June 13, 2016

Spring Quarter

Security system: the smart light bulb controller

EECS 113 – Final project

Bernardita Štitić

#88092134

1. **Introduction**

The device that was built is a smart light bulb controller which is similar to a security light. It has a light sensor and a motion sensor so that only if the area is dark enough and movement is sensed, a light bulb will be turned on for approximately 30 seconds using a mechanical relay controller (relay + amplifier). After this time period, the light bulb will be turned off automatically and the environment will be sensed again using the sensors.

Additionally, this system goes one step further than a typical security light – it can be incorporated into an IoT system. This means that by pressing a switch that activates an interrupt service routine, the light bulb can be turned on and off through Wifi using the MQTT protocol (CloudMQTT) for emergency/security purposes. It is then possible to use this Wifi control to deactivate the interrupt service routine and go back to sensing the environment. More details about this will be given later.

1. **Bill of materials (BOM)**

For this project, the required materials are the following:

* **Raspberry Pi 3 Model B:** a microcontroller unit (MCU); the one used for this project had an SD card with Raspbian installed. For this MCU, it is also required to have a power adapter (a 5V 2A in this case) to power it up. Finally, it is also needed to have a GPIO cable and a breakout board to run the Python code for the prototype.
* **Breadboard:** the board used to build the simple prototype.
* **Light dependent resistor (LDR/photocell):** also called a light sensor, an LDR outputs an analog signal that represents the resistance of a light variable resistor. Particularly, the value decreases as light intensity increases and vice versa. The one used for this project also came with a potentiometer.
* **PIR (Passive Infra Red) motion sensor:** a sensor which reacts to infrared heat, consequently indicating movement. The one used for this prototype outputs a digital signal using 3.3V logic, so that it can be directly connected to the breakout board.
* **ADC:** a MCP3008 chip was used to read the analog outputs coming from the LDR.
* **Tactile switch:** a tactile switch (with 4 terminals) was used to activate the interrupt service routine.
* **10K resistor:** used with the tactile switch to create an appropriate voltage drop.
* **Mechanical relay controller:** this included a relay and an amplifier inside it. It is capable of turning on the device connected to it (a light bulb in this case) if it receives 3.3V. It can also turn it off if it is given 0V.
* **Light bulb:** with a base, it’s the component that will be turned on or off.
* **Power strip:** this will power up the prototype, since the Raspberry Pi’s adapter and the relay will be connected to it.
* **Jumper wires:** these are used to make connections within the prototype.
* **Male to female wires:** these are used to connect the sensors to the MCU or the ADC.
* **Wood board:** to mount the prototype and isolate it electrically.
* **Tape:** to mount and stick the parts of the prototype to the board.

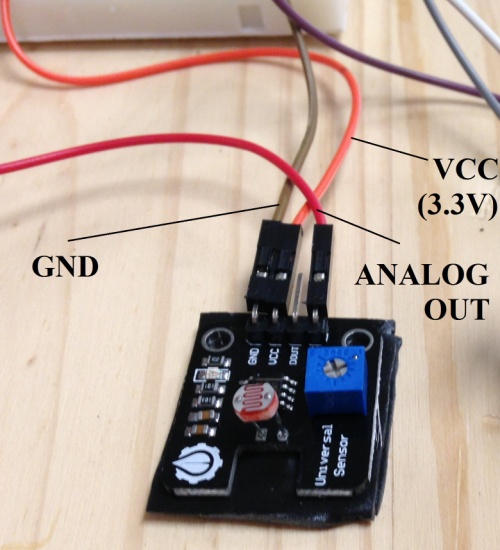
Also, it is important to emphasize that a special open source library for Python called the Paho library (MQTT client library) was installed on the Raspberry Pi. More details about this will be given later.

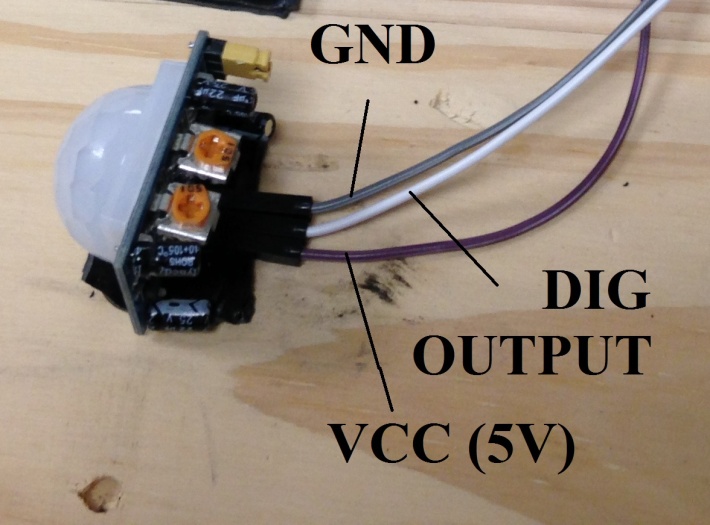
1. **Trade-offs and limitations of the design**

There are two important trade-offs/limitations in this project. The first one is the speed of publishing. The program is designed so that the Raspberry Pi 3 publishes to CloudMQTT what the PIR and the LDR are sensing. For the first case, it’s a string that’s either 1 (movement) or 0 (no movement) under the topic ‘Movement’. For the second case, it will post both the analog output under the topic ‘Light\_raw’ and its logical value with the topic ‘Light’. Therefore, if the analog input is above 800 the logical value will be 0 (no light). The contrary happens if it’s below 800.

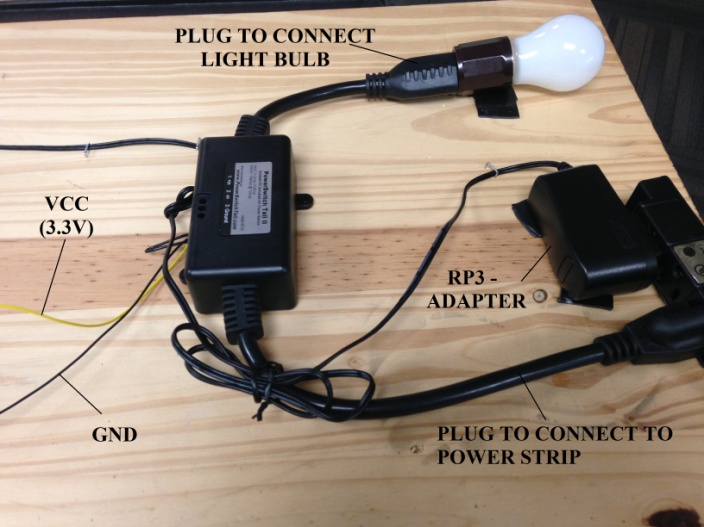
One cannot have real time sensing (with publishing) and a suitable speed on the CloudMQTT terminal. This means that if one seeks real time sensing, the messages will be sent extremely fast. On the contrary, one can add delays (with for loops) to make the messages appear with a more suitable speed, but the sensing activity will not be realistic. Therefore, a delay (with for loops) was added but suitable enough so that the sensing and publishing tasks were still efficient.

The second and final trade off/limitation was due to the Paho library for the MQTT. This library makes it so that if one wants to send a message to the cloud and have the device act upon it, only this activity can be done. This means that it’s not possible to publish the results of the sensors and read messages to react at the same time. This is why the Wifi control of the light bulb is activated through an interrupt service routine using a tactile switch for security purposes.

1. **Diagrams of PIR, LDR and the mechanical relay**



**Figure 1. LDR and its terminals Figure 2. PIR sensor and its connections**

****

**Figure 3. Mechanical relay with connections, light bulb and power strip**

In Figure 1, the ANALOG OUT is connected to Channel 0 of the ADC. In Figure 2, the DIG OUTPUT goes to GPIO 16. In Figure 3, VCC goes to GPIO 12.

1. **Limitations of the sensors and actuator**

The main limitation of the sensors is that they output a signal with a certain delay. However, this delay is so that the output is almost instantaneous, so a delay with for loops was introduced in the code to publish at a reasonable speed. Also, another limitation could be the output voltage logic of a sensor. In the case of the PIR, it was 3.3V, but in the case of the LDR it depended on the value chosen to feed it (5V or 3.3V). To obtain reasonable outputs 3.3V was chosen, since if 5V were used the ADC would read values that would stay high. Finally, a mechanical relay takes up more space than a solid state relay. However, this last one has to be connected directly to AC current so it’s not as safe. Therefore, the mechanical one was chosen with a big wood board to mount it.

1. **Code structure and source files**

The code (project2.py) begins by importing the required libraries. Here, the Paho library is imported. To be able to do this, one must install the Paho library by Eclipse for Python on the RPi3. After this, 5 functions are defined. The first four are for the MQTT protocol. Specifically, the second one, on\_message, was modified so that the system can react differently according to the message and topic sent to the cloud from the laptop. The final one is the readadc function taken from assignment 6 (analogread.py) to be able to read an analog input using the ADC. The GPIO pins connected to the ADC chip were kept the same as in the analogread.py file.

Then, certain configurations are made including the setup of the GPIO pins and the breakout board. With these declarations, the required variables for the code are also declared. Two of these are lists to optimize the program. After this, the MQTT is configured with my account’s id, password, port terminal and host name (it is necessary to create an account at cloudmqtt.com). Immediately after this, the subscriptions to the topics used for this device are made. The template for connecting, publishing, subscribing and the MQTT functions were taken from the file mqtt\_pythoncode.py.

After this, the program enters the main code – a ‘while True’ loop. Inside, it enters another while loop that meets a condition (that will be false if the switch for the ISR is pressed). Firstly, the switch is checked and then a delay is introduced. After this, the PIR and the LDR are read (this last one using the ADC). Their outputs are published to the cloud (digital and analog). If the combination of the outputs is the desired one (movement and darkness), the light bulb will be turned on for 30 seconds.

If the switch is pressed, then the code will enter the mqttc.loop() where the device will only check messages sent to the cloud from the laptop (this is done by modifying a variable). With the topic ‘SOS’, one can turn the light bulb on (message ‘1’) /off (message ‘0’). With the topic ‘Sense’, one can go back to reading from the sensors (and the variable that keeps the program in the mqttc.loop() will be modified so that it goes back to the while True loop). With the topic ‘Over’, the program will exit the giant while True loop, execute GPIO.cleanup() and finish the program.

1. **Important**

* Upon booting, the Raspberry Pi 3 will send an email with its IP address to my email, from my gmail email. This must be fixed if other email addresses are desired in the startup\_mailer.py code.
* For the sensors, the primary sensor would be the PIR. The secondary sensor would be the LDR and arguably the switch and the MQTT control, from a broader perspective.
* Similarly for the topic of the actuators, the MQTT control can also be considered as so.
* For the algorithms, there are three: the motion controlled security light (with the PIR and LDR which trigger the light bulb), the remote control of the relay with Wifi and the publishing of the values of the sensors to CloudMQTT.
* I work since January for Calplug under Calit2, where IoT is the core of most projects. The idea of using the MQTT protocol came from research experience.