**Intro**

For Project 1, I used a Raspberry PI along with sensors, actuators, and accessories to create a Python script that controls LEDs and a Servo motor with a keypad. My Raspberry PI 3B+ runs Raspberry PI OS (previously called Raspbian), and my project code was written in Python 3. I connected my Raspberry PI to a solderless breadboard using a 26-pin ribbon cable and T-Cobbler breakout, used to ease GPIO pin access. For sensors, I used a four-by-four membrane keypad. For actuators, I used three LEDs (green, yellow, and red) and a SG90 Servo motor with a cross-type bar installed.

**Design and Implementation**

Diagram, schematic

Description automatically generated

(Circuit diagram, image also included separately in project folder)

My project design centers around controlling LEDs and a Servo motor with a four-by-four membrane keypad. My code (.py file included in project folder) implements the entirety of the keypad; however, I only used the first two rows for this project, specifically keys 1, 2, 4, 5, and 6. When the project starts, the green and yellow LEDs start blinking and a command line interface (CLI) informs the user what the five buttons do. The green LED implements step 1 of the project problem statement, the yellow LED satisfies step 2, and the Servo is my, professor approved, replacement for step 3. Implementing a keypad instead of simple buttons represents my bonus section.

The main try loop cycles through checking for input, interpreting the input, and manipulating the actuators based on the stored input. The algorithm for detecting input sends a pulse to each of the four keypad rows (I only pulse the first two for my project since rows 3 and 4 are unused), then checks which column is pulled high per row. The column that is pulled high when its row is pulsed is the currently pressed key. The next algorithm reads an array based on which row and column combination were read and stores that key’s assigned symbol to the input variable. Finally, the actuators are manipulated in different ways depending on the input stored while printing a description of the manipulation on the CLI.

Initially, the green LED blinks at a constant rate of once per second. Pressing button 1 will begin a cycle in which the one second interval is sped up by 0.25 seconds after each blink, returning to one second once it reaches zero. Pressing button 1 again will freeze the speed up cycle. Pressing button 2 will change the behavior of button 1 to control the yellow LED. Unlike the simple green LED, the yellow LED uses PWM modulation to control brightness. Once button 2 has been pressed, pressing button 1 will dim the yellow LED by cutting its PWM value in half. Pressing button 1 again will return the yellow LED to full brightness and pressing button 2 again will swap button 1 back to controlling the green LED.

Buttons 4, 5, and 6 control the Servo motor and red LED. Pressing button 4 rotates the Servo clockwise (-90-degree position) and turns on the red LED. Pressing button 5 returns the Servo to its middle position (0-degree position) and turns off the red LED. Finally, pressing button 6 rotates the Servo counterclockwise (90-degree position) and blinks the red LED five times. Originally, this part of the project (Step 3) was going to read a temperature sensor and adjust the red LED based on temperature readings; however, due to a faulty temperature sensor I was given permission to use a Servo instead.

My project uses three Python 3 libraries. First, RPi.GPIO (<https://pypi.org/project/RPi.GPIO/>) is used to set up the Raspberry Pi’s GPIO pins. Next, gpiozero (<https://gpiozero.readthedocs.io/>) is used to simplify LED, PWM LED, and Servo control. The project code only uses the base functions of this library. For example, it do not use the blink() LED function, instead using the on() and off() functions to gain greater control over the component behavior. Finally, the Python3 Time library (<https://docs.python.org/3/library/time.html>) is used to control the flow of component behavior. Initially, a fourth library, Adafruit-DHT 1.4.0 (<https://pypi.org/project/Adafruit-DHT/>), was going to be used for the DHT11 Temperature and Humidity sensor. Due to a faulty sensor, this library was excluded from the final build.

**Difficulties and Lessons Learned**

The greatest challenge of this project was encountering a faulty component. Originally, I planned to use a DHT11 Temperature and Humidity sensor for step 3 of the project. However, after hooking up the sensor to the power, ground, and GPIO pin, the sensor completely failed to produce any readings. I tested the sensor in a new python script using the native Adafruit DHT library, and then tested with two third party DHT libraries. I simplified their code examples as much as possible but was still never able to get a single reading. Finally, I tried multiple power, ground, and GPIO pins, as well as multiple wires and breadboard rails to rule out other hardware issues. Ultimately, I was forced to conclude that the temperature sensor was faulty and secured permission from the instructor to replace step 3 with a Servo control step instead.

Other than the faulty sensor, the rest of the project was completed without issues. I had not used a breadboard since a physics class several years ago. While I already owned the Raspberry PI before this course, I had never used its GPIO pins. I enjoyed learning how to control external components using GPIO, Python, and various sensors and actuators. I have also rarely used Python and enjoyed writing code in language with which I am less practiced. I plan to extend this project to add a Servo controller to allow me to control multiple Servos which is not recommended without an external controller and a separate power source from the Raspberry Pi. My goal for this project in the future is to control multiple Servos to manipulate several appendages of a moveable statue I own.