1: 30
Servo S2: -60
```