

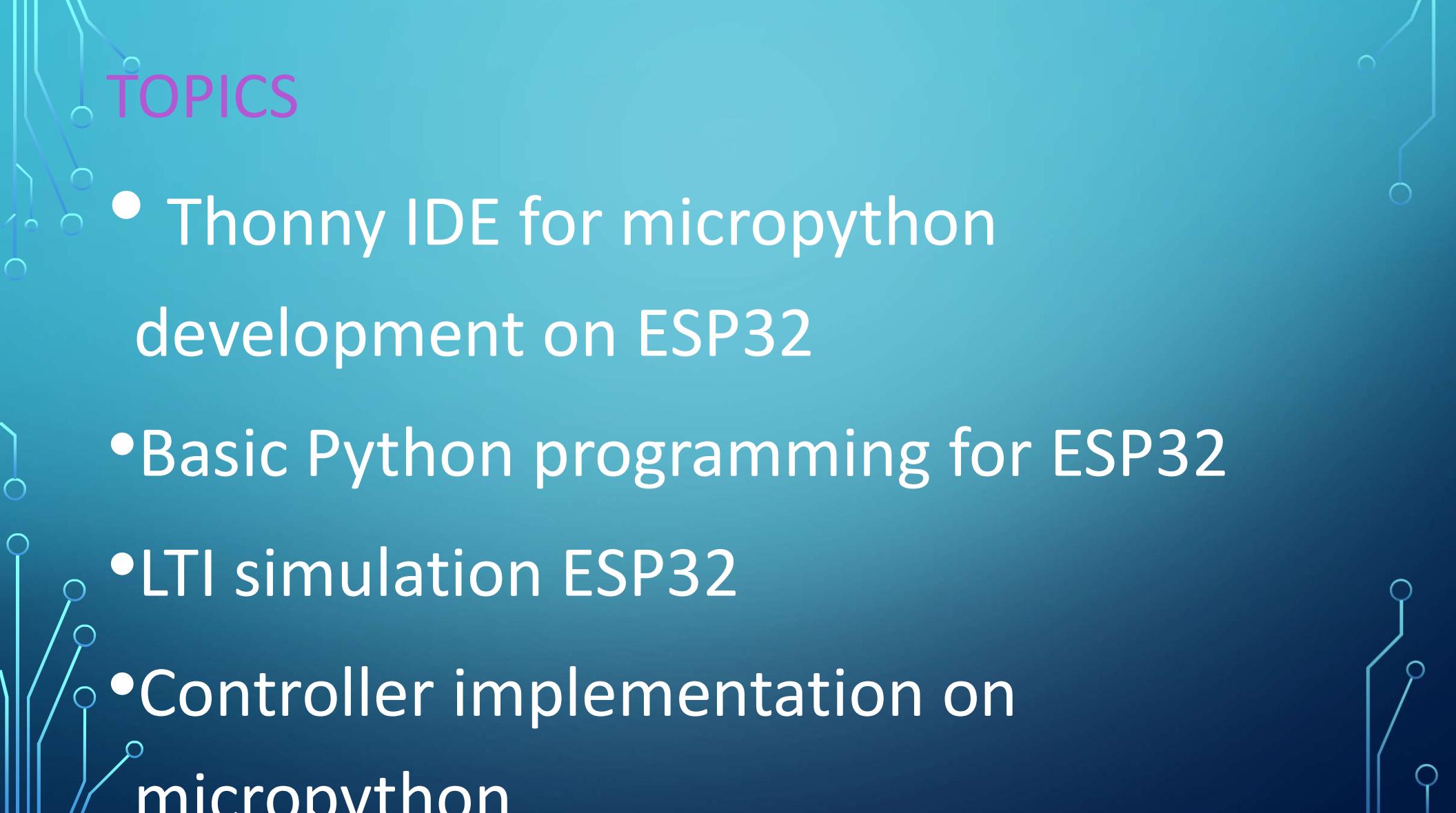
MICROPYTHON DEVELOPMENT ON ESP32

DR. VARODOM TOOCHINDA

DEPT. OF MECHANICAL ENGINEERING

KASETSART UNIVERSITY

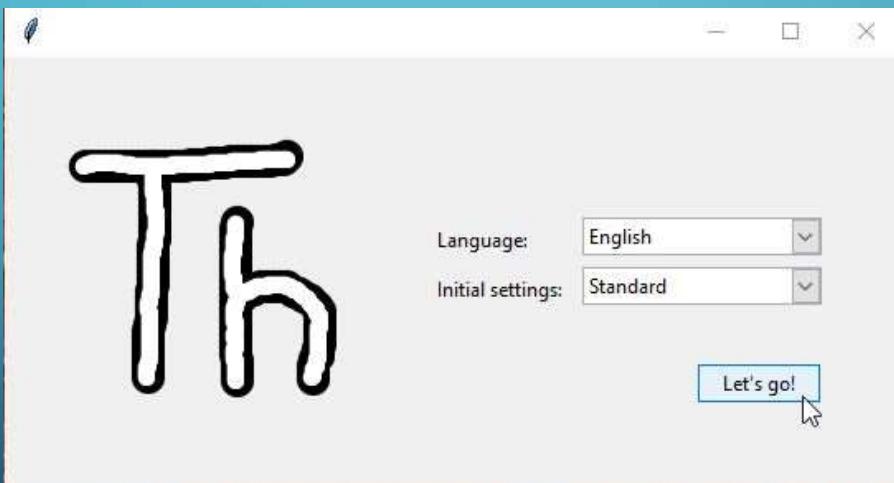
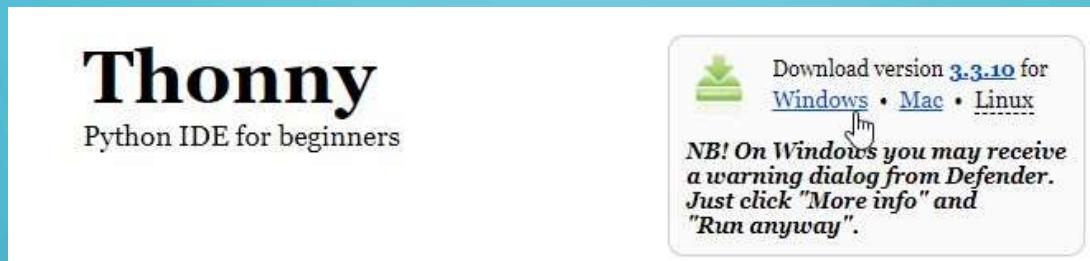
01205479 IoT for Electrical Engineering



TOPICS

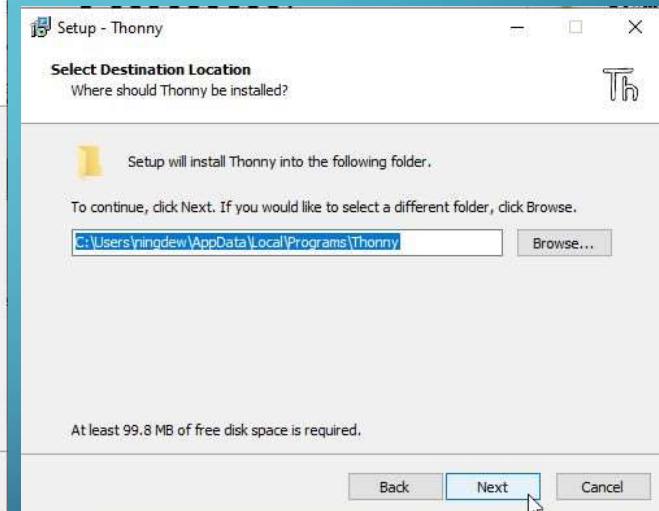
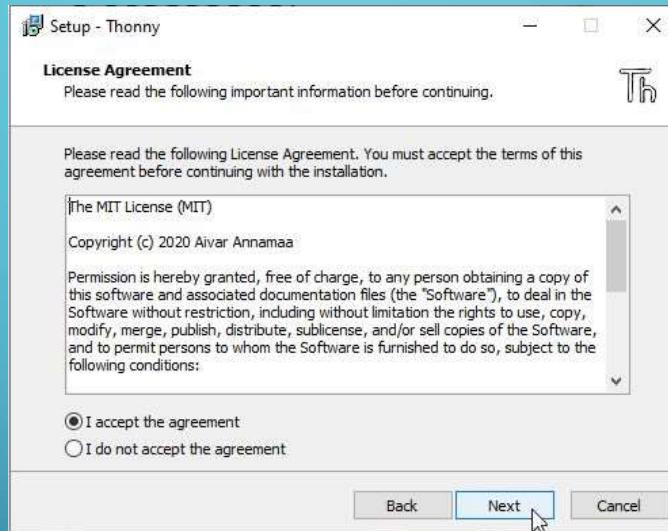
- Thonny IDE for micropython development on ESP32
- Basic Python programming for ESP32
- LTI simulation ESP32
- Controller implementation on micropython

DOWNLOAD AND INSTALL THONNY IDE

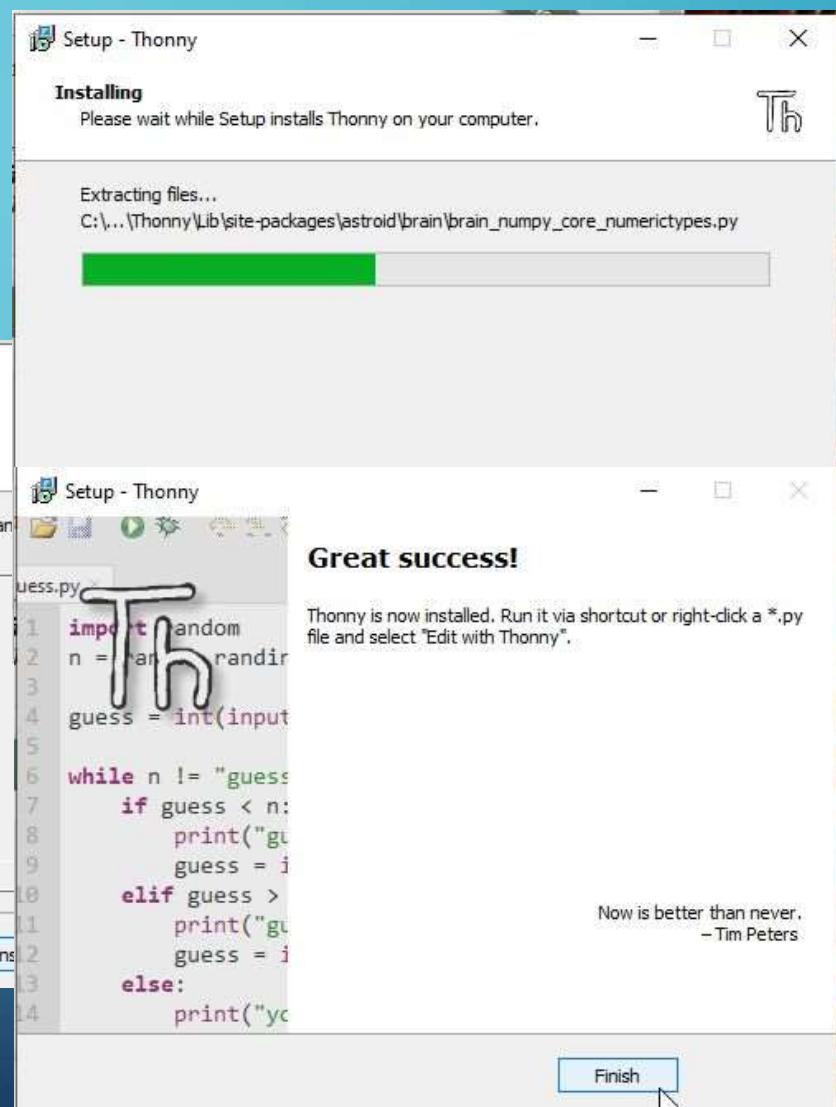
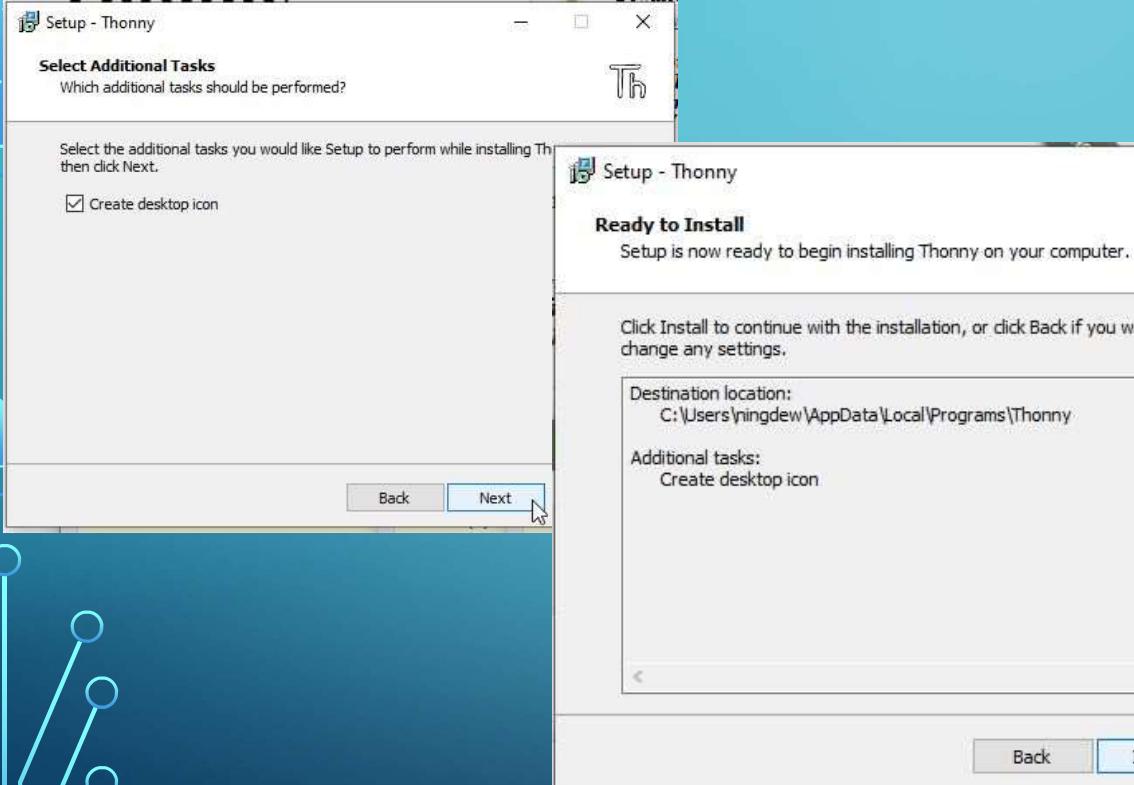


Note : newer version may have slightly different look, but the steps are the same.

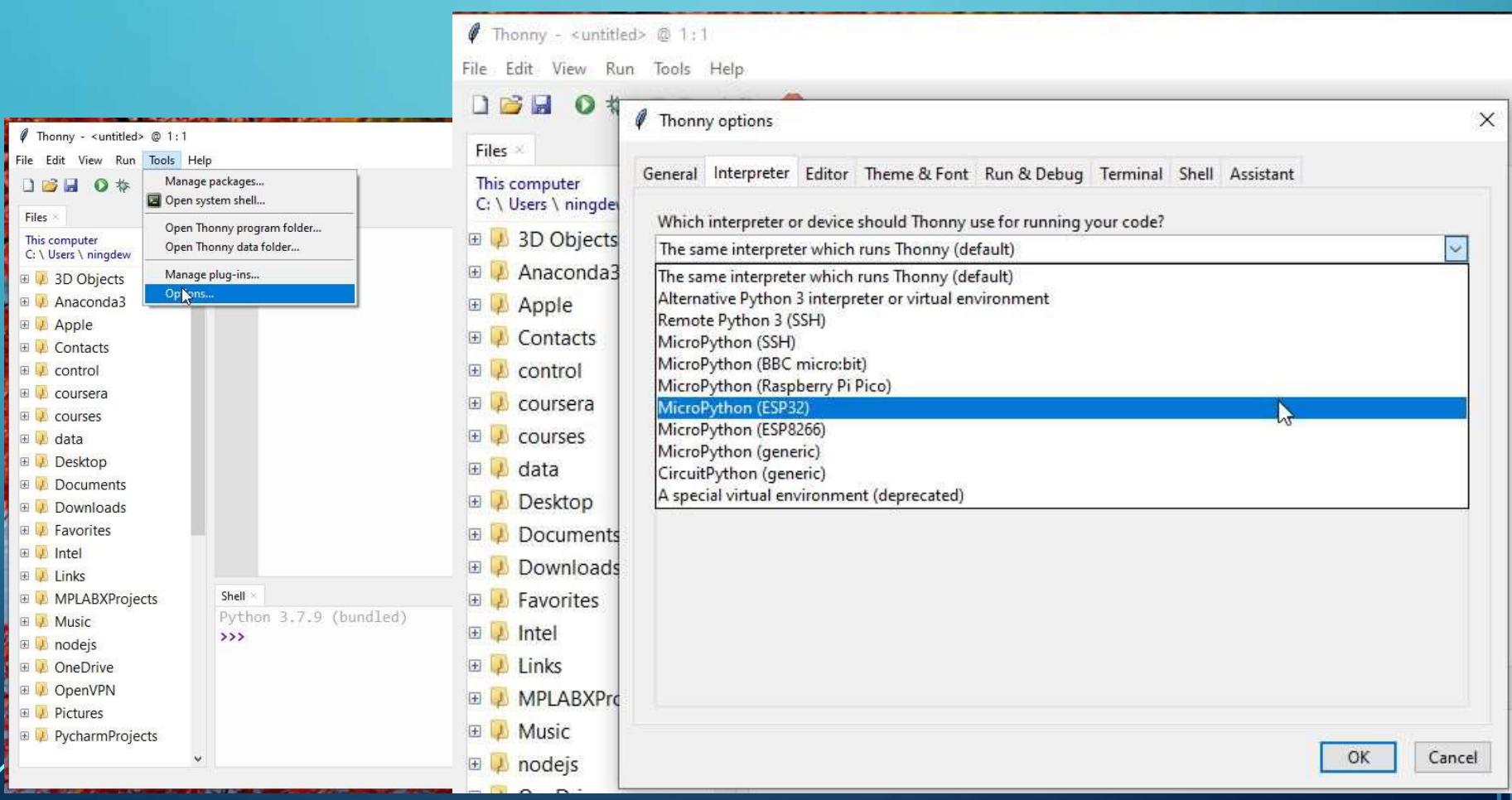
INSTALL THONNY (1)



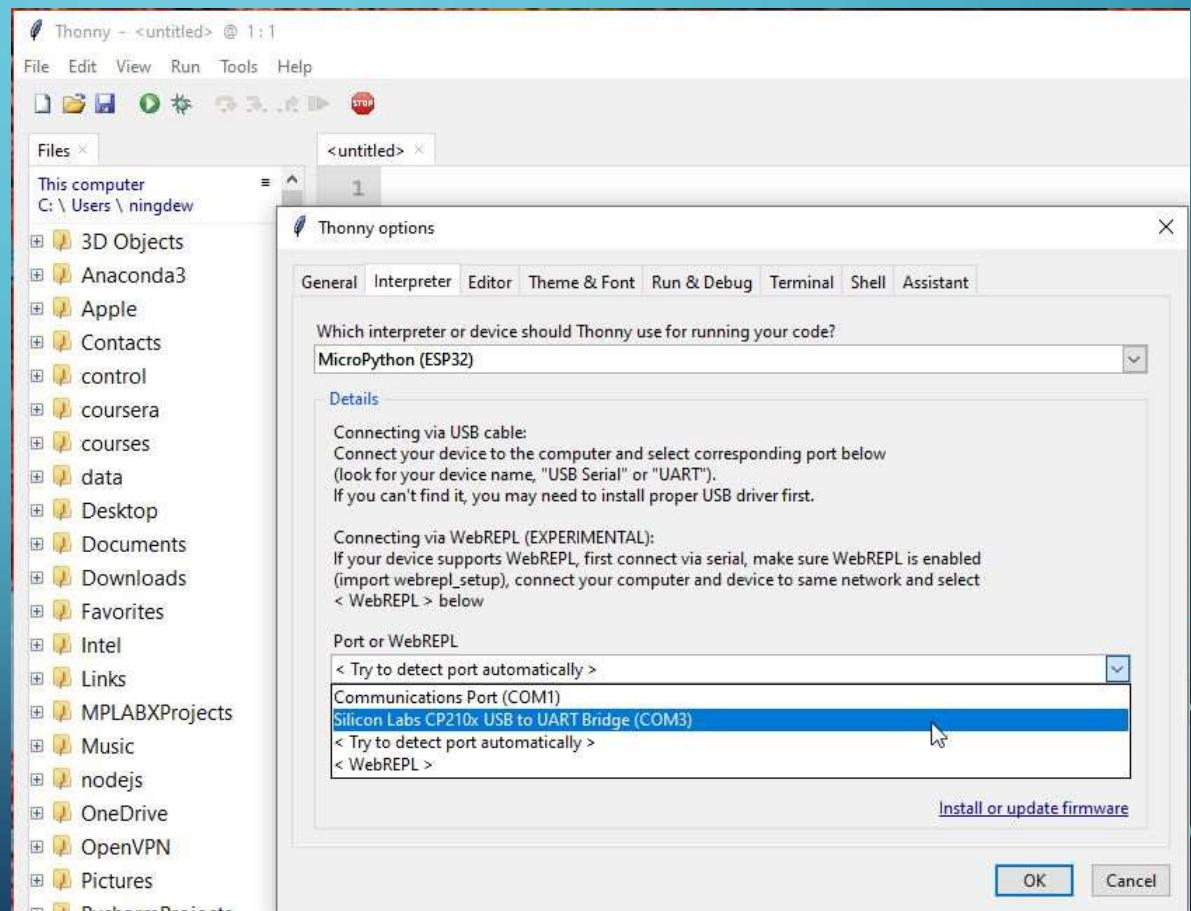
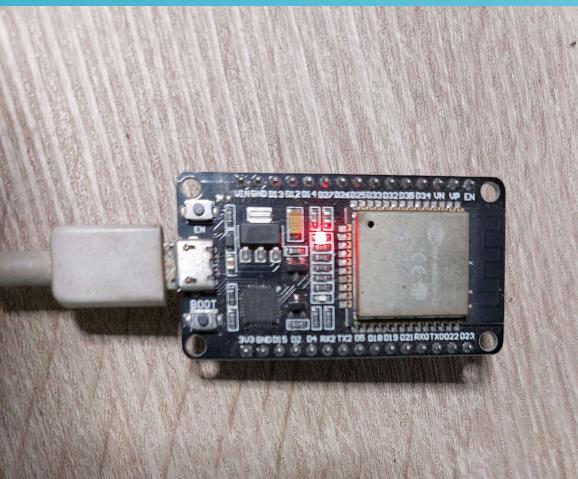
INSTALL THONNY (2)



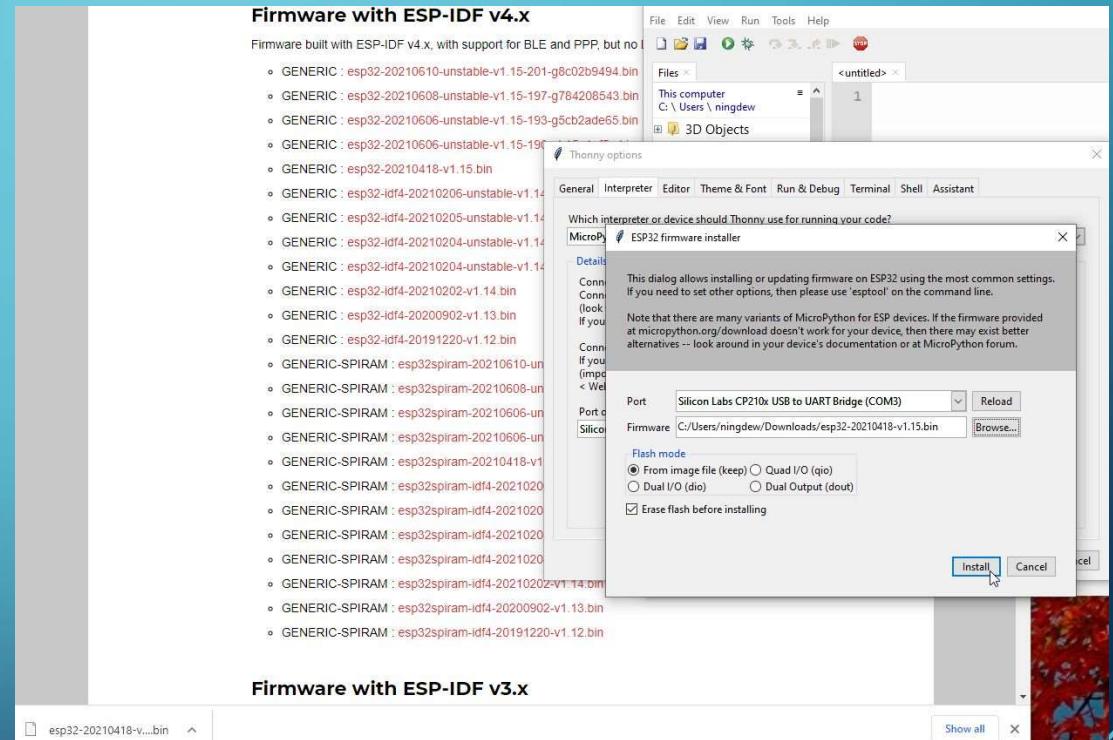
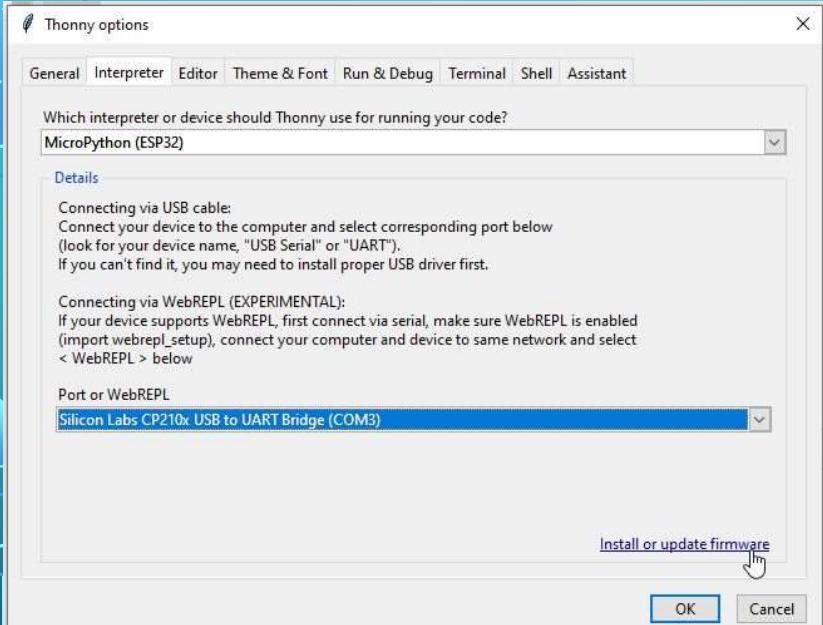
OPEN THONNY AND CHOOSE OPTION →INTERPRETER



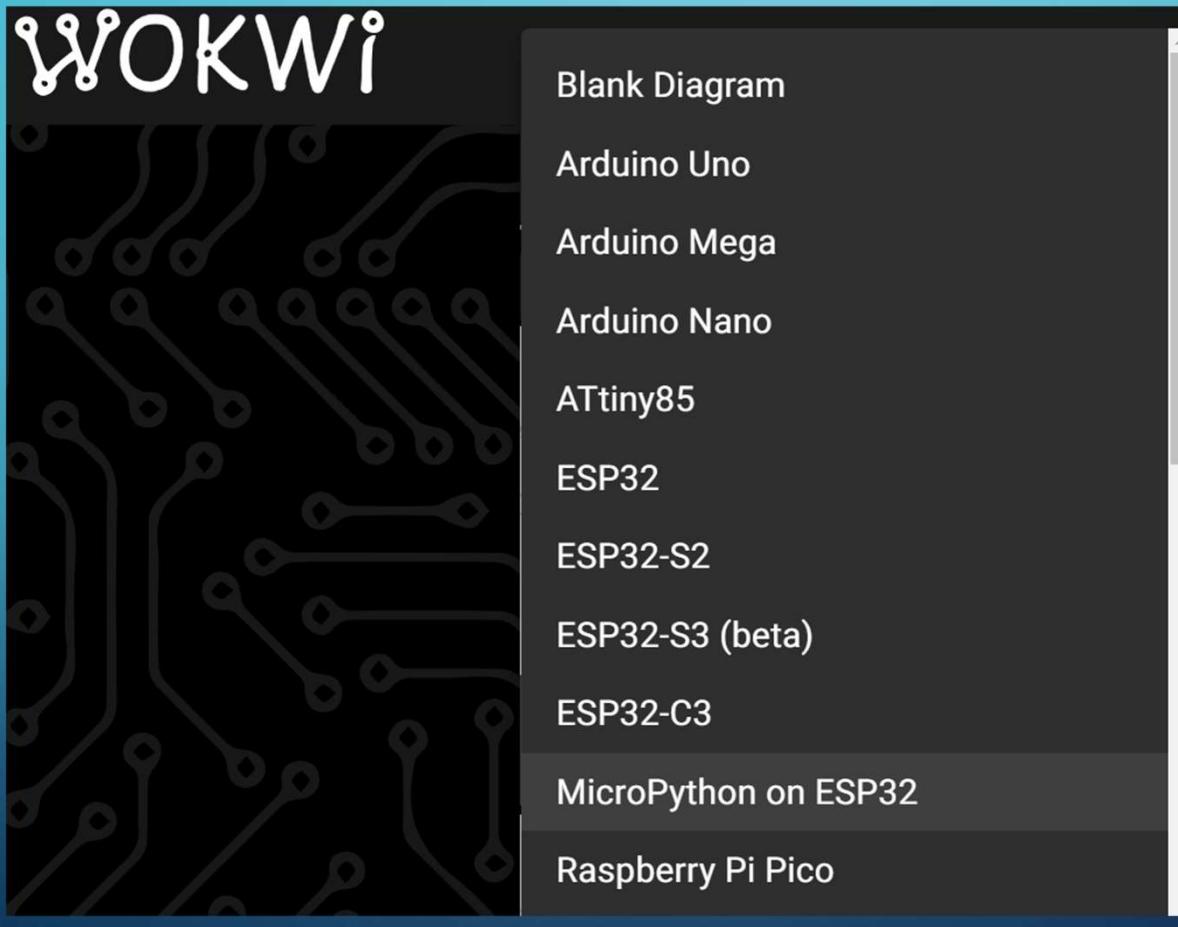
CONNECT ESP32 TO USB AND INSTALL FIRMWARE (1)



INSTALL FIRMWARE (2)



Simulation on Wokwi



LED_BLINK.PY



```
from machine import Pin  
from utime import sleep_ms  
  
led = Pin(2, Pin.OUT)  
  
while True:  
    led.on()  
    print("LED is ", led.value())  
    sleep_ms(500)  
    led.off()  
    print("LED is ", led.value())  
    sleep_ms(500)
```

TEST PWM (PWM_OUT.PY)



```
from machine import Pin, PWM
from utime import sleep_ms
PWMMAX = 1023 # maximum value for PWM output
pwm2 = PWM(Pin(2)) # assign output to pin 2
pwm2.freq(2000) # set frequency
pwmval = 0 # PWM value
step = 10
direction = 0 # 0 = increase, 1 = decrease

while True:
    if direction==0:
        pwmval += step
    else:
        pwmval -= step
    if pwmval>PWMMAX:
        pwmval = PWMMAX
        direction = 1 # decrease
    elif pwmval<0:
        pwmval = 0
        direction = 0 # increase
    pwm2.duty(pwmval) # send output
    sleep_ms(10)
```

TEST ADC (ADC0_TEST.PY)

```
# command to set range 0 - 3.6 volts
adc.atten(ADC.ATTN_11DB)
```

The screenshot shows the Thonny IDE interface for an ESP32 MicroPython project. The central area displays the code for `adc0_test.py`:

```
1 #adc0_test.py
2 # dew.ninja June 2021
3 # test reading from ADC0 (GPIO36)
4
5 from machine import Pin, ADC
6 from utime import sleep_ms
7
8 adc = ADC(Pin(36))
9
10 while True:
11     adcval = adc.read()
12     print(adcval)
13     sleep_ms(1000)
14
```

The left sidebar shows the file tree:

- This computer
- / Users / dewdotninja / Desktop / Dewwork / Dew2021 / active / research / upython / ESP32
 - chapter5
 - firmware
 - images
 - lag3
 - line_trigger_eventhook
 - NETPIE2020
 - nodered
 - udemy_demo
 - adc0_test.py**
 - dht22.py
 - hc_sr04.py
 - led_blink.py
 - memtest.py
 - netpie_cmddtest.py
 - netpie_dht_test.py
 - netpie_test.py
 - netpie_ultrasonic.py

The bottom right pane shows the MicroPython device shell output and a graph of the ADC values over time.

Shell Output:

```
787
2473
4095
4005
2951
2476
2477
2479
2477
2478
2475
2475
2476
2480
2477
```

Graph: A line graph showing the raw ADC values. The y-axis ranges from 0 to 5000. The signal fluctuates between approximately 2000 and 3000, with several sharp peaks reaching up to 5000.

Read digital input

din_test.py

The screenshot shows a software interface for developing microcontroller projects. On the left, a code editor displays the file `main.py` with the following content:

```
1 # din_test.py
2 # dew.ninja June 2023
3 # test digital input with switch on NodeMCU-32S
4
5 import math
6 import time
7 from time import sleep_ms
8 from machine import Pin, PWM
9
10 obled = PWM(Pin(2)) # on-board led
11 button = Pin(13,Pin.IN, Pin.PULL_UP) # pushbutton switch
12
13 PWMMAX = 1023
14 PWMMID = 511
15 PWMMIN = 0
16
17 # genchirp() check button status
18 # and generate chirp signal if pressed
19
20 def genchirp(): # check button status and generate chirp signal if pressed
21     stepnum = 10
22     sinstep = 6.28/stepnum
23     sinangle = 0
24     dt = 30
25     cyclecomplete = False
26     if not button.value():
27         sleep_ms(100) # debounce
28         while not button.value(): # blocked if switch pressed
```

The simulation window on the right shows a breadboard setup with an ESP32 module. A green pushbutton is connected between pin 13 and ground. A red LED is connected between the ESP32's digital output pin 2 and ground, with a resistor in series. The simulation status bar indicates the time as 02:21.005.

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp
:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:4656
load:0x40078000,len:13284
ho 0 tail 12 room 4
load:0x40080400,len:3712
entry 0x4008064c
```

DAC1_TEST.PY

```
from machine import Pin, ADC, DAC
from utime import sleep_ms
DACMAX = 255
adc = ADC(Pin(39)) # ADC1_CH3
adc.atten(ADC.ATTN_11DB) # range 0 - 3.6 V
dac1 = DAC(Pin(25))
while True:
    adcval = adc.read()
    dacval = int(adcval/4)
    if dacval>DACMAX:
        dacval = DACMAX
    dac1.write(dacval)
    sleep_ms(1000)
```

Note : DAC does not work on Wokwi simulation!

TIMER_TEST.PY

```
import machine
from machine import Pin, Timer #importing pin, and timer class
led= Pin(2, Pin.OUT)          # GPIO2 as on-board led

led.value(0)                  #LED is off
timer=Timer(-1)

timer.init(period=1000, mode=Timer.PERIODIC, callback=lambda
t:led.value(not led.value()))
```

Utilize timer by writing callback function (Timer_test2.py)

```
import machine
from machine import Pin, Timer          #importing pin, and timer class
led= Pin(2, Pin.OUT)                    # GPIO2 as on-board led

led.value(0)                           #LED is off
timer1=Timer(1)

def timer_isr(event):
    led.value(not led.value())

timer1.init(period=1000, mode=Timer.PERIODIC, callback=timer_isr)
#initializing the timer
```

Set period using time library

time_ticks.py

```
import time
from machine import Pin

led= Pin(2, Pin.OUT)          # GPIO2 as led output
blink_period = 1000
time_prev = 0

while True:
    time_current = time.ticks_ms()
    if time_current<time_prev:
        time_prev = time_current # prevent overflow
    if (time_current - time_prev)>blink_period:
        led.value(not led.value()) # blink on-board led
        time_prev = time_current
```

DHT SENSOR (DHT22.PY)

```
import dht
import machine
from utime import sleep_ms
d = dht.DHT22(machine.Pin(4))
while True:
    d.measure()
    temperature=d.temperature()
    humidity=d.humidity()
    print("temperature = ",temperature)
    print("humidity = ",humidity)
    sleep_ms(2000)
```

DHT22 simulation on Wokwi

WOKWI SAVE SHARE Do

main.py • diagram.json •

```
1 #DHT22_wokwi.py
2 # dew.ninja Sep 2023
3
4 import dht
5 import machine
6 from utime import sleep_ms
7 d = dht.DHT22(machine.Pin(4))
8 while True:
9
10     d.measure()
11     temperature=d.temperature()
12     humidity=d.humidity()
13     print("temperature = ",temperature)
14     print("humidity = ",humidity)
15     sleep_ms(2000)
16
```

Simulation

temperature = 24.0
humidity = 40.0
temperature = 24.0
humidity = 40.0

DHT TEST (DHT22.PY)



```
dht22.py      led_blink.py    netpie_test.py
1  # dht22.py
2
3  import dht
4  import machine
5  from utime import sleep_ms
6  d = dht.DHT22(machine.Pin(4))
7  while True:
8
9      d.measure()
10     temperature=d.temperature()
11     humidity=d.humidity()
12     print("temperature = ",temperature)
13     print("humidity = ",humidity)
14     sleep_ms(2000)
```

```
Shell
humidity = 61.3
temperature = 25.4
humidity = 61.3
temperature = 25.3
humidity = 61.2
```

cmd_interpreter.py (or cmd interpreter ww.py for wokwi)

The screenshot shows a code editor window with a Python script named `main.py`. The script implements a command-line interpreter that can handle commands like `dbupdate` with parameters or no parameters. The code uses `sys` and `utime` modules for input and sleep operations. The right side of the interface displays a terminal window titled "Simulation" showing the execution of the script.

```
1  # cmd_interpreter_ww.py
2  # dew.ninja August 2023
3  # for Wokwi simulation. Use input() instead of sys.stdin.readline()
4  # Separate command from parameter
5
6  #import sys
7  from utime import sleep_ms
8
9  #command interpreter function
10 def cmdInt(userstr):
11     result = userstr.find("=")
12     if result == -1:
13         noparm = 1
14         cmdstr = userstr.strip()
15     else:
16         noparm = 0
17         splitstr = userstr.split("=")
18         cmdstr = splitstr[0].strip()
19         parmstr = splitstr[1].strip()
20         print("command = " + cmdstr)
21         if noparm:
22             print('No parameter in command')
23         else:
24             print("parameter = " + parmstr)
25
26 def user_input():
27     print('=====')
28     print('Enter command :')
```

Simulation

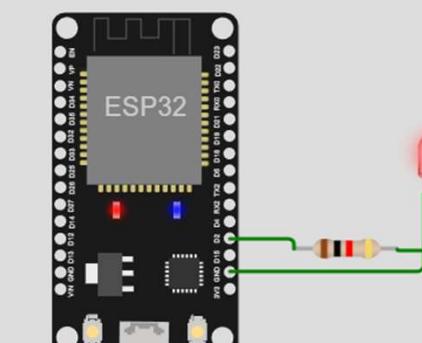
```
led=1
command = led
parameter = 1
=====
Enter command :
dbupdate
command = dbupdate
No parameter in command
=====
Enter command :
```

cmdtest2_ww.py

```
main.py    diagram.json
```

```
1  # cmdtest2_ww.py
2  # dew.ninja Sep 2023
3  # for Wokwi simulation
4
5  from machine import Pin, Timer
6  #import sys
7  from utime import sleep_ms
8
9  led = Pin(2, Pin.OUT)
10 led_period = 1000 # default
11 toggle_mode = 1
12 led_status = 0
13
14 timer=Timer(1)
15 def timer_isr(event):
16     led.value(not led.value())
17
18 timer.init(period=led_period, mode=Timer.PERIODIC,
19             callback=timer_isr)
20
21 # helper function for cmdInt()
22 # separate command and parameter
23 # at "="
24 def splitCmd(userstr):
25     result = userstr.find("=")
26     if result == -1:
27         noparm = 1
28         cmdstr = userstr.strip()
```

Simulation



Enter command :
toggle=0

Enter command :
led
Current led status = 1

Enter command :
period
Current led period = 1000

Enter command :

Exercise 1:

Develop a program to change RGB LED color on LAG3 board randomly (or on Wokwi)

R, G, B are connected to pin 19,18,17, respectively. (on Wokwi, change pins as you like.)

Rate to change color is adjustable via command $\text{rate} = x$ (Hz)

The command interpreter must limit rate to within 0.1 – 10 Hz

Template on wokwi : <https://wokwi.com/projects/406465660086615041>

HCSR04 example

<https://wokwi.com/projects/406627227063083009>

WOKWi SAVE SHARE netpie_ultrasonic_oled Docs

main.py diagram.json hcsr04.py ssd1306.py

```
169     while True:
170         if distance < mindist:
171             alert_status = 1
172             led.on()
173         else:
174             alert_status = 0
175             led.off()
176         oled.show(oled,mindist,distance,alert_status)
177         if online:
178             time_current = time.ticks_ms()
179             publish_delta = time_current - time_prev
180             if publish_delta>PUBLISH_PERIOD: # publish interval must be
181                 time_prev = time_current
182                 client.check_msg()
183
184
185
186             # -- this is code for writing data to shadow ---
187             # --- change to direct message method if you want to see
188             # --- to dashboard or node-red
189             sensor_data['min_distance'] = mindist
190             sensor_data['distance'] = distance
191             sensor_data['alert'] = alert_status
192             publish_str = ujson.dumps({"data": sensor_data})
193             print(publish_str)
194             client.publish("@shadow/data/update", publish_str)
195
196
197             #time.sleep_ms(led_period)
198
```

Simulation

00:10.032 99%

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00
,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030, len:4728
load:0x40078000, len:14876
ho 0 tail 12 room 4
load:0x40080400, len:3368
entry 0x400805cc

MPU6050 example

<https://wokwi.com/projects/406632207671524353>

WOKWI SAVE SHARE nuws24_mpu6050

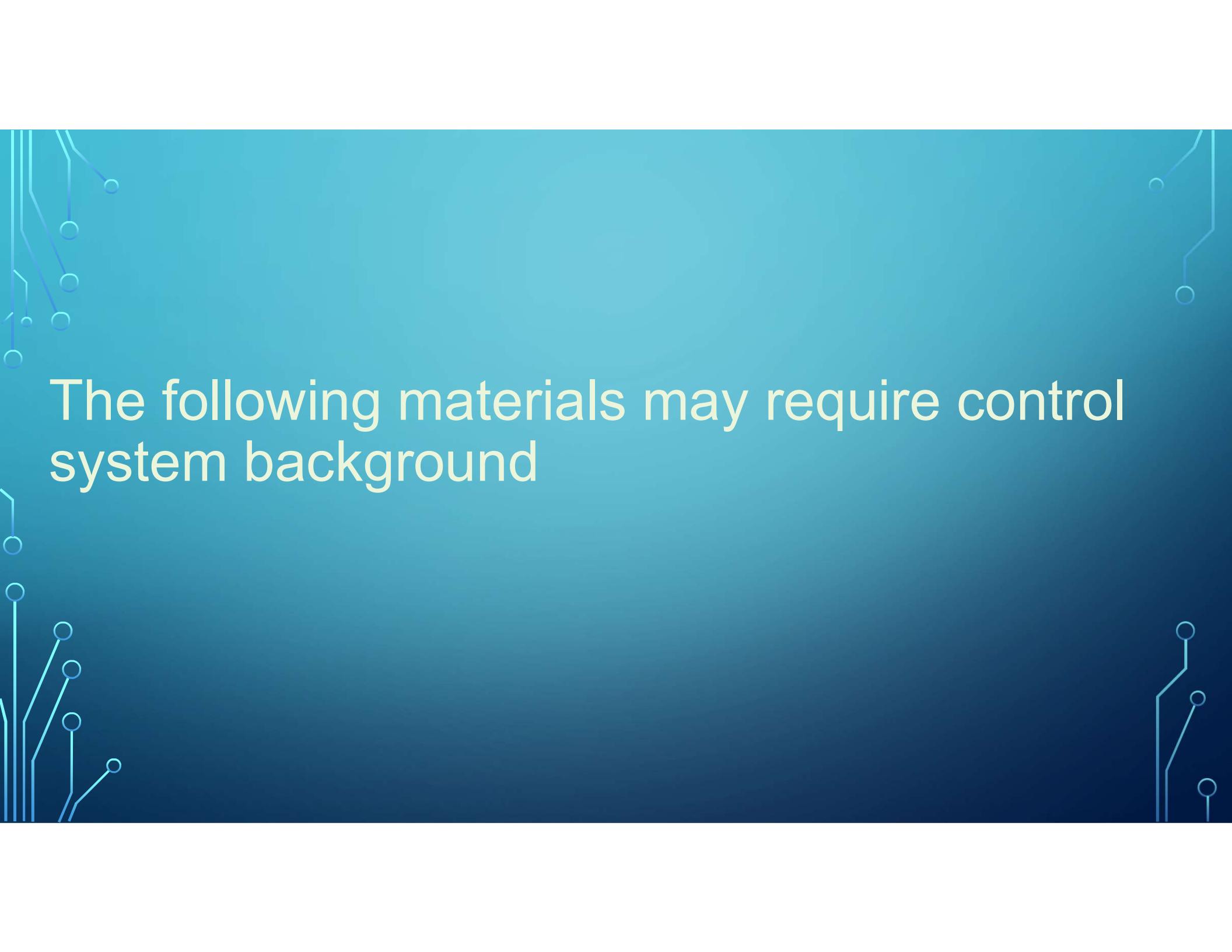
main.py README.md diagram.json mpu6050.py

Simulation Description

```
1 from machine import SoftI2C
2 from machine import Pin
3 #from machine import sleep
4 from time import sleep
5 import mpu6050
6 i2c = SoftI2C(scl=Pin(22), sda=Pin(21))      #initialise I2C
7 mpu= mpu6050.accel(i2c)
8 while True:
9     global mpudata
10    mpudata = mpu.get_values()
11    print(mpudata)
12    sleep(1)
```

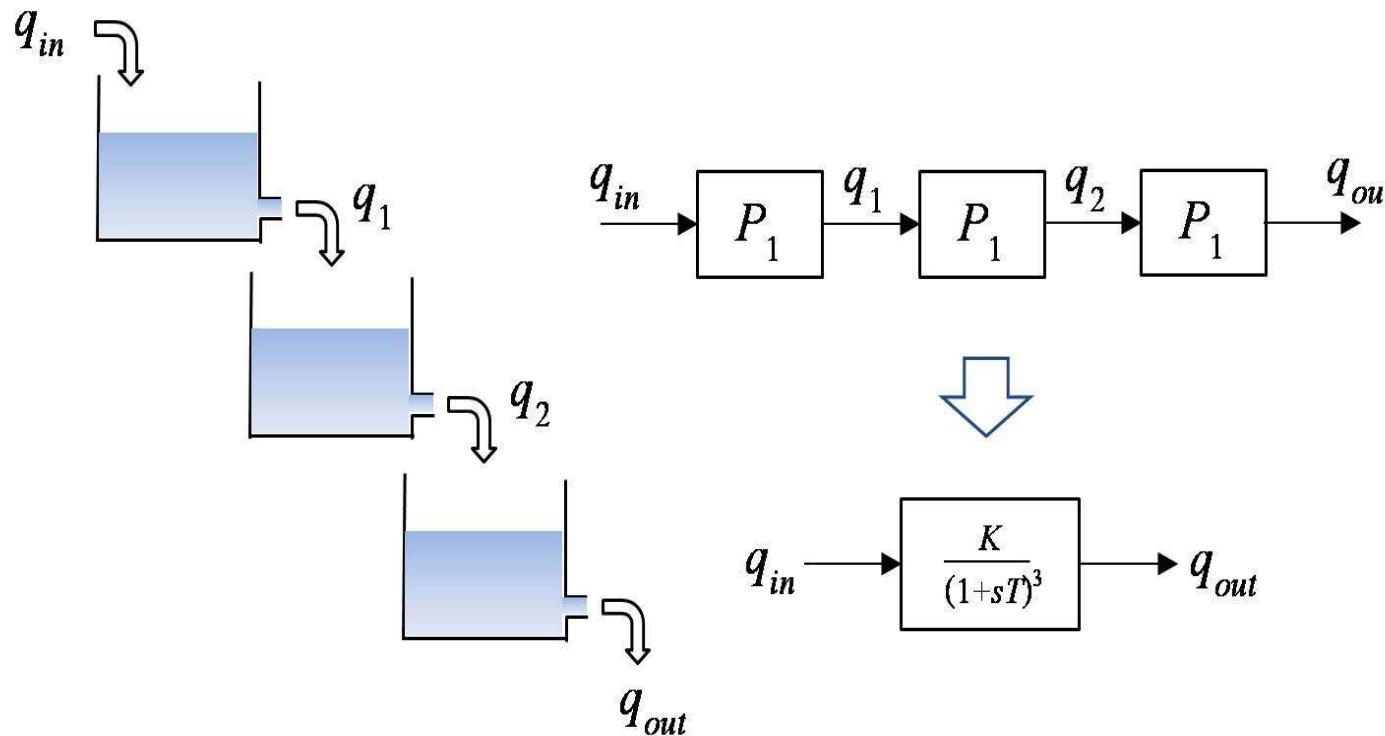
The simulation interface shows an ESP32 development board connected to a blue breadboard. The breadboard has several components: a green LED, a yellow pushbutton, a red pushbutton, and a blue potentiometer. Wires connect the ESP32 pins to these components. The simulation window displays a series of JSON objects representing sensor data:

```
{'GyZ': 0, 'GyY': 0, 'GyX': 15065, 'Tmp': 24.00059, 'AcZ': 0}
{'GyZ': 0, 'GyY': 0, 'GyX': 15065, 'Tmp': 24.00059, 'AcZ': 0}
{'GyZ': 0, 'GyY': 0, 'GyX': 15065, 'Tmp': 24.00059, 'AcZ': 0}
{'GyZ': 0, 'GyY': 0, 'GyX': 15065, 'Tmp': 24.00059, 'AcZ': 0}
{'GyZ': 0, 'GyY': 0, 'GyX': 15065, 'Tmp': 24.00059, 'AcZ': 0}
```



The following materials may require control system background

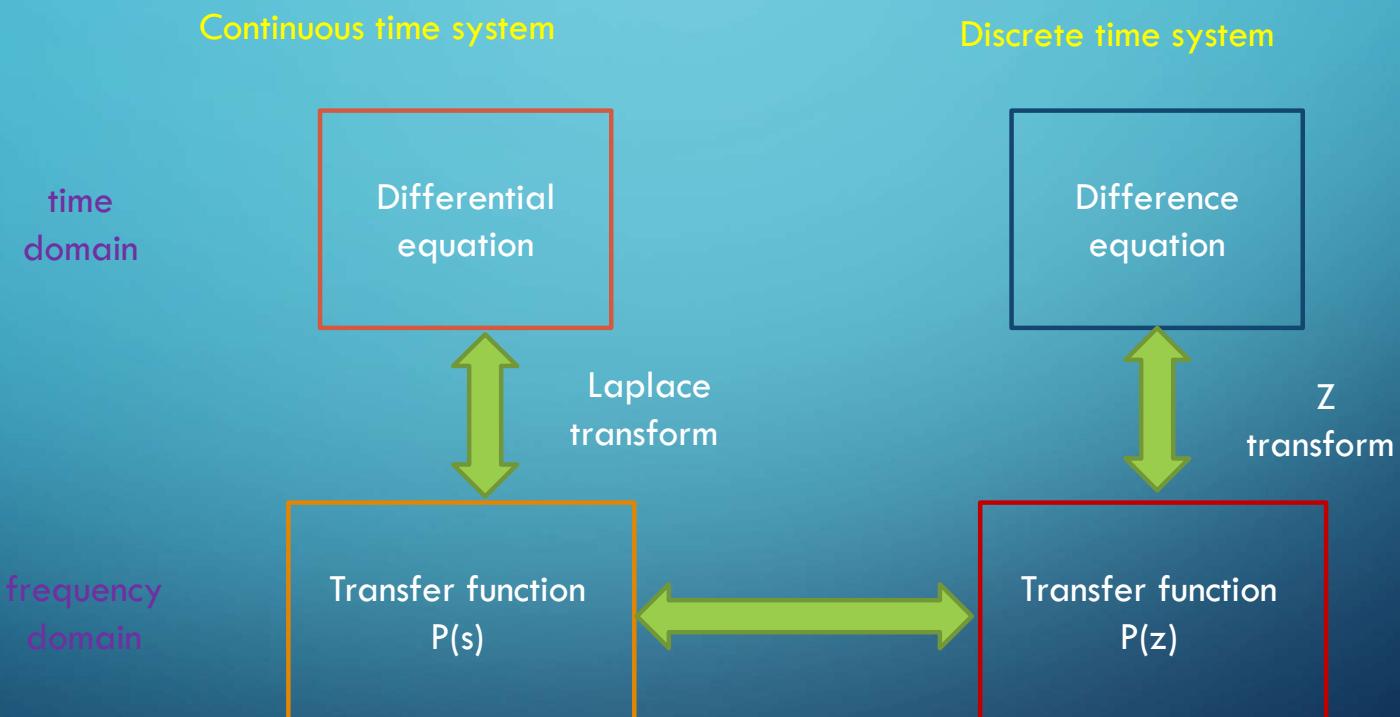
LTI SIMULATION



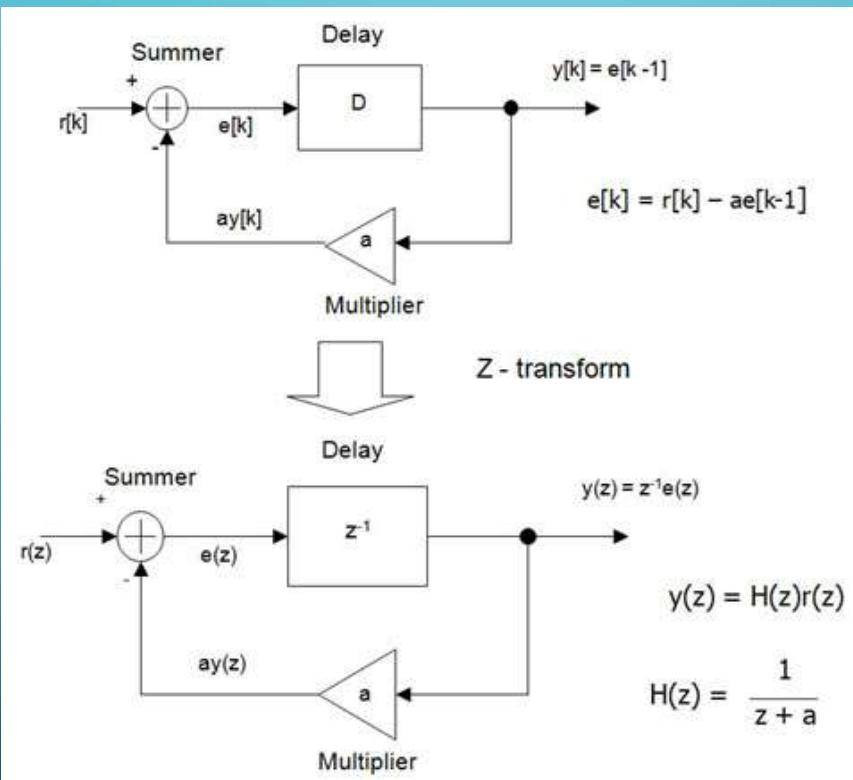
normalized
transfer function

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

CONTINUOUS-DISCRETE RELATIONSHIPS



DISCRETE-TIME SYSTEM DESCRIPTION



Difference equation

Discrete-time
Transfer function

TRANSFORMATION OF C(S) TO C(Z)

Forward difference

$$C(z) = C(s) \Big|_{s \rightarrow \frac{z-1}{T}}$$

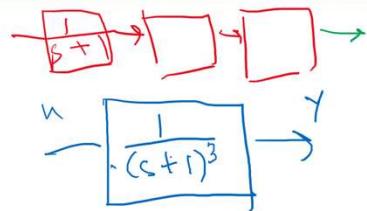
Backward difference

$$C(z) = C(s) \Big|_{s \rightarrow \frac{z-1}{Tz}}$$

Bilinear transform

$$C(z) = C(s) \Big|_{s \rightarrow \frac{2z-1}{Tz+1}}$$

CONVERT $1/(S+1)$ TO DISCRETE TRANSFER FUNCTION AND DIFFERENCE EQUATION



Bilinear Transform
 $s \leftarrow \frac{2(z-1)}{T(z+1)}$

$$\frac{1}{s+1} = \frac{1}{\frac{2(z-1)}{T(z+1)} + 1} = \frac{1}{\frac{2(z-1) + T(z+1)}{T(z+1)}} = \frac{Tz + T}{2z - 2 + Tz + T}$$



$$Y_1(z) = \frac{1 + z^{-1}}{a + b z^{-1}}$$

$$\frac{(z-1)Tz + T}{(z-1)(2+T)z + T-2} \checkmark = \frac{T + Tz^{-1}}{(2+T) + (T-2)z^{-1}}$$

$$Y_1(z) = (T + Tz^{-1})U(z)$$

$$\Downarrow$$

$$ay_1(k) + by_1(k-1) = T(u(k) + u(k-1))$$

SIMULATION ON JUPYTER NOTEBOOK (1)

$$P(s) = \frac{1}{s+1} \Rightarrow P(z) = \frac{T+z^{-1}}{a+bz^{-1}}$$
$$a = (z+T) \\ b = (T-z)$$
$$u \rightarrow \boxed{P(z)} \quad y_1$$
$$ay_1(k) + by_1(k-1) = T(u(k) + u(k-1))$$
$$y_1(k) = \frac{1}{a} (-by_1(k-1) + T(u(k) + u(k-1)))$$

```
import numpy as np  
import matplotlib.pyplot as plt  
import control as ctrl
```

input vectors

- `u_states[0]` = previous input of 1st section
- `u_states[1]` = current input of 1st section
- `u_states[2]` = previous input of 2nd section
- `u_states[3]` = current input of 2nd section
- `u_states[4]` = previous input of 3rd section
- `u_states[5]` = current input of 3rd section

output vectors

- `y_states[0]` = previous output of 1st section
- `y_states[1]` = current output of 1st section
- `y_states[2]` = previous output of 2nd section
- `y_states[3]` = current output of 2nd section
- `y_states[4]` = previous output of 3rd section
- `y_states[5]` = current output of 3rd section

SIMULATION ON JUPYTER NOTEBOOK (2)

$$P(z) = \frac{1}{z+1} \rightarrow P(z) = \frac{T+z^{-1}}{a+bz^{-1}} \quad a = (z+T) \quad b = (T-z)$$

$u \rightarrow [P(z)] \gamma_1$

$$\alpha y_1(z) + b y_1(z^{-1}) = T(u(z) + u(z^{-1}))$$

$$y_1(z) = \frac{1}{a} (-bz + T(u(z) + u(z^{-1})))$$

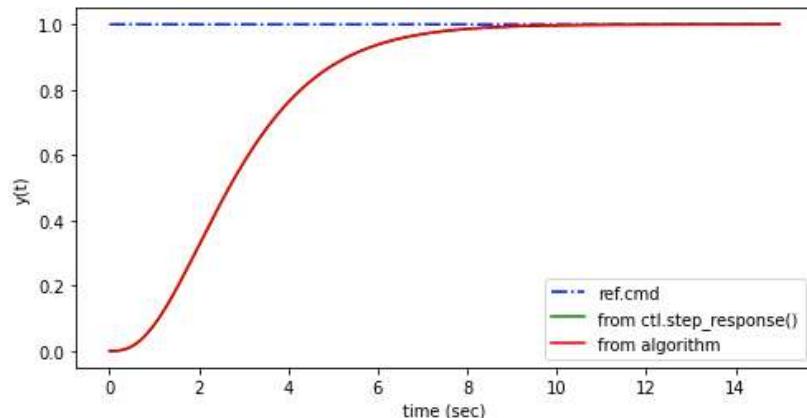
```
def lag3(a,b,T, u, u_states, y_states):
    for k in range(3):
        y_states[2*k] = y_states[2*k+1]
        u_states[2*k] = u_states[2*k+1]
        if k == 0:
            u_states[2*k+1] = u
        else:
            u_states[2*k+1] = y_states[2*k-1]
        y_states[2*k+1] = (1/a)*(-b*y_states[2*k]+T*(u_states[2*k+1]+u_states[2*k]))
    return y_states[5]
```

```
# lag3 simulation function
def lag3sim(uvec, tvec):
    yvec = np.zeros(tvec.shape)
    T = tvec[1] - tvec[0]
    a = 2 + T
    b = T - 2
    u_states = np.zeros((6,1))
    y_states = np.zeros((6,1))
    for i in range(len(tvec)):
        yvec[i] = lag3(a,b,T, uvec[i], u_states,
y_states)
    return yvec
```

SIMULATION ON JUPYTER NOTEBOOK (3)

In [31]:

```
1 s = ctl.tf('s')
2 P = 1/(s+1)**3
3 tvec = np.arange(0,15,0.01)
4 uvec = np.ones(tvec.shape)
5 tout, y1 = ctl.step_response(P, tvec)
6 y2 = lag3sim(uvec, tvec)
7 plt.figure(figsize=(8,4))
8 plt.plot(tvec,uvec,'b-.',tvec,y1,'g-',tvec,y2,'r-')
9 plt.xlabel('time (sec)')
10 plt.ylabel('y(t)')
11 plt.legend(['ref.cmd','from ctl.step_response()','from algorithm'])
12 plt.show()
```



SIMULATION ON ESP32 (1)

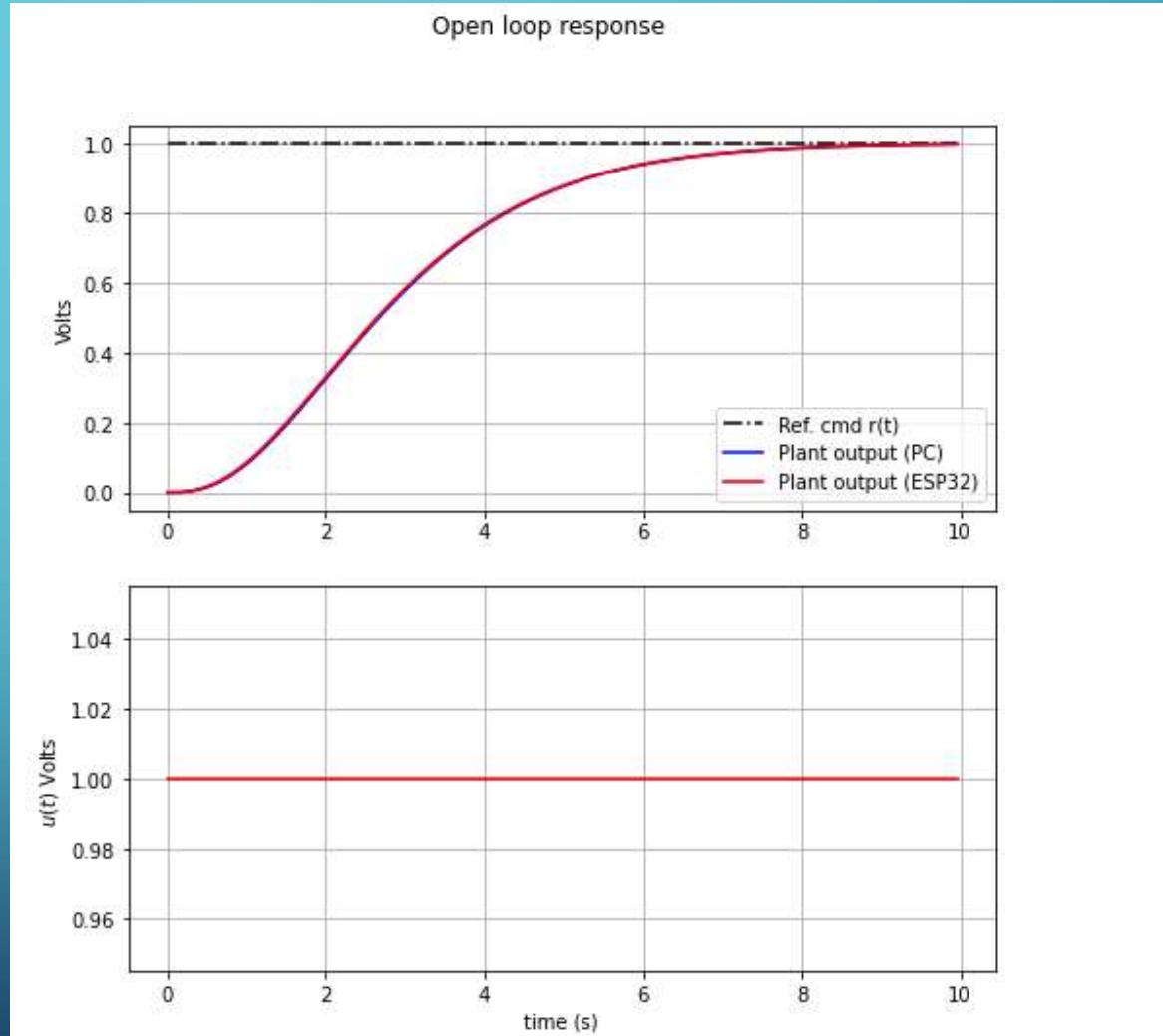
```
lag3.plantsim.py
23 u = 1
24
25 def lag3(a,b,T, u, u_states, y_states):
26     for k in range(3):
27         y_states[2*k] = y_states[2*k+1]
28         u_states[2*k] = u_states[2*k+1]
29         if k == 0:
30             u_states[2*k+1] = u
31         else:
32             u_states[2*k+1] = y_states[2*k-1]
33             y_states[2*k+1] = (1/a)*( b*y_states[2*k]+T*(u_states[2*k+1]+u_states[2*k]))
34     return y_states[5]
35
36
37 for i in range(datasize):
38     if i==0:
39         print("datamat = np.array([")
40
41     while dt < T: # block execution until T expires
42         t_current = time.ticks_ms()
43         dt = t_current - t_previous
44
45         y = lag3(a,b,T,u, u_states, y_states)
46         print("{}, {}, {}, {}, ".format(t,u,y,u))
47         if i==datasize-1:
48             print("])")
49         t+=T
50
```

Shell >

```
MicroPython v1.15 on 2021-04-18; ESP32 module with ESP32
Type "help()" for more information.
MicroPython v1.15 on 2021-04-18; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
datamat = np.array([
[0,1,1.450937e-05,1],
[0.05,1,9.944229e-05,1],
[0.1,1,0.0003459546,1],
[0.15,1,0.0008458681,1],
[0.2,1,0.001676403,1],
[0.25,1,0.002901642,1],
[0.3,1,0.004573871,1],
[0.35,1,0.006734811,1],
[0.4,1,0.009416744,1],
[0.45,1,0.01264355,1],
[0.5,1,0.01643163,1],
```

SIMULATION ON ESP32 (2)

Compare simulation between
PC (Python control library)
and ESP32



COMPARE OPEN-LOOP RESPONSES FROM LAG3

The image shows a Mac desktop with three open windows:

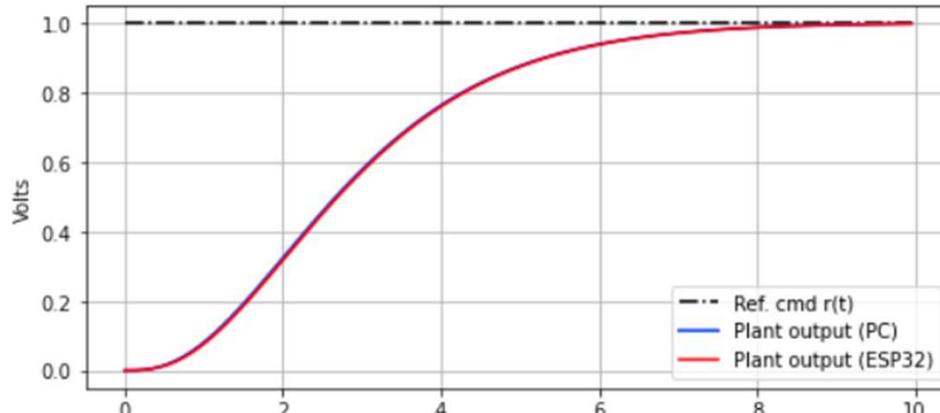
- Jupyter Notebook (olresp_compare2.ipynb):** Displays Python code for plotting open-loop responses and a resulting line graph comparing three data series: "Ref. cmd r(t)" (black dashed line), "Plant output (PC)" (blue solid line), and "Plant output (ESP32)" (red solid line). The x-axis is time (s) from 0 to 10, and the y-axis is Volts from 0.0 to 1.0. The PC and ESP32 outputs track the reference command closely.
- Thonny IDE:** Shows a script named `lag3_ol_net.py` with code for handling commands like `capture` and `datasize`, and a function `update_freeboard()`. It also lists files in the project: `netpie_cmdt1`, `lag3_ol_net`, `dac1_tes`, `pwm16_te`, `pwm_ou`, `timer_tes`, and `lag3_plants`.
- Terminal Window:** A MicroPython shell window titled "MicroPython device" showing a list of files: `lib` and `boot.py`. The shell also displays a list of data points:

Time (s)	Volts
9.3	0.9248351
9.35	0.930989
9.4	0.9292308
9.45	0.9292308
9.5	0.9292308
9.55	0.9292308
9.6	0.9292308
9.65	0.9301099
9.7	0.930989
9.75	0.9292308
9.8	0.9301099
9.85	0.9318681
9.9	0.9345055
9.95	0.9327472

OPEN-LOOP RESPONSE SIMULATION FROM ALGORITHM

```
In [26]: fig, (ax1, ax2) = plt.subplots(2, figsize=(8,8))
fig.suptitle('Open loop response')
ax1.plot(tvec, rvec, 'k-.',tvec, y1,'b-',tvec,yvec,'r-')
ax1.set_ylabel('Volts')
ax1.legend(['Ref. cmd r(t)', 'Plant output (PC)', 'Plant output (ESP32)'])
ax1.grid(True)
ax2.plot(tvec,uvec, 'r-')
ax2.set_xlabel('time (s)')
ax2.set_ylabel('$u(t)$ Volts')
ax2.grid(True)
```

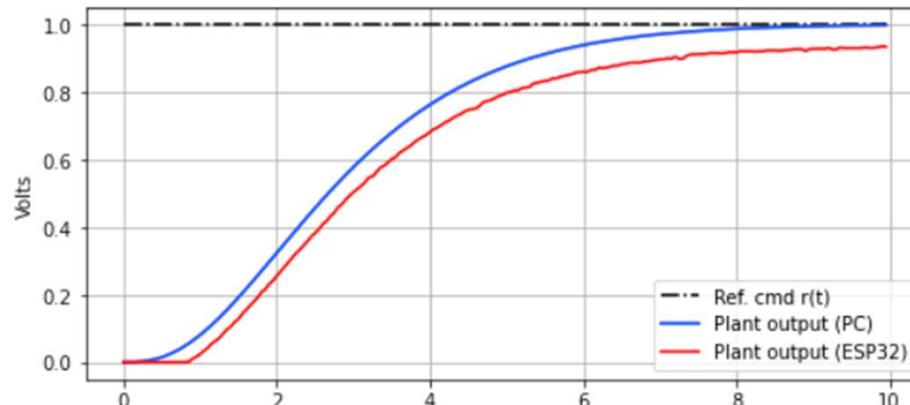
Open loop response



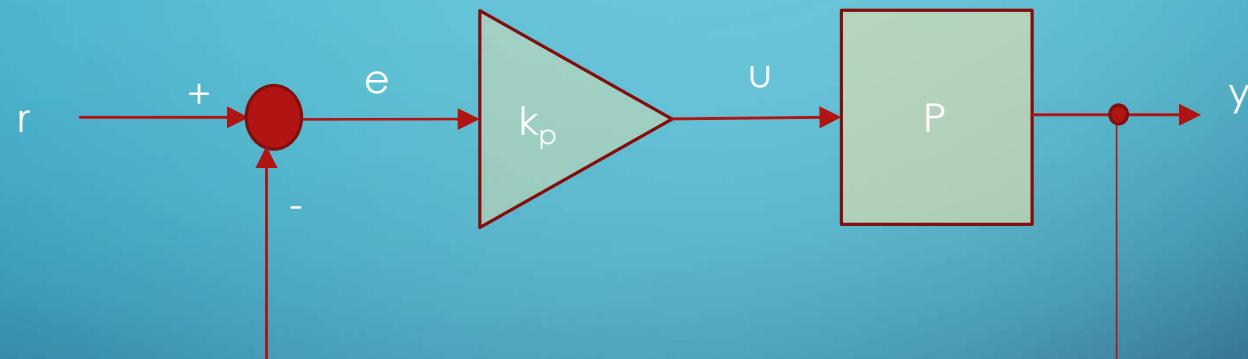
OPEN-LOOP RESPONSE FROM ELECTRONIC CIRCUITS

```
In [23]: fig, (ax1, ax2) = plt.subplots(2, figsize=(8,8))
fig.suptitle('Open loop response')
ax1.plot(tvec, rvec, 'k-.',tvec, y1, 'b-',tvec,yvec, 'r-')
ax1.set_ylabel('Volts')
ax1.legend(['Ref. cmd r(t)', 'Plant output (PC)', 'Plant output (ESP32)'])
ax1.grid(True)
ax2.plot(tvec,uvec, 'r-')
ax2.set_xlabel('time (s)')
ax2.set_ylabel('$u(t)$ Volts')
ax2.grid(True)
```

Open loop response



PROPORTIONAL CONTROL

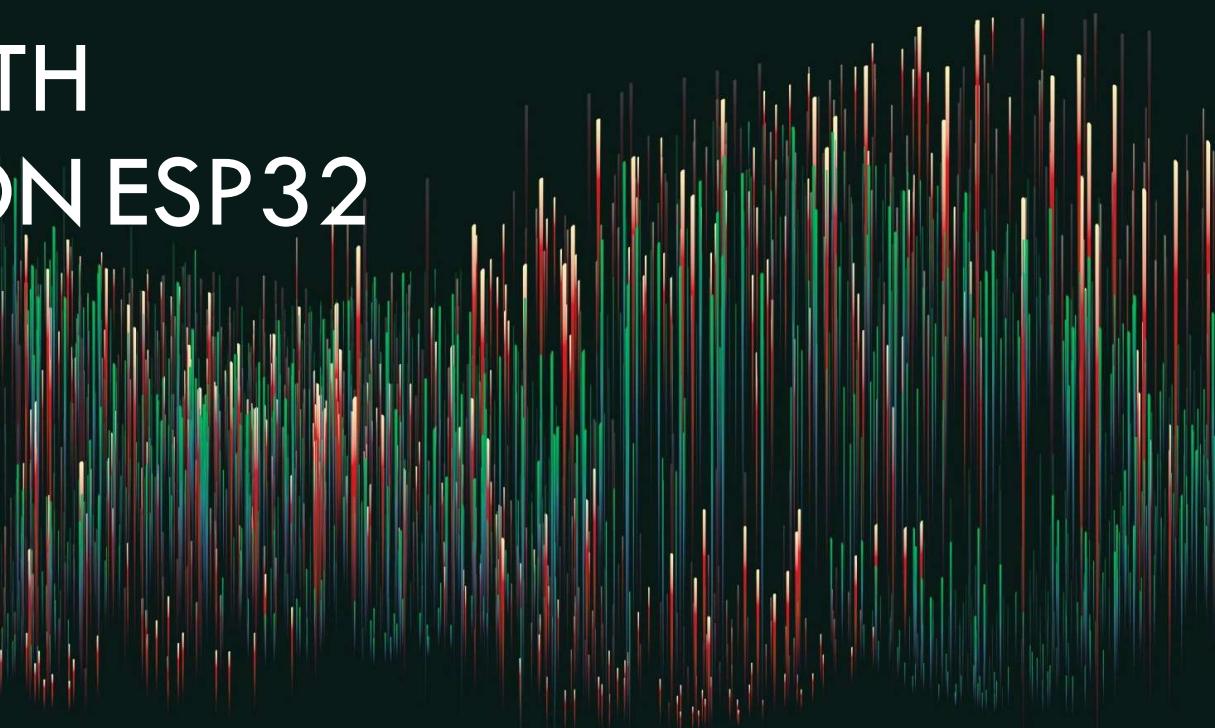


PID CONTROL WITH MICROPYTHON ON ESP32

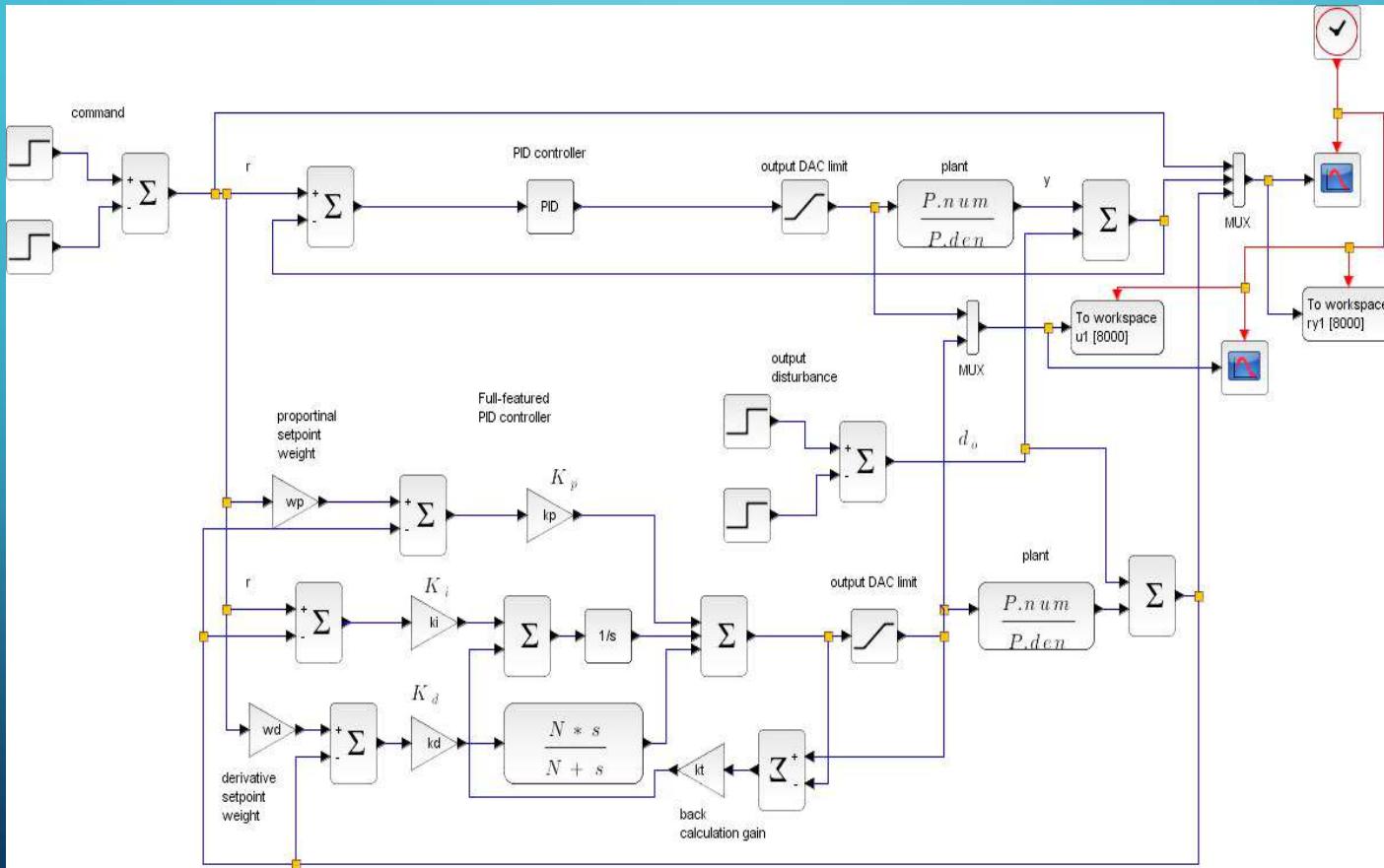
DR.VARODOM TOOCHINDA

DEPT. OF MECHANICAL ENGINEERING

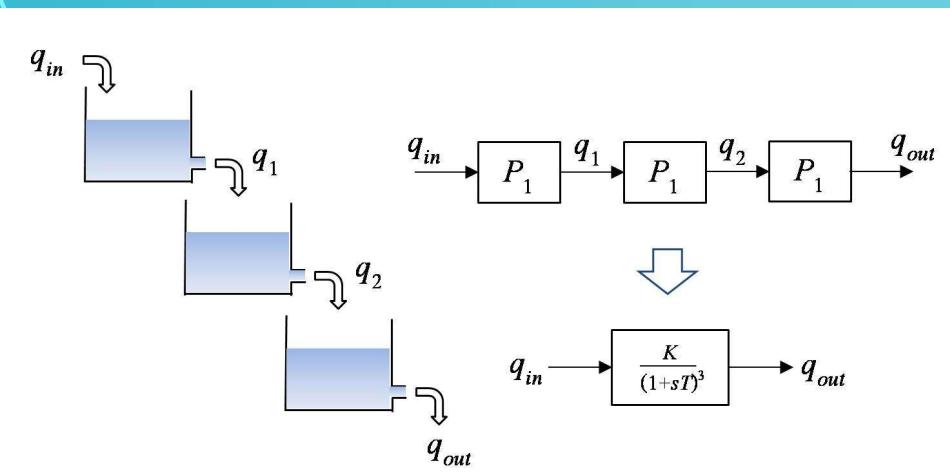
KASETSART UNIVERSITY



STRUCTURE OF FULL-FEATURED PID



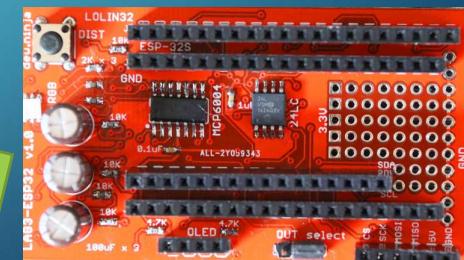
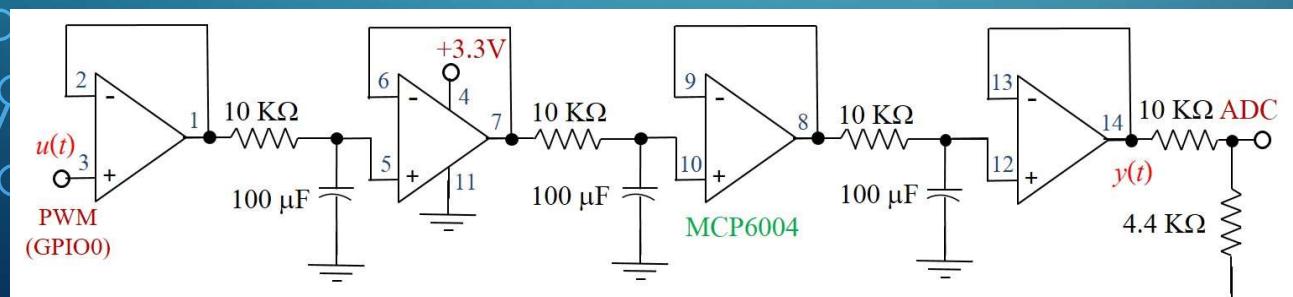
TEST ON LAG3-ESP32 (OR USING PLANT SIMULATION)



normalized transfer
function

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

Simulated by RC and op-amp circuit



CONVERT CONTINUOUS-TIME TO DISCRETE (1)

$$U = K_p e_p + K_i e_t + K_t l_{us} + \frac{K_d N s}{s+N} e_d(s)$$

Anti Windup

$$= U_{p0} + U_{i0} + U_{d0}$$

Bilinear Transform

$$s = \frac{2(z-1)}{T(z+1)}$$

$$U_{p0} = K_p e_p ; \quad e_p = w_p r - y \quad (1)$$

$$U_{i0} = \frac{K_i T(z+1)}{2(z-1)} e_t + \frac{K_t T(z+1)}{2(z-1)} l_{us} ; \quad e_t = y - y_s$$

$$e_t = U_{SAT} - U$$

$$U_{i0} = U_{i1} + b_i (e_0 + e_1) \quad (2)$$

$$U_{i1} = U_{i0} + b_i (e_0 + e_1) + b_t (l_{us0} + l_{us1})$$

$$U_{i0} - U_{i1} = b_i (e_0 + e_1) + b_t (l_{us0} + l_{us1})$$

$$U_d = \frac{K_d N (1-z^{-1})}{(1+0.5NT)+(0.5NT-1)z^{-1}} e_d ; \quad e_d = w_d r - y$$

$$(ad_1 + ad_2 z^{-1}) U_d = K_d N e_d - K_d N z^{-1} \rightarrow ad_1 U_{d0} + ad_2 U_{d1} = K_d N e_{d0} - K_d N e_{d1}$$

$$U_{d0} = -\frac{ad_1}{ad_1} U_{d1} + \frac{K_d N e_{d0} - K_d N e_{d1}}{ad_1} \quad (3)$$

$$U_{d0} = ad_1 U_{d1} + b_d (e_{d0} - e_{d1})$$

CONVERT CONTINUOUS TIME TO DISCRETE (2)

$$U_0 = U_{p0} + U_{i0} + U_{d0}$$

$$U_{p0} = k_p e_p \quad ; \quad e_p = w_p \tau - y$$

$$U_{i0} = U_{i1} + b_i (e_0 + e_1) + b_t (e_{us0} + e_{us1}) \quad ; \quad e_{us} = u_{sat} - u$$

$$U_{d0} = a_d U_{d1} + b_d (e_{d0} - e_{d1}) \quad ; \quad e_d = w_d \tau - y$$

$$b_i = 0.5 T k_i$$

$$b_t = 0.5 T K_T$$

$$a_{d1} = (1 + 0.5 NT) \quad b_d = \frac{k_d N}{a_{d1}}$$

$$a_{d2} = (0.5 NT - 1)$$

$$a_d = -\frac{a_{d2}}{a_{d1}}$$

FUNCTION TO COMPUTE COEFFICIENTS

```
def PID_update():
    global ad, bd, bi, bt
    bi = 0.5*T*ki
    bt = 0.5*T*kt
    ad1 = 1+0.5*N*T
    ad2 = 0.5*N*T - 1
    ad = -ad2/ad1
    bd = kd*N/ad1
    update_freeboard()
```

PID CONTROLLER FUNCTION

```
def PID_controller():
    global e1,e0,ed1,ed0, eus1,eus0,
    uil, ui0, ud1, ud0, u

    # state transfer
    e1 = e0
    ed1 = ed0
    eus1 = eus0

    uil = ui0
    ud1 = ud0
    # compute errors for each term
    e0 = r - y
    ep0 = wp*r - y # weighted
                  # proportional error
    ed0 = wd*r - y # weighted
                  # derivative error
```

```
        up0 = kp*ep0 # output of P term
        ui0 = uil +bi*(e0+e1) + bt*(eus0+eus1)
        # output of I term
        ud0 = ad*ud1 +bd*(ed0 - ed1) # output
        of D term
        u = up0 + ui0 + ud0
        u_lim = u
        if u > UMID:
            eus0 = UMID - u # compute error
                               # for back calculation term
            u_lim = UMID      # limit u to
                               # UMID
        elif u < -UMID:
            eus0 = -u - UMID # compute error
                               # for back calculation term
            u_lim = -UMID     # limit u to
                               #-UMID
        u_lim += UMID
        return u_lim
```

Call function in infinite loop (while True:)

```
if feedback:  
    ulim = PID_controller()  
else: # open-loop  
    u = r  
    ulim = u  
u_pwm = int(ulim*v2pwm)  
pwm_out.duty(u_pwm) # output via PWM pin 16  
u_dac = int(ulim*v2dac)  
dac_out.write(u_dac) # also output to DAC1
```

add commands in cmdInt()

```
:  
elif cmdstr.lower() == "kp":  
    if noparm==1:  
        print("Current kp =  
{ }".format(kp))  
    else:  
        kp = float(parmstr)  
        if kp > 100: # set maximum kp  
            kp = 100  
        elif kp < 0: # minimum kp  
            kp = 0  
        PID_update()  
  
elif cmdstr.lower() == "ki":  
:
```

copy and paste step response to Jupyter notebook

jupyter plot_pid_stepcompare Last Checkpoint: 16 minutes ago (unsaved changes) Logou

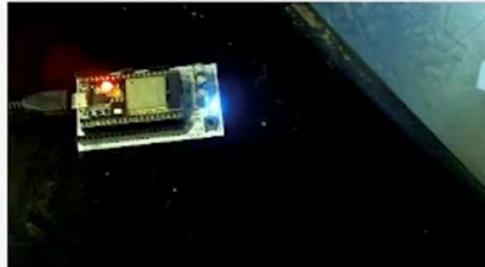
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3.7 (controlenv)

In [34]:

```
import numpy as np
import matplotlib.pyplot as plt
import control as ctrl
```

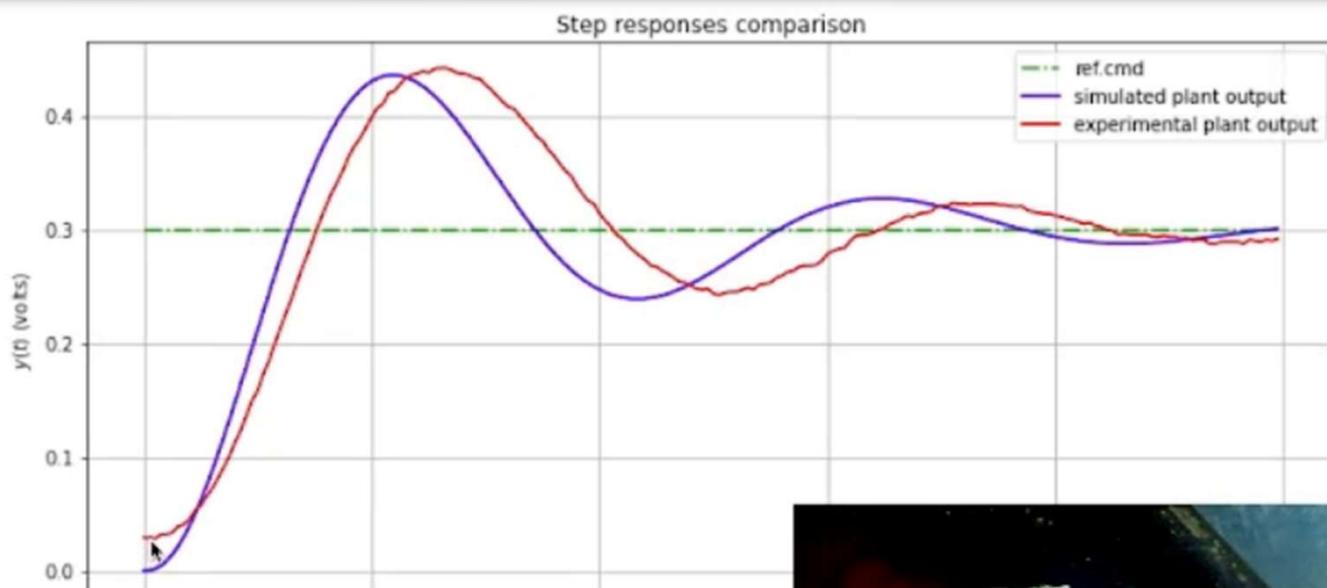
In [35]:

```
datamat = np.array([
[0,0.3,0.02901099,7.330201,3.3],
[0.05,0.3,0.02901099,2.318924,3.3],
[0.1,0.3,0.02813187,0.7021441,2.352144],
[0.15,0.3,0.03164835,0.04646528,1.696465],
[0.2,0.3,0.03164835,-0.04357779,1.606422],
[0.25,0.3,0.03340659,-0.1068736,1.543126],
[0.3,0.3,0.03868132,-0.2249923,1.425008],
[0.35,0.3,0.04043956,-0.1422761,1.507724],
[0.4,0.3,0.04483516,-0.1834713,1.466529],
[0.45,0.3,0.05274725,-0.3039473,1.346053],
[0.5,0.3,0.05714285,-0.2319831,1.418017],
[0.55,0.3,0.06505495,-0.3158633,1.334137],
[0.6,0.3,0.07208791,-0.3195103,1.33049],
[0.65,0.3,0.08087912,-0.3808489,1.269151],
[0.7,0.3,0.09054945,-0.4391118,1.210888],
[0.75,0.3,0.1002198,-0.4712322,1.178768],
[0.8,0.3,0.1125275,-0.5822057,1.067794],
[0.85,0.3,0.1230769,-0.584793,1.065207],
[0.9,0.3,0.1362637,-0.6909042,0.9590958],
[0.95,0.3,0.1494505,-0.7547222,0.8952778],
[1.0,0.3,0.1582417,-0.66117,0.98883],
[1.05,0.3,0.1731868,-0.8482413,0.8017587],
[1.1,0.3,0.1846154,-0.8326234,0.8173765],
[1.15,0.3,0.1995604,-0.9700761,0.6799239],
[1.2,0.3,0.2136264,-1.027014,0.6229861],
[1.25,0.3,0.2268132,-1.055505,0.5944954],
[1.3,0.3,0.24,-1.101719,0.5482807],
```



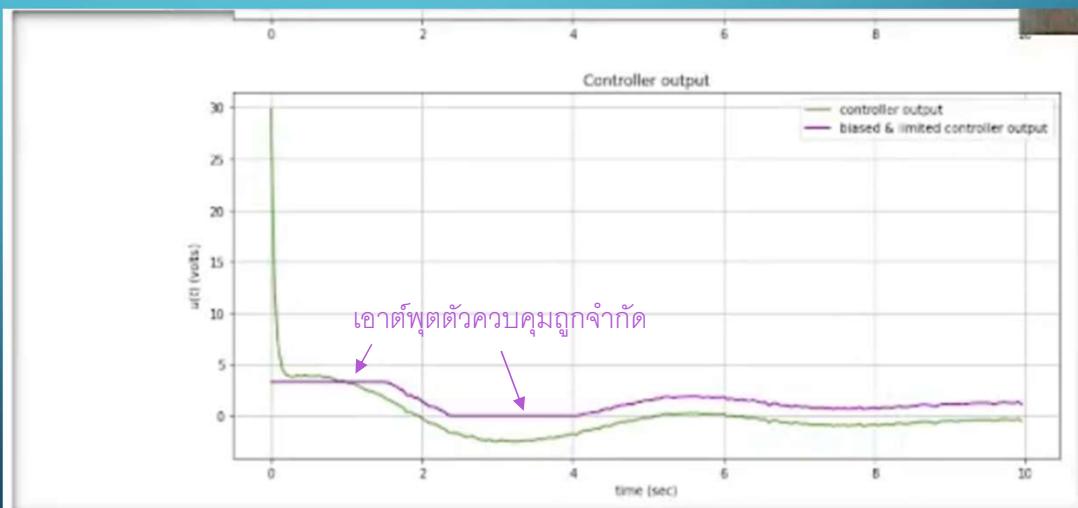
step response comparison

```
In [38]: # use the same parameters as in the ESP32
kp = 4.8
ki = 2.74
kd = 2.1
pid_compare(kp,ki,kd,datamat,scale=0.3)
```



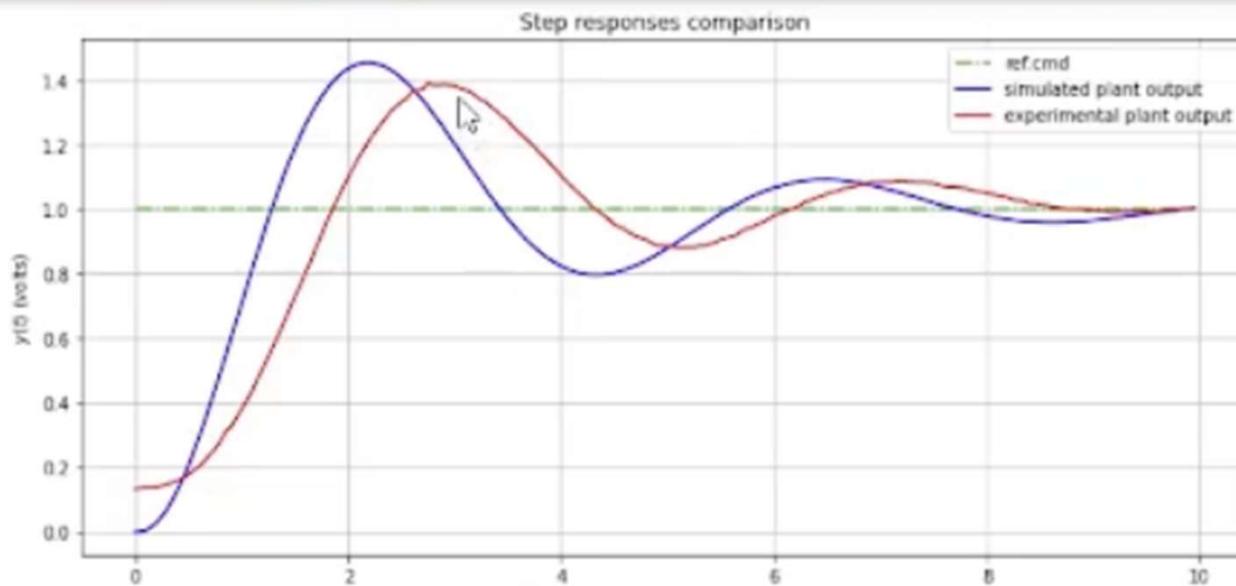
saturation
effect when
setting
ref.cmd = 1.0

```
In [6]: # use the same parameters as in the ESP32
kp = 4.8
ki = 2.74
kd = 2.1
pid_compare(kp,ki,kd,datamat,scale=1)
```



compensate with back calculation anti-windup ($kt=0.5$)

```
In [13]: # use the same parameters as in the ESP32  
kp = 4.8  
ki = 2.74  
kd = 2.1  
pid_compare(kp,ki,kd,datamat,scale=1)
```



LAG3 SIMULATION ON WOKWI

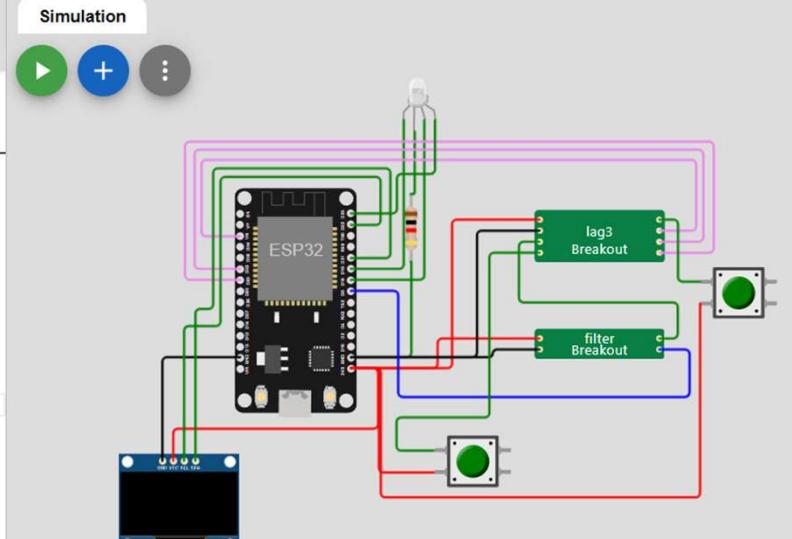
WOKWI SAVE SHARE lag3_iot_controllers Docs

`main.py` `diagram.json` `lag3.chip.json` `lag3.chip.c` `filter.chip.json` `filter.chip.c`

`ssd1306.py`

```
78     def init_client():
79         ...
80
81
82     def sub_cb(topic, msg):
83         global private_msg, cmdtime_prev
84         print((topic, msg))
85         if topic == b'@msg/cmd':
86             rcvdstrs = str(msg).split("") # get rid of b'
87             rcvdstr = rcvdstrs[1]
88             cmdInt(rcvdstr)
89             # this delay is needed for nodered implementation
90             cmdtime_current = time.ticks_ms()
91             delta_cmdtime = cmdtime_current - cmdtime_prev
92             if delta_cmdtime > CMD_DELAY:
93                 cmdtime_prev = cmdtime_current
94             cmdInt(rcvdstr)
95             # added in iot_controller3.py to retrieve shadow data
96             elif topic == b'@private/shadow/data/get/response':
97                 print(msg)
98                 private_msg = str(msg)
99
100
101
102     online = False # change to True to connect NETPIE
103     if online:
104         wifi_connect() # connect to WiFi network
105         init_client()
106
107     button = Pin(0,Pin.IN, Pin.PULL_UP) # on-board switch
108     button_state = True
109
110
111     def update_dashboard():
112         updatestr = ("{}","{}","{}","{}","{}","{}","{}","{}","{}","{}","{}","{}","{}","{}","{}","{}")
113         .format(T, plantsim, datasize, capture, feedback,
114               controller.autotune,lsid,kp,ki,kd,kt,wp,wd,N)
```

Simulation



clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0000, len:1656
load:0x400780
ho 0 tail 12
load:0x40080400,len:3712
entry 0x4008064c
0.05,0,200,0,1,0,0,0,4.8,2.74,2.1,0,1,1,50
Enter command :
<https://wokwi.com/projects/406624428008336385>

SIMULATION EXAMPLE

W dcm_1joint - Wokwi ESP32, ST x + wokwi.com/projects/374740347412079617

ESP32 Gmail YouTube Maps Machine Learning coursera misc Android democracy finance books deep learning courses python julia IoT Mac M1 fibo ros Arduino book formatting

WOKWI SAVE SHARE Docs 🔥

main.py diagram.json dcm1.chip.json dcm1.chip.c filter.chip.json
filter.chip.c ssd1306.py

```
430 u_pwm = int(ulim*v2pwm) # v2pwm = int(PWMMAX/3.3)
431 pwm_out.duty(u_pwm) # send output via PWM pin 16
432 # pwm_out.duty(PWMMAX)
433 #u_dac = int(ulim*v2dac) # v2dac = int(DACMAX/3.3)
434 #dac_out.write(u_dac) # also send output to DAC1
435 # --- remove these lines if SSD1306 is not available ---
436 oled_tcurrent = time.ticks_ms()
437 if (oled_tcurrent - oled_tprev)>300: # display every 300 ms
438     oled_tprev = oled_tcurrent
439     oled_showryu(oled,r,y,ulim)
440     pixels_display() # display angle on NeoPixels
441 #
442
443
444 if capture_flag:
445     if controller==2: # state-feedback controller
446         print("[0,0,0,0,0,0,0,0].format(round(t_data,2),r,y,ulim,u))
447     else:
448         print("[0,0,0,0,0,0].format(round(t_data,2),r,y,ulim,u))
449     t_data+=T
450     data_idx+=1
451     if data_idx==datasize:
452         data_idx = 0
453         t_data = 0
454         capture_flag = False # stop capture data
455         print("]")
456         prompt=True # prompt on
457         if lsid:
458             print("LSID ends.")
459             lsid = False # LSID mode ends
460
461 # helper function for cmdInt()
```

Simulation 00:49.800 100%

The simulation interface shows a circuit diagram with an ESP32 microcontroller at the center. It is connected to a circular potentiometer, a digital-to-analog converter (DAC), a red LED, a breadboard, and a computer. A terminal window at the bottom displays code execution results and a command-line interface.

load:0x40078000,len:13284
ho 0 tail 12 room 4
load:0x40080400,len:3712
entry 0x4008064c

Enter command : <https://wokwi.com/projects/406624983024976897>

r=-1
Ref. comd is now from VR

Enter command :

Thank You

