

## **Teaching IoT in Both Physical and Virtual Environments**

**Prof. James R. Mallory, Rochester Institute of Technology (COE)**

**Edmund Lucas, National Technical Institute for the Deaf**

**William Arnold**

## **Teaching IoT in Both Physical and Virtual Environments**

Authors: Arnold, W., Fontaine, J., Griggs, S., Huff, G., Johnson, D., Linares, C., Patel, S., Reader, J., Roman, J., Sawaqed, Y., Yadav, R., Lucas, E. & Mallory, J. National Technical Institute for the Deaf / Rochester Institute of Technology

Primary Division: Computing and Information Technology Division

Secondary Division: Education and Research Methods Division

Tags: IoT, VM, Virtual, Raspberry Pi, student labs, project development, virtual labs

### **Abstract**

The growing field of the Internet of Things (IoT) is valuable for Engineering and Engineering Technology students to know. Due to COVID and often limited resources, this can be a difficult topic to teach. The authors pioneered a way to implement the same IoT systems both with physical devices and a Virtual Machine (VM) environment using a Raspberry Pi with servos, buttons, and lights. The VM used the Quick Emulator (QEMU) on the Ubuntu Linux platform. QEMU is a type 2 hypervisor that runs within the user space and performs virtual hardware emulation. The authors developed educational activities which allowed AAS/AOS level students to implement a Raspberry Pi lab using both physical and virtual devices including servos, buttons, and lights. The trials and tribulations will be shared as well as the successes with this student lab project implementation.

### **Background**

The Internet of Things (IoT) which incorporates objects communicating and interacting over the internet is a field that is growing with the market expected to reach \$1.8 trillion by 2028. There is a growing demand for technically skilled workers with IoT expertise. This represents an opportunity for postsecondary institutions to develop IoT curricula. Unfortunately, an IoT curriculum can be costly to implement, as this requires physical inventory and lab/storage space and students incur costs and are often left with superfluous hardware. The solution to this would be to implement virtual IoT laboratories which could be done at a reduced risk and cost. Using Virtual Machines (VM) also allows flexibility in the delivery of the coursework.

### **Prior Work (Literature Review)**

J. He et al [8] created a physical IoT lab consisting of Raspberry Pi and Arduino boards and a set of sensors with Zigbee as the wireless communication method. They developed a lab for an Embedded Systems Analysis and Design course. The lab was a collection of self-contained modules which presented concepts and hands-on exercises on embedded systems. This approach requires the use of physical hardware, so it can be costly to implement or scale up.

Also, labs cannot be delivered remotely. R. Krishnamurthi [12] used Node-RED, a visual programming language developed by IBM, to develop a two-credit, lab-only course offered to undergraduate engineering students. This was a project-based course to study IoT sensors, gateways, and cloud services. Node-RED can model application functionality between IoT devices but not the IoT devices themselves. This is useful for students to study the interaction between IoT devices, but it still requires investing in physical IoT hardware. M. Leisenberg and M. Stepponat [13] developed IoT demonstrators to be used for teaching. They used ThingSpeak, a publicly available cloud aggregator, and MATLAB to create a remote laboratory experiment on IoT-based analysis of moving images. While this approach allows for remote learning, students interact with the IoT data stream, not the IoT devices. Scaling up still requires investing in additional hardware and could result in a higher cloud service cost.

## **Methods**

Two of the faculty authors of this article implemented a Level 1 lab activity that required their AAS level Applied Computer Technology students to develop a system with both a physical Raspberry Pi model and in a VM environment and compare the two systems. A level (L1) activity means that the project involves both research and implementation and will take several weeks to complete. These would include the QEMU hypervisor, Python programming, the GPIO Library, and Interfacing with Lights, Servos, and LEDs. The comparisons included both functionality and the actual python coding similarities and differences.

The Raspberry Pi is the heart of both the physical and VM systems. The Raspberry Pi is well suited for IoT and uses the Linux-based Operating System and a general purpose Input/Output (GPIO).

### *Virtual (VM) System*

Steps students were required to do to set up the VM System included:

1. Install the Hypervisor VMWare
2. Install Ubuntu Linux from an Image
3. Install the Qemu Emulator
4. Run Raspberry Pi Emulation in Qemu, emulate GPIO devices using the Python Library called TKGPIO

The QEMU is a type 2 hypervisor capable of emulating the Raspberry Pi hardware. QEMU can emulate arm, mips, and sparc as well as X86 architectures and is highly customizable.

GPIO Emulation is accomplished using a TKGPIO which is a python library that emulates GPIO. It uses a GPIO zero API and allows the virtual environment to represent physical hardware with images and allows interaction using a mouse.

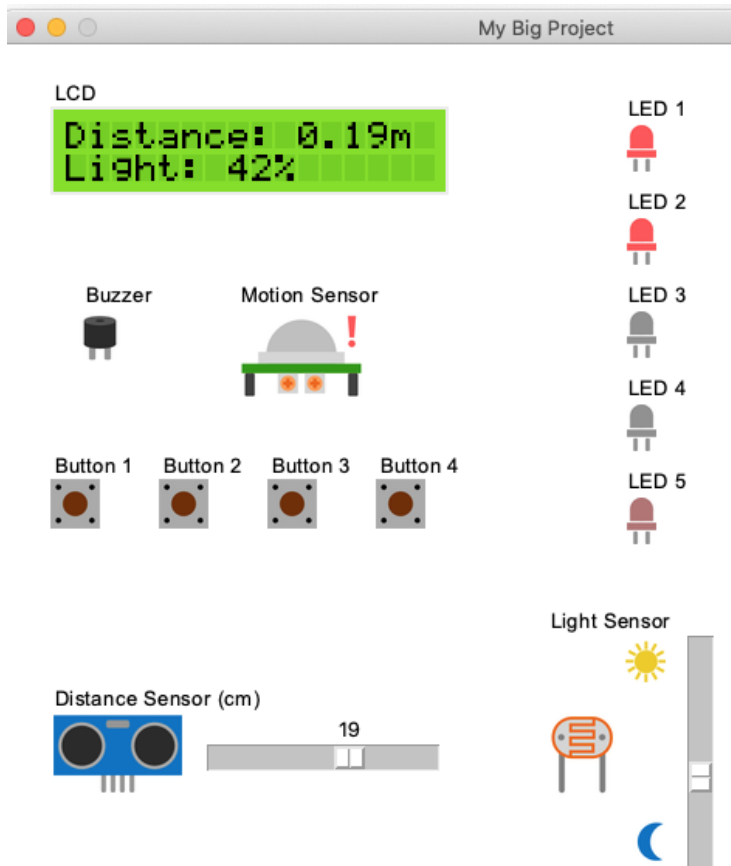


Figure 1. GPIO Emulation

Once the simulated Raspberry Pi environment is set up, the `tkgpio` python library is used to simulate Raspberry Pi GPIO devices and displays a graphical representation of the devices in a GUI constructed with Tkinter. Python programs interact with the `tkgpio` library using the `gpiozero` API. GPIO devices currently emulated include buttons, LEDs, motors, servos, motion sensors, potentiometers, and LCD displays.

`Tkgpio` acts like a kind of wrapper for the students' `gpiozero` python code. A `setup` function is used to define the simulated GPIO environment. The function can be written as a separate file and imported into their python program. The file should have a ".py" extension. The function describes the dimensions of the Tkinter GUI, the devices being emulated, the GPIO pins associated with the devices and the location of the devices within the GUI.

Take the following example code:

```
from tkgpio import TkCircuit

# initialize the circuit inside the GUI

configuration = {
    "width": 300,
    "height": 200,
    "leds": [
        {"x": 50, "y": 40, "name": "LED 1", "pin": 21},
        {"x": 100, "y": 40, "name": "LED 2", "pin": 22}
    ],
    "buttons": [
        {"x": 50, "y": 130, "name": "Press to toggle LED 2", "pin": 11},
    ]
}

def run (main_function):
    circuit = TkCircuit(configuration)
    circuit.run(main_function)
```

The “height” and “width” define the size of the GUI window. Each class of devices is defined as a list of individual devices in the class. In the above example, we define two LEDs and one button. Each device’s placement within the GUI is defined by an x and y coordinate within the GUI. The “name” is the label displayed over the device, and the pin is the GPIO pin associated with the device. The “run” function will be used in the students’ python program to start the GUI and run the program inside of it.

To use the virtual GPIO devices, simply import the setup file.

```
from <your-setup-file> import run
```

Do not include the .py extension here. Import the corresponding devices from the gpiozero library.

```
from gpiozero import LED, Button
```

Import and/or write any other functions as a student normally would, including his/her main function. Lastly, execute the main function inside the setup environment.

```
main = run(main)
```

## Physical System

The basic steps the students needed to implement in the physical system included getting the Raspberry Pi set up and interfaced with the proper pins, voltages, and input/output connections.

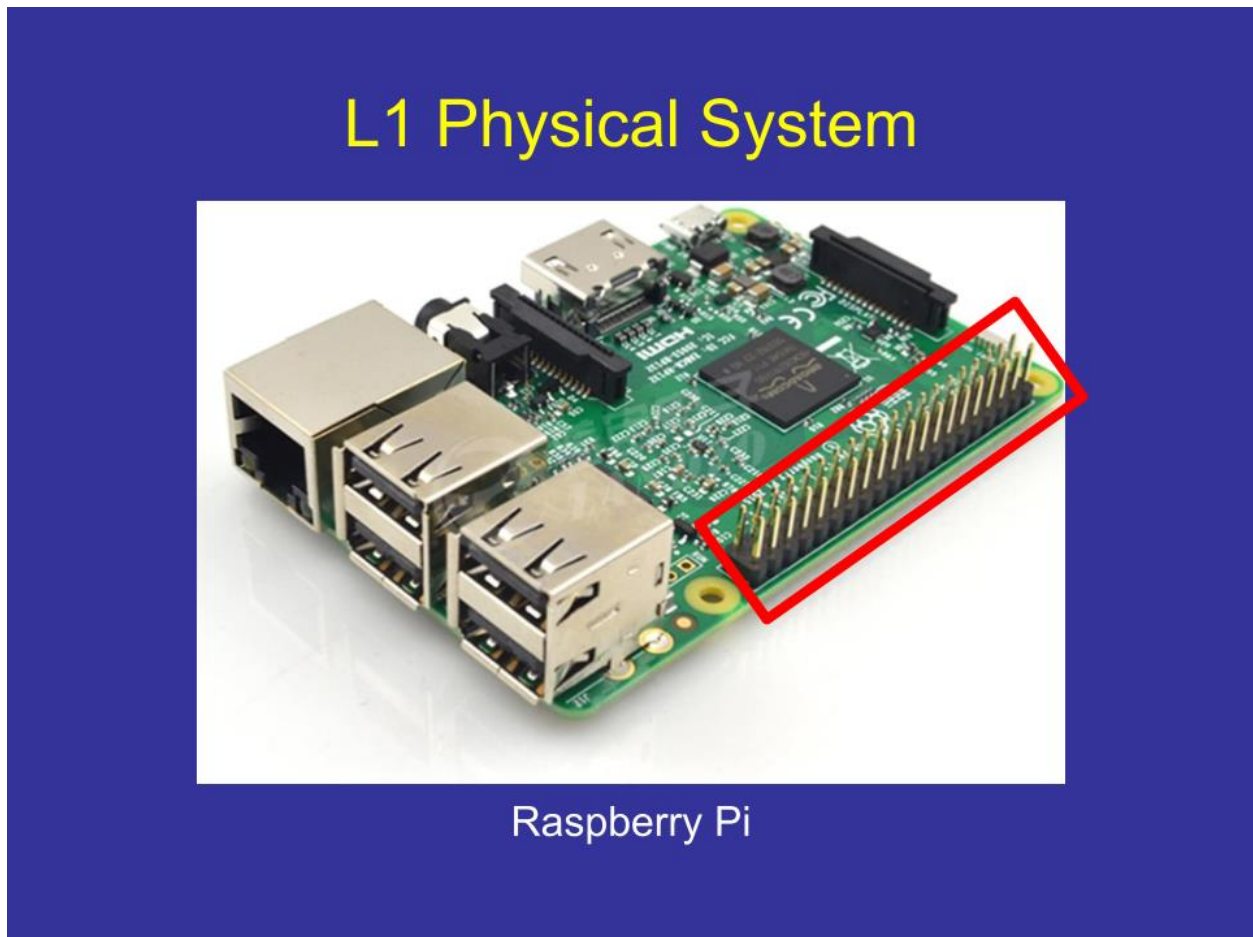


Figure 2. Raspberry Pi with I/O Pins

# L1 Physical Pin Layout

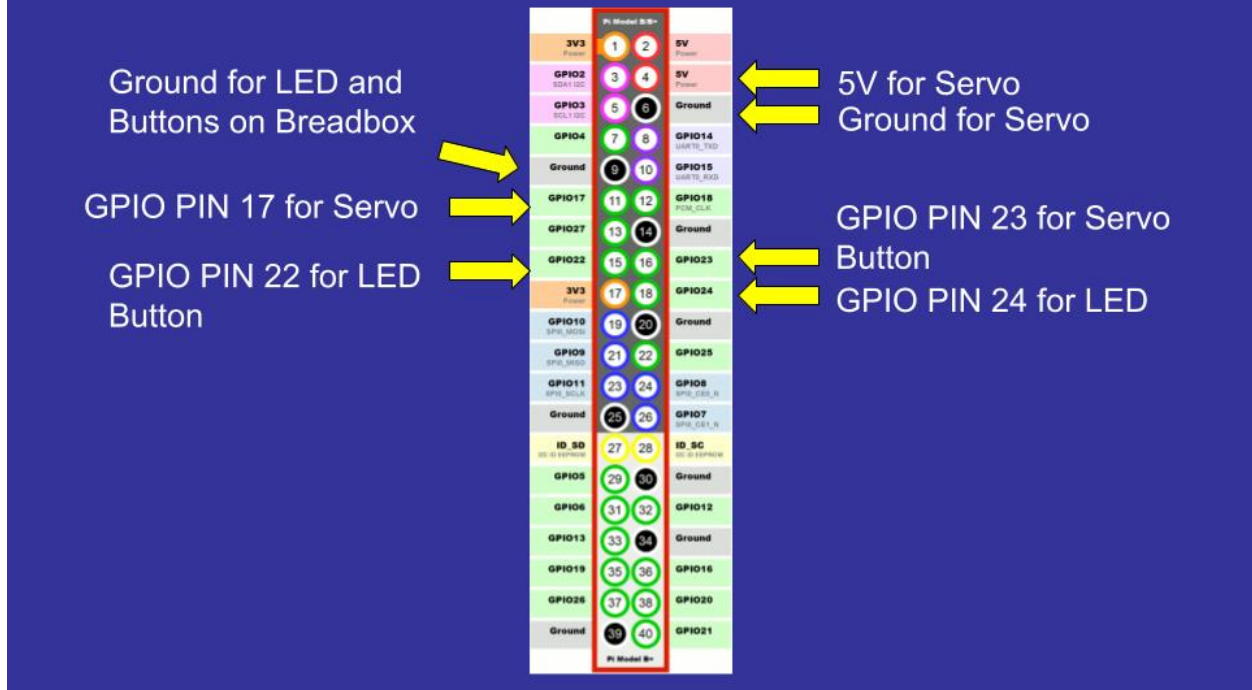


Figure 3. Raspberry Pi with I/O Pin Layout for Student Project

# L1 Physical System

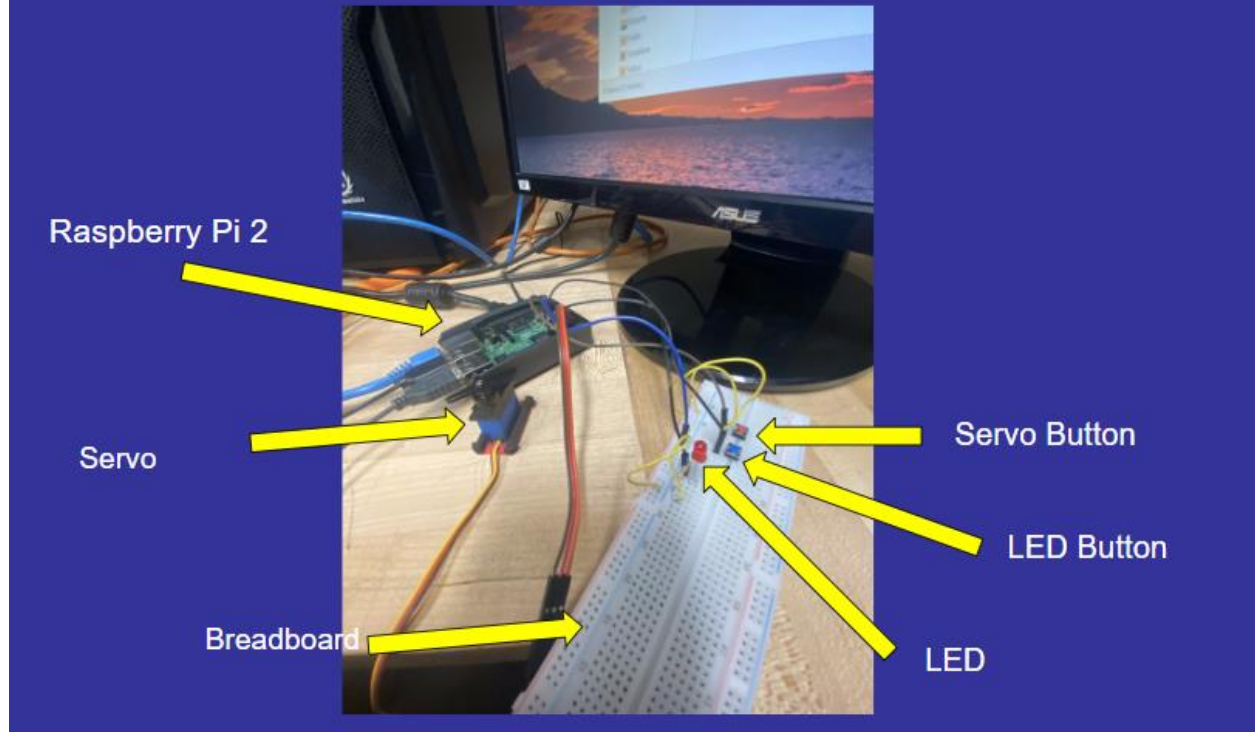


Figure 4. Raspberry Pi Physical System Setup



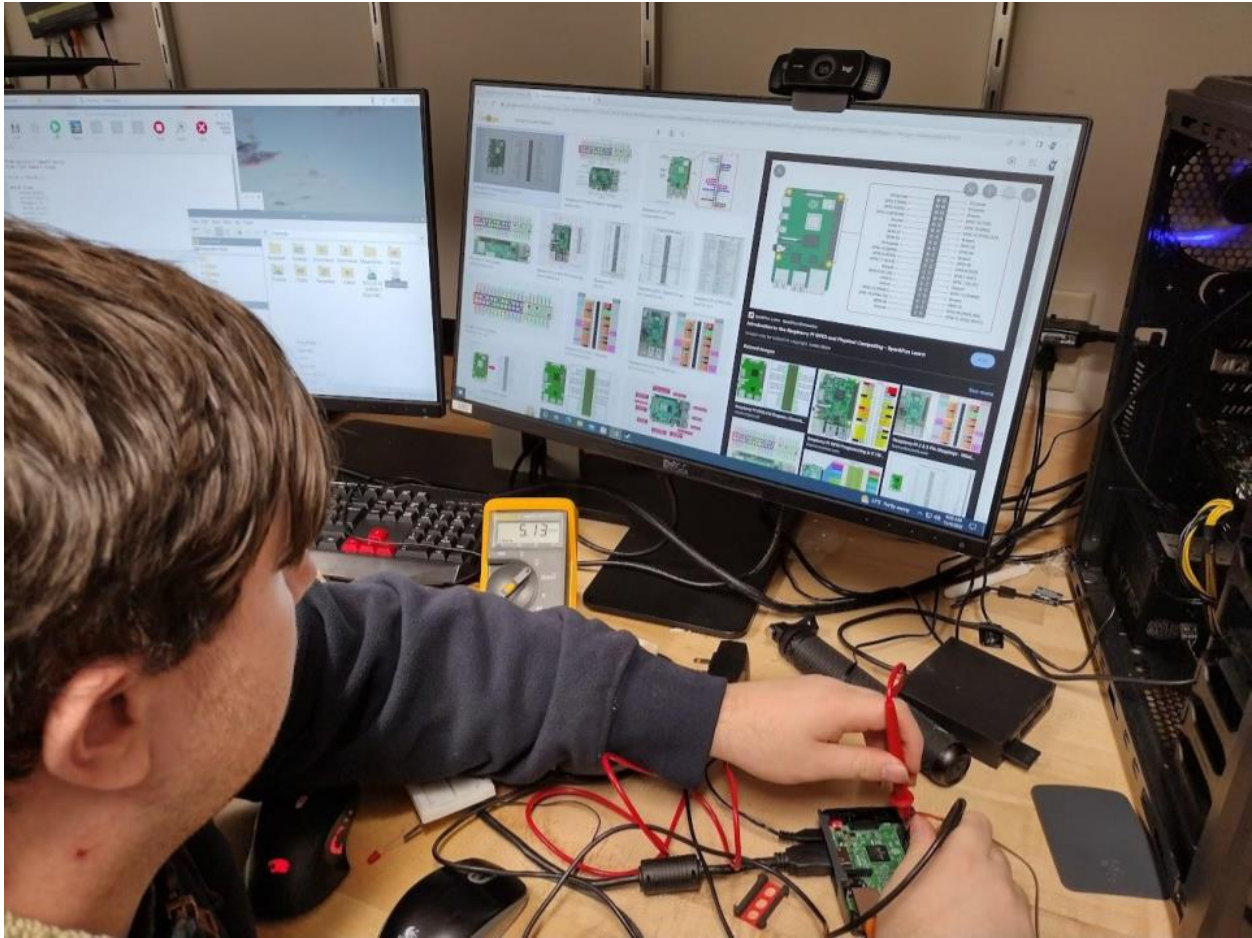
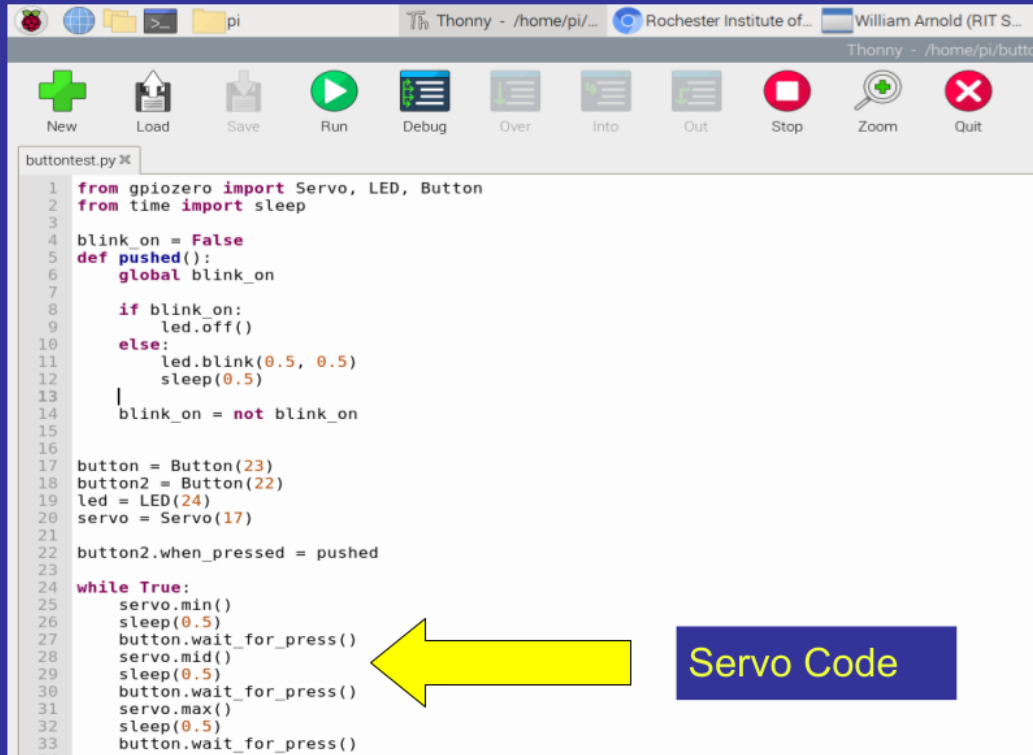


Figure 5. Student Measuring Voltages on Raspberry Pi

```
1 from gpiozero import Servo, LED, Button
2 from time import sleep
3
4 blink_on = False
5 def pushed():
6     global blink_on
7
8     if blink_on:
9         led.off()
10    else:
11        led.blink(0.5, 0.5)
12        sleep(0.5)
13
14    blink_on = not blink_on
15
16
17 button = Button(23)
18 button2 = Button(22)
19 led = LED(24)
20 servo = Servo(17)
21
22 button2.when_pressed = pushed
23
24 while True:
25     servo.min()
26     sleep(0.5)
27     button.wait_for_press()
28     servo.mid()
29     sleep(0.5)
30     button.wait_for_press()
31     servo.max()
32     sleep(0.5)
33     button.wait_for_press()
```

Figure 6. Python Code showing PIN Definition

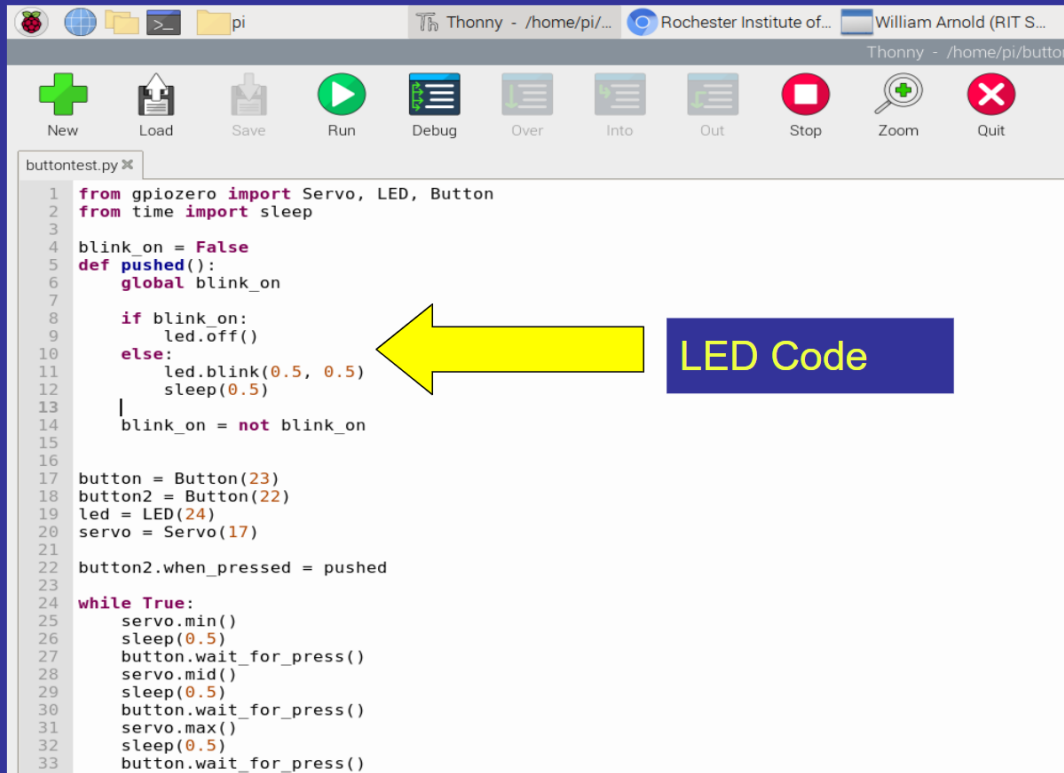
# Code for Physical System



```
1 from gpiozero import Servo, LED, Button
2 from time import sleep
3
4 blink_on = False
5 def pushed():
6     global blink_on
7
8     if blink_on:
9         led.off()
10    else:
11        led.blink(0.5, 0.5)
12        sleep(0.5)
13
14    blink_on = not blink_on
15
16
17 button = Button(23)
18 button2 = Button(22)
19 led = LED(24)
20 servo = Servo(17)
21
22 button2.when_pressed = pushed
23
24 while True:
25     servo.min()
26     sleep(0.5)
27     button.wait_for_press()
28     servo.mid()
29     sleep(0.5)
30     button.wait_for_press()
31     servo.max()
32     sleep(0.5)
33     button.wait_for_press()
```

Figure 7. Python Code showing Code for Servo

# Code for Physical System



```
1 from gpiozero import Servo, LED, Button
2 from time import sleep
3
4 blink_on = False
5 def pushed():
6     global blink_on
7
8     if blink_on:
9         led.off()
10    else:
11        led.blink(0.5, 0.5)
12        sleep(0.5)
13    |
14    blink_on = not blink_on
15
16
17 button = Button(23)
18 button2 = Button(22)
19 led = LED(24)
20 servo = Servo(17)
21
22 button2.when_pressed = pushed
23
24 while True:
25     servo.min()
26     sleep(0.5)
27     button.wait_for_press()
28     servo.mid()
29     sleep(0.5)
30     button.wait_for_press()
31     servo.max()
32     sleep(0.5)
33     button.wait_for_press()
```

Figure 8. Python Code showing Code for LEDs

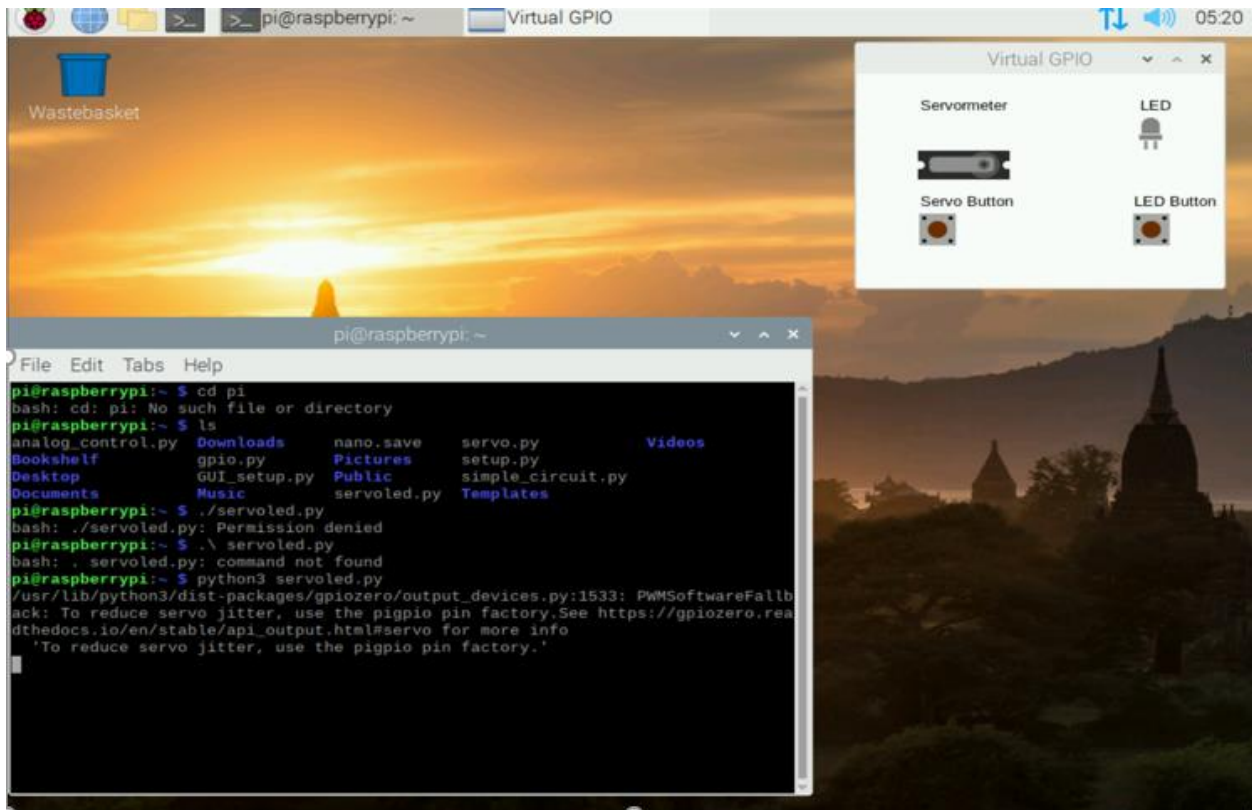


Figure 9. VM Environment

```

from tkgpio import TkCircuit

configuration = {
    "width": 300,
    "height": 200,

    "servos": [
        {"x": 50, "y": 40, "name": "Servometer", "pin": 17, "min_angle": -90,
         "max_angle": 90, "initial_angle": 20}
    ],

    "leds": [
        {"x": 230, "y": 40, "name": "LED", "pin": 18}
    ],

    "buttons": [
        {"x": 50, "y": 130, "name": "Servo Button", "pin": 11},
        {"x": 225, "y": 130, "name": "LED Button", "pin": 12}
    ]
}

def run (main_function):
    circuit = TkCircuit(configuration)
    circuit.run(main function)

```

← Set up Window

Figure 10. Python Code showing Code for Window Set up

```

from tkgpio import TkCircuit

configuration = {
    "width": 300,
    "height": 200,

    "servos": [
        {"x": 50, "y": 40, "name": "Servometer", "pin": 17, "min_angle": -90,
         "max_angle": 90, "initial_angle": 20}
    ],

    "leds": [
        {"x": 230, "y": 40, "name": "LED", "pin": 18}

    "buttons": [
        {"x": 50, "y": 130, "name": "Servo Button", "pin": 11},
        {"x": 225, "y": 130, "name": "LED Button", "pin": 12}
    ]
}

def run (main_function):
    circuit = TkCircuit(configuration)
    circuit.run(main function)

```

← LED Definition

Function called by main program

Figure 10. Python Code showing Code for LED Definition



```

from tkgpio import TkCircuit

configuration = {
    "width": 300,
    "height": 200,

    "servos": [
        {"x": 50, "y": 40, "name": "Servometer", "pin": 17, "min_angle": -90,
         "max_angle": 90, "initial_angle": 20}
    ],

    "leds": [
        {"x": 230, "y": 40, "name": "LED", "pin": 18}
    ],

    "buttons": [
        {"x": 50, "y": 130, "name": "Servo Button", "pin": 11},
        {"x": 225, "y": 130, "name": "LED Button", "pin": 12}
    ]
}

def run (main_function):
    circuit = TkCircuit(configuration)
    circuit.run(main_function)

```

← Function called by main program

Figure 11. Code Showing Main Program Function Calls

```

from tkgpio import TkCircuit

configuration = {
    "width": 300,
    "height": 200,

    "servos": [
        {"x": 50, "y": 40, "name": "Servometer", "pin": 17, "min_angle": -90,
         "max_angle": 90, "initial_angle": 20}
    ],

    "leds": [
        {"x": 230, "y": 40, "name": "LED", "pin": 18}
    ],

    "buttons": [
        {"x": 50, "y": 130, "name": "Servo Button", "pin": 11},
        {"x": 225, "y": 130, "name": "LED Button", "pin": 12}
    ]
}

def run (main_function):
    circuit = TkCircuit(configuration)
    circuit.run(main_function)

```

← Function called by main program

Figure 12. Python Code showing Code for Servo and LED button Definitions

```

from tkgpio import TkCircuit

configuration = {
    "width": 300,
    "height": 200,

    "servos": [
        {"x": 50, "y": 40, "name": "Servometer", "pin": 17, "min_angle": -90,
         "max_angle": 90, "initial_angle": 20}
    ],

    "leds": [
        {"x": 230, "y": 40, "name": "LED", "pin": 18}
    ],

    "buttons": [
        {"x": 50, "y": 130, "name": "Servo Button", "pin": 11},
        {"x": 225, "y": 130, "name": "LED Button", "pin": 12}
    ]
}

def run (main_function):
    circuit = TkCircuit(configuration)
    circuit.run(main_function)

```

Annotations in the image:

- Set up Window (points to width and height)
- Servo Motor Definition (points to the servo configuration)
- LED Definition (points to the LED configuration)
- Servo and LED Buttons Definition (points to the buttons configuration)
- Function called by main program (points to the run function)

Figure 13. Summary of Python Code for VM System

## Results

Students learned that the code listed below in bold was the same for both the physical and the VM implementation of their system:

```

from servo_setup import run
from time import sleep
def main():
    while True:
        sleep(0.1)
main=run(main)

```

## Conclusions

This hands-on laboratory was successful, as students learned how to use Raspberry Pi and code in Python, and implement both physical and virtual systems. There was a steep learning curve for AAS-level students who never had experience with Raspberry Pi or Linux. The students described in this paper had exposure to Linux before the project but not to Raspberry



Pi. The Tkgpio simulation was slow to open but performed reasonably well once the simulation began.

Students struggled in the beginning due to the learning curve, more so on the Virtual side than they did on the physical side. One of the other negatives to this lab activity was the Raspberry Pi VM was very slow to start because of the older hardware it was implemented on. This was great exposure to the IoT world and complemented their computer and networking knowledge in their current AAS technical program.

## References

- [1] N. Ahmad, P. Laplante, and J. F. DeFranco, "Life, IoT, and the Pursuit of Happiness," *IT Professional*, vol. 22, no. 6, pp. 4–7, Nov. 2020, doi: 10.1109/mitp.2019.2949944.
- [2] C. Border, "The Development and Deployment of a multi-user, Remote Access Virtualization System for networking, security, and System Administration Classes," presented at the Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, 2007.
- [3] M. Chernyshev, Z. Baig, O. Bello, and S. Zeadally, "Internet of Things (IoT): Research, Simulators, and Testbeds," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1637–1647, Jun. 2018, doi: 10.1109/jiot.2017.2786639.
- [4] M. Dawson, F. G. Martinez, and P. Taveras, "Framework for the Development of Virtual Labs for Industrial Internet of Things and Hyperconnected Systems," *IEEE Xplore*, Oct. 01, 2019. <https://ieeexplore.ieee.org/document/8939660> (accessed Jun. 19, 2022).
- [5] "Emulate Raspberry Pi with QEMU," *Azeria-Labs*, 2022. <https://azeria-labs.com/emulate-raspberry-pi-with-qemu/> (accessed 2022).
- [6] "Emulating ARM64 Raspberry Pi Image using QEMU," *www.youtube.com*, Sep. 04, 2020. <https://www.youtube.com/watch?v=Y-FUvi1z1aU> (accessed Mar. 20, 2022).
- [7] R. Foley, "How to connect two aarch64 QEMU guests with a bridge," *ARM-Datacenter*, Aug. 06, 2020. <https://futurewei-cloud.github.io/ARM-Datacenter/qemu/network-aarch64-qemu-guests/> (accessed Mar. 12, 2022).
- [8] J. He, D. Chia-Tien Lo, Y. Xie, and J. Lartigue, "Integrating Internet of Things (IoT) into STEM undergraduate education: Case study of a modern technology infused courseware for embedded system course," presented at the IEEE Frontiers in Education Conference (FIE), Erie, PA, 2016.
- [9] "How to connect GPIO in QEMU-emulated machine to an object in host?," *www.embeddedrelated.com*, Mar. 20, 2020. <https://www.embeddedrelated.com/showthread/comp.arch.embedded/272409-1.php> (accessed Feb. 12, 2022).

- [10] J. K. S, "tkgpio: A Python library to simulate electronic devices connected to the GPIO on a Raspberry Pi , using TkInter," *GitHub*, Jul. 22, 2021. <https://github.com/wallysalami/tkgpio> (accessed Jun. 20, 2022).
- [11] S. Koch, "Hosting QEMU VMs with Public IP Addresses using TAP Interfaces - s.koch blog," *blog.stefan-koch.name*, Oct. 25, 2020. <https://blog.stefan-koch.name/2020/10/25/qemu-public-ip-vm-with-tap> (accessed Mar. 11, 2022).
- [12] R. Krishnamurthi, "Teaching Methodology for IoT Workshop Course Using Node-RED," presented at the Eleventh International Conference on Contemporary Computing (IC3), Noida India, Aug. 2018.
- [13] M. Leisenberg and M. Stepponat, "Internet of Things Remote Labs: Experiences with Data Analysis Experiments for Students Education," presented at the IEEE Global Engineering Education Conference (EDUCON), Dubai, United Arab Emirates, Apr. 2019.
- [14] B. Nuttall and D. Jones, "gpiozero — GPIO Zero 1.6.2 Documentation," *gpiozero.readthedocs.io*, 2015. <https://gpiozero.readthedocs.io/en/stable/> (accessed Jun. 2022).
- [15] L. J. Pérez and S. Rodriguez, "Simulation of scalability in IoT applications," presented at the International Conference on Information Networking (ICOIN), Chiang Mai, Thailand, Jan. 2018.
- [16] "QEMU documentation," *www.qemu.org*. <https://www.qemu.org/docs/master> (accessed Jun. 20, 2022).
- [17] B. Ramprasad, M. Fokaefs, J. Mukherjee, and M. Litoiu, "EMU-IoT - A Virtual Internet of Things Lab," presented at the IEEE International Conference on Autonomic Computing (ICAC), Umea, Sweden, Jun. 2019.
- [18] "Setting up Qemu with a tap interface," *Gist*, Feb. 13, 2018. <https://gist.github.com/extremecoders-re/e8fd8a67a515fee0c873dcafc81d811c> (accessed Mar. 11, 2022).
- [19] Source Meets Sink, "Emulating ARM64 Raspberry Pi Image using QEMU," *www.youtube.com*, Sep. 04, 2020. <https://www.youtube.com/watch?v=Y-FUvi1z1aU> (accessed Mar. 2022).
- [20] SUDONULL, "Virtual GPIO driver with QEMU ivshmem interrupt controller for Linux.," *SudoNull*, 2019. <https://sudonull.com/post/80905-Virtual-GPIO-driver-with-QEMU-ivshmem-interrupt-controller-for-Linux> (accessed Mar. 20, 2022).
- [21] S. Ugwuanyi and J. Irvine, "Security Analysis of IoT Networks and Platforms," presented at the International Symposium on Networks, Computers and Communications (ISNCC), Montreal, QC, Canada, 2020.
- [22] R. Vella Galea, "Raspberry Pi GPIO Emulator," *Roderick Vella Galea*, Jun. 28, 2016.

<https://roderickvella.wordpress.com/2016/06/28/raspberry-pi-gpio-emulator/> (accessed Apr. 20, 2022).

[23] Fortune Business Insights, "Internet of Things (IoT) market size, share and industry analysis," Oct. 2021. [Online]. Available: <https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307>. Accessed on: Dec. 6, 2021

[24] Rochester Institute of Technology.(2021, Spring). Foundations of IoT-ISTE730.