Half-term Challenge

Manchester Robotics

{Learn, Create, Innovate};

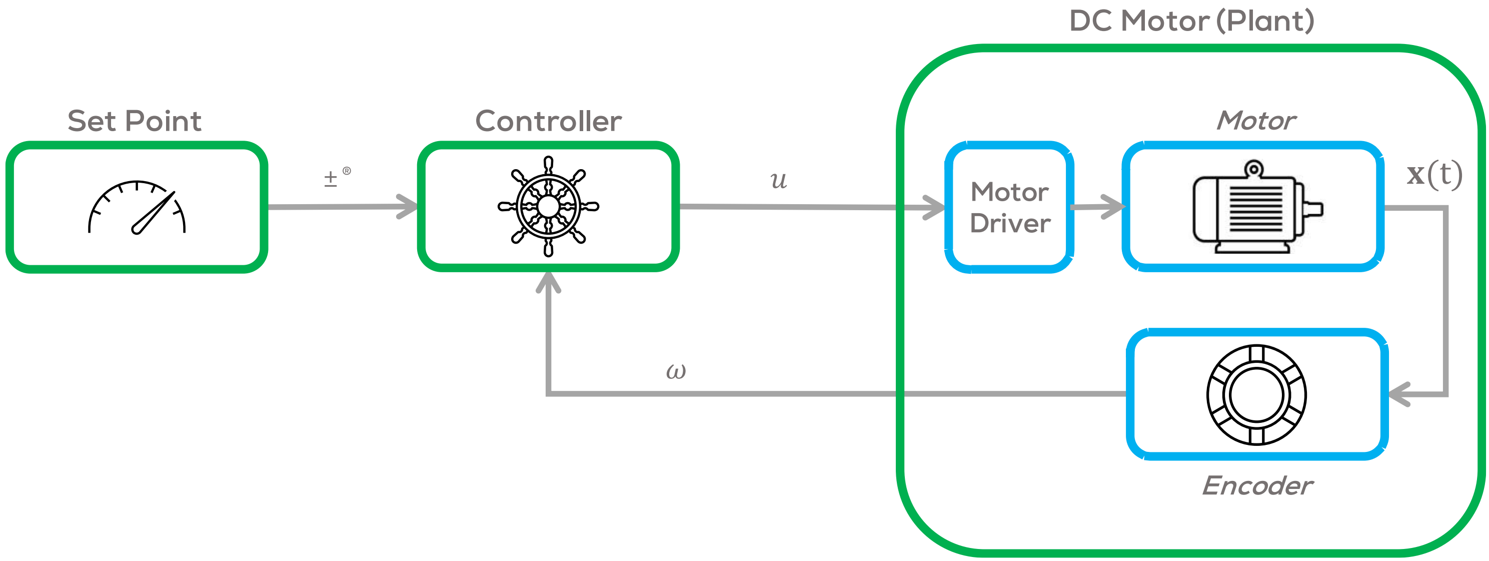
Half-term Challenge

Introduction

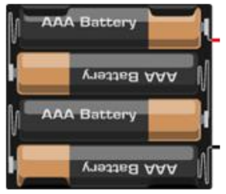
This challenge is intended for the student to review all the concepts introduced in this course.

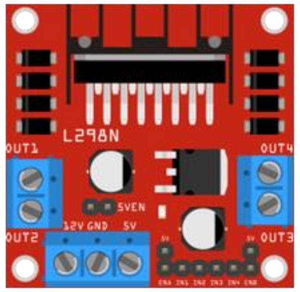
The activity consists of controlling the speed of a DC Motor.

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

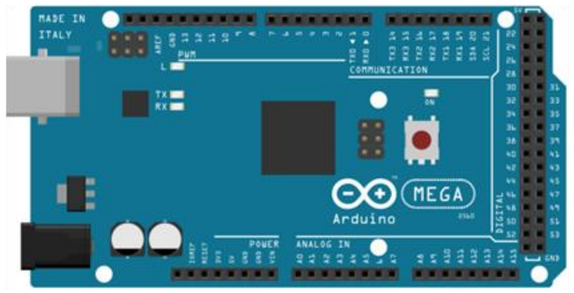


Half-term Challenge


Battery Pack
5 - 12 V


Motor Driver
L298n
(Steren)

Wires (Dupont or any wire)

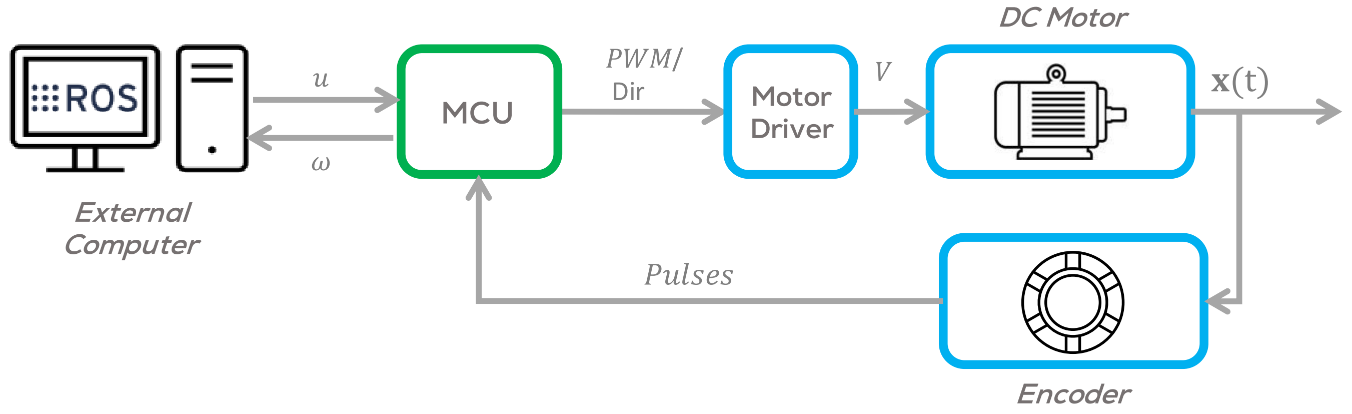

Arduino Mega

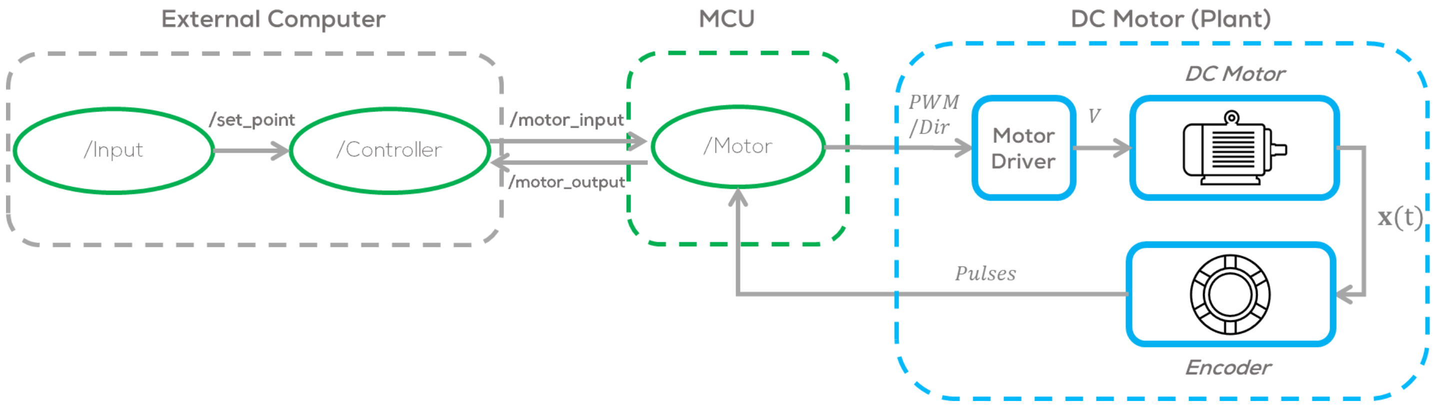
6 VDC Brushed Motor

Half-term 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 and /Controller Nodes must run in the **external computing unit**, whilst the /Motor node must run in the **microcontroller**.

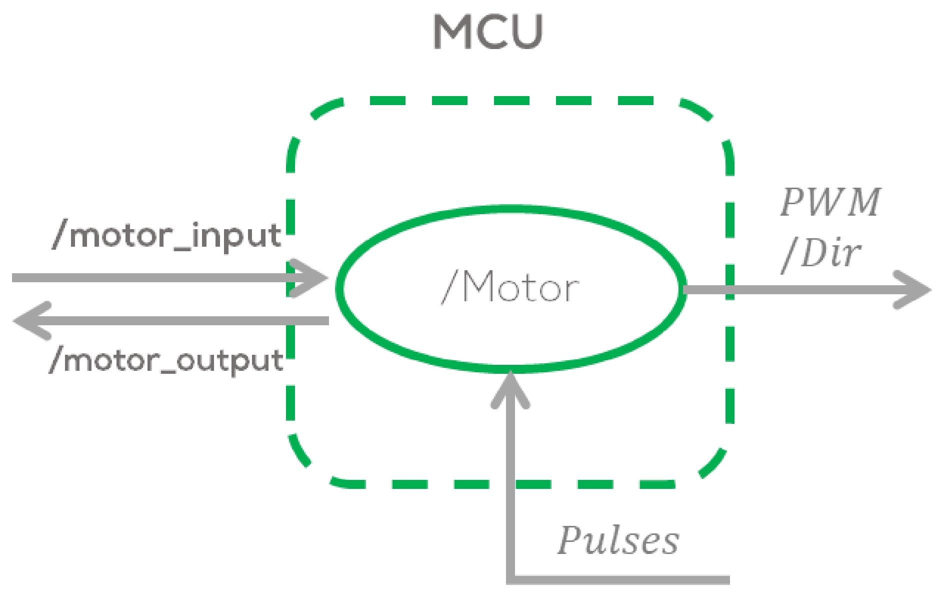




Half-term Challenge

Motor Node

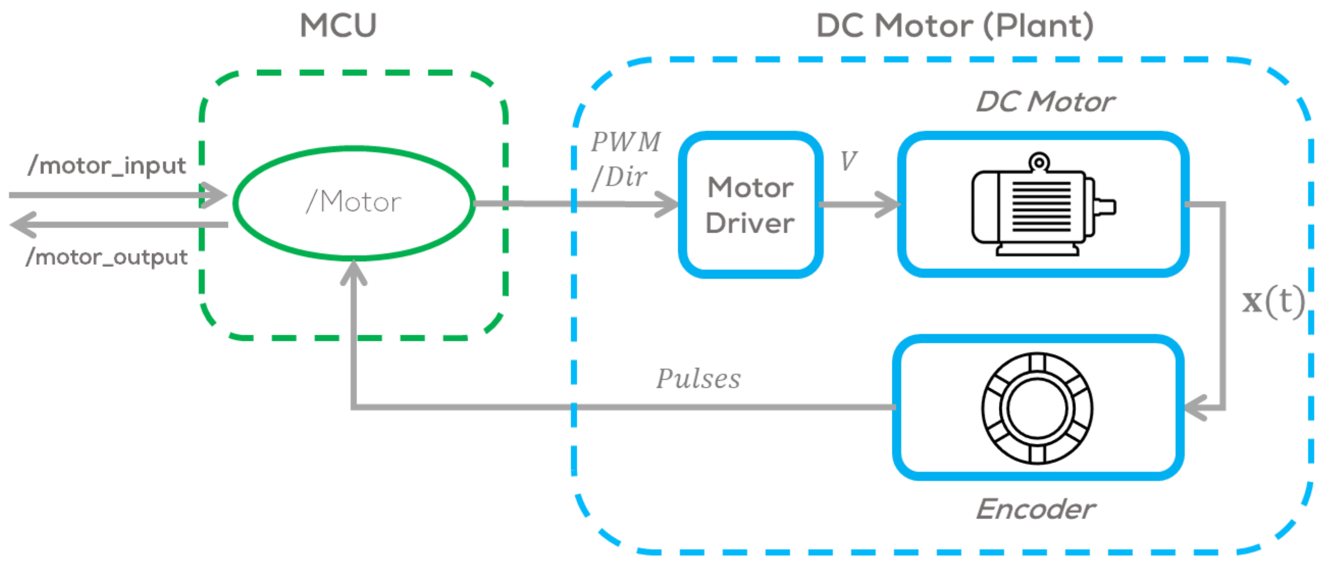
- The "/Motor" node must run on the MCU (Hackerboard or Arduino).
- This node should act as a transducer, to send information to the motor driver and, at the same time, estimate the speed of the motor to be sent to the controller.
  - The node must subscribe and translate the control input "$u(t)$" in the "/motor_input" topic to a PWM signal to the Motor Driver.
  - The node must estimate the motor speed using the encoders' information and publish the data in the topic "/motor_output".



Half-term Challenge

Motor Node

- The output of the MCU to the motor driver must be a “PWM signal” to the motor driver.
    - The PWM duty cycle, must be mapped to the interval $[0, 1]$.
        - The duty cycle percentage (%) range $[0,100]\% \rightarrow [0,1]$.
- The output of the “/Motor” node (publish) to the controller, must be the estimated angular velocity of the motor in $[\frac{rad}{s}]$,in the range $[0, \omega_{max}]$. Where the ω is the angular speed, obtained by counting the pulses of the encoder for a certain period of time.

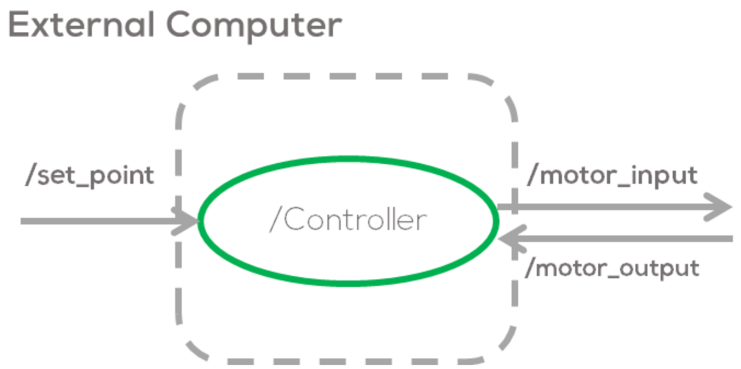


*A simple filter example for a noisy signal can be found here

Half-term Challenge

Controller Node

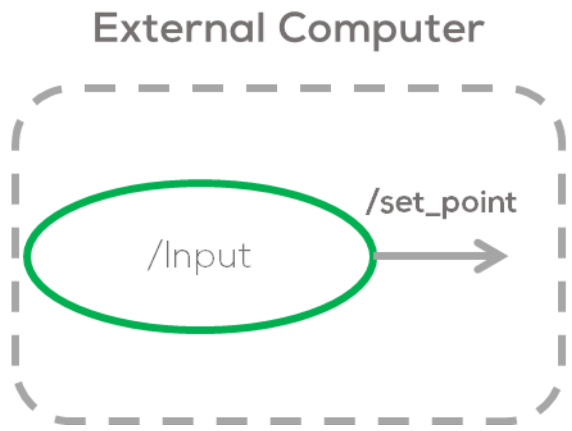
1. Make a node called /Controller" in the External Computing Unit, to generate a control input to the "/Motor" node.
2. The node must publish in the "/motor_input" topic and subscribe to the "/motor_output" and "/set_point" topics.
3. The output of the controller "/motor_input" must be bounded between in the interval 0 to 1 i.e., $u(k) \in [0,1]$.
4. The message for the "/set_point" topic <u>must be defined by the student.</u>
    1. Must be a custom message.
    2. Must include at least 2 different variables.
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. <u>It is strictly forbidden to use any other python library, other than NumPy. The controller must be made without using any predefined online controllers.</u>



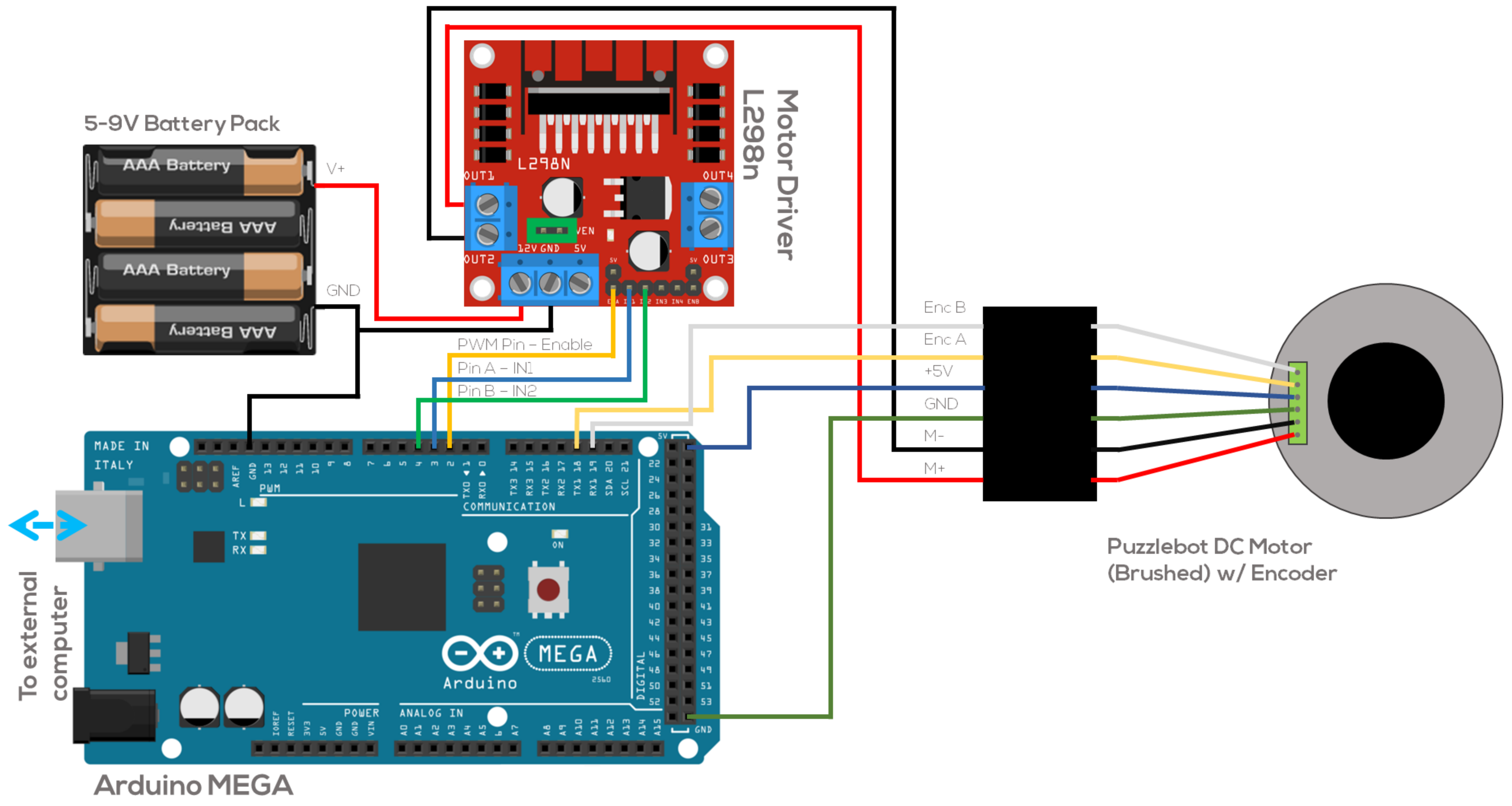
Half-term Challenge

Input Node

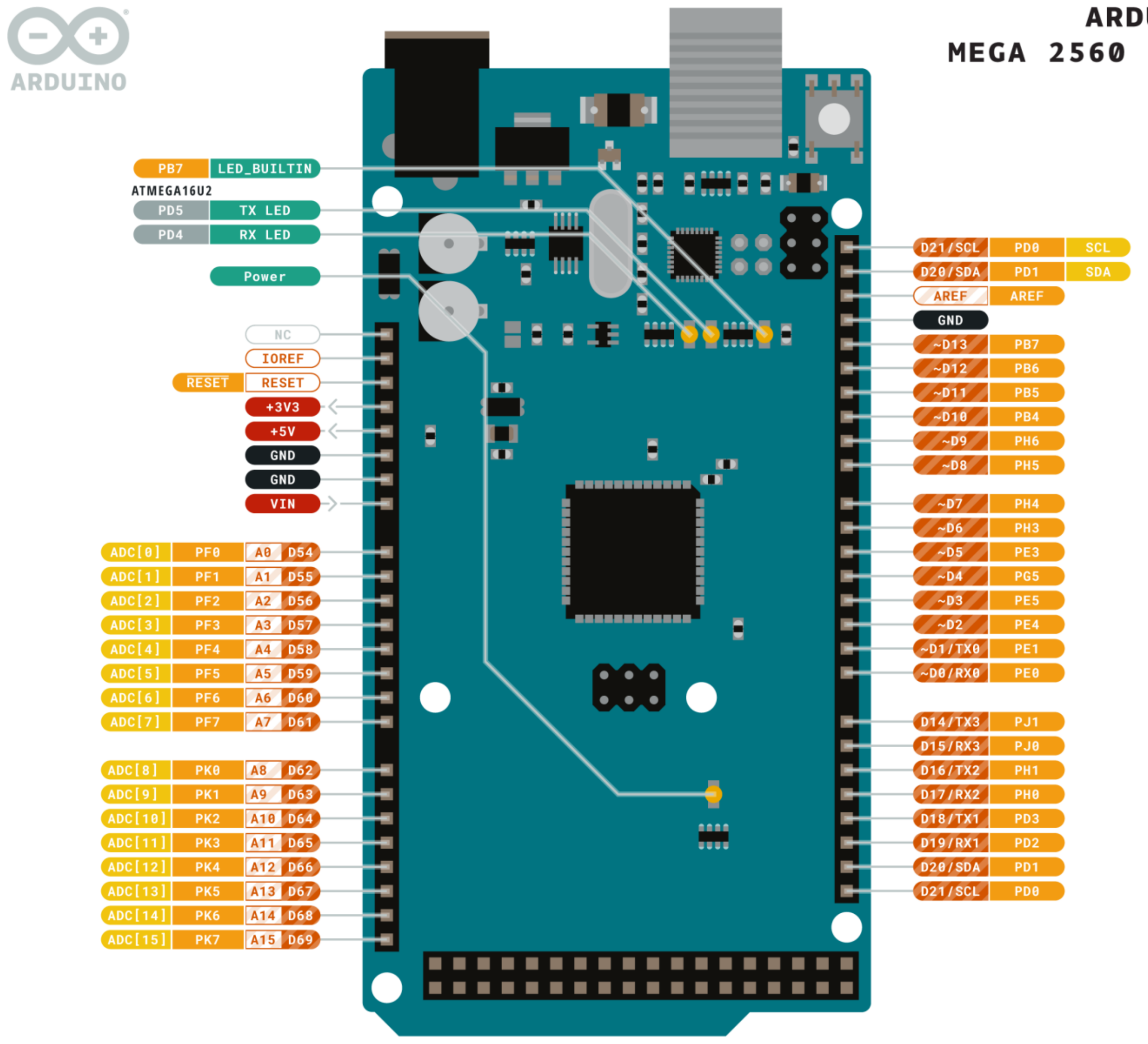
- The "/Input" node must publish into the previously defined topic "/set_point".
  - The node must publish an input signal to the motor (select between step, sinusoidal, etc.).
  - As before, It is forbidden to use any libraries, except from NumPy for this exercise.
- Make the necessary plots to analyse the system in rqt_plot



Half-term Challenge: Connection Diagrams



ARDUINO MEGA 2560 REV3



- ■ Ground
- 🟥 Power
- 🟩 LED
- ■ Internal Pin
- ■ SWD Pin
- □ Other Pin
- ▨ Digital Pin
- ▨ Analog Pin
- 🟨 Default
- 🟧 Microcontroller's Port

ARDUINO.CC

Half-term Challenge

Hints

Discrete PID controller:

$$u(k) = K_p e(k) + K_i T_s \sum_{n=0}^{k} e(n) + K_d \frac{e(k) - e(k-1)}{T_s}$$

where $u(k), e(k)$ are the controller output and error at time step k, such that time $t = kT_s$ where T_s is the sampling time. K_p, K_i, K_d 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:

$$\frac{t_{95}}{12} \leq t_{sampling} \leq \frac{t_{95}}{10},$$

where t_{95} is the settling time of the system.

Half-term Challenge

Hints (Arduino PWM)

- The Arduino “analogWrite(pin, value)” command generates a PWM signal, into a defined pin.
- pin: the Arduino pin to write to.
- value: the duty cycle: between 0 (always off) and 255 (always on).
- Allowed data types: int.
- Usually, for control purposes, the interval [0, 1] is mapped to the interval of the parameter “value” [0, 255].
- More information can be found here.

BOARD	PWM PINS	PWM FREQUENCY
Uno, Nano, Mini	3, 5, 6, 9, 10, 11	490 Hz (pins 5 and 6: 980 Hz)
Mega	2 - 13, 44 - 46	490 Hz (pins 4 and 13: 980 Hz)

```
int ledPin = 9;
void setup()
{
 pinMode(ledPin, OUTPUT);
}
void loop()
{
  analogWrite(ledPin,127);
}
```

Half-term Challenge

Hints (Arduino 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; we use volatile variables in this case.

- 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.

BOARD	INTERRUPT PINS	PWM FREQUENCY
Uno, Nano, Mini	2,3	
Mega	2, 3, 18, 19, 20, 21	(pins 20 & 21 are not available to use for interrupts while they are used for I2C communication)

Half-term Challenge

Hints (Arduino Logging)

- Like any ROS node, you can log messages at the appropriate verbosity level.
- More information about logging messages for debugging purposes or warnings, etc, can be found here.

Hints (Arduino Rosserial)

- Remember to remove all the Serial.begin, Serial.write, etc., before using rosserial, since ROS uses the serial port and cannot be used unless all communications with the Arduino serial monitor are removed from the code.

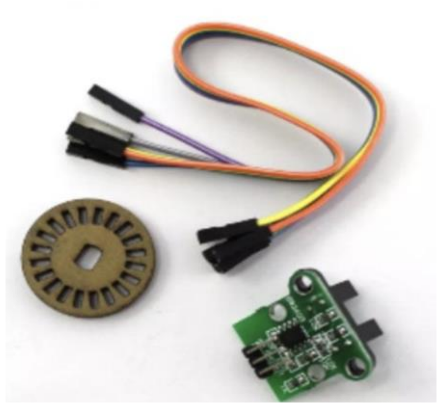
Hints (Hardcoding)

- Parametrise your code to make it more flexible. For example, in case you use another encoder, the number of pulses, time intervals, pins, etc. Make all of those as parameters, and try to no hardcode them, especially when dealing with large codes.
- In Arduino, the parameters can be defined in the same space as the variables at the top of the program.
- In ROS you can use parameter files.

Extra Information

Motors

- Due to the limited availability of motors and encoders; the students can search and bring their own motors and/or encoders (they don't have to be expensive).
- In Monterrey some electronic shops where you might find components are:
  - Lionchip
  - Steren
  - Techmake
  - MVElectronica
  - SEMTY
  - ElectronicaIoT
- The objective of this challenge is to control a motor using the concepts and tools provided by ROS such as rossserial to communicate with an MCU (Arduino) as an intermediary to control the motor.



Extra Information

Recommendations: Divide the project into different sections and establish the times and activities to be made in parallel or in series (Only to be used as an example):

Tasks/Steps	Step 1	Step 2	Step 3	Step 4	Step 5, 6, …	Goal
Motor Speed regulation	Analyse the DC Motor behaviour: • Connect the motor to the motor driver only and verify the correct connections. • Send Simple inputs (logical values) to verify if the motor is working properly.	Arduino-Motor Driver Communication: • Make a simple script to establish a simple communication between the Arduino and the Motor Driver. • Don't forget to debug!	Improve your motor speed regulation program: • Make the program more robust. • Make it more efficient. Connect your Arduino program with ROS. • Make a node for receiving data. • Write a simple subscriber in the Arduino that takes simple values to make the motor move. • (Blink activity example) • Don't forget to debug!	Improve the communication with ROS (if applies). Don't forget to debug!	Merge both working programs into a single node.	Control the speed of a motor using ROS.
Encoder reading	Analyse the Encoder behaviour: • Connect the motor encoder to an oscilloscope and observe the data being sent by the encoder. • Make sure the data is what you expect. • Don't forget to debug!	Arduino-Motor Encoder Communication: • Make a simple script to establish a simple communication between the Arduino and the Motor encoder (Activities/pushbutton example). • Make sure the Arduino detects the pulses generated by the encoder. • Don't forget to debug!	Arduino-Motor Encoder Communication: • Make a simple script to count the pulses generated by the encoder. • Establish a time for counting such pulses and determine the speed of the motor. • Don't forget to debug!	Make a simple publisher to communicate with ROS. (sonar activity example). • Make a node for publishing data. • Estimate and publish the speed of the motor. • Make your program more robust		
Control Desing	…	…	…	…	…	
Set Point design	…	…	…	…	…	

Rules

- This is a challenge, **not** a class. The students are encouraged to research, improve tune, and explain their algorithms by themselves.
- MCR2(Manchester Robotics) Reserves the right to answer a question if it is determined that the question contains partially or an answer.
- The students are welcome to ask only about the theoretical aspect of the class.
- 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 libraries except for standard libraries or NumPy.
- If in doubt about libraries 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 of the improvements.
- All the students must respect 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.