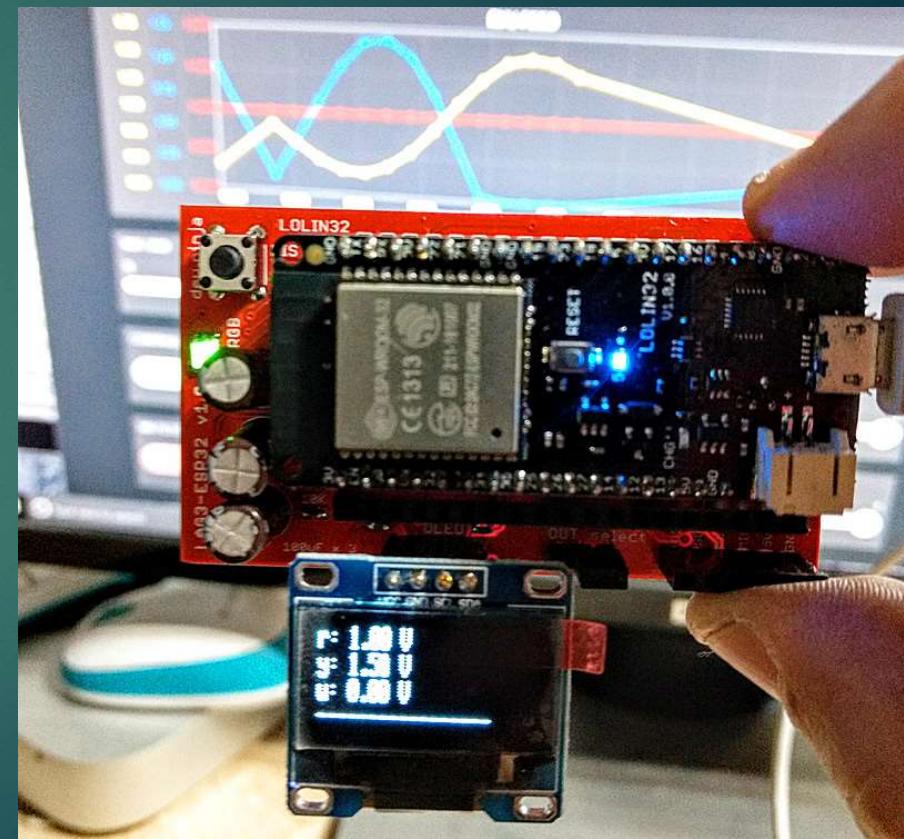


# Workshop 1 : ESP32 Basics

Dr.Varodom Toochinda  
Dept. of Mechanical Engineering  
Kasetsart University

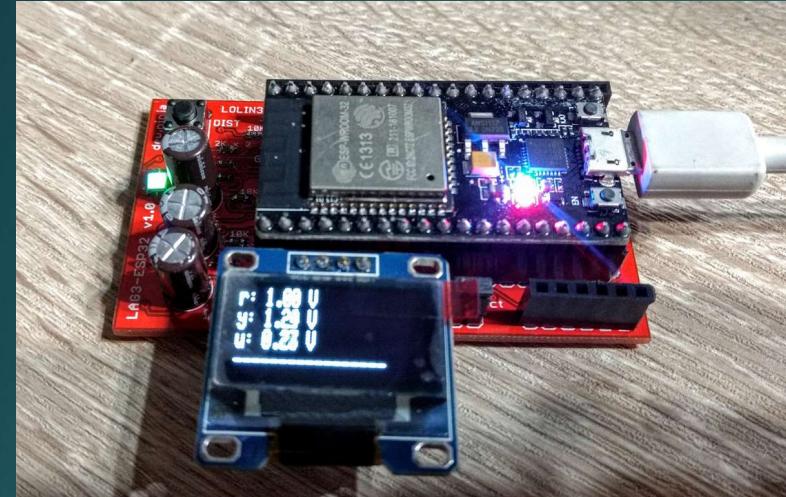


# Outline

- ▶ ESP32 Arduino core and libraries installation
- ▶ Learn to program ESP32 using Arduino IDE and C language for ESP32 main components such as ADC, PWM
- ▶ Learn basics of external interrupt and timer
- ▶ Learn how to construct a command interpreter function
- ▶ Plotting experimental data
- ▶ Connect ESP32 to NETPIE 2020

# Hardware and Software

- ▶ Hardware
  - ▶ NodeMCU-32S
  - ▶ LAG3-ESP32 by dew.ninja
- ▶ Software
  - ▶ Arduino IDE ver 1.8.13 + ESP32 core + libraries
  - ▶ CP2104 USB to serial driver
- ▶ Cloud platform
  - ▶ NETPIE 2020
- ▶ Notebook computer
  - ▶ Windows preferred. Setup might be problematic with some version of MAC-OSX



# Hardware/Software

## SETUP

lab1\_4 | Arduino

dew.ninja

Boards Manager...

Arduino AVR Boards >

ESP32 Arduino >

ESP8266 Boards (2.7.4) >

AS

Python arrays

```
// use GPIO16 as output
// use A3 as pin for analog input
#ILTIN; // square wave input
```

3 pin assignments

```
BLED=17; // BLED = 17
0; // values for pins
```

\*\*\*\*\*

NodeMCU-32S

MH ET LIVE ESP32DevKIT

MH ET LIVE ESP32MiniKit

ESP32vn IoT Uno

LED

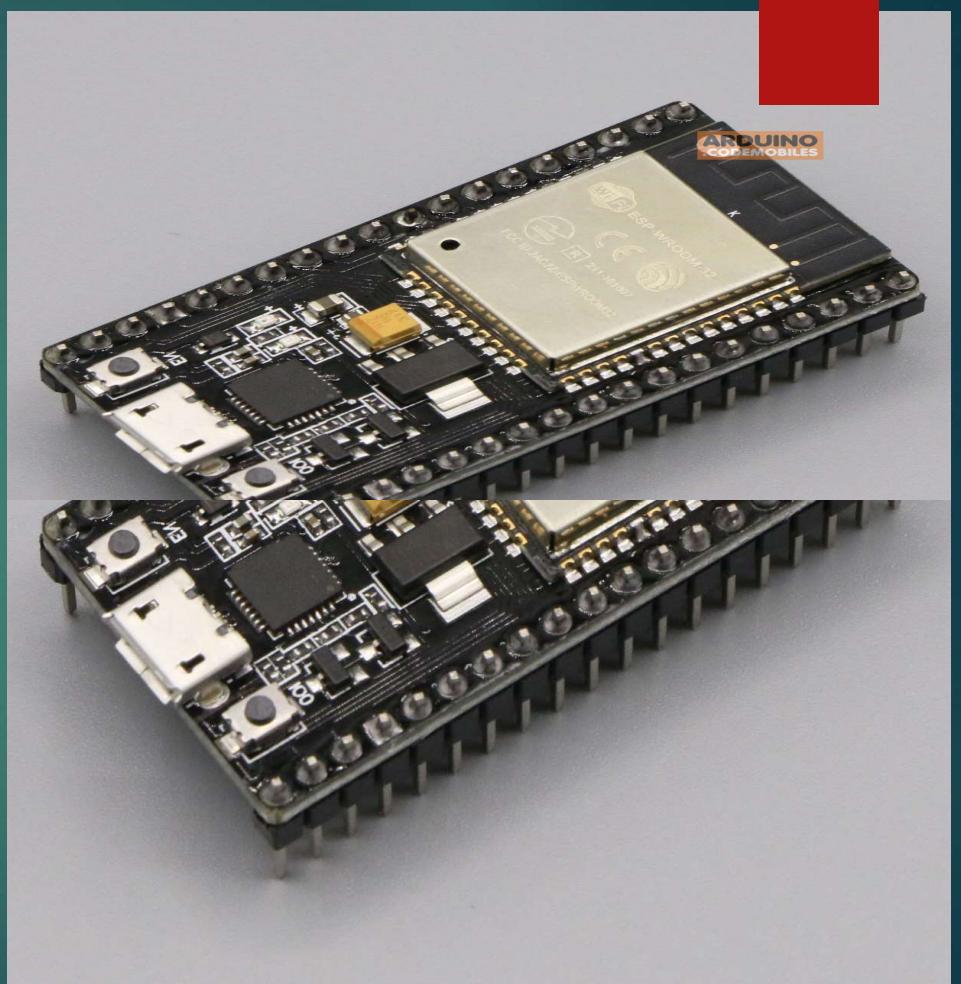
\*\*\*\*\*

B led

\*\*\*\*\*

# NodeMCU-32S

- ▶ ESP32 SoC with integrated WiFi/Bluetooth (dual-core Tensilica Xtensa LX6 microprocessor)
- ▶ Clock speed 240 MHz
- ▶ 4 MB Flash
- ▶ Digital I/O pins : 26
- ▶ Analog input pins: 12
- ▶ 8-bit analog outputs (DAC) : 2
- ▶ SPI, I2C, UART and sensors (see datasheet)

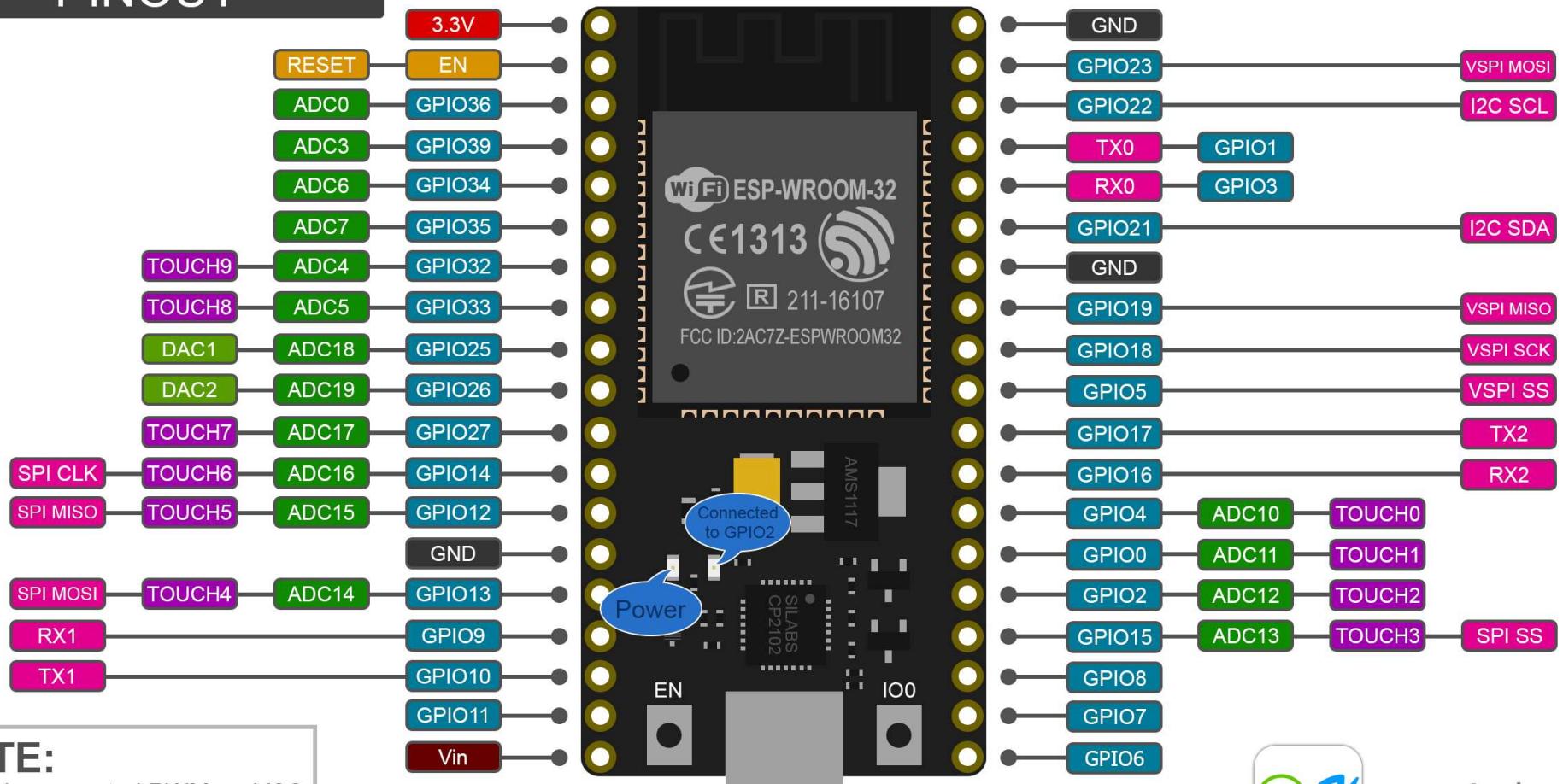




CC BY 4.0

# NodeMCU-32S

## PINOUT



### NOTE:

All pin supported PWM and I2C  
Pin current 6mA (Max. 12mA)



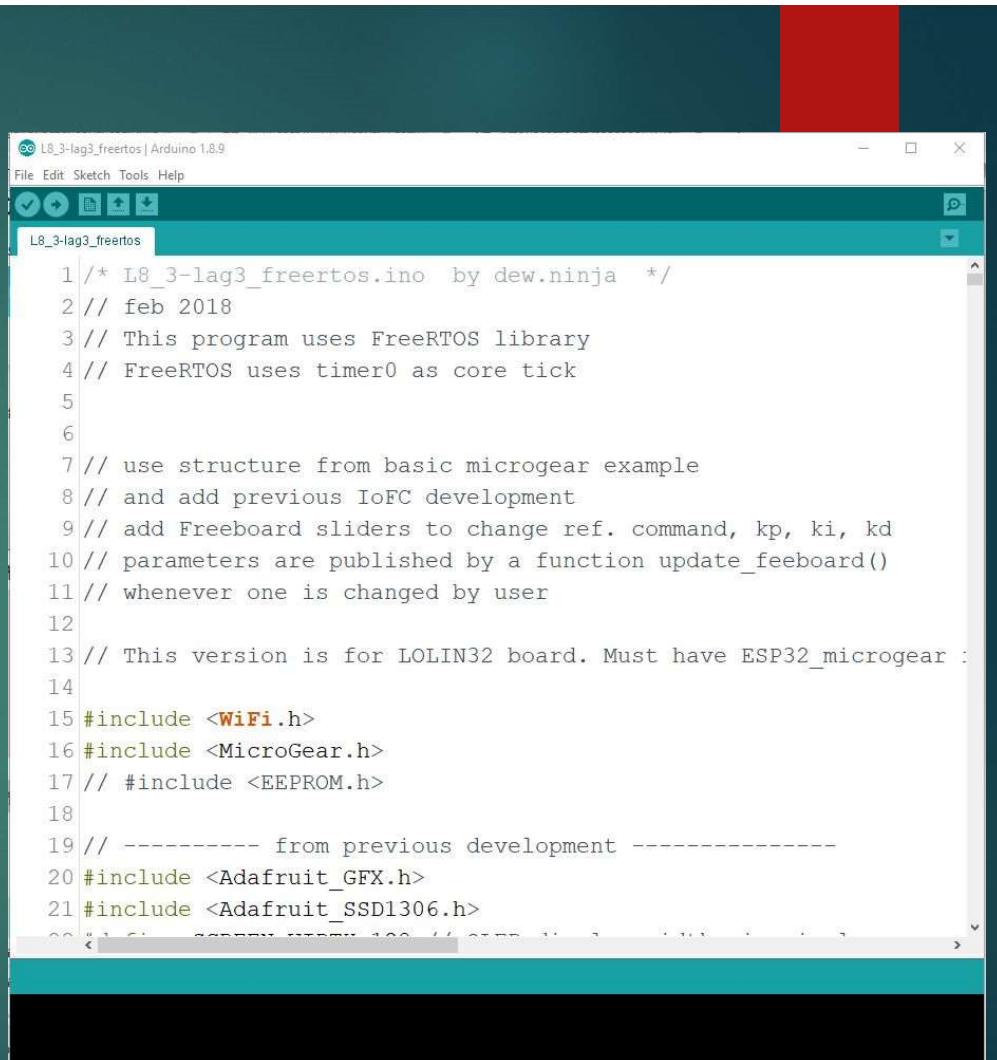
[www.ioxhop.com](http://www.ioxhop.com)

# Use NodeMCU-32S with Arduino IDE

- ▶ Install Arduino IDE <https://www.arduino.cc/en/Main/Software>
- ▶ Install CP210x USB-to-serial driver
- ▶ Install ESP32 Arduino Core

# Install Arduino IDE

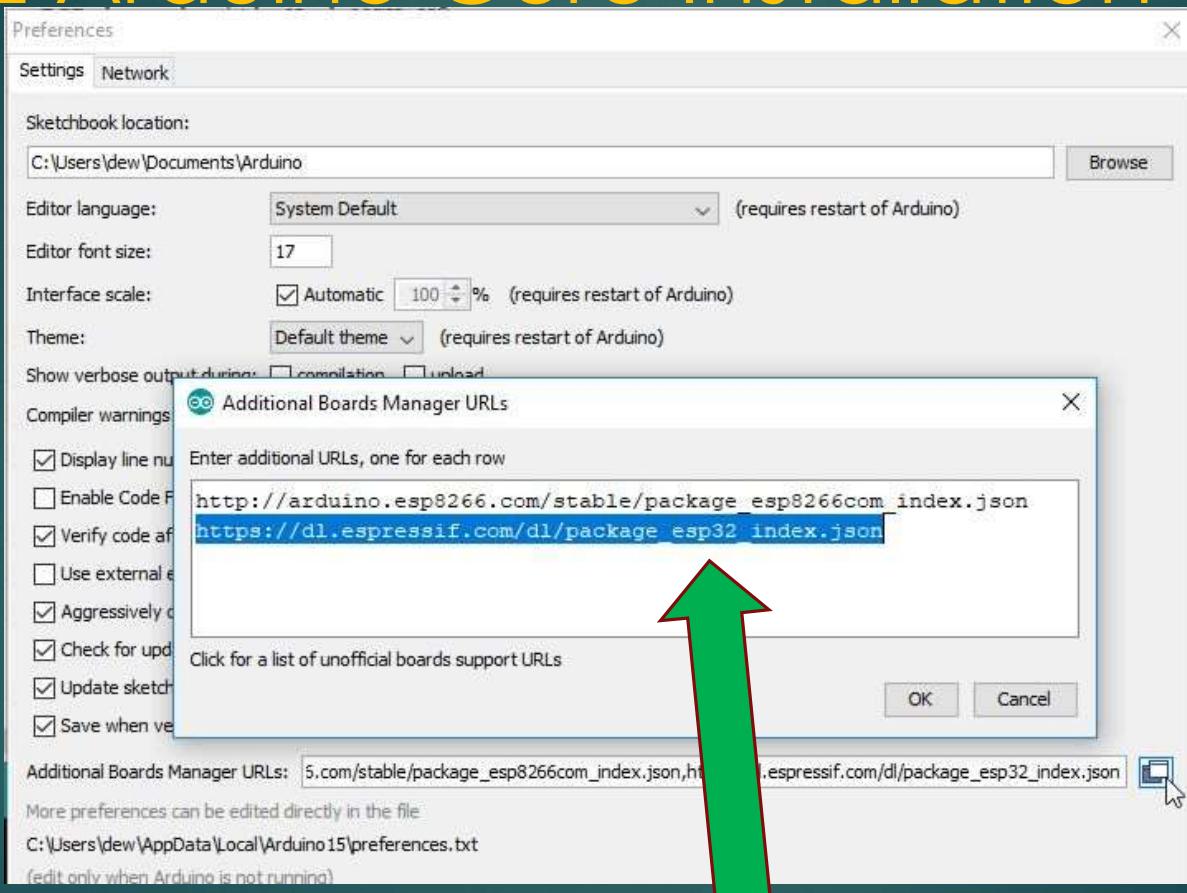
- ▶ Latest version (March 21) 1.8.13
- ▶ Download and run installer (or download zip file and extract to the desired directory)
- ▶ Install CP210x USB driver
- ▶ Check Tools->Board . There is no support for ESP32. We need to install Arduino Core for ESP32



The screenshot shows the Arduino IDE interface with a sketch named "L8\_3-lag3\_freertos" open. The code editor displays the following C++ code:

```
1 /* L8_3-lag3_freertos.ino by dew.ninja */
2 // feb 2018
3 // This program uses FreeRTOS library
4 // FreeRTOS uses timer0 as core tick
5
6
7 // use structure from basic microgear example
8 // and add previous IoFC development
9 // add Freeboard sliders to change ref. command, kp, ki, kd
10 // parameters are published by a function update_freeboard()
11 // whenever one is changed by user
12
13 // This version is for LOLIN32 board. Must have ESP32_microgear :
14
15 #include <WiFi.h>
16 #include <MicroGear.h>
17 // #include <EEPROM.h>
18
19 // ----- from previous development -----
20 #include <Adafruit_GFX.h>
21 #include <Adafruit_SSD1306.h>
22 #include <FreeRTOS.h>
23 #include <task.h>
24 #include <queue.h>
25 #include <math.h>
```

# ESP32 Arduino Core Installation



[https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json)

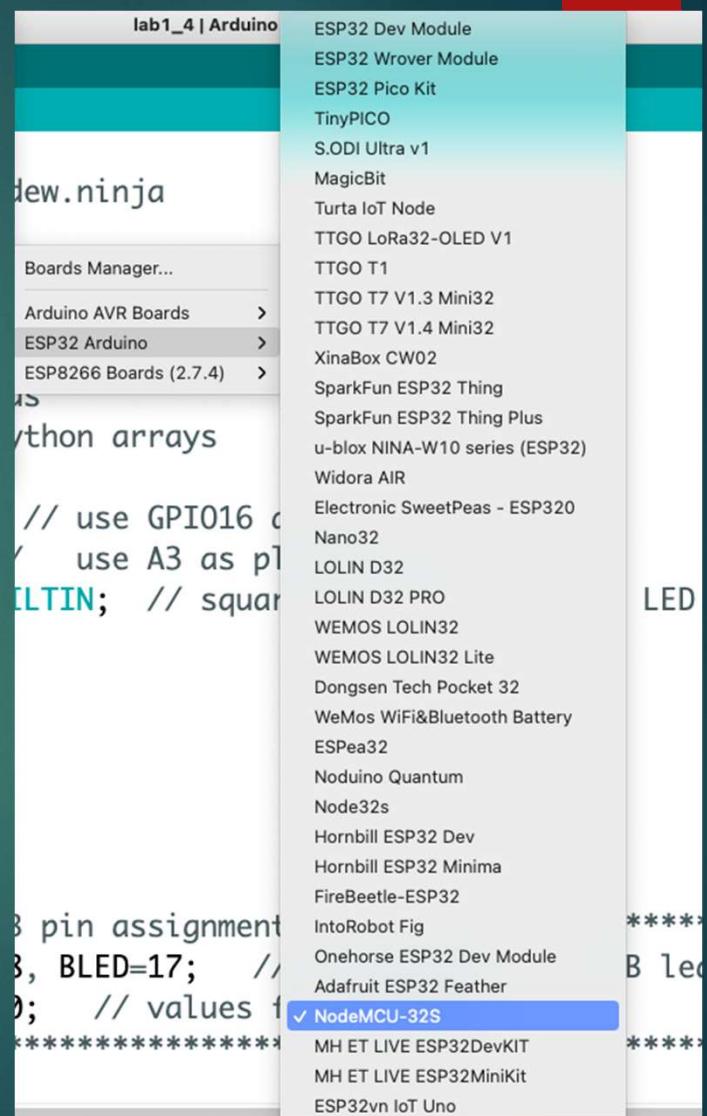
# ESP32 Arduino Core Installation



✓ ติดตั้งสำเร็จ

# After ESP32 Arduino Core is installed

Select Tools -> Board ->ESP32 Arduino -> NodeMCU-32S

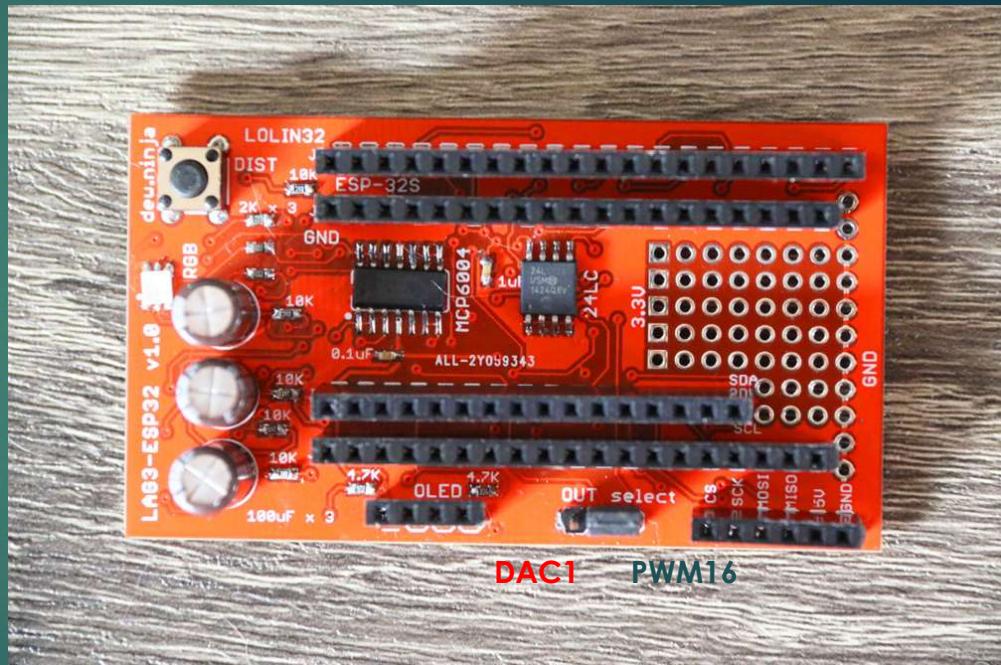


# Install Library (use library manager)

- Pubsubclient for NETPIE 2020

# LAG3-ESP32 board

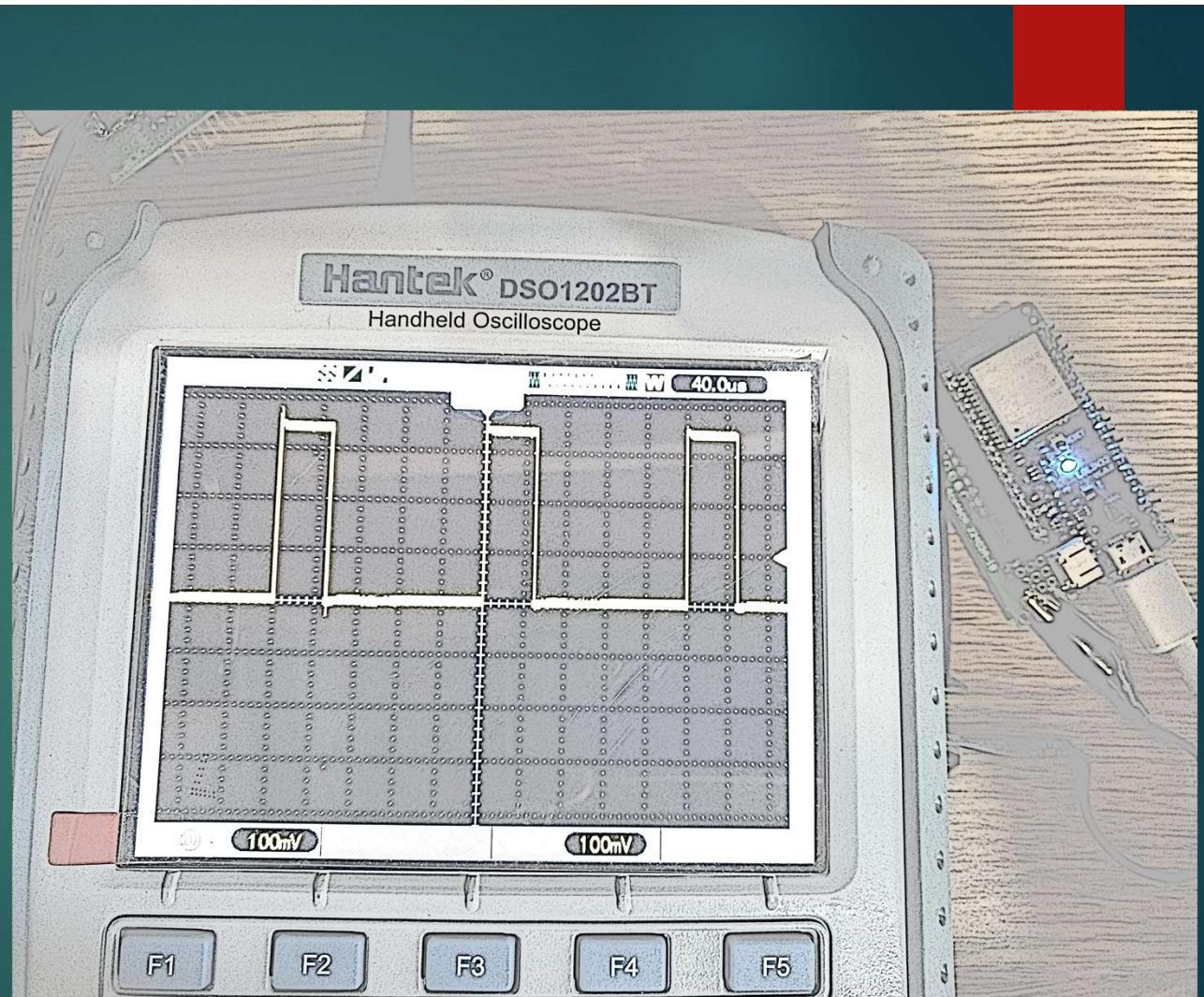
- ▶ 3-level cascade tank using RC circuit and op-amp
- ▶ Connector for OLED display
- ▶ RGB LED and EEPROM (optional)
- ▶ Can be used with WEMOS LOLIN32 or NODEMCU-32S (commercial ESP32 modules)
- ▶ DAC1/PWM16 outputs (selectable via jumper)



# NodeMCU-32S pins used

<b>GPIO</b>	<b>Name</b>	<b>Type</b>	<b>Function</b>
39	ADC3	Analog IN	Plant output (Y)
32	ADC4	Analog IN	State variable (X2)
33	ADC5	Analog IN	State variable (X1)
25	DAC1	Analog OUT	Controller output (Analog)
16	PWM	OUT	Controller output (PWM)
22	SCL	OUT	I2C
21	SDA	OUT	I2C
19	PWMR	OUT	PWM for red LED
18	PWMG	OUT	PWM for green LED
17	PWMB	OUT	PWM for blue LED
5	LED	Digital OUT	On-board LED

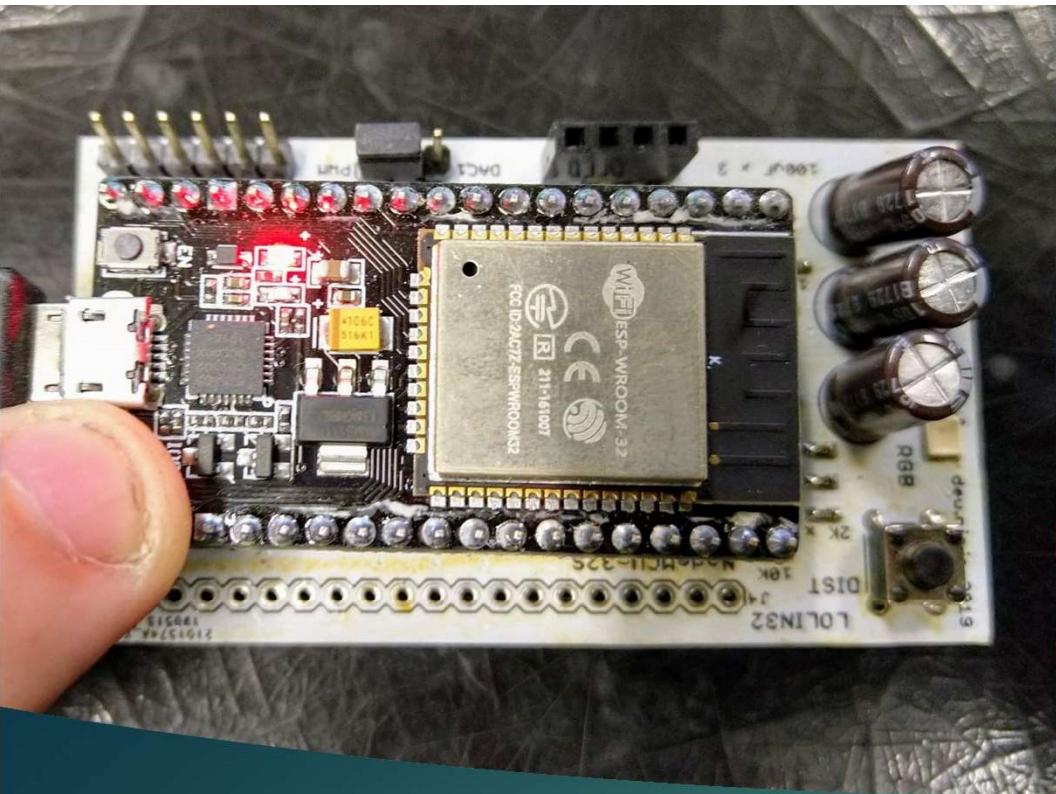
# ESP32 basic module programming



# Arduino program structure

```
setup() {  
    // Module initialization.  
    // run only once at  
    // beginning  
}  
  
loop() {  
    // run iteratively  
    user_func();  
}  
  
user_func() { // optional  
}
```

```
external_interrupt_function() {  
    // respond to external  
    //interrupt  
    //optional  
}  
  
timer_callback() {  
    //respond to timer interrupt  
    // optional  
}
```



Press button to upload code

```
12     r = parmvalfloat;
13     rold = r;    // save previous com
14   }
15   Serial.println("datamat = [");
16   datacnt = 1.    // set the flag +
```

loading...

```
etch uses 229844 bytes (17%) of program st
obal variables use 13580 bytes (4%) of dyn
ptool.py v3.0-dev
rial port /dev/cu.usbserial-0001
nnecting.....
```

# Serial port communication

- ▶ Initialized by `Serial.begin()`, such as
  - ▶ `Serial.begin(115200);`
  - ▶ Other side must use same baudrate.
- ▶ Output string with `Serial.print()` or `Serial.println()`
- ▶ Read data with `Serial.readString()`
- ▶ Check data in buffer with `Serial.available()`
- ▶ Can use Serial Monitor/ Serial Plotter from Tools menu of Arduino IDE to monitor or send commands

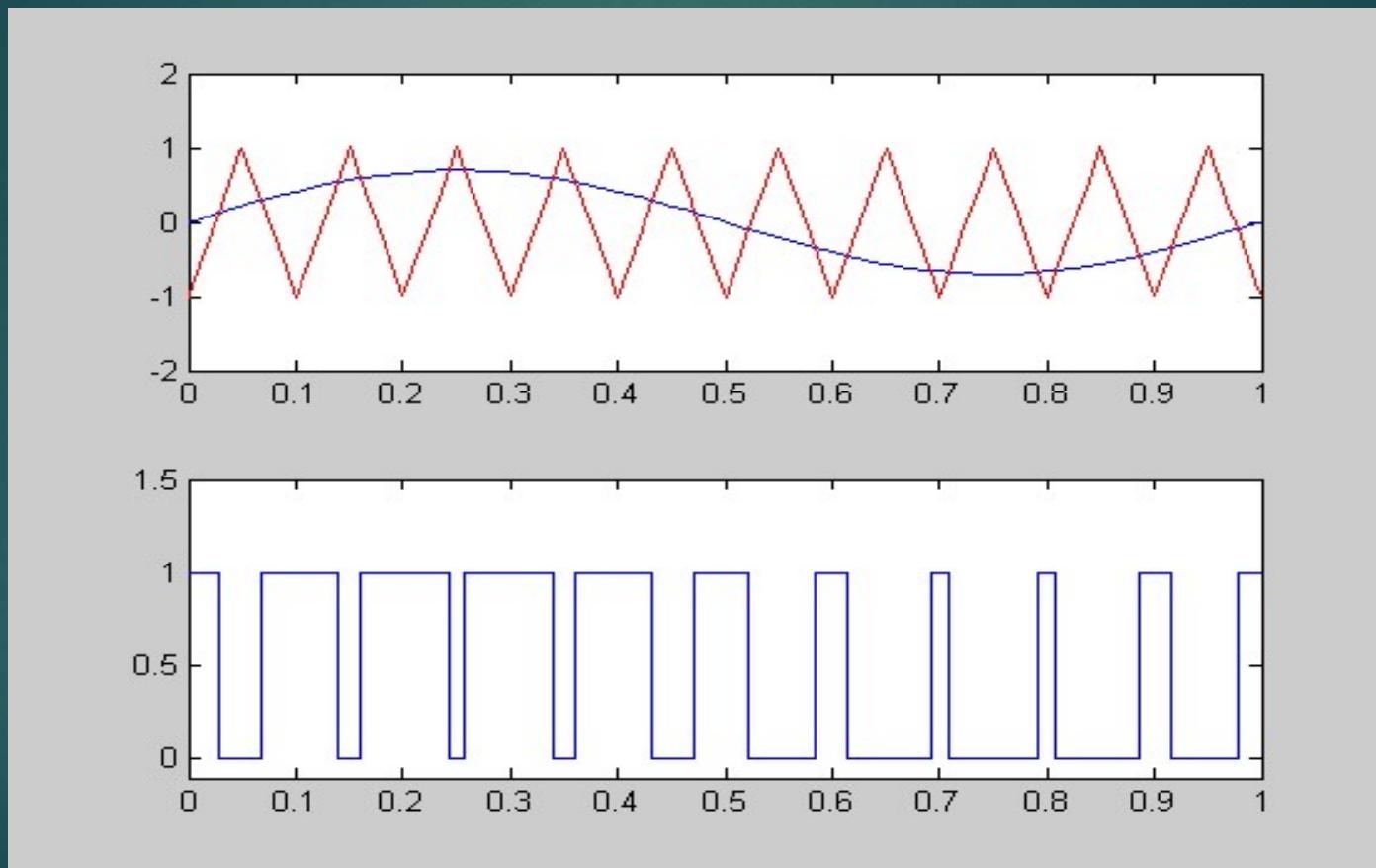
# Analog inputs

- ▶ Maximum 16 channels
- ▶ Maximum voltage equals Vcc (3.3 V)
- ▶ 12-bit resolution (0 – 4095)
- ▶ Read data using `analogRead();`
- ▶ Can specify GPIO number or analog channel
  - ▶ `int adcval = analogRead(A0);`
  - ▶ `int adcval = analogRead(36);`

# Analog outputs

- ▶ 8-bit DAC modules
- ▶ 2 channels : GPIO25 (DAC1) and GPIO26 (DAC2)
- ▶ Write function : `dacWrite()`;
- ▶ Can specify GPIO or DAC channels. For example, to send 1.65 volts out at pin DAC1
  - ▶ `dacWrite(25, 127);`
  - ▶ `dacWrite(DAC1, 127);`
- ▶ Limited output current. In practice, an op-amp buffer may be needed to supply more current.

# PWM generation



# PWM outputs



- ▶ Native ESP32 core does not have `analogWrite()` function (additional library is needed)
- ▶ Use `ledc()` , which must be setup with `ledcSetup()` ; ex.,
  - ▶ `ledcSetup(1, 5000, 12);` use timer 1 PWM frequency 5 KHz and 12-bit resolution
- ▶ Attach timer to GPIO using `ledcAttachPin(timer, pin)`
  - ▶ `ledcAttachPin(2,1);` attach timer 1 to GPIO2
- ▶ After setup, use `ledcWrite()` to write to PWM module (the first argument is timer number, not GPIO pin!)
  - ▶ `ledcWrite(1, 1023);` generate PWM with duty cycle 25%

# Using analogWrite library

- ▶ Must install ESP32 analogWrite library
- ▶ `#include <analogWrite.h>`
- ▶ Set resolution in `setup()`
  - ▶ `analogWriteResolution(LED_BUILTIN, 12);`
- ▶ Output some value
  - ▶ `analogWrite(LED_BUILTIN, brightness);`

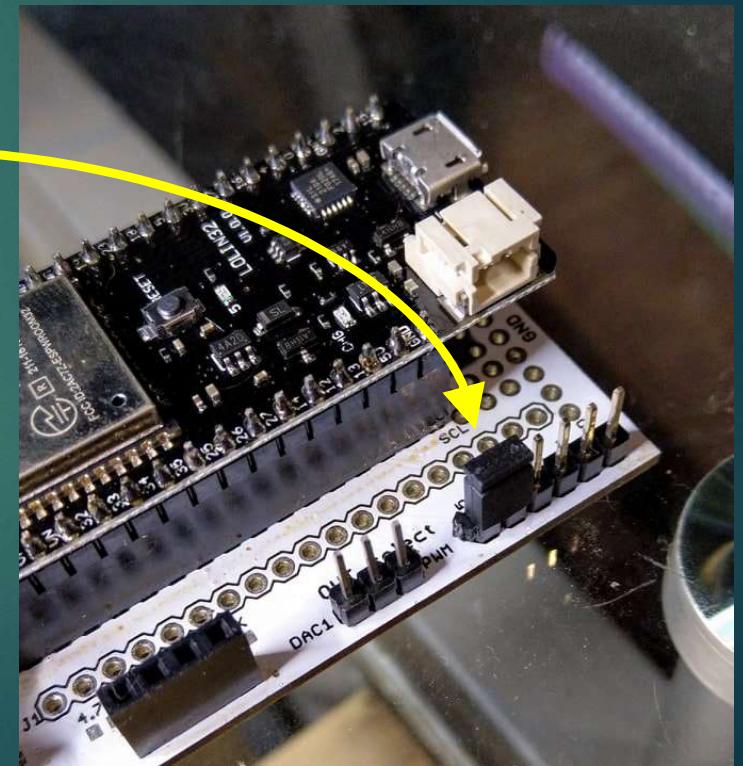
From my experiment on pin 15, the PWM waveform does not give expected result ?

# External interrupt

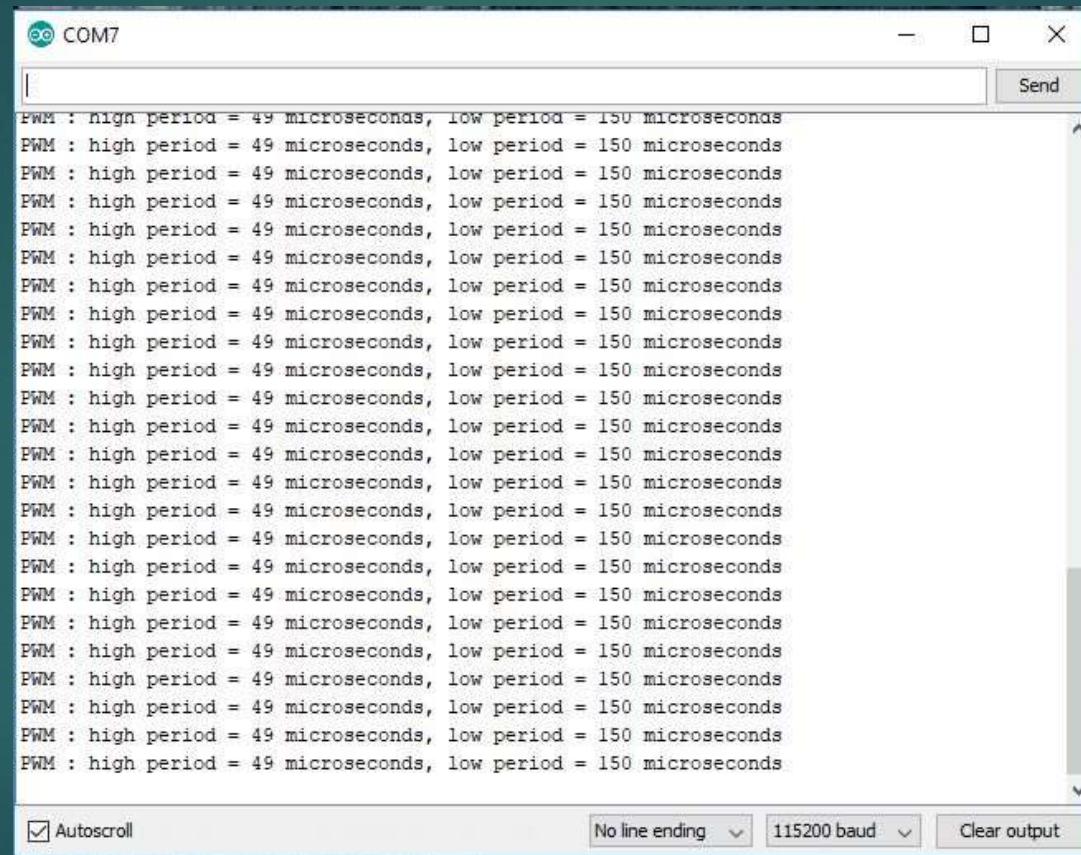
- ▶ Use `attachInterrupt()`
- ▶ `attachInterrupt(34, funcA, RISING);` setup GPIO34 such that when the logic level changes from 0 to 1 (rising edge), execute `funcA()`
- ▶ Choice of external interrupt events
  - ▶ **LOW** – when input has logic 0
  - ▶ **HIGH** – when input has logic 1
  - ▶ **CHANGE** – when input changes logic level
  - ▶ **RISING** – at rising edge of input
  - ▶ **FALLING** – at falling edge of input

# Lab 1.1 : measure PWM periods without using an oscilloscope

- ▶ Assign PWM output to GPIO15
  - ▶ Use timer 1 PWM frequency 5 KHz
  - ▶ Connect GPIO15 to GPIO14
  - ▶ Setup GPIO14 to interrupt every logic change
  - ▶ Create PWM with duty cycle 25%
  - ▶ Use micros() to measure high and low periods



# Lab 1.1 : Serial Monitor Output



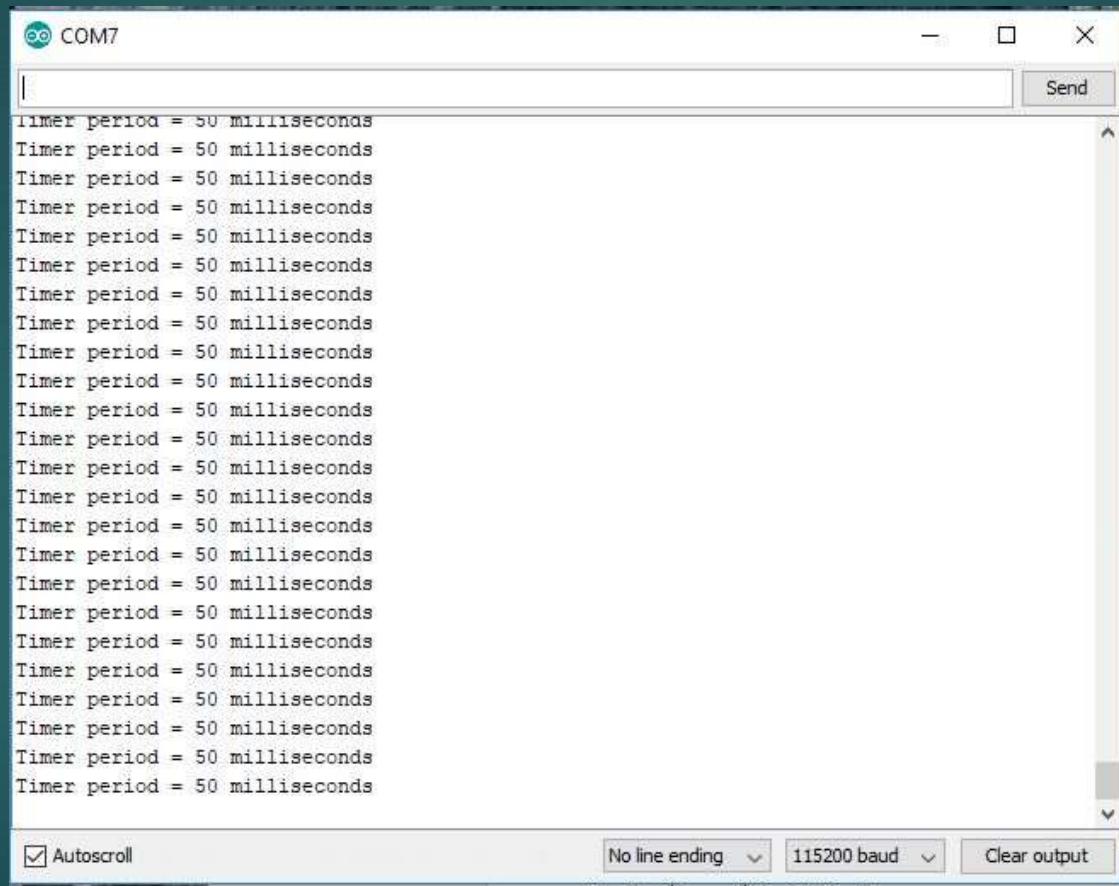
# Using timer on ESP32

- ▶ Use hardware timer (can cause watchdog reset for long interrupt period)
- ▶ Setup using `timerBegin()`
- ▶ Write a callback and attach to timer using `timerAttachInterrupt()`
- ▶ Set interrupt period with `timerAlarmWrite()`
- ▶ Enable interrupt with `timerAlarmEnabled()`

# Lab 1.2 : using a timer on ESP32

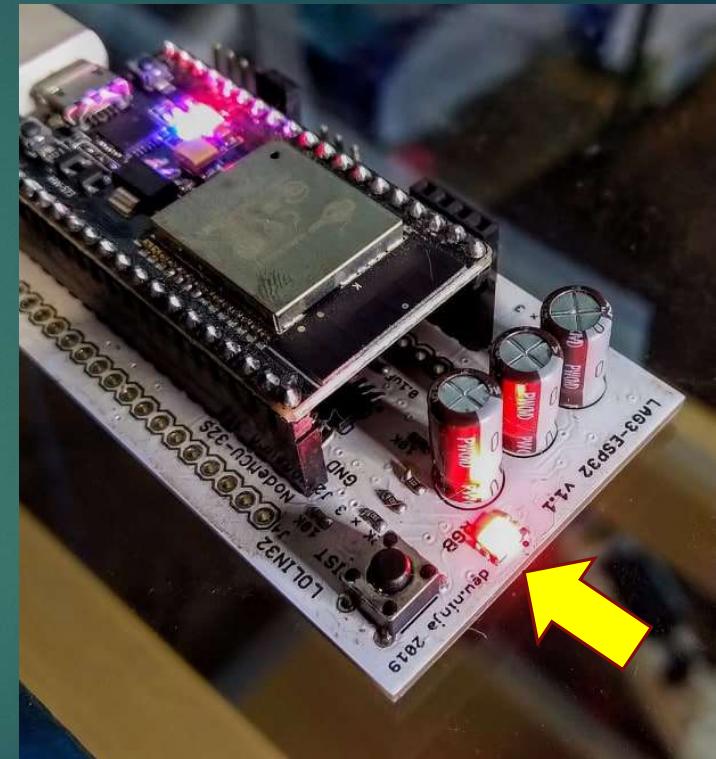
- ▶ Setup timer 0 to interrupt every 50 milliseconds
- ▶ Blink onboard LED on NodeMCU-32S
- ▶ Measure period with millis()
- ▶ Output result on Serial Monitor

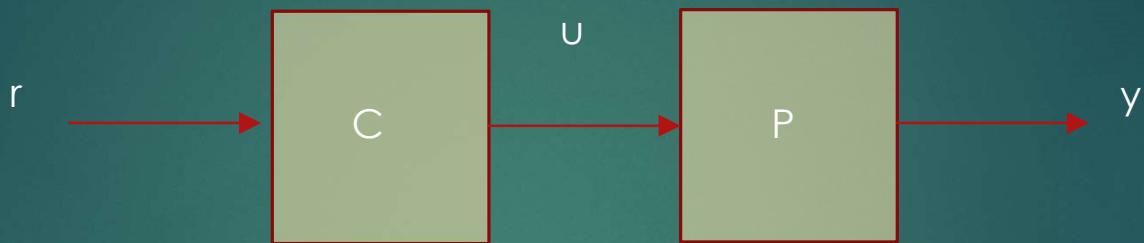
# Lab 1.2 Serial Monitor Output



## Exercise 1:

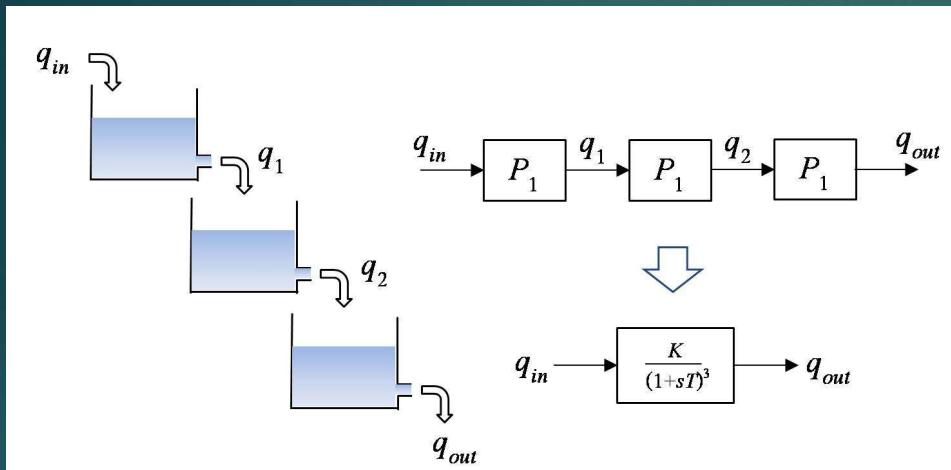
modify `LAB1_2.ino` to  
blink red LED  
independently from  
on-board LED





OPEN-LOOP SYSTEM

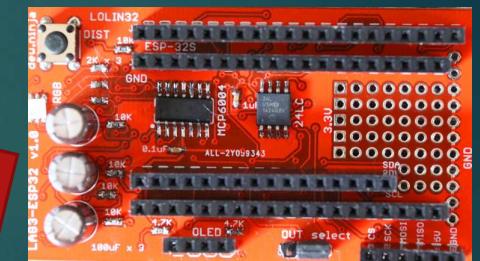
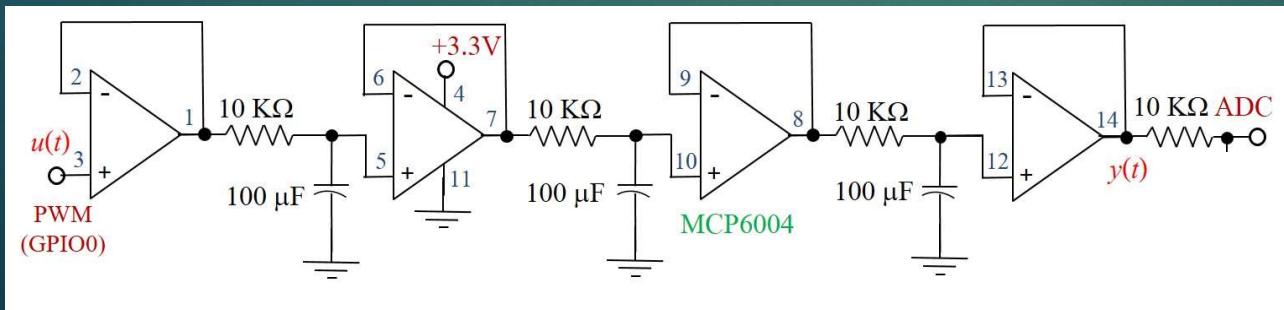
# 3 cascade tank model(LAG3)



normalized  
transfer function

$$\frac{1}{(s + 1)^3}$$

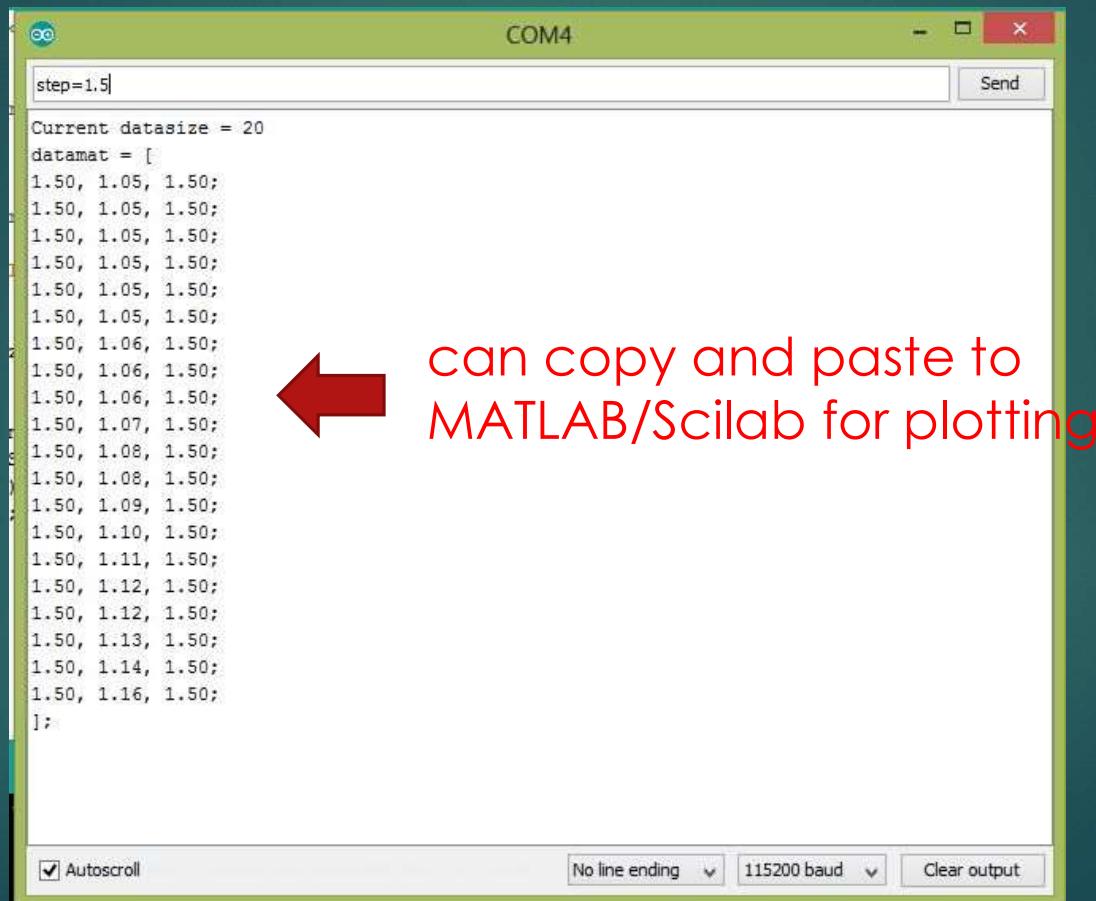
Simulation by electronic circuit



## Lab 1.4 Command interpreter function

- ▶ Construct cmdInt() for the following commands
  - ▶ step=<value> : adjust reference input
  - ▶ datasize=<num> : specify number of data points
  - ▶ Other commands you may want to add, using format command=<parameter>

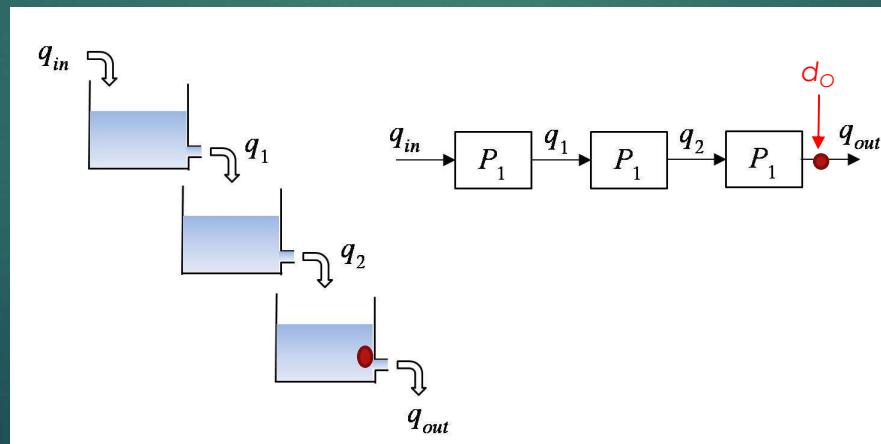
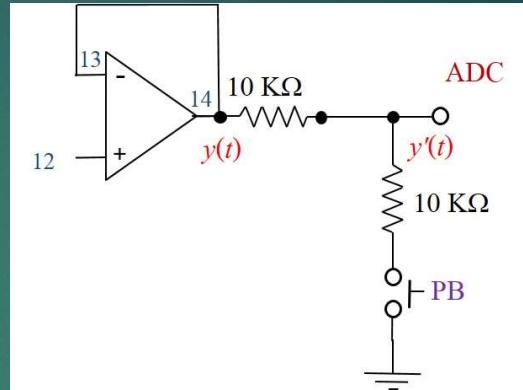
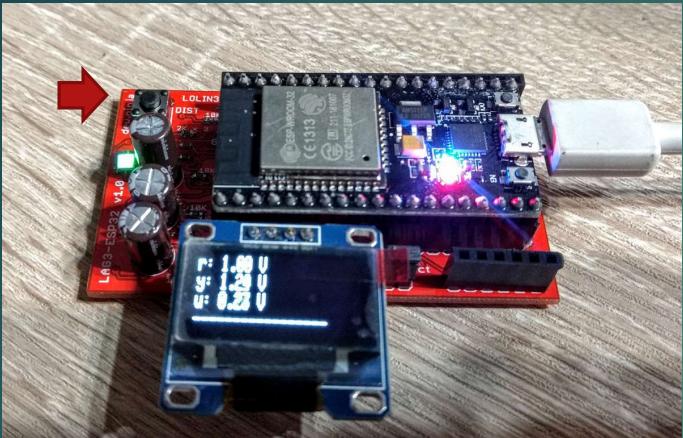
# Test command via serial monitor



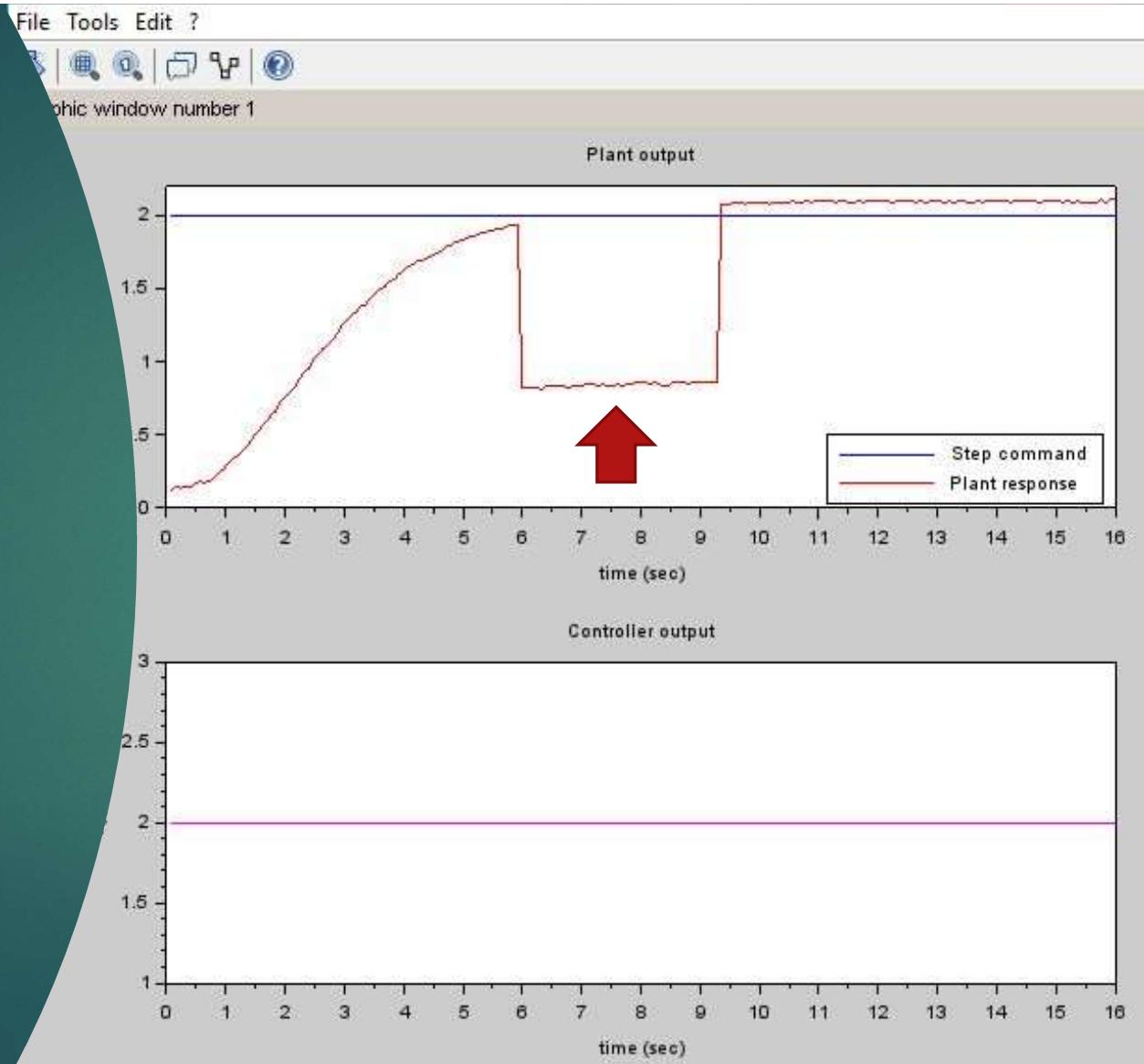
```
step=1.5|
Current datasize = 20
datamat =
1.50, 1.05, 1.50;
1.50, 1.05, 1.50;
1.50, 1.05, 1.50;
1.50, 1.05, 1.50;
1.50, 1.05, 1.50;
1.50, 1.05, 1.50;
1.50, 1.05, 1.50;
1.50, 1.06, 1.50;
1.50, 1.06, 1.50;
1.50, 1.06, 1.50;
1.50, 1.07, 1.50;
1.50, 1.08, 1.50;
1.50, 1.08, 1.50;
1.50, 1.09, 1.50;
1.50, 1.10, 1.50;
1.50, 1.11, 1.50;
1.50, 1.12, 1.50;
1.50, 1.12, 1.50;
1.50, 1.13, 1.50;
1.50, 1.14, 1.50;
1.50, 1.16, 1.50;
];
```

can copy and paste to  
MATLAB/Scilab for plotting

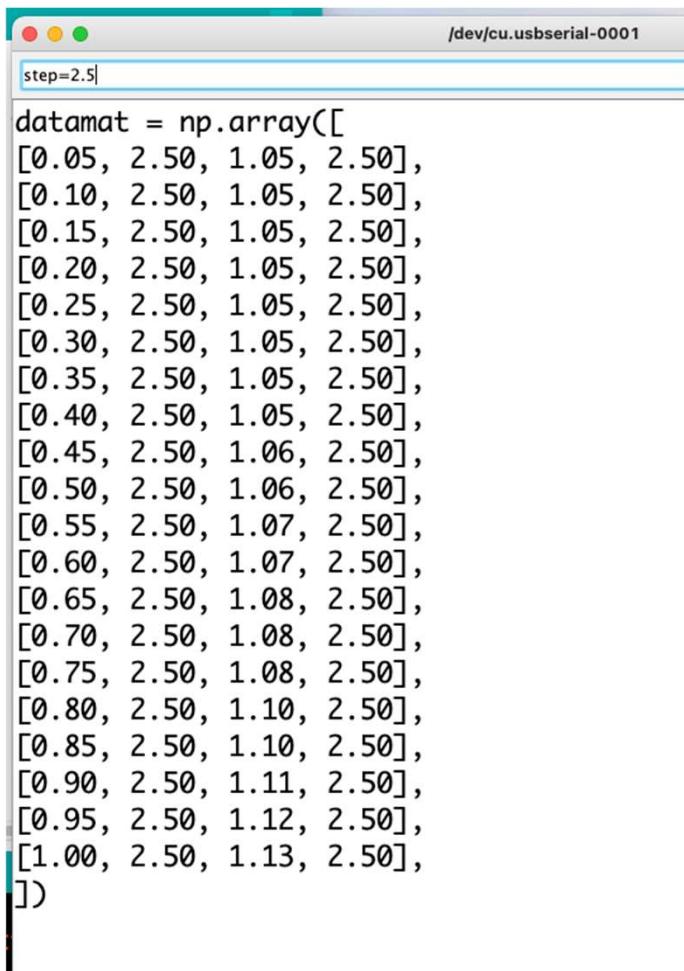
# Output disturbance switch



When  
disturbance  
switch is  
pressed

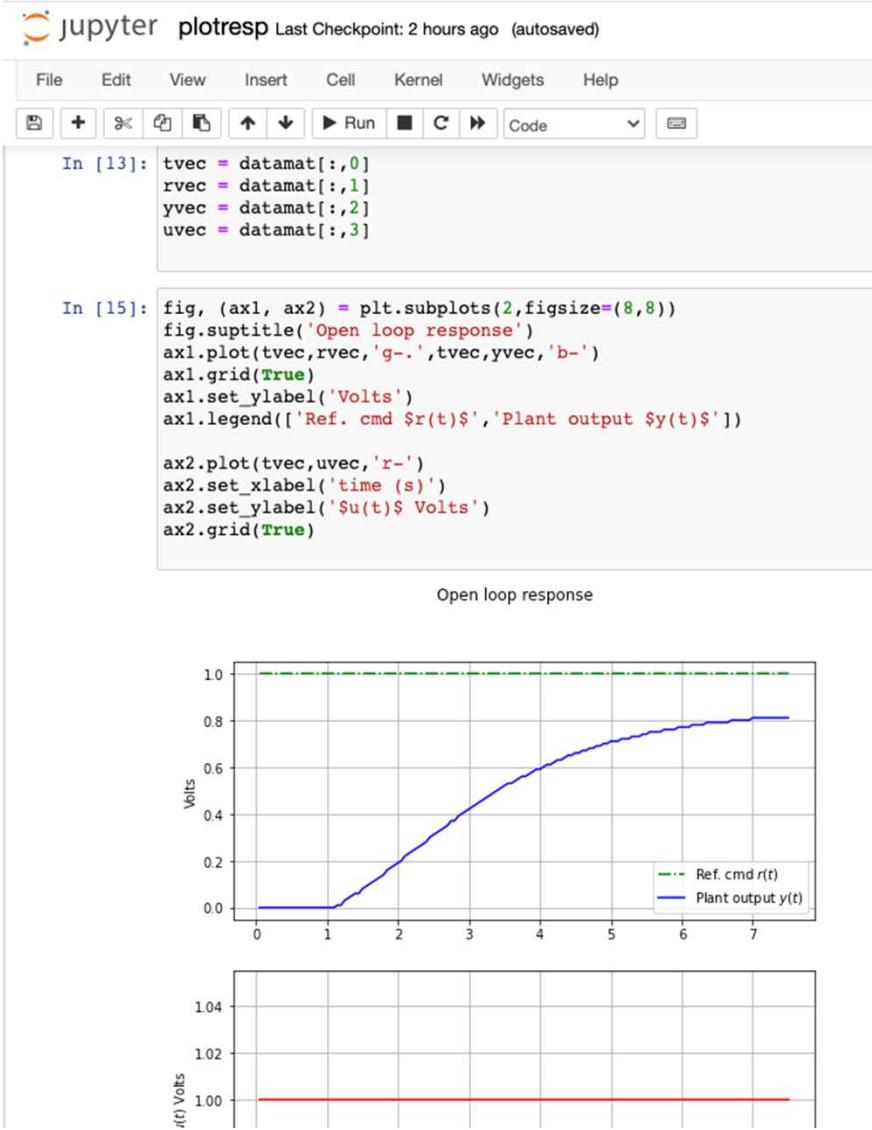


# Output as numpy array



A screenshot of a terminal window titled "/dev/cu.usbserial-0001". The window contains the following text:

```
step=2.5
datamat = np.array([
[0.05, 2.50, 1.05, 2.50],
[0.10, 2.50, 1.05, 2.50],
[0.15, 2.50, 1.05, 2.50],
[0.20, 2.50, 1.05, 2.50],
[0.25, 2.50, 1.05, 2.50],
[0.30, 2.50, 1.05, 2.50],
[0.35, 2.50, 1.05, 2.50],
[0.40, 2.50, 1.05, 2.50],
[0.45, 2.50, 1.06, 2.50],
[0.50, 2.50, 1.06, 2.50],
[0.55, 2.50, 1.07, 2.50],
[0.60, 2.50, 1.07, 2.50],
[0.65, 2.50, 1.08, 2.50],
[0.70, 2.50, 1.08, 2.50],
[0.75, 2.50, 1.08, 2.50],
[0.80, 2.50, 1.10, 2.50],
[0.85, 2.50, 1.10, 2.50],
[0.90, 2.50, 1.11, 2.50],
[0.95, 2.50, 1.12, 2.50],
[1.00, 2.50, 1.13, 2.50],
])
```



Plot on  
Jupyter  
Notebook

# What we have learnt from Lab 1

- ▶ Generate PWM output
- ▶ How to use external interrupt and timer
- ▶ Serial communication and command interpreter
- ▶ Drawback of open-loop control

[Home](#) ▾[Ecosystem](#) ▾[Developer](#) ▾[LOGIN](#)[Sign up](#)

## Connect Everything

NETPIE is an IoT cloud-based platform-as-a-service that helps connect your IoT devices together seamlessly by pushing the complexity from the hands of application developers or device manufacturers to the cloud

[GET STARTED](#) [WATCH VIDEO](#)

# NETPIE 2020

From Makers Nation  
Toward Smart Nation

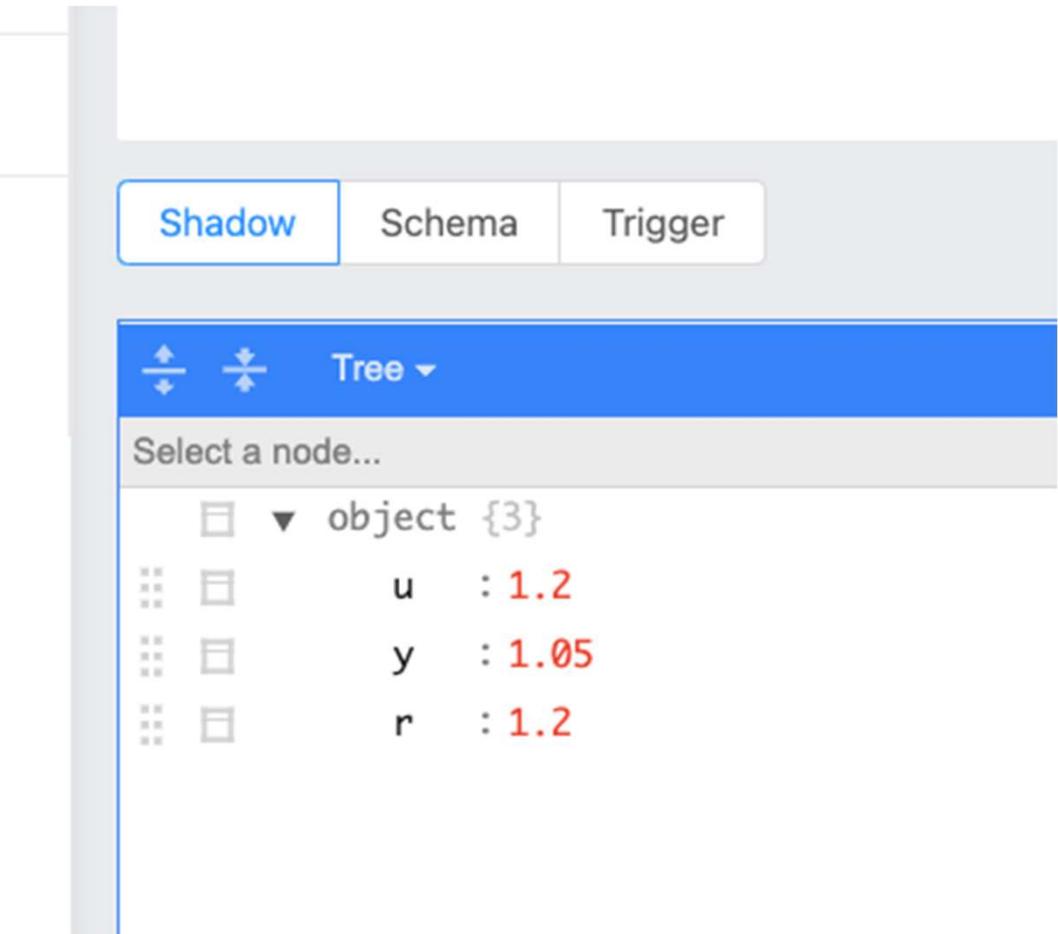
## Monitoring and control via NETPIE 2020



- ▶ Cloud platform developed by NECTEC
- ▶ Can use with various hardware such as ESP8266, ESP32, Raspberry PI, Android
- ▶ Latest version is NETPIE 2020
- ▶ Details can be read from <https://netpie.io>

# Lab1\_4\_NETPIE.ino

- Add WiFi and internet connection code to Lab 1.4
- On NETPIE account, create a project and device
- Copy Client-ID, Username, Token information to Lab1\_4\_NETPIE.ino
- Publish r,y,u data to NETPIE shadow
- Create Freeboard and datasource
- Add 3 gauge widgets for r, y, u data
- Add a slider for ref. command adjustment



r,y,u data  
on  
NETPIE  
Shadow

## WIDGET

TYPE	<b>Gauge</b>	
TITLE	r	
VALUE	<code>datasources["datasource1"]["shadow"]["r"]</code>	<b>DATASOURCE</b>
UNITS	Volts	
MINIMUM	0	
MAXIMUM	3	

# Gauge widget settings

Click on [+DATASOURCE] on the right and make selection

# Slider settings

```
netpie["datasourcename"].publish("@msg/cmd","step='"+String(value))
```

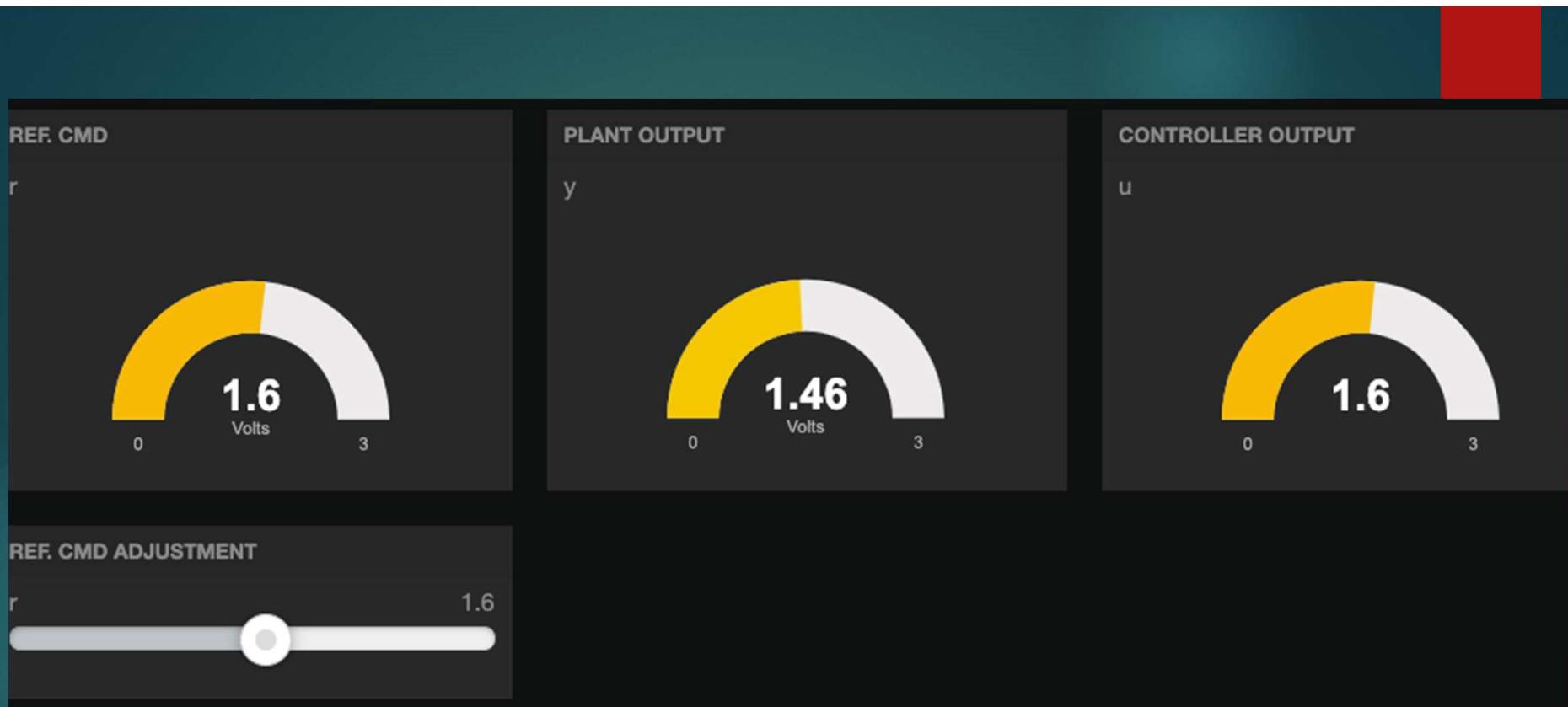
The screenshot shows the configuration interface for a Slider widget. The left side has a dark teal background with the title "Slider settings". The right side is a configuration panel with a red header bar.

**Slider Settings:**

- TYPE:** Slider
- SLIDER CAPTION:** r
- FILLED COLOR:** Grey
- DISPLAY VALUE:** YES (with an orange square icon)
- MIN VALUE:** 0
- MAX VALUE:** 3
- STEP:** 0.1
- INITIAL VALUE:** 0
- The default value set only the first time the widget is loaded.
- AUTO UPDATED VALUE:** `datasources["datasource1"]["shadow"]["r"]`
- Slider will be updated upon the change of variables (e.g. other data sources).
- ONSTART ACTION:** (empty)
- Add some Javascript here. You can access to a slider attribute using variables 'value' and 'percent'.
- ONSLIDE ACTION:** (empty)
- Add some Javascript here. You can access to a slider attribute using variables 'value' and 'percent'.
- ONSTOP ACTION:** `netpie["datasource1"].publish("@msg/cmd","step='"+String(value))`
- Add some Javascript here. You can access to a slider attribute using variables 'value' and 'percent'.

**Buttons at the bottom:**

- ONSTOP ACTION (button)
- JS code to run after a button is created (button)
- SAVE
- CANCEL



# Freeboard Layout

# References

- ▶ Books

- ▶ K.J. Astrom and T. Hagglund. Advanced PID Control. Instrumentation, Systems, and Automation Society, 2006.
- ▶ Varodom Toochinda. Feedback Control with Scilab and Arduino. dew.ninja, 2017.

- ▶ Websites

- ▶ <http://dewnijathai.blogspot.com>
- ▶ <https://netpie.io>

Thank You

