Embedded Systems lab 1

# Objective

The objective of this lab is to learn basic principles of input/output with the Raspberry Pi platform. These principles include:

* Create and clone a GitHub repository to store the project and allow it to be moved from the development machine to the Raspberry Pi
* Become familiar with available Python libraries for running web servers and threading

# Materials

Materials needed to complete this lab are:

* 1x Raspberry Pi (I used a Raspberry Pi 400) with Raspberry Pi OS installed
* 1x USB-C Power supply sufficient for powering the Raspberry Pi
* 1x Red LED
* 1x Yellow LED
* 1x Green LED
* 3x 220Ω Resistor
* 3x M-M jumper wires
* 1x Half size breadboard
* 1x Raspberry Pi GPIO ribbon cable
* 1x Raspberry Pi GPIO breakout

# Procedures

## Hardware

1. Connect the GPIO ribbon cable to the Raspberry Pi’s GPIO pins. On a Raspberry Pi 400, the connector is keyed, so it can only be inserted in the correct orientation. Connect the other end of the ribbon cable to the GPIO breakout.
2. Connect the GPIO breakout to the breadboard. Make sure the power pins on the side get connected to the power rails of the breadboard such that the +5v and +3.3v are on the rail marked as positive (a red stripe in my example).
3. Connect the LEDs, resistors, and jumper wires as pictured A picture containing text, electronics

   Description automatically generated
4. Connect the power adapter to AC power, and plug in the Raspberry Pi.

## Software

If you are writing the code and accessing the webpage from the Pi itself, skip to step 4. Full source code is available in Appendix A.

1. Acquire the IP address of the Raspberry Pi by running the command **ifconfig** in a terminal.
2. Set up a GitHub repository to store the code, and allow easy movement of code from the development machine to the Raspberry Pi.
3. Model the functionality required for this project as shown here:

Diagram

Description automatically generated

1. Install python dependencies using **pip3 install threading flask time** in the terminal.
2. Create a web server to run on the Raspberry Pi using Python Flask with the following endpoints:
   1. /[filename] – this path will serve up any file within a subdirectory of the project (the www folder) and is used to serve the index.html file and control.js. This improves the experience when using the IDE so it can do syntax highlighting for HTML and JavaScript. This path is only available for use using HTTP GET requests.
   2. /control/ – this endpoint is used for receiving JSON data from a web browser, and controls the lights based on the values received. This endpoint only processes HTTP POST requests.

These endpoints are designed to be easy to interact with in an automated fashion.

1. Create an HTML file called index.html that contains the web interface. It will include buttons to manually change which LED is lit while the traffic light is in manual mode, as well as a button to change the mode to automatic mode. When loaded in a web browser, the interface will look like this:

Chart, bubble chart

Description automatically generated

1. Create a JavaScript file to listen for click events on the buttons listed in step 6 which will send HTTP POST requests to the endpoint listed in 5b.
2. If needed, add these files to the Raspberry Pi’s file system.
3. Run the program by navigating to the directory where the Python file is located, then running the command **python3 [name of python file]**
4. The page can now be accessed using the IP address of the Raspberry Pi from 1, or at 127.0.0.1 if accessing from the Raspberry Pi itself.

I am using Python3 because Python is a simple language with a robust support for most things including web servers and GPIO libraries for the Raspberry Pi.

# Sources

Flask routers tutorial: <https://flask.palletsprojects.com/en/2.0.x/quickstart/>

Serving static HTML in flask: <https://stackoverflow.com/questions/20646822/how-to-serve-static-files-in-flask>

Global mode to modify global variables within flask routes: <https://stackoverflow.com/questions/19182963/global-variable-and-python-flask>

Threading in python: <https://www.geeksforgeeks.org/how-to-create-a-new-thread-in-python/>

Toggle switch code (I ended up scrapping this part because it kept sending duplicate requests): <https://stackoverflow.com/questions/39846282/how-to-add-the-text-on-and-off-to-toggle-button>

# Thought Questions

1. What language did you choose for implementing this project? Why?

I chose to implement this lab in Python and JavaScript because Python has good libraries for handling GPIO and listening to web requests.

1. What is the purpose of the resistor in this simple circuit? What would happen if you omitted it?

The resistor is to limit the amount of current that can flow through the LEDs. If it were omitted, the resistor would glow brighter, but would have a significantly shorter life, and may burn out if too much current flows through.

1. How do you ensure that all http-requests are non-malicious? What possible exploits could an attacker do against your system? How do you protect against these?

I don’t ensure they aren’t malicious, but only requests with the exactly correct key values in the request JSON are processed, all others are ignored.

1. What are practical applications of this device? What enhancements or modifications would you make?

This device could be used to control an actual traffic light. I wrote it specifically to be easy to automate from an external controller. I would enhance it to be able to respond with the lights’ actual status.

1. Please estimate the total time you spent on this lab and report

I spent about 6 hours on the lab, and about 3 hours on the report.

Certification of Work

I certify that the solution presented in this lab represents my own work. In the case where I have borrowed code or ideas from another person, I have provided a link to the author’s work in the references, and included a citation in the comments of my code.

# Appendix A

## ./app.py

# allow threading for automatic mode

from threading import Thread

# set up flask

from flask import Flask

from flask import request

from flask import send\_from\_directory

from time import sleep

app = Flask(\_\_name\_\_)

# set things up for GPIO

import RPi.GPIO as GPIO

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)

RED = 23

YELLOW = 24

GREEN = 25

GPIO.setup(RED, GPIO.OUT)

GPIO.setup(YELLOW, GPIO.OUT)

GPIO.setup(GREEN, GPIO.OUT)

# set things up for control of mode on the server side so we can get out of automatic mode

global mode

mode = 'manual'

def lights(light: str):

    """Change the light to the color indicated"""

    if light == 'green':

        GPIO.output(GREEN, True)

        GPIO.output(YELLOW, False)

        GPIO.output(RED, False)

    if light == 'yellow':

        GPIO.output(GREEN, False)

        GPIO.output(YELLOW, True)

        GPIO.output(RED, False)

    if light == 'red':

        GPIO.output(GREEN, False)

        GPIO.output(YELLOW, False)

        GPIO.output(RED, True)

    if light == 'off':

        GPIO.output(GREEN, False)

        GPIO.output(YELLOW, False)

        GPIO.output(RED, False)

# a function for automatic mode

def auto(time\_red: int = 15, time\_yellow: int = 3, time\_green: int = 15):

    """Automatically cycle between red, green, then yellow"""

    # this seems redundant, but we need to be able to stop the loop at any point.

    # if this is not completed, the loop will finish executing, leaving the light yellow

    # even if the light was manually set to another color while the loop was executing

    # before that line

    while mode == 'auto':

        if mode == 'auto':

            lights('red')

        if mode == 'auto':

            sleep(time\_red)

        if mode == 'auto':

            lights('green')

        if mode == 'auto':

            sleep(time\_green)

        if mode == 'auto':

            lights('yellow')

        if mode == 'auto':

            sleep(time\_yellow)

# add route to serve webpage and javascript

@app.route('/<path:path>', methods=['GET'])

def home(path):

    return send\_from\_directory('www', path)

# add route to listen for control data from the webpage

@app.route('/control/', methods=['POST'])

def control():

    global mode

    thread = Thread(target = auto, args = ())

    # automatic mode

    if (request.json.get('mode') == 'auto'):

        mode = 'auto'

        # start the thread to control the LEDs in automatic mode without holding up the flask loop

        thread.start()

    # manual mode

    elif (request.json.get('mode') == 'manual'):

        mode = 'manual'

        if (request.json.get('color')):

            lights(request.json.get('color'))

    return request.json.get('mode')

# @app.route('/js/<path:path>')

# def send\_js(path):

#     return send\_from\_directory('js', path)

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(host='0.0.0.0')

## ./www/index.html

<title>Traffic Light</title>

<head>

    <style>

        .circle {

            width: 100px;

            height: 100px;

            -webkit-border-radius: 25px;

            -moz-border-radius: 25px;

            border-radius: 50px;

        }

        #red {

            background: red;

        }

        #yellow {

            background: yellow;

        }

        #green {

            background: green;

        }

        .modeButton {

            padding: 32px 64px;

            color: white;

        }

        #Auto {

            background-color: green;

        }

        #manual {

            background-color: red;

        }

    </style>

</head>

<h1>Traffic Light</h1>

<body>

    <button type="button" id="manual" class="modeButton">Manual</button>

    <button type="button" id="auto" class="modeButton">Automatic</button>

    <div id="red" class="circle"></div>

    <div id="yellow" class="circle"></div>

    <div id="green" class="circle"></div>

</body>

<footer>

    <script src='./control.js'></script>

</footer>

## ./www/control.js

var url = "control/"

var mode = "manual"

sendRequestMode(mode)

function sendRequest(color){

    let xhr = new XMLHttpRequest();

    xhr.open("POST", url, true);

    xhr.setRequestHeader("Content-Type", "application/json; charset=UTF-8");

    xhr.send(JSON.stringify({"mode": mode, "color": color}));

}

function sendRequestMode(){

    let xhr = new XMLHttpRequest();

    xhr.open("POST", url, true);

    xhr.setRequestHeader("Content-Type", "application/json; charset=UTF-8");

    xhr.send(JSON.stringify({"mode": mode}));

}

document.getElementById("red").addEventListener("click", function () {

    if (mode == "manual"){

        sendRequest("red")

    }

    else{

        alert("cannot set color manually while not in manual mode.")

    }

});

document.getElementById("yellow").addEventListener("click", function () {

    if (mode == "manual"){

        sendRequest("yellow")

    }

    else{

        alert("cannot set color manually while not in manual mode.")

    }

});

document.getElementById("green").addEventListener("click", function () {

    if (mode == "manual"){

        sendRequest("green")

    }

    else{

        alert("cannot set color manually while not in manual mode.")

    }

});

document.getElementById("auto").addEventListener("click", function () {

    mode = "auto"

    sendRequestMode()

});

document.getElementById("manual").addEventListener("click", function () {

    mode = "manual"

    sendRequestMode()

});