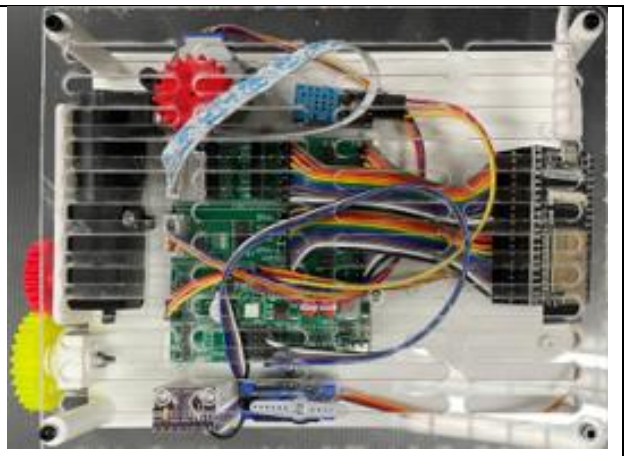
Tutorial 4 ROS Programming – Actuators

Objectives

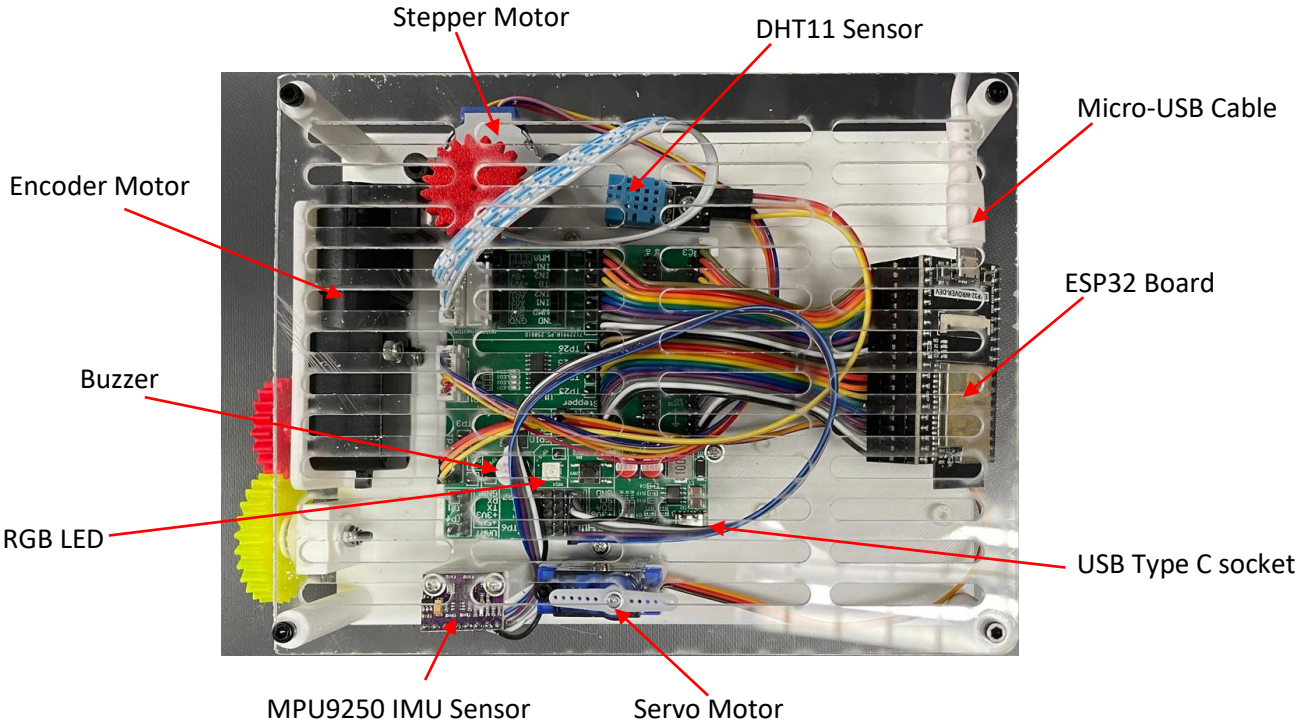
1. Learn to develop an Arduino program to control various actuators (e.g., LED, buzzer, servo motor, stepper motor, and DC encoder motor) using an ESP32 microcontroller.
2. Learn to integrate these actuators with ROS2 (Jazzy) through the Micro-ROS agent, providing control interfaces for the device's functions by subscribing to ROS2 topics and publishing the device's real-time status to ROS2 topics.
3. Understand how to interface hardware with ROS2, implement actuator control, and publish sensor data, fostering skills in embedded systems programming and ROS2 communication for robotics applications.

1 Introduction to the hardware items

1. A list of hardware items

1	ESP32 sensors and actuators kit	
2	A micro-USB Type-A cable	
3	A USB Type-C cable (If the PD Charger provides a Type-C socket)	
4	A PD Charger	 or 

2. Major components of the ESP32 sensors and actuator kit



3. Control circuit block diagram for buzzer

   A digital signal pin can control the buzzer output. ESP32 sets the digital signal pin high to enable the buzzer to output an audio tone. Set the digital signal pin low to stop the buzzer output.



4. Control circuit block diagram for the WS2812B LED

   The WS2812B LED uses a single-wire protocol for data transmission. ESP32 sends a stream of data bits (24 bits) to control the colour and brightness of the LED.



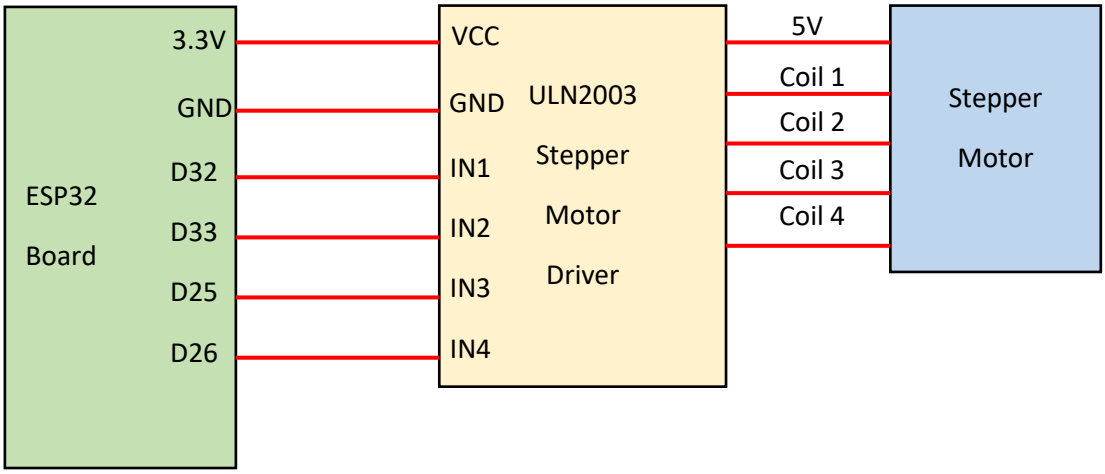
Composition of 24bit data:

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

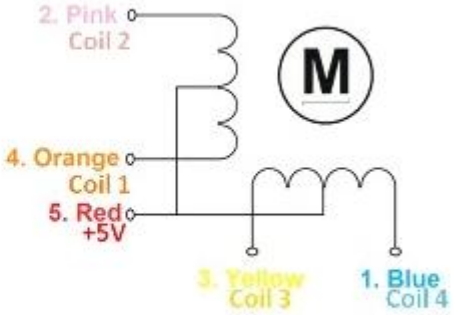
Note: Follow the order of GRB to sent data and the high bit sent at first.

5. Control circuit block diagram for stepper motor

   A 5V 5-wire stepper motor is driven by a stepper motor driver (e.g., ULN2003). The stepper motor driver delivers enough power to maintain the position of the stepper motor. ESP32 sends a sequence of signals (IN1, IN2, IN3, IN4) to control the position and moving speed of the stepper motor.

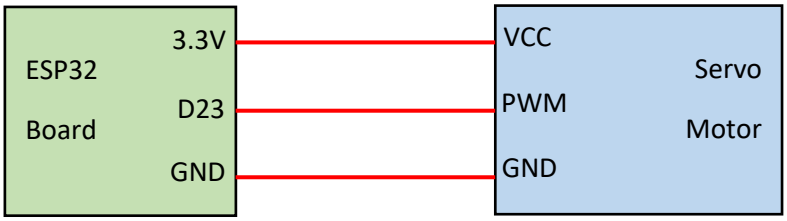






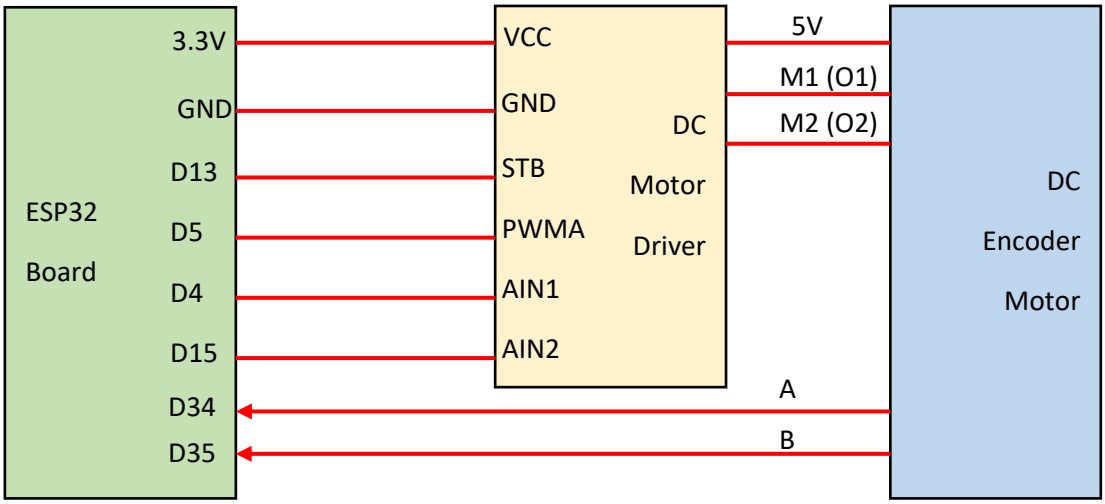
6. Control circuit block diagram for a servo motor

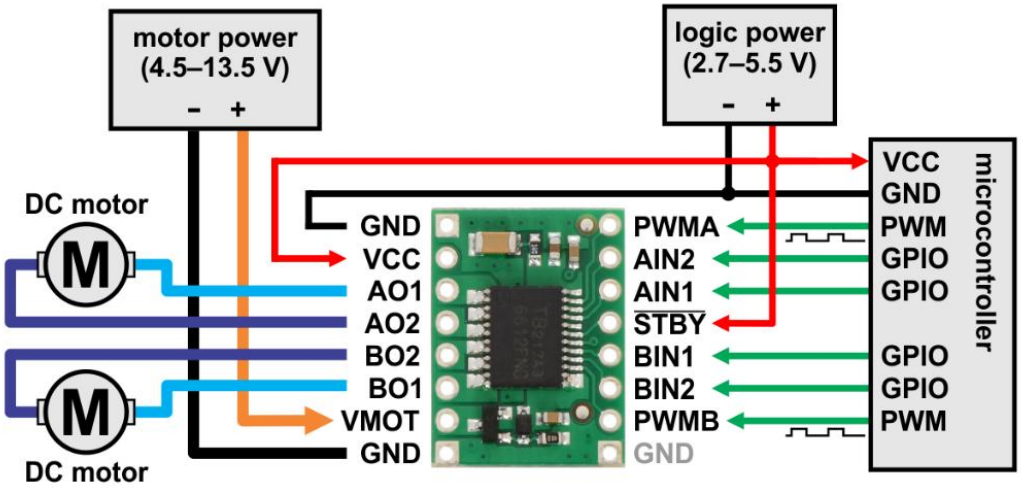
   The servo rotation angle can be controlled by a PWM signal. ESP32 outputs a PWM signal from the digital output pin 23 to control the angle of the servo motor.



7. Control circuit block diagram for a DC encoder motor

   The ESP32 controls the DC encoder motor using a motor driver (for example, TB6612). The motor driver receives control signals from the ESP32 (e.g., PWM, IN1, IN2) and then generates the necessary M1 (O1) and M2 (O2) power signals to drive the DC motor. M1 and M2 outputs can be adjusted to control the rotation direction and speed of the DC motor. The DC motor includes an A-B encoder. The encoder signals are returned to the ESP32, which measures speed and direction.





enter				Output		
IN1	IN2	PWM	STBY	O1	O2	Mode status
H	H	H/L	H	L	L	brake
L	H	H	H	L	H	Reverse
L	H	L	H	L	L	brake
H	L	H	H	H	L	Forward
H	L	L	H	L	L	brake
L	L	H	H	OFF		Stop
H/L	H/L	H/L	L	OFF		Standby

2 Control the buzzer and the WS2812B LED through a program

What is WS2812B?

The WS2812B is an individually addressable RGB LED strip manufactured by WorldSemi, commonly used for creating vibrant, programmable lighting effects. Each WS2812B LED is a 5050 RGB LED with an integrated WS2811 controller chip, allowing individual control of red, green, and blue colors (256 levels each, yielding 16.7 million colors). The LEDs are mounted on a flexible PCB strip, typically with 30, 60, or 144 LEDs per meter.

1. Launch Arduino IDE, install the "Adafruit_NeoPixel" library from Tools > Manage Libraries.
2. Create a new sketch by File>New Sketch
3. Include the necessary libraries and define the pin number of the buzzer by adding this code to the sketch:

```
#include <micro_ros_arduino.h>


#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>


#include <std_msgs/msg/int32.h>
#include <std_msgs/msg/color_rgba.h>


#include <Adafruit_NeoPixel.h>


#define BUZZER_PIN 2
#define LED_PIN 14
#define LED_COUNT 1


Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);


unsigned long tBuzzerStarted;
bool bBuzzerStarted = false;
```

a) BUZZER_PIN: The GPIO pin connected to the buzzer.

b) LED_PIN: The GPIO pin connected to the LED.
c) LED_COUNT: The number of LEDs to be controlled. Several WS2812B LEDs can be connected in series and controlled by one GPIO pin.
d) strip: A NeoPixel object for controlling the WS2812B LED.
e) tBuzzerStarted: Store the time when the buzzer starts beeping.
f) bBuzzerStarted: Store the buzzer beeping state. The program can stop the beeping if the desired beeping time is reached.

4. Implement macros for error checking in micro-ROS by adding these codes:

```
#define RCCHECK(fn) \
  { \
    rcl_ret_t temp_rc = fn; \
    if ((temp_rc != RCL_RET_OK)) { error_loop(); } \
  }
#define RCSOFTCHECK(fn) \
  { \
    rcl_ret_t temp_rc = fn; \
    if ((temp_rc != RCL_RET_OK)) {} \
  }
```

5. Define micro-ROS variables and implement the error_loop, subscription_callback_buzzer_beep, subscription_callback_led_rgb functions with these codes:

```
rcl_subscription_t subscriber_led_rgb;
std_msgs__msg__ColorRGBA msg_led_rgb;
rcl_subscription_t subscriber_buzzer_beep;
std_msgs__msg__Int32 msg_buzzer_beep;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;

void error_loop() {
```

```
  while (1) {
    digitalWrite(BUZZER_PIN, !digitalRead(BUZZER_PIN));
    delay(100);
  }
}

void subscription_callback_buzzer_beep(const void *msgin) {
  const std_msgs__msg__Int32 *msg = (const std_msgs__msg__Int32 *)msgin;

  if (msg->data == 1) {
    digitalWrite(BUZZER_PIN, HIGH);
    tBuzzerStarted = millis();
    bBuzzerStarted = true;
  }
}

void subscription_callback_led_rgb(const void *msgin) {
  const std_msgs__msg__ColorRGBA *msg =
    (const std_msgs__msg__ColorRGBA *)msgin;

  digitalWrite(BUZZER_PIN, HIGH);
  tBuzzerStarted = millis();
  bBuzzerStarted = true;

  // Set LED to current color
  uint32_t new_color = static_cast<uint32_t>(msg->r) << 16
                     | (static_cast<uint32_t>(msg->g) << 8)
                     | (static_cast<uint32_t>(msg->b));
  strip.setPixelColor(0, new_color);
  strip.show();
}
```

a) The error_loop function will control the buzzer to generate an error tone if a critical error is detected.

b) The subscription_callback_buzzer_beep function will be called if new data is received from subscribed topics.
c) The subscription_callback_led_rgb function will be called if new data is received from subscribed topics.

6. Replace the setup function with the following code:

```
void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);

  set_microros_transports();
  delay(2000);

  allocator = rcl_get_default_allocator();

  //create init_options
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // create node
  RCCHECK(rclc_node_init_default(
    &node, "hku_dase_micro_ros_node", "", &support));

  // create subscriber
  RCCHECK(rclc_subscription_init_default(
    &subscriber_led_rgb,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, ColorRGBA),
    "led_rgb"));

  // create subscriber
  RCCHECK(rclc_subscription_init_default(
    &subscriber_buzzer_beep,
    &node,
```

```
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "buzzer_beep"));


  // create executor
  // total number of handles = #subscriptions + #timers
  RCCHECK(rclc_executor_init(
    &executor, &support.context, 2, &allocator));
  RCCHECK(rclc_executor_add_subscription(
    &executor, &subscriber_buzzer_beep, &msg_buzzer_beep,
    &subscription_callback_buzzer_beep,
    ON_NEW_DATA));
  RCCHECK(rclc_executor_add_subscription(
    &executor, &subscriber_led_rgb, &msg_led_rgb,
    &subscription_callback_led_rgb,
    ON_NEW_DATA));
}
```

a) set_microros_transports(): Configures communication by default.
b) rclc_support_init, rclc_node_init_default: Set up the ROS client and node.
c) rclc_subscription_init_default: Setup subscription to a topic.
d) rclc_executor_init, rclc_executor_add_subscription: Manage the execution functions and features.

7. Replace the loop function with this code.

```
void loop() {
  // delay(100);
  RCCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(10)));


  // Turn off the Buzzer if it is turned on for more than 500ms
  if ((bBuzzerStarted == true) && ((millis() - tBuzzerStarted) > 500)) {
    digitalWrite(BUZZER_PIN, LOW);
    bBuzzerStarted = false;
 }
}
```

8. Select the board by Tools>Board>esp32>ESP32 Dev Module
9. Select the port as "/dev/ttyUSB0" from Tools>Port
10. Verify and compile the sketch by Sketch>Verify/Compile
11. Open a new terminal:

```
$ ls /dev/tty*

# (After running "ls /dev/tty*", verify if there is an item named
"/dev/ttyUSB0")

$ sudo chmod 666 /dev/ttyUSB0

$ sudo usermod -aG dialout $USER

$ newgrp docker

$ groups

# (verify if there is "docker" in the output)
```

12. Upload the sketch by Sketch>Upload
13. After uploading is done, open a new terminal and turn on the micro-ROS-agent:

```
$ docker run -it --rm -v /dev:/dev --privileged

--net=host microros/micro-ros-agent:jazzy serial --dev /dev/ttyUSB0 -v6
```

14. Check the topics available by opening a new terminal and running:

```
$ source /opt/ros/jazzy/setup.bash

$ ros2 topic list
```

Topics "/buzzer_beep" and "/led_rgb" will be listed.

15. Send a message to the kit and let it generate a beep by running:

```
$ ros2 topic pub --once /buzzer_beep std_msgs/msg/Int32 "{data: 1}"
```

a) The buzzer will generate a beep.

16. Send a message to the kit and let it change the color of the LED by running one of the following commands:

```
# set LED display color to RED

$ ros2 topic pub  --once /led_rgb std_msgs/msg/ColorRGBA

"{r: 255.0, g: 0.0, b: 0.0, a: 1.0}"


# set LED display color to GREEN

$ ros2 topic pub  --once /led_rgb std_msgs/msg/ColorRGBA

"{r: 0.0, g: 255.0, b: 0.0, a: 1.0}"
```

```
# set LED display color to BLUE
$ ros2 topic pub  --once /led_rgb std_msgs/msg/ColorRGBA
"{r: 0.0, g: 0.0, b: 255.0, a: 1.0}"

# set LED display color to WHITE
$ ros2 topic pub  --once /led_rgb std_msgs/msg/ColorRGBA
"{r: 255.0, g: 255.0, b: 255.0, a: 1.0}"

# set LED OFF (no color output)
$ ros2 topic pub  --once /led_rgb std_msgs/msg/ColorRGBA
"{r: 0.0, g: 0.0, b: 0.0, a: 1.0}"
```

3 Control the servo motor through a program

What is a servo motor?

A servo motor is an electric motor and actuator system that allows for exact control of angular or linear position, velocity, and acceleration. It accomplishes this using a closed-loop control system in which a sensor provides feedback on the motor's current state, which is compared to a desired state by a controller, allowing for continuous, high-accuracy adjustments to reach and keep the target motion. A motor, a position sensor (e.g., an encoder), and a controller are all necessary components.

1. In Arduino IDE, install the “ESP32Servo” library from Tools > Manage Libraries.
2. Create a new sketch by File>New Sketch
3. Include the necessary libraries and define the pin number of the servo motor by adding these codes into the sketch:

```
#include <micro_ros_arduino.h>

#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <std_msgs/msg/int32.h>
```

```
#include <ESP32Servo.h>


#define BUZZER_PIN 2
#define SERVO_PIN 23


Servo servo;


unsigned long tBuzzerStarted;
bool bBuzzerStarted = false;
```

a) SERVO_PIN: the pin number of the servo motor's PWM is 23.

4. Implement macros for error checking in micro-ROS by adding these codes:

```
#define RCCHECK(fn) \
  { \
    rcl_ret_t temp_rc = fn; \
    if ((temp_rc != RCL_RET_OK)) { error_loop(); } \
  }
#define RCSOFTCHECK(fn) \
  { \
    rcl_ret_t temp_rc = fn; \
    if ((temp_rc != RCL_RET_OK)) {} \
  }
```

5. Define micro-ROS variables and implement error_loop and subscription_callback_servo_pos functions with these codes:

```
rcl_subscription_t subscriber_servo_pos;
std_msgs__msg__Int32 msg_servo_pos;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;


void error_loop() {
```

```
  while (1) {
    digitalWrite(BUZZER_PIN, !digitalRead(BUZZER_PIN));
    delay(100);
  }
}


void subscription_callback_servo_pos(const void *msgin) {
  const std_msgs__msg__Int32 *msg = (const std_msgs__msg__Int32 *)msgin;


  servo.write(msg->data);


  digitalWrite(BUZZER_PIN, HIGH);
  tBuzzerStarted = millis();
  bBuzzerStarted = true;
}
```

a) The error_loop function will control the buzzer to generate an error tone if a critical error is detected.
b) The subscription_callback_servo_pos function will be called if new data is received from subscribed topics.
c) The servo.write function will rotate the servo to the specified angle.

6. Replace the setup function with the following code:

```
void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);


  servo.attach(SERVO_PIN);


  set_microros_transports();
  delay(2000);


  allocator = rcl_get_default_allocator();


  //create init_options
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));
```

```
  // create node
  RCCHECK(rclc_node_init_default(
    &node, "hku_dase_micro_ros_node", "", &support));

  // create subscriber
  RCCHECK(rclc_subscription_init_default(
    &subscriber_servo_pos,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "servo_pos"));

  // create executor
  RCCHECK(rclc_executor_init(
    &executor, &support.context, 1, &allocator));
  RCCHECK(rclc_executor_add_subscription(
    &executor, &subscriber_servo_pos, &msg_servo_pos,
    &subscription_callback_servo_pos,
    ON_NEW_DATA));
}
```

a) set_microros_transports(): Configures communication by default.
b) rclc_support_init, rclc_node_init_default: Set up the ROS client and node.
c) rclc_subscription_init_default: Setup subscription to a topic.
d) rclc_executor_init, rclc_executor_add_subscription: Manage the execution functions and features.

7. Replace the loop function with this code.

```
void loop() {
  // delay(100);
  RCCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(10)));

  // Turn off the Buzzer if it is turned on for more than 500ms
  if ((bBuzzerStarted == true) && ((millis() - tBuzzerStarted) > 500)) {
    digitalWrite(BUZZER_PIN, LOW);
    bBuzzerStarted = false;
```

```
  }
}
```

8. Select the board by Tools>Board>esp32>ESP32 Dev Module
9. Select the port as "/dev/ttyUSB0" from Tools>Port
10. Verify and compile the sketch by Sketch>Verify/Compile
11. Open a new terminal:

```
$ ls /dev/tty*

# (After running "ls /dev/tty*", verify if there is an item named
"/dev/ttyUSB0")

$ sudo chmod 666 /dev/ttyUSB0

$ sudo usermod -aG dialout $USER

$ newgrp docker

$ groups

# (verify if there is "docker" in the output)
```

12. Upload the sketch by Sketch>Upload
13. After uploading is done, open a new terminal and turn on the micro-ROS-agent:

```
$ docker run -it --rm -v /dev:/dev --privileged

--net=host microros/micro-ros-agent:jazzy serial --dev /dev/ttyUSB0 -v6
```

14. Send a message to the kit, and the servo motor should rotate to the specified angle. The valid range of pos is from 0 to 180.

```
# set servo pos to 0 degree

$ ros2 topic pub --once /servo_pos std_msgs/msg/Int32 "{data: 0}"


# set servo pos to 90 degree

$ ros2 topic pub --once /servo_pos std_msgs/msg/Int32 "{data: 90}"


# set servo pos to 180 degree

$ ros2 topic pub --once /servo_pos std_msgs/msg/Int32 "{data: 180}"
```

4 Control the stepper motor 28BYJ-48 through a program

What is a stepper motor?

A stepper motor is a type of rotary or linear actuator that allows precise control of angular or linear position, velocity, and acceleration. It consists of a motor coupled with a feedback mechanism (typically a potentiometer or encoder) and a control circuit, enabling accurate positioning within a specific range.

1. In Arduino IDE, install the "Unistep2" library from Tools > Manage Libraries.
2. Create a new sketch by File>New Sketch
3. Include the necessary libraries and define the pin number of the stepper motor by adding these codes into the sketch:

```
#include <micro_ros_arduino.h>


#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>


#include <std_msgs/msg/int32.h>


#include <Unistep2.h>
#include <vector> // For storing task handles


#define BUZZER_PIN 2
#define IN1 32
#define IN2 33
#define IN3 25
#define IN4 26
const int stepsPerRevolution = 4096;


Unistep2 Stepper(IN1, IN2, IN3, IN4, stepsPerRevolution, 1000);


unsigned long tBuzzerStarted;
bool bBuzzerStarted = false;
```

```
TaskHandle_t Task1; // Task handle for CPU Core 1

int32_t stepsReq = 0;
```

a) IN1, IN2, IN3, IN4: pin number of stepper motor's four phases
b) stepsPerRevolution: The number of steps per a full revolution.
c) Stepper: A Unistep2 object that controls a stepper motor. It may not perform properly if the program has blocking code. For example, the micro-ROS function "rclc_executor_spin_some" is a blocking code that waits until the task is completed or the specified timeout is reached.
d) stepsReq: a variable stores the number of steps to be moved.

4. Implement macros for error checking in micro-ROS by adding these codes:

```
#define RCCHECK(fn) \
  { \
    rcl_ret_t temp_rc = fn; \
    if ((temp_rc != RCL_RET_OK)) { error_loop(); } \
  }
#define RCSOFTCHECK(fn) \
  { \
    rcl_ret_t temp_rc = fn; \
    if ((temp_rc != RCL_RET_OK)) {} \
  }
```

5. Define micro-ROS variables and implement error_loop, subscription_callback_stepper_steps, TaskStepper, and createStepperTask functions with these codes:

```
rcl_subscription_t subscriber_stepper_steps;

std_msgs__msg__Int32 msg_stepper_steps;

rclc_executor_t executor;

rclc_support_t support;

rcl_allocator_t allocator;

rcl_node_t node;

rcl_timer_t timer;


void error_loop() {
  while (1) {
    digitalWrite(BUZZER_PIN, !digitalRead(BUZZER_PIN));
    delay(100);
```

```
  }
}

void subscription_callback_stepper_steps(const void *msgin) {
  const std_msgs__msg__Int32 *msg = (const std_msgs__msg__Int32 *)msgin;

  digitalWrite(BUZZER_PIN, HIGH);
  tBuzzerStarted = millis();
  bBuzzerStarted = true;

  stepsReq = msg->data;
}

void TaskStepper(void * pvParameters) {
  for (;;) {
    if (stepsReq != 0) {
      Stepper.move(stepsReq);
      stepsReq = 0;
    }
    Stepper.run();

    vTaskDelay(1 / portTICK_PERIOD_MS); // 1ms delay
  }
}

void createStepperTask() {
  xTaskCreatePinnedToCore(
      TaskStepper,   // Function to implement the task
      "Task1",       // Name of the task
      10000,         // Stack size in words
      NULL,          // Task input parameter
      1,             // Priority of the task
      &Task1,        // Task handle
      1);            // Pin to Core 1
```

```
}
```

a) The error_loop function will control the buzzer to generate an error tone if a critical error is detected.
b) The subscription_callback_servo_pos function will be called if new data is received from subscribed topics.
c) The servo.write function will rotate the servo to the specified angle.

6. Replace the setup function with these codes:

```
void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);

  // Create stepper motor task and
  // run on CPU core 1
  createStepperTask();

  set_microros_transports();
  delay(2000);

  allocator = rcl_get_default_allocator();

  //create init_options
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // create node
  RCCHECK(rclc_node_init_default(
    &node, "hku_dase_micro_ros_node", "", &support));

  // create subscriber
  RCCHECK(rclc_subscription_init_default(
    &subscriber_stepper_steps,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "stepper_steps"));
```

```
  // create executor
  RCCHECK(rclc_executor_init(
    &executor, &support.context, 1, &allocator));
  RCCHECK(rclc_executor_add_subscription(
    &executor, &subscriber_stepper_steps, &msg_stepper_steps,
    &subscription_callback_stepper_steps,
    ON_NEW_DATA));
}
```

a) set_microros_transports(): Configures communication by default.
b) rclc_support_init, rclc_node_init_default: Set up the ROS client and node.
c) rclc_subscription_init_default: Setup subscription with a topic.
d) rclc_executor_init, rclc_executor_add_subscription: Manages the execution functions and features.

7. Replace the loop function with these codes.

```
void loop() {
  // delay(100);
  RCCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(10)));

  // Turn off the Buzzer if it is turned on for more than 500ms
  if ((bBuzzerStarted == true) && ((millis() - tBuzzerStarted) > 500)) {
    digitalWrite(BUZZER_PIN, LOW);
    bBuzzerStarted = false;
  }
}
```

8. Select the board by Tools>Board>esp32>ESP32 Dev Module
9. Select the port as "/dev/ttyUSB0" from Tools>Port
10. Verify and compile the sketch by Sketch>Verify/Compile
11. Open a new terminal:

```
$ ls /dev/tty*
# (After running "ls /dev/tty*", verify if there is an item named
"/dev/ttyUSB0")
$ sudo chmod 666 /dev/ttyUSB0
$ sudo usermod -aG dialout $USER
$ newgrp docker
```

```
$ groups

# (verify if there is “docker” in the output)
```

12. Upload the sketch by Sketch>Upload
13. After uploading is done, open a new terminal and turn on the micro-ROS-agent:

```
$ docker run -it --rm -v /dev:/dev --privileged

--net=host microros/micro-ros-agent:jazzy serial --dev /dev/ttyUSB0 -v6
```

14. Send a message to the kit, and the stepper motor should move the specific number of steps.

```
# stepper motor moves 4096 steps forward

$ ros2 topic pub --once /stepper_steps std_msgs/msg/Int32 "{data: 4096}"


# stepper motor moves 4096 steps backward

$ ros2 topic pub --once /stepper_steps std_msgs/msg/Int32 "{data: -4096}"
```

5 Control the DC encoder motor through a program

What is an encoder motor?

An encoder motor is an electric motor (typically DC or brushless) equipped with an encoder, a sensor that provides feedback on the motor's position, speed, or direction. The encoder converts mechanical motion into digital signals, enabling precise control in closed-loop systems.

1. In Arduino IDE, create a new sketch by File>New Sketch
2. Include the necessary libraries and define the pin number of dc encoder motor with adding these codes into the sketch:

```
#include <micro_ros_arduino.h>


#include <stdio.h>

#include <rcl/rcl.h>

#include <rcl/error_handling.h>

#include <rclc/rclc.h>

#include <rclc/executor.h>
```

```
#include <std_msgs/msg/int32.h>


#define BUZZER_PIN 2
#define STB 13
#define PWMA 5
#define AIN1 4
#define AIN2 15
#define ENCODER_A 34
#define ENCODER_B 35


unsigned long tBuzzerStarted;
bool bBuzzerStarted = false;
volatile long encoderCount = 0;
volatile bool interruptTriggered = false;
```

a) STB: Standby pin for motor driver (enables/disables the driver) (pin 13).
b) PWMA: PWM pin for controlling motor speed (pin 5).
c) AIN1 and AIN2: Control motor direction (forward/backward) (pin 4 & 15).
d) ENCODER_A and ENCODER_B: Encoder channel A and B for tracking motor rotation (pin 34 & 35).
e) encoderCount: Tracks the encoder's position (incremented/decremented based on rotation direction).

3. Implement macros for error checking in micro-ROS by adding these codes:

```
#define RCCHECK(fn) \
  { \
    rcl_ret_t temp_rc = fn; \
    if ((temp_rc != RCL_RET_OK)) { error_loop(); } \
  }
#define RCSOFTCHECK(fn) \
  { \
    rcl_ret_t temp_rc = fn; \
    if ((temp_rc != RCL_RET_OK)) {} \
  }
```

4. Define micro-ROS variables and implement error_loop, encoderISR, timer_callback, and subscription_callback_motor_pwm functions with these codes:

```
rcl_publisher_t publisher_encoder_count;
std_msgs__msg__Int32 msg_encoder_count;
rcl_subscription_t subscriber_motor_pwm;
std_msgs__msg__Int32 msg_motor_pwm;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;

void error_loop() {
  while (1) {
    digitalWrite(BUZZER_PIN, !digitalRead(BUZZER_PIN));
    delay(100);
  }
}

void IRAM_ATTR encoderISR() {
  if (digitalRead(ENCODER_A) == digitalRead(ENCODER_B)) {
    encoderCount++;
  } else {
    encoderCount--;
  }
  interruptTriggered = true;
}

void timer_callback(rcl_timer_t * timer, int64_t last_call_time) {
  RCLC_UNUSED(last_call_time);
  if (timer != NULL) {
    msg_encoder_count.data = encoderCount;
```

```
    RCSOFTCHECK(rcl_publish(&publisher_encoder_count, &msg_encoder_count,
NULL));
  }
}

void subscription_callback_motor_pwm(const void *msgin) {
  const std_msgs__msg__Int32 *msg = (const std_msgs__msg__Int32 *)msgin;

  int32_t pwm;
  pwm = msg->data;
  if (pwm > 0) {
    // forward
    digitalWrite(AIN1, HIGH);
    digitalWrite(AIN2, LOW);
    pwm = (pwm > 255) ? 255 : pwm;
    analogWrite(PWMA, pwm);
  } else if (pwm < 0) {
    // backward
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, HIGH);
    pwm = (pwm < -255) ? 255 : -pwm;
    analogWrite(PWMA, pwm);
  } else {
    // stop
    analogWrite(PWMA, 0);
  }

  digitalWrite(BUZZER_PIN, HIGH);
  tBuzzerStarted = millis();
  bBuzzerStarted = true;
}
```

a) The “encoderISR” function is an interrupt service routine to keep track the value of the DC motor’s encoder.

b) The “timer_callback” function will be called in the specified time interval. It publishes the current value of the encoder count to the topic “/encoder_count”.
c) The “subscription_callback_motor_pwm” function will be called if new data is received from subscribed topics. It controls the DC encoder motor’s moving speed and direction.

5. Replace the setup function with these codes:

```
void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);

  // Initialize motor pins
  pinMode(STB, OUTPUT);
  pinMode(PWMA, OUTPUT);
  pinMode(AIN1, OUTPUT);
  pinMode(AIN2, OUTPUT);
  digitalWrite(STB, HIGH); // Enable motor driver
  digitalWrite(AIN1, LOW);
  digitalWrite(AIN2, LOW);
  analogWrite(PWMA, 0);

  // Initialize encoder pins
  pinMode(ENCODER_A, INPUT);
  pinMode(ENCODER_B, INPUT);

  // Attach interrupt
  attachInterrupt(digitalPinToInterrupt(ENCODER_A), encoderISR, CHANGE);

  set_microros_transports();
  delay(2000);

  allocator = rcl_get_default_allocator();

  //create init_options
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));
```

```
// create node
RCCHECK(rclc_node_init_default(
  &node, "hku_dase_micro_ros_node", "", &support));

// create timer
const unsigned int timer_timeout = 500;
RCCHECK(rclc_timer_init_default(
  &timer, &support,
  RCL_MS_TO_NS(timer_timeout),
  timer_callback));

// create publisher
RCCHECK(rclc_publisher_init(
  &publisher_encoder_count,
  &node,
  ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
  "/encoder_count",
  &rmw_qos_profile_sensor_data));

// create subscriber
RCCHECK(rclc_subscription_init_default(
  &subscriber_motor_pwm,
  &node,
  ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
  "motor_pwm"));

// create executor
// total number of handles = #subscriptions + #timers
RCCHECK(rclc_executor_init(
  &executor, &support.context, 2, &allocator));
RCCHECK(rclc_executor_add_timer(&executor, &timer));
RCCHECK(rclc_executor_add_subscription(
  &executor, &subscriber_motor_pwm,
```

```
    &msg_motor_pwm,
    &subscription_callback_motor_pwm,
    ON_NEW_DATA));
}
```

a) set_microros_transports: Configures communication by default.
b) rclc_support_init, rclc_node_init_default: Set up the ROS client and node.
c) rclc_publisher_init: Create publishers for /encoder_count topic.
d) rclc_timer_init_default: Sets up a timer to call timer_callback every 0.5 second.
e) rclc_executor_init, rclc_executor_add_timer, rclc_executor_add_subscription: Manage the executor functions.

6. Replace the loop function with this code.

```
void loop() {
  // delay(100);
  RCCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(10)));

  // Turn off the Buzzer if it is turned on for more than 500ms
  if ((bBuzzerStarted == true) && ((millis() - tBuzzerStarted) > 500)) {
    digitalWrite(BUZZER_PIN, LOW);
    bBuzzerStarted = false;
  }
}
```

7. Select the board by Tools>Board>esp32>ESP32 Dev Module
8. Select the port as "/dev/ttyUSB0" from Tools>Port
9. Verify and compile the sketch by Sketch>Verify/Compile
10. Open a new terminal:

```
$ ls /dev/tty*
# (After running "ls /dev/tty*", verify if there is an item named
"/dev/ttyUSB0")
$ sudo chmod 666 /dev/ttyUSB0
$ sudo usermod -aG dialout $USER
$ newgrp docker
$ groups
# (verify if there is "docker" in the output)
```

11. Upload the sketch by Sketch>Upload
12. After uploading is done, open a new terminal and turn on the micro-ROS-agent:

```
$ docker run -it --rm -v /dev:/dev --privileged
--net=host microros/micro-ros-agent:jazzy serial --dev /dev/ttyUSB0 -v6
```

13. Open a new terminal and run the following command to monitor the value of DC motor encoder count.

```
$ ros2 topic echo /encoder_count
```

14. Open a new terminal and send a message to the kit. The motor should move at the specific speed.

```
# forward with PMW value 200
$ ros2 topic pub --once /motor_pwm std_msgs/msg/Int32 "{data: 200}"


# backward with PMW value 150
$ ros2 topic pub --once /motor_pwm std_msgs/msg/Int32 "{data: -150}"


# stop
$ ros2 topic pub --once /motor_pwm std_msgs/msg/Int32 "{data: 0}"
```