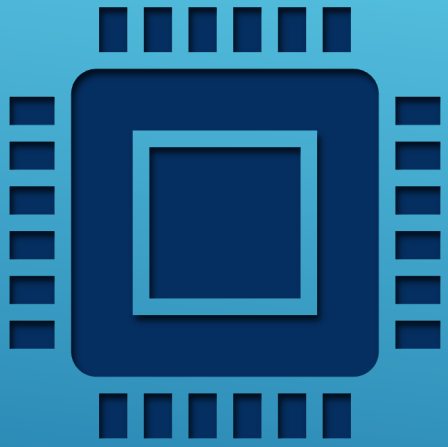
2024 Semester 1

EE 01205479
INTERNET OF THINGS FOR ELECTRICAL ENGINEERING

Dr. Varodom Toochinda

RDiPT, ME, Kasetsart University

COURSE ESSENTIALS

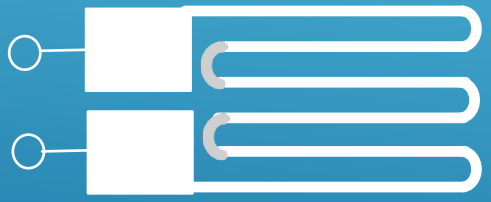


- Emphasize embedded systems and Internet of Things (IoT)
- Basic electronic/electrical circuits or analog interfacing
- Hardware (ESP8266/ESP32) and software (Arduino IDE, Scilab, Python+Jupyter Notebook, Micropython)
- IoT cloud platform (NETPIE)
- Continuous and discrete LTI systems
- Digital signal processing
- Real-time multitasking system design
- Advanced Topics

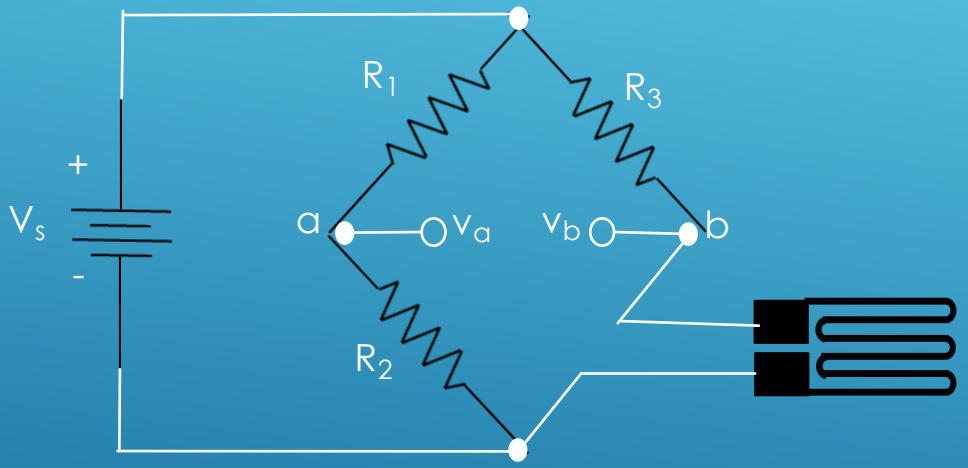
BASIC CIRCUITS & SIGNAL CONDITONING

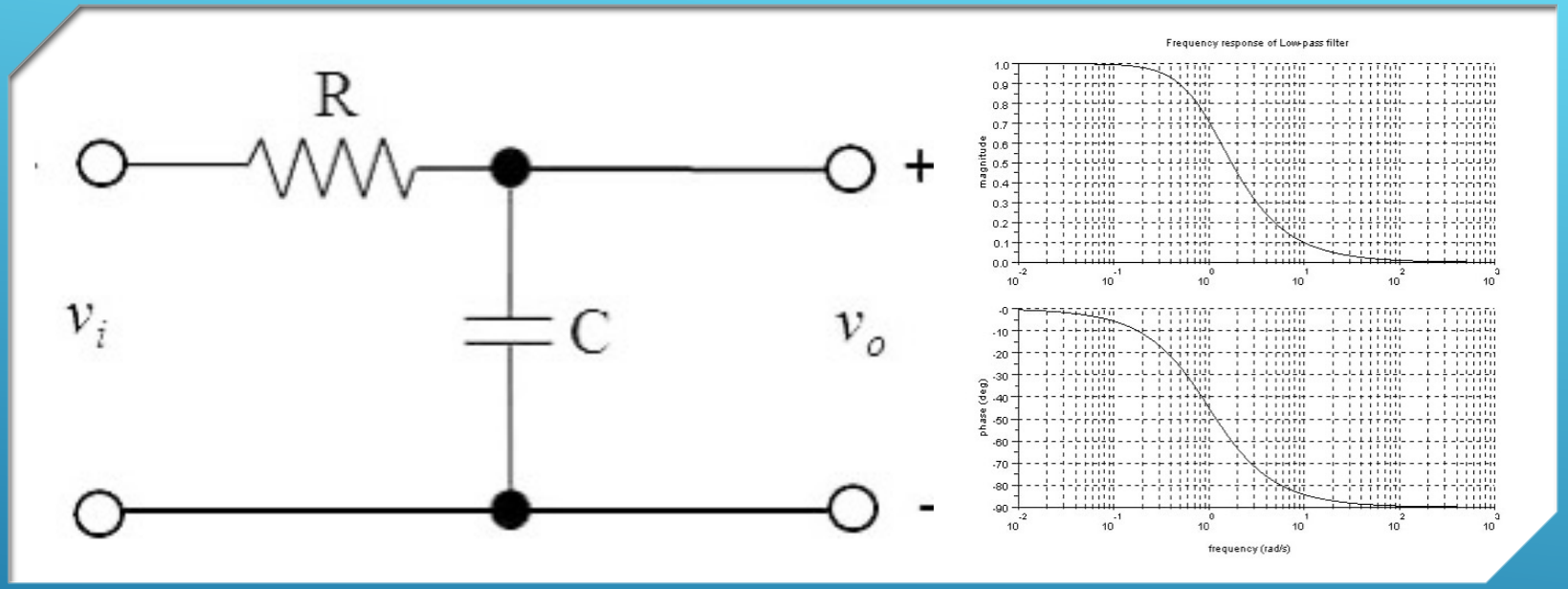
In order to develop a successful IoT device, we need to know how to design hardware and interfacing circuit.

DC CIRCUIT EXAMPLE : RESISTIVE NETWORK

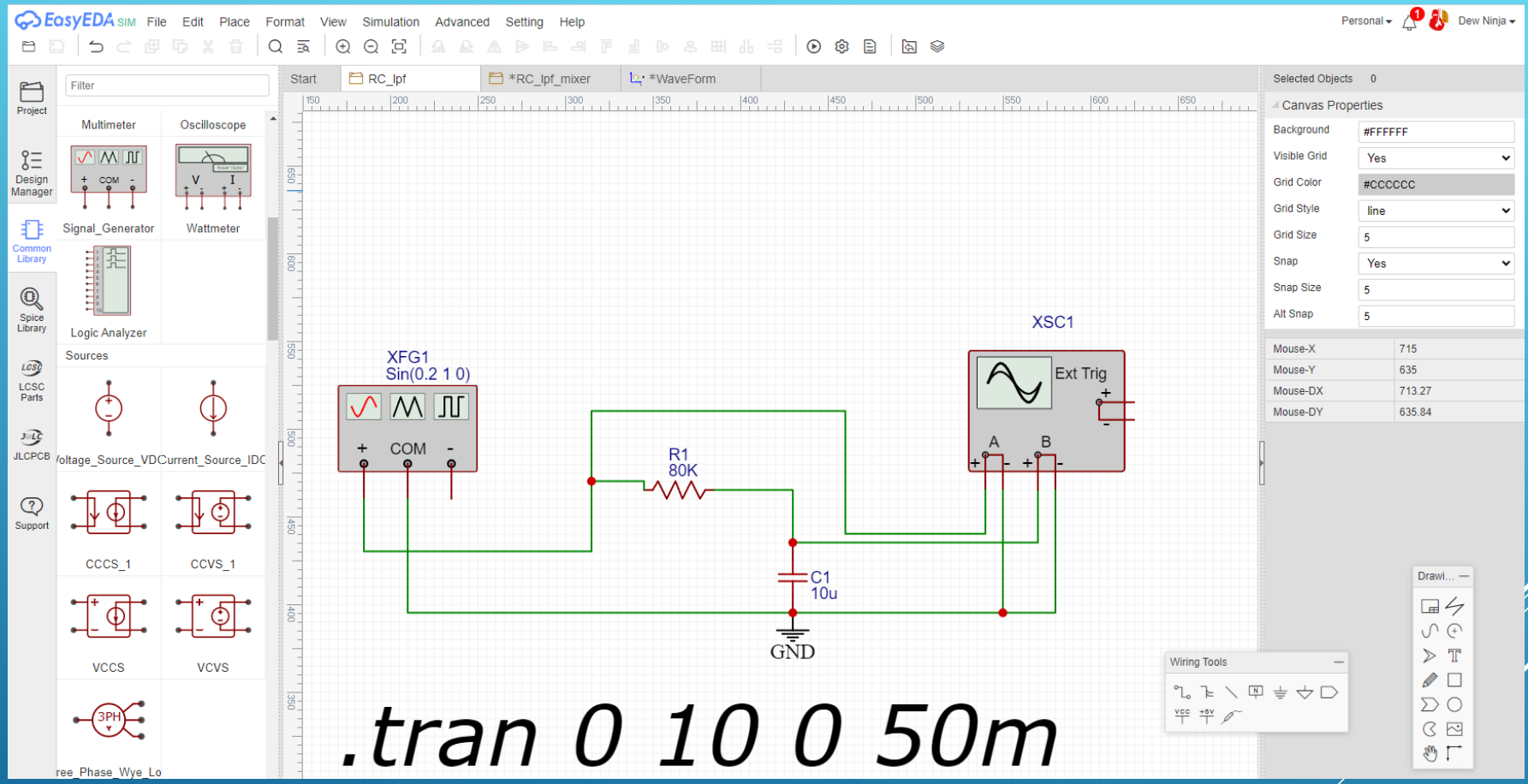


Stain gauge

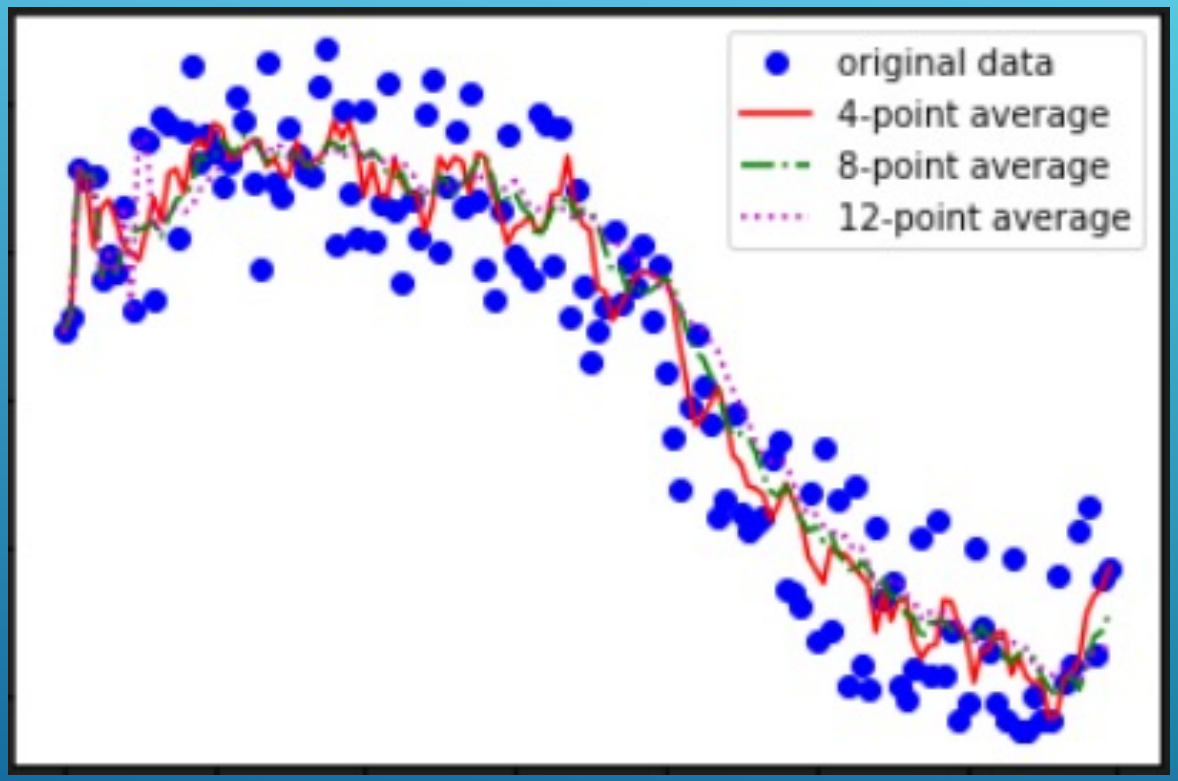




AC CIRCUIT EXAMPLE : PASSIVE LPF

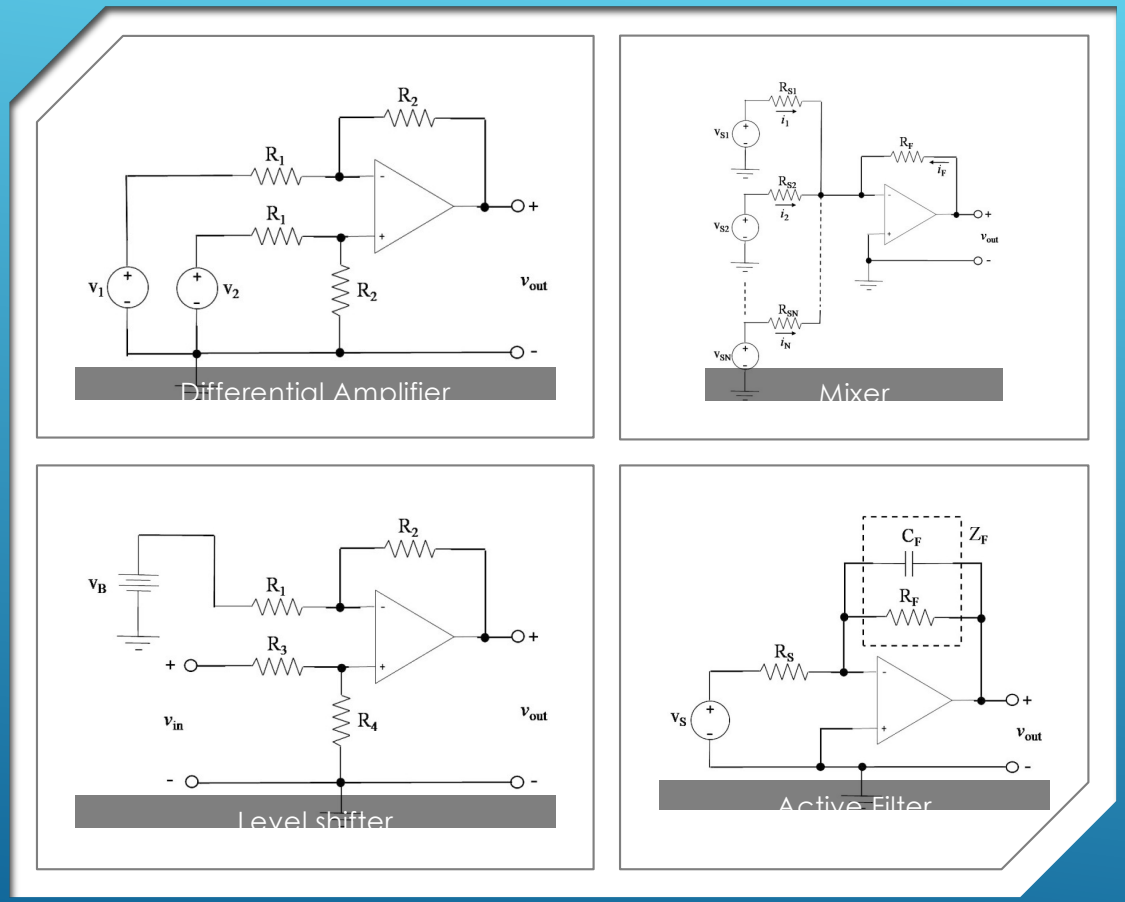


LPF SIMULATION ON EASYEDA

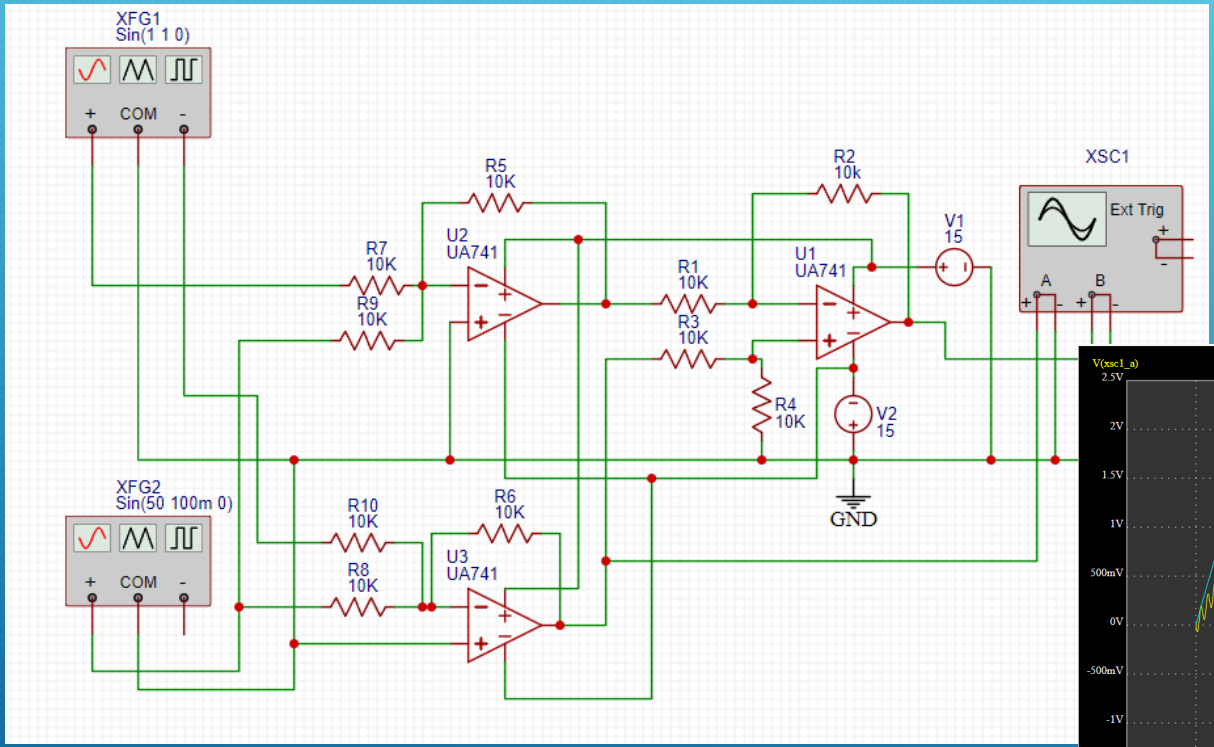


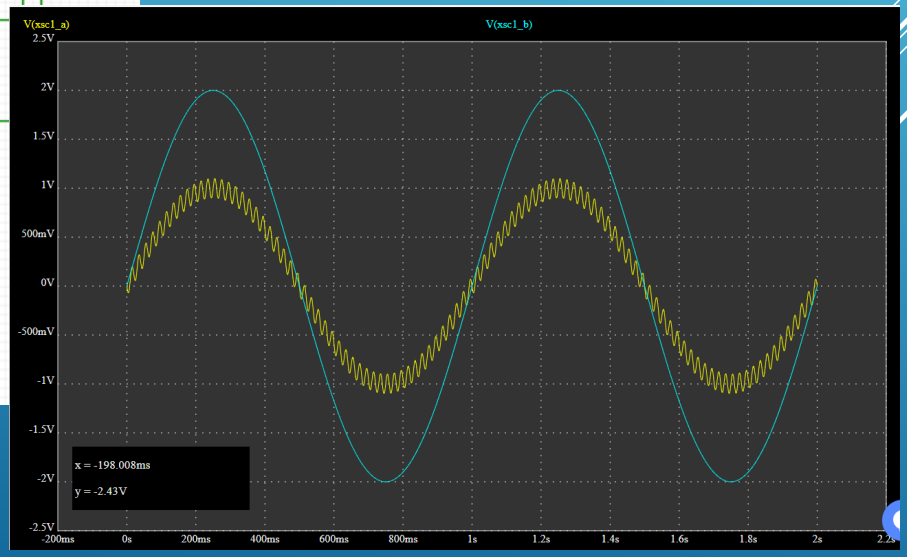
NOISE FILTERING BY AVERAGING

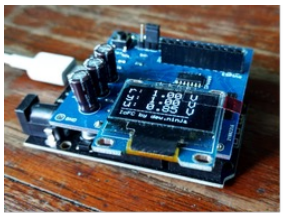
OPERATIONAL AMPLIFIER EXAMPLE

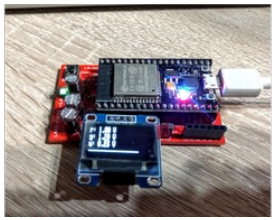


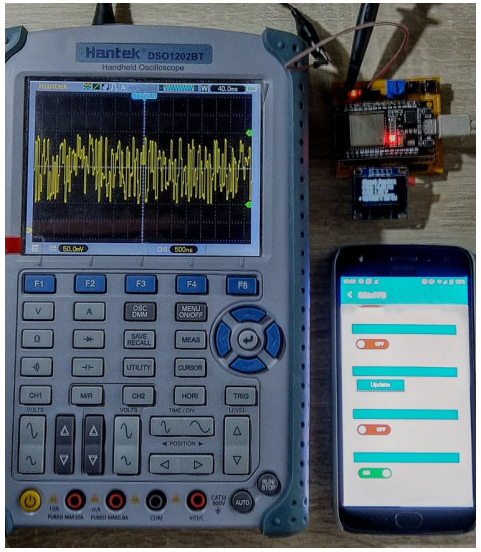
Differential amp simulation



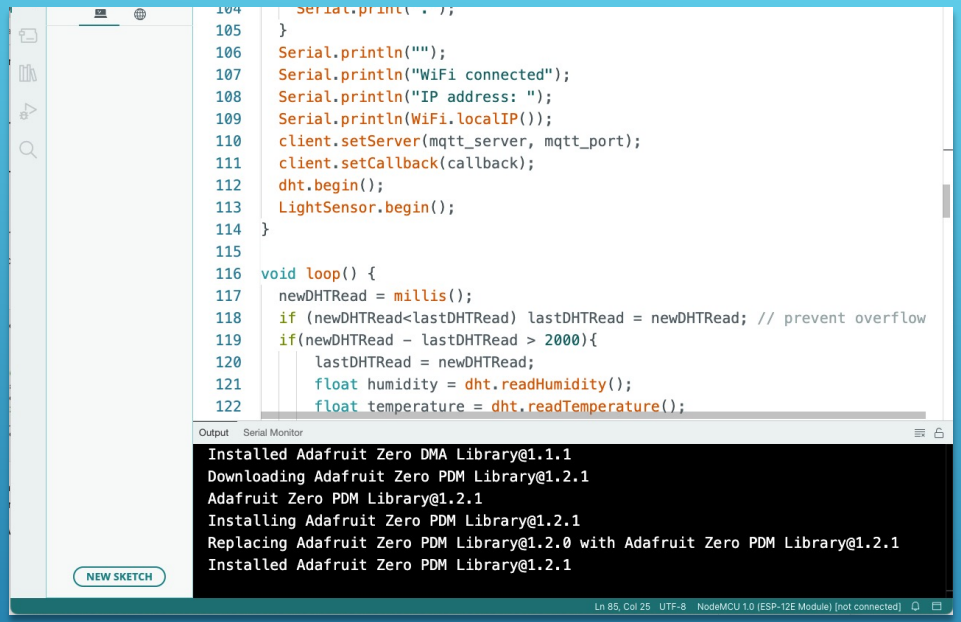






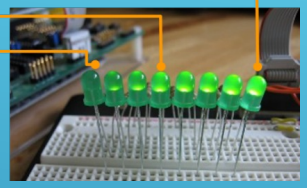


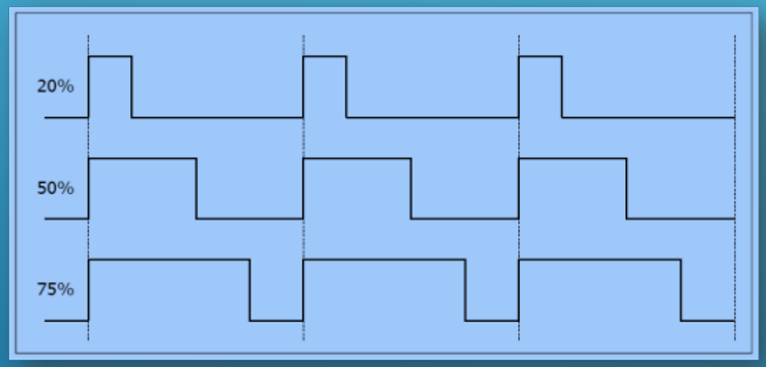
IOT-BASED HARDWARE

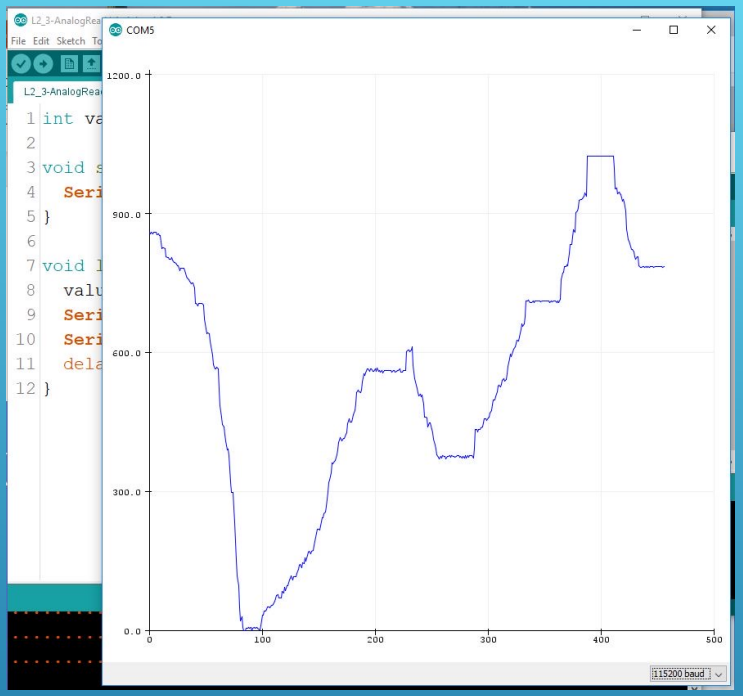


C++ PROGRAMMING ON ESP8266/ESP32

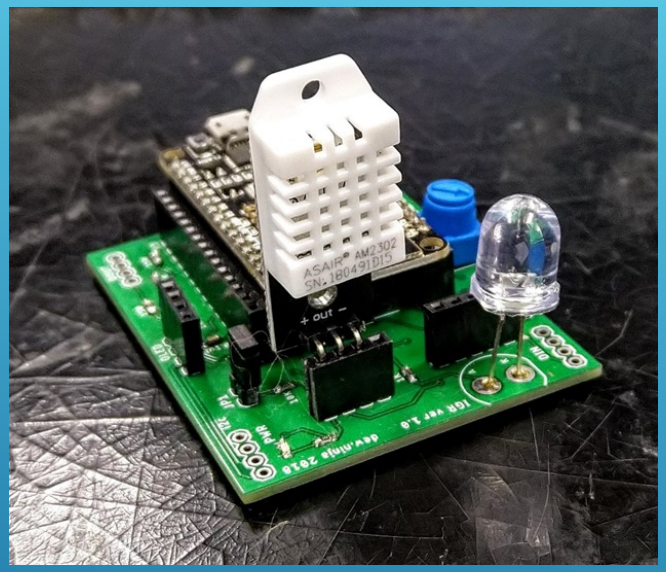
```
analogWrite(D5,1023);
analogWrite(D5,512);
analogWrite(D5,0);
```

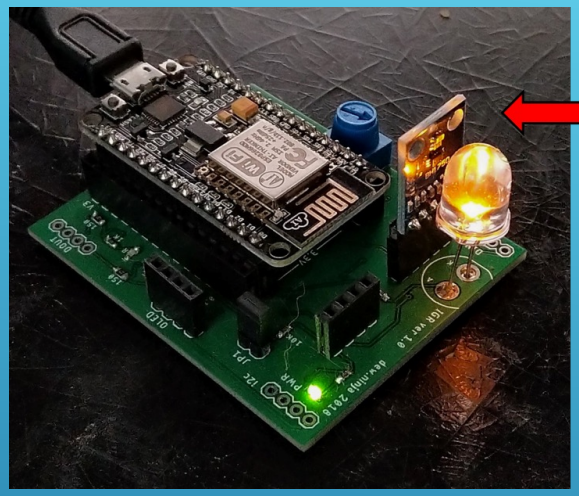


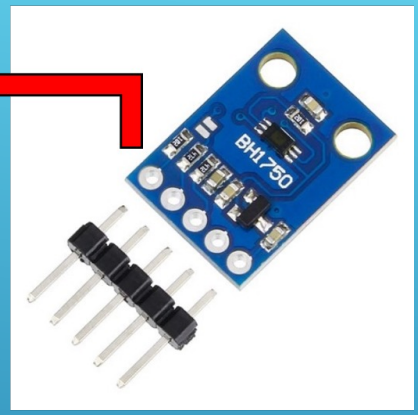




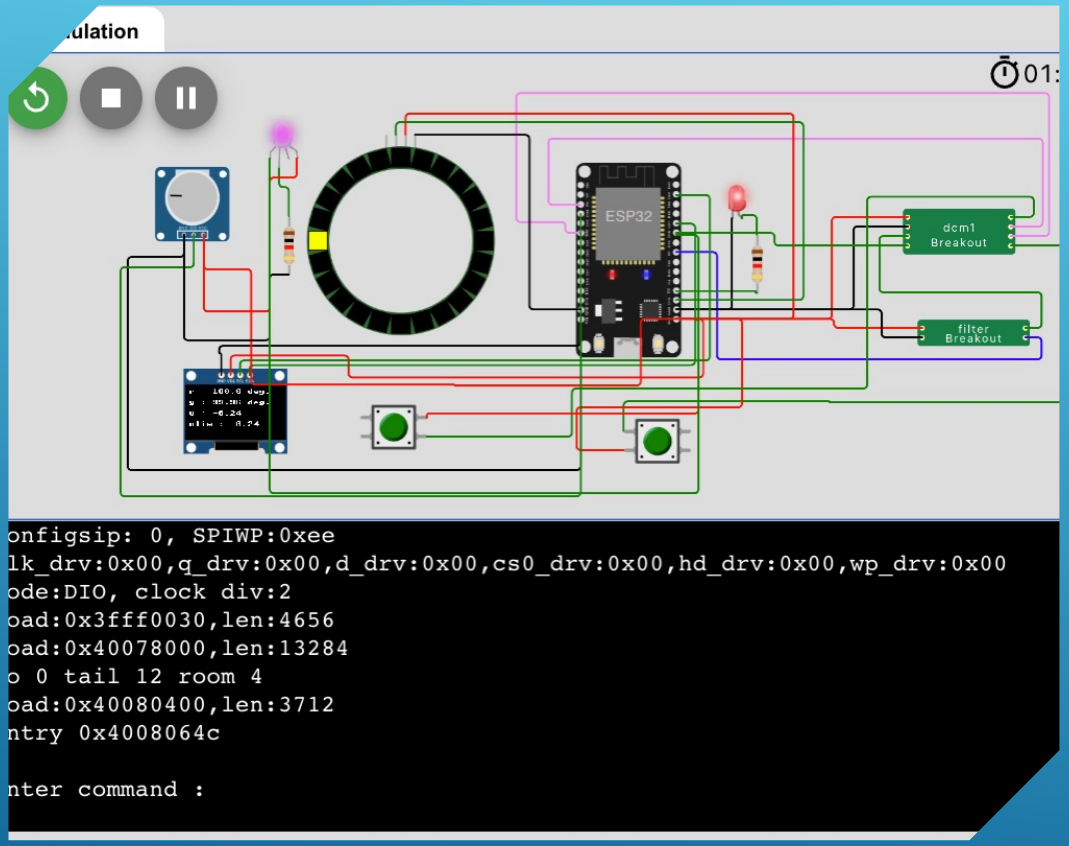
BASIC I/O'S





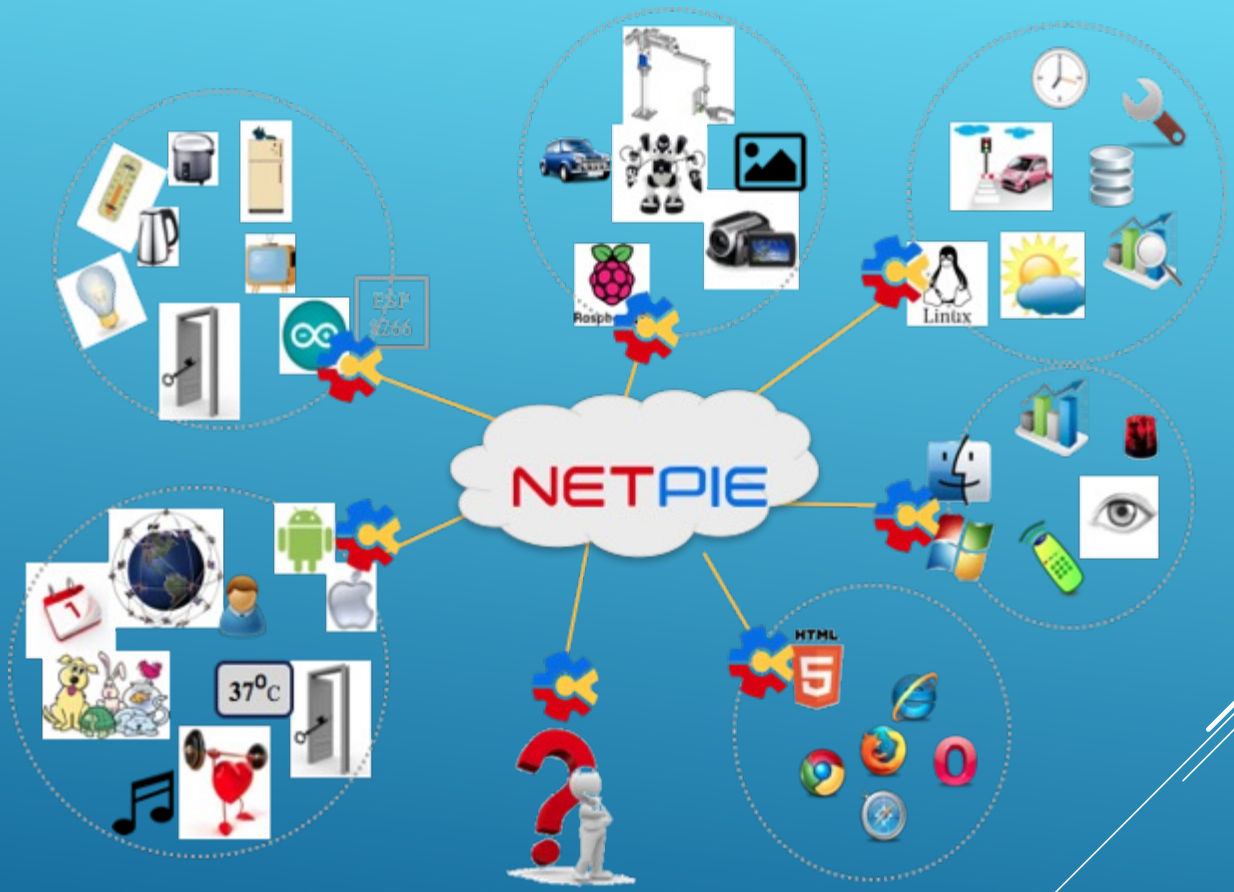


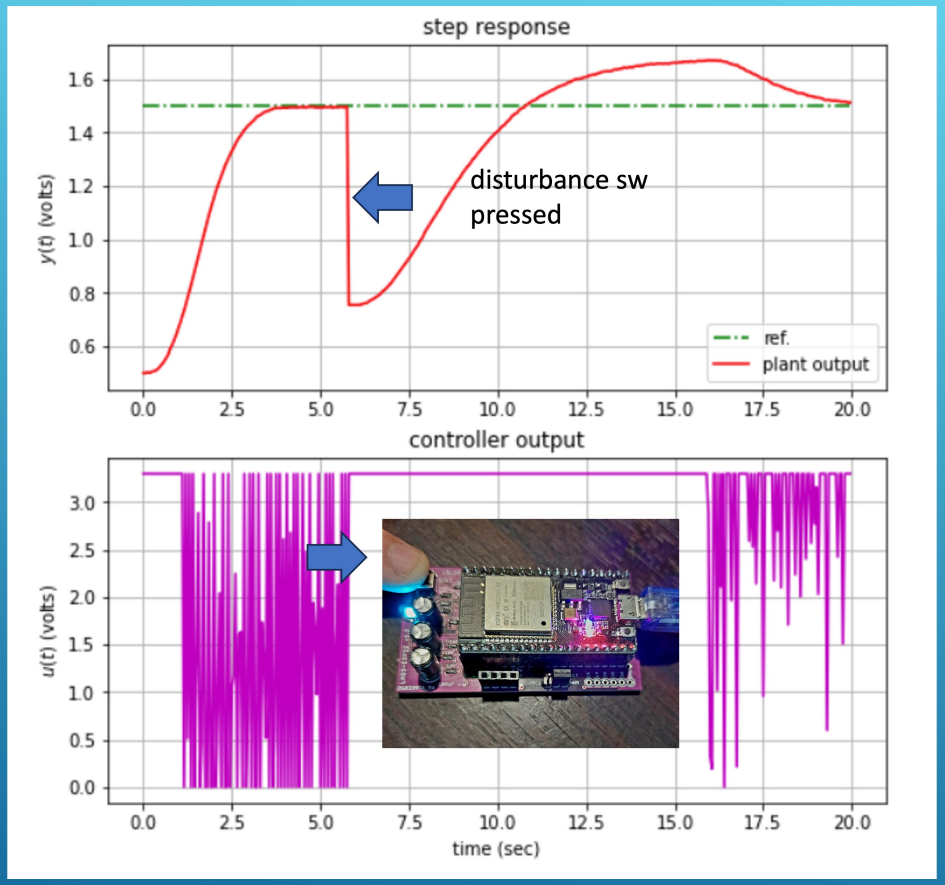
SENSORS INTERFACE

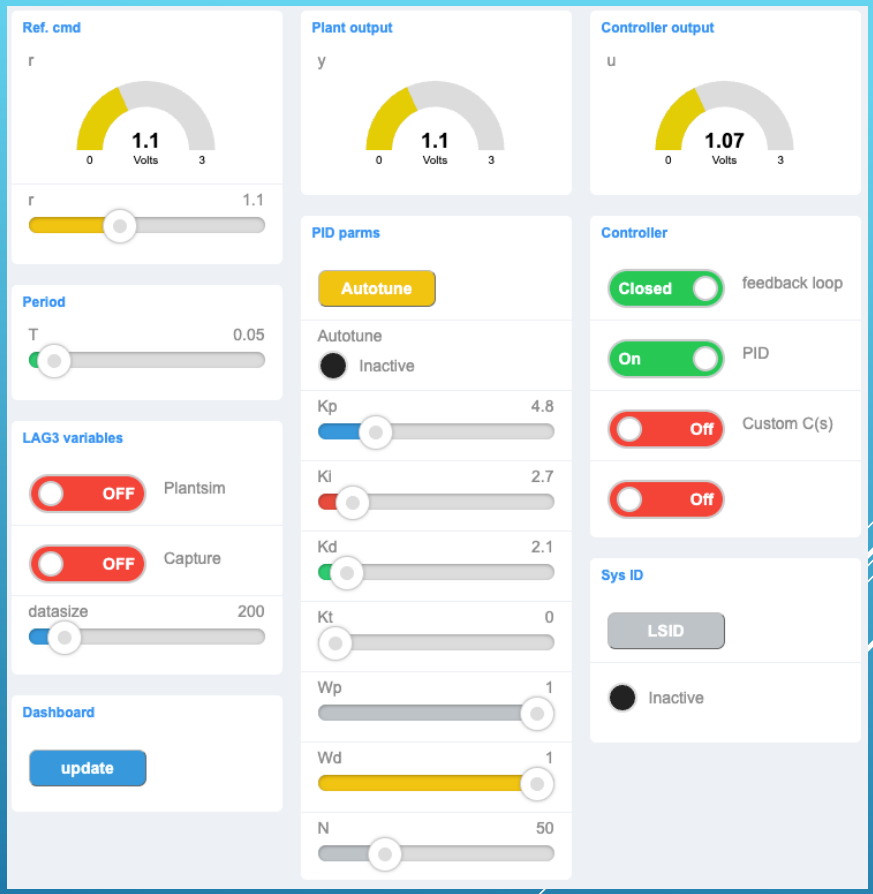


EMBEDDED SYSTEM SIMULATION

NETPIE IOT

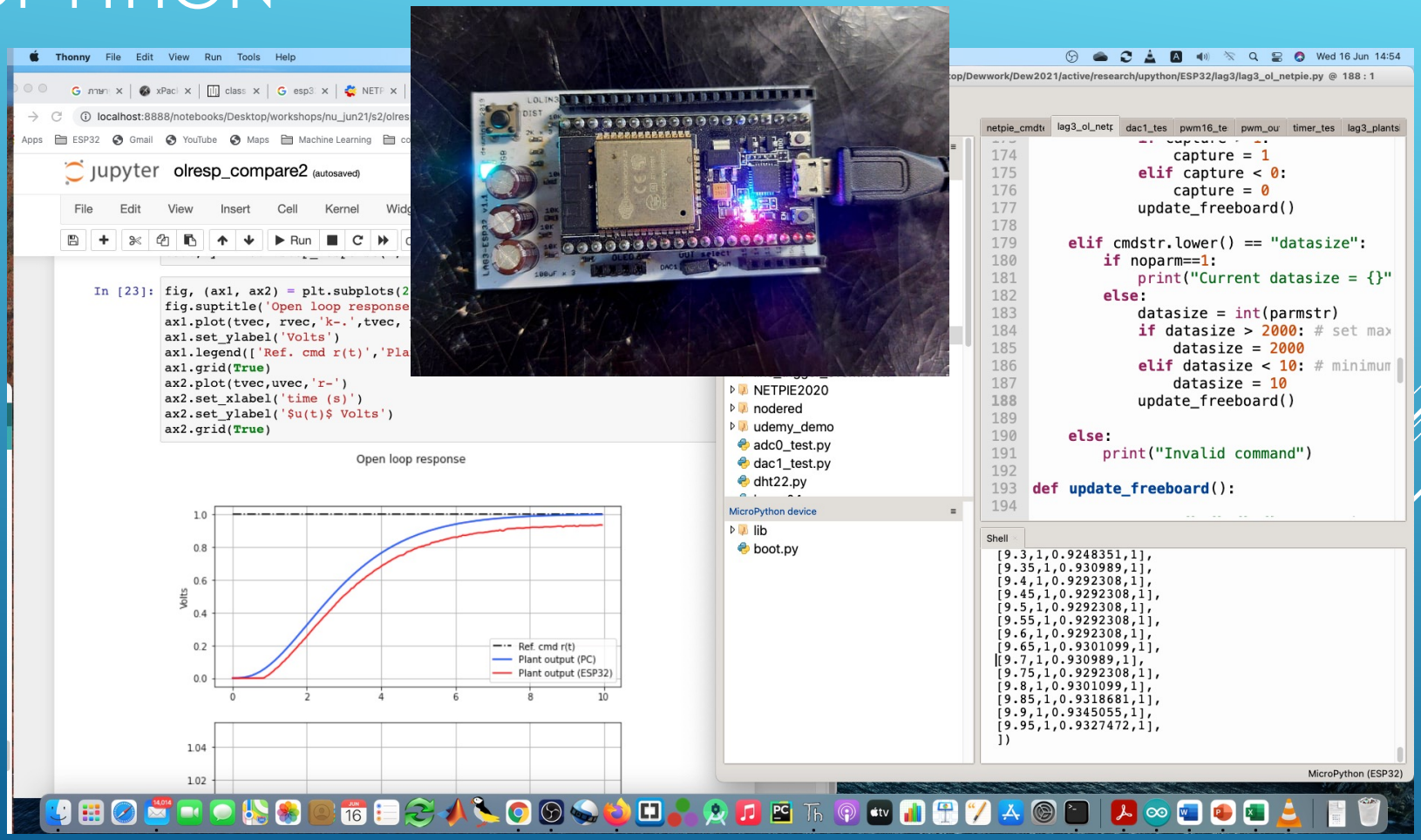




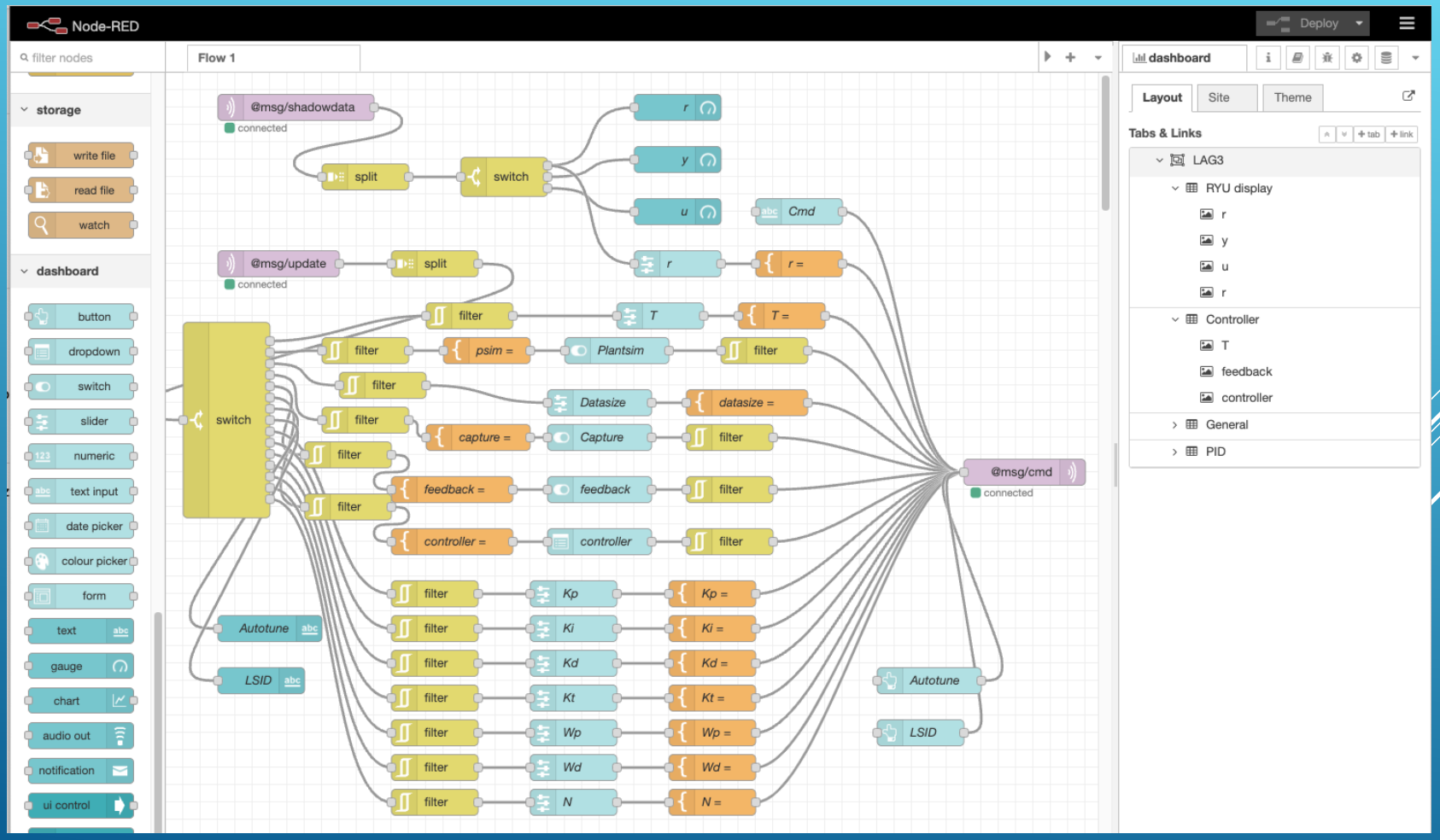


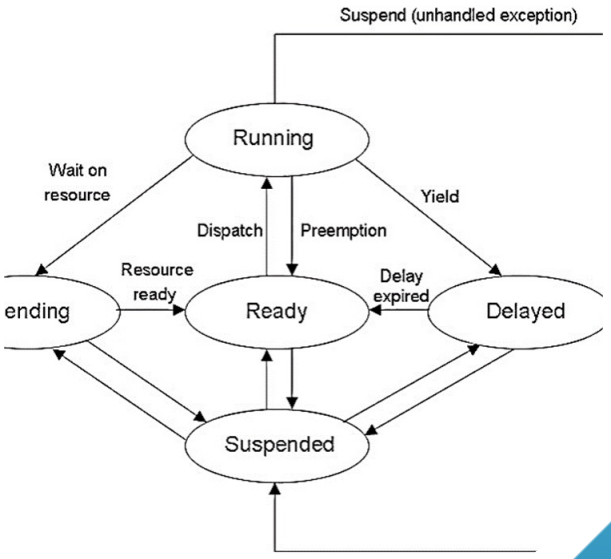
IoT for industrial applications

MICROPYTHON



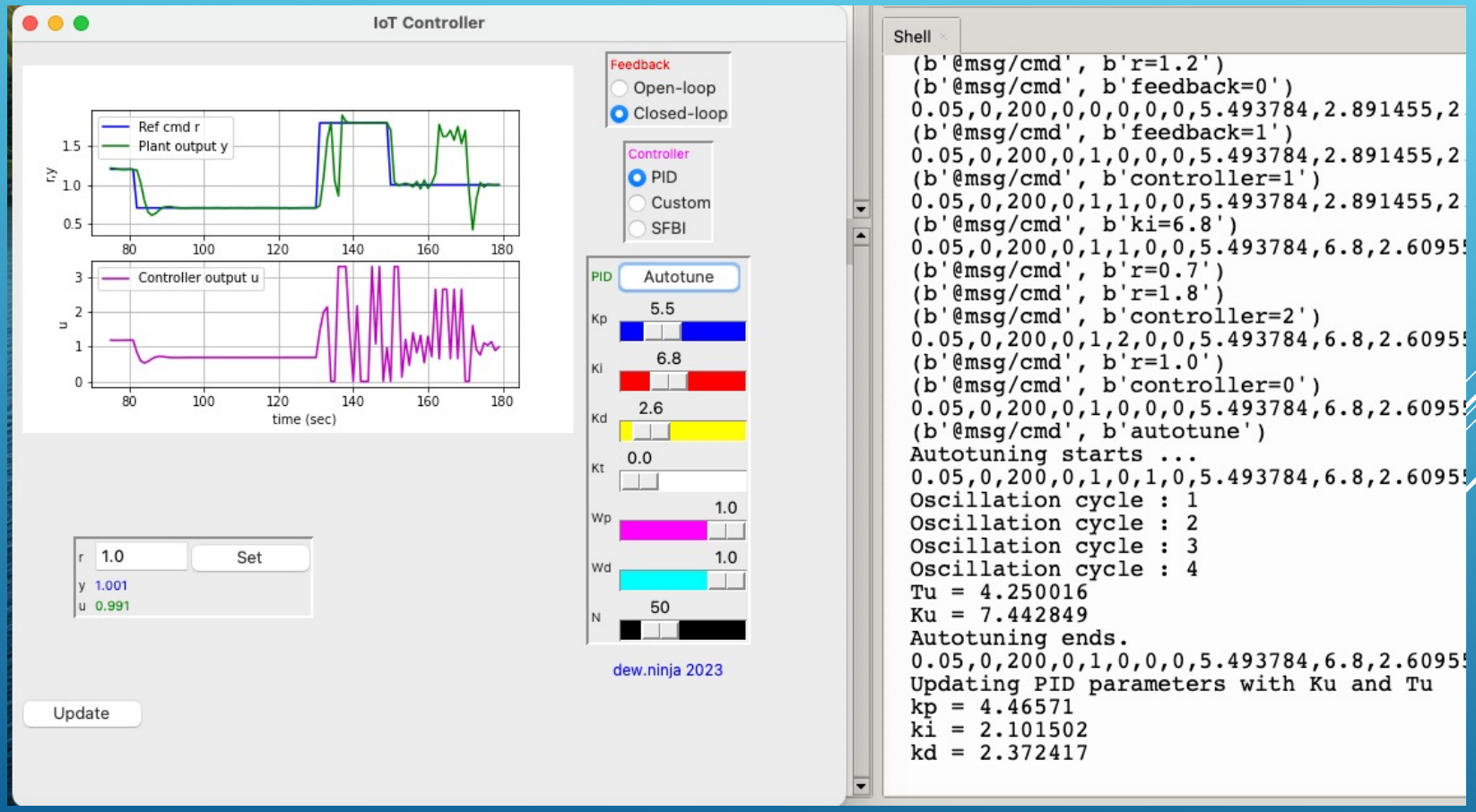
NODE-RED


Node-RED
Deploy
filter nodes
Flow 1
dashboard
storage
write file
read file
watch
dashboard
button
dropdown
switch
slider
numeric
text input
date picker
colour picker
form
text
gauge
chart
audio out
notification
ui control
Layout
Site
Theme
Tabs & Links
LAG3
RYU display
r
y
u
r
Controller
T
feedback
controller
General
PID
@msg/shadowdata
connected
split
switch
r
y
u
r
Cmd
r =
@msg/update
connected
split
filter
T
T =
switch
filter
psim =
Plantsim
filter
filter
Datasize
datasize =
filter
capture =
Capture
filter
filter
feedback =
feedback
filter
filter
controller =
controller
filter
@msg/cmd
connected
filter
Kp
Kp =
filter
Ki
Ki =
Autotune
filter
Kd
Kd =
LSID
filter
Kt
Kt =
filter
Wp
Wp =
Autotune
filter
Wd
Wd =
LSID
filter
N
N =



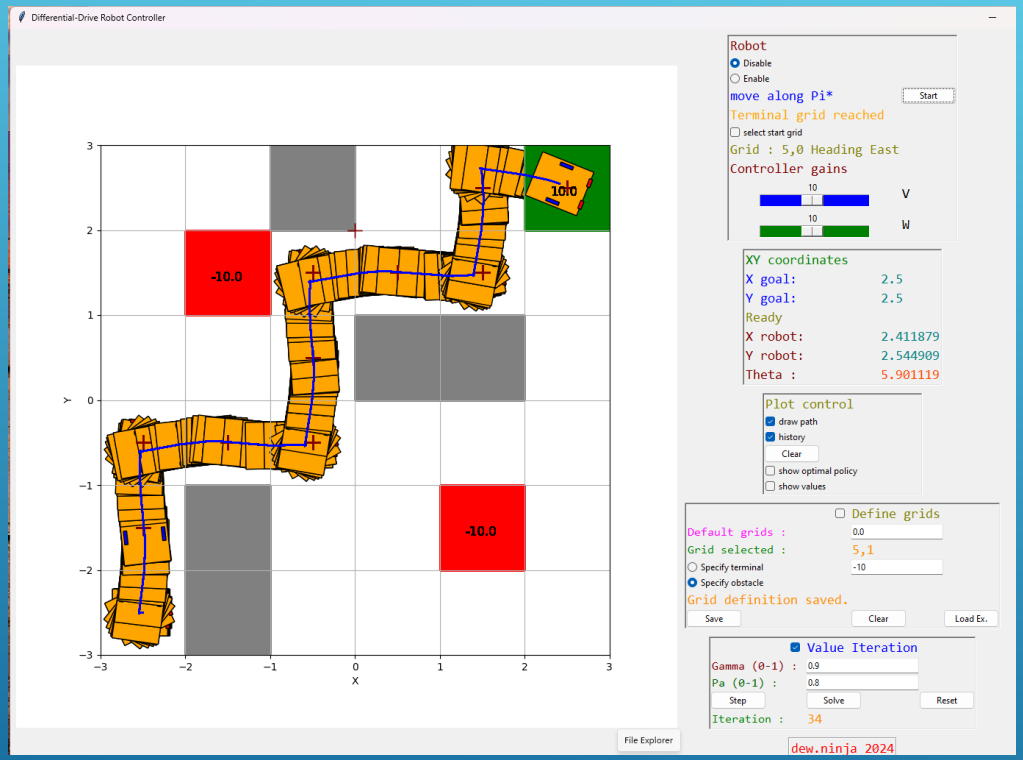
REAL-TIME MULTITASKING WITH FREERTOS

OOP (OBJECT-ORIENTED PROGRAMMING

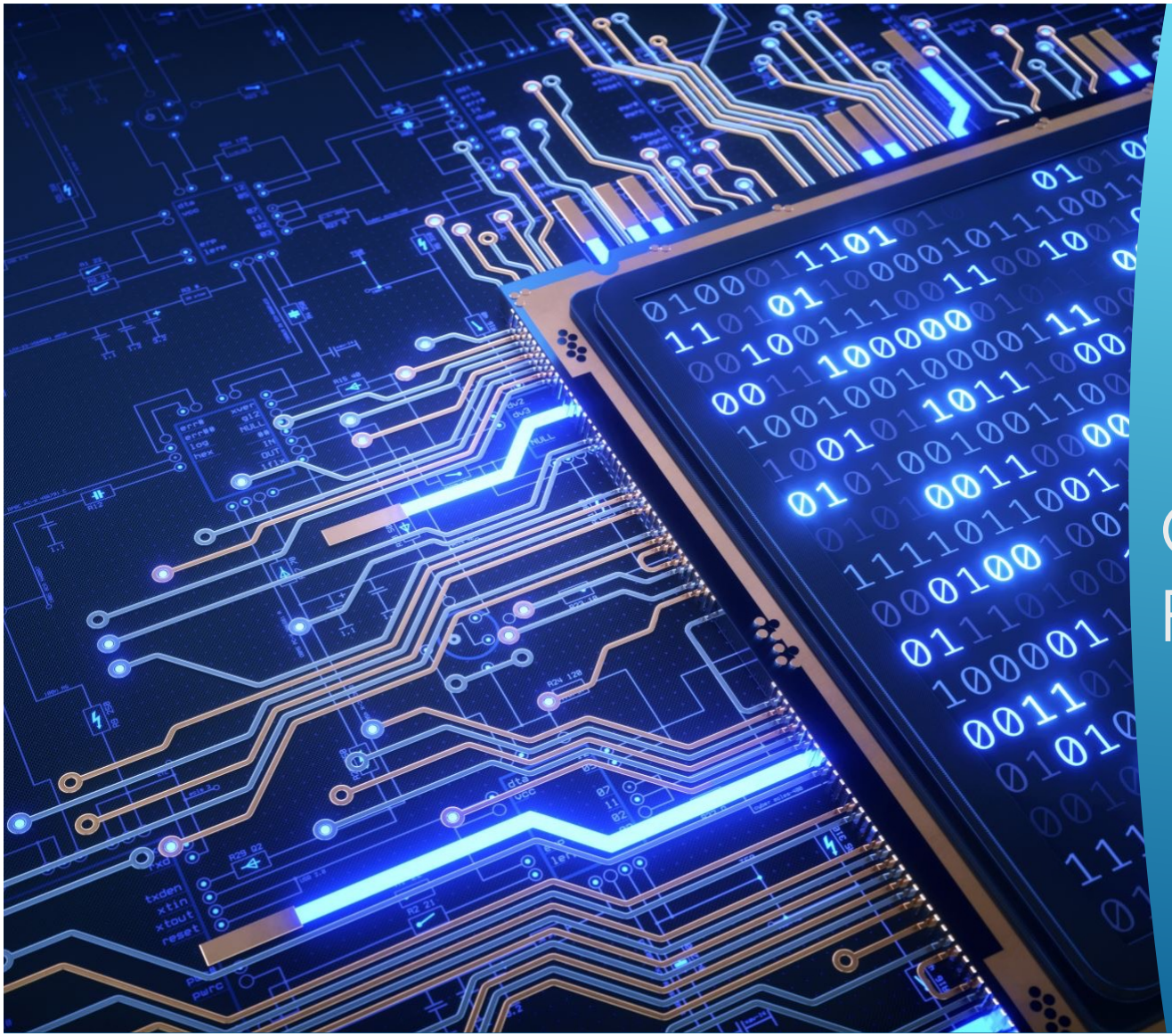


ADVANCED TOPICS & CASE STUDY

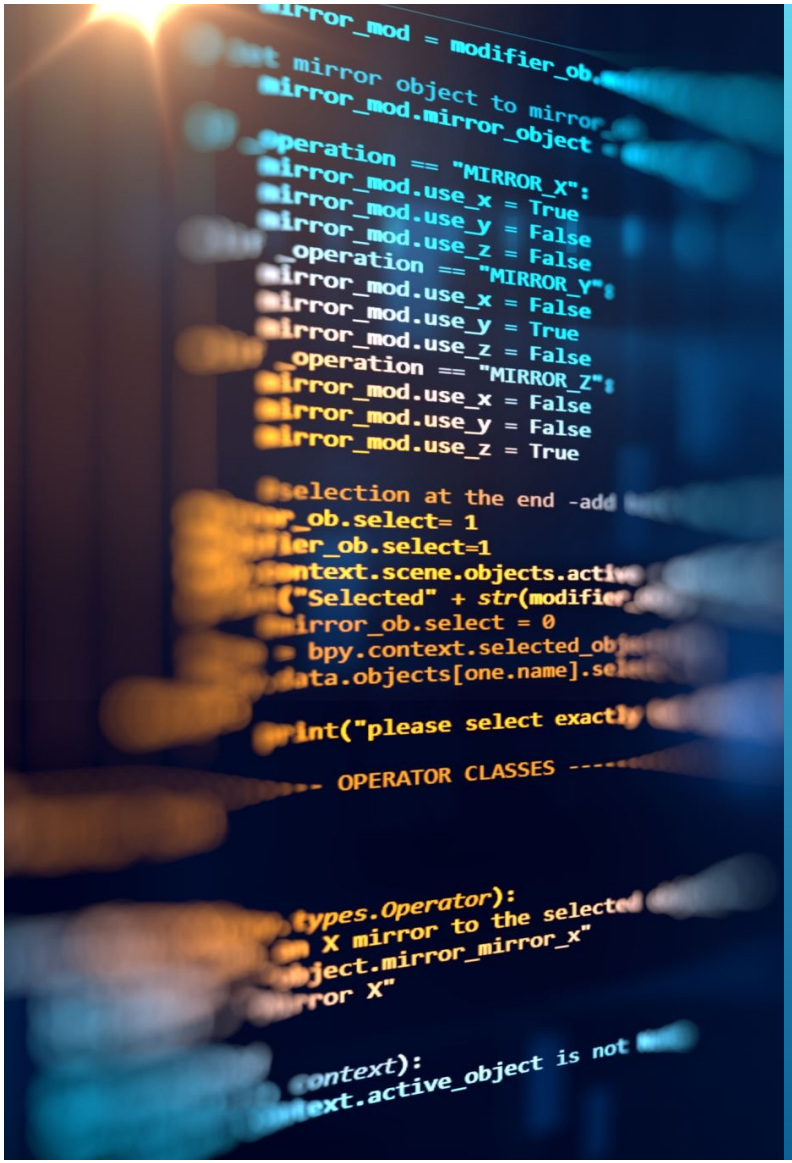
* as time permits



INTEGRATE IOT WITH REINFORCEMENT LEARNING



COMPUTER PROGRAMMING



PROGRAMMING LANGUAGES USED IN THIS COURSE

- C (and some C++) : for Arduino platform
- Python : for Jupyter notebook and ESP32 (micropython)
- JavaScript : some IoT programming

SW (OPEN-SOURCE OR FREE FOR BASIC USAGE)

- Arduino IDE and libraries
- NETPIE 2020 : https://netpie.io/
- micropython and libraries
- Thonny
- Jupyter notebook (colab or local)

PYTHON PROGRAMMING ON JUPYTER NOTEBOOK

Example 2: To experiment with the tracking and disturbance attenuation performance of PID control, we setup a feedback diagram in Figure 2.

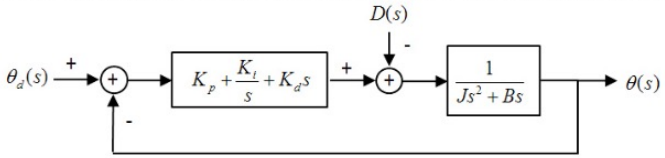


Figure 2 PID feedback with input disturbance

Such diagram is quite easy to construct using simulation engine like Simulink in MATLAB, or Xcos in Scilab. It is more tricky using time-domain simulation function in Python Control Systems library.

First, it can be derived that the transfer function from $D(s)$ to $\theta(s)$ is $P(s)S(s)$. So the disturbance response is the output from this transfer function. Suppose that The step disturbance at the plant input is active at half of time vector, with step value of 80. Adding the outputs together yields the step response with step disturbance input.

In [4]:

```
sys1 = ctl.feedback(L) # system for step command
S = 1/(1+L)
sys2 = P*S

tvec = np.arange(0,20,0.01) # adjut time vector if necessary
r = np.ones(tvec.shape)
d = 80*np.ones(tvec.shape)
d[:int(len(tvec)/2)] = 0.0
T1, y1 = ctl.step_response(sys1, tvec)
T2, y2 = ctl.forced_response(sys2, tvec, d)
y = y1 + y2
```

Note that in this example, we have the luxury of knowing the exact plant model. So it is possible to solve for a good set of PID gains. An approach used in [2] turns off the K_i gain first and solve for K_p and K_d that yields a critical-damped system at chosen natural frequency. Then K_i is adjusted to eliminate steady-state error.

To elaborate, it is shown in [2] that when K_i is set to 0, the resulting second order closed-loop characteristic polynomial is described by

$$s^2 + \frac{(B + K_d)}{J}s + \frac{K_p}{J} = s^2 + 2\zeta\omega s + \omega^2 \tag{7}$$

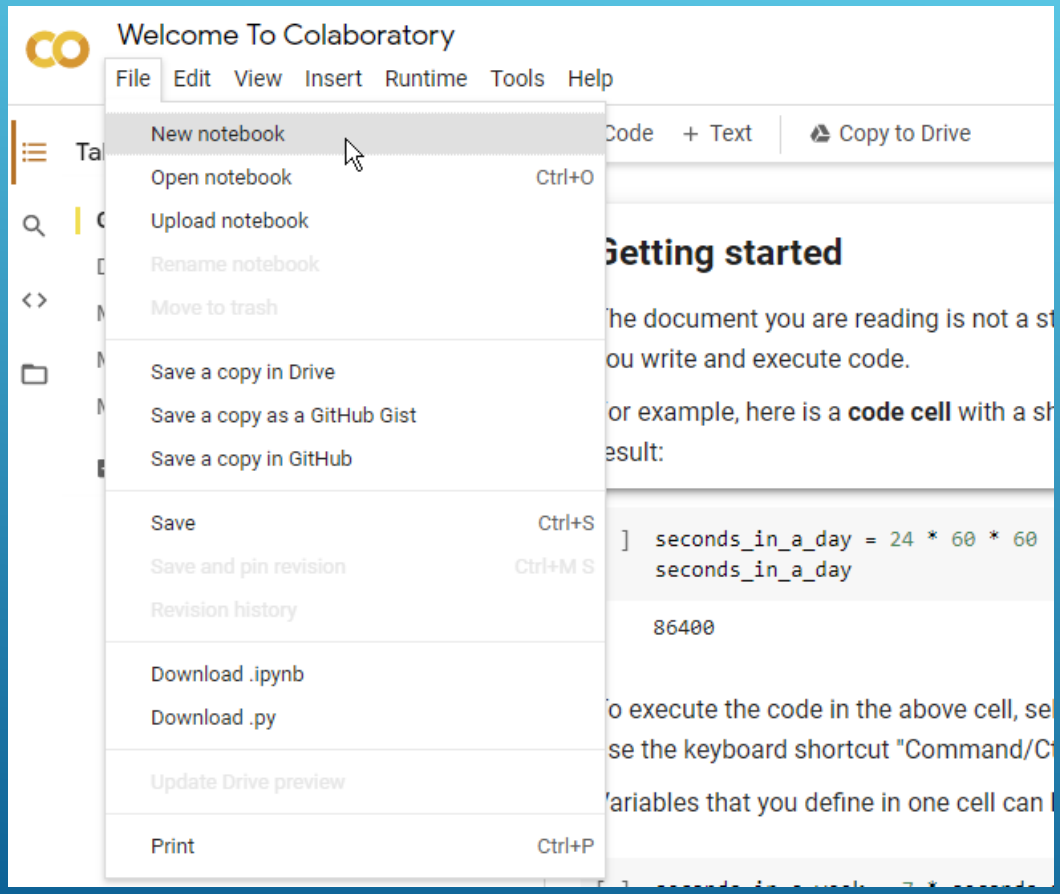
(7)

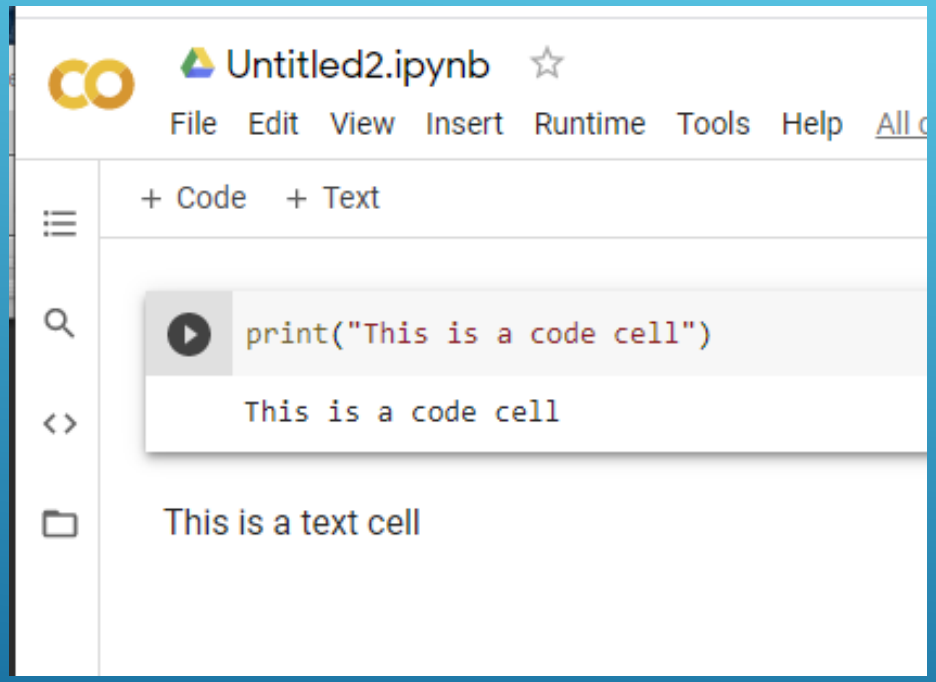
which gives

$$K_p = \omega^2 J, K_d = 2\zeta\omega J - B \tag{8}$$

(8)

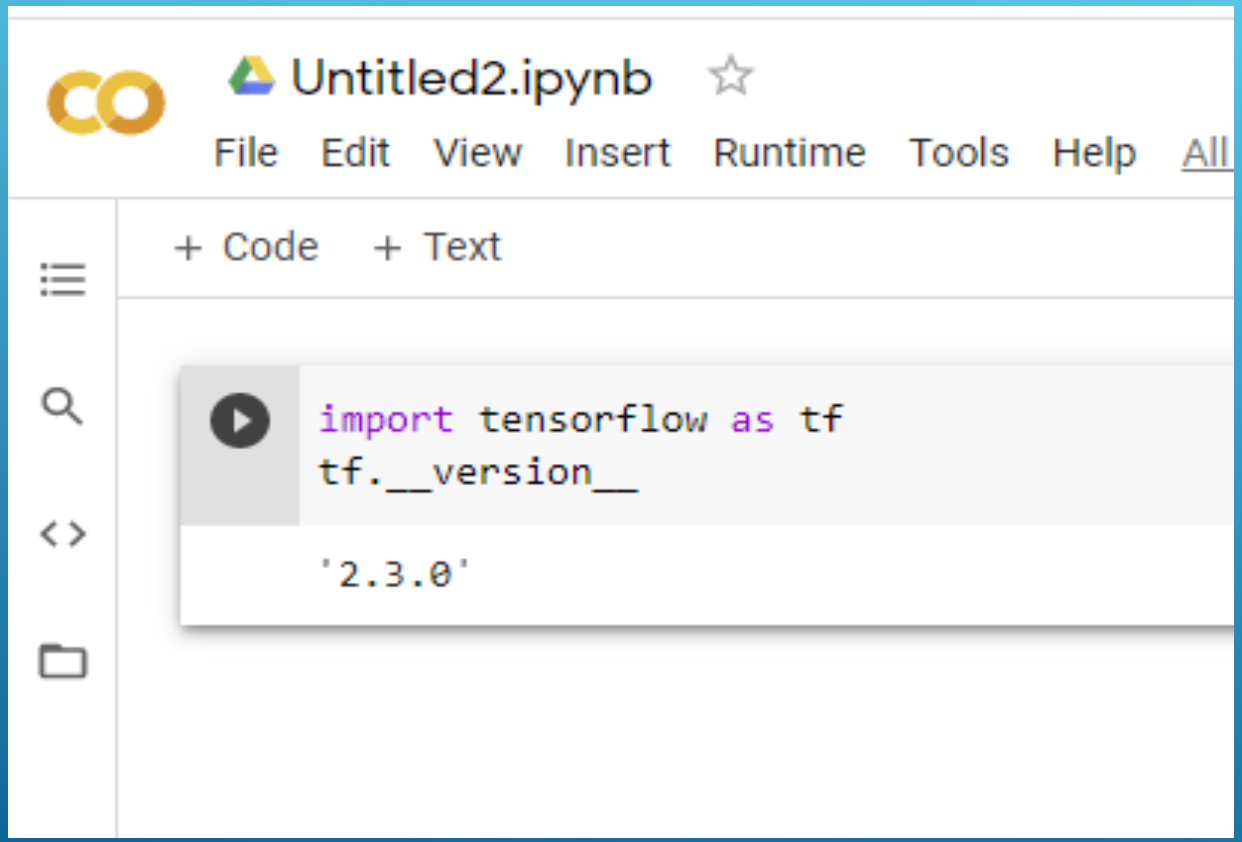
Begin Python Development in Colab





2 TYPES OF CELL IN NOTEBOOK

TensorFlow is already installed in Colab



TEXTS AND MATERIALS

- Texts
  - Handouts from instructor
- Online resources
  - github.com/dewdotninja
  - NETPIE website https://netpie.io/
  - Arduino https://www.arduino.cc/