Micro-ROS Communication

Quality of Service (QoS)

{Learn, Create, Innovate};

Manchester **Robotics**

Introduction

Challenges in Robotic Communication

- Mobile robots typically rely on Wi-Fi or other wireless networks, where bandwidth and latency can fluctuate based on the communication protocol.
- Signal strength and network stability can vary unpredictably due to the robot's movement and its distance from the user.
- Not all robot operations require the same level of reliability—some tasks demand real-time, guaranteed communication, while others can tolerate occasional data loss.

Manchester Robotics

Introduction

Simple Example

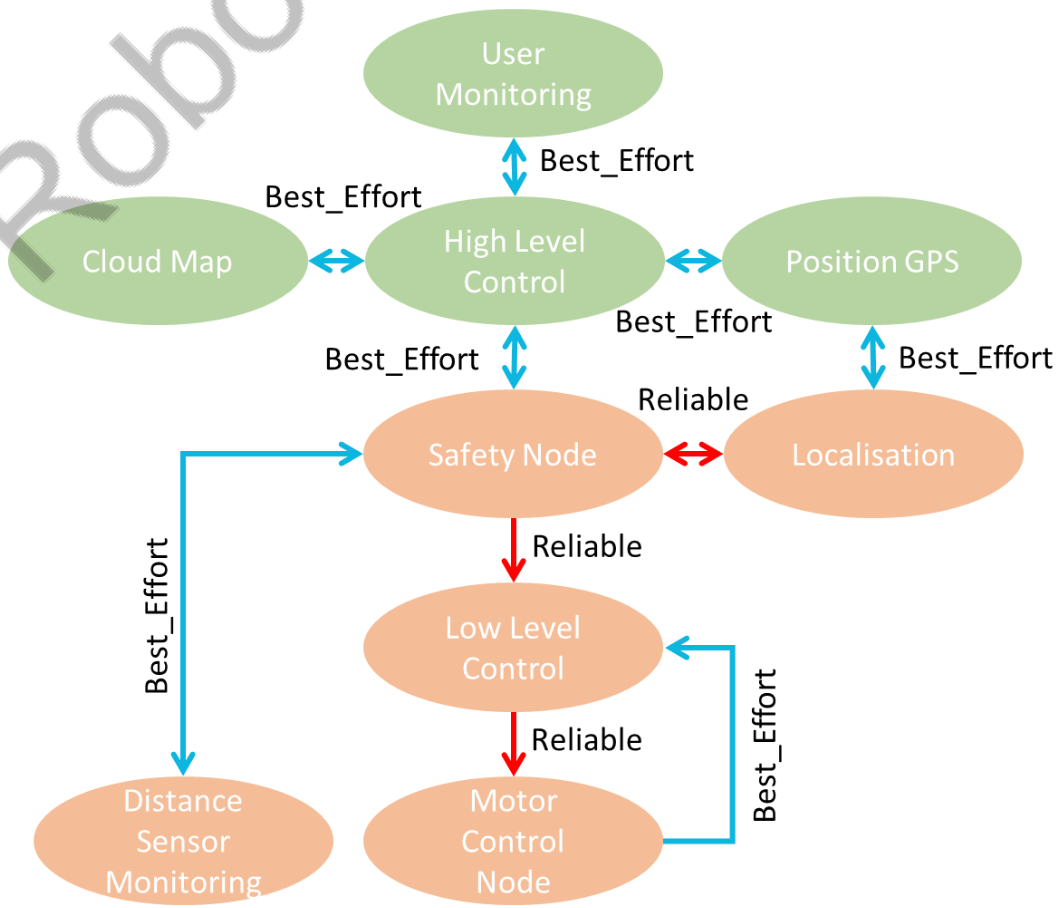
A teleoperated mobile robot:

Reliable QoS (e.g., motor commands)

- The robot must always receive movement commands.
- Uses Reliable QoS to ensure commands are delivered.

Best-Effort QoS (e.g., laser sensor streaming)

- The robot sends a laser sensor information to a node.
- Uses Best-Effort QoS because occasional frame drops don't affect the robot's operation; also, you prefer the latest data.
- It is preferable to have the latest information from the laser sensor.



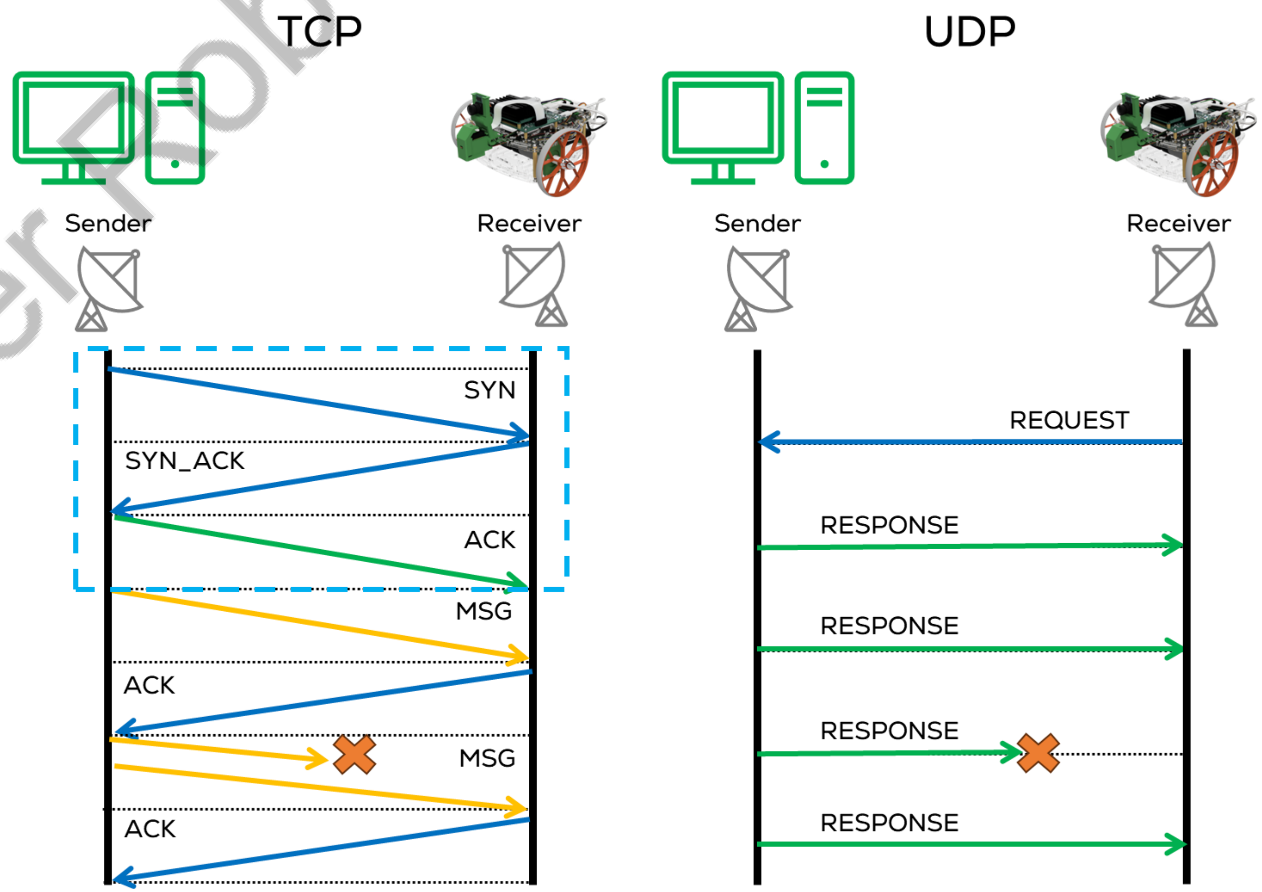
Manchester Robotics

Quality of Service (QoS)

Manchester Robotics

Quality of Service

- QoS lets us express these performance priorities to the underlying communication system.
- QoS (Quality of Service) are a set of policies that define how data is transmitted and managed between nodes.
- QoS policies control message delivery behaviour in a ROS 2 network and determine how messages are handled, including reliability, durability, and history.
- “ROS 2 communication can be as reliable as TCP or as best-effort as UDP.”
- QoS policies help optimize communication for real-time robotics applications

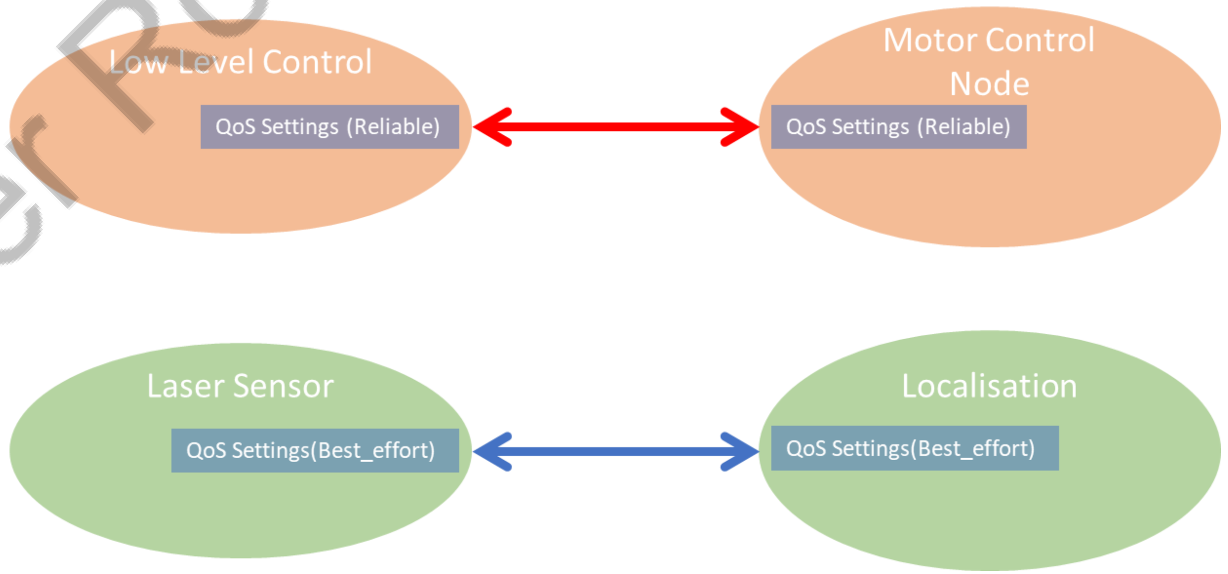


Quality of Service (QoS)

Manchester Robotics

Quality of Service

- QoS policies modify communication for publishers, subscribers, and services.
- QoS Policies allow:
  - Reliable (TCP) connections for mission-critical robot operations.
  - Best-effort (UDP) connections for debugging & non-critical monitoring. Preference for the latest data quickly rather than reliable data.
- A set of QoS “policies” combine to form a QoS “profile”.



Quality of Service (QoS)

Manchester Robotics

Key QoS Policies

- **History:** How middleware handles buffering messages waiting to be sent to the network or passed to a callback function.
  - Keep Last: Stores a fixed number of recent messages, configurable via the queue [depth] option (default is 10).
  - Keep All: Stores all messages until processed (subject to the configured resource limits of the underlying middleware).
- **Depth:** Size of the message queue, honoured if the "History" policy was set to "keep last". (e.g., 10 messages).
- **Reliability:** Ensures the middleware that each message is delivered to the receiver.
  - Reliable: Guarantees message delivery, suitable for critical data, may retry multiple times.
  - Best Effort: Delivers messages as best as possible; may lose them if the network is not robust. Suitable for non-critical data.
- **Durability:** New subscribers get the latest message or wait until the next published message.
  - Volatile: Messages are not stored; only current subscribers receive messages.
  - Transient Local: Stores/Sends messages for late-joining subscribers.

Quality of Service (QoS)

QoS

- More QoS policies, such as Deadline, Lifespan, Liveliness, and Lease Duration, are available; more information is here.
- Setting individual setting values for each publisher/subscriber/service can be tedious and inefficient.
- Profiles are used to integrate a set of policies that can be implemented for different tasks.
- ROS and micro-ros include some predefined QoS profiles that apply to most applications; ROS2 allows the user to reconfigure (tune) them if necessary.
- The user can implement its own QoS Profile and Policies using the QoS class.
- Standard ROS included QoS Profiles:
  - System Defaults Profile
  - Sensor Data Profile:

Manchester Robotics

Quality of Service (QoS)

QoS Profiles

- System Defaults Profile (rmw_qos_profile_default):
  - Uses all the system default values as defined by the middleware implementation currently being used.
    - History: “keep last”
    - Depth: 10
    - Reliability: “reliable”
    - Durabiltiy: “volatile”
    - Liveliness, deadline, lifespan, and lease durations = “default”.

```
static const rmw_qos_profile_t
rmw_qos_profile_default =
{
  RMW_QOS_POLICY_HISTORY_KEEP_LAST,
  10,
  RMW_QOS_POLICY_RELIABILITY_RELIABLE,
  RMW_QOS_POLICY_DURABILITY_VOLATILE,
  RMW_QOS_DEADLINE_DEFAULT,
  RMW_QOS_LIFESPAN_DEFAULT,
  RMW_QOS_POLICY_LIVELINESS_SYSTEM_DEFAULT,
  RMW_QOS_LIVELINESS_LEASE_DURATION_DEFAULT,
  false
};
```

Manchester Robotics

Quality of Service (QoS)

```
static const rmw_qos_profile_t
rmw_qos_profile_sensor_data =
{
  RMW_QOS_POLICY_HISTORY_KEEP_LAST,
  5,
  RMW_QOS_POLICY_RELIABILITY_BEST_EFFORT,
  RMW_QOS_POLICY_DURABILITY_VOLATILE,
  RMW_QOS_DEADLINE_DEFAULT,
  RMW_QOS_LIFESPAN_DEFAULT,
  RMW_QOS_POLICY_LIVELINESS_SYSTEM_DEFAULT,
  RMW_QOS_LIVELINESS_LEASE_DURATION_DEFAULT,
  false
};
```

- Sensor Data Profile (rmw_qos_profile_sensor_data):
  - Getting the latest data quickly, even if it misses some data.
    - History: "keep last"
    - Depth: 5
    - Reliability: "best_effort"
    - Durabiltiy: "volatile"
    - Liveliness, deadline, lifespan, and lease durations = "default".

Manchester Robotics

Quality of Service (QoS)

QoS Profiles

- QoS Setting needs to be agreed upon by the publisher and the subscriber.
- Some combinations of settings are compatible, and others are incompatible.
- ROS Provides some tables here, to verify the compatibility between policies.
- The user must check the compatibility of these QoS Profiles. One incompatible setting will prevent the connection entirely.

Publisher	Subscription	Compatible
Best effort	Best effort	Yes
Best effort	Reliable	No
Reliable	Best effort	Yes
Reliable	Reliable	Yes

Publisher	Subscription	Compatible	Result
Volatile	Volatile	Yes	New messages only
Volatile	Transient local	No	No communication
Transient local	Volatile	Yes	New messages only
Transient local	Transient local	Yes	New and old messages

Manchester Robotics

QoS in ROS

Types of QoS Policies in Micro-ROS

- Best Effort Communication (Fast but Unreliable)

```
//Best Effort publisher/Subscriber use the
rmw_qos_profile_sensor_data

//Best Effort subscriber
rclc_subscription_init_best_effort(
    &setPoint_sub,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "motor/set_point")

//Best Effort publisher
rclc_publisher_init_best_effort(
    &motorCtrlErr_pub,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "motor/ctrl/error")
```

Manchester Robotics

QoS in micro ros

Types of QoS Policies in Micro-ROS

- Reliable Communication (Ensured Delivery but Slower)
  - Reliable communication requires a confirmation for each message sent.
  - This mode can detect errors in the communication process at the cost of increasing the message latency and the resources usage.
  - This message confirmation process can increase blocking time on rcl_publish or executor spin calls as reliable publishers, services and clients will wait for acknowledgement for each sent message.
  - Used for critical messages like commands and mission control (e.g., motor control, safety-critical alerts).

```
//Reliable publisher/Subscriber use the
rmw_qos_profile_default

//Reliable subscriber
rclc_subscription_init_default(
    &setPoint_sub,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "motor/set_point")

//Reliable publisher
rclc_publisher_init_default(
    &motorCtrlErr_pub,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "motor/ctrl/error")
```

Manchester Robotics

QoS in micro ros

Types of QoS Policies in Micro-ROS

- Best Effort Communication (Fast but Unreliable)
  - No acknowledgement is needed when sending a message.
  - Improves publication throughput and reduces resources usage
  - Vulnerable to communication errors (The system tries to deliver messages but does not guarantee they will arrive.)
  - Used for high-frequency sensor data where occasional losses are acceptable (e.g., LiDAR, IMU).

```
//Best Effort publisher/Subscriber use the
rmw_qos_profile_sensor_data

//Best Effort subscriber
rclc_subscription_init_best_effort(
    &setPoint_sub,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "motor/set_point")

//Best Effort publisher
rclc_publisher_init_best_effort(
    &motorCtrlErr_pub,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "motor/ctrl/error")
```

Manchester Robotics

Micro-ROS Serial Communication

Activity 1: QoS Publisher – Subscriber

Manchester Robotics

{Learn, Create, Innovate};

MCR2 Manchester Robotics

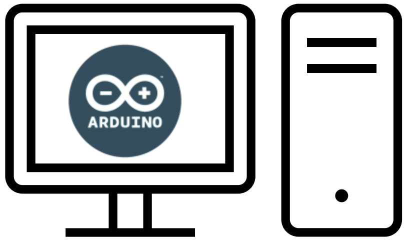
Requirements

- 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



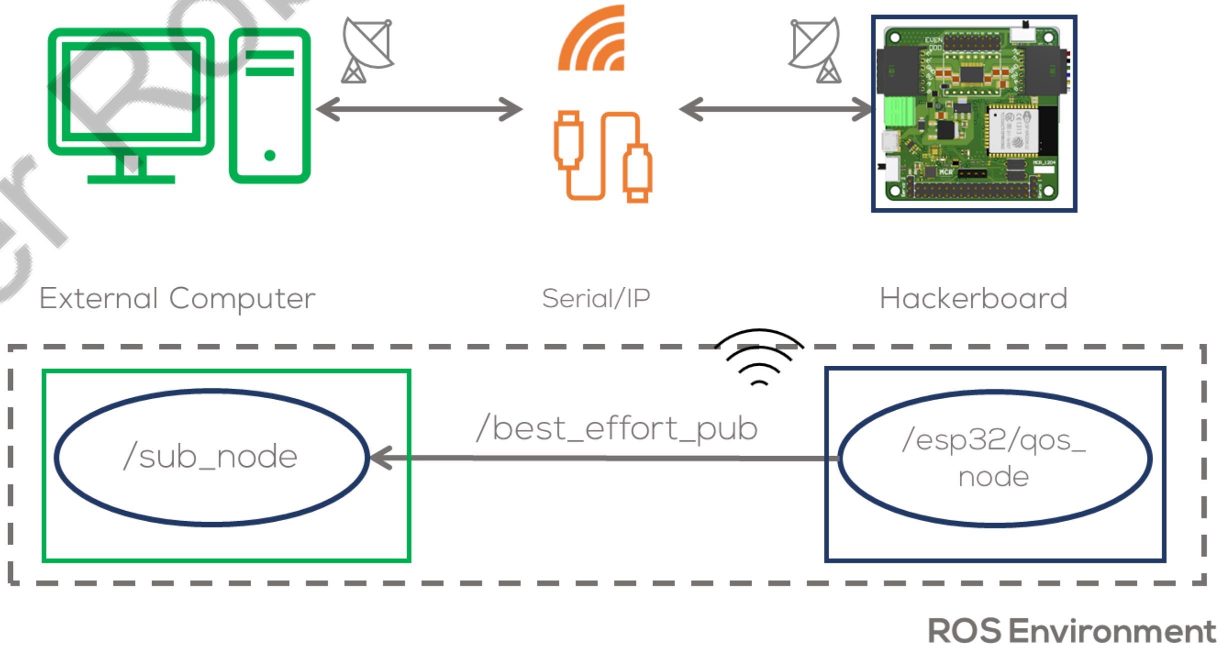
Computer

Manchester Robotics

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 or serial.
- The node will publish a simple Float32 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.

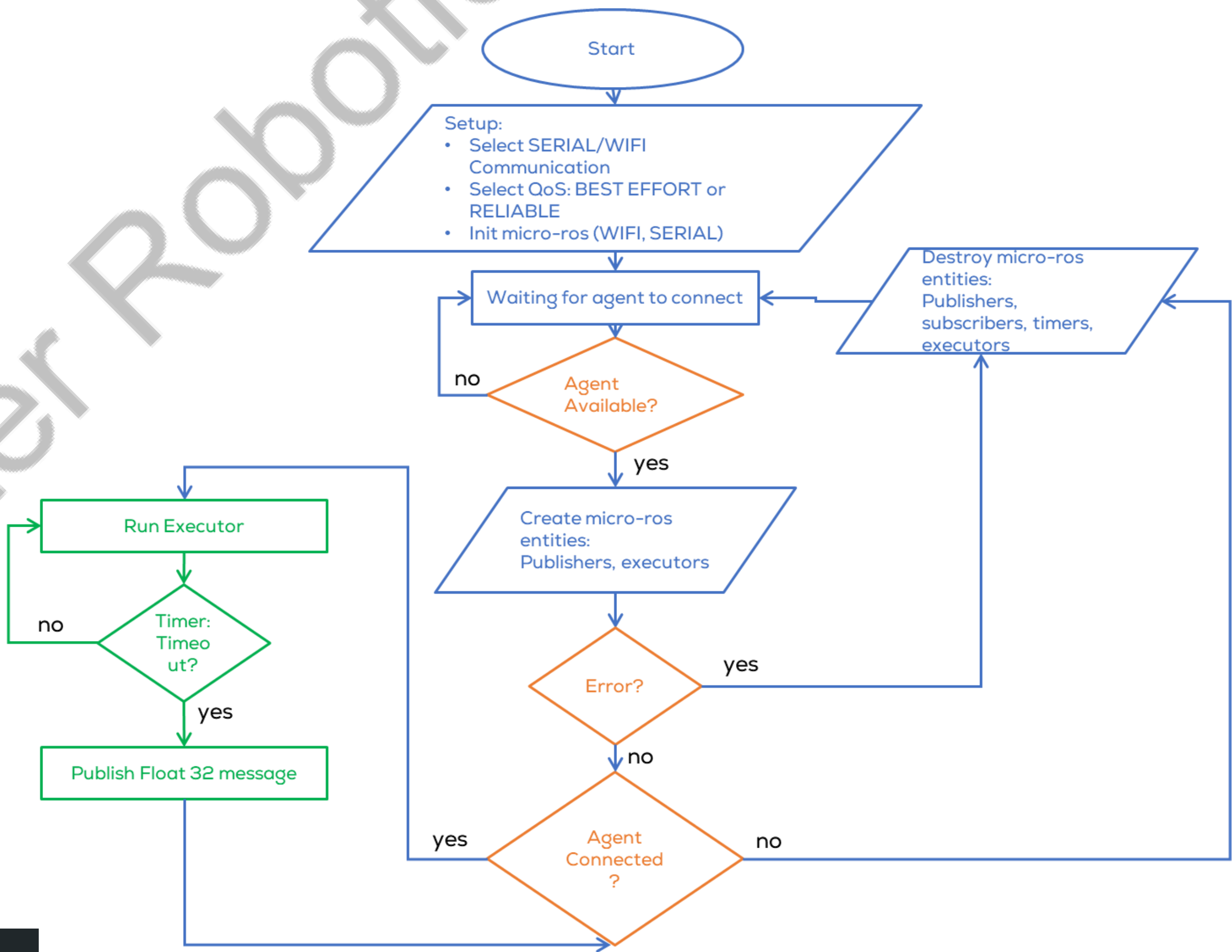


Manchester Robotics

Description

- The following activity publishes a sinusoidal wave continuously using a timer.
- The publisher can be selected by the user, as a "BEST_EFFORT" publisher or a "RELIABLE" publisher. The topic name varies according to the selected QoS: "/best_effort_pub" or "/reliable_pub" accordingly.
- The type of communication can also be set by the user to be "WIFI" or "SERIAL".
- Configuration of these parameters should be done in line 14, by replacing the appropriate value, as

```
14  /////// ///////////////////CONFIGURE HERE ////////////////////////////////
15  #define BEST_EFFORT         //Select between BEST_EFFORT or RELIABLE
16  #define SERIAL             //Select between SERIAL and WIFI Communication
17  //////////////////////////////////////////////////////////////////////////
```



Manchester Robotics

```
#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/float32.h>  // Predefined ROS 2 message type (float
messages)

/////// ///////////////////CONFIGURE HERE //////////////////////////////
#define BEST_EFFORT         //Select between BEST_EFFORT or RELIABLE
#define SERIAL             //Select between SERIAL and WIFI Communication
//////////////////////////////////////////////////////////////////////////

// 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__Float32 msg;  // Integer message type

micro_ros_agent_locator locator;    // Stores connection details for Micro-ROS
Agent

#ifdef WIFI
// 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 = 9999;
#endif
```

Manchester Robotics

```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;

// Variables for sine wave generation
float t = 0.0;   // Phase variable (used to generate the wave)
const float frequency = 1.0;   // Frequency of the sine wave (Hz)
const float amplitude = 1.0;   // Amplitude of the sine wave
const float dt = 0.05;   // Time step (50ms period)


// 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);
  }
}
```

```c
// 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
    // Generate the sinusoidal signal
    msg.data = amplitude * sin(2 * M_PI * frequency * t);
    t += dt;  // Increment phase variable
  }
}
```

Manchester Robotics

```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, "esp32_qos_node", "", &support));

  // Create a best-effort publisher (non-reliable, no message history) (QoS)
  #ifdef BEST_EFFORT
  RCCHECK(rclc_publisher_init_best_effort(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "best_effort_pub"));
  #else
  // Create a reliable publisher (non-reliable, no message history) (QoS)
  RCCHECK(rclc_publisher_init_default(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
    "reliable_pub"));
  #endif

  // Create a timer to publish messages every 1000ms (1 second)
  const unsigned int timer_timeout = (dt*1000)/2;
  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
}
```

```c
// 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);
}
```

Manchester Robotics

```cpp
// Setup function - Runs once when ESP32 starts
void setup() {
  // Initialize memory allocator
  allocator = rcl_get_default_allocator();

  #ifdef WIFI
  // 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
  ));
  #else
    set_microros_transports();
  #endif

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

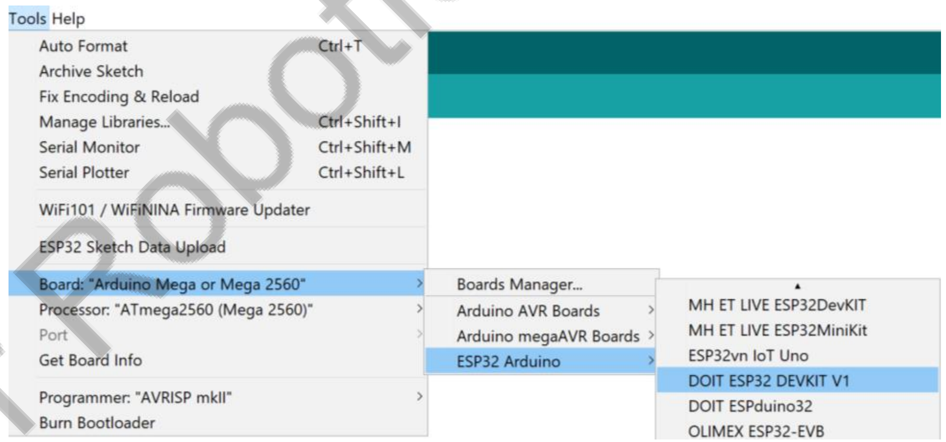
```cpp
// Loop function - Runs continuously
void loop() {
  switch (state) {

    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;
  }

}
```

Manchester Robotics

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:

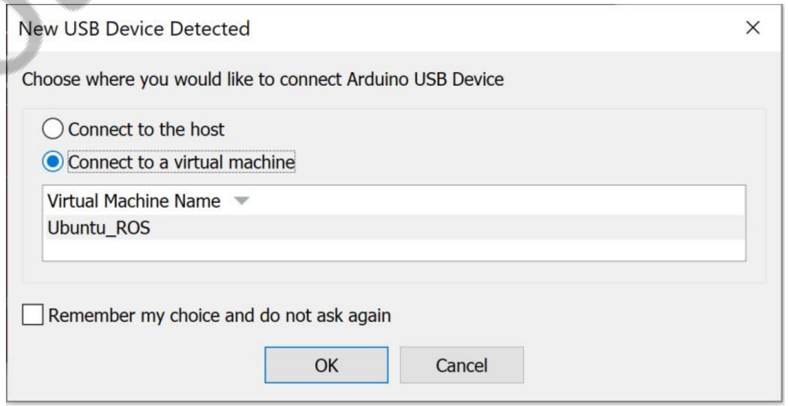
```
Done compiling.
Sketch uses 9424 bytes (3%) of program storage space. Maximum is 253952 bytes.
Global variables use 1826 bytes (22%) of dynamic memory, leaving 6366 bytes for local variables. Maximum is 8192 bytes.
```

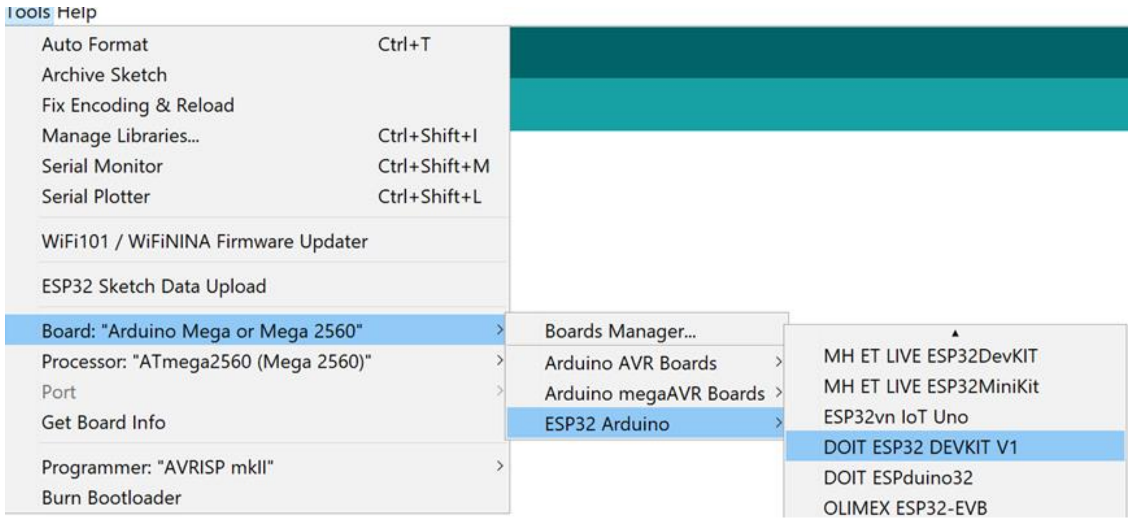
Manchester Robotics

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





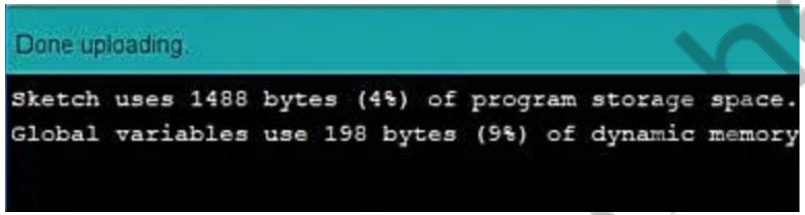
Manchester Robotics

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*
```

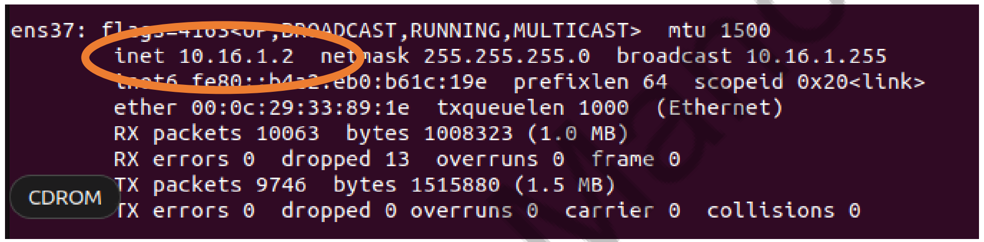
Manchester Robotics

Activity 1

TEST: IF WIFI IS SELECTED (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

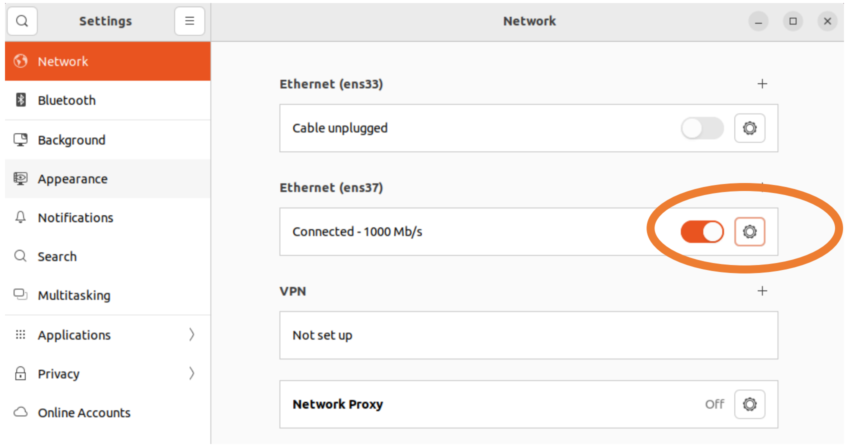
```
$ ifconfig
```



   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.



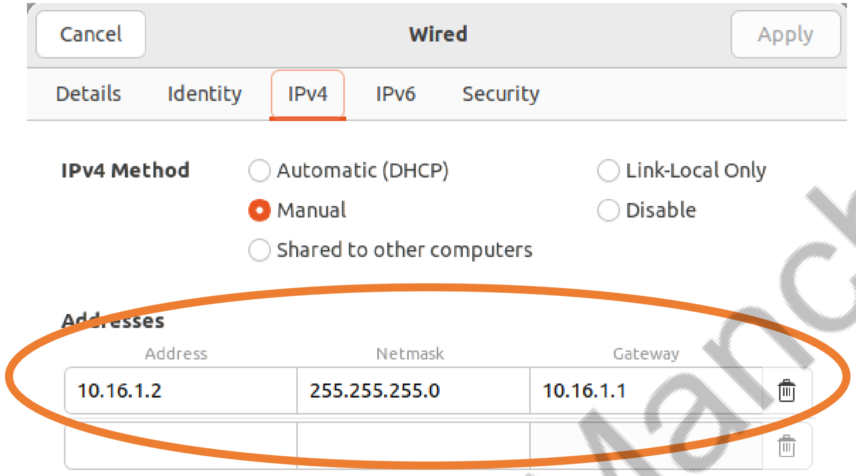
3. A pop-up window will open

Manchester Robotics

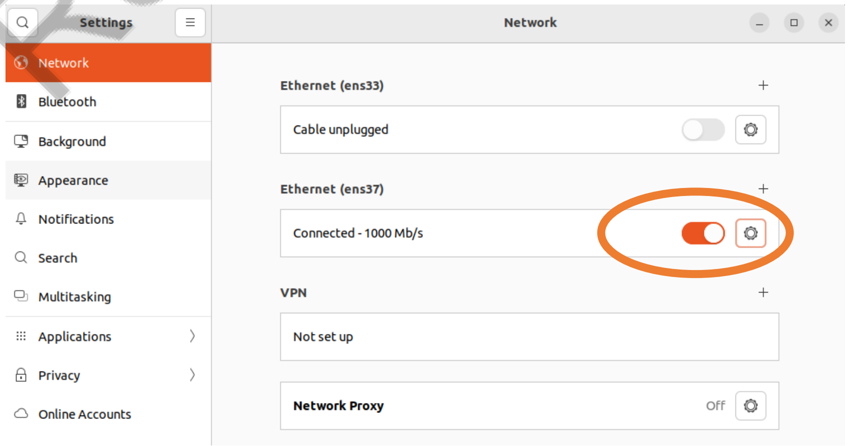
Activity 1

Configure IP (Ubuntu)

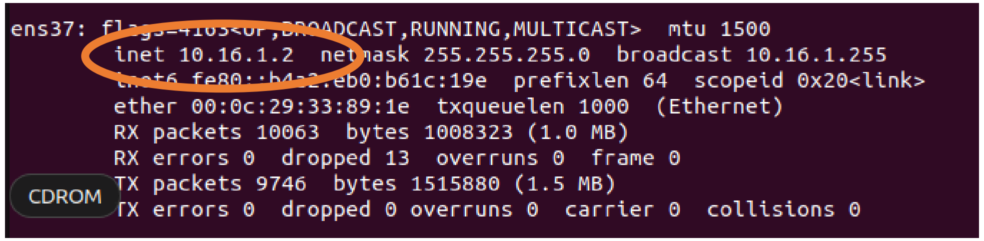
4. Go to the IPV4 tab, select manual and type the following. Click Apply.



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"



Manchester Robotics

Activity 1

Test (Computer)

1. Open a terminal and type the following if WIFI is Selected

```
$ ros2 run micro_ros_agent micro_ros_agent udp4 --port 9999
```

2. If SERIAL is selected.

```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyUSB0
```

```
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_ws$ ros2 topic list
/best_effort_pub
/parameter_events
/rosout
```

- Info the topic "/best_effort_pub" or "/reliable_pub" .

```
$ ros2 topic info /best_effort_pub -v
```

```
mcr2@mcr2-virtual-machine:~/ros2_ws$ ros2 topic info /best_effort_pub -v
Type: std_msgs/msg/Float32

Publisher count: 1

Node name: esp32_qos_node
Node namespace: /
Topic type: std_msgs/msg/Float32
Endpoint type: PUBLISHER
GID: 01.0f.c0.13.99.5c.86.8b.07.00.00.00.00.00.01.03.00.00.00.00.00.00.00.00
QoS profile:
  Reliability: BEST_EFFORT
  History (Depth): UNKNOWN
  Durability: VOLATILE
  Lifespan: Infinite
  Deadline: Infinite
  Liveliness: AUTOMATIC
  Liveliness lease duration: Infinite

Subscription count: 0
```

Manchester Robotics

Activity 1

Host Computer

- For the host computer, two subscriber nodes will be declared, each one for a different type of communication: "best_effort" and "reliable"
- Create a package called "micro_ros_subscriber"

```
$ ros2 pkg create --build-type ament_python micro_ros_subscriber --
   dependencies std_msgs rclpy ros2launch --node-name
   best_effort_sub
```

- Open the created package and add a new node called "reliable_sub.py" to the package
- Add a launch folder with the following files:
  - best_effort_serial_launch.py
  - best_effort_wifi_launch.py
  - reliable_serial_launch.py
  - reliable_wifi_launch.py

- Do not forget to give execution permission to the files.

```
$ sudo chmod +x src/micro_ros_subscriber/micro_ros_subscriber/*
$ sudo chmod +x src/micro_ros_subscriber/launch/*
```

```
src/micro_ros_subscriber/
├── launch
│   ├── best_effort_serial_launch.py
│   ├── best_effort_wifi_launch.py
│   ├── reliable_serial_launch.py
│   └── reliable_wifi_launch.py
├── micro_ros_subscriber
│   ├── best_effort_sub.py
│   ├── __init__.py
│   └── reliable_sub.py
├── package.xml
├── resource
│   └── micro_ros_subscriber
├── setup.cfg
├── setup.py
└── test
    ├── test_copyright.py
    ├── test_flake8.py
    └── test_pep257.py
```

Manchester Robotics

Activity 1

Configure the “setup.py”

```python
from setuptools import find_packages, setup
import os
from glob import glob

package_name = 'micro_ros_subscriber'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share', package_name, 'launch'),
glob(os.path.join('launch', '*launch.[pxy][yma]*'))),
    ],
install_requires=['setuptools'],
    zip_safe=True,
    maintainer='mcr2',
    maintainer_email='mcr2@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'best_effort_sub = micro_ros_subscriber.best_effort_sub:main',
            'reliable_sub = micro_ros_subscriber.reliable_sub:main'
        ],
    },
)
```

Manchester Robotics

Activity 1

best_effort_sub.py

```
import rclpy
from rclpy.node import Node
import numpy as np
from std_msgs.msg import Float32
from rclpy.qos import qos_profile_sensor_data

class MicroROSSubscriber(Node):
    def __init__(self):
        super().__init__('micro_ros_subscriber')

        self.best_effort_subscription = self.create_subscription(
            Float32,
            'best_effort_pub',
            self.best_effort_callback,
            qos_profile=qos_profile_sensor_data
        )

        self.best_effort_subscription

        self.signal_msg = Float32()

    def best_effort_callback(self, signal_in):
        self.get_logger().info(f"Received: {signal_in.data}")
```

```
def main(args=None):
    rclpy.init(args=args)

    node = MicroROSSubscriber()

    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        if rclpy.ok():  # Ensure shutdown is only called once
            rclpy.shutdown()
        node.destroy_node()

if __name__ == '__main__':
    main()
```

Manchester Robotics

Activity 1

reliable_sub.py

```python
import rclpy
from rclpy.node import Node
import numpy as np
from std_msgs.msg import Float32
from rclpy.qos import qos_profile_system_default

class MicroROSSubscriber(Node):
    def __init__(self):
        super().__init__('micro_ros_subscriber')

        self.reliable_subscription = self.create_subscription(
            Float32,
            'reliable_pub',
            self.best_effort_callback,
            qos_profile=qos_profile_system_default
        )

        self.reliable_subscription

        self.signal_msg = Float32()

    def best_effort_callback(self, signal_in):
        self.get_logger().info(f"Received: {signal_in.data}")
```

```python
def main(args=None):
    rclpy.init(args=args)

    node = MicroROSSubscriber()

    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        if rclpy.ok():  # Ensure shutdown is only called once
            rclpy.shutdown()
        node.destroy_node()

if __name__ == '__main__':
    main()
```

Manchester Robotics

Activity 1

best_effort_serial_launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():

    micro_ros_agent = Node(name="micro_ros_agent",
                       package='micro_ros_agent',
                       executable='micro_ros_agent',
                       output='screen',
                       arguments=[
                        'serial',
                        "--dev", '/dev/ttyUSB0',
                        ]
                       )

    sub_node = Node(name="sub_node",
                       package='micro_ros_subscriber',
                       executable='best_effort_sub',
                       output='screen'
                       )

    rqt_node = Node(name='rqt_plot',
                       package='rqt_plot',
                       executable='rqt_plot',
                       arguments=['/best_effort_pub/data']
                       )

    l_d = LaunchDescription([micro_ros_agent, sub_node, rqt_node ])

    return l_d
```

Manchester Robotics

Activity 1

best_effort_wifi_launch.py

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():

    micro_ros_agent = Node(name="micro_ros_agent",
                           package='micro_ros_agent',
                           executable='micro_ros_agent',
                           output='screen',
                           arguments=[
                            "udp4",
                            "--port", '9999',
                            "--session-timeout", "0"
                            ]
                           )


    sub_node = Node(name="sub_node",
                       package='micro_ros_subscriber',
                       executable='best_effort_sub',
                       output='screen'
                       )

rqt_node = Node(name='rqt_plot',
                   package='rqt_plot',
                   executable='rqt_plot',
                   arguments=['/best_effort_pub/data']
                   )

   l_d = LaunchDescription([micro_ros_agent, sub_node, rqt_node ])

   return l_d
```

Manchester Robotics

Activity 1

reliable_serial_launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    micro_ros_agent = Node(name="micro_ros_agent",
                           package='micro_ros_agent',
                           executable='micro_ros_agent',
                           output='screen',
                           arguments=[
                            'serial',
                            "--dev", '/dev/ttyUSB0',
                            ]
                           )

    sub_node = Node(name="sub_node",
                    package='micro_ros_subscriber',
                    executable='reliable_sub',
                    output='screen'
                    )
```

```
rqt_node = Node(name='rqt_plot',
                package='rqt_plot',
                executable='rqt_plot',
                arguments=['/reliable_pub/data']
                )

   l_d = LaunchDescription([micro_ros_agent, sub_node, rqt_node ])

   return l_d
```

Manchester Robotics

Activity 1

reliable_wifi_launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    micro_ros_agent = Node(name="micro_ros_agent",
                       package='micro_ros_agent',
                       executable='micro_ros_agent',
                       output='screen',
                       arguments=[
                        'udp4',
                        "--port", '9999',
                        ]
                       )

    sub_node = Node(name="sub_node",
                package='micro_ros_subscriber',
                executable='reliable_sub',
                output='screen'
                )
```

```
rqt_node = Node(name='rqt_plot',
                package='rqt_plot',
                executable='rqt_plot',
                arguments=['/reliable_pub/data']
                )

   l_d = LaunchDescription([micro_ros_agent, sub_node, rqt_node ])

   return l_d
```

Manchester Robotics

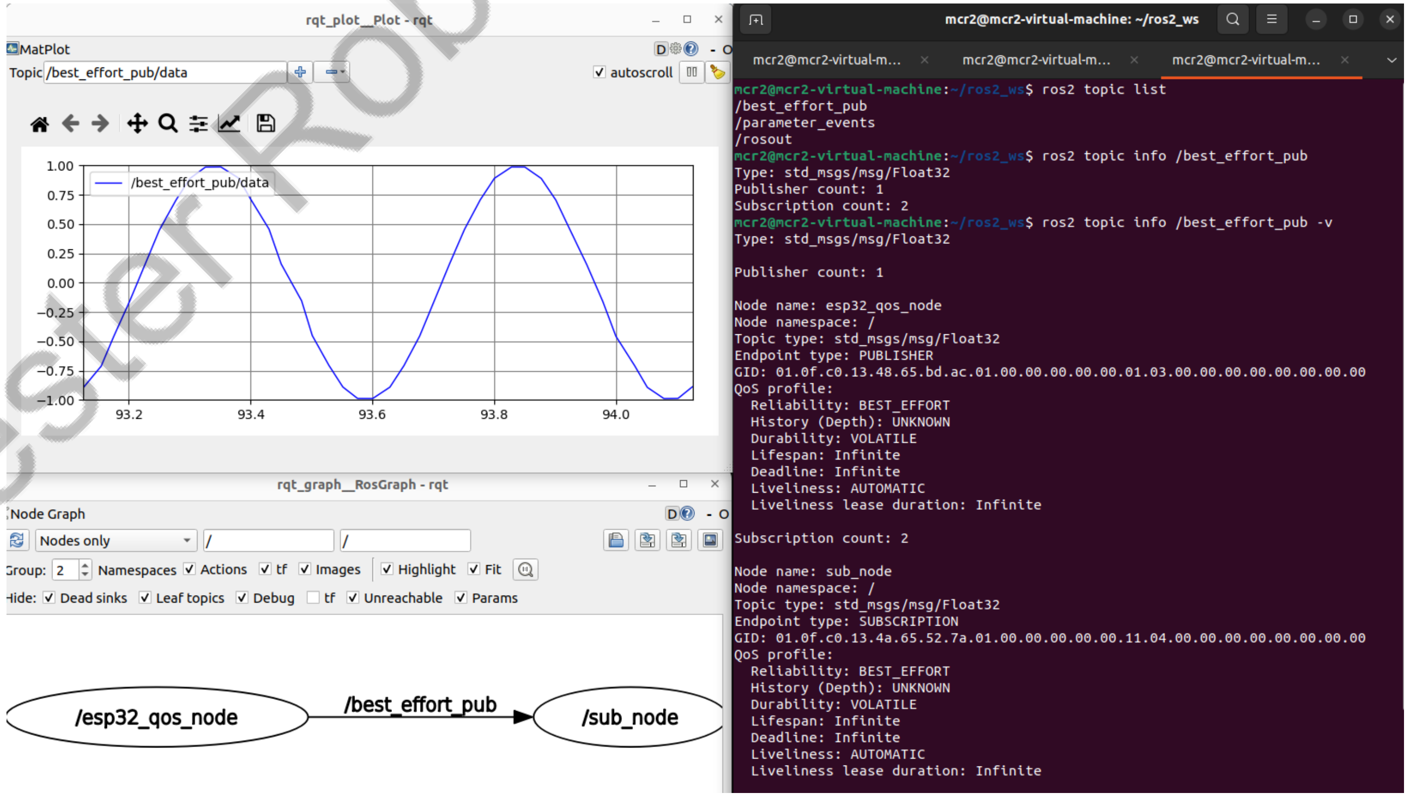
Activity 1

- Build and source the package

```
$ colcon build
$ source install/setup.bash
```

- Launch the corresponding file, according to the configuration (SERIAL/WIFI, BEST_EFFORT/RELIABLE)

```
$ ros2 launch micro_ros_subscriber reliable_serial_launch.py
```



Manchester Robotics

Thank you

{Learn, Create, Innovate};

Manchester Robotics

T&C

Terms and conditions

{Learn, Create, Innovate};

Manchester Robotics

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.*

Manchester Robotics