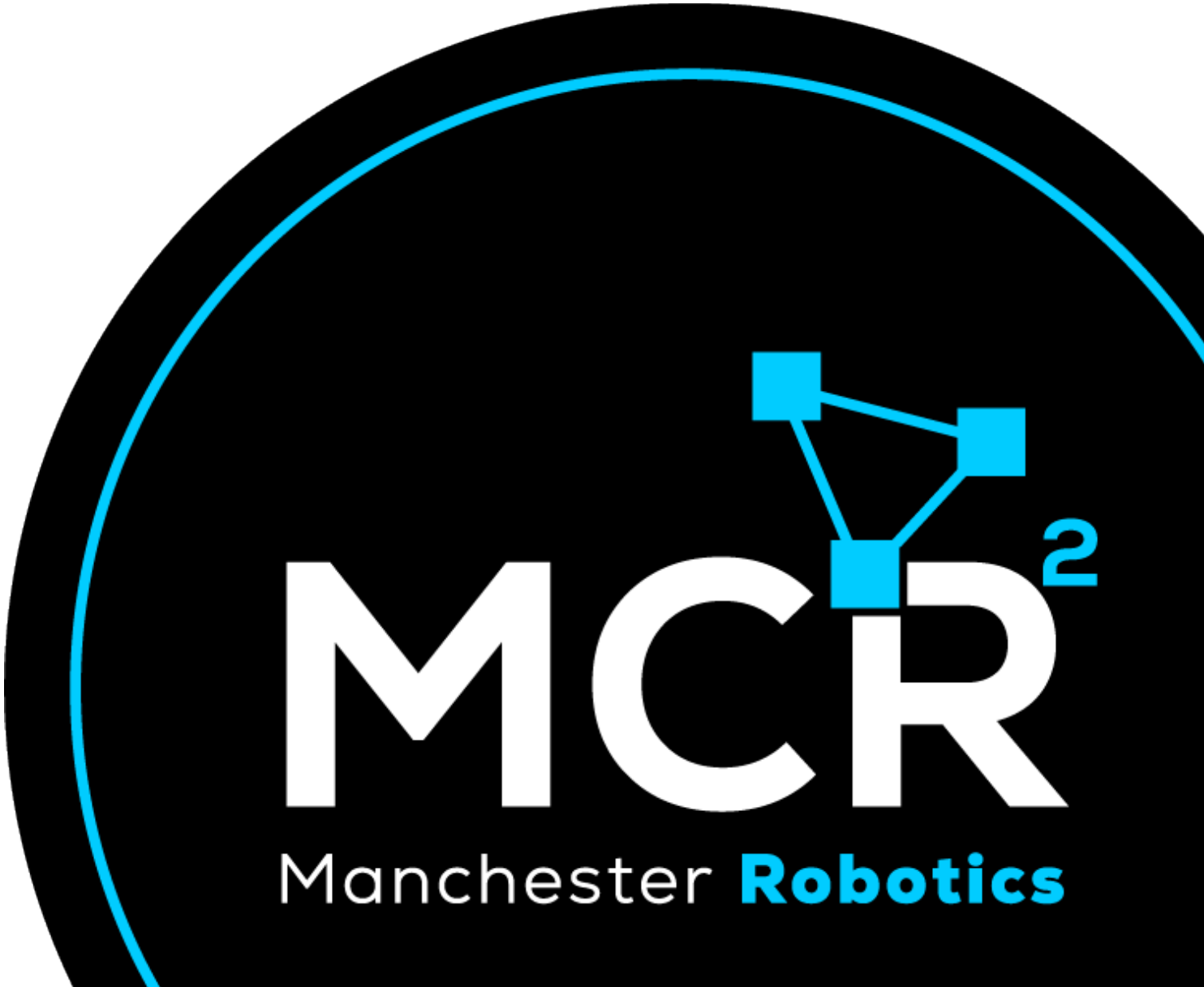
# Final Challenge

*{Learn, Create, Innovate};*



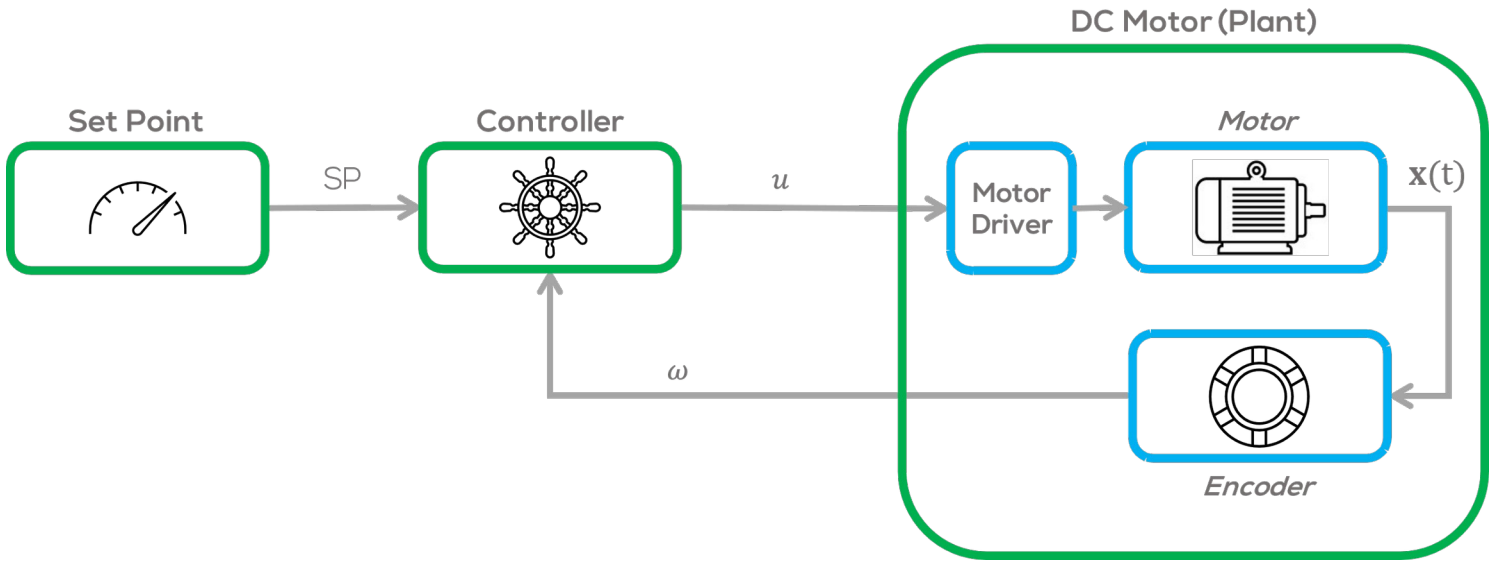
# Final Challenge

## Introduction

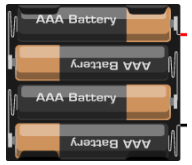
This challenge is intended for the student to review the concepts introduced in this course.

The activity consists in controlling the speed of a DC Motor.

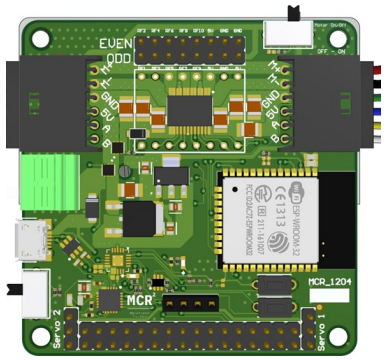
- The motor speed, must be controlled using an external computer, a microcontroller, and a motor driver.
    - See following slide for requirements.



# Mini Challenge 3: Requirements

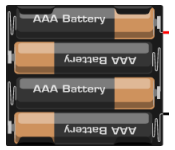



Battery Pack
5 - 12 V

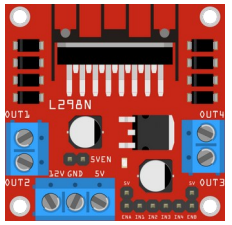



Hackerboard

6 VDC
Brushed
Motor




Battery Pack
5 - 12 V




Motor Driver
L298n

Wires
(Dupont or
any wire)

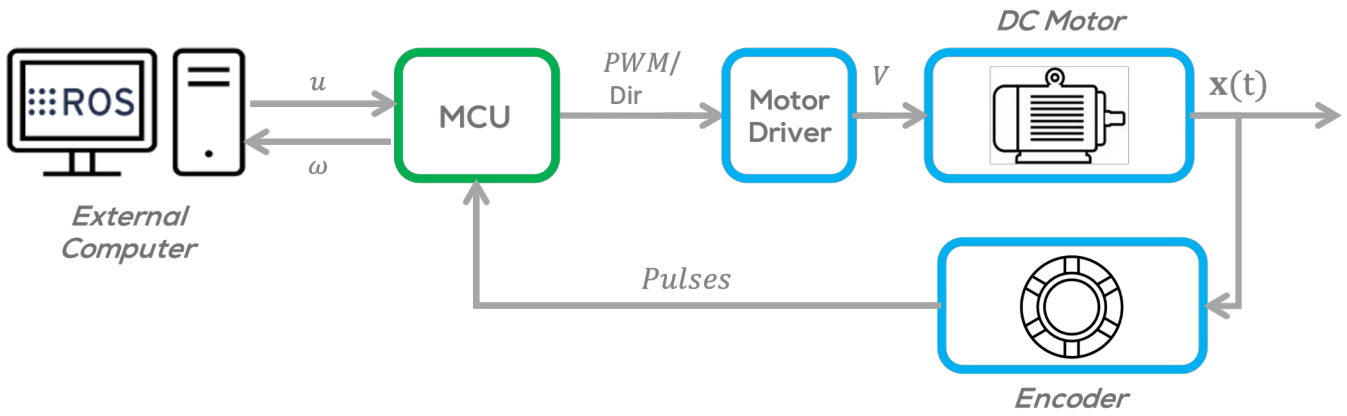
ESP32

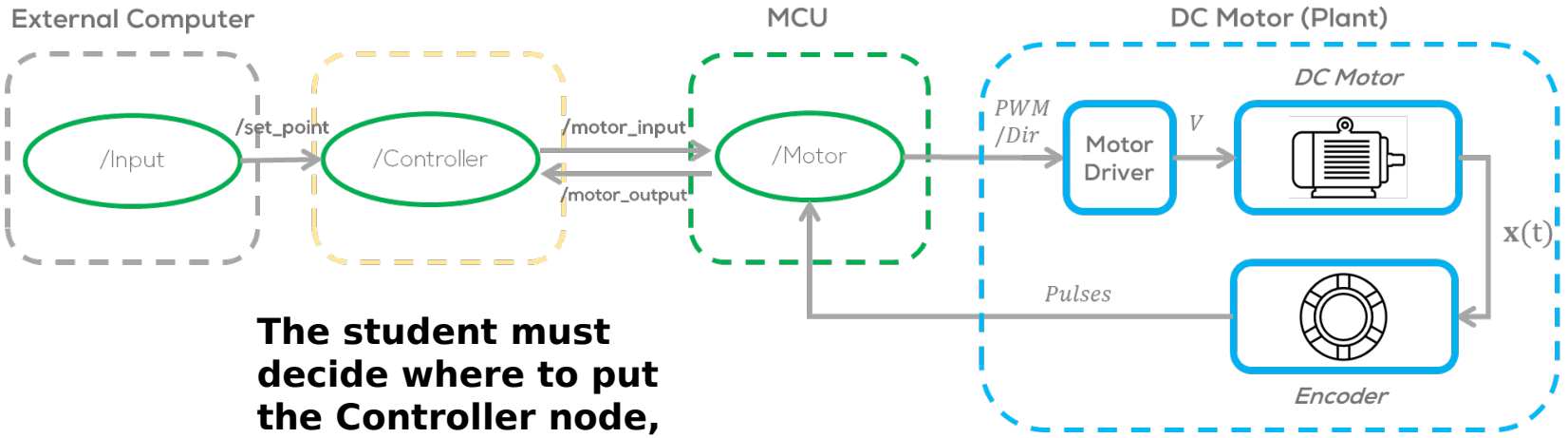
6 VDC
Brushed
Motor

# Final Challenge

## Structure

- As stated before, for this challenge the student is required to  develop a speed control for a DC motor.
- To perform this task, at least tree nodes must be developed: /Input, /Controller and /Motor.
- The /Input node must on the external computing unit the **/Controller Node can run on the microcontroller, or in the External Computing Unit** the /Motor node must run in the **microcontroller**.



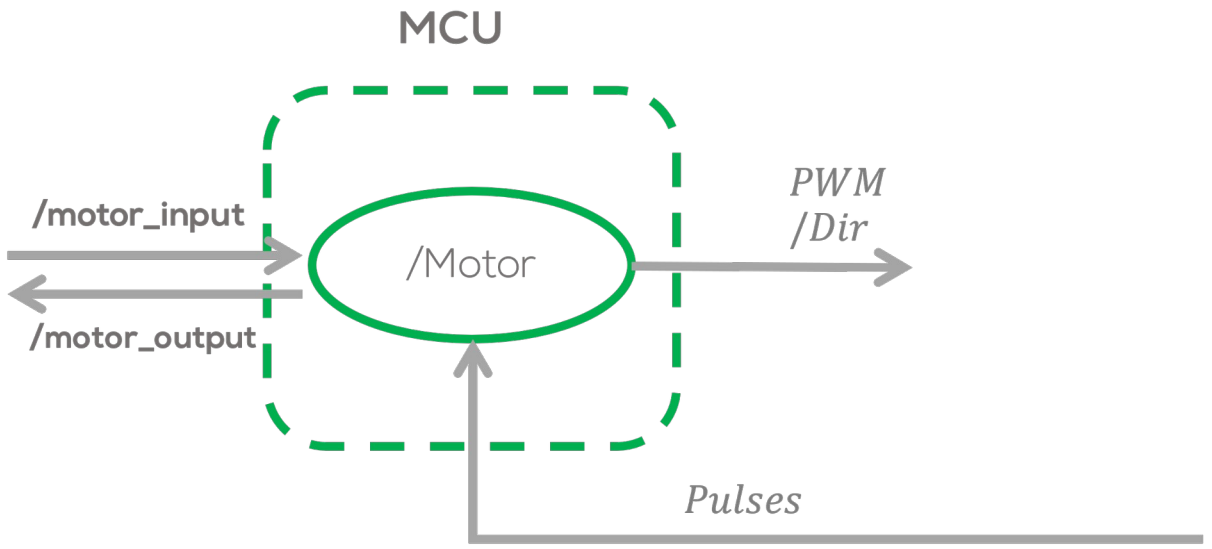




**The student must decide where to put the Controller node, in the external computing Unit or in**

# Final Challenge

## Motor Node

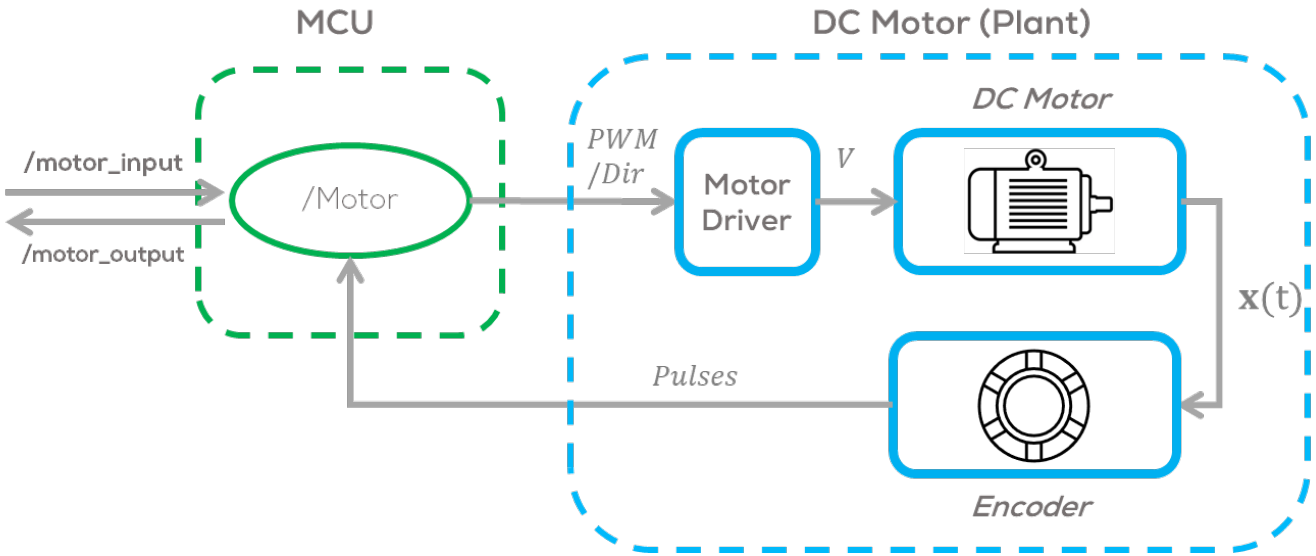
- The “/Motor” node must run on the MCU (Hackerboard or ESP32).
- This node should act as a transducer, to send information for the motor driver and at the same time, estimating the speed of the motor to be sent to the controller.
    - The node must subscribe and translate the control input “” in the “/motor_input” topic to a PWM and direction signals to the Motor Driver.
    - The node must estimate the motor speed using the information form the encoders and publish the data in the topic “/motor_output”.



# Final Challenge

## Motor Node

- The output of the MCU to the motor driver must be a “PWM signal” and a direction signal.
  - The PWM duty cycle and direction must be mapped to the interval .
    - Where the sign represents the direction of the motor, i.e., (+) CCW rotation, and (-) CW rotation of the motor.
    - The duty cycle percentage (%) range .
- The output of the “/Motor” node (publish) to the controller, must be the estimated angular velocity of the motor in ,in the range . Where the sign (+.-) represents the direction of rotation obtained from the dual encoder channel, and the is the angular speed, obtained by counting the pulses of the encoder for a certain period of time.

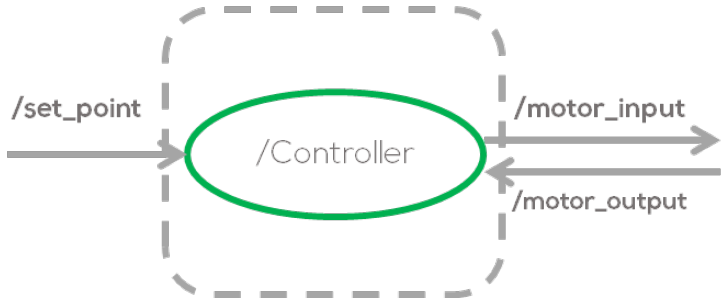



*A simple filter example for a noisy signal can be found here

# Final Challenge

## Controller Node

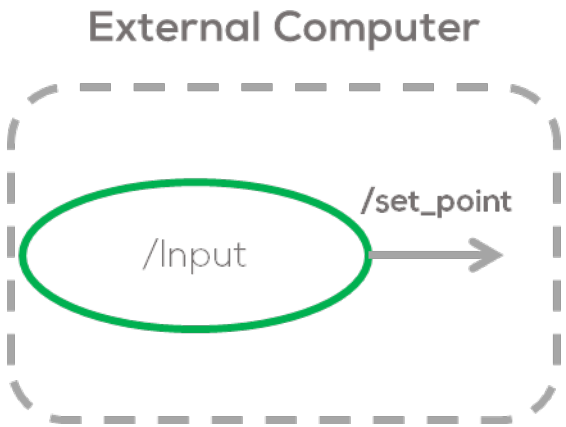
1. Make a node called /Controller” in the External Computing Unit or in the microcontroller, to generate a control input to the “/Motor” node.
2. The node must publish in the “/motor_input” topic and subscribe to the “/motor_output” and “/set_point” topics.
3. The output of the controller “/motor_input” must be bounded between in the interval -1 to 1 i.e., .
5. The control node, must use a parameter file, for all the required tunning variables.
6. The controller can be  “P”, “PI” or “PID” controller (other controllers can be accepted upon agreement with the professor.).
7. The sampling time and rate must be defined by the student (Check Hints).
8. **It is strictly forbidden to use any other python library, other than NumPy. The controller must be made without using any predefined online controllers.**



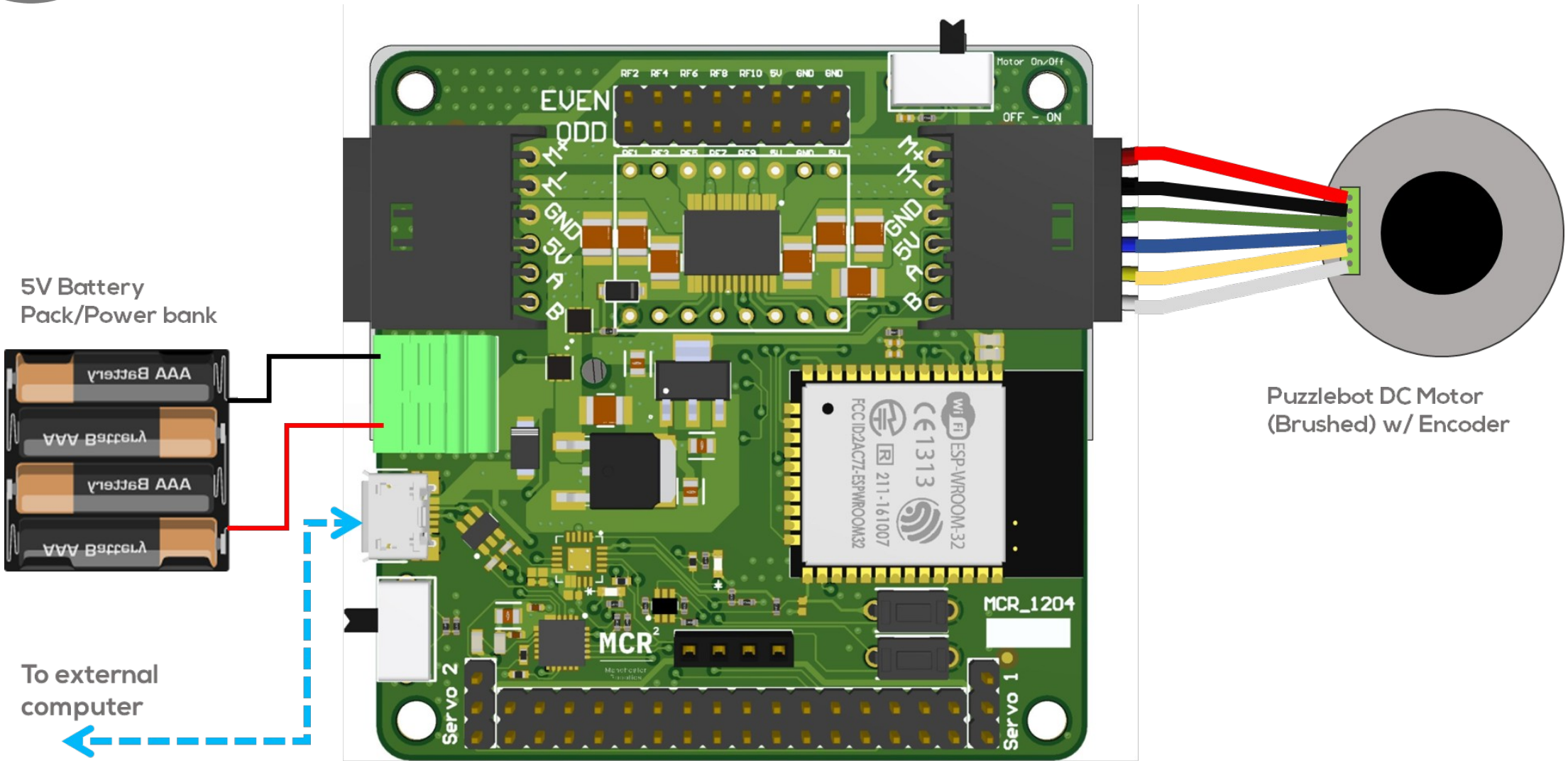
# Final Challenge

## Input Node

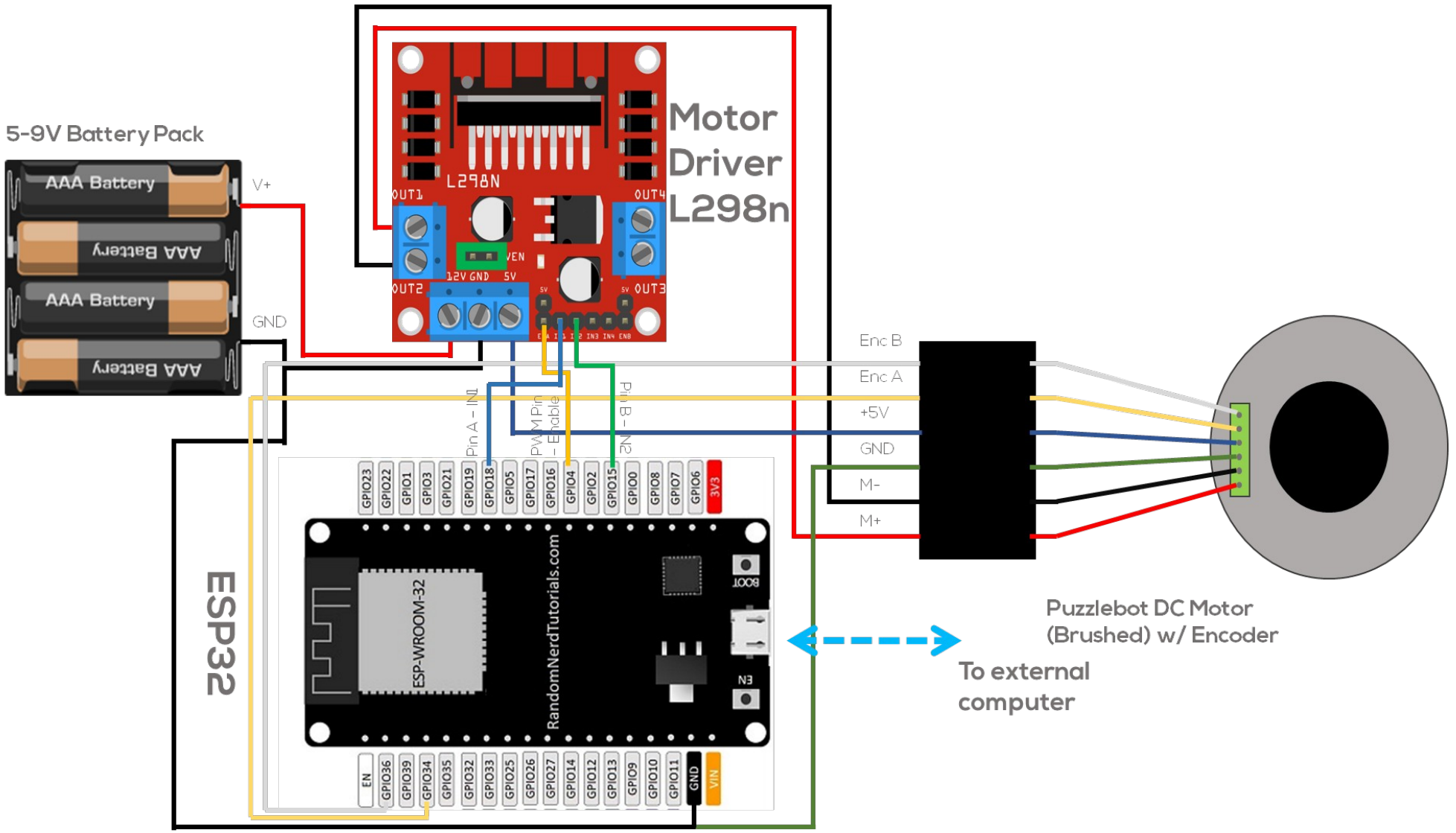
- The “/Input” node must publish into the previously defined topic “/set_point”.
  - The node must publish different input signals (step, square wave, sinusoidal) to the motor.
  - The user must be able to choose and characterise these signals these signals using a flags and parameters in the parameter file.
  - As before It is forbidden to use any libraries, except from NumPy for this exercise.
- Make the necessary plots to analyse the system in rqt_plot



# Final Challenge: Connection Diagrams

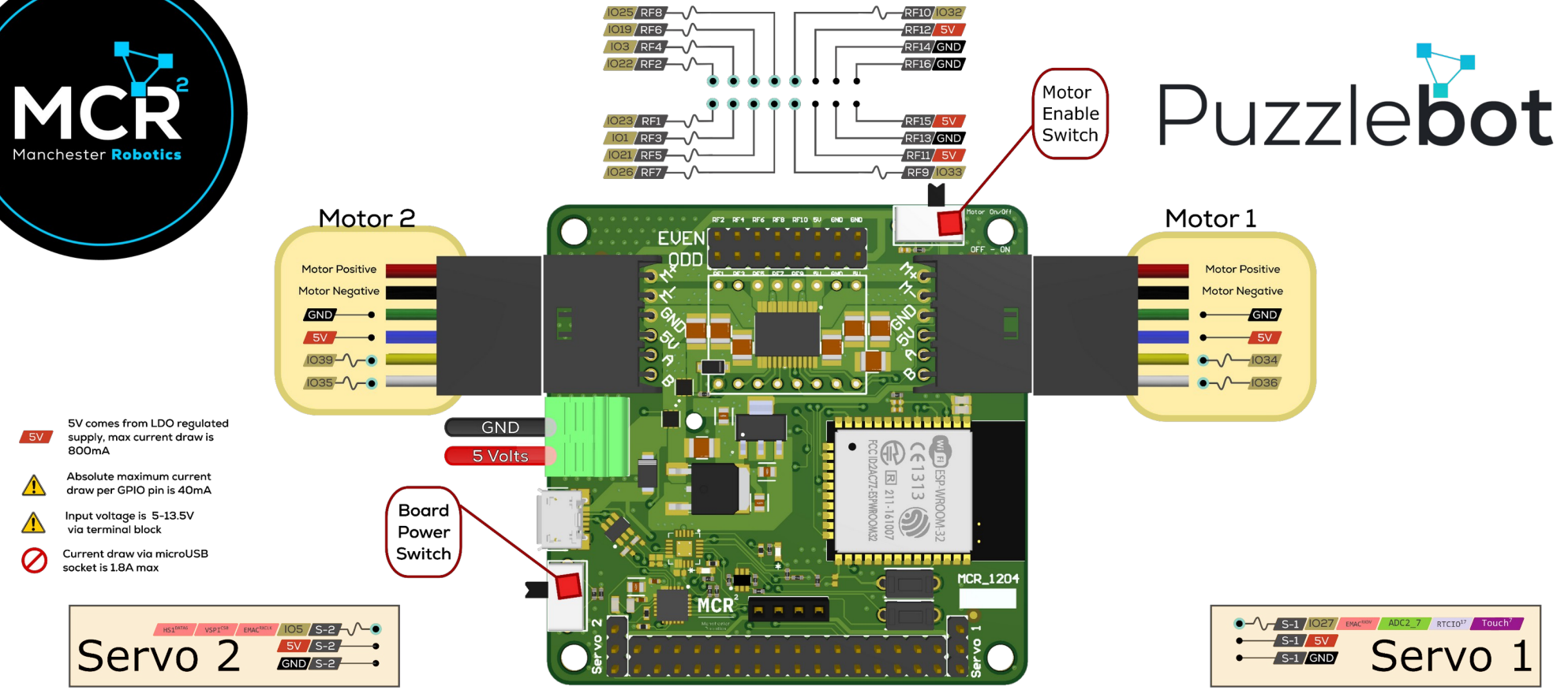


# Final Challenge: Connection Diagrams



MCR² Manchester Robotics

# Puzzlebot




5V comes from LDO regulated supply, max current draw is 800mA

Absolute maximum current draw per GPIO pin is 40mA

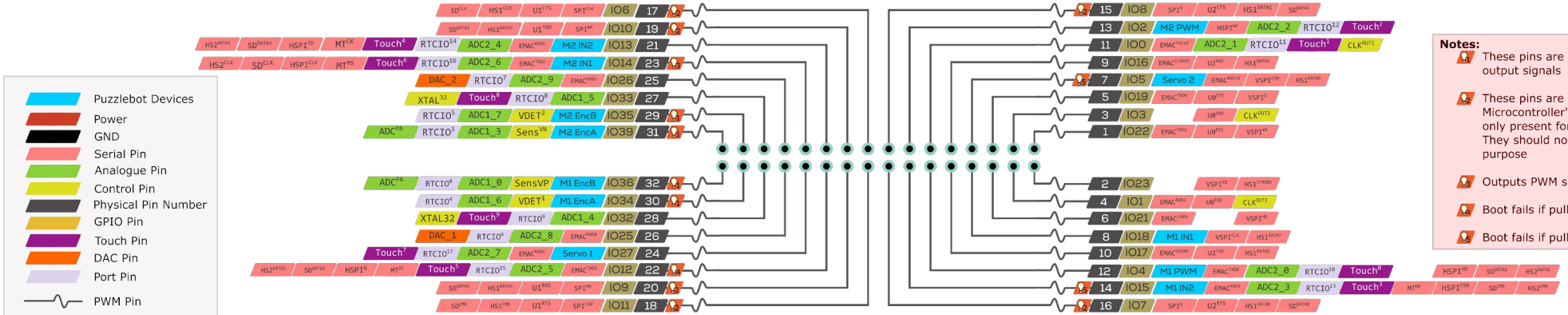
Input voltage is 5-13.5V via terminal block

Current draw via microUSB socket is 1.8A max










**Notes:**

- These pins are ***input only***. They cannot output signals
- These pins are integrated to the Microcontroller's flash memory, and are only present for debugging purposes. They should not be used for general purpose
- Outputs PWM signal at boot time
- Boot fails if pulled high
- Boot fails if pulled low

**https://www.manchester-robotics.com**



01 FEB 2022
v1

# Final Challenge

## Hints

Discrete PID controller:

where  are the controller output and error at time step , such that time  where  is the sampling time.  are the proportional, integral and derivative gains, respectively. More information here.

Sampling Time:

- This can be stablished by applying an input value of 1 to the motor in open loop, using a small sampling time (e.g. 5 ms), and plot the values of the output speed. The sampling time can be approximated using the following expression:

,

where  is the settling time of the system.

# Final Challenge

## Hints (ESP32 PWM)

- The ESP 32 does not use the “analogwrite()” command to generate a PWM signal.
- To generate a PWM signal the following steps must be followed.

1. Choose a PWM channel. There are 16 channels from 0 to 15.
2. Set the PWM signal frequency (980 Hz standard).
3. Set the signal’s duty cycle resolution. From 1 to 16 bits resolution (8 bits typical, 0 to 255 resolution).
4. Specify the output GPIO.
5. More information can be found here.

```
void setup(){

ledcSetup(ledChannel, freq, resolution);

ledcAttachPin(ledPin, ledChannel);

}


void loop(){

ledcWrite(ledChannel, dutyCycle);

delay(15);

 }
```

# Mini Challenge 4

## Hints (Arduino/ESP32 Interrupts)

- The Arduino "`attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)`" establishes an interrupt, to a designated pin, triggered by different modes.
- pin: the Arduino pin where the interrupt is expected.
- ISR: the function to call when the interrupt occurs; this function must take no parameters and return nothing.
- Modes:
  - LOW to trigger the interrupt whenever the pin is low,
  - CHANGE to trigger the interrupt whenever the pin changes value
  - RISING to trigger when the pin goes from low to high,
  - FALLING for when the pin goes from high to low.
- When using encoders, some variables need to be shared between the ISR and the other function for this case we use volatile variables.
- A volatile variable is a variable that is marked or cast with the keyword "volatile" so that it is established that the variable can be changed by some outside factor, such as the operating system or other software.
- In other words, volatile variables, tell the compiler to recheck the value of the variable from RAM memory (not a copy in other memory) because the value might change due to an external factor in this case the encoder.
- More information about interrupts for Arduino can be found here, here and here.
- More information about interrupts for ESP32 can be found here, here and here.

# Mini Challenge 4

Both example codes, read a switch connected to an input and toggle the output value. For both cases, an LED is connected to the output therefore it changes the state of the LED.

```
//Arduino Interrupts Example
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
   pinMode(ledPin, OUTPUT);
   pinMode(interruptPin, INPUT_PULLUP);
   attachInterrupt(digitalPinToInterrupt(interruptPin), blink,
CHANGE);
 }
void loop() {
   digitalWrite(ledPin, state);
}

void blink() {
   state = !state;
}
```

```
//ESP32 Interrupts Example
#define pushButton_pin 33
#define LED_pin 32

void IRAM_ATTR toggleLED() {
 digitalWrite(LED_pin, !digitalRead(LED_pin));
}

void setup() {
 pinMode(LED_pin, OUTPUT);
 pinMode(pushButton_pin, INPUT_PULLUP);
 attachInterrupt(pushButton_pin, toggleLED, RISING);
}

void loop() {
 }
```

*** In this example, as the Arduino example, the toggle of the LED can be done using a volatile variable.

# Rules

- This is challenge **not** a class. The students are encouraged to research, improve tune explain their algorithms by themselves.
- MCR2(Manchester Robotics) Reserves the right to answer a question if it is determined that the questions contains partially or totally an answer.
- The students are welcomed to ask only about the theoretical aspect of the classed.
- No remote control or any other form of human interaction with the simulator or ROS is allowed (except at the start when launching the files).
- It is **forbidden** to use any other internet libraires with the exception of standard libraires or NumPy.
- If in doubt about libraires please ask any teaching assistant.
- Improvements to the algorithms are encouraged and may be used as long as the students provide the reasons and a detailed explanation on the improvements.
- All the students must be respectful towards each other and abide by the previously defined rules.
- Manchester Robotics reserves the right to provide any form of grading. Grading and grading methodology are done by the professor in charge of the unit.