{Learn, Create, Innovate};

ROS serial communication

Interfacing a microcontroller and ROS

Manchester Robotics

Table of contents

Micro-ROS: Introduction

Introduction

- Micro-ROS (µROS) is an extension of ROS 2 designed for microcontrollers.
- It enables embedded systems to communicate within ROS2 ecosystems.
- Built on ROS 2, DDS (Data Distribution Service), and FreeRTOS, Zephyr, NuttX, or bare-metal systems.
  - Lightweight: Optimised for resource-constrained microcontrollers.
  - Real-time support: Meets real-time requirements for embedded robotics.
  - Interoperability: Seamlessly integrates with ROS2.
  - Standardised communication: Uses XRCE-DDS for efficient data transmission.

micro-ROS | ROS 2 for microcontrollers

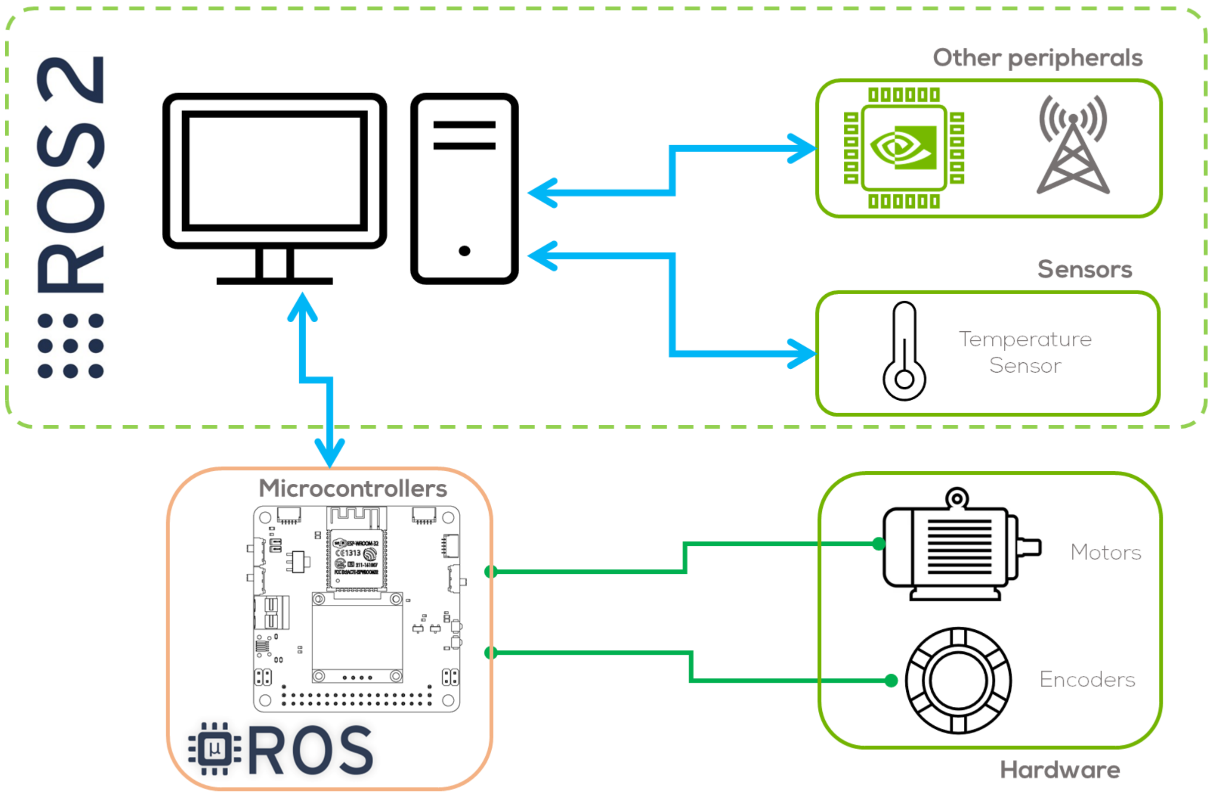
Micro-ROS: Introduction

Why Use Micro-ROS?

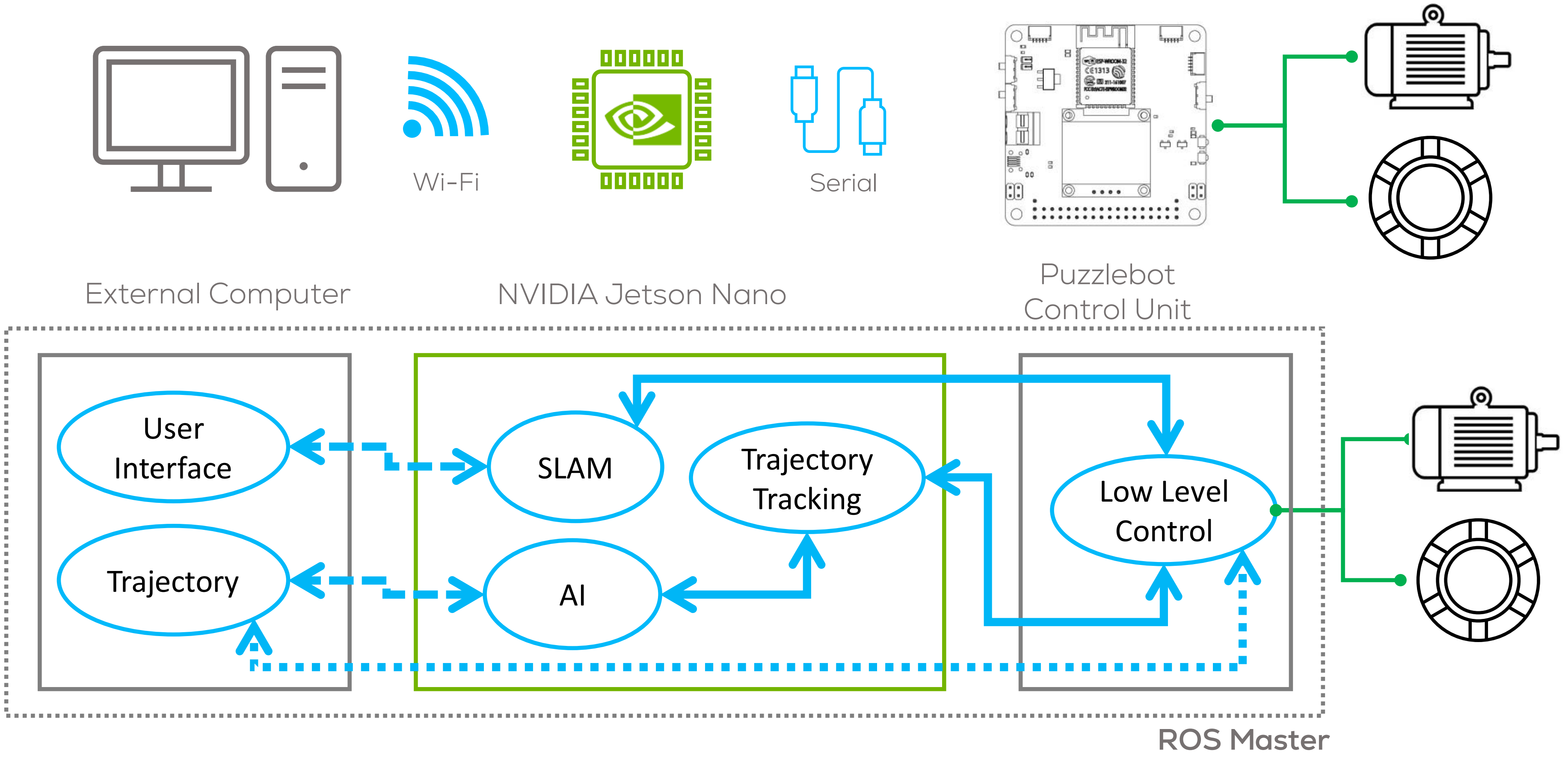
- **Bridges microcontrollers and ROS 2** for enhanced functionality.
- **Reduces power consumption** compared to full ROS 2 nodes.
- **Enables real-time control** of actuators and sensors.
- **Compatible with various embedded platforms**, including STM32, ESP32, and Raspberry Pi Pico.

Common Applications:

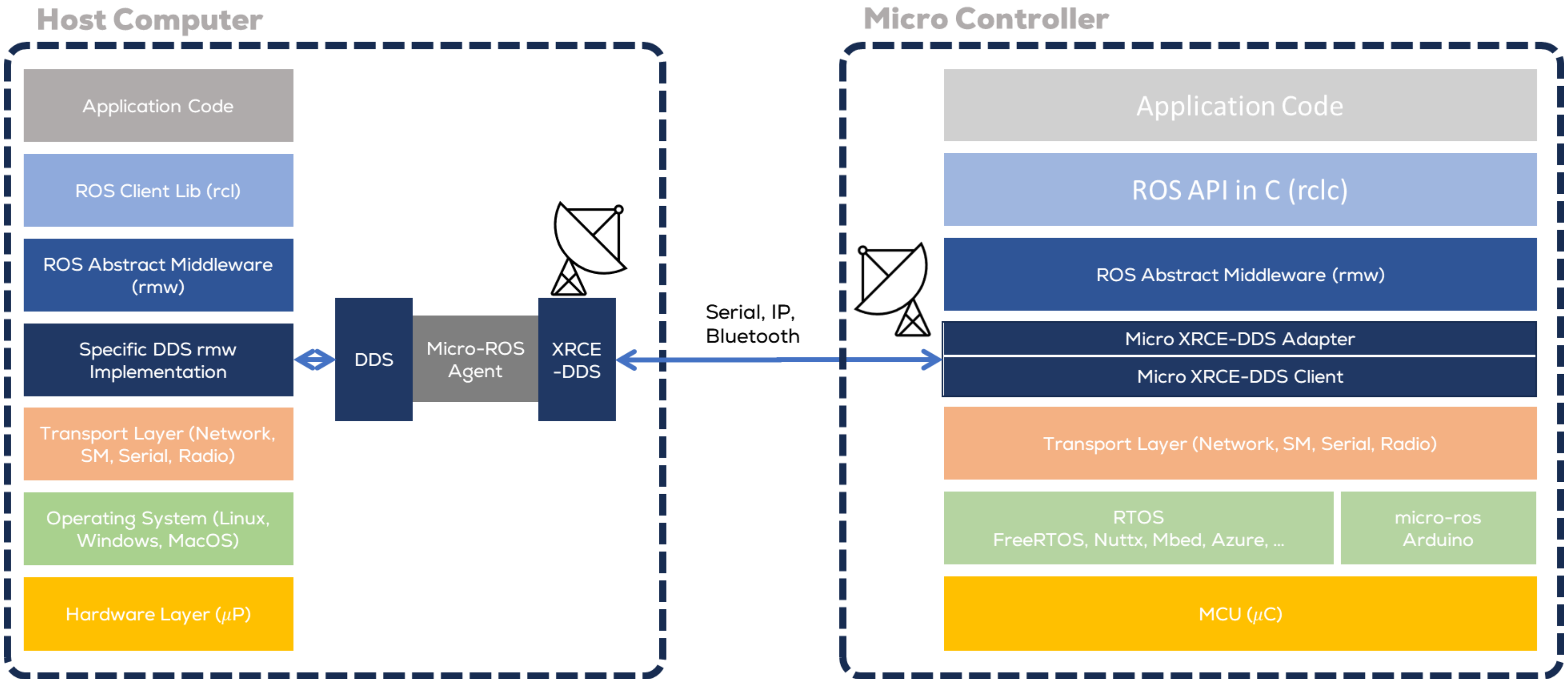
- Embedded robotic controllers.
- Autonomous vehicles and drones.
- IoT-based robotic systems.
- Industrial automation and smart sensors.



Micro-ROS: Introduction



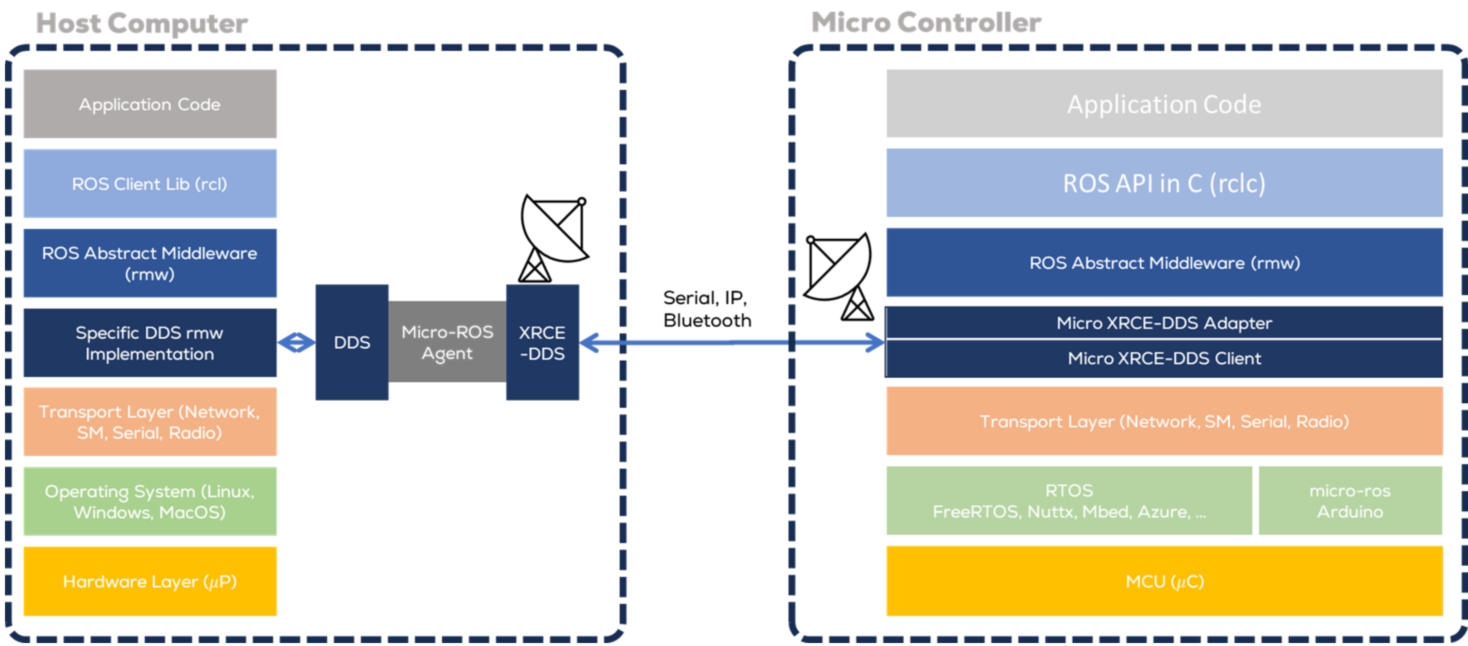
Micro-ROS: Architecture



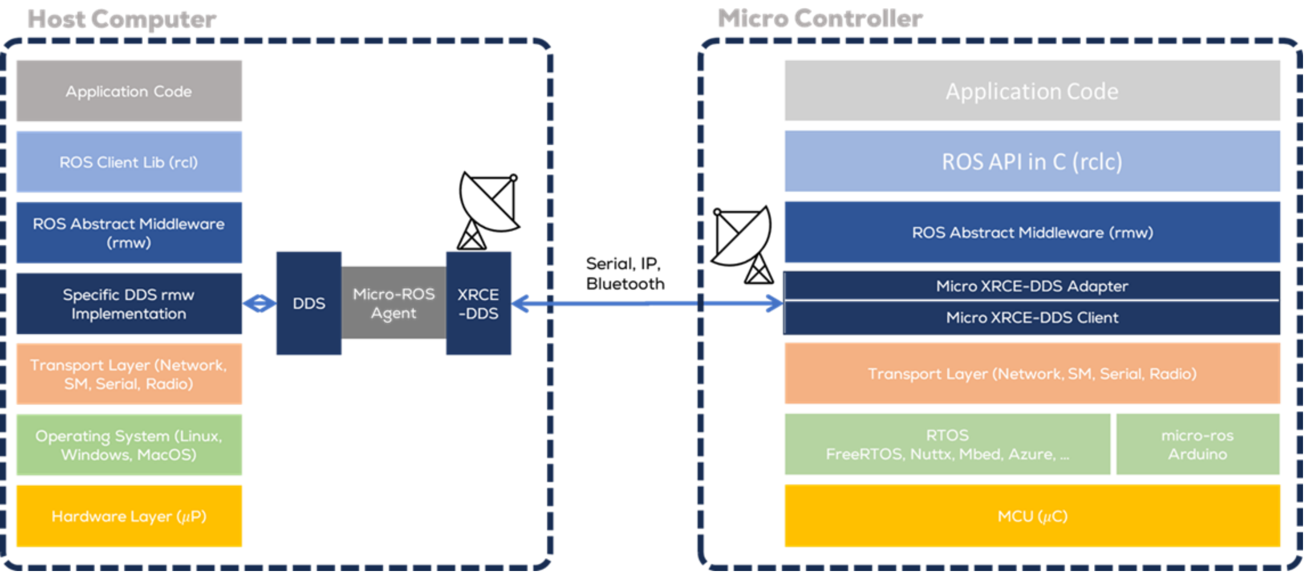
Micro-ROS: Architecture

ROS 2 Agent (Left Side)

- The **ROS 2 Agent** acts as a gateway between the **Micro-ROS Client** (running on a microcontroller) and the **full ROS 2 stack**.
- It runs on a standard operating system (e.g., Ubuntu) and handles **communication with Micro-ROS clients**.
- Communication happens via **Ethernet, Bluetooth, or Serial** connections.



Micro-ROS: Architecture

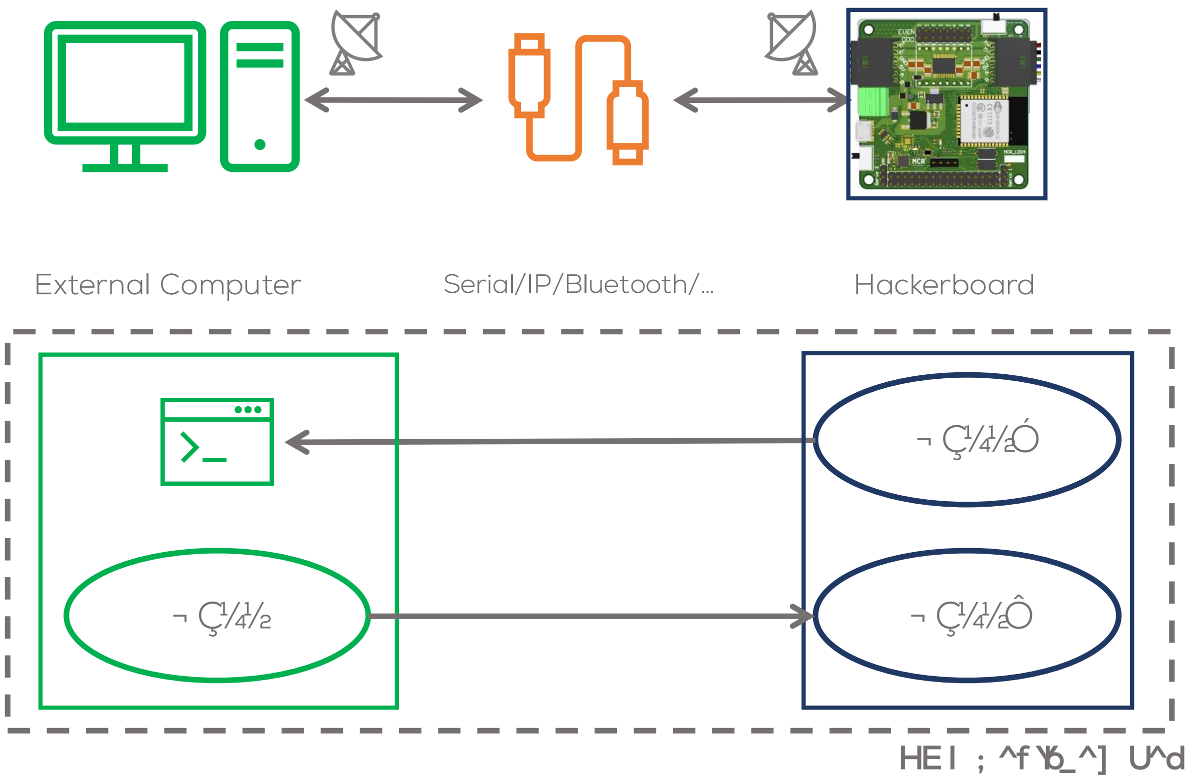


Micro-ROS Client (Right Side)

- Runs directly on a **microcontroller (µC)** and consists of multiple layers:
- Client Library:
  - **ROS API in C (rcl, rclc):** Implements ROS functions, including node management, message handling, and execution.
  - **Middleware Interface (rmw):** Bridges ROS API with communication layers.
  - **Micro XRCE-DDS Adapter:** Adapts standard DDS (Data Distribution Service) for low-power devices.
- Middleware:
  - **Micro XRCE-DDS Client:** Implements lightweight DDS communication, allowing microcontrollers to publish and subscribe to ROS 2 topics with minimal overhead.
- RTOS (Real-Time Operating System) Support:
  - **Zephyr, FreeRTOS, and NuttX** provide real-time capabilities needed for embedded applications.
  - **Micro-ROS Arduino:** Arduino Bare Metal implementation.

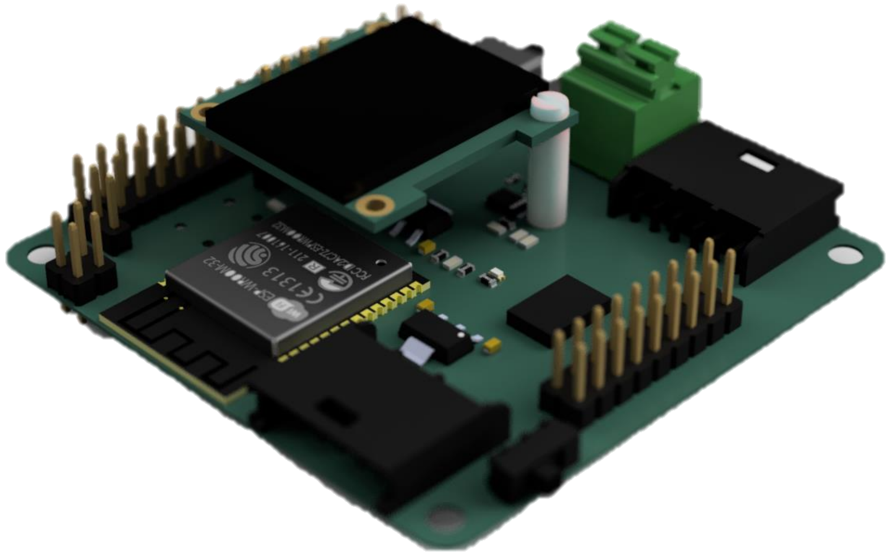
Micro-ROS: Architecture

- Unlike a computer running with ROS, the dedicated OS of the microcontrollers, allows the user to have more control over the timing functions required for certain hardware and control algorithms.
- Micro-ros allows the board to become part of the ROS2 environment ("creating nodes") that can directly publish and subscribe to ROS2 messages, publish TF transforms, and get the ROS2 system time.



Micro-ROS Compatibility

- Not all the microcontrollers can be used with micro-rose.
- Renesas EK RA6M5, ESP32, STM and Arduino are some microcontrollers that micro-ROS support.
- No support for Arduino UNO or Arduino MEGA2560



Micro-ROS Communication

MCU Programming

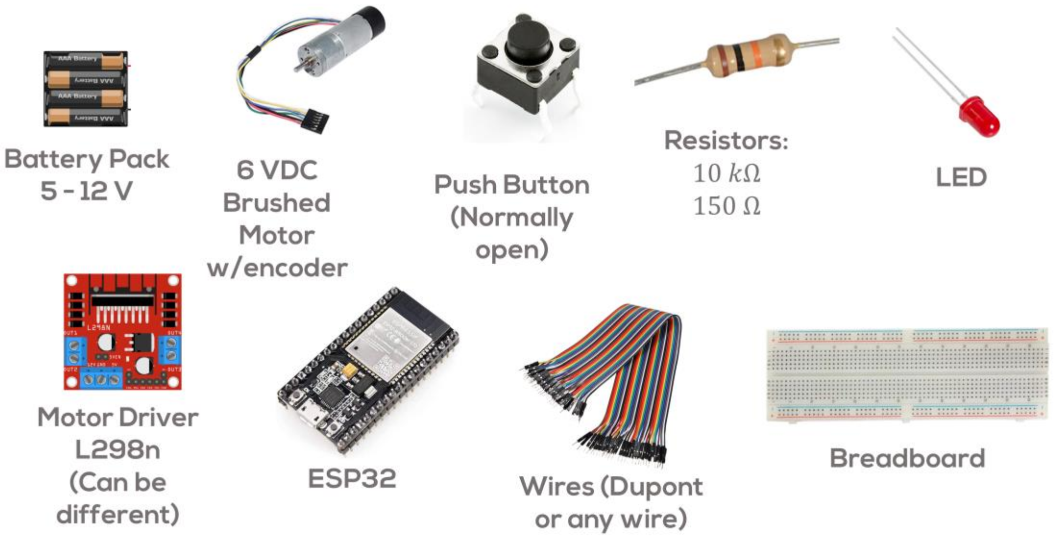
{Learn, Create, Innovate};

Manchester **Robotics**

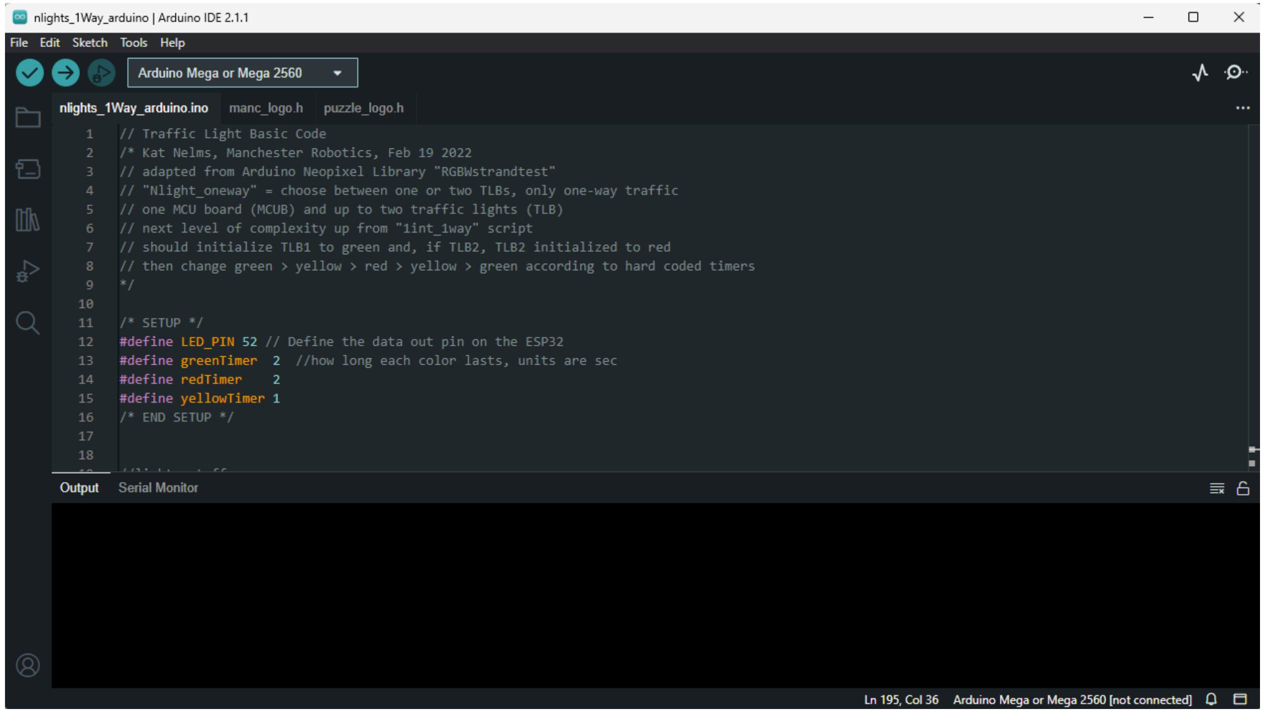
MCU Programming

General information

- As stated before, STM32, Arduino and ESP32 are some of the most used development platforms because of their ease of use.
- Arduino and ESP32 boards can be programmed using the Arduino IDE.
- For all the activities and challenges in this session, the Arduino IDE will be used for programming (Other IDE's can be used such as Platform IO).
- The activities and challenges shown in this presentation will be performed using a ESP32 WROOM or a **MCR2 Hackerboard**.
- Please refer to the prerequisites of this session for the complete list of required components.



MCU Programming



Arduino IDE

- An IDE, or Integrated Development Environment, helps programmers' productivity by combining common activities of writing software into a single application: editing source code, building executables, and debugging.
- Arduino IDE supports C and C++ programming languages.
- A sketch is a program written with the Arduino IDE.
- Sketches are saved on the development computer as text files with the file extension .ino.

MCU Programming

Sketch

- The simplest syntaxis for writing a sketch consists of only two functions:
- setup(): This function is called once when a sketch starts after power-up or reset. It is used to initialize variables, input and output pin modes, and other libraries needed in the sketch. It is analogous to the function main().
- loop(): The loop() function is executed repeatedly in the main program after the setup() function. It controls the board until the board is powered off or is reset.

Sketch Structure

```
// Variable declaration section

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Variable Declaration: Libraries, Components, Variables, constants, Definitions, etc.

Setup Section: Set up sensors, variables, Ports, Functions, Serial comms.

Loop Section: Loops and repeats actions.

Micro-ROS Communication

Micro-ros Program Structure

{Learn, Create, Innovate};

Manchester Robotics

Micro-ros Program Structure

Micro-ros program architecture

- Unlike traditional Arduino sketches, Micro-ROS follows a **modular and event-driven architecture**.
- Instead of a simple setup() and loop(), micro-ros uses an event driven approach, where an **executor** dynamically **schedules tasks** based on the user or ROS2 events.
- Micro-ros does not rely on an OS-based scheduler (like Linux), so the executor provides structured task execution.
- It guarantees that each event (message, timer, service request) is processed in a controlled manner.

Normal Arduino Program Structure

```
Libraries to be used.

Setup:
     Instantiate objects,
     Configure the variables and
     libraires to be used.

Loop:
     Run the program
     sequentially and in a loop
```

Micro ROS Program Structure

```
Libraries to be used.

Setup:
     • Instantiate objects (timers, subscribers, etc.)
     • Define callbacks
     • Configure the variables
     • Config executor

Loop:
     Executor(checkCallbacks)
     if (callback ready || new data || trigger)
        Execute(callbacks)
```

Micro-ros Program Structure

What is the Micro-ROS Executor?

- The **executor** is a key component in **Micro-ROS** that **manages callbacks asynchronously**, ensuring **efficient execution of multiple tasks**.
- It is responsible for handling:
  - **Timers** (periodic function execution).
  - **Subscriptions** (processing incoming ROS 2 messages).
  - **Services** (handling requests and responses).
  - **Publishers** (sending messages to the ROS 2 network).

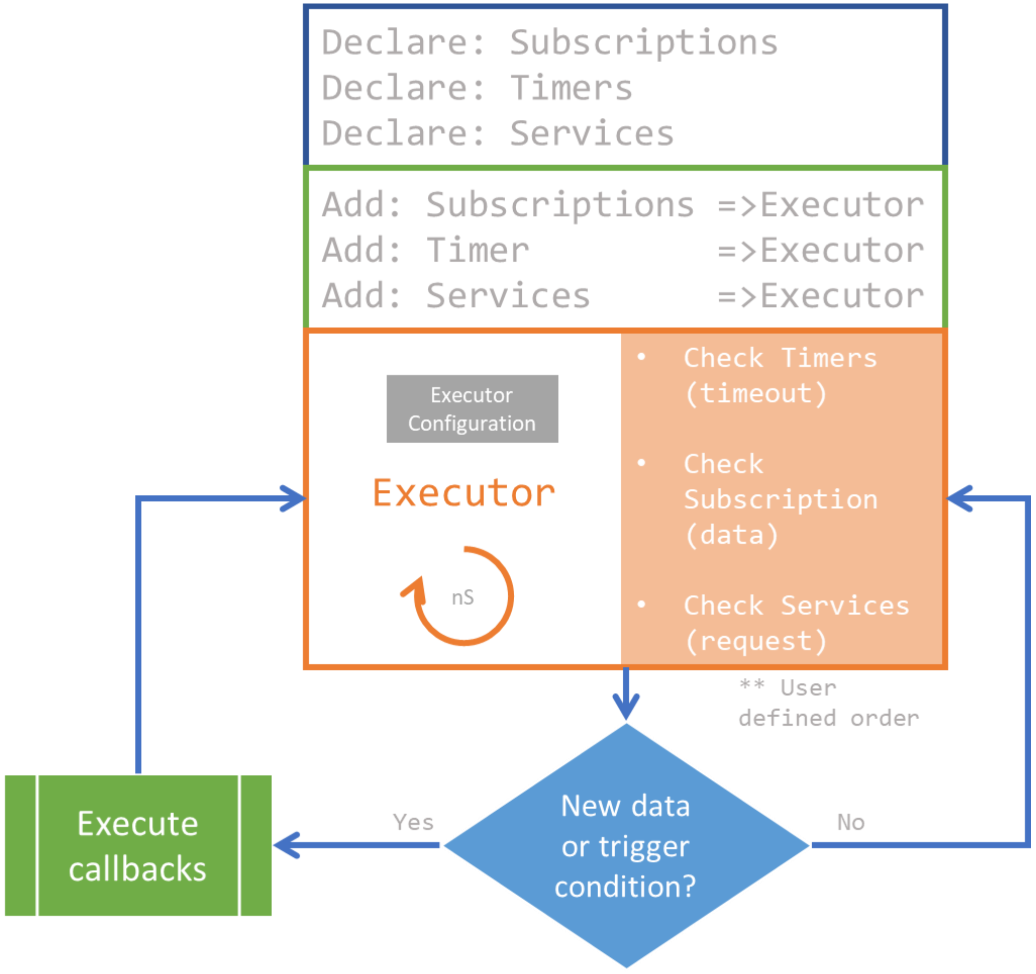
Why an Executor?

- Ensuring predictable execution under real-time constraints is essential for many robotic applications.
- The service-based paradigm of ROS lacks fine-grained control over execution management, i.e., there are no built-in mechanisms to enforce a specific execution order for callbacks within a node.

Micro-ros Program Structure

How the Executor Works in Micro-ROS

- Micro-ROS Initializes the Handlers (timers, subscribers, etc.) and Executors.
  - The executor is created and linked to a ROS 2 node.
  - Timers, subscriptions, publishers are created.
- Callbacks Are Added to the Executor
  - The developer registers subscriptions, timers, and services to the executor.
  - Each callback function is assigned to a specific event (e.g., new message received).
- Executor Runs in loop()
  - The executor "spins" (This checks for incoming messages or scheduled events)
  - Executes the corresponding callbacks.



ROS2 Arduino Sketch Structure

Init Section

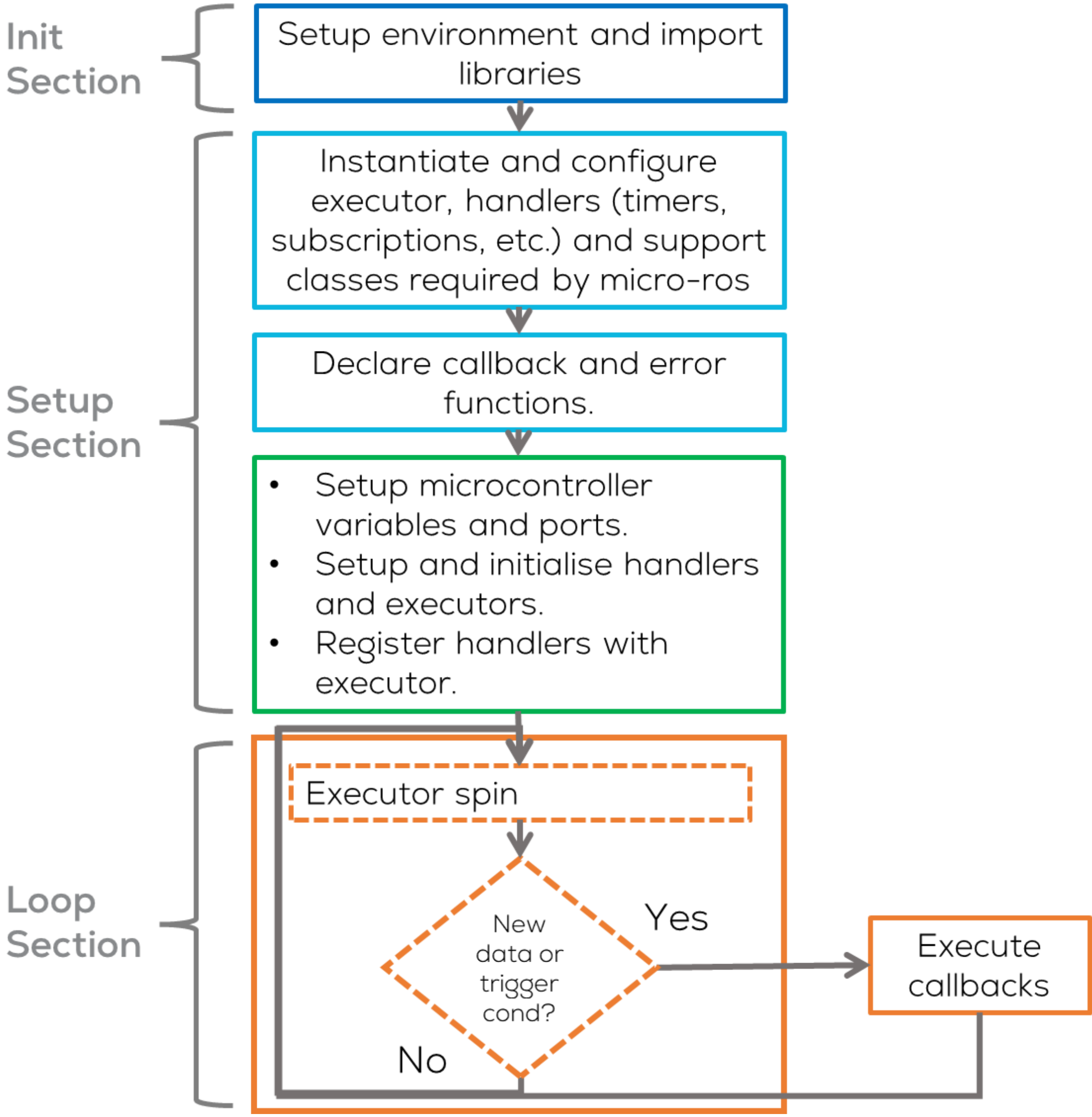
- Setup Environment
- Import ROS and user libraries

Setup section

- Instantiate the executor, handlers and support objects required by micro-ROS.
- Declare callback and error functions (if required).
- Initialise variables, ports, functions, etc.
- Register declared handlers with executor.
- Configure Executor.

Loop section

- Spin Executor.
- Execute Callback, according to executor configuration (New data or trigger conditions).



Executor Configuration

Steps to Configure an Executor in Micro-ROS

1. Initialize Memory Allocator: Required for **memory allocation** in Micro-ROS.
2. Initialize Support and Create a Node:
   - Support: manages the execution context of Micro-ROS, including its communication state, memory management, and initialisation data
   - Node: acts as a ROS 2 processing unit.
3. Create and Configure the Executor:
4. Initializes an executor.
   - 1 in rclc_executor_init() defines the number of handles (e.g., timers, subscribers).
5. Add Callbacks (e.g., Timer) to the Executor
6. Process Executor in loop()

```
// Include Libraries to be used
#include <micro_ros_arduino.h>    //micro-ros-arduino library
#include <rcl/rcl.h>              //Core ROS 2 Client Library (RCL) for node management.
#include <rcl/error_handling.h>   //Error handling utilities for Micro-ROS.
#include <rclc/rclc.h>            //Micro-ROS Client library for embedded devices.
#include <rclc/executor.h>        //Micro-ROS Executor to manage callbacks
//Instantiate executor and its support classes
rclc_executor_t executor;   //Manages task execution (timers, callbacks, etc.).
rclc_support_t support;     //Data structure that holds the execution context of Micro-ROS, including its communication state, memory management, and initialization data.
rcl_allocator_t allocator;  //Manages memory allocation.
//Setup
void setup() {
...
  //Initializes memory allocation for Micro-ROS operations.
  allocator = rcl_get_default_allocator();
  //Creates a ROS 2 support structure to manage the execution context.
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));
  // create node
  RCCHECK(rclc_node_init_default(&node, "micro_ros_sub_node", "", &support));
  // create executor
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  // Register subscription with the executor
  RCCHECK(rclc_executor_add_subscription(&executor, &subscriber, &msg, &subscription_callback, ON_NEW_DATA));
    // Register timer with executor
  RCCHECK(rclc_executor_add_timer(&executor, &timer));
}

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

Micro-ROS Serial Communication

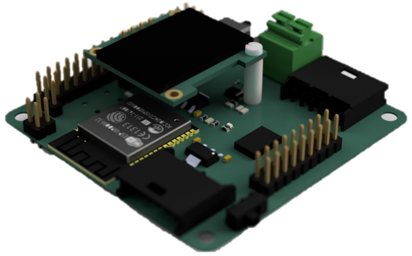
Activity 1: Simple Publisher

{Learn, Create, Innovate};

MCR^2 Manchester Robotics

Requirements

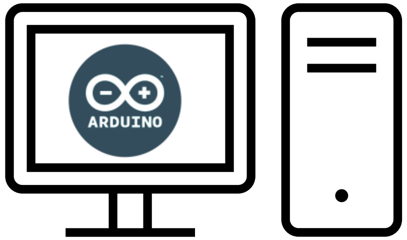
- The following activity is based on the example tutorial found in the provided micro-ROS libraries.
- This activity requires Arduino IDE to be installed and configured as shown in "MCR2_Micro_ROS_Installation".
- Requirements:
    - Microcontroller
    - Computer
    - micro-usb to USB cable (Data)
    - Breadboard
    - LED
    - 150 Ohm Resistor



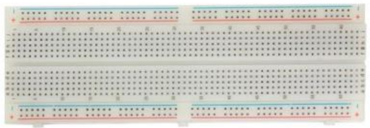
Hackerboard

Or

ESP32 board



Computer

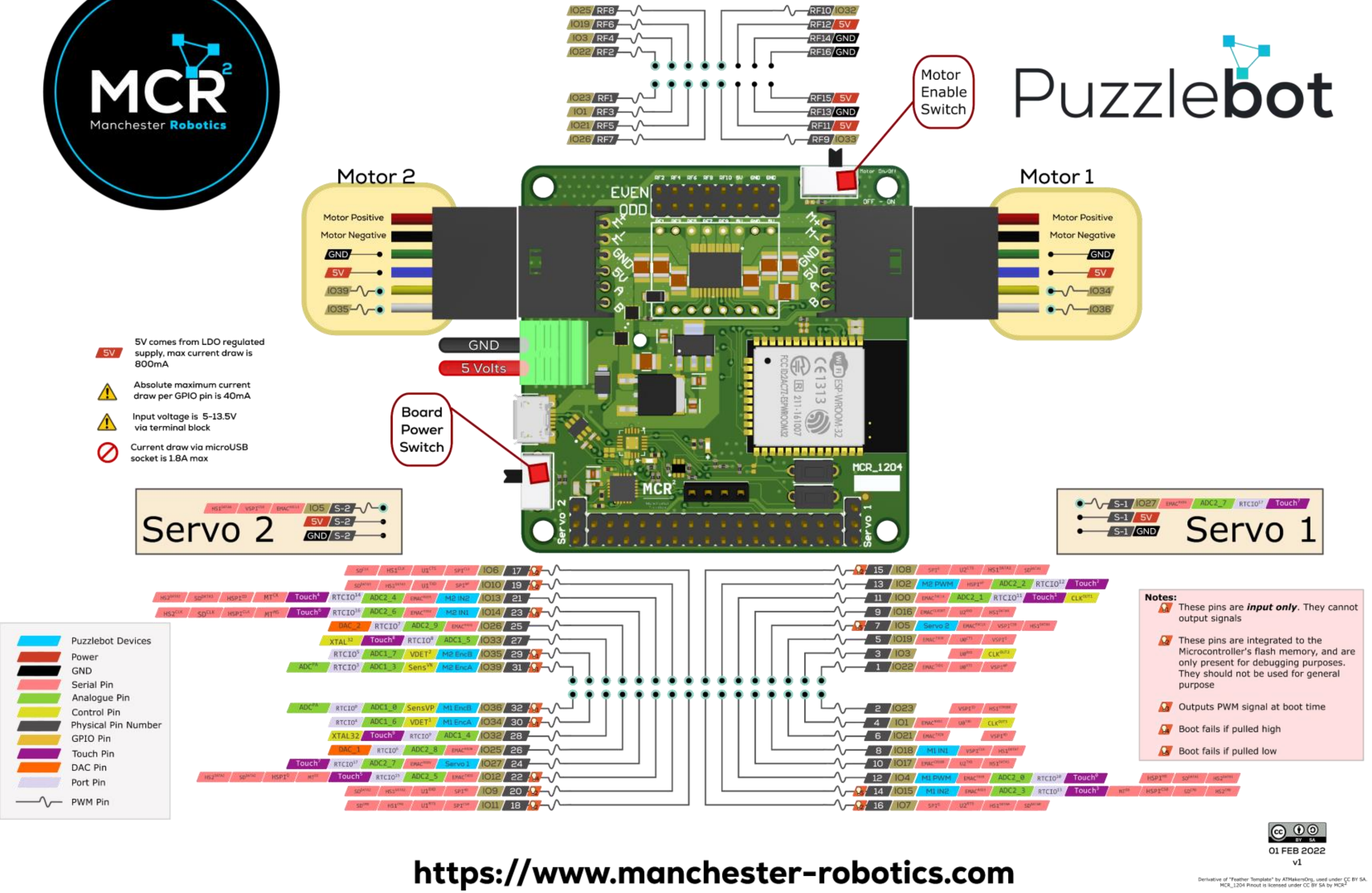


Breadboard

LED

150Ω Resistor

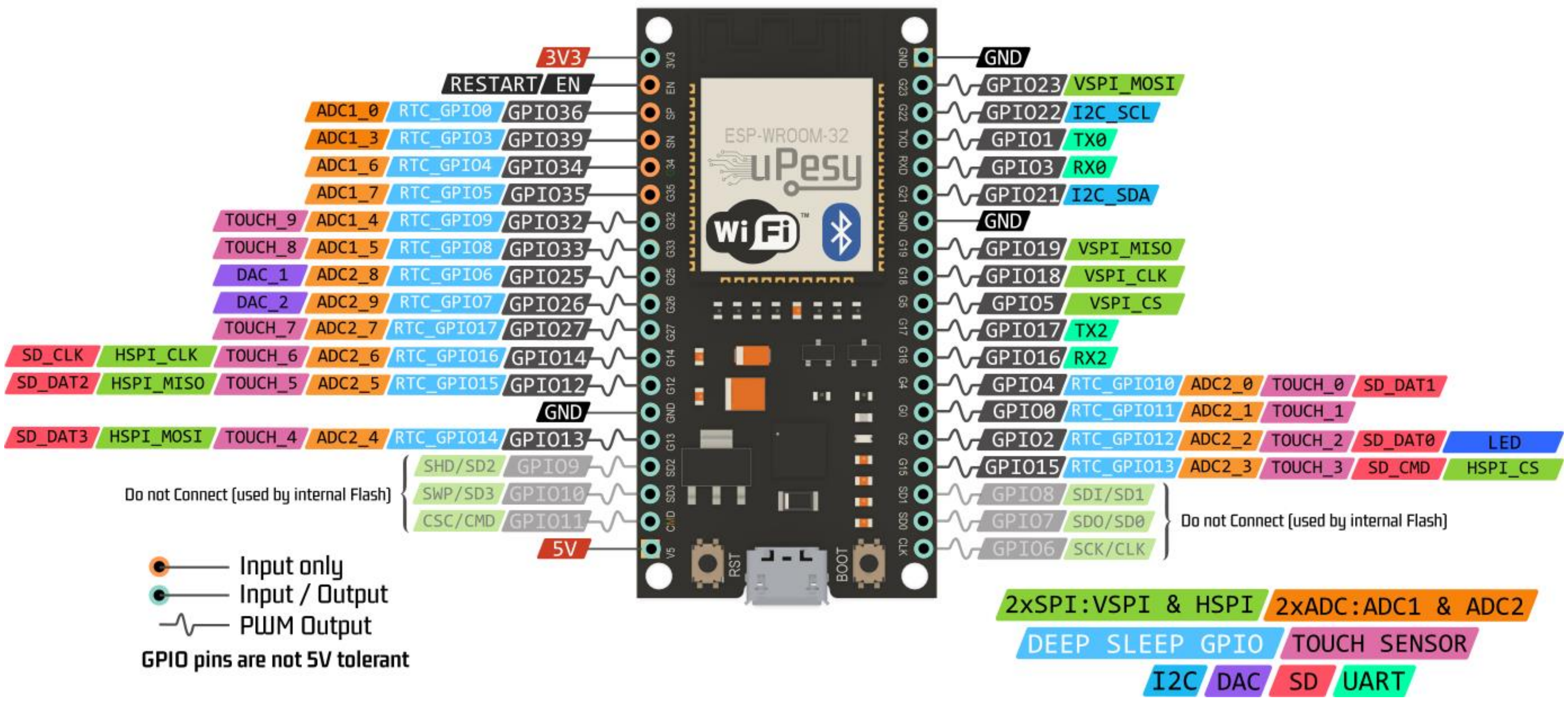
Hackerboard Pinout



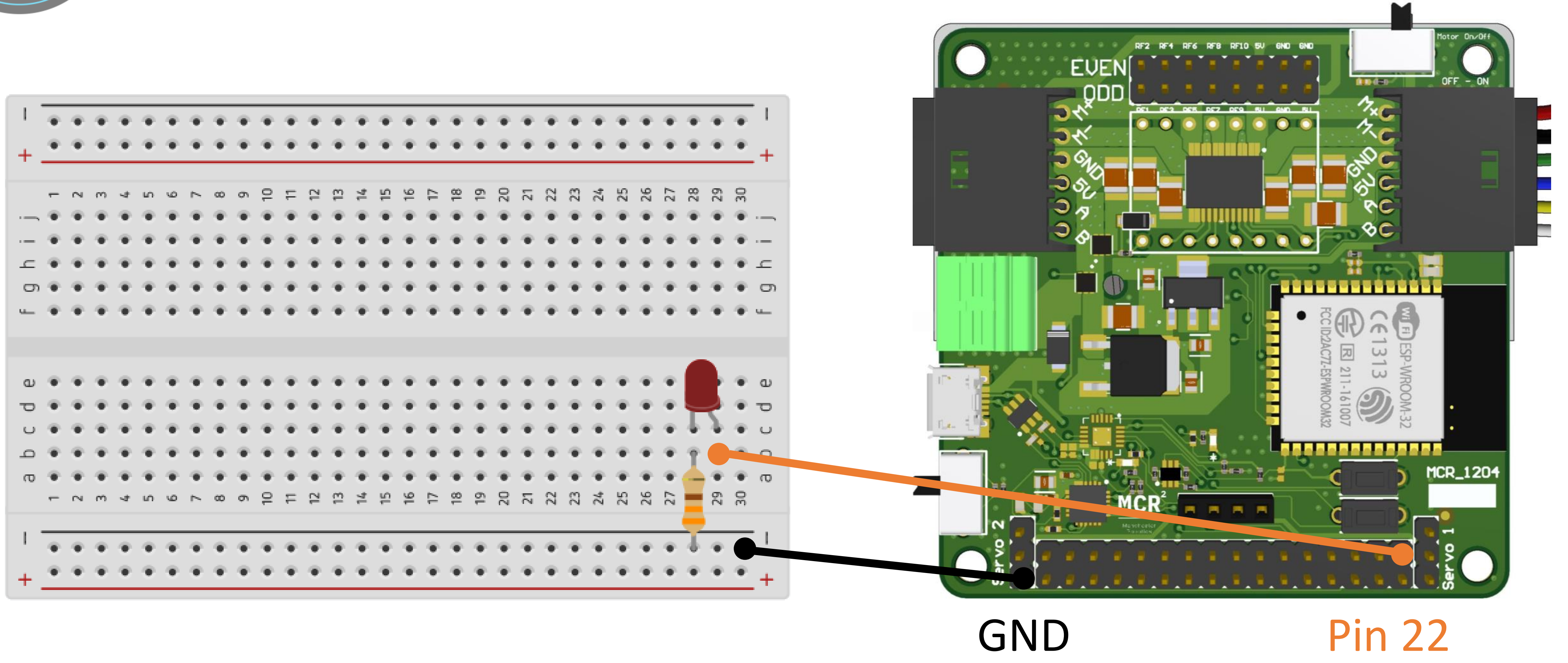
https://www.manchester-robotics.com

ESP32 Pinout

ESP32 Wroom DevKit Full Pinout



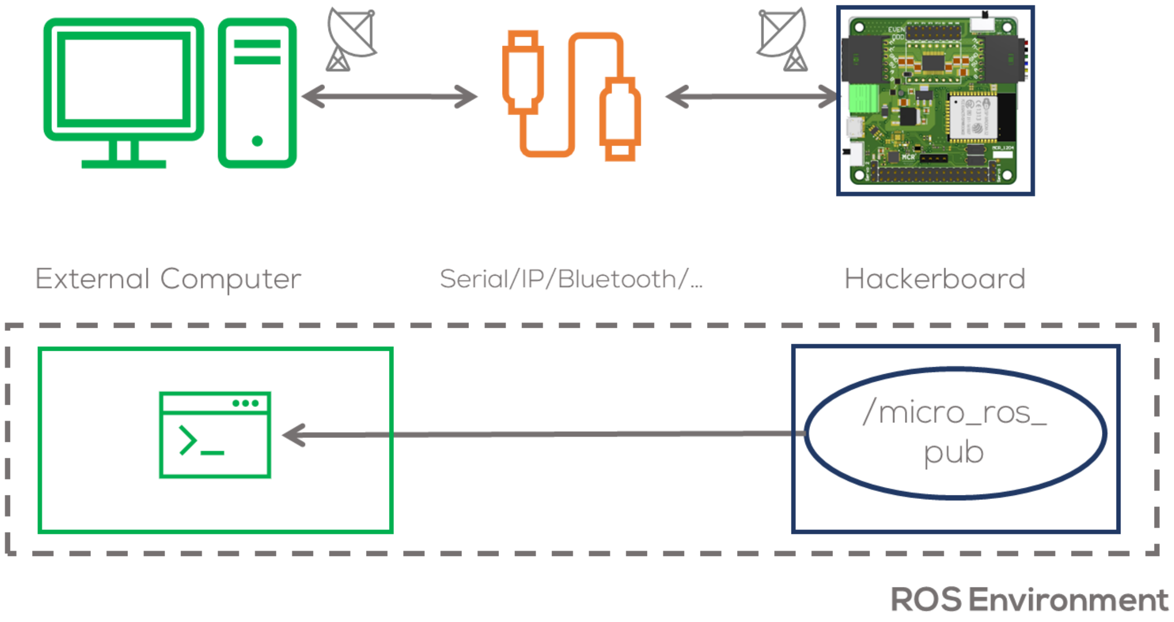
Connections



Introduction

Introduction

- In this activity, a node running a simple publisher inside the microcontroller will be declared.
- This node will run inside the microcontroller and will communicate with the computer via UART.
- The node will publish a simple int32 message.
- This activity will be divided into two parts. The first part involves the Arduino IDE to program the MCU.
- The second part involving the commands required to connect to the board to the computer.



Publisher set up

Configuring a Publisher in Micro-ROS

What is a Publisher?

- A **publisher** sends data to a **ROS 2 topic**, allowing other nodes to receive it.
- In Micro-ROS, publishers work by **sending messages periodically or on-demand**.

1. Define the Message Type
2. Initialize the Publisher
3. Publish Messages Periodically
4. Use a Timer or loop to Publish at Intervals

```
#include <std_msgs/msg/int32.h>    //Predefined ROS 2 message type

//Declare Messages to be used
std_msgs__msg__Int32 msg;  //Defines a message of type int32.
//Declare Publishers to be used
rcl_publisher_t publisher;  //Declares a ROS 2 publisher for sending messages.
...
void setup() {
  // create publisher
 RCCHECK(rclc_publisher_init_default(
   &publisher,
   &node,
   ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
   "micro_ros_counter"));
}
void loop() {
...
//Fill Message
msg.data = 2;
//Publishes msg to the ROS 2 topic.
RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
delay(100);
}
```

Timer set up

Configuring a Timer in Micro-ROS

What is a Timer in Micro-ROS?

- A timer executes a function at fixed intervals without blocking the system.
- Useful for periodic publishing, sensor readings, and state updates.Define the Message Type

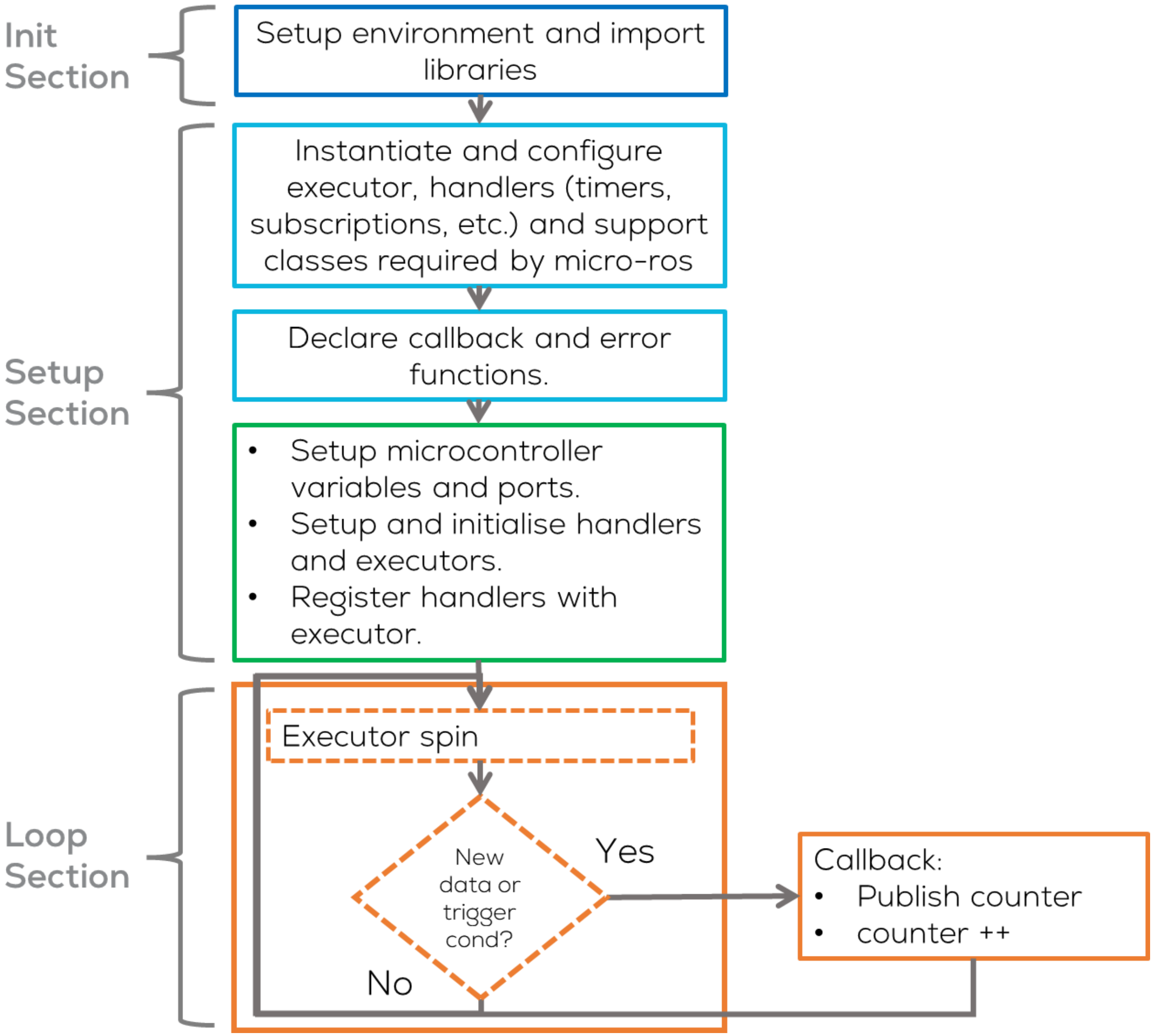
1. Define a timer and a callback function
2. Initialize the Timer inside the setup and define a timeout.
3. Add Timer to the Executor
4. Process the Executor in loop()

```
//Declare timers to be used
rcl_timer_t timer;          //Creates a timer

void timer_callback(rcl_timer_t * timer, int64_t last_call_time) {
    (void) last_call_time;
    if (timer != NULL) {
        Serial.println("Timer triggered!");
    }
}

void setup() {
const unsigned int timer_timeout = 1000; // 1 second
rclc_timer_init_default(
    &timer,
    &support,
    RCL_MS_TO_NS(timer_timeout),
    timer_callback);

  // Initializes the Micro-ROS Executor, which manages tasks and callbacks.
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  // Register timer with executor
  RCCHECK(rclc_executor_add_timer(&executor, &timer));
}

void loop() {
  //Executor Spin
  delay(100);
  RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```

Micro-ros Publisher

Introduction

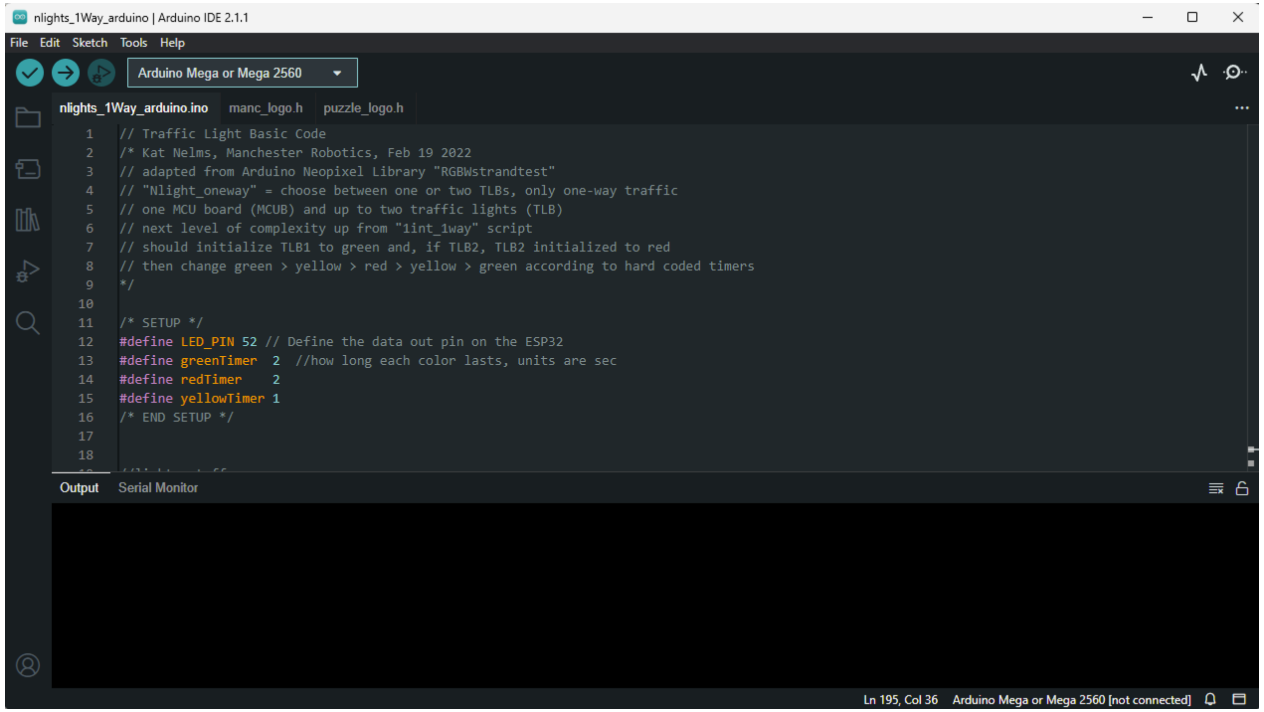
- The node will be named “micro_ros_pub”
- The node must perform the following:
    - Setup a timer to run control the publishing of information.
    - When the timer times out, the node will publish the value of a counter in the topic “*micro_ros_counter*”.
    - The value of the counter must be increased.
- The computer will receive display subscribe to that topic and publish the information on the terminal.



Activity

Instructions (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 or Arduino Due
    - For ESP32 select ESP32 Arduino › DOIT ESP32 DEVKIT V1
- Copy and paste the following code (next slides)



Micro-ros Publisher

```
// Include Libraries to be used
#include <micro_ros_arduino.h>    //micro-ros-arduino library
#include <rcl/rcl.h>              //Core ROS 2 Client Library (RCL) for node management.
#include <rcl/error_handling.h>   //Error handling utilities for Micro-ROS.
#include <rclc/rclc.h>            //Micro-ROS Client library for embedded devices.
#include <rclc/executor.h>        //Micro-ROS Executor to manage callbacks
#include <std_msgs/msg/int32.h>   //Predefined ROS 2 message type
#include <stdio.h>                //Standard I/O library for debugging.

//Declare nodes to be used
rcl_node_t node;            //Represents a ROS 2 Node running on the microcontroller.

//Instantiate executor and its support classes
rclc_executor_t executor;   //Manages task execution (timers, callbacks, etc.).
rclc_support_t support;     //Handles initialization & communication setup.
rcl_allocator_t allocator;  //Manages memory allocation.

//Declare Publishers to be used
rcl_publisher_t publisher;  //Declares a ROS 2 publisher for sending messages.

//Declare timers to be used
rcl_timer_t timer;          //Creates a timer to execute functions at intervals.

//Declare Messages to be used
std_msgs__msg__Int32 msg;  //Defines a message of type int32.

//Define Macros to be used
//Executes fn and calls error_loop() if it fails.
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
// Executes fn, but ignores failures.
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}

#define LED_PIN 22 //Specifies GPIO pin 13 for controlling an LED
```

Init Section

- Libraries:
  - **micro_ros_arduino:** Main **Micro-ROS library** for Arduino
  - **RCL (Ros Client Library):** Core **ROS 2 Client Library (RCL)** for node management.
  - **RCLC: Micro-ROS Client** library for embedded devices.
  - **std_msgs**: ROS messages library
- Instantiate the executor
- Instantiate the publisher and timer
- Define Macros to verify if there is an error
  - RCCHECK(fn) → Executes fn and calls error_loop() if it fails.
  - RCSOFTCHECK(fn) → Executes fn, but ignores failures.
- Define Output pins

Micro-ros Publisher

```
//Define Error Functions
void error_loop(){
  while(1){
    // Toggle LED state
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    // Wait 100 milliseconds
    delay(100);
  }
}

//Define callbacks
void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
  //Prevents compiler warnings about an unused parameter.
  RCLC_UNUSED(last_call_time);
  //Ensures the timer event is valid before executing actions.
  if (timer != NULL) {
    //Publishes msg to the ROS 2 topic.
    RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
   //Increments the integer message value.
    msg.data++;
  }
}
```

Setup Section (Functions)

Define callback and error functions.

- Error Handling Function (error_loop()):
  - This function is called when a **critical error** occurs in Micro-ROS (**LED blinks continuously**).
  - The microcontroller **enters an infinite loop** to indicate an error, preventing the system from continuing **in an unstable state**.
- Timer Callback Function (timer_callback())
  - This function is triggered periodically by the Micro-ROS timer.The function publishes a message (msg) to a ROS 2 topic.
  - After each execution, msg.data is incremented.

Micro-ros Publisher

```
void setup() {
  // Initializes communication between ESP32 and the ROS 2 agent (Serial).
  set_microros_transports();

  //Setup Microcontroller Pins
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, HIGH);

  //Connection delay (waiting for agent to be available)
  delay(2000);

  //Initializes memory allocation for Micro-ROS operations.
  allocator = rcl_get_default_allocator();

  //Creates a ROS 2 support structure to manage the execution context.
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

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

  // create publisher
  RCCHECK(rclc_publisher_init_default(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "micro_ros_counter"));

  // create timer,
  const unsigned int timer_timeout = 1000;
  RCCHECK(rclc_timer_init_default(
    &timer,
    &support,
    RCL_MS_TO_NS(timer_timeout),
    timer_callback));

  // Initializes the Micro-ROS Executor, which manages tasks and callbacks.
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  // Register timer with executor
  RCCHECK(rclc_executor_add_timer(&executor, &timer));

  // Initialise message
  msg.data = 0;
}
```

Micro-ros Publisher

Setup Section

- set_microros_transports(): Initializes communication between ESP32/Arduino and the ROS 2 agent (via Serial, Wi-Fi, Ethernet, etc.).
- delay(): Waits to allow stable connection establishment.
- rcl_get_default_allocator(): Initializes memory allocation for Micro-ROS operations.
- rclc_publisher_init_default(): Creates a publisher for Int32 messages on the topic "micro_ros_pub".
- rclc_timer_init_default(): Creates a timer that executes timer_callback() every 1000 milliseconds. RCL_MS_TO_NS() converts milliseconds to nanoseconds.
- rclc_executor_init(): Initializes the Micro-ROS Executor, which manages tasks and callbacks.
- rclc_executor_add_timer(): Registers the timer with the executor so that it can execute tasks non-blocking.

Micro-ros Publisher

```
void loop() {
  //Executor Spin
  delay(100);
  RCSOFTCHECK(rclc_executor_spin_some(&executor,
RCL_MS_TO_NS(100)));
}
```

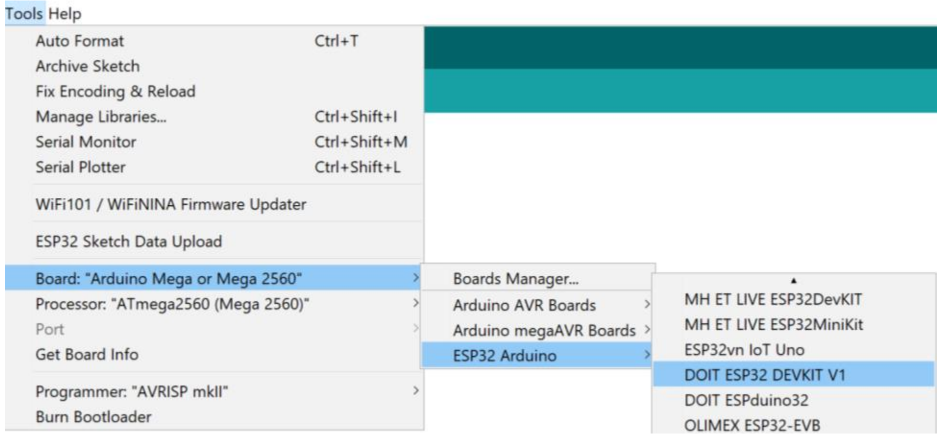
Loop Section

- This function checks for new data/timer timeouts/triggers and executes pending callbacks (e.g., timers, subscriptions, services).
- Does not block execution, meaning it allows other tasks to continue running.
- The argument RCL_MS_TO_NS(100) specifies the timeout in nanoseconds (100ms).

Activity

Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for **Hackerboard** is the same)
  - For ESP32 select ESP32 Arduino › DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.

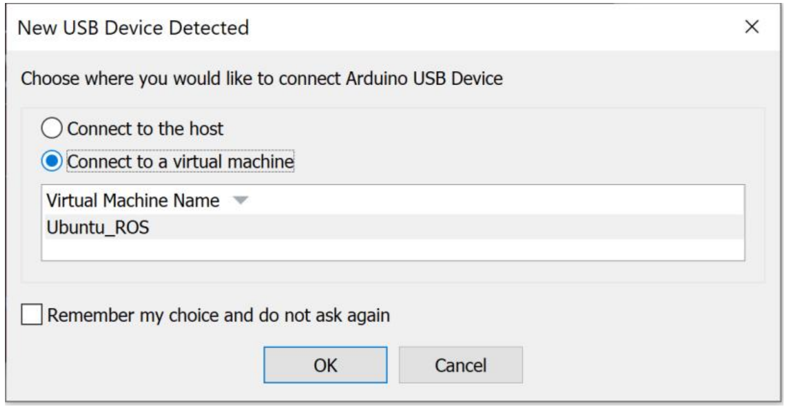
- The following message should be displayed:

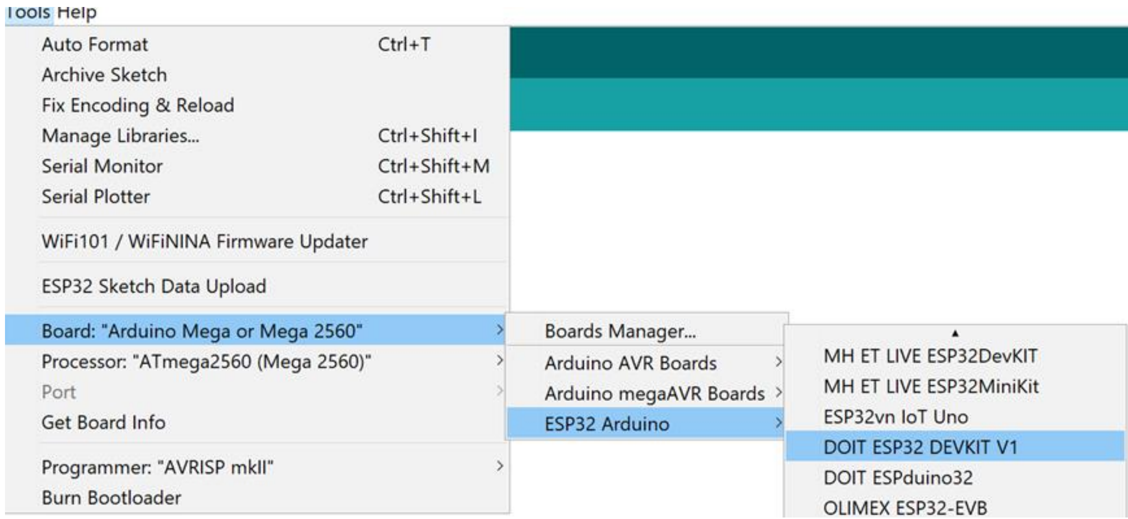
```
Done compiling.
Sketch uses 9424 bytes (3%) of program storage space. Maximum is 253952 bytes.
Global variables use 1826 bytes (22%) of dynamic memory, leaving 6366 bytes for local variables. Maximum is 8192 bytes.
```

Activity

Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools›Port
  - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
  - If in WSL, follow the steps on the presentation: “MCR2_Micro_ROS_Installation”.
- Select the board to be used Tools -> ESP32 Arduino › DOIT ESP32 DEVKIT V1



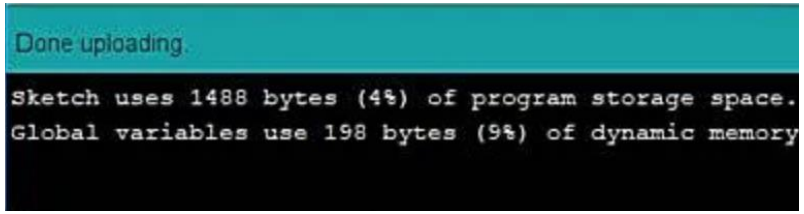


Activity

Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.

- The following message should appear o the IDE



Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
  - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*
sudo chmod 666 /dev/ttyUSB*
```

Activity

Test (Computer)

1. Open a terminal and type the following.

```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```

2. Reset the ESP32 (pressing the reset button) to reconnect to the computer (agent waiting timeout).

```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent mi
cro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info     | TermiosAgentLinux.cpp
 | init                    | running...
 | fd: 3
[1737636692.138467] info     | Root.cpp                 |
set_verbose_level          | logger setup             |
verbose_level: 4
[1737636698.665805] info     | Root.cpp                 |
create_client              | create                   |
client_key: 0x0C9424D2, session_id: 0x81
[1737636698.666090] info     | SessionManager.hpp |
establish_session          | session established      |
client_key: 0x0C9424D2, address: 0
[1737636698.702311] info     | ProxyClient.cpp          |
create_participant         | participant created      |
client_key: 0x0C9424D2, participant_id: 0x000(1)
[1737636698.721583] info     | ProxyClient.cpp          |
create_topic               | topic created            |
client_key: 0x0C9424D2, topic_id: 0x000(2), particip
ant_id: 0x000(1)
[1737636698.733573] info     | ProxyClient.cpp          |
create_publisher           | publisher created        |
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

- Open another terminal and type the following.

```
$ ros2 topic list
```

```
mario@MarioPC:~$ ros2 topic  list
/micro_ros_counter
/parameter_events
/rosout
```

- Echo the topic "/micro_ros_arduino_node_publisher"

```
$ ros2 topic echo /micro_ros_arduino_node_publisher
```

```
mario@MarioPC:~$ ros2 topic echo /micro_ros_counter
data: 24
---
data: 25
---
data: 26
---
data: 27
---
data: 28
---
data: 29
---
```

Micro-ROS Serial Communication

Activity 1.2: Publisher w/reconnection

{Learn, Create, Innovate};

Manchester Robotics

Activity 1.2

Introduction

- In the previous activity a simple publisher was declared.
- When initialising the communication with the agent a delay() function was used to wait for the agent to be available

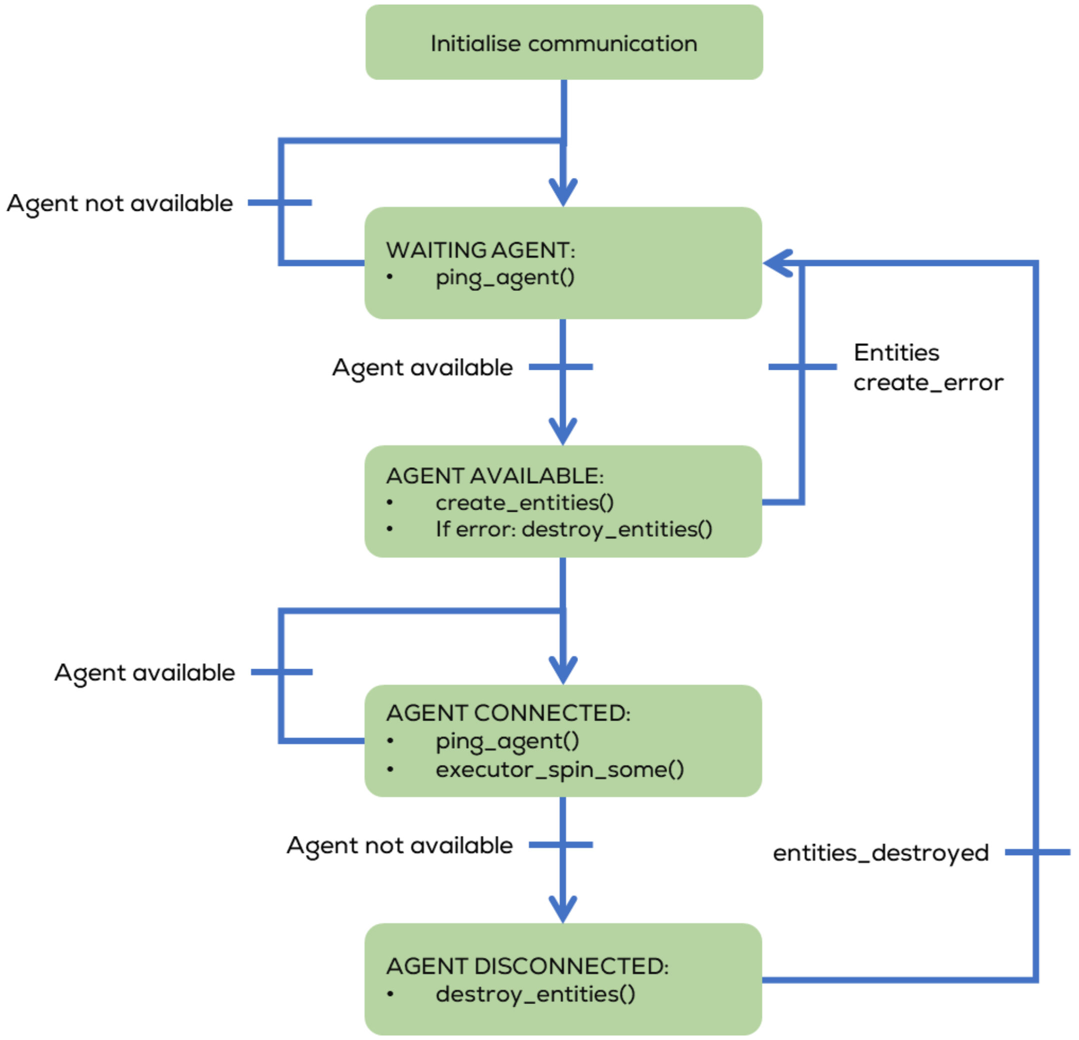
```
void setup() {
  // Initializes communication between ESP32 and the ROS 2 agent (Serial).
  set_microros_transports();
...

  //Connection delay (waiting for agent to be available)
  delay(2000);
...
}
```

- If no agent was available during that time, the user had to restart the microcontroller **after** the agent had been started on the host computer.
- Usually a restart from themicrocontroller, is not recommended when working with robots.
- A reconnection function is required.

Activity 1.2

Reconnection state machine

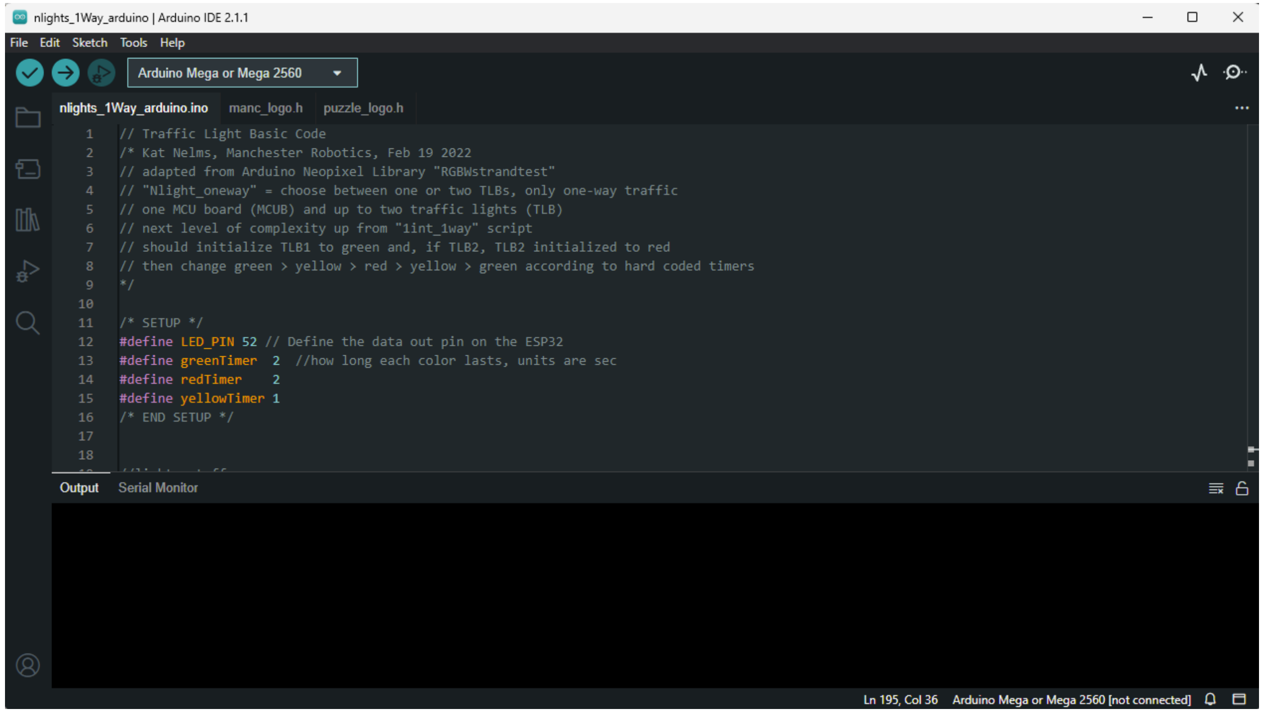
- The reconnection function is made using a state machine.
- The machine, uses the ping function to verify if an agent is running on the host computer (waiting for a connection).
- If an agent is detected, the program initialises the "entities" required by micro-ros to communicate i.e., supports, executors, and handlers.
- If an error occurs during the creation, the program destroys the created entities.
- If there is no error, the executor starts spinning and pinging the host to verify if the agent is still available.
- If the agent gets disconnected, the entities get destroyed, and the agent continues waiting for the agent to get connected again.



Activity 1.2

Instructions (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 or Arduino Due
  - For ESP32 select ESP32 Arduino › DOIT ESP32 DEVKIT V1
- Open the previous Activity and modify it according to the following slides. Modifications are highlighted in Black.
- For this activity, no LED on Pin 22 is required



Activity 1.2

```c
// Include Libraries to be used
#include <micro_ros_arduino.h>    //micro-ros-arduino library
#include <rcl/rcl.h>              //Core ROS 2 Client Library (RCL) for node management.
#include <rcl/error_handling.h>   //Error handling utilities for Micro-ROS.
#include <rclc/rclc.h>            //Micro-ROS Client library for embedded devices.
#include <rclc/executor.h>        //Micro-ROS Executor to manage callbacks
#include <std_msgs/msg/int32.h>   //Predefined ROS 2 message type
#include <stdio.h>                //Standard I/O library for debugging.

//Declare nodes to be used
rcl_node_t node;            //Represents a ROS 2 Node running on the microcontroller.

//Instantiate executor and its support classes
rclc_executor_t executor;   //Manages task execution (timers, callbacks, etc.).
rclc_support_t support;     //Handles initialization & communication setup.
rcl_allocator_t allocator;  //Manages memory allocation.

//Declare Publishers to be used
rcl_publisher_t publisher;  //Declares a ROS 2 publisher for sending messages.

//Declare timers to be used
rcl_timer_t timer;          //Creates a timer to execute functions at intervals.

//Declare Messages to be used
std_msgs__msg__Int32 msg;  //Defines a message of type int32.
```

```c
//Define Macros to be used
//Executes fn and returns false if it fails.
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){return false;}}

// Executes fn, but ignores failures.
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}

//  Executes a given statement (X) periodically every MS milliseconds
#define EXECUTE_EVERY_N_MS(MS, X)  do { \
  static volatile int64_t init = -1; \
  if (init == -1) { init = uxr_millis();} \
  if (uxr_millis() - init > MS) { X; init = uxr_millis();} \
} while (0)\

//Defines State Machine States
enum states {
  WAITING_AGENT,
  AGENT_AVAILABLE,
  AGENT_CONNECTED,
  AGENT_DISCONNECTED
} state;

//Define callbacks
void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
  RCLC_UNUSED(last_call_time);
  if (timer != NULL) {
    RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
    msg.data++;
  }
}
```

Activity 1.2

```
bool create_entities()
{
   //Initializes memory allocation for Micro-ROS operations.
  allocator = rcl_get_default_allocator();

   //Creates a ROS 2 support structure to manage the execution context.
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

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

  // create publisher
  RCCHECK(rclc_publisher_init_default(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "micro_ros_counter"));

    // create timer,
  const unsigned int timer_timeout = 1000;
  RCCHECK(rclc_timer_init_default(
    &timer,
    &support,
    RCL_MS_TO_NS(timer_timeout),
    timer_callback));

  // create zero initialised executor (no configured) to avoid memory problems
  executor = rclc_executor_get_zero_initialized_executor();
  // Initializes the Micro-ROS Executor, which manages tasks and callbacks.
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  // Register timer with executor
  RCCHECK(rclc_executor_add_timer(&executor, &timer));

  return true;
}
```

```
void destroy_entities()
{
  rmw_context_t * rmw_context = rcl_context_get_rmw_context(&support.context);
  (void) rmw_uros_set_context_entity_destroy_session_timeout(rmw_context, 0);

  rcl_publisher_fini(&publisher, &node);
  rcl_timer_fini(&timer);
  rclc_executor_fini(&executor);
  rcl_node_fini(&node);
  rclc_support_fini(&support);
}

//Setup
void setup() {
  // Initializes communication between ESP32 and the ROS 2 agent (Serial).
  set_microros_transports();

  //Initial State
  state = WAITING_AGENT;

  // Initialise message
  msg.data = 0;
}
```

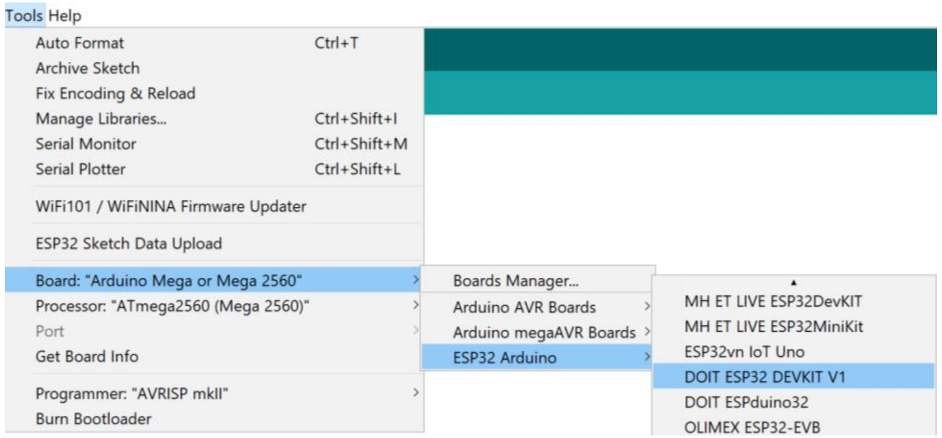
Activity 1.2

```
void loop() {
  switch (state) {

    case WAITING_AGENT:
      EXECUTE_EVERY_N_MS(500, state = (RMW_RET_OK == rmw_uros_ping_agent(100, 1)) ? AGENT_AVAILABLE : WAITING_AGENT;);
      break;

    case AGENT_AVAILABLE:
      state = (true == create_entities()) ? AGENT_CONNECTED : WAITING_AGENT;
      if (state == WAITING_AGENT) {
        destroy_entities();
      };
      break;

    case AGENT_CONNECTED:
      EXECUTE_EVERY_N_MS(200, state = (RMW_RET_OK == rmw_uros_ping_agent(100, 1)) ? AGENT_CONNECTED : AGENT_DISCONNECTED;);
      if (state == AGENT_CONNECTED) {
        rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));
      }
      break;

    case AGENT_DISCONNECTED:
      destroy_entities();
      state = WAITING_AGENT;
      break;

    default:
      break;
  }
}
```

Activity

Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for **Hackerboard** is the same)
  - For ESP32 select ESP32 Arduino › DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.

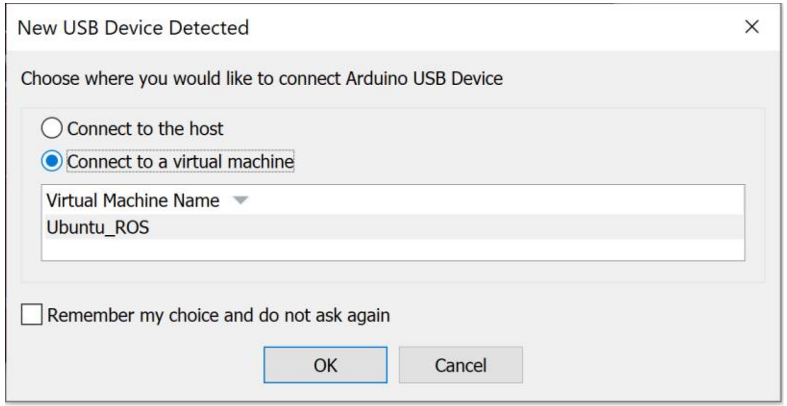
- The following message should be displayed:

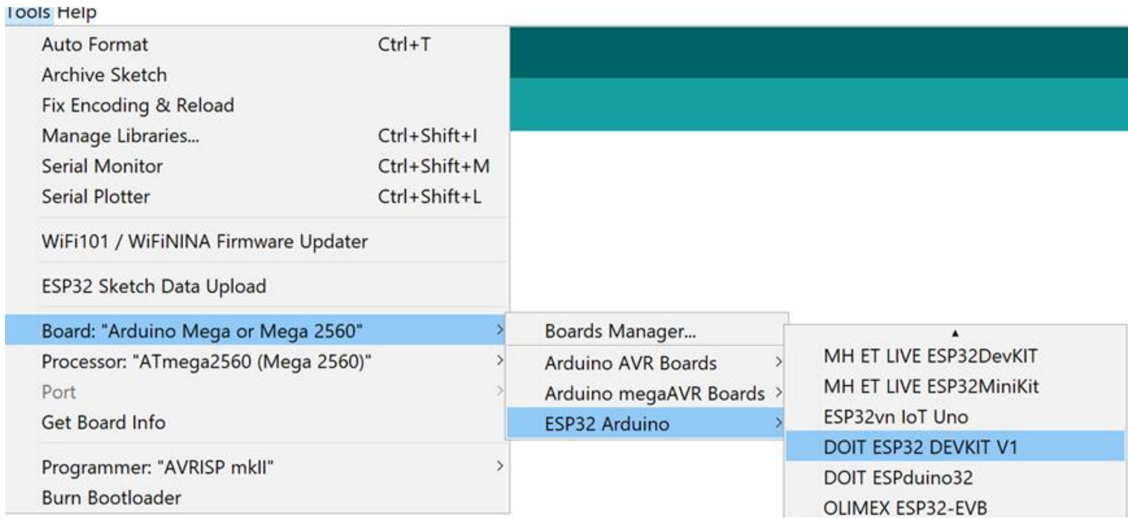
```
Done compiling.
Sketch uses 9424 bytes (3%) of program storage space. Maximum is 253952 bytes.
Global variables use 1826 bytes (22%) of dynamic memory, leaving 6366 bytes for local variables. Maximum is 8192 bytes.
```

Activity

Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools›Port
  - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
  - If in WSL, follow the steps on the presentation: “MCR2_Micro_ROS_Installation”.
- Select the board to be used Tools -> ESP32 Arduino › DOIT ESP32 DEVKIT V1



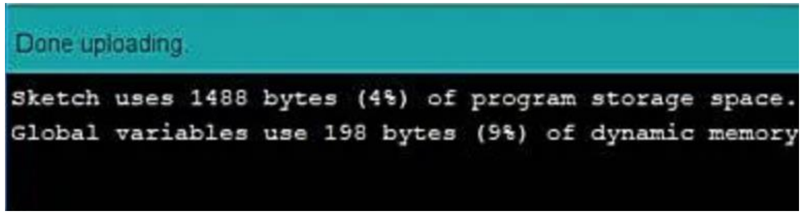


Activity

Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.

- The following message should appear o the IDE



Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
  - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

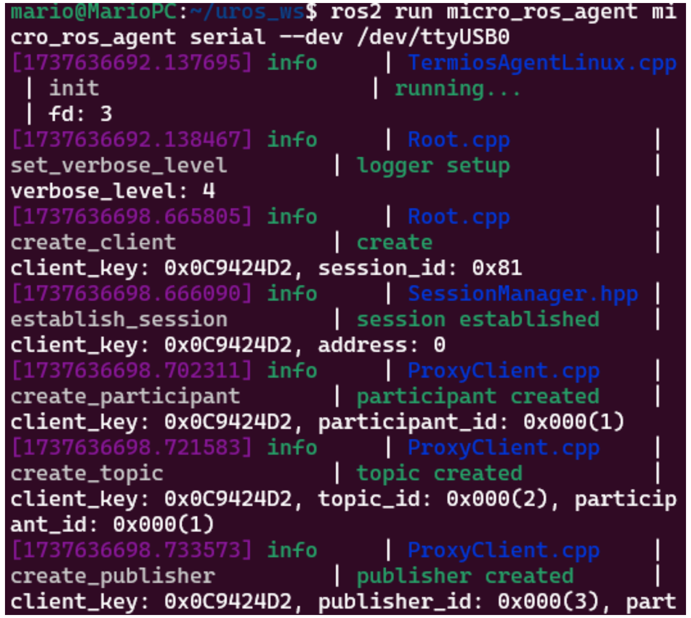
```
sudo chmod 666 /dev/ttyACM*
sudo chmod 666 /dev/ttyUSB*
```

Activity 1.2

Test (Computer)

1. Open a terminal and type the following.

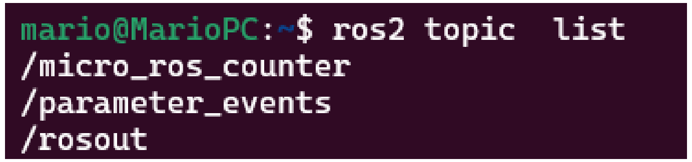
```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```



```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent mi
cro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info     | TermiosAgentLinux.cpp
 | init                      | running...
 | fd: 3
[1737636692.138467] info     | Root.cpp              |
set_verbose_level         | logger setup          |
verbose_level: 4
[1737636698.665805] info     | Root.cpp              |
create_client             | create                |
client_key: 0x0C9424D2, session_id: 0x81
[1737636698.666090] info     | SessionManager.hpp |
establish_session         | session established   |
client_key: 0x0C9424D2, address: 0
[1737636698.702311] info     | ProxyClient.cpp       |
create_participant        | participant created   |
client_key: 0x0C9424D2, participant_id: 0x000(1)
[1737636698.721583] info     | ProxyClient.cpp       |
create_topic              | topic created         |
client_key: 0x0C9424D2, topic_id: 0x000(2), particip
ant_id: 0x000(1)
[1737636698.733573] info     | ProxyClient.cpp       |
create_publisher          | publisher created     |
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

- Open another terminal and type the following.

```
$ ros2 topic list
```

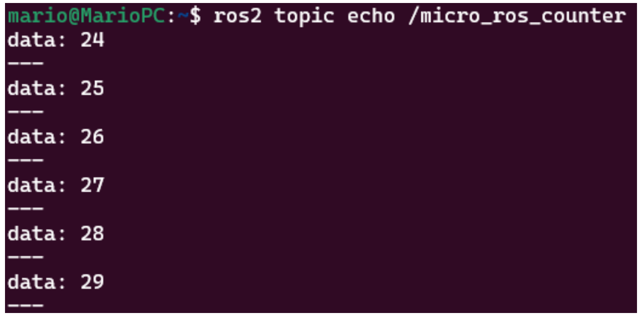


```
mario@MarioPC:~$ ros2 topic  list
/micro_ros_counter
/parameter_events
/rosout
```

- Echo the topic

  "/micro_ros_arduino_node_publisher"

```
$ ros2 topic echo /micro_ros_arduino_node_publisher
```



```
mario@MarioPC:~$ ros2 topic echo /micro_ros_counter
data: 24
---
data: 25
---
data: 26
---
data: 27
---
data: 28
---
data: 29
---
```

Micro-ROS Serial Communication

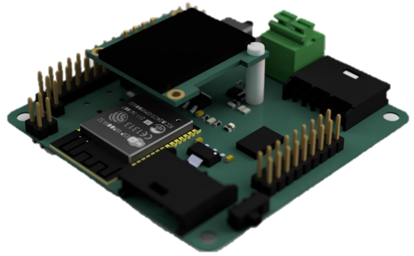
Activity 2: Subscriber

`{Learn, Create, Innovate};`

Manchester Robotics

Requirements

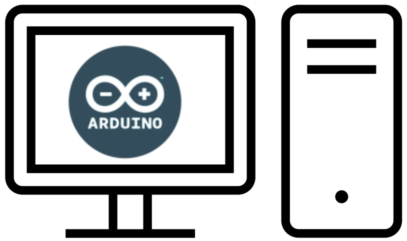
- The following activity is based on the example tutorial found in the provided micro-ROS libraries.
- This activity requires Arduino IDE to be installed and configured as shown in "MCR2_Micro_ROS_Installation".
- Requirements:
    - Microcontroller
    - Computer
    - micro-usb to USB cable (Data)
    - Breadboard
    - LED
    - 2 x 150 Ohm Resistor



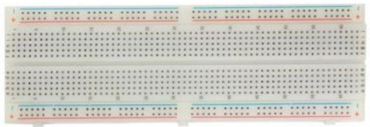
Hackerboard

Or

ESP32 board



Computer

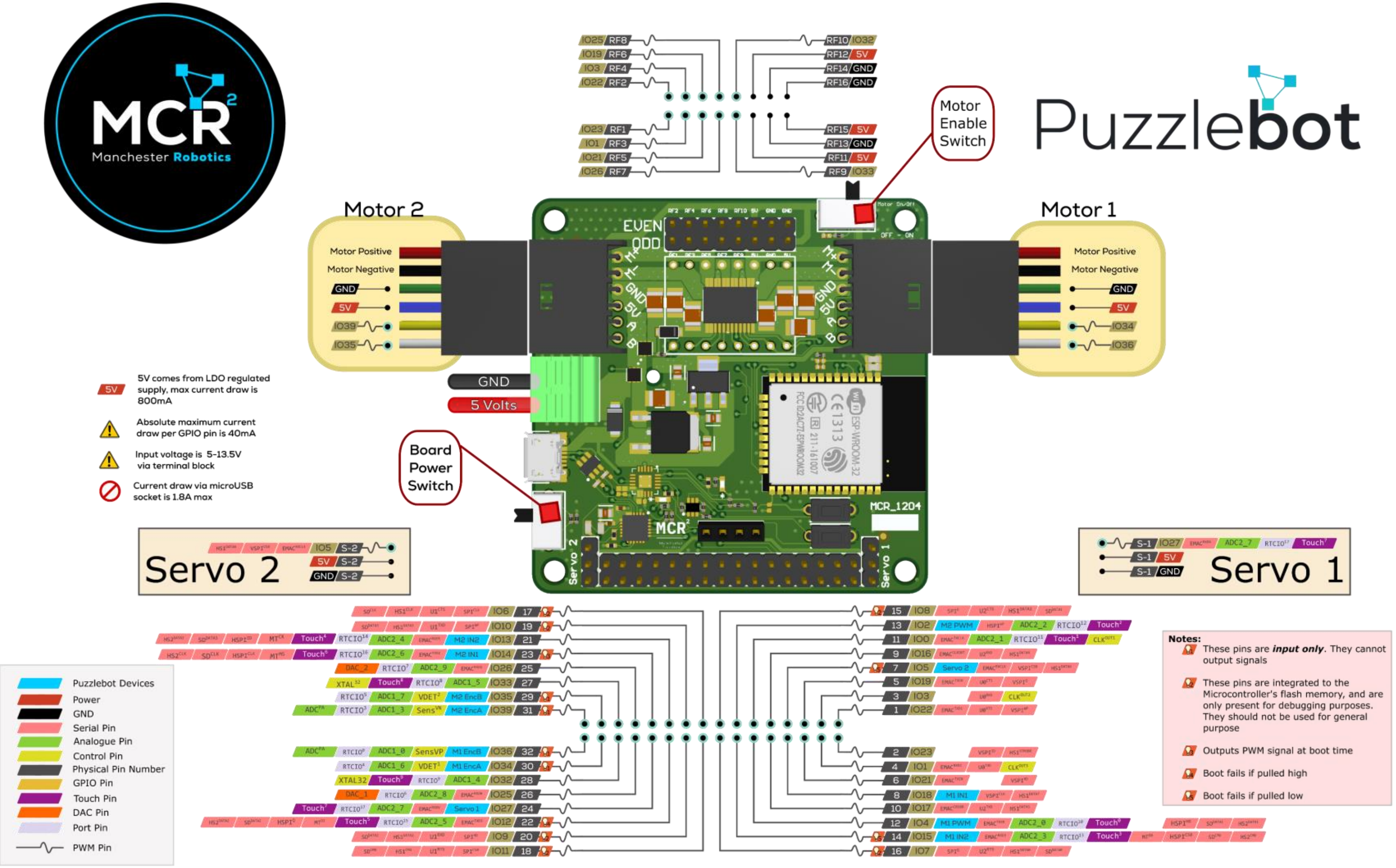


Breadboard

LED

150Ω Resistor

Hackerboard Pinout

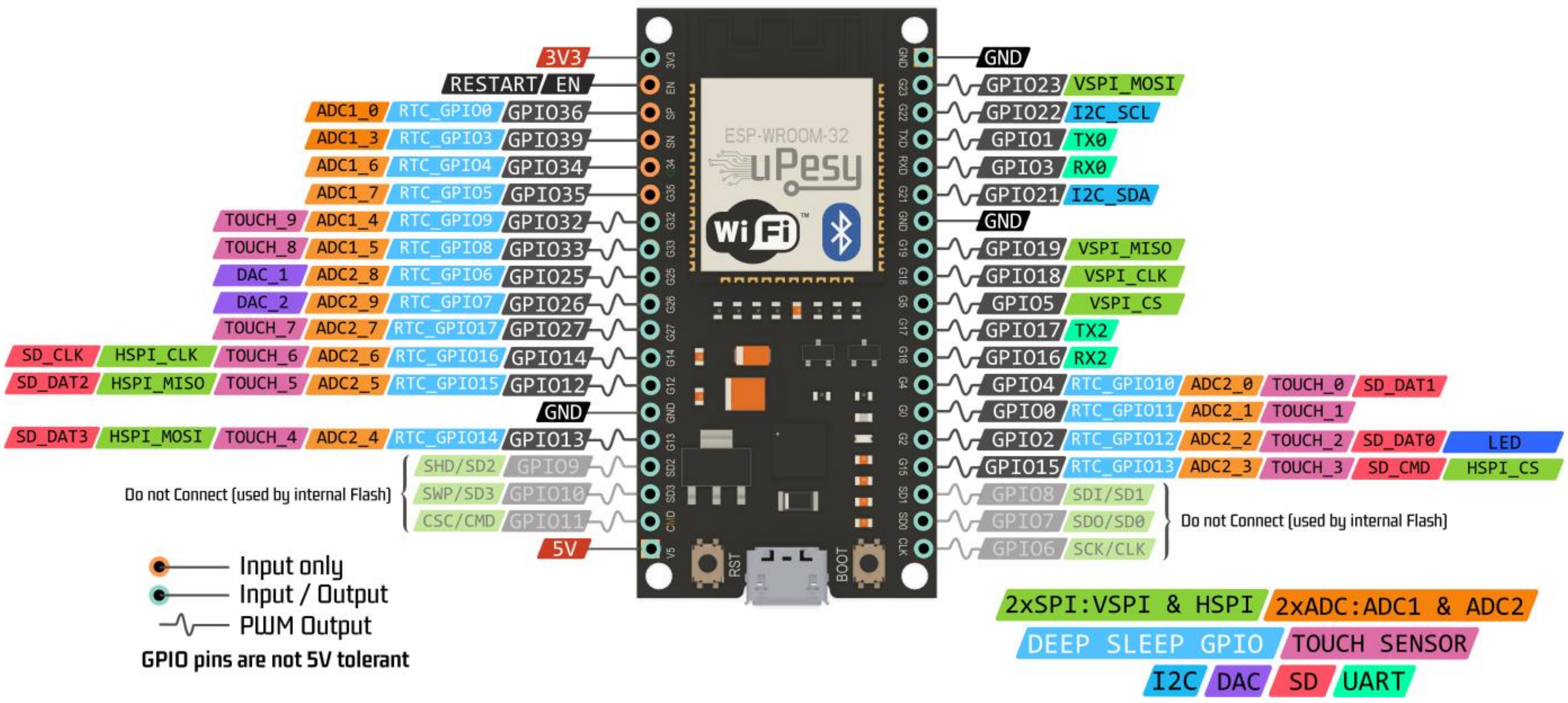


https://www.manchester-robotics.com

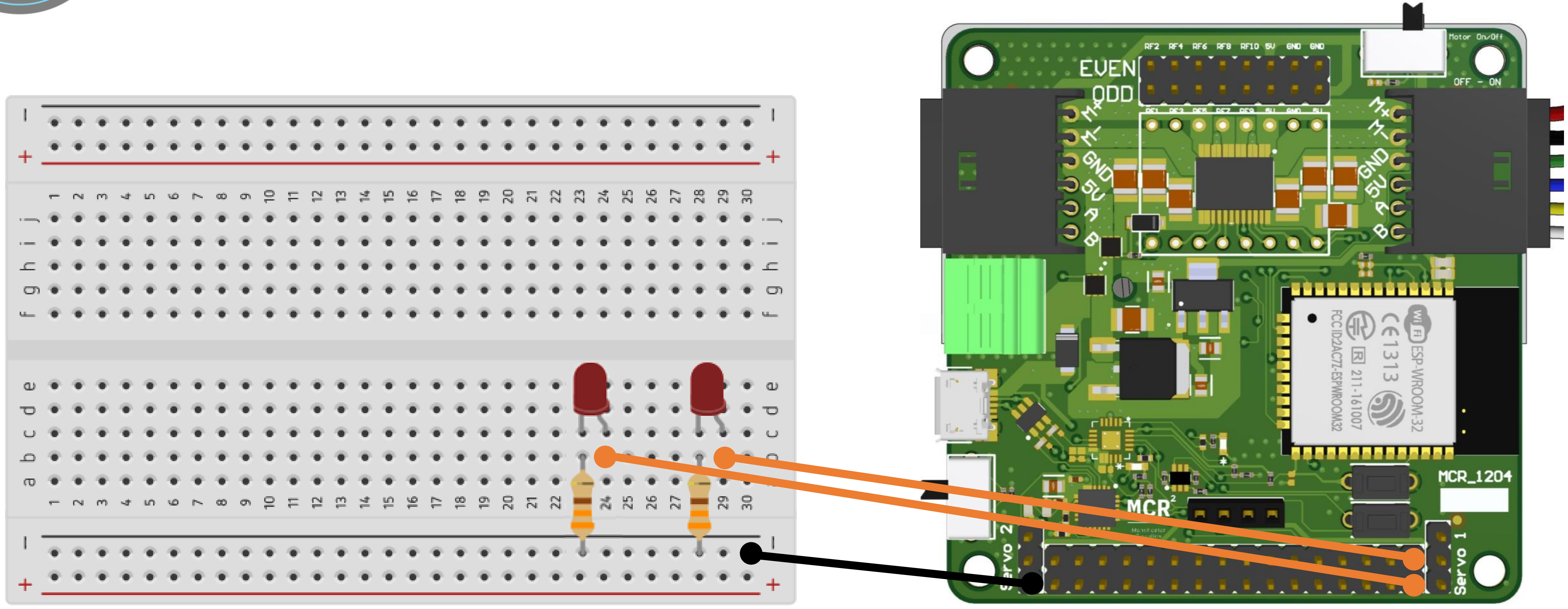
01 FEB 2022
v1

ESP32 Pinout

ESP32 Wroom DevKit Full Pinout



Connections



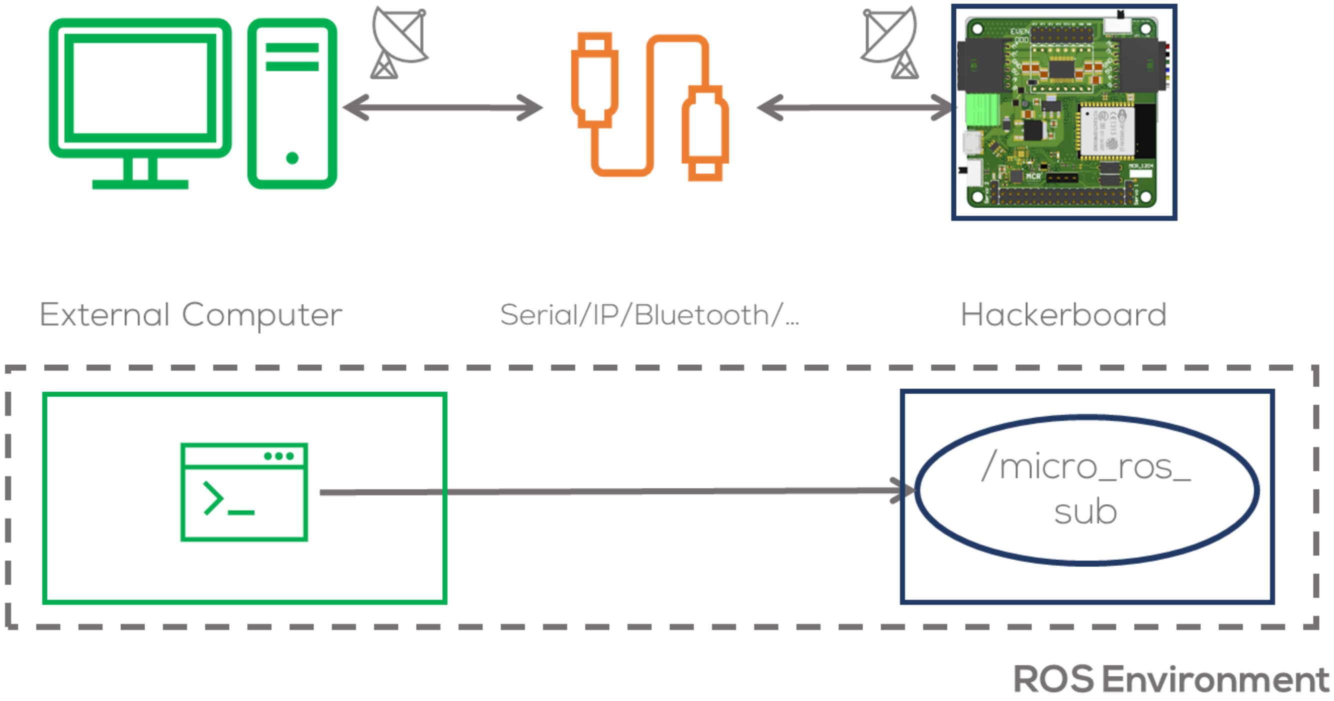
GND

Pin 22

Pin 23

Description

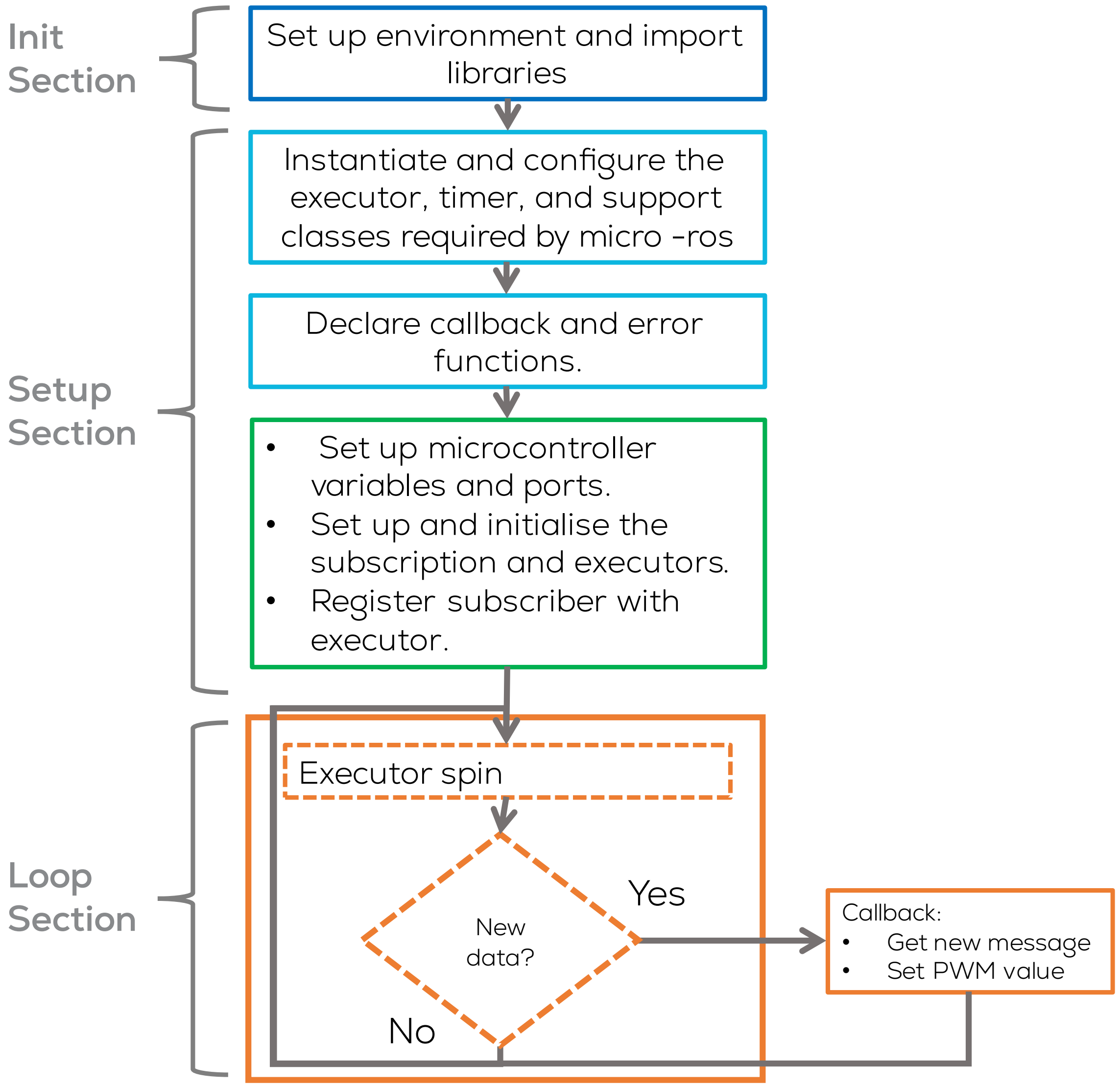
- In this activity, a node running a simple subscriber will be made.
- This node will run inside the microcontroller and will communicate with the computer via UART.
- The node will subscribe to a Float32 message.
- This activity will be divided into two parts. The first part involves the Arduino IDE to program the MCU.
- The second part involves the commands required to connect to the board to the computer.



Description

Objective

- The objective of this activity is for the microcontroller to control the brightness of an LED from the computer using ROS2.
- To this end, the microcontroller must set a node called "micro_ros_sub_node" and subscribe to a topic "micro_ros_sub".
- The computer will send a value in the range [0,1].
- The microcontroller must set a PWM Value from [0%,100%] to dim the LED.



Subscriber set up

Configuring a Subscriber in Micro-ROS

What is a Subscriber?

- A subscriber listens to a ROS 2 topic and processes incoming messages.
- In Micro-ROS, subscribers receive messages asynchronously using a callback function.

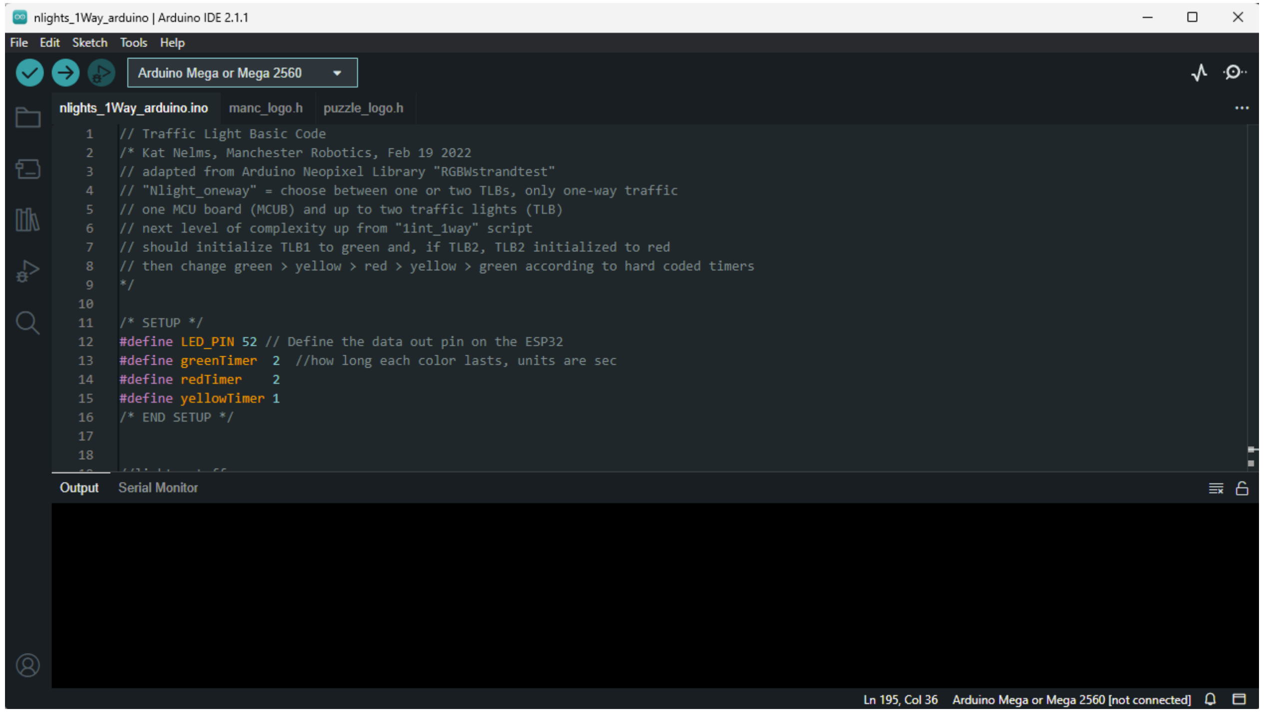
1. Define the Message Type and a subscriber object.
2. Create the Subscriber Callback
3. Initialize/configure the Subscriber
   - ON_NEW_DATA: Executes the callback only when new data arrives.
   - ALWAYS: Executes the callback on every executor cycle, even if no new data is available.
4. Add the Subscriber to the Executor
5. Process the Executor in loop()

```
#include <std_msgs/msg/int32.h>    //Predefined ROS 2 message type
//Declare Messages to be used
std_msgs__msg__Int32 msg;  //Defines a message of type int32.
rcl_subscription_t subscriber;
...
void subscription_callback(const void * msg_in) {
    const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msg_in;
    Serial.print("Received: ");
    Serial.println(msg->data);
}

void setup() {
  // create subscriber
rclc_subscription_init_default(
    &subscriber,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "micro_ros_subscriber_topic");
  // create executor
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  // Register suscription with executor
  RCCHECK(rclc_executor_add_subscription(&executor, &subscriber, &msg,
&subscription_callback, ON_NEW_DATA));
}

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

Activity 2

Instructions (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 or Arduino Due
    - For ESP32 select ESP32 Arduino › DOIT ESP32 DEVKIT V1
- For this activity, no LED on Pin 22 is required



Activity 2

```
// Include Libraries to be used
#include <micro_ros_arduino.h>    //micro-ros-arduino library
#include <rcl/rcl.h>              //Core ROS 2 Client Library (RCL) for node management.
#include <rcl/error_handling.h>   //Error handling utilities for Micro-ROS.
#include <rclc/rclc.h>            //Micro-ROS Client library for embedded devices.
#include <rclc/executor.h>        //Micro-ROS Executor to manage callbacks
#include <std_msgs/msg/float32.h>   //Predefined ROS 2 message type
#include <stdio.h>                //Standard I/O library for debugging.

//Declare nodes to be used
rcl_node_t node;            //Represents a ROS 2 Node running on the microcontroller.

//Instantiate executor and its support classes
rclc_executor_t executor;   //Manages task execution (timers, callbacks, etc.).
rclc_support_t support;     //Data structure that holds the execution context of Micro-ROS, including its communication state, memory management, and initialization data.
rcl_allocator_t allocator;  //Manages memory allocation.

//Declare Subscribers to be used
rcl_subscription_t subscriber;

//Declare timers to be used
rcl_timer_t timer;          //Creates a timer to execute functions at intervals.

//Declare Messages to be used
std_msgs__msg__Float32 msg;  //Defines a message of type float32.

//Define Macros to be used
//Executes fn and goes to error_loop() function if fn fails.
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
// Executes fn, but ignores failures.
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}

//Specifies GPIO pin 13 for controlling an LED
#define LED_PIN 23 //Define LED_PIN
#define PWM_PIN 22 //DEFINE PWM_PIN
#define PWM_FRQ 5000 //Define PWM Frequency
#define PWM_RES 8  //Define PWM Resolution
#define PWM_CHNL 0    //Define Channel
#define MSG_MIN_VAL 0 //Define min input value
#define MSG_MAX_VAL 1 //Define max input value
//Variables to be used
float pwm_set_point = 0.0;

//Define Error Functions
void error_loop(){
  while(1){
    digitalWrite(LED_PIN, !digitalRead(LED_PIN)); // Toggle LED state
    delay(100); // Wait 100 milliseconds
  }
}

//Define callbacks
void subscription_callback(const void * msgin)
{
  //Get the message received and store it on the message msg
  const std_msgs__msg__Float32 * msg = (const std_msgs__msg__Float32 *)msgin;
  pwm_set_point = constrain(msg->data, MSG_MIN_VAL, MSG_MAX_VAL);
  ledcWrite(PWM_CHNL, (uint32_t) (pow(2, PWM_RES) * (pwm_set_point  / 1.0)));
}
```

Activity 2

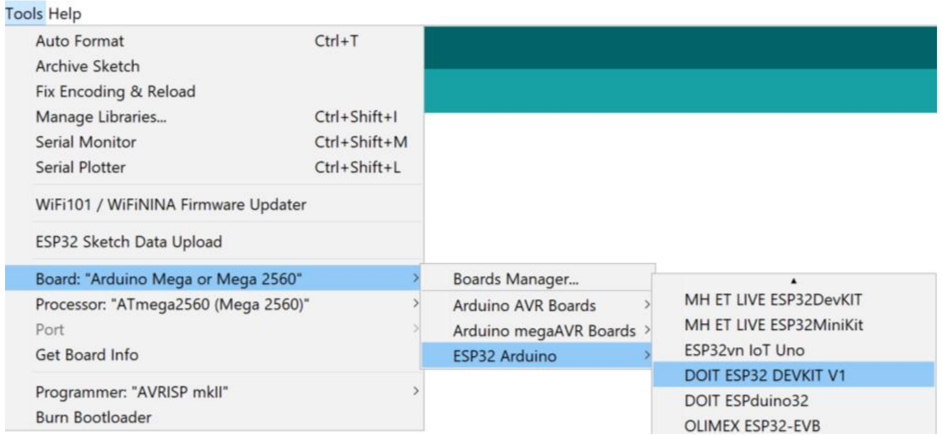
```
//Setup
void setup() {
  // Initializes communication between ESP32 and the ROS 2 agent (Serial).
  set_microros_transports();
  //Setup Microcontroller Pins
  pinMode(LED_PIN, OUTPUT);
  pinMode(PWM_PIN, OUTPUT);
  digitalWrite(LED_PIN, HIGH);
  ledcSetup(PWM_CHNL, PWM_FRQ, PWM_RES);  //Setup the PWM
  ledcAttachPin(PWM_PIN, PWM_CHNL);       //Setup Attach the Pin to the Channel
  //Connection delay
  delay(2000);
  //Initializes memory allocation for Micro-ROS operations.
  allocator = rcl_get_default_allocator();

  //Creates a ROS 2 support structure to manage the execution context.
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

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

  // create subscriber
  RCCHECK(rclc_subscription_init_default(
    &subscriber,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "micro_ros_sub"));

  // create executor
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  // Register suscription with executor
  RCCHECK(rclc_executor_add_subscription(&executor, &subscriber, &msg,
&subscription_callback, ON_NEW_DATA));
}
```

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

Activity 2

Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for **Hackerboard** is the same)
    - For ESP32 select ESP32 Arduino › DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.

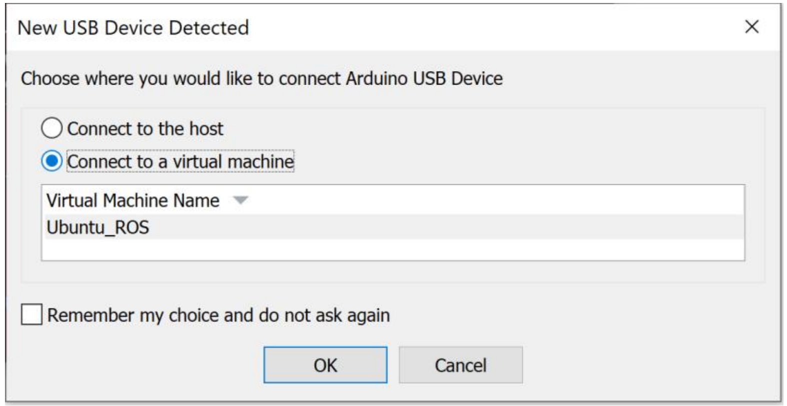
- The following message should be displayed:

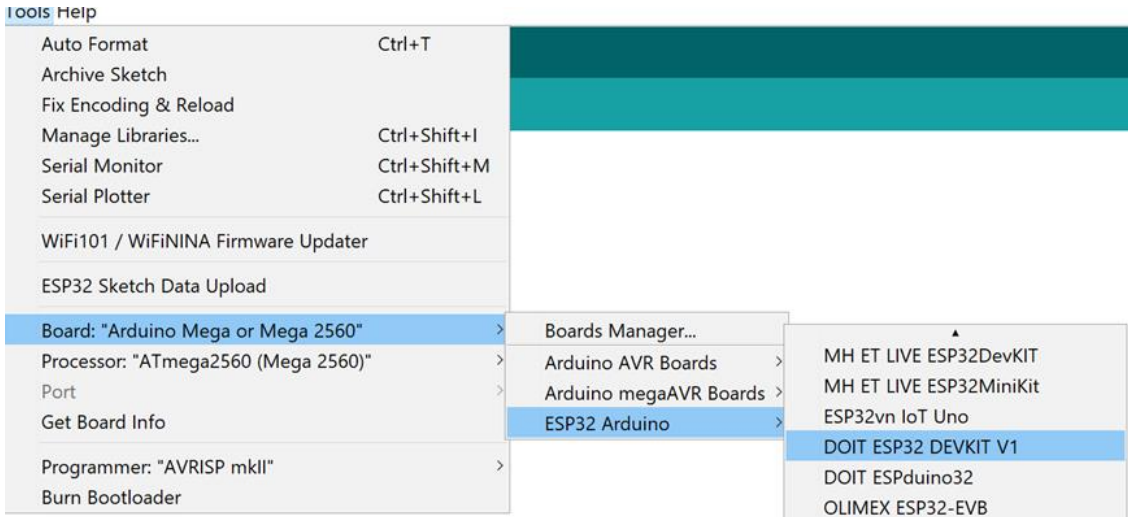
```
Done compiling.
Sketch uses 9424 bytes (3%) of program storage space. Maximum is 253952 bytes.
Global variables use 1826 bytes (22%) of dynamic memory, leaving 6366 bytes for local variables. Maximum is 8192 bytes.
```

Activity 2

Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools›Port
  - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
  - If in WSL, follow the steps on the presentation: “MCR2_Micro_ROS_Installation”.
- Select the board to be used Tools -> ESP32 Arduino › DOIT ESP32 DEVKIT V1



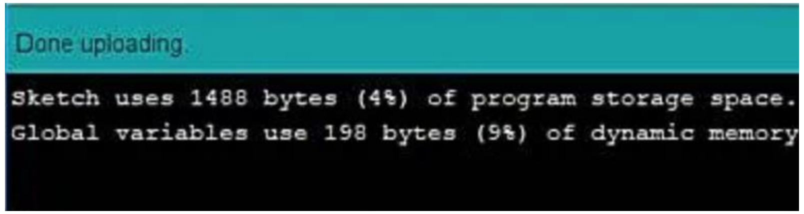


Activity 2

Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.

- The following message should appear o the IDE



Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
  - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*
sudo chmod 666 /dev/ttyUSB*
```

Activity 2

Test (Computer)

1. Open a terminal and type the following.

```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```

2. Reset the ESP32 (pressing the reset button) to reconnect to the computer (agent waiting timeout).

```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent mi
cro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info     | TermiosAgentLinux.cpp
 | init                     | running...
 | fd: 3
[1737636692.138467] info     | Root.cpp                 |
set_verbose_level          | logger setup             |
verbose_level: 4
[1737636698.665805] info     | Root.cpp                 |
create_client              | create                   |
client_key: 0x0C9424D2, session_id: 0x81
[1737636698.666090] info     | SessionManager.hpp |
establish_session          | session established      |
client_key: 0x0C9424D2, address: 0
[1737636698.702311] info     | ProxyClient.cpp          |
create_participant         | participant created      |
client_key: 0x0C9424D2, participant_id: 0x000(1)
[1737636698.721583] info     | ProxyClient.cpp          |
create_topic               | topic created            |
client_key: 0x0C9424D2, topic_id: 0x000(2), particip
ant_id: 0x000(1)
[1737636698.733573] info     | ProxyClient.cpp          |
create_publisher           | publisher created        |
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

- Open another terminal and type the following.

```
$ ros2 topic list
```

```
mario@MarioPC:~$ ros2 topic list
/micro_ros_sub
/parameter_events
/rosout
```

- Publish to the topic "/micro_ros_subscriber_topic"

```
$ ros2 topic pub /micro_ros_sub std_msgs/msg/Float32 "data: 0.3"
```

```
mario@MarioPC:~$ ros2 topic pub /micro_ros_sub std_m
sgs/msg/Float32 "data: 0.3"
publisher: beginning loop
publishing #1: std_msgs.msg.Float32(data=0.3)

publishing #2: std_msgs.msg.Float32(data=0.3)

publishing #3: std_msgs.msg.Float32(data=0.3)

publishing #4: std_msgs.msg.Float32(data=0.3)

publishing #5: std_msgs.msg.Float32(data=0.3)
```

Micro-ROS Serial Communication

Activity 2.2: Subscriber w/reconnection

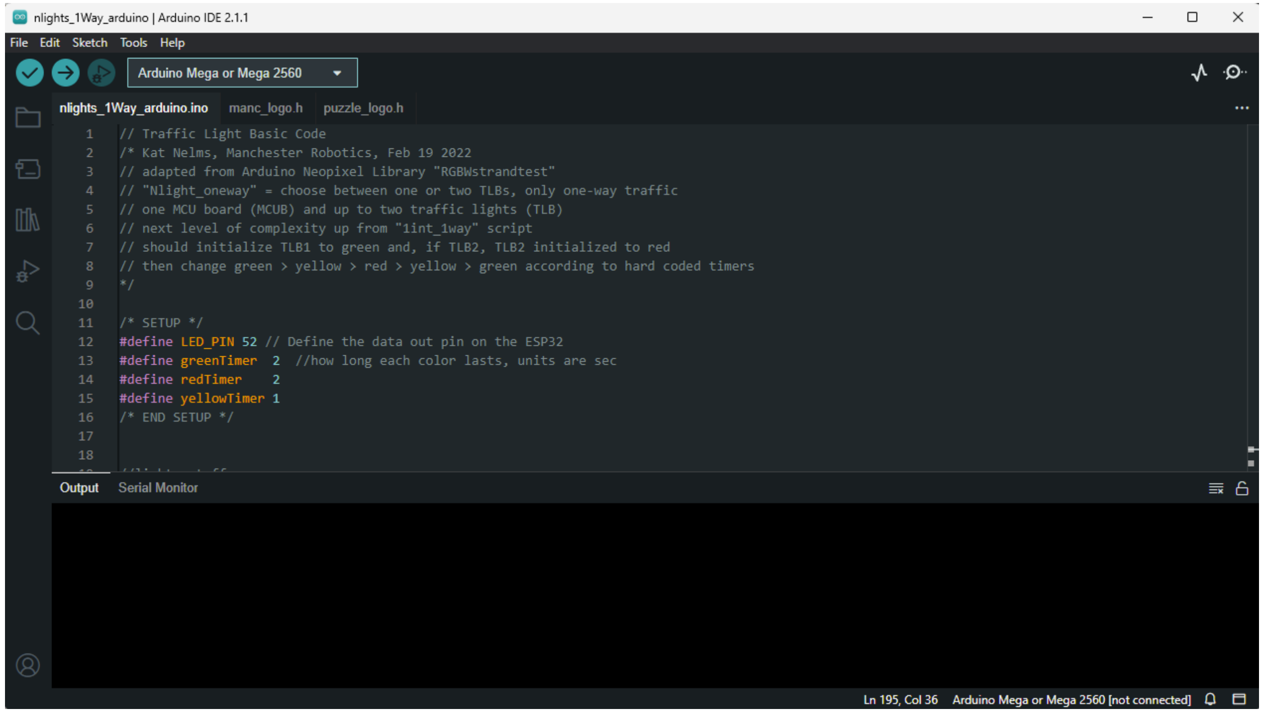
{Learn, Create, Innovate};

Manchester Robotics

Activity 2.2

Instructions (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 or Arduino Due
    - For ESP32 select ESP32 Arduino › DOIT ESP32 DEVKIT V1
- Open the previous Activity and modify it according to the following slides. Modifications are highlighted in Black.
- For this activity, no LED on Pin 22 is required



Activity 2.2

```
// Include Libraries to be used
#include <micro_ros_arduino.h>    //micro-ros-arduino library
#include <rcl/rcl.h>              //Core ROS 2 Client Library (RCL) for node management.
#include <rcl/error_handling.h>   //Error handling utilities for Micro-ROS.
#include <rclc/rclc.h>            //Micro-ROS Client library for embedded devices.
#include <rclc/executor.h>        //Micro-ROS Executor to manage callbacks
#include <std_msgs/msg/float32.h>   //Predefined ROS 2 message type
#include <rmw_microros/rmw_microros.h> //ROS Middleware for Micro-ROS, provides functions for interfacing Micro-ROS with DDS.
#include <stdio.h>                //Standard I/O library for debugging.

//Declare nodes to be used
rcl_node_t node;            //Represents a ROS 2 Node running on the microcontroller.

//Instantiate executor and its support classes
rclc_executor_t executor;   //Manages task execution (timers, callbacks, etc.).
rclc_support_t support;     //Data structure that holds the execution context of Micro-ROS, including its communication state, memory management, and initialization data.
rcl_allocator_t allocator;  //Manages memory allocation.

//Declare Subscribers to be used
rcl_subscription_t subscriber;

//Declare Messages to be used
std_msgs__msg__Float32 msg;  //Defines a message of type float32.
```

```
//Define Macros to be used
//Executes fn and returns false if it fails.
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){return false;}}
// Executes fn, but ignores failures.
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}
//  Executes a given statement (X) periodically every MS milliseconds
#define EXECUTE_EVERY_N_MS(MS, X)  do { \
  static volatile int64_t init = -1; \
  if (init == -1) { init = uxr_millis();} \
  if (uxr_millis() - init > MS) { X; init = uxr_millis();} \
} while (0)\

//Defines State Machine States
enum states {
  WAITING_AGENT,
  AGENT_AVAILABLE,
  AGENT_CONNECTED,
  AGENT_DISCONNECTED
} state;

//Specifies GPIO pin 13 for controlling an LED
#define PWM_PIN 22 //DEFINE PWM_PIN
#define PWM_FRQ 5000 //Define PWM Frequency
#define PWM_RES 8  //Define PWM Resolution
#define PWM_CHNL 0    //Define Channel
#define MSG_MIN_VAL 0 //Define min input value
#define MSG_MAX_VAL 1 //Define max input value
//Variables to be used
float pwm_set_point = 0.0;
```

Activity 2.2

```
//Create entity functions
bool create_entities();
void destroy_entities();

//Define callbacks
void subscription_callback(const void * msgin)
{
  //Get the message received and store it on the message msg
  const std_msgs__msg__Float32 * msg = (const std_msgs__msg__Float32
*)msgin;
  pwm_set_point = constrain(msg->data, MSG_MIN_VAL, MSG_MAX_VAL);
  ledcWrite(PWM_CHNL, (uint32_t) (pow(2, PWM_RES) * (pwm_set_point  /
1.0)));
}

//Setup
void setup() {
  set_microros_transports(); // Initializes communication between ESP32 and
the ROS 2 agent (Serial).
  //Setup Microcontroller Pins
  pinMode(PWM_PIN, OUTPUT);
  ledcSetup(PWM_CHNL, PWM_FRQ, PWM_RES);  //Setup the PWM
  ledcAttachPin(PWM_PIN, PWM_CHNL); //Setup Attach the Pin to the Channel
}
```

```
void loop() {
  switch (state) {

    case WAITING_AGENT:
      EXECUTE_EVERY_N_MS(500, state = (RMW_RET_OK == rmw_uros_ping_agent(100,
1)) ? AGENT_AVAILABLE : WAITING_AGENT;);
      break;

    case AGENT_AVAILABLE:
      state = (true == create_entities()) ? AGENT_CONNECTED : WAITING_AGENT;
      if (state == WAITING_AGENT) {
        destroy_entities();
      };
      break;

    case AGENT_CONNECTED:
      EXECUTE_EVERY_N_MS(200, state = (RMW_RET_OK == rmw_uros_ping_agent(100,
1)) ? AGENT_CONNECTED : AGENT_DISCONNECTED;);
      if (state == AGENT_CONNECTED) {
        rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));
      }
      break;

    case AGENT_DISCONNECTED:
      destroy_entities();
      state = WAITING_AGENT;
      break;

    default:
      break;
  }
}
```

Activity 2.2

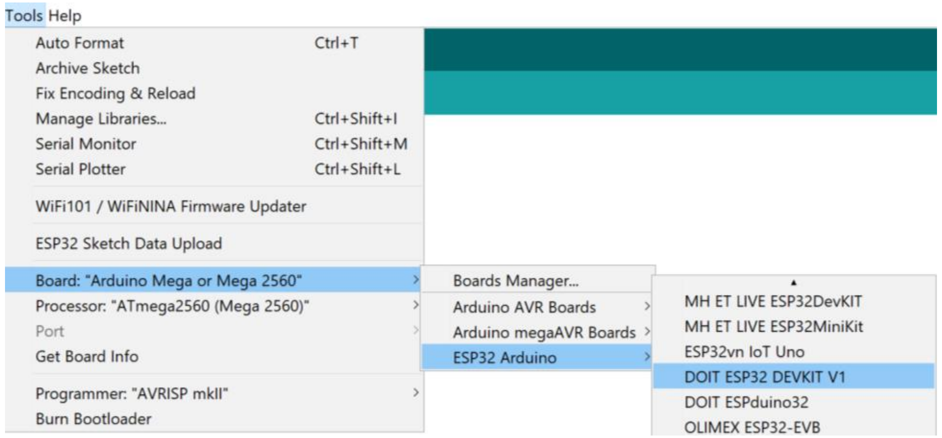
```
bool create_entities()
{
   //Initializes memory allocation for Micro-ROS operations.
  allocator = rcl_get_default_allocator();

   //Creates a ROS 2 support structure to manage the execution context.
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

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

  // create subscriber
  RCCHECK(rclc_subscription_init_default(
    &subscriber,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "micro_ros_sub"));

  // create zero initialised executor (no configured) to avoid memory
problems
  executor = rclc_executor_get_zero_initialized_executor();
  // Initializes the Micro-ROS Executor, which manages tasks and callbacks.
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  // Register suscription with executor
  RCCHECK(rclc_executor_add_subscription(&executor, &subscriber, &msg,
&subscription_callback, ON_NEW_DATA));

  return true;
}
```

```
void destroy_entities()
{
  rmw_context_t * rmw_context = rcl_context_get_rmw_context(&support.context);
  (void) rmw_uros_set_context_entity_destroy_session_timeout(rmw_context, 0);

  rcl_subscription_fini(&subscriber, &node);
  rclc_executor_fini(&executor);
  rcl_node_fini(&node);
  rclc_support_fini(&support);
}
```

Activity

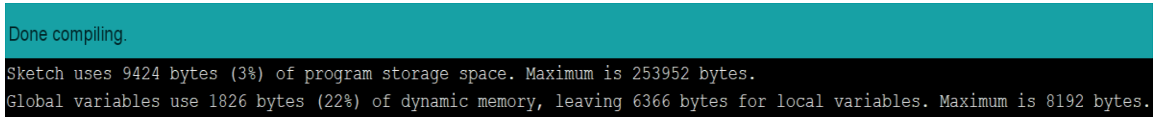
Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for **Hackerboard** is the same)
  - For ESP32 select ESP32 Arduino › DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.

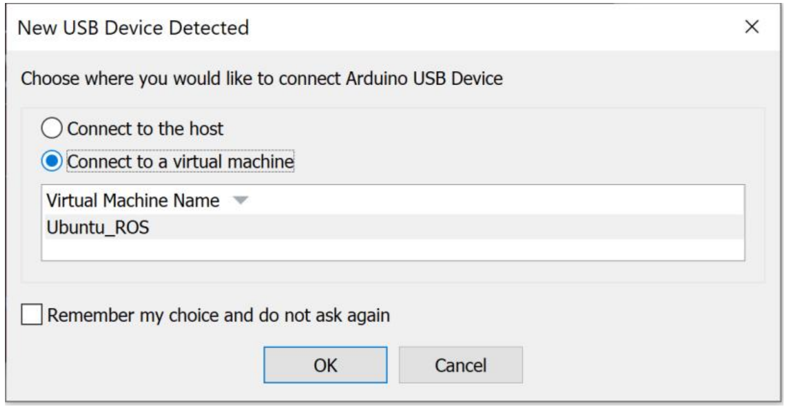
- The following message should be displayed:

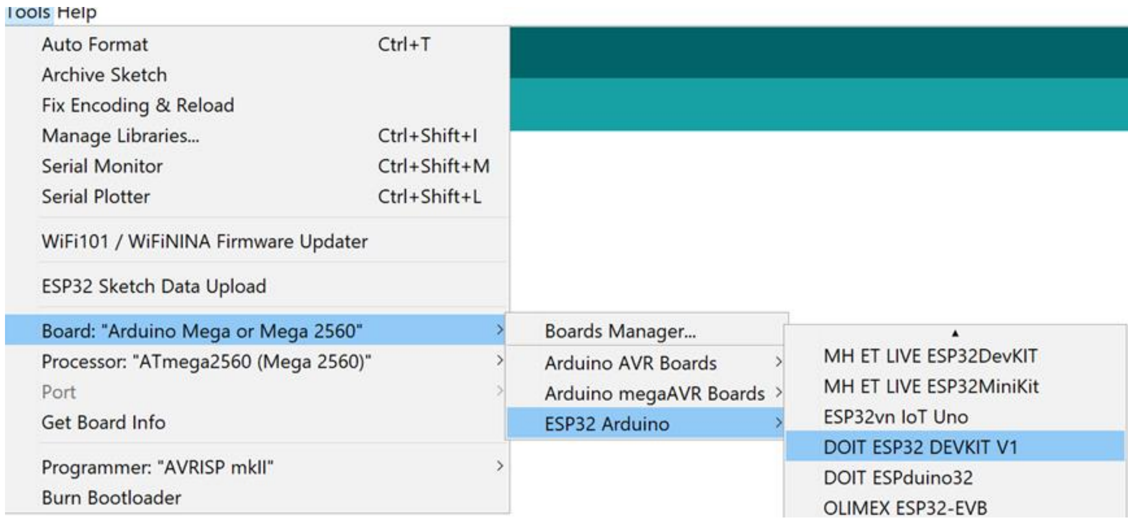


Activity

Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools›Port
  - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
  - If in WSL, follow the steps on the presentation: “MCR2_Micro_ROS_Installation”.
- Select the board to be used Tools -> ESP32 Arduino › DOIT ESP32 DEVKIT V1



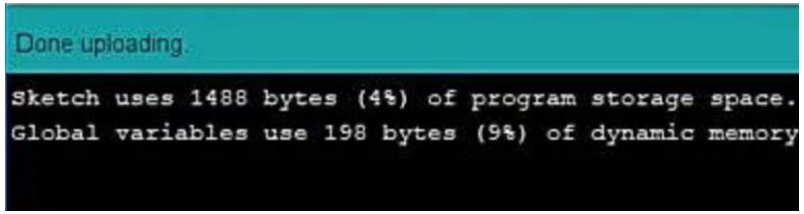


Activity

Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.

- The following message should appear o the IDE


Done uploading.

Sketch uses 1488 bytes (4%) of program storage space.
Global variables use 198 bytes (9%) of dynamic memory

Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
  - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*
sudo chmod 666 /dev/ttyUSB*
```

Activity 2.2

Test (Computer)

1. Open a terminal and type the following.

```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```

```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent mi
cro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info     | TermiosAgentLinux.cpp
 | init                     | running...
 | fd: 3
[1737636692.138467] info     | Root.cpp             |
set_verbose_level         | logger setup           |
verbose_level: 4
[1737636698.665805] info     | Root.cpp             |
create_client             | create                 |
client_key: 0x0C9424D2, session_id: 0x81
[1737636698.666090] info     | SessionManager.hpp |
establish_session         | session established    |
client_key: 0x0C9424D2, address: 0
[1737636698.702311] info     | ProxyClient.cpp      |
create_participant        | participant created    |
client_key: 0x0C9424D2, participant_id: 0x000(1)
[1737636698.721583] info     | ProxyClient.cpp      |
create_topic              | topic created          |
client_key: 0x0C9424D2, topic_id: 0x000(2), particip
ant_id: 0x000(1)
[1737636698.733573] info     | ProxyClient.cpp      |
create_publisher          | publisher created      |
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

- Open another terminal and type the following.

```
$ ros2 topic list
```

```
mario@MarioPC:~$ ros2 topic list
/micro_ros_sub
/parameter_events
/rosout
```

- Publish to the topic "/micro_ros_sub"

```
$ ros2 topic pub /micro_ros_sub std_msgs/msg/Float32 "data: 0.3"
```

```
mario@MarioPC:~$ ros2 topic pub /micro_ros_sub std_m
sgs/msg/Float32 "data: 0.3"
publisher: beginning loop
publishing #1: std_msgs.msg.Float32(data=0.3)

publishing #2: std_msgs.msg.Float32(data=0.3)

publishing #3: std_msgs.msg.Float32(data=0.3)

publishing #4: std_msgs.msg.Float32(data=0.3)

publishing #5: std_msgs.msg.Float32(data=0.3)
```

Micro-ROS Serial Communication

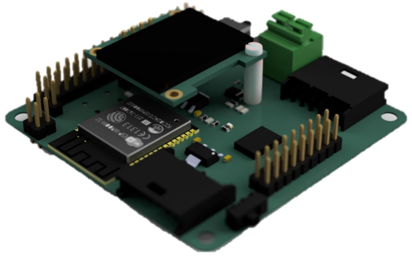
Activity 3: Publisher and Subscriber

{Learn, Create, Innovate};

Manchester Robotics

Requirements

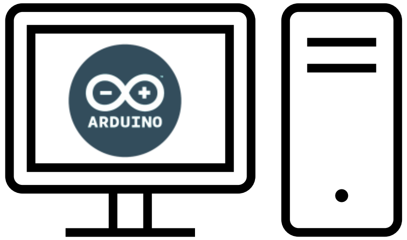
- The following activity is based on the example tutorial found in the provided micro-ROS libraries.
- This activity requires Arduino IDE to be installed and configured as shown in “MCR2_Micro_ROS_Installation”.
- Requirements:
  - Microcontroller
  - Computer
  - micro-usb to USB cable (Data)
  - Breadboard
  - LED
  - 1 x 150 Ohm Resistor
  - 1x 10kOhm Resistor
  - NO Pushbutton



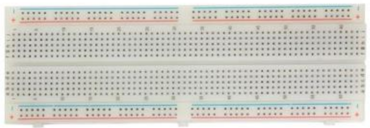
Hackerboard

Or

ESP32 board



Computer



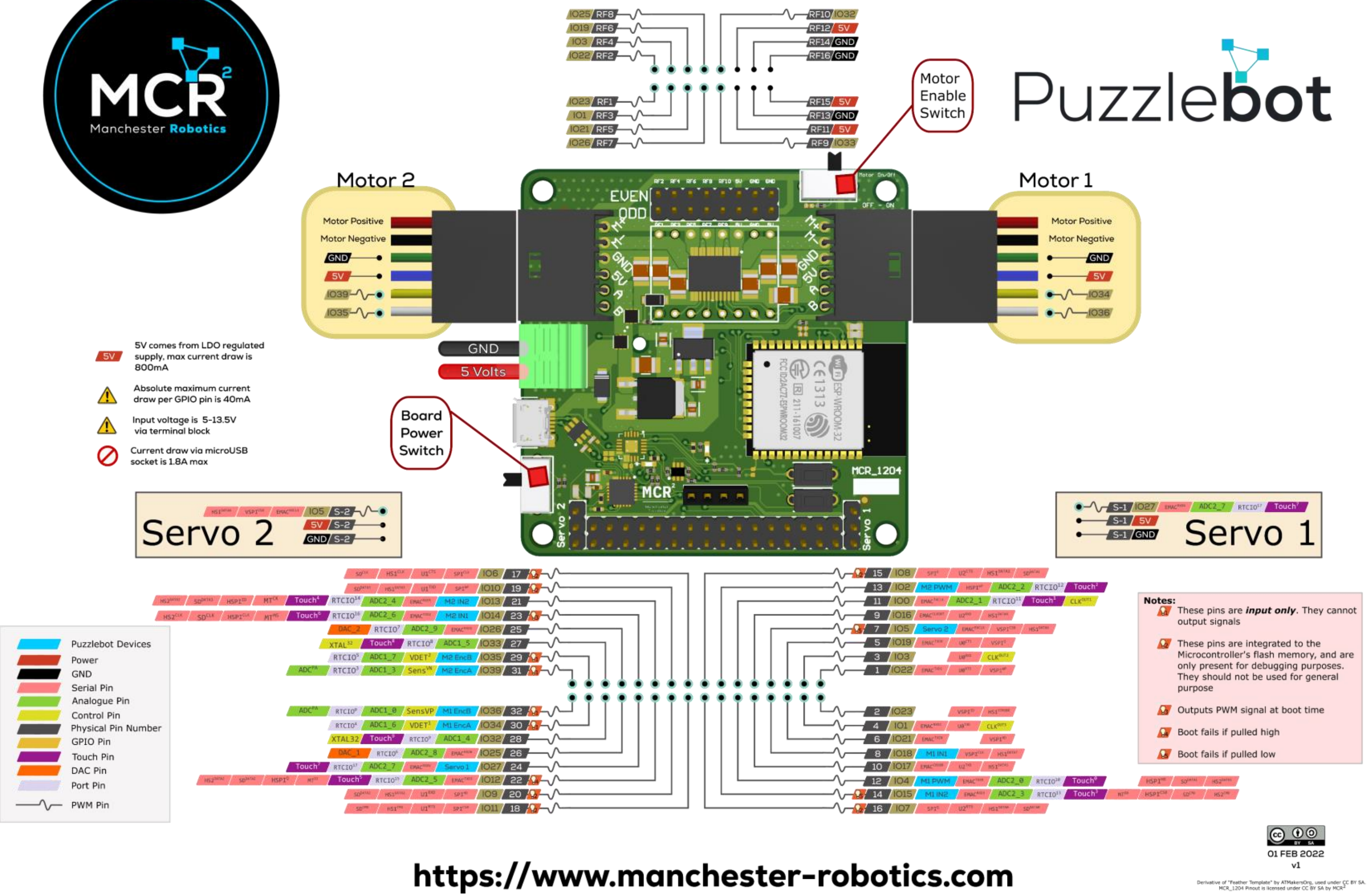
Breadboard

LED

Push Button (Normally open)

$10\mathrm{k}\Omega$, 150Ω Resistor

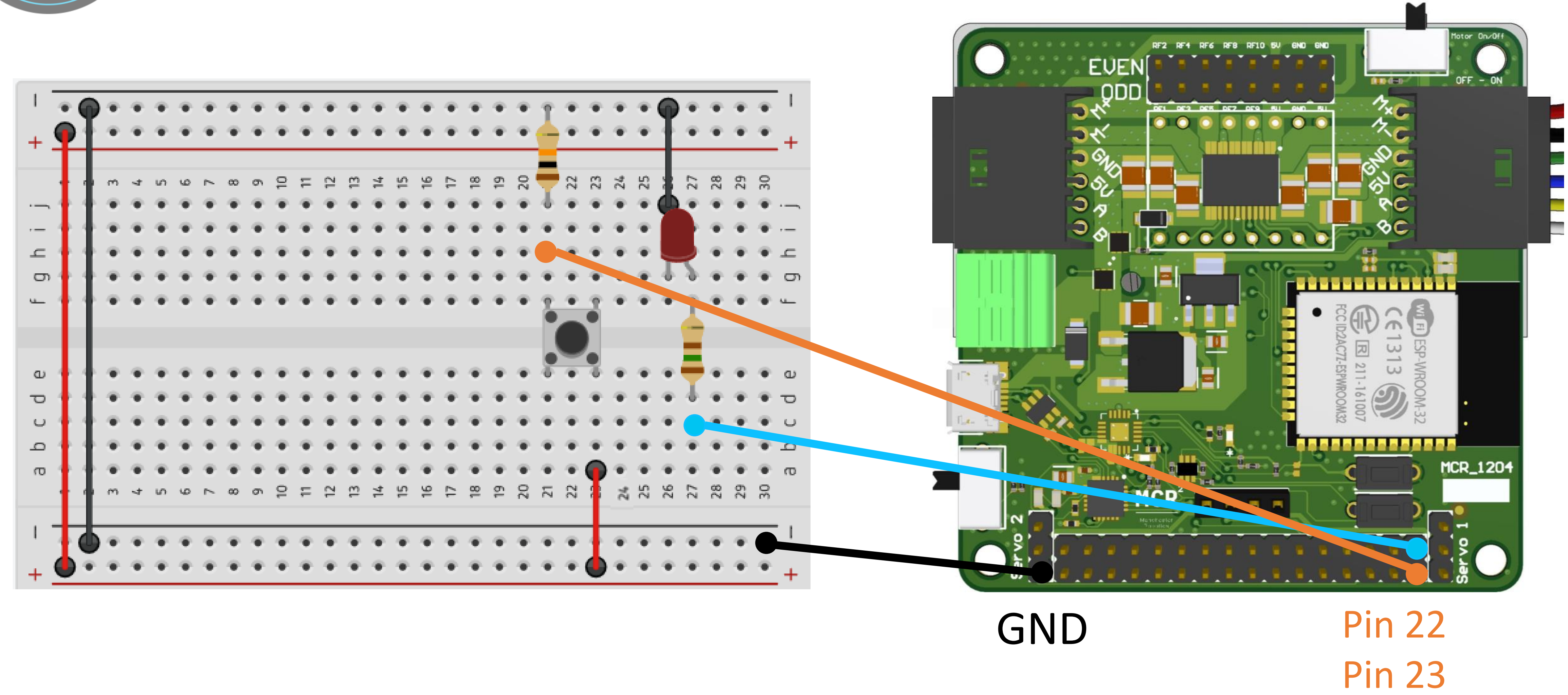
Hackerboard Pinout



https://www.manchester-robotics.com

ESP32 Pinout

Connections



Description

Objective

- The microcontroller should detect the state of a pushbutton to enable or disable LED brightness control.
  - If active, the user can control the LED brightness from a host computer via ROS 2.
  - If inactive, the LED brightness remains fixed at 0% (off).

Implementation:

- The system must be implemented using ROS 2 and Micro-ROS.
- The button's state should only update after a **complete press-release cycle.**

ROS 2 Node & Communication

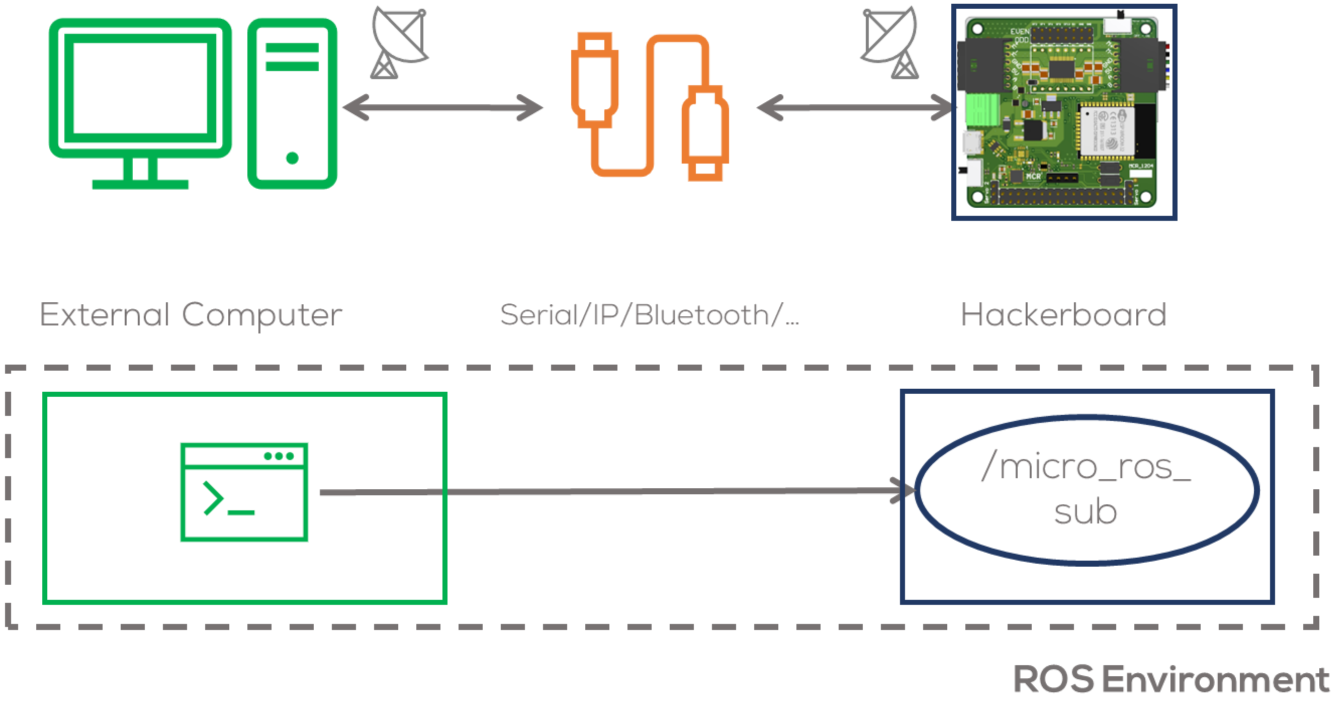
- The microcontroller will run a ROS 2 node called "esp32_button_led_node".
- Publish button state using a String ("active" or "inactive“) to “button_state” topic (every 100ms).
- Subscribe to “led_brightness” (0 to 1) to control LED intensity.

System Behavior

- The pushbutton state is debounced to prevent false activations.
- The host computer sends a value in the range [0,1] to adjust LED brightness. 0 = LED OFF (0%), 1 = LED FULL BRIGHTNESS (100%)

Description

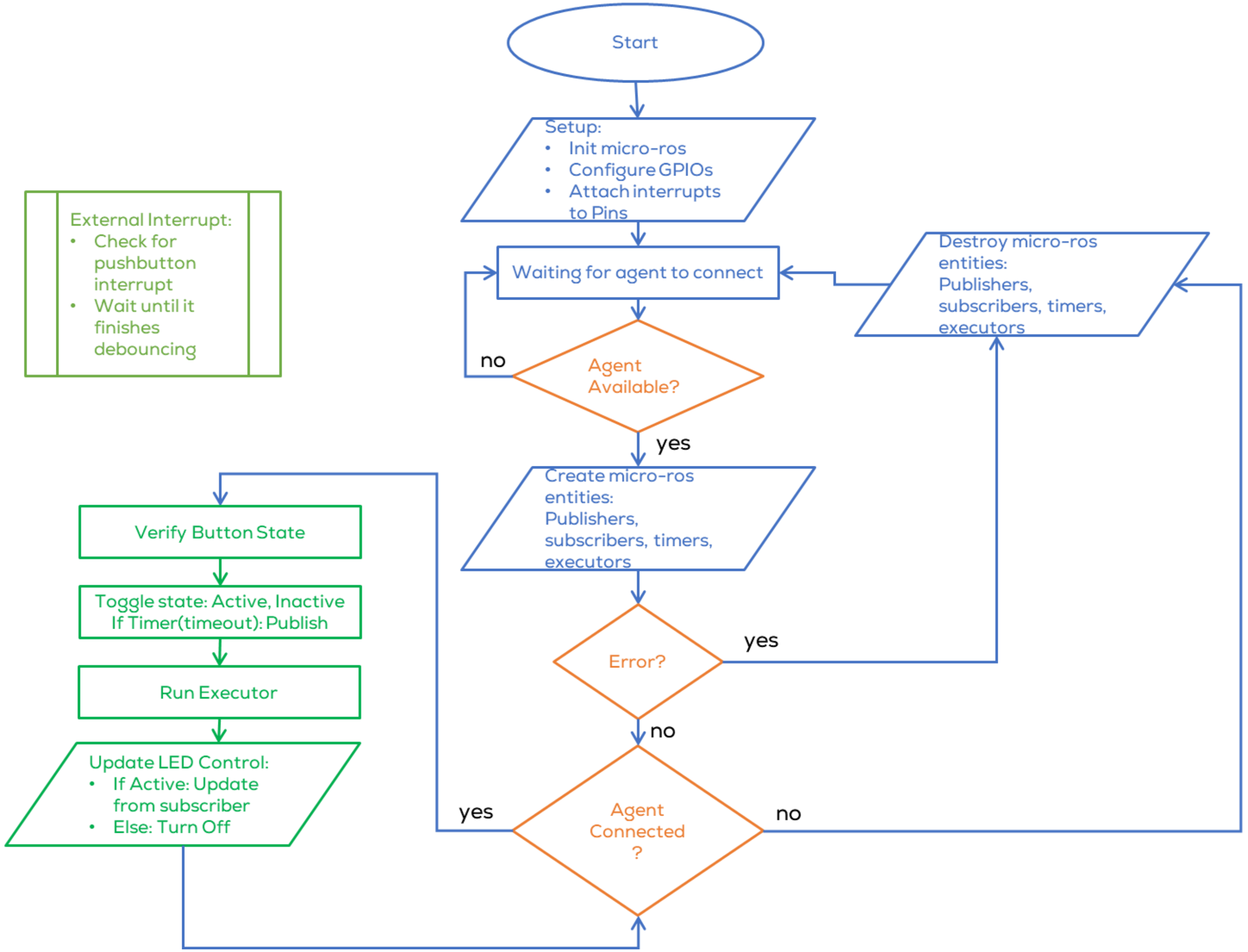
- In this activity, a node running a publisher and a subscriber will be made.
- This node will run inside the microcontroller and will communicate with the computer via UART.
- The node will subscribe to a Float32 message and publish a String message.
- This activity will be divided into two parts. The first part involves the Arduino IDE in programming the MCU.
- The second part involves the commands to connect the board to the computer.



Activity 3: Publisher and Subscriber

Introduction

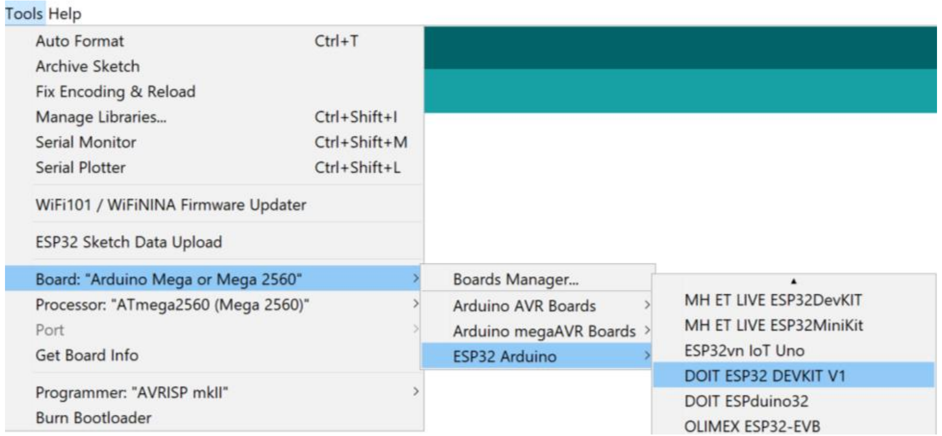
- This activity is focused on integrating the previous concepts into a bigger project.
- Verify the correct wiring and connections before powering up the system.
- Open the file “publisher_subscriber.ino”
- The code for the ESP32 is too large, a flowchart shown describes how the code works.



Activity

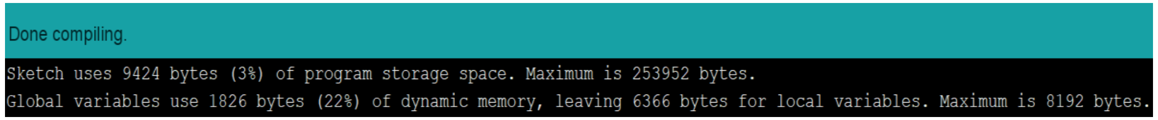
Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for **Hackerboard** is the same)
    - For ESP32 select ESP32 Arduino › DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.

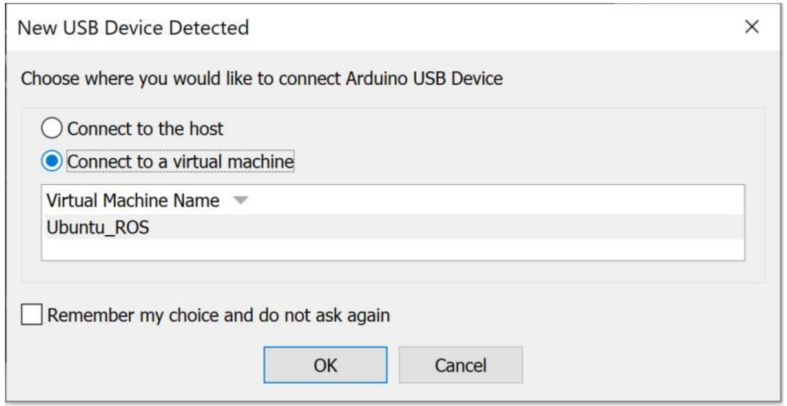
- The following message should be displayed:

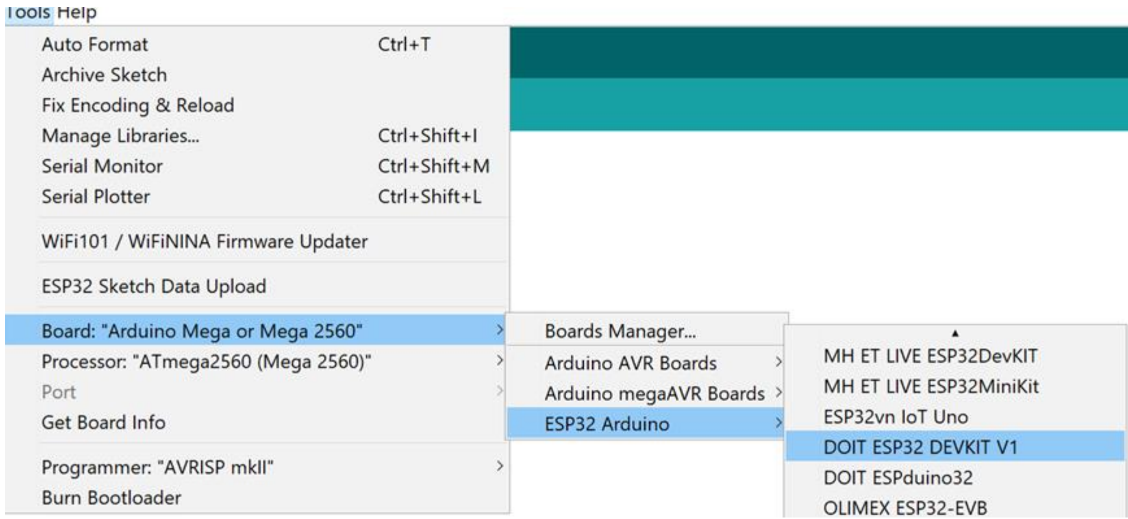


Activity

Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools›Port
  - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
  - If in WSL, follow the steps on the presentation: “MCR2_Micro_ROS_Installation”.
- Select the board to be used Tools -> ESP32 Arduino › DOIT ESP32 DEVKIT V1



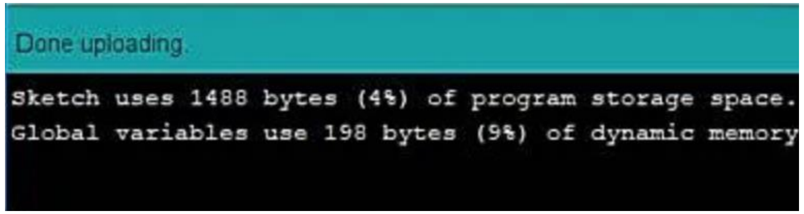


Activity

Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.

- The following message should appear o the IDE



Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
  - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*
sudo chmod 666 /dev/ttyUSB*
```

Activity 3

Test (Computer)

1. Open a terminal and type the following.

```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```

```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent mi
cro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info     | TermiosAgentLinux.cpp
 | init                     | running...
 | fd: 3
[1737636692.138467] info     | Root.cpp              |
set_verbose_level         | logger setup          |
verbose_level: 4
[1737636698.665805] info     | Root.cpp              |
create_client             | create                |
client_key: 0x0C9424D2, session_id: 0x81
[1737636698.666090] info     | SessionManager.hpp |
establish_session         | session established   |
client_key: 0x0C9424D2, address: 0
[1737636698.702311] info     | ProxyClient.cpp       |
create_participant        | participant created   |
client_key: 0x0C9424D2, participant_id: 0x000(1)
[1737636698.721583] info     | ProxyClient.cpp       |
create_topic              | topic created         |
client_key: 0x0C9424D2, topic_id: 0x000(2), particip
ant_id: 0x000(1)
[1737636698.733573] info     | ProxyClient.cpp       |
create_publisher          | publisher created     |
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

2. Open another terminal and type the following.

```
$ ros2 topic list
```

```
mario@MarioPC:~$ ros2 topic list
/button_state
/led_brightness
/parameter_events
/rosout
```

3. Echo the topic "/button_state" . Press the Button!

```
$  ros2 topic echo /button_state
```

```
mario@MarioPC:~$ ros2 topic echo /button_state
data: inactive
---
data: inactive
---
data: inactive
---
```

4. Publish to the LED on the topic "/led_brightness "

```
$ ros2 topic pub /led_brightness std_msgs/msg/Float32 "data: 0.5"
```

```
mario@MarioPC:~$ ros2 topic pub /led_brightness std_
msgs/msg/Float32 "data: 0.5"
publisher: beginning loop
publishing #1: std_msgs.msg.Float32(data=0.5)

publishing #2: std_msgs.msg.Float32(data=0.5)

publishing #3: std_msgs.msg.Float32(data=0.5)
```

Micro-ROS Communication

Micro-ros Wi-Fi communication

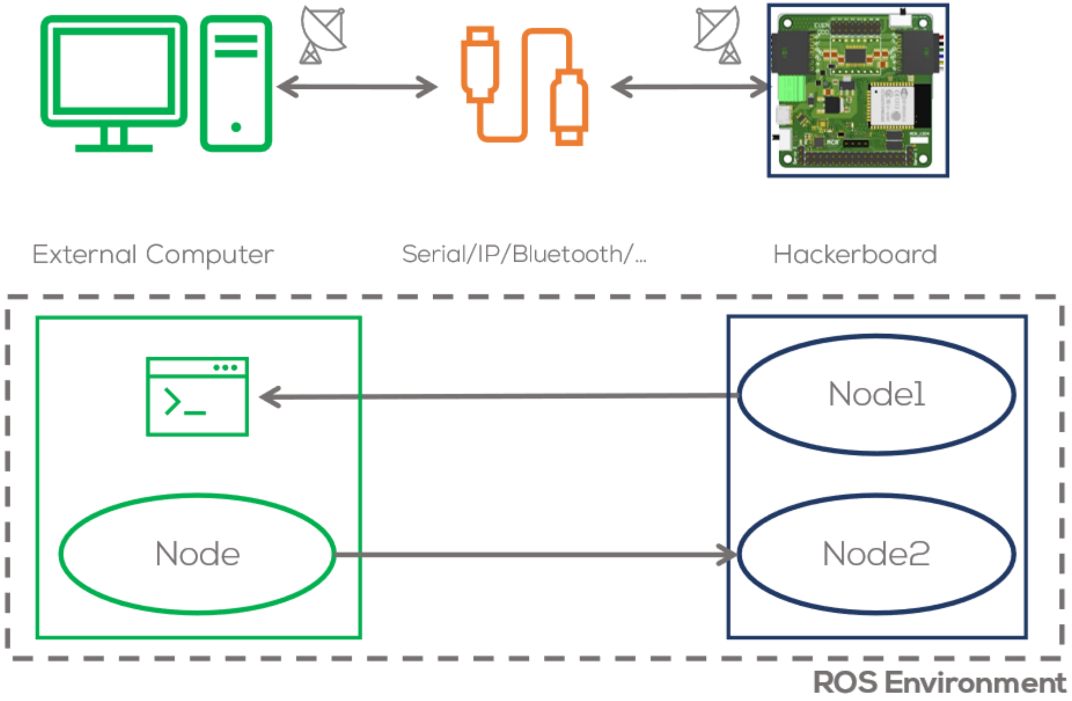
{Learn, Create, Innovate};

MCR2
Manchester Robotics

Wi-Fi communication

Introduction

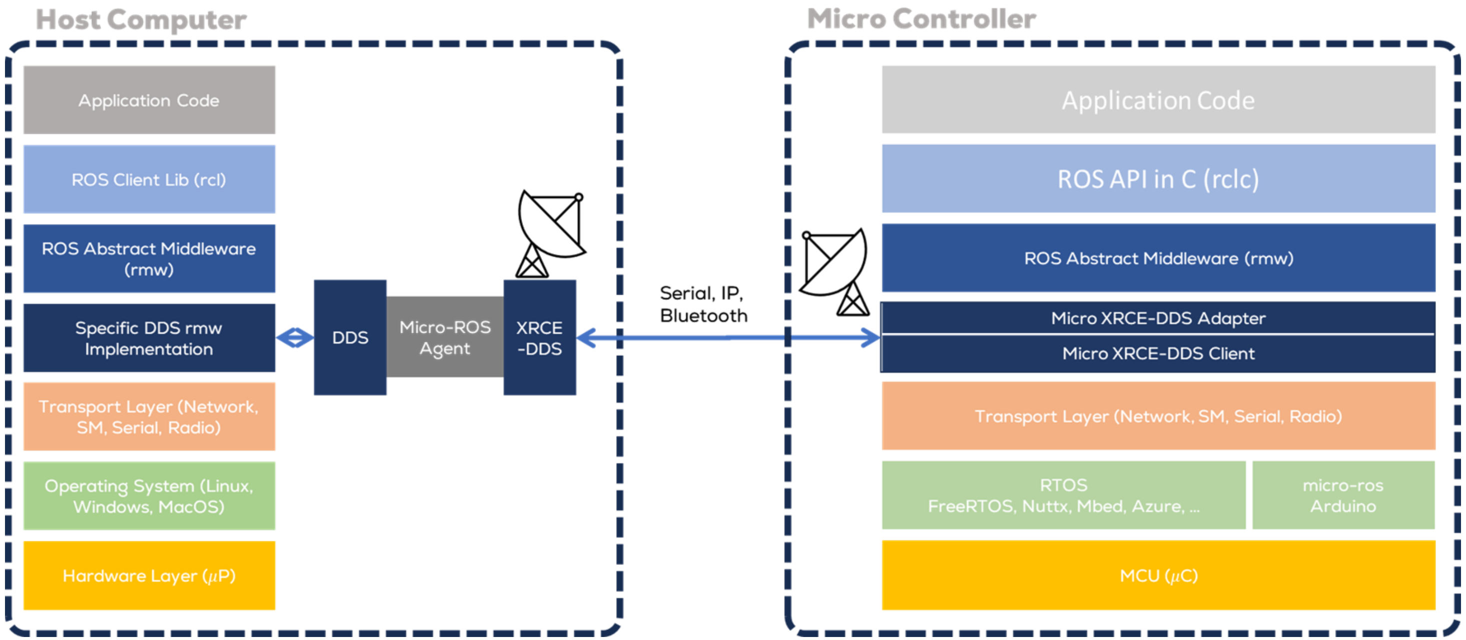
- Embedded systems. Typically communicate via serial, communication.
- Micro-ROS enables communication between embedded systems and a ROS 2 network using various wired and wireless protocols.
- Micro-ROS provides flexibility in choosing transport layers for data exchange.
- Selecting the right transport depends on hardware capabilities, network requirements, and application constraints.



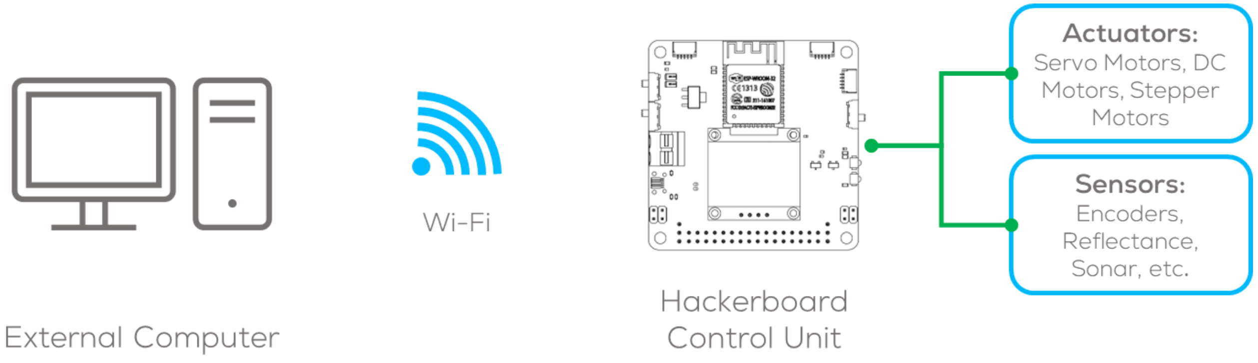
Wi-Fi communication

Micro-ros transports

- Micro-ROS transports define how data is sent between embedded devices and the ROS 2 agent.
- They enable communication between microcontrollers and the ROS 2 agent running on a host.
- The user can define its own communication protocol (micro-ros transports)



Wi-Fi communication



Types of predefined Micro-ROS Transports

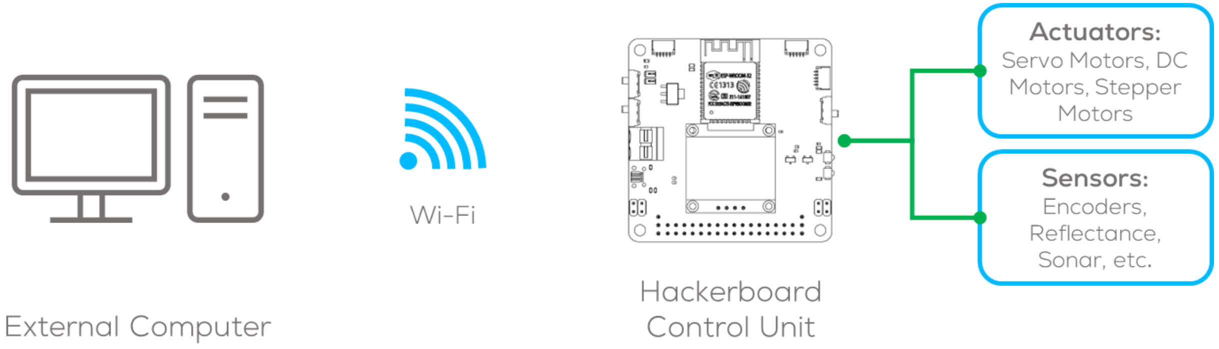
- Serial (UART, USB): Direct connection via serial ports (e.g., USB, UART).
- Wi-Fi (UDP/TCP): Wireless communication over a network.
- Ethernet: Wired high-speed communication.
- CAN Bus: Common in automotive and industrial applications.

This section will focus on Wi-Fi communication using micro-ros.

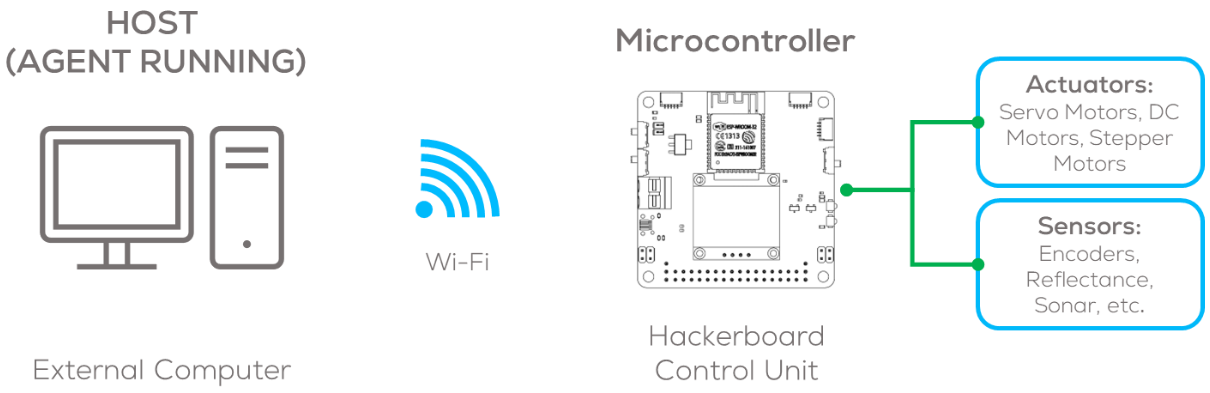
Wi-Fi communication

Wi-Fi for Micro-ROS

- Provides wireless communication between microcontrollers and a ROS 2 host.
- Enables remote control and monitoring of embedded robotic systems.
- Suitable for mobile robots that require untethered operation.



Wi-Fi communication



How Wi-Fi Transport Works

The microcontroller connects to a Wi-Fi network or generates and access point (AP).

The Micro-ROS Agent runs on a host computer.

Data is exchanged between the microcontroller and the host using UDP or TCP.

In this section the microcontroller will be set as an access point and communicate using Wi-Fi to the agent.

Micro-ROS Serial Communication

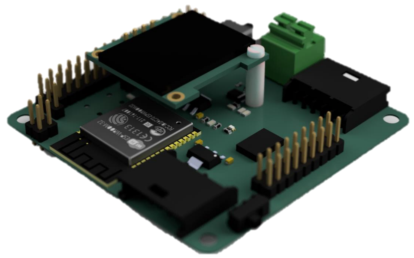
Activity 4: Wi-Fi Publisher

`{Learn, Create, Innovate};`

Manchester Robotics

Requirements

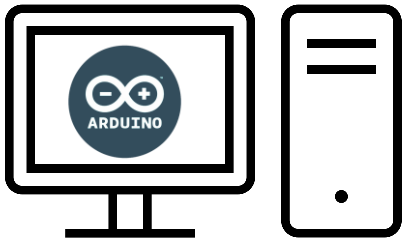
- The following activity is based on the example tutorial found in the provided micro-ROS libraries.
- This activity requires Arduino IDE to be installed and configured as shown in "MCR2_Micro_ROS_Installation".
- Requirements:
  - Microcontroller
  - Computer
  - micro-usb to USB cable (Data)



Hackerboard

Or

ESP32 board

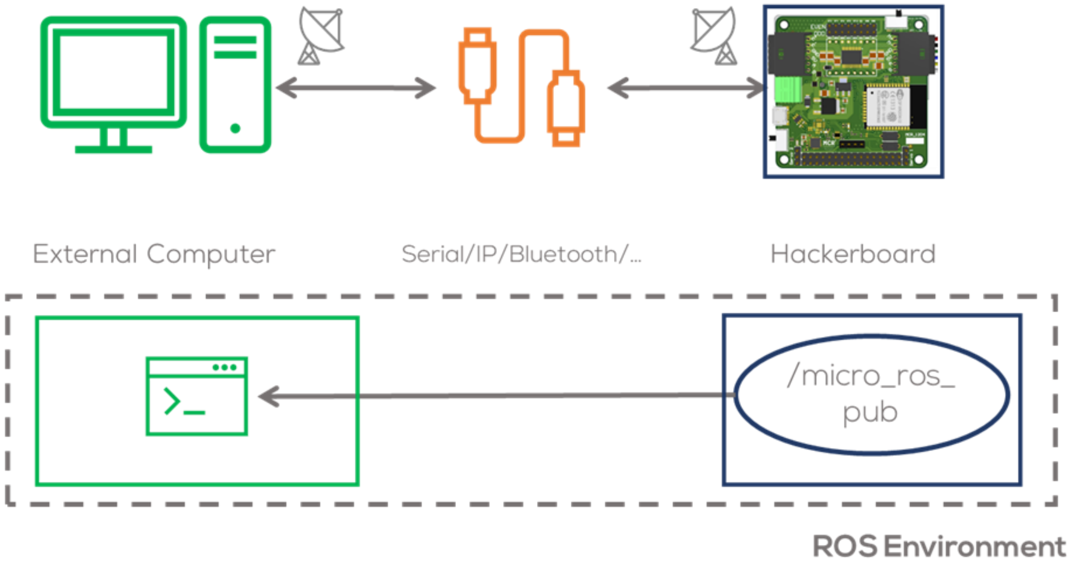


Computer

Description

Objective

- In this activity, a node running a simple publisher inside the microcontroller will be declared.
- This node will run inside the microcontroller and will communicate with the computer via Wi-Fi.
- The node will publish a simple int32 message.
- This activity will be divided into two parts. The first part involves the Arduino IDE to program the MCU.
- The second part involving the commands required to connect to the board to the computer.



Activity 4

```
#include <micro_ros_arduino.h>  // Micro-ROS library for Arduino
#include <WiFi.h>               // Wi-Fi library for ESP32
#include <WiFiUdp.h>            // UDP communication over Wi-Fi

#include <stdio.h>              // Standard I/O library
#include <rcl/rcl.h>            // Core ROS 2 Client Library (RCL) for node
management
#include <rcl/error_handling.h> // Error handling utilities
#include <rclc/rclc.h>          // Micro-ROS client library for embedded
devices
#include <rclc/executor.h>      // Micro-ROS Executor to manage callbacks
#include <rmw_microros/rmw_microros.h> // ROS Middleware for Micro-ROS

#include <std_msgs/msg/int32.h>  // Predefined ROS 2 message type (integer
messages)


// Macros for Error Checking
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc !=
RCL_RET_OK)){return false;}}  // Return false on failure
#define RMCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc !=
RMW_RET_OK)){error_loop();}}  // Enter error loop on failure
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc !=
RCL_RET_OK)){}}

// Macro for executing a function every N milliseconds
#define EXECUTE_EVERY_N_MS(MS, X)  do { \
  static volatile int64_t init = -1; \
  if (init == -1) { init = uxr_millis();} \
  if (uxr_millis() - init > MS) { X; init = uxr_millis();} \
} while (0)\
```

```
// Micro-ROS entities
rclc_support_t support;    // Holds the execution context of Micro-ROS
rclc_executor_t executor;  // Manages task execution (timers, callbacks,
etc.)
rcl_allocator_t allocator;  // Memory allocation manager

rcl_node_t node;       // Represents a ROS 2 Node running on the
microcontroller
rcl_timer_t timer;     // Timer for periodic message publishing
rcl_publisher_t publisher;  // Publisher for sending messages to ROS 2

std_msgs__msg__Int32 msg;  // Integer message type

micro_ros_agent_locator locator; // Stores connection details for Micro-ROS
Agent
// Static IP configuration for ESP32 Access Point
IPAddress local_ip = {10, 16, 1, 1};
IPAddress gateway = {10, 16, 1, 1};
IPAddress subnet = {255, 255, 255, 0};
// Wi-Fi credentials (ESP32 acting as an Access Point)
const char* ssid     = "ESP32-Access-Point";
const char* password = "123456789";
// Micro-ROS Agent configuration (host machine)
const char* agent_ip = "10.16.1.2";
const int agent_port = 8888;
```

Activity 4

```c
// Enum representing different connection states of the microcontroller
enum states {
  WAITING_AGENT,        // Waiting for a connection to the Micro-ROS agent
  AGENT_AVAILABLE,      // Agent found, trying to establish communication
  AGENT_CONNECTED,      // Connected to the agent, publishing messages
  AGENT_DISCONNECTED    // Lost connection, trying to reconnect
} state;


// Function that gets called if there is a failure in initialization
void error_loop(){
  while(1){
    // Toggle LED state
      printf("Failed initialisation. Aborting.\n");  // Print error message
    // Wait for 100 milliseconds before retrying
    delay(100);
  }
}

// Timer callback function, runs periodically to publish messages
void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
  (void) last_call_time;
  if (timer != NULL) {
    rcl_publish(&publisher, &msg, NULL); // Publish message to ROS 2 topic
    msg.data++; // Increment message data for the next cycle
  }
}
```

```c
// Function to create Micro-ROS entities (node, publisher, timer)
bool create_entities()
{
  // Initialize Micro-ROS support
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // Create ROS 2 node
  RCCHECK(rclc_node_init_default(&node, "int32_publisher_rclc", "",
&support));

  // Create a best-effort publisher (non-reliable, no message history) (QoS)
  RCCHECK(rclc_publisher_init_best_effort(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "std_msgs_msg_Int32"));

  // Create a timer to publish messages every 1000ms (1 second)
  const unsigned int timer_timeout = 1000;
  RCCHECK(rclc_timer_init_default(
    &timer,
    &support,
    RCL_MS_TO_NS(timer_timeout),
    timer_callback));

  // Initialize Executor (handles timer callbacks)
  executor = rclc_executor_get_zero_initialized_executor();
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  RCCHECK(rclc_executor_add_timer(&executor, &timer));
  return true;  // Return true if all entities are successfully created
}
```

Activity 4

```
// Function to clean up Micro-ROS entities when disconnected
void destroy_entities()
{
  rmw_context_t * rmw_context =
rcl_context_get_rmw_context(&support.context);
  (void)
rmw_uros_set_context_entity_destroy_session_timeout(rmw_context, 0);

  rcl_publisher_fini(&publisher, &node);
  rcl_timer_fini(&timer);
  rclc_executor_fini(&executor);
  rcl_node_fini(&node);
  rclc_support_fini(&support);
}
```

```
// Setup function - Runs once when ESP32 starts
void setup() {

  // Initialize memory allocator
  allocator = rcl_get_default_allocator();

  // Set up Micro-ROS agent connection details
  locator.address.fromString(agent_ip);
  locator.port = agent_port;

  // Set up ESP32 as a Wi-Fi Access Point
  WiFi.mode(WIFI_AP_STA);
  WiFi.softAP(ssid,password);
  delay(1000);
  WiFi.softAPConfig(local_ip, gateway, subnet);

  // Configure Micro-ROS transport using Wi-Fi
  RMCHECK(rmw_uros_set_custom_transport(
    false,
    (void *) &locator,
    arduino_wifi_transport_open,
    arduino_wifi_transport_close,
    arduino_wifi_transport_write,
    arduino_wifi_transport_read
  ));

  // Set initial state to waiting for ROS 2 Agent
  state = WAITING_AGENT;
  // Initialize message data
  msg.data = 0;
}
```

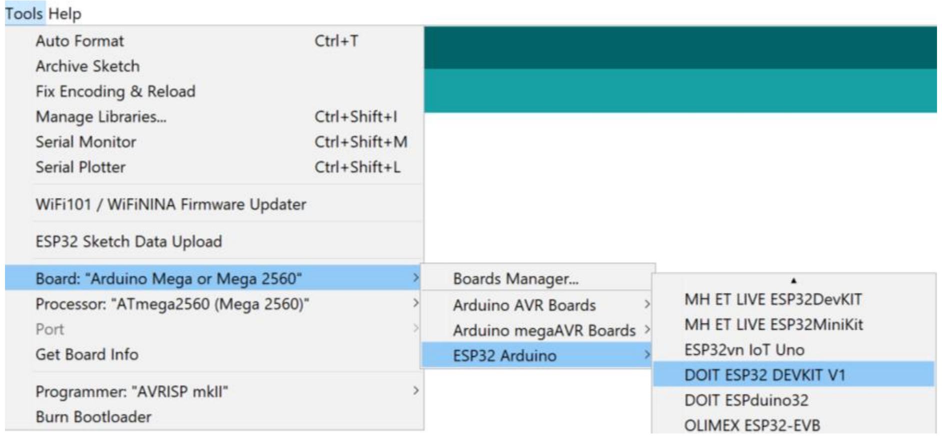
Activity 4

```
// Loop function - Runs continuously
void loop() {
  switch (state) {

    case WAITING_AGENT:
      // Try to ping the Micro-ROS agent every second
      EXECUTE_EVERY_N_MS(1000, state = (RMW_RET_OK ==
rmw_uros_ping_agent(100, 1)) ? AGENT_AVAILABLE : WAITING_AGENT;);
      break;

    case AGENT_AVAILABLE:
      // Try to create ROS entities, move to connected state if successful
      state = (true == create_entities()) ? AGENT_CONNECTED : WAITING_AGENT;
      if (state == WAITING_AGENT) {
        destroy_entities();
      };
      break;

    case AGENT_CONNECTED:
      // Check connection every second, if lost move to disconnected state
      EXECUTE_EVERY_N_MS(1000, state = (RMW_RET_OK ==
rmw_uros_ping_agent(500, 1)) ? AGENT_CONNECTED : AGENT_DISCONNECTED;);
      if (state == AGENT_CONNECTED) {
        rclc_executor_spin_some(&executor, RCL_MS_TO_NS(1));
      }
      break;

    case AGENT_DISCONNECTED:
      // Destroy entities and try reconnecting
      destroy_entities();
      state = WAITING_AGENT;
      break;

    default:
      break;
  }
}
```

Activity

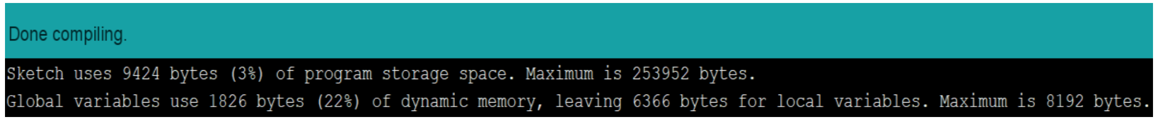
Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for **Hackerboard** is the same)
  - For ESP32 select ESP32 Arduino › DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.

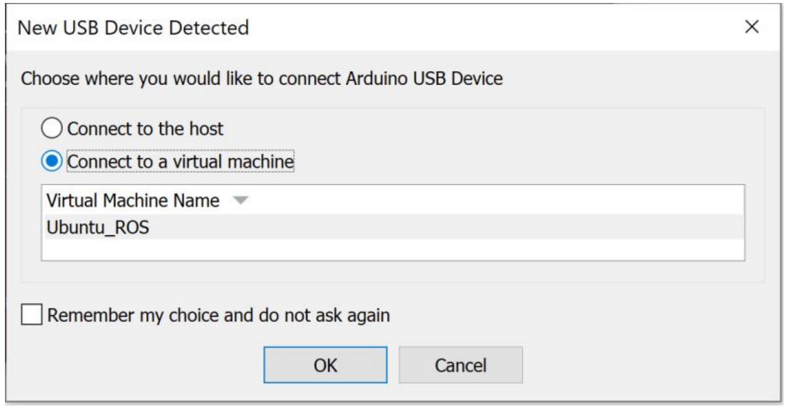
- The following message should be displayed:

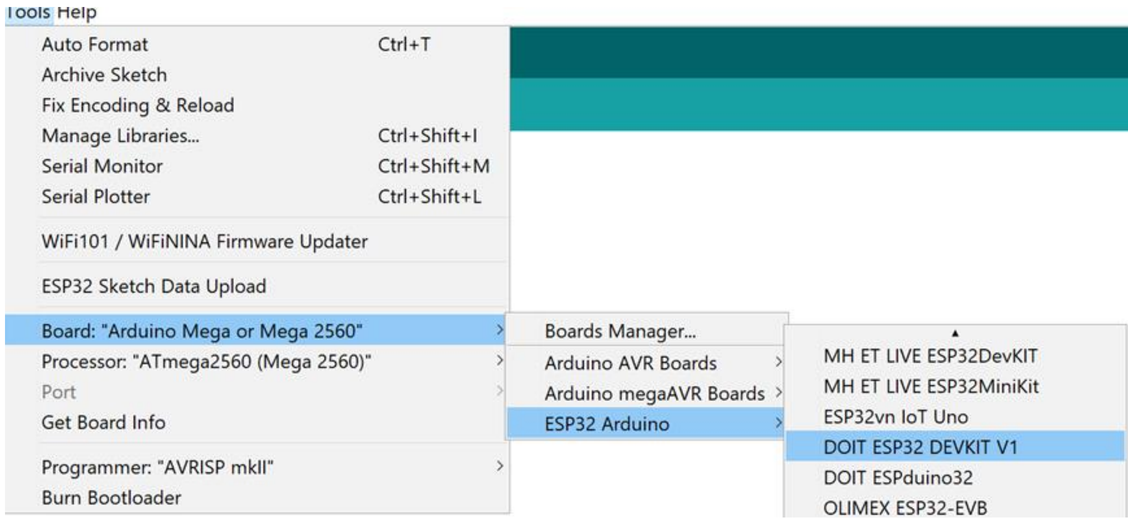


Activity

Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools›Port
    - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
    - If in WSL, follow the steps on the presentation: “MCR2_Micro_ROS_Installation”.
- Select the board to be used Tools -> ESP32 Arduino › DOIT ESP32 DEVKIT V1



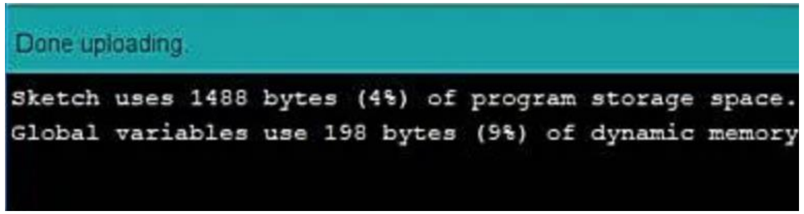


Activity

Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.

- The following message should appear o the IDE



Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
  - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

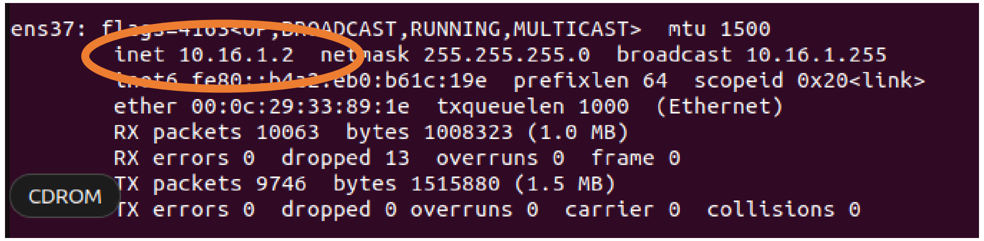
```
sudo chmod 666 /dev/ttyACM*
sudo chmod 666 /dev/ttyUSB*
```

Activity 4

Test (Computer)

1. Connect to the AP “ESP-Access-Point”. Connect like a normal Wi-Fi Network.
   - The network won’t have Internet access (AP).
2. Make sure your IP Address is “10.16.1.2”.
   1. Open a terminal and type

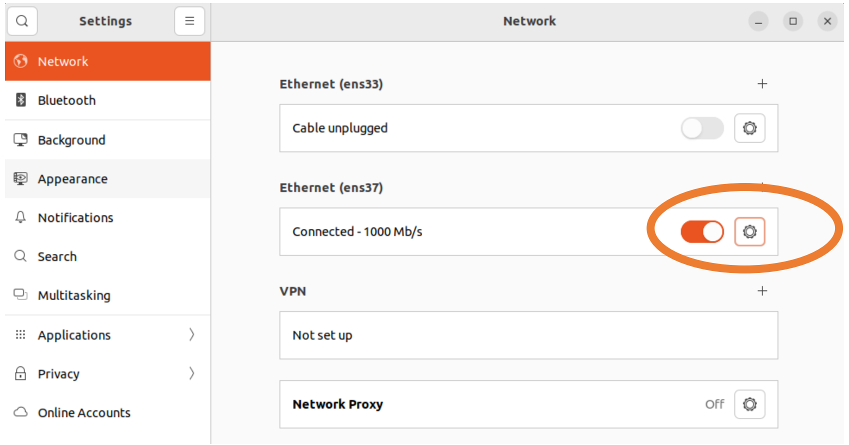
```
$ ifconfig
```



   2. If not “10.16.1.2” go to configure IP section, else skip the section

Configure IP (Ubuntu)

1. Open Ubuntu settings>>Network
2. Click on the “gear” figure of the network adapter.

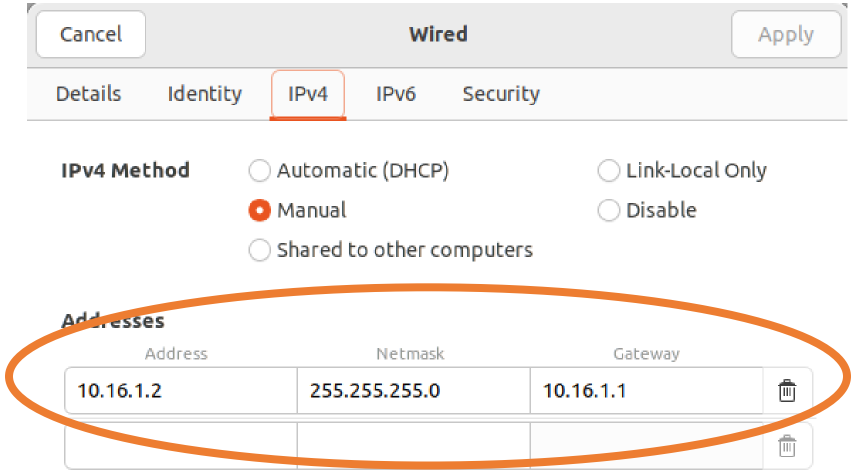


3. A pop-up window will open

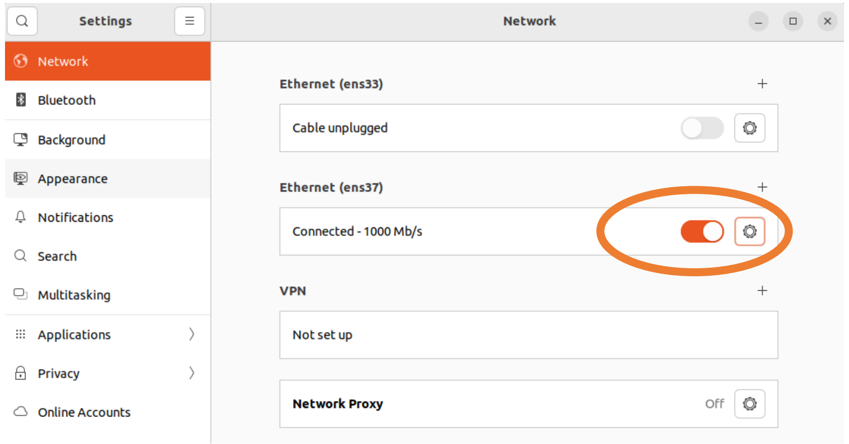
Activity 4

Configure IP (Ubuntu)

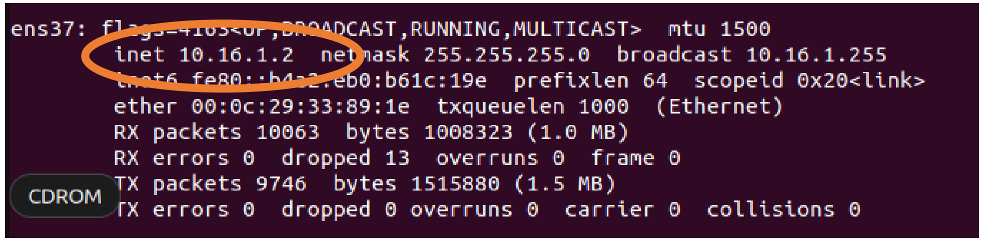
4. Go to the IPV4 tab, select manual and type the following. Click Apply.



5. Reset the adapter by turning on and off the slider on the side of the “gear” figure.



6. Check your IP Address again using “ifconfig”



Activity 3

Test (Computer)

1. Open a terminal and type the following.

```
$ ros2 run micro_ros_agent micro_ros_agent udp4 --port 8888
```

```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent mi
cro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info     | TermiosAgentLinux.cpp
 | init                      | running...
 | fd: 3
[1737636692.138467] info     | Root.cpp              |
set_verbose_level           | logger setup          |
verbose_level: 4
[1737636698.665805] info     | Root.cpp              |
create_client               | create                |
client_key: 0x0C9424D2, session_id: 0x81
[1737636698.666090] info     | SessionManager.hpp |
establish_session           | session established   |
client_key: 0x0C9424D2, address: 0
[1737636698.702311] info     | ProxyClient.cpp       |
create_participant          | participant created   |
client_key: 0x0C9424D2, participant_id: 0x000(1)
[1737636698.721583] info     | ProxyClient.cpp       |
create_topic                | topic created         |
client_key: 0x0C9424D2, topic_id: 0x000(2), particip
ant_id: 0x000(1)
[1737636698.733573] info     | ProxyClient.cpp       |
create_publisher            | publisher created     |
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

- Open another terminal and type the following.

```
$ ros2 topic list
```

```
mcr2@mcr2-virtual-machine:~$ ros2 topic  list
/parameter_events
/rosout
/std_msgs_msg_Int32
```

- Echo the topic "/button_state" . Press the Button!

```
$  ros2 topic echo /std_msgs_msg_Int32
```

```
mcr2@mcr2-virtual-machine:~$ ros2 topic echo /std_msgs_msg_Int32
data: 48
---
data: 49
---
data: 50
---
data: 51
```

Thank you

{Learn, Create, Innovate};

MCR2
Manchester Robotics

T&C

Terms and conditions

{Learn, Create, Innovate};

Terms and conditions

- *THE PIECES, IMAGES, VIDEOS, DOCUMENTATION, ETC. SHOWN HERE ARE FOR INFORMATIVE PURPOSES ONLY. THE DESIGN IS PROPRIETARY AND CONFIDENTIAL TO MANCHESTER ROBOTICS LTD. (MCR2). THE INFORMATION, CODE, SIMULATORS, DRAWINGS, VIDEOS PRESENTATIONS ETC. CONTAINED IN THIS PRESENTATION IS THE SOLE PROPERTY OF MANCHESTER ROBOTICS LTD. ANY REPRODUCTION, RESELL, REDISTRIBUTION OR USAGE IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MANCHESTER ROBOTICS LTD. IS STRICTLY PROHIBITED.*
- *THIS PRESENTATION MAY CONTAIN LINKS TO OTHER WEBSITES OR CONTENT BELONGING TO OR ORIGINATING FROM THIRD PARTIES OR LINKS TO WEBSITES AND FEATURES IN BANNERS OR OTHER ADVERTISING. SUCH EXTERNAL LINKS ARE NOT INVESTIGATED, MONITORED, OR CHECKED FOR ACCURACY, ADEQUACY, VALIDITY, RELIABILITY, AVAILABILITY OR COMPLETENESS BY US.*
- *WE DO NOT WARRANT, ENDORSE, GUARANTEE, OR ASSUME RESPONSIBILITY FOR THE ACCURACY OR RELIABILITY OF ANY INFORMATION OFFERED BY THIRD-PARTY WEBSITES LINKED THROUGH THE SITE OR ANY WEBSITE OR FEATURE LINKED IN ANY BANNER OR OTHER ADVERTISING.*