{Learn, Create, Innovate};

ROS serial communication

Interfacing a microcontroller and ROS

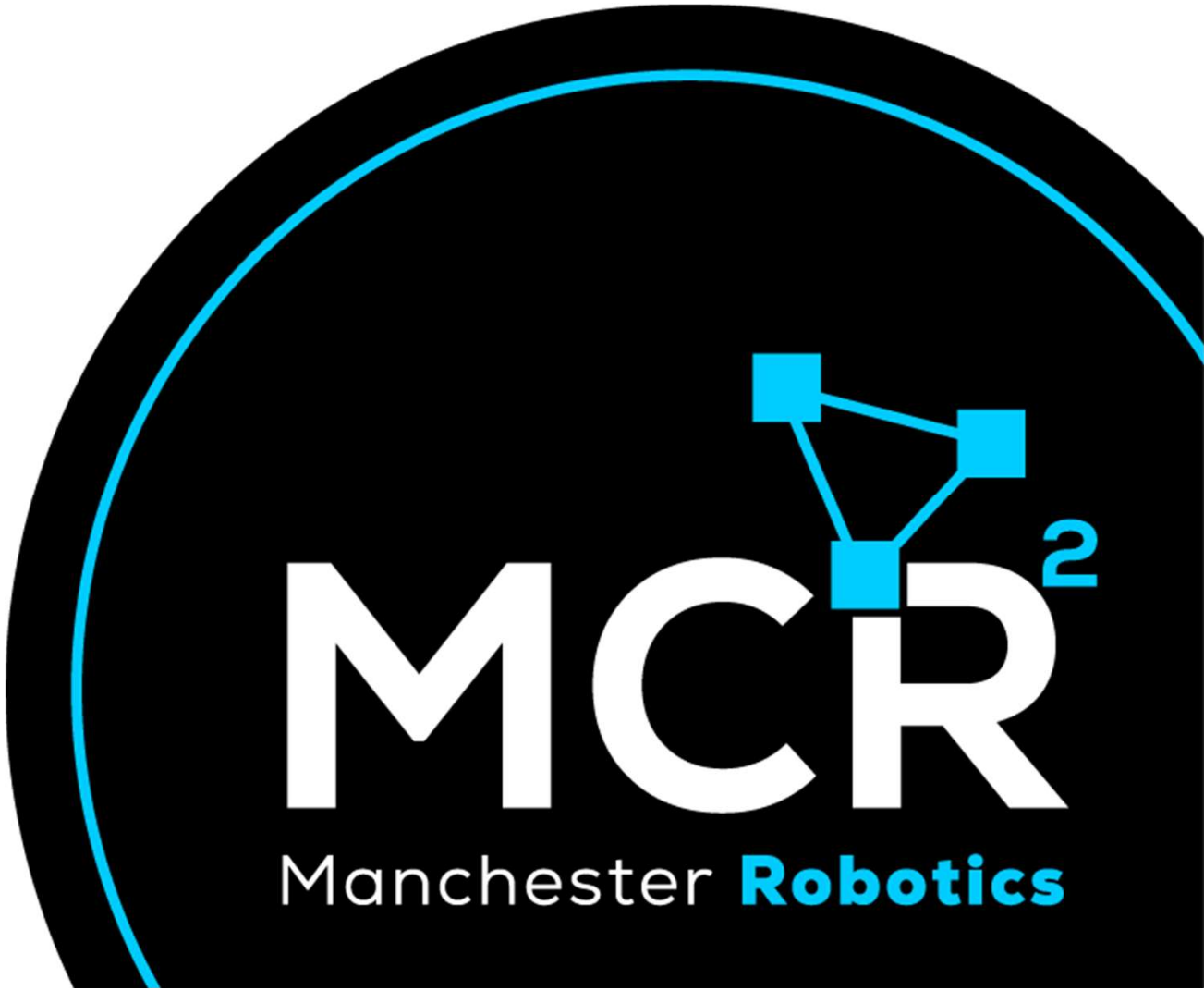
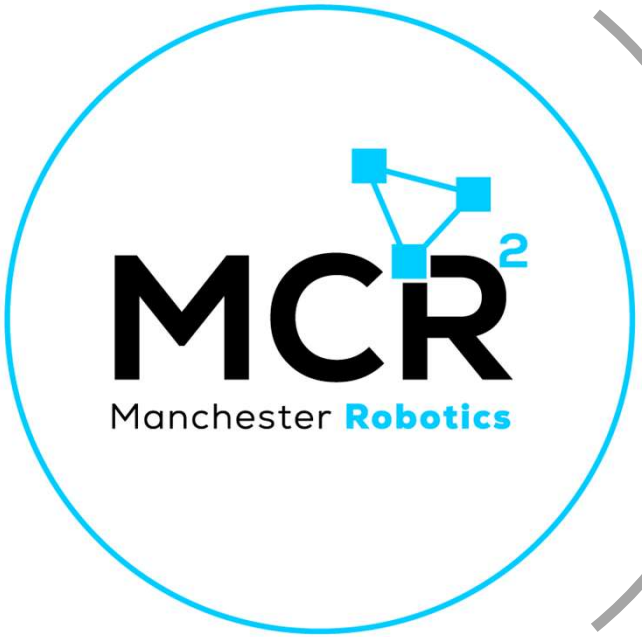


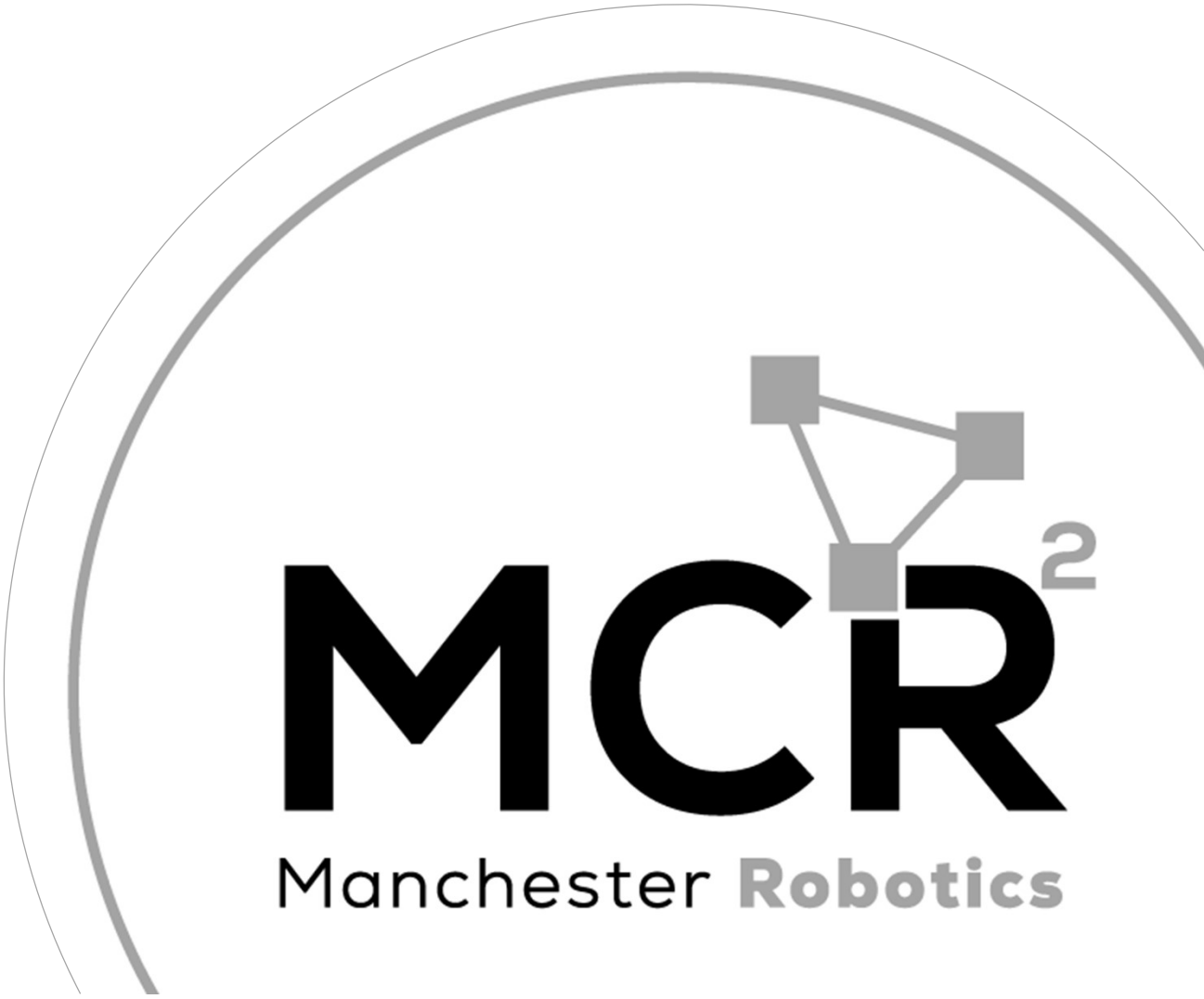
Table of contents



ROS Serial Communication

Rosserial

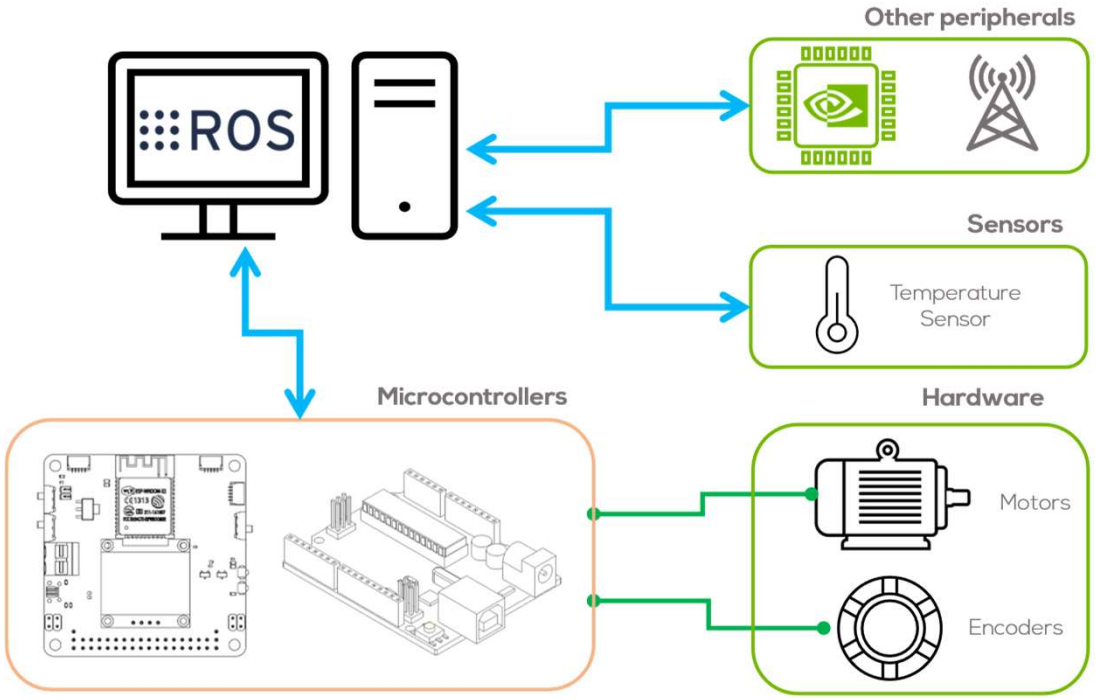
{Learn, Create, Innovate};



Rosserial

Rosserial

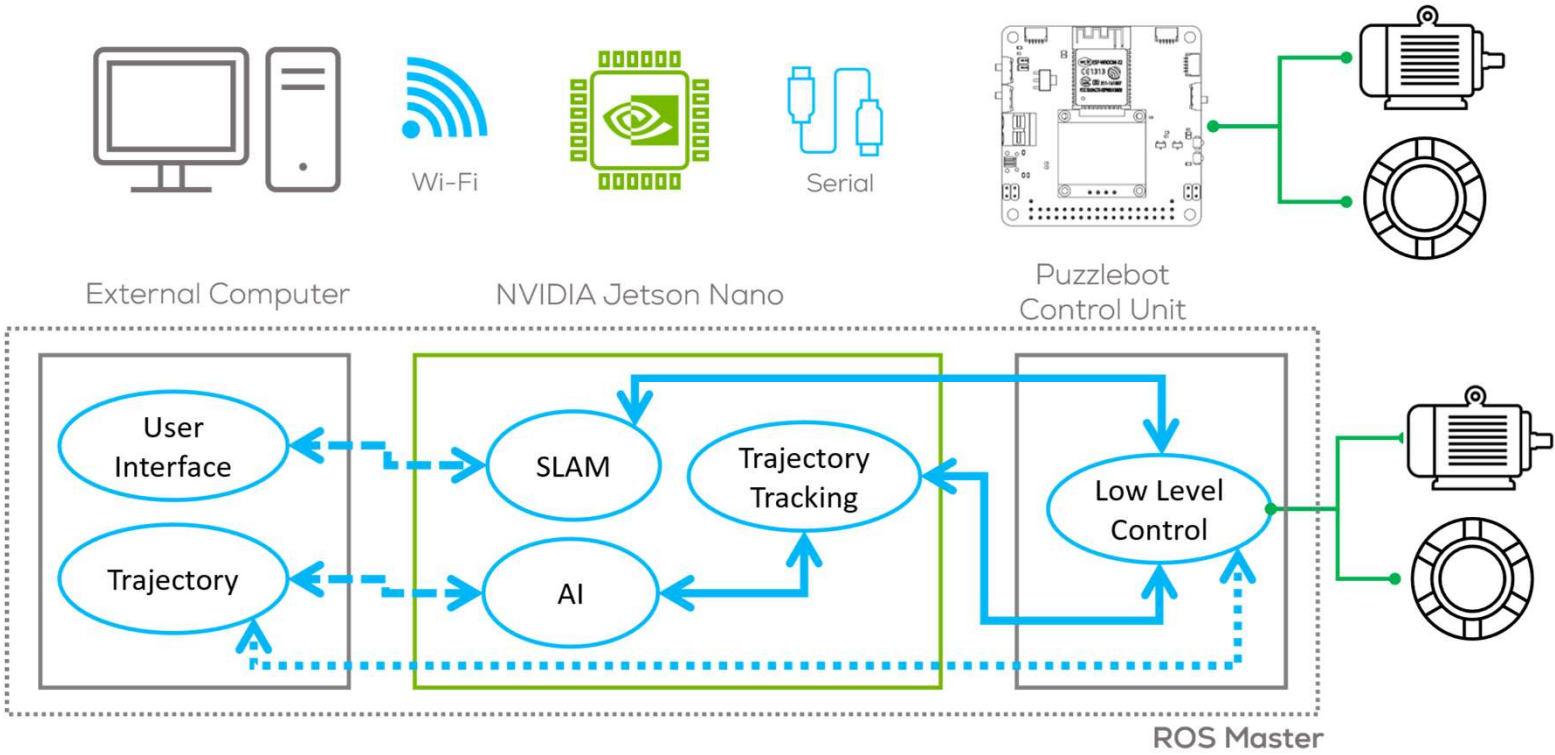
- Rosserial was created with the purpose of standardizing the communication between computers and robotic hardware.
- Rosserial is a protocol for wrapping standard ROS serialized messages and multiplexing multiple topics and services over a character device such as a serial port or network socket.
- Allows the hardware to communicate with the rest of the ROS system.
- Rosserial allows them to use topics, services and logging features of ROS.



Rosserial

Rosserial

- In short, it allows to create ROS Nodes inside microcontrollers or different hardware, allowing communication with different systems inside the ROS network.
- One example is the Puzzlebot. The robot utilizes Rosserial communication to access the node created on the microcontroller that drives the different sensors and actuators.



Rosserial

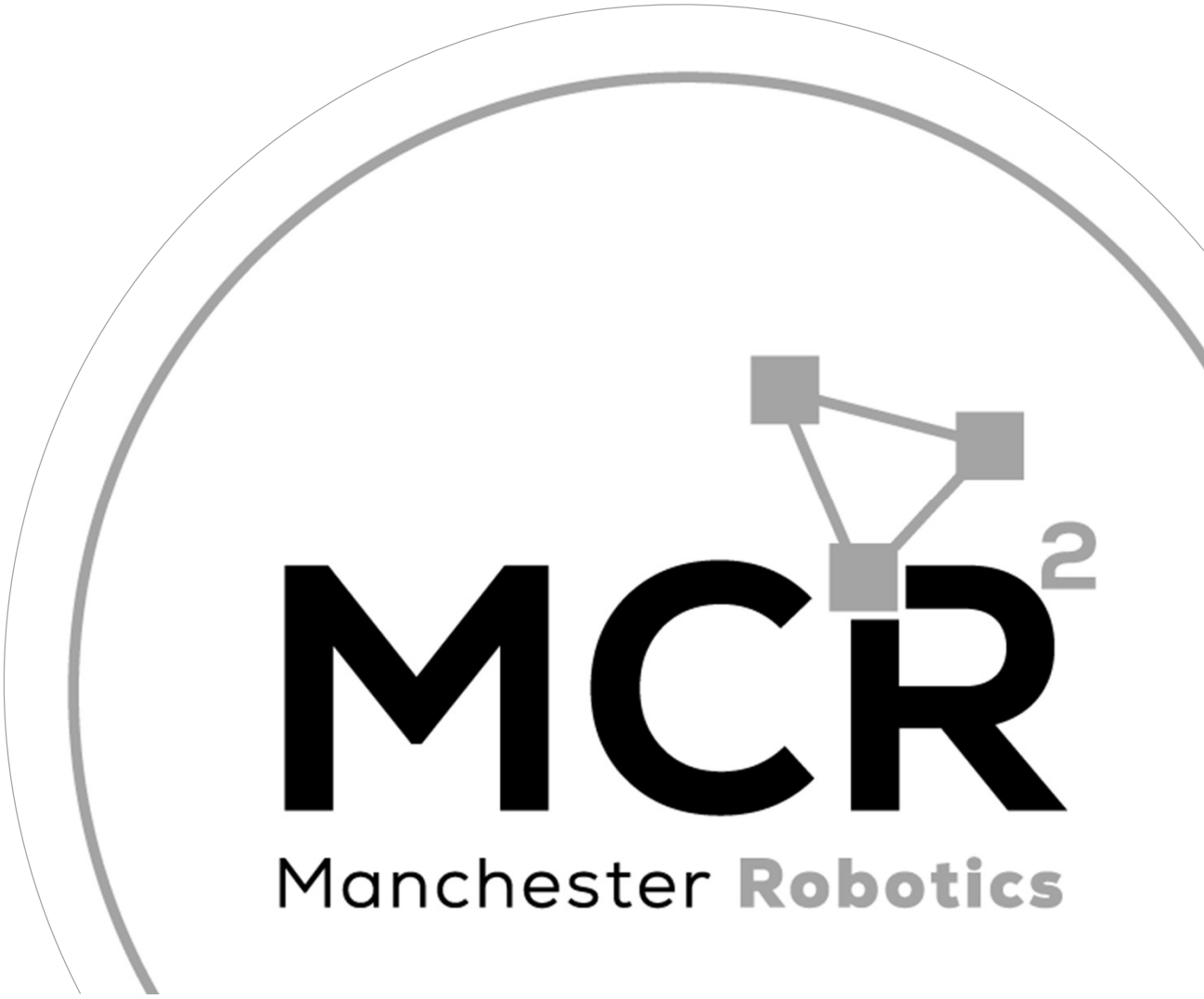
Rosserial

- Rosserial is has some limitations based on the microcontroller to be used.
- One of the most common ones is the maximum size of a message, maximum number of publishers and subscribers.
  - ATMEGA168 (Arduino UNO) : 150/150 bytes input/output buffer and 6 publishers and 6 subscribers.
  - ATMEGA328P (Arduino MEGA) 280/280 bytes input/output buffer and 25 publishers and 25 subscribers.
- More information about rosserial and its limitations can be found here.
- One of the most common usages of rosserial is to be used with a microcontroller (as described before) to communicate with sensors, actuators, etc.
- 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.
- Arduino and ESP32 are some of the most common microcontrollers used with rosserial.
- Rosserial is also used with other hardware like Xbee, sensors and actuators with an embedded linux, Mbed platforms, etc.

ROS Serial Communication

MCU Program

{Learn, Create, Innovate};



MCU Programming

General information

- As stated before, Arduino and ESP32 are some of the most used MCU's.
- Both can be programmed using the Arduino IDE.
- For all the activities and challenges in this session, the Arduino IDE will be used for programming.
- The activities and challenges shown in this presentation can be performed using the MCR2 Hackerbard or the Arduino Mega, alongside a DC motor and a motor driver module (L298n) for the challenge.
- Please refer to the prerequisites of this session for the complete list of required components.

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.

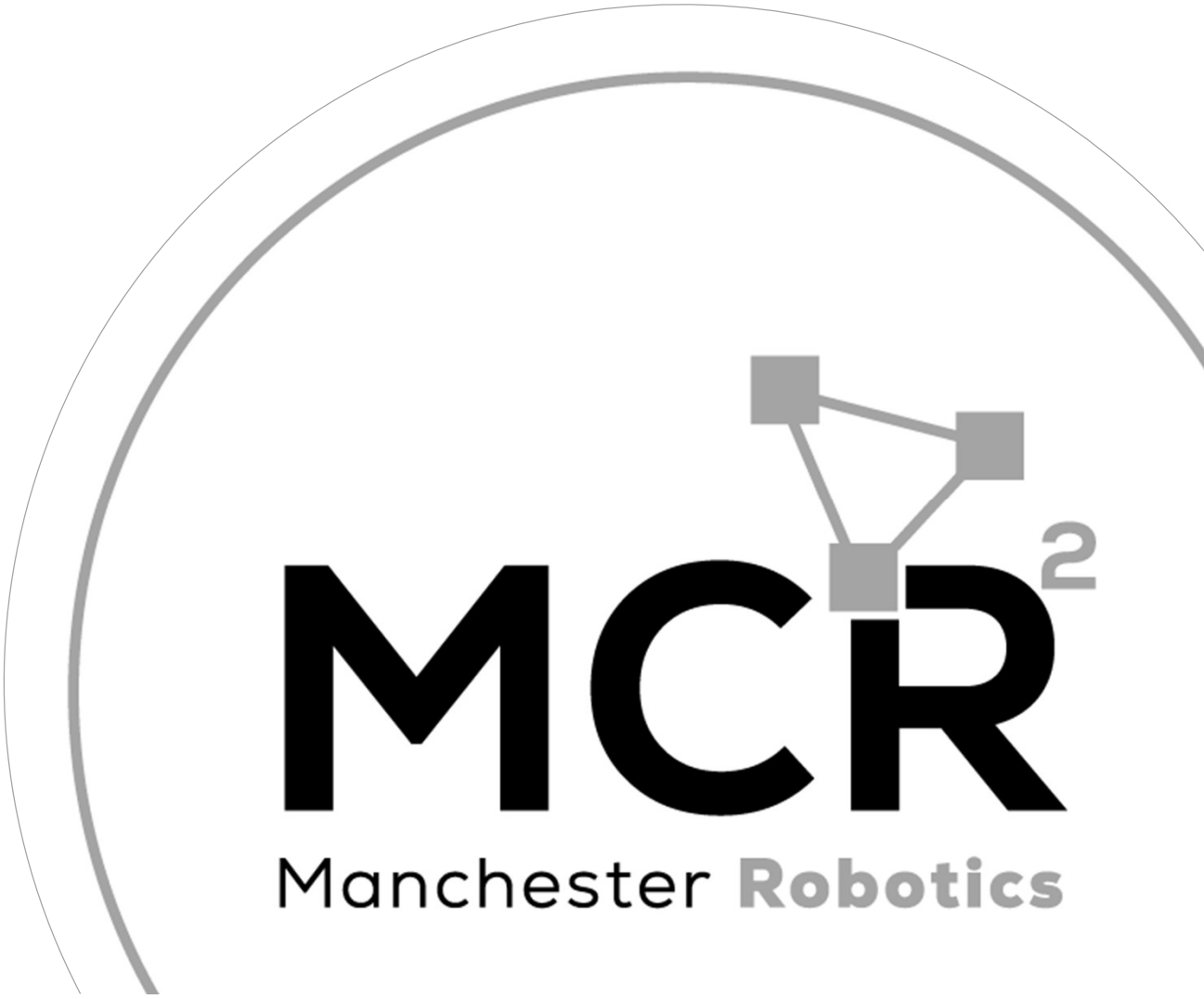
Setup Section:
Set up sensors, variables, Ports, Functions, Serial comms.

Loop Section:
Loops and repeats actions.

ROS Serial Communication

ROS Sketch Structure

{Learn, Create, Innovate};



ROS Sketch Structure

ROS Libraries Arduino

- Rosserial provides a ROS communication protocol that works over Arduino/ESP32 UART.
- Allows Arduino/ESP32 to become a ROS node which can directly publish and subscribe to ROS messages, publish TF transforms, and get the ROS system time.
- To install ROS library for Arduino, follow the installation steps in the ppt:

  MCR2_ROS_ArduinoIDE_Configuration

```
#include <ros.h>

ros::NodeHandle nh;

void setup()
{
  nh.initNode();
}

void loop()
{
  nh.spinOnce();
}
```

ROS Sketch Structure

```
#include <ros.h>

ros::NodeHandle nh;

void setup()
{
  nh.initNode();
}

void loop()
{
  nh.spinOnce();
}
```

Variable Declaration:
Declare the ros library, variables and instantiate the node handle object, messages publishers and subscribers.

Setup Section:
Initialisation of the node, advertise and subscribe to the topics used, initialise variables, ports, functions, etc.

Loop Section:
Loop and repeat actions. Process callbacks if they exist.

Variable declaration

- For every ROS Arduino program, ros.h header must be included before any other library or ROS message.
- Instantiate the node handle, which allows our program to create publishers and subscribers. The node handle also takes care of serial port communications.
- Instantiate any publishers, subscribers to be used.
- Declare any ROS messages and variables to be used.
- Declare/define callback functions to be used with the subscribers.

ROS Sketch Structure

Setup section

- Initialize your ROS node handle. Node initialisation,
- Advertise any topics being published.
- Subscribe to the topics being used.
- Initialise variables, ports, functions, etc.

Loop section

- Run the main program.
- Loop and repeat actions.
- Handle callback functions.

```
#include <ros.h>
ros::NodeHandle nh;
std_msgs::String str_msg;
ros::Publisher pub("foo", &str_msg);

void setup(){
    ...
    nh.advertise(pub);
    ...
}

void loop()
{
   pub.publish( &str_msg );
   nh.spinOnce();
}
```

Variable Declaration:
Declare the ros library, variables and instantiate the node handle object, messages publishers and subscribers.

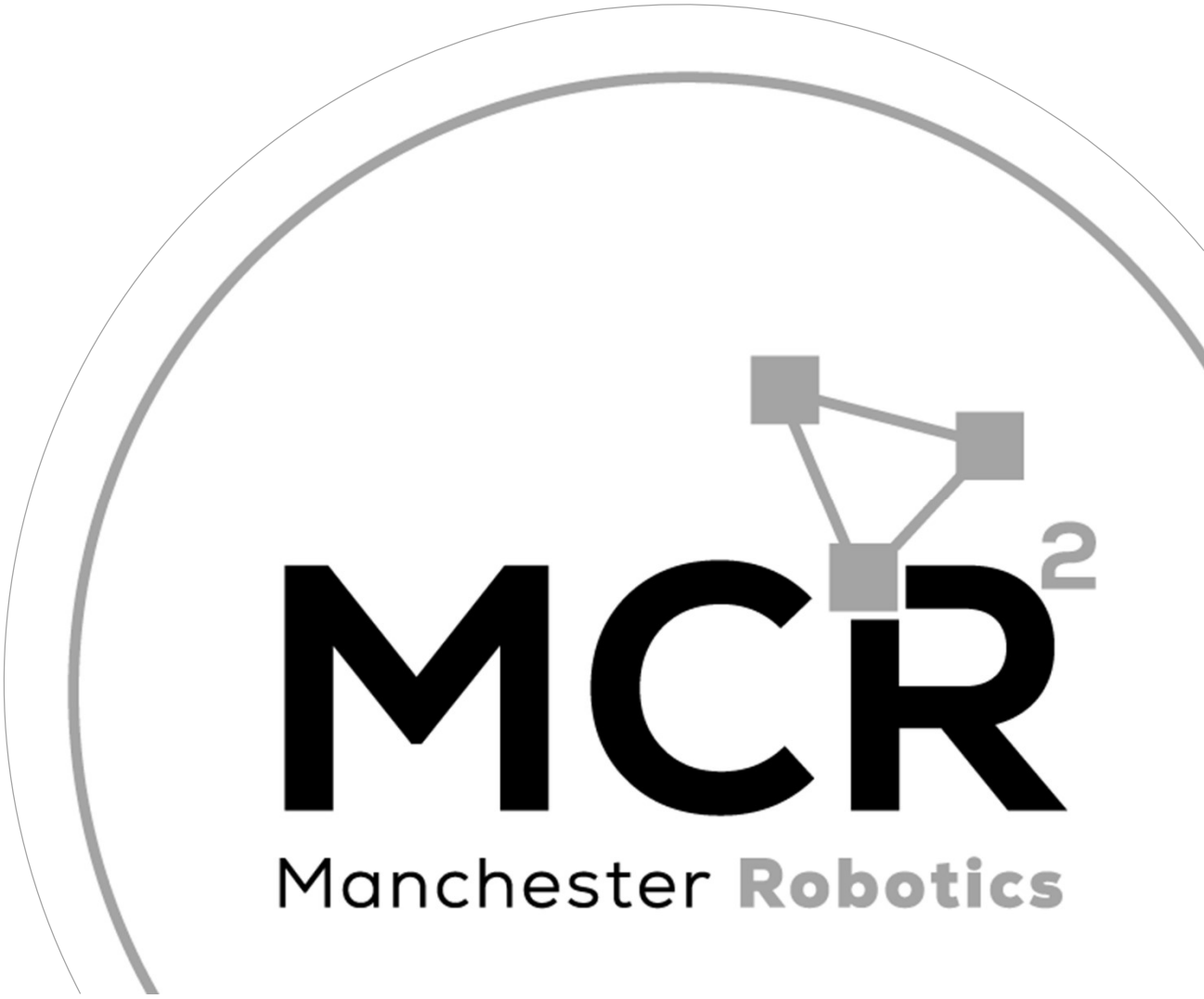
Setup Section:
Initialisation of the node, advertise and subscribe to the topics used, initialise variables, ports, functions, etc.

Loop Section:
Loop and repeat actions. Process callbacks if they exist.

ROS Serial Communication

Activity

{Learn, Create, Innovate};



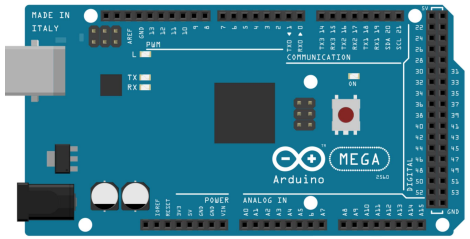
Activity

Requirements

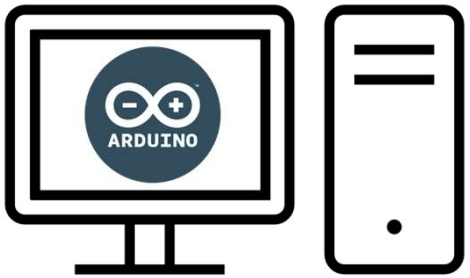
- The following activity is based on the example tutorial “Hello World” which can be found here.
- This activity requires Arduino IDE to be installed. Follow the tutorial on how to install it in presentation:

  MCR2_ROS_ArduinoIDE_Configuration

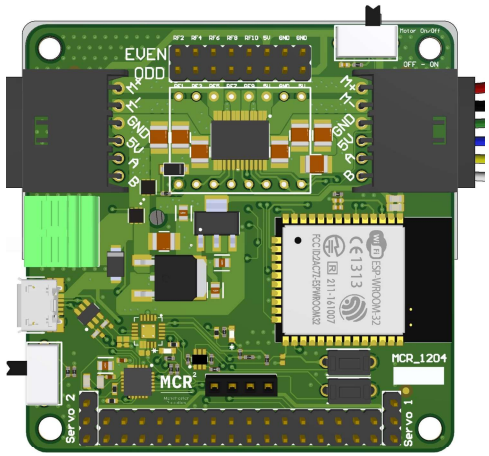
- For this activity is required to have a Hackerboard or an Arduino Mega.



Arduino Mega



Computer

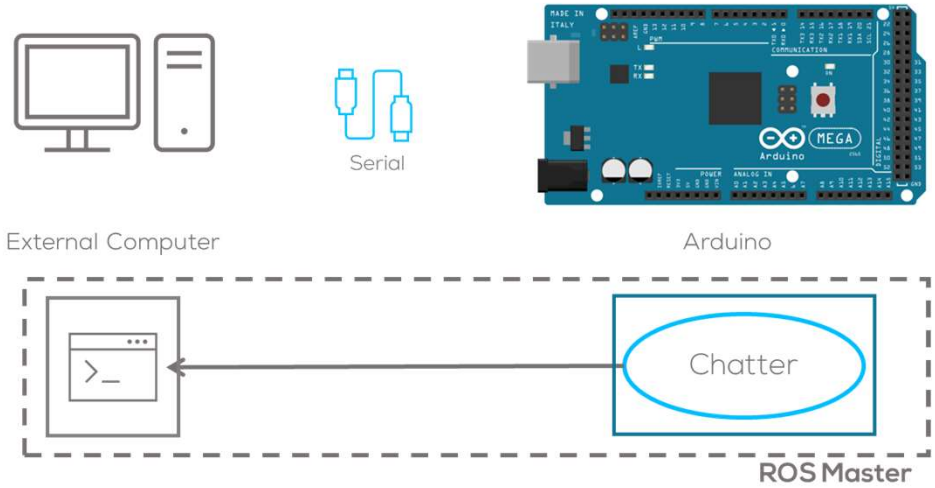


Hackerboard

Activity

Description

- In this activity, a node running a simple publisher will be made.
- This node will run inside the microcontroller and will communicate with the computer through serial communication using the rosserial protocol.
- The node will publish a simple string message “Hello World”, which will be read by the computer using a ros command line for simplicity.
- This activity will be divided into two parts. The first part involving Arduino IDE for programming the MCU.
- The second part involving the commands required to connect to the MCU to the computer.



Activity

```
#include <ros.h>
#include <std_msgs/String.h>
```

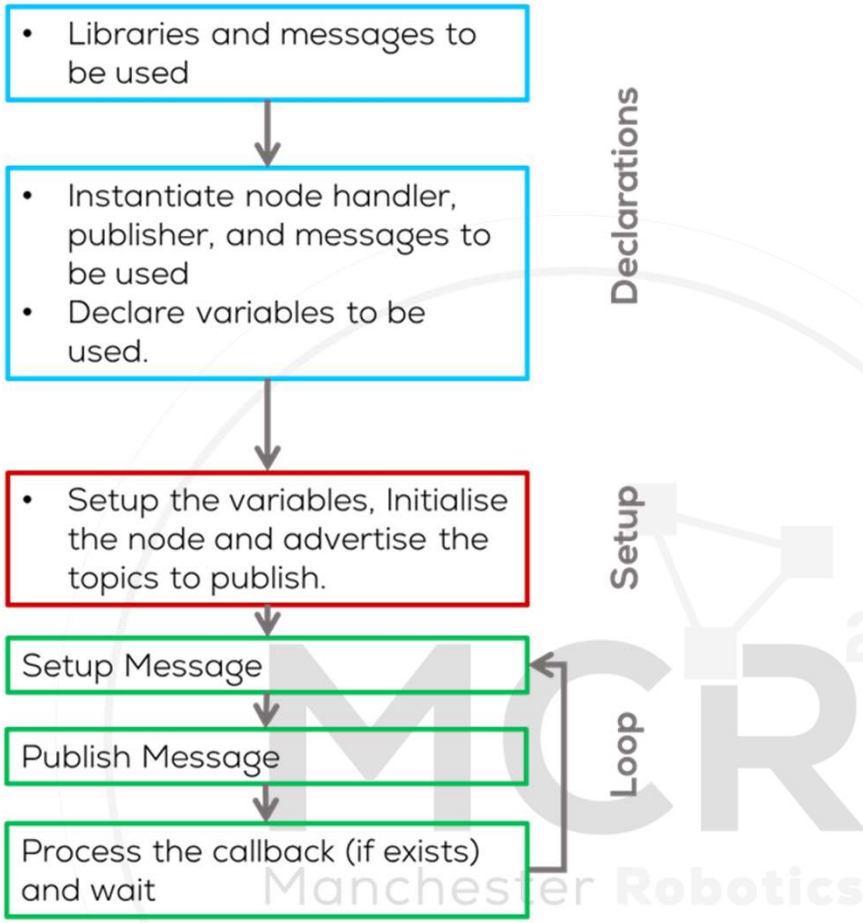
```
ros::NodeHandle  nh;

std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg);

char hello[13] = "hello world!";
```

```
void setup()
{
  nh.initNode();
  nh.advertise(chatter);
}
```

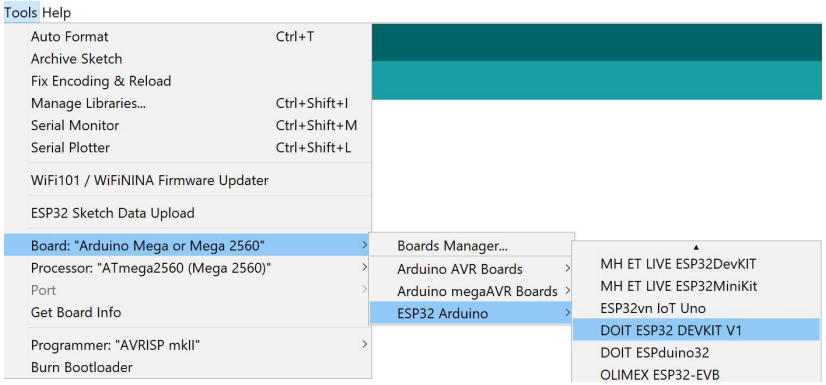
```
void loop()
{
  str_msg.data = hello;
  chatter.publish( &str_msg );
  nh.spinOnce();
  delay(1000);
}
```

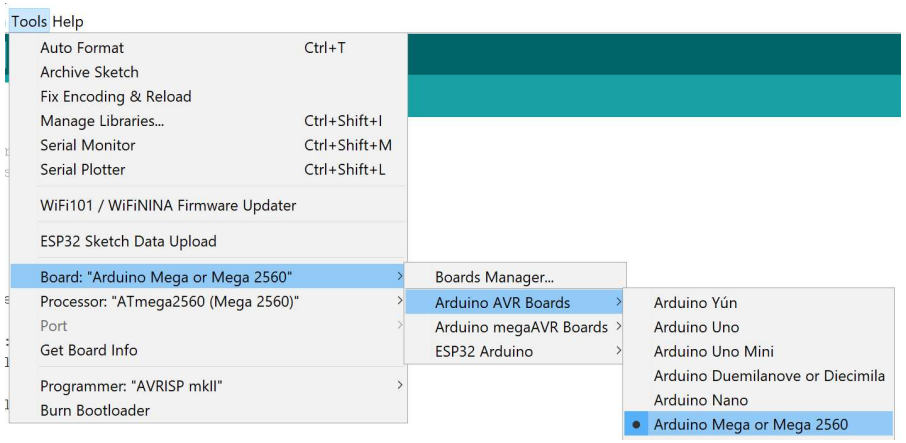


Activity

Compilation (Arduino IDE)

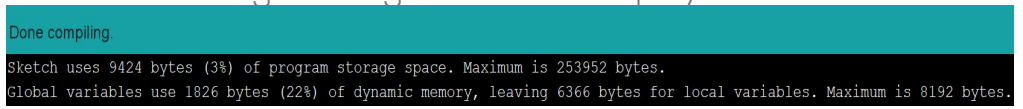
- Open Arduino IDE (previously configured).
- Type the code in the previous slide.
- Select the board to be used Tools>Board ESP32 for Hackeboard or Arduino Mega
  - For Arduino Select Arduino AVR Boards>Arduino Mega or Mega 2560
  - For Hackerboard select ESP32 Arduino > DOIT ESP32 DEVKIT V1





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

- The following message should be displayed:

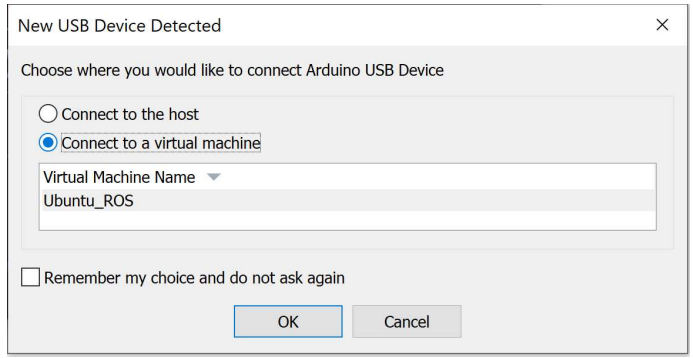


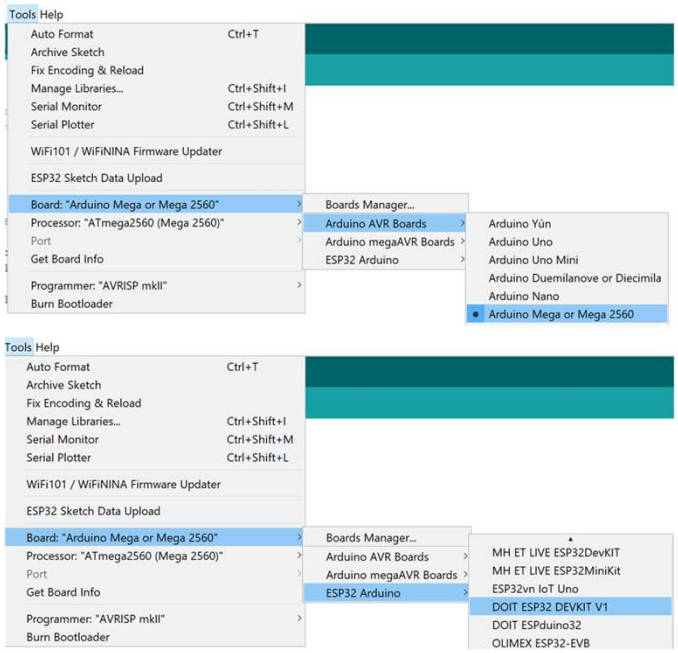
- For compilation errors or troubleshoot with the libraries, see presentation MCR2_ROS_Arduino_IDE_Configuration.

Activity

Uploading (Arduino IDE)

- Connect the 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.
- Select the board to be used Tools>Board
  - For Arduino Select Arduino AVR Boards>Arduino Mega or Mega 2560
  - For Hackerboard select ESP32 Arduino > DOIT ESP32 DEVKIT V1



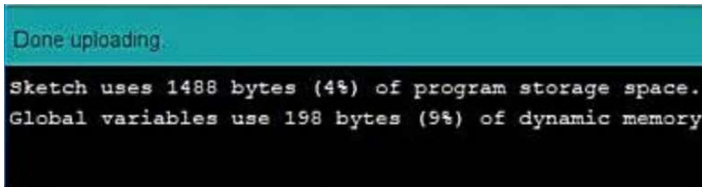


Activity

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.
  - 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*
```

- Launch the roscore on a new terminal

```
roscore
```

Activity

Uploading (Arduino IDE)

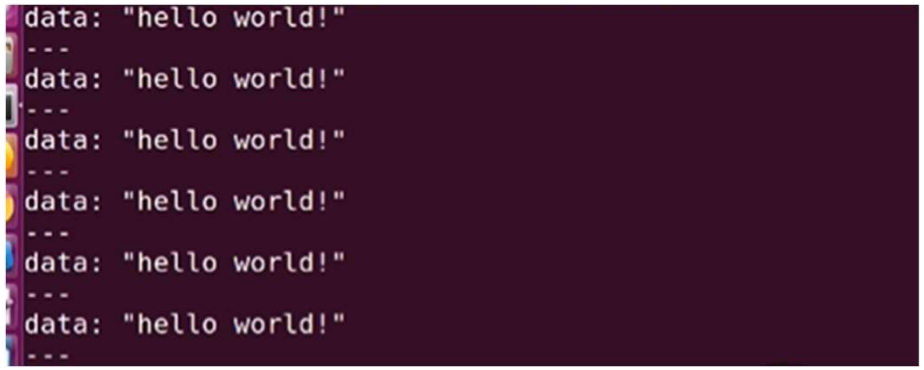
- In a new terminal use the command line tool rosrun and select the port type (USB or ACM).

```
rosrun rosserial_python serial_node.py /dev/ttyUSB0
```

- In a new terminal subscribe to the topic using the command

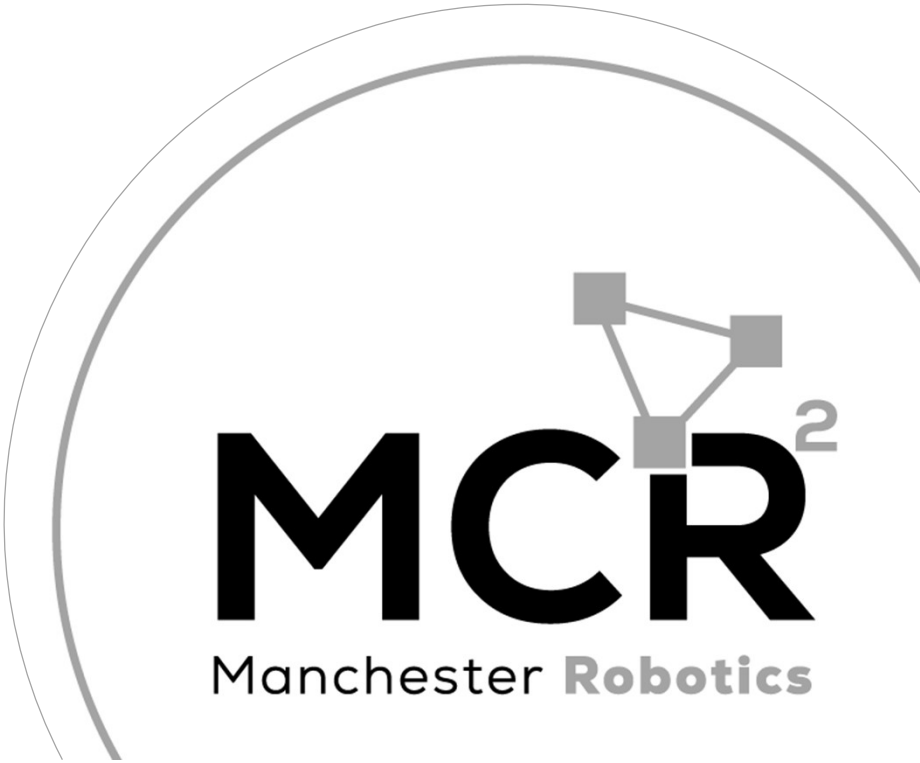
```
rostopic echo chatter
```

- More information can be found here.
- You should see the following results



Thank you

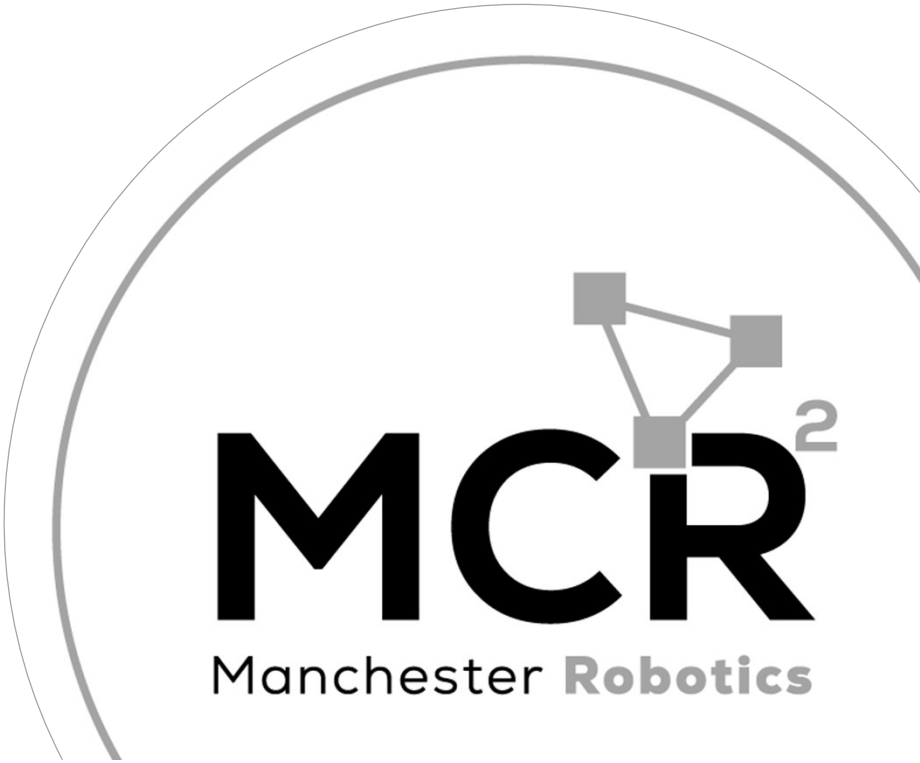
{Learn, Create, Innovate};



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.*