STEMKIT 4SCHOOLS

TRAFFIC LIGHTS WITH PYTHON AND GPIO

LESSON PLAN 2

Co-funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission.

Table of Contents

1. Traffic Lights with Python and GPIO

1.1 General information

1.1.1 Short description

In this lesson plan we will learn how to build traffic lights in the real world and control them through the GPIO and the Python programming language. The lesson plan involves the creation of the circuit using the GPIO pins of our Raspberry Pi and electronic components, and the development of a program in Python that will control the traffic lights sequence.

1.1.2 Learning objectives

The main learning objectives of this lesson plan are:

- Concept and content understanding of queuing theory, traffic engineering and theory of bottlenecks.
- Familiarising with electronic components and creating circuits and programs to interact with GPIO pins of your Raspberry Pi.
- Understanding basic structures of Python programming language (syntax, functions, loops, if statements, etc.).

1.1.3 Links to curriculum

The domains, subdomains, subjects/topics that this lesson plan can be linked to are:

- Maths: queuing theory, waiting lines, queues.
- Engineering: bottlenecks, service systems, queues.
- Computer Science / Informatics: processing unit and peripherals, interfaces, programming language and main structures, coding.
- Technology: electronics, open-source hardware and software, sensors, digital signal, circuits, single board computers.

1.1.4 Materials required

For this lesson plan (and for each student group) besides the STEMKIT console and its Raspberry Pi we will need:

- 6 x Male-to-Female jumper wires
- 1 x Male-to-Male jumper wire
- 1 x Breadboard
- 1 x Push button
- 1 x Buzzer
- 3 x LED (Red, Amber and Green)
- 3 x 220 Ohm Resistors

1.1.5 Duration

The duration of this lesson plan is estimated to be about 45-60 mins, i.e., one classroom hour.

1.2 Lesson plan

The lesson plan is divided in three phases, which are introduction, preparation, and conclusion.

1.2.1 Introduction

Queueing theory is the mathematical study of waiting lines, or queues. A queue is a container of objects that are inserted and removed according to the first-in-first-out (FIFO) principle. New additions to the line made to the back of the queue, while removal happens in the front.

Queueing is also the study of traffic behaviour near a certain section where demand exceeds capacity. Queues can be seen in many situations: boarding a bus/plane, freeway bottlenecks, shopping checkout, exiting the classroom at the end of a lecture, or waiting for a hamburger at the local burger joint.

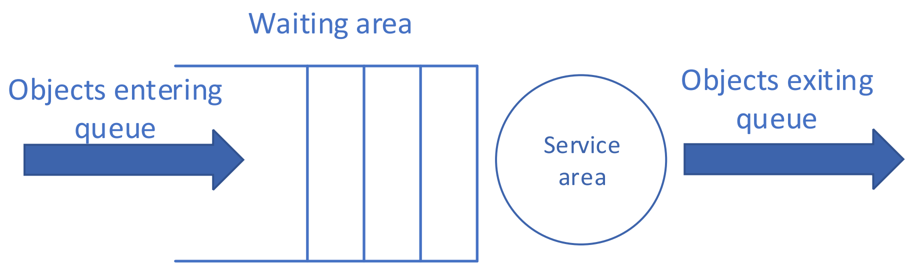

FIGURE 1 SIMPLE QUEUE DIAGRAM (FIRST-IN-FIRST-OUT)

There are different statistical distributions that explains a queue. Arrival distribution could be deterministic (random) or random (e.g., Poisson). Service distribution is either deterministic or random. Furthermore, there are different service models:

- First In First Out (FIFO)
- Last In First Out (LIFO)
- Priority (bypasses the queue)

In traffic engineering, queueing can occur at red lights, stop signs, bottlenecks, or any design-based or traffic-based flow constriction. When not treated properly, queues can cause severe network congestions, therefore making them something important to be studied and understood. The most common service model that explains a queue at a traffic light is FIFO. However, under certain circumstances the service model change (e.g., When traffic is controlled by a traffic policeman).

In queueing theory, we use the following notation:

- Arrival rate = λ
- Departure rate = μ
- Utilization rate $\rho = \lambda / \mu$

Understanding the degree of saturation of a queue, we consider the following:

- Oversaturated: $\lambda > \mu$
- Undersaturated: $\lambda < \mu$
- Saturated: $\lambda = \mu$

A traffic queue is explained using Little's formula:

Little's Formula: $E(n) = \lambda E(v)$

The formula depicts that the average queue size (measured in vehicles) equals the arrival rate (vehicles per unit of time) multiplied by the average waiting time (both delay time in queue and service time) (in units of time). This result is independent of particular arrival distributions and is an important fundamental principle that was not proven until 1961.

In our case we will use the theory of queues to identify the optimal traffic light signalling (in time units – seconds) so we arrival rate λ is equal or almost equal to departure rate μ.

Now that we have acquired the basic understanding of queues, let's process with the preparation of our experiment.

1.2.2 Preparation

1.2.2.1 Creating the circuit

First thing we need to do is to create a circuit and connect it to the GPIO pins of our Raspberry Pi. Turn off the Raspberry and unplug any cables attached to it. For our circuit we need a breadboard, LEDs (red, amber, green), resistors, jumper wires, a buzzer and a push button.

We need to place all the components on the breadboard and connect them to the appropriate GPIO pins on the Raspberry Pi. Understanding how each component is connected, we need to bear in mind the following:

- A push button requires 1 ground pin and 1 GPIO pin
- An LED requires 1 ground pin and 1 GPIO pin, with a current limiting resistor.
- A buzzer required 1 ground pin and 1 GPIO pin. Components can share a ground pin. We use the breadboard to enable this.

The complete circuit is in the schematic diagram that follows.

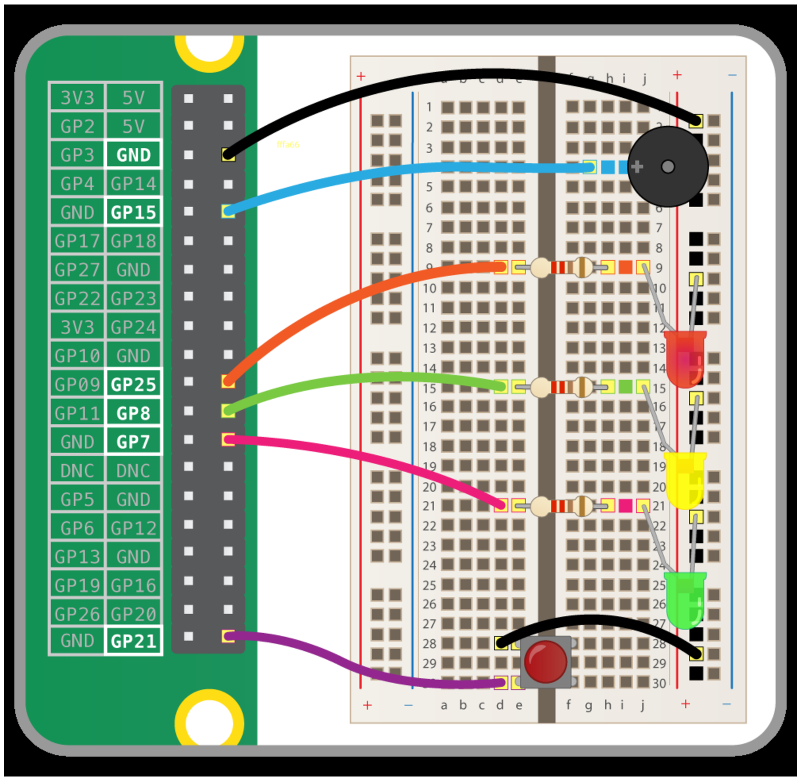

FIGURE 1 SCHEMATIC DIAGRAM OF TRAFFIC LIGHTS CIRCUIT CONNECTED TO GPIO PINS. SOURCE: PROJECTS.RASPBERRYPI.ORG

Note that the row along the long side of the breadboard is connected to a ground pin on the Raspberry Pi, so all the components in that row are connected to ground.

Regarding the GPIO pins, we use the following:

- Button to 21
- Red LED to 25
- Amber LED to 8
- Green LED to 7
- Buzzer to 15

When we have completed the circuit, we can turn on the Raspberry Pi and start developing our program in Python.

1.2.2.2 Python programming for the circuit

We need to create a program that control the circuit we created in the previous step.

In brief, in our program we first import the Python modules that we need, then set the GPIO pins for LEDs, button and buzzer.

We use the module ***gpiozero*** because it builds on *RPi.GPIO* and has much simpler coding rules.

```
##############################################################################
# import libraries and set gpio numbering mode
from gpiozero import Button, TrafficLights, Buzzer
from time import sleep

# Button interface for controlling the push button
button = Button(21)
#TrafficLights interface for simulating an actual traffic light
lights = TrafficLights(25, 8, 7)
# Buzzer interface for controlling the buzzer
buzzer = Buzzer(15)

while True:
# Sequence for green light
    lights.green.on()
    button.wait_for_press()             #like in pedestrian crossing
    sleep(3)                            #delay to execute the code

#Sequence for amber light
    lights.green.off()
    lights.amber.on()
    sleep(3)

#Sequence for red light
    lights.red.on()
    buzzer.beep()
    lights.amber.off()
    sleep(15)

    lights.amber.on()
    buzzer.off()
    sleep(1)

    lights.red.off()
    lights.amber.off()
```

In GPIO Zero, we create objects for each component used. Each component interface must be imported from the *gpiozero* module, and an instance created on the GPIO pin number to which it is connected.

The *TrafficLights* interface takes three GPIO numbers, one for each pin: red, amber, green (in that order). We can use *on, off* and *blink,* all of which control all three lights at once.

The *Buzzer* interface takes one GPIO number and can be controlled by: *on, off* and *beep*. Using number within the parentheses we can manipulate the buzzing time.

The program above creates the following traffic lights sequence:

- Green on
- Amber on
- Red on
- Red and Amber on
- Green on

The button simulates a pedestrian crossing. When a “pedestrian” hits the button, the lights move to red (with some delay) and give the pedestrians some time to cross the street before moving the lights back to green. The addition of a buzzer enhances the simulation, as in a real crossing there is a buzzer for the benefit of visually impaired pedestrians.

Check that your program works. Hit the button a couple of times and see your traffic lights work. Try changing the time for green light and red light.

1.2.3 Conclusion

We have succeeded creating a traffic lights system and pedestrian crossing using Python programming and the GPIO of our Raspberry. In this phase we recapitulate what we did and how, which were the main steps and discuss difficulties experienced. As a follow-up exercise, discuss ways of optimizing the scripts used for this experiment.

1.3 References or Resources

Here are some useful references and additional resources related to this lesson plan.

- https://projects.raspberrypi.org/en/projects/physical-computing/10
- https://gpiozero.readthedocs.io/en/stable/
- https://www.cs.cmu.edu/~adamchik/15-121/lectures/Stacks%20and%20Queues/Stacks%20and%20Queues.html
- https://en.wikipedia.org/wiki/Queueing_theory
- https://en.wikibooks.org/wiki/Fundamentals_of_Transportation/Queueing#cite_ref-1
- https://en.wikipedia.org/wiki/Traffic_engineering_(transportation)