A Raspberry Pi-based autonomous home lighting system controlled by Google Calendar events

Abstract

With high-speed Internet access being available in practically all locations, people have begun to bridge the gap between the virtual and physical world. More particularly, users now have the ability to control the state of objects that are in different geographic locations. When connected to the Internet, a small computer system such as the Raspberry Pi (RPi) can be embedded into objects in order to remotely collect data from the environment and send commands to the objects. This enables objects to react on the events that take place in their environment. This concept, known as the Internet of Things (IoT), has not only led to increased efficiency in companies and homes, but has also contributed to the "Green Movement" that has started to surround our society.

In this paper, we propose an Autonomous Home Lighting System and describe a prototype that we developed utilizing Raspberry Pi (RPi) and Google calendar. In the past, RPi's have been used to turn lights on and off, but much of these works include the use of timers and switches. There has been little done in terms of using the Internet to make the process autonomous. The prototype takes a green initiative by automatically turning off lights when a user isn't home to conserve electricity while also granting the opportunity to turn lights on when needed. Determining whether or not a person is at home will be done by checking the events on the user's Google calendar. Google's popular tool allows for easy access to a user's events via a variety of methods, such as e-mail alerts or HTML extraction, making it an optimal source for gathering data. RPi can gather input from its environment, output commands to connected electronics, and connect to the Internet wirelessly. All these features make RPi an ideal component for an autonomous electrical system.

Introduction

Rapid advancements pertaining to the Internet of Things and small computer systems has created a technological landscape with endless opportunities to better the lives of people around the world. The Internet of Things, a concept which aims to grant Internet connectivity to everyday objects, has provided professionals of all kinds with the opportunity to analyze and alter objects so that they can behave optimally in an ever-changing environment. These objects can not only be controlled by humans at different geographic locations, but the data that they constantly collect can be used to enhance future improvements. Moreover, since all of these objects are connected via the Internet, this means that they can interact with each other. An event triggered by one object can cause others to react in a particular way. Therefore, in a holistic sense, the Internet of Things intertwines all of the components of large scale systems by providing a means of communication among them.

The booming popularity of objects within the Internet of Things can be largely attributed to the growing power of small computer systems. Advancements within the realms of their processing power, memory, sensors, and user-friendly operating systems has made them suitable for being embedded in a variety of everyday objects. Their skyrocketing popularity and power has driven down prices, giving the common consumer the opportunity to make "smart" objects of their own. In the past, these embedded small computer systems were used primarily by governments and large companies to monitor their infrastructure and dynamically manage resources. With small computer systems and their sensors now being affordable, “smart” objects have become more than just a tool for improving operation efficiency. Consumers are able to buy credit card-sized computers, such as the Raspberry Pi 2 or Arduino, and develop systems that can improve their quality of life. This has led to an influx of creative ideas ranging from home security systems to customized baby monitors. As a result, these small computer systems, linked together through the Internet of Things, are slowly making their way into many aspects of our lives.

One aspect in which it is growing increasingly prevalent is in the “Green Movement.” A term associated with environmentalism, the “Green Movement” describes an international effort to protect the Earth’s ecosystems from destruction.[1] Although a social movement to conserve the environment has existed in the United States since the 1970’s, the modern day “Green Movement” focuses on minimizing the effects of climate change and conserving finite resources we consume for energy.[1] The Internet of Things and small computer systems have provided those who support the movement with the tools to better conserve energy. Not only are companies and government programs able to collect more complex data pertaining to energy usage, but devices now have a method of adjusting their consumption on an individual basis. This allows them to act efficiently in their dynamic environments. For example, many cities have begun implementing systems that control the intensity of street lights based on the number of people within its proximity.[2] While this is likely the tip of the iceberg in terms of “smart” devices that take a green initiative, there have been many systems developed to address energy conservation on a smaller scale.[3, 4, 5]

Up to this point, individuals invested in the “Green Movement” have primarily linked small computer systems to the Internet as a means of controlling electronics remotely. For instance, many people have developed Raspberry Pi-based systems that allows for control of all of the lights in their home. An example of this can be seen in the open source code produced by Lelylan Laboratories, which turns Raspberry Pi devices into a “smart switch” for a light source.[6] When saved locally to the RPi, users can log onto Lelylan’s web-based user dashboard and alter the state of lights that are connected to the “smart switch.”[6] The system grants users the convenience of conserving light energy regardless of geographic location.[6] However, Lelylan’s system lacks the structure to conserve a maximum amount of energy. Energy wasted on lighting is often due to people being unaware that they left an unneeded light on. To expect them to recognize their mistake and find the time to log into a website amidst their busy schedule is simply unrealistic. Similarly, a system that relies on visiting a URL to toggle the state of lights has also been developed.[7] This form of light control uses a web-based framework known as Flask to turn the RPi into a virtual switch.[7] Each time the user visits the specified URL, Flask triggers the execution of locally-stored Python code to toggle the state of the light.[7] While web-based projects such as this contribute to conserving light energy, the requirement of receiving explicit user input hinders its impact. More particularly, having to remember the URL or access a shortcut via a web browser is both time consuming and inconvenient for those who are not tech-savvy. In order to receive the full benefits of a “smart device,” an autonomous system would need to be developed so that lights are never left on when they are not needed. An effective autonomous system would not only guarantee that lights aren’t left on unnecessarily, but would also minimize the amount of user interaction required to do so.

Unfortunately, an autonomous system is of great need as light pollution has become a major issue across the world. Currently, lighting is responsible for one-fourth of all electricity consumption worldwide.[8] In Australia, the government reports that light energy is the largest source of greenhouse gas emissions in the country.[8] The United States is no exception to this worldwide trend, as one-third of all lighting in the country is wasted.[9] This amounts to the unnecessary release of 14.1 million tons of carbon dioxide into the atmosphere.[9] With efforts already being made to relieve the ozone layer of greenhouse gas emissions, limiting the amount released due to unnecessary lighting is imperative in order to shape the future for a sustainable environment.

To begin addressing the ongoing problem of wasting resources on unnecessary lighting, a Raspberry Pi 2-based autonomous home lighting system was proposed. The system's autonomy is derived from the notion that many homeowners now utilize online calendars to allot their time each day. By obtaining calendar data in the form event alerts and then storing it locally, a concrete determination can be made as to whether lights should be on or off at any given time. Once the data is analyzed, the Raspberry Pi 2 can adjust the state of the lights if the determined and current state differ. Such a system maximizes the amount of energy conserved while requiring minimal user input when compared to existing light conservation systems.

This paper is composed of seven different sections in the following order: the abstract, introduction, proposed system, implementation, conclusions and future work, and references. Statistics and facts relating to the motivation behind the project, in addition to a detailed analysis of the aforementioned components are covered in the "Introduction." "The Proposed System" section provides an overview of the three phases of implementation, explaining why each system component is ideal for the project and describes their role in each phase. The methodology behind accomplishing the goal of each phase is presented in "The Implementation" section alongside the programming techniques used to create the prototype. "Conclusions and Future Work" discusses the original contribution that the prototype makes to energy conservation and the potential contributions that could be made to increase its efficiency.

System Architecture

Below, we give an overview of the proposed system in the prior section. We also offer a justification as to why we chose to use Google Calendar and Raspberry Pi in the proceeding (second) sub-section. Finally, we explain the details of each facet of the proposed system in the third sub-section.

The Overview of the Proposed System

The implementation of the Raspberry Pi-based autonomous home lighting system was broken into three phases as in Figure 1: (I) retrieving events from a user's Google calendar, (II) processing the data and (III) checking/adjusting the state of the lights (if necessary). During the first phrase, the Raspberry Pi uses a PHP script to extract Google calendar events that are sent to the device in the form of an e-mail notification. The pertinent information from the notifications are stored locally on the Raspberry Pi and then assessed in phase two. In this phase, a Python script analyzes each received event and determines what the state of the light should be when that event begins. The third and final step takes the determination and compares it against the current state of the light. If the state of the light differs from what the user indicated it should be, an adjustment (turning the light on/off) is made.

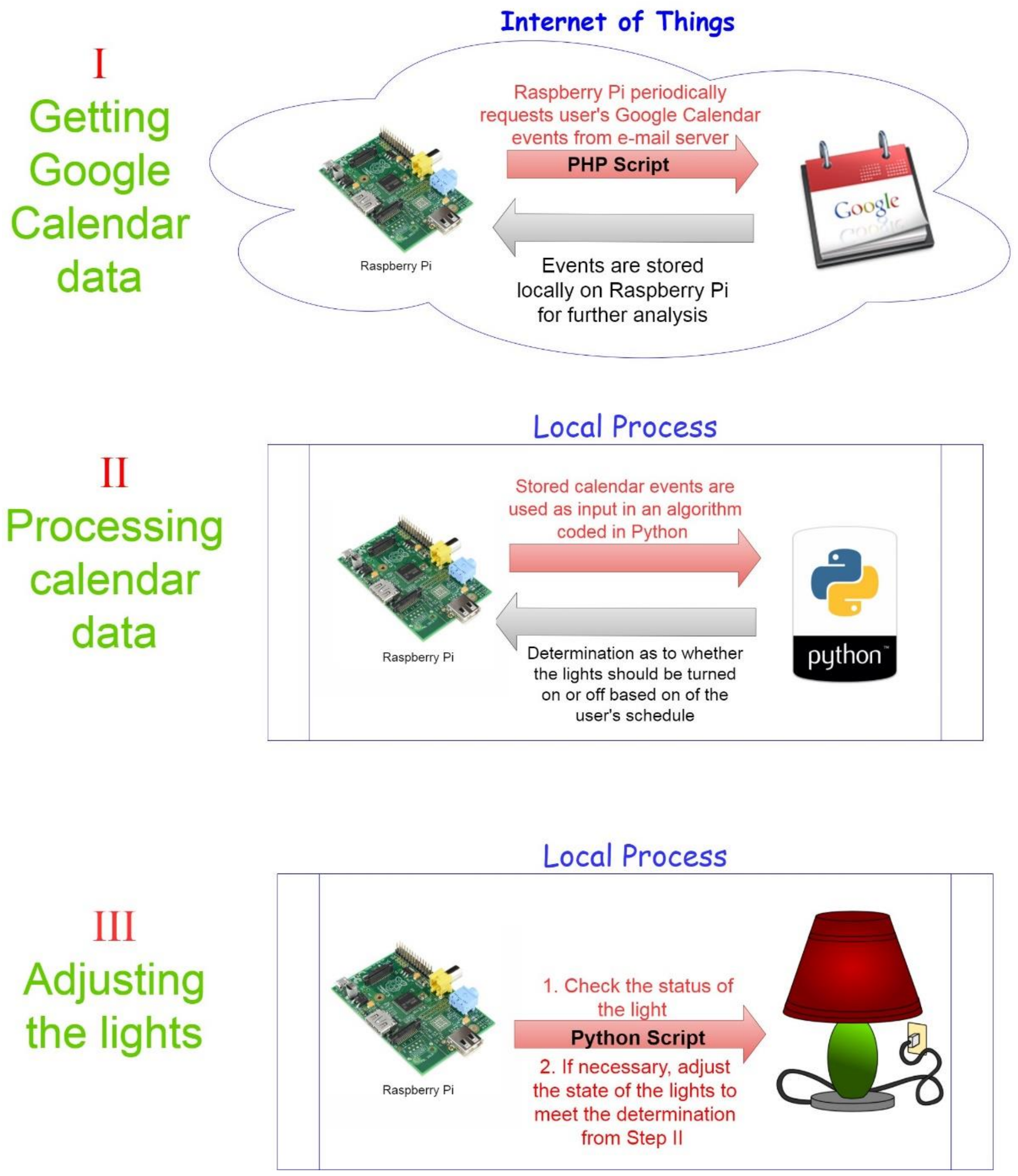
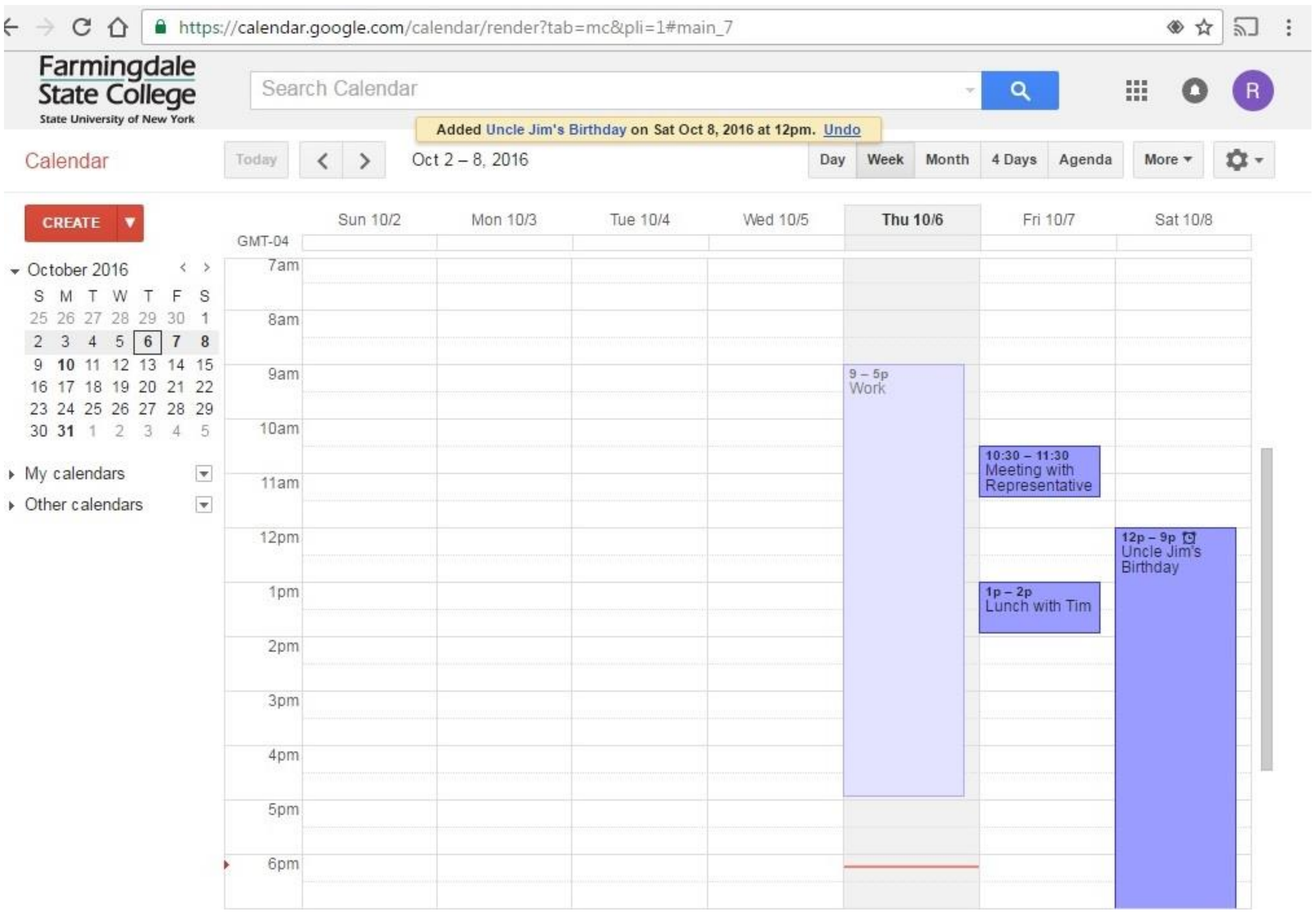

Internet of Things
I
Getting Google Calendar data
Raspberry Pi periodically requests user's Google Calendar events from e-mail server
PHP Script
Events are stored locally on Raspberry Pi for further analysis
Raspberry Pi
Google
Local Process
II
Processing calendar data
Stored calendar events are used as input in an algorithm coded in Python
Determination as to whether the lights should be turned on or off based on of the user's schedule
Raspberry Pi
python™
Local Process
III
Adjusting the lights
1. Check the status of the light
Python Script
2. If necessary, adjust the state of the lights to meet the determination from Step II
Raspberry Pi

Figure 1. Visualization of the Three Proposed Phases of Implementation - A schematic exemplifying the role that each system component plays in retrieving and analyzing Google calendar data to alter the state of a light source.

More on the Google Calendar and Raspberry Pi

It was decided that the system would be designed using Google Calendar primarily due to its popularity and the fact that is entirely web-based. From a technical standpoint, Google Calendar is easy to design around as there are several methods of data extraction. The fact that all of its information is contained on a single web page (as seen in Figure 2) means that it could be extracted directly from HTML code and saved locally as a different file type. There are plenty of web-based open source tools, such as those that convert Google Calendars to Comma Separated Files, which allow for a seamless extraction. In addition, the calendar also has an ability to send out alerts for each event over e-mail. The endless options for extraction guaranteed that the Raspberry Pi's processing software could be designed in a variety of ways and be easily repaired in the event that Google pushed a major update. Another major reason for using Google Calendar is that it can import calendars from other software and web services. People often have separate calendars on their work network or share a calendar with members of their home. Having the ability to import all of the needed calendars to a single account would ultimately allow the system to operate with greater efficiency.


https://calendar.google.com/calendar/render?tab=mc&pli=1#main_7

Added Uncle Jim's Birthday on Sat Oct 8, 2016 at 12pm. Undo

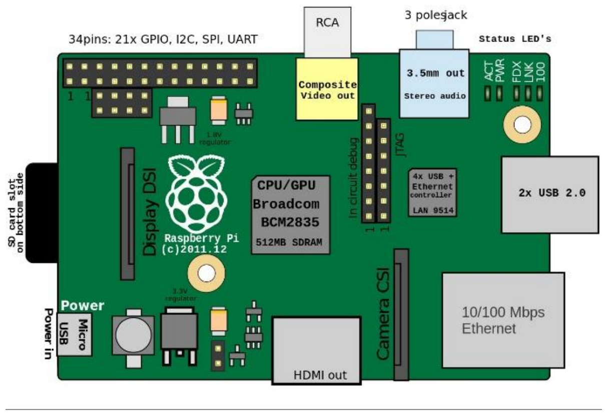
Figure 2. Google Calendar's User Interface - The main screen of the Google Calendar web page which displays its intuitive and user-friendly interface for viewing a user's upcoming events.

The second entity of the system, the Raspberry Pi 2 (Figure 3), was an ideal small computer system for the project for two major reasons. The first, is the ability of the system to interpret and execute programs written in Python. Python has many pre-written libraries that are meant to deal with electrical engineering projects, thus making it an adequate language for an autonomous home lighting system. To elaborate, the LED lights that the Raspberry Pi 2 is capable of interacting with are connected to a breadboard via the GPIO pins seen in Figure 3. For instance, a pre-compiled library called "GPIO" contains class functions that allows for programmers to check and alter the state of the LED lights with a few simple lines of code. The second reason why the Raspberry Pi 2 was an ideal computer system was that it runs off of a UNIX-based operating system. The operating system, named Raspbian, is based off of a popular Linux distribution known as Debian.[10] Raspbian not only allows for the installation of Python development tools, but also all of the other applications associated with Linux operating systems. For instance, Raspbian has the CRON script scheduler, a tool that allows for a user to set when and/or how often a particular script can be run by the system. Moreover, having a Linux-like operating system was useful for extracting information from the web-based Google calendar. Since server-side scripting was required for the autonomous lighting system, possessing an operating system that can feasibly run Apache, PHP and a web browser was necessary.

Details of Each Facet of the Proposed System

As visualized in Figure 1, the first phase of implementation requires the use of a server-side scripting language, such as PHP, to extract a user's calendar notifications that are sent to a Raspberry Pi-exclusive e-mail address. Once all of the necessary data is retrieved, it will be formatted for easy analysis and then saved locally to the RPi's storage. From there, the second phase would be to execute a local script using Python to analyze the extracted data. More specifically, the Python script would determine what state the lights must be in at a particular time based off of the information from the calendar notification and save this information. Finally, in the third phase of implementation, the decision made by the script in Phase II will be put into action. This will be accomplished by executing a Python script that will either turn the lights on or off at the time specified from the results of the second phase.

(a)



(b)

Figure 3. Internal Components of the Raspberry Pi 2 - (a) The credit card-sized Raspberry Pi 2 Model B[11] (b) a diagram highlighting the major internal components of Raspberry Pi.[12]

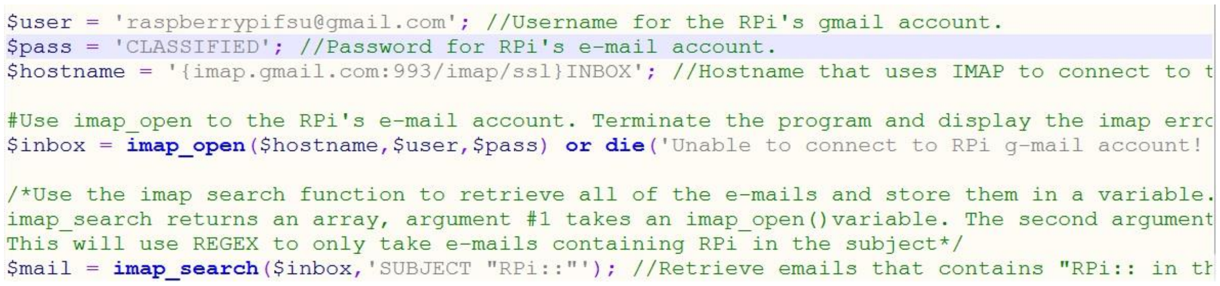
The Implementation

In this section we talk about the implementation details of Phase I, II, and III of the proposed system, in the following three sub-sections, respectively.

Implementation of Phase I

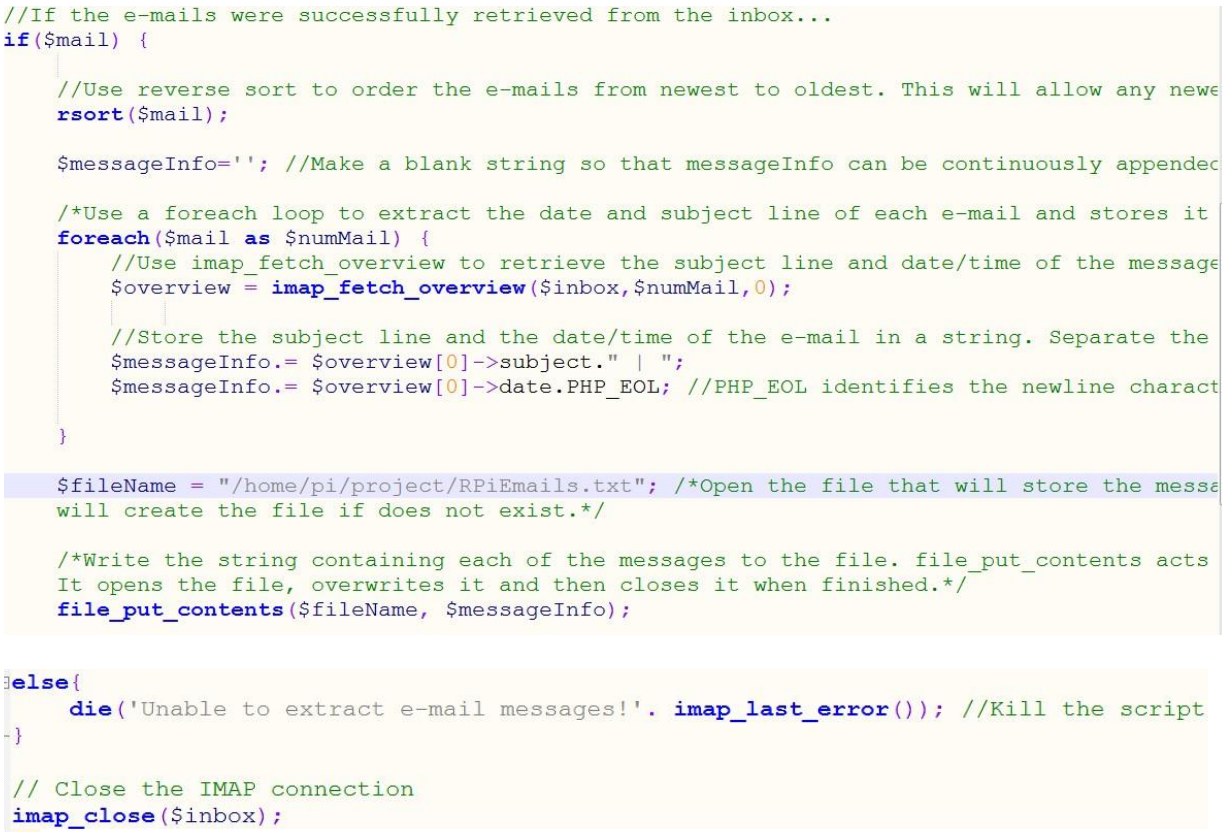
For the first of the aforementioned phases, it was decided that it would be best to retrieve a user's calendar events through e-mail. In terms of advantages, this method enhanced the security and efficiency of the system in comparison to alternative methods. By setting up an e-mail account exclusive to the Raspberry Pi, Google calendar events could be received in the subject lines of e-mails without having to store any user credentials. This came with a small trade-off regarding the ease of use for the user. Since direct access to their account is not granted to the Raspberry Pi, the user will have to modify the settings of their Google account to send each calendar event in the form of an e-mail notification to the device. Moreover, the user must include the strings "RPi::On" or "RPi::Off" in the event name as a means for disambiguation. This allows for events that are intended to check or modify the status of lights to be discerned from those that have no significance. Both of these actions that the end user is required to take also relieves the Raspberry Pi from other arduous methods of Google calendar extraction, such as continually selecting necessary information from the calendar's HTML page.

Regarding the first phase of implementation, receiving a Google calendar event notification e-mail really only completes half the task. The next step was to retrieve the pertinent messages from the e-mail server and store them locally (on the Raspberry Pi). The server-side scripting language PHP was ideal for this due to its complex built-in libraries designed for managing e-mail accounts. In order to be included in the project, the installation of PHP and Apache on the device was necessary. Both of these pieces of software allowed for PHP's Internet Message Access Protocol (IMAP) library to be used to connect to the Raspberry Pi's e-mail account (Figure 4). With an established connection, IMAP's search function was used to only extract e-mails that contained the string "RPi::". Left with the e-mails intended to control the state of the lights, the next step was to use the "imap_fetch_overview" function to extract the header information from each relevant e-mail (Figure 5). Among this header information was the subject of the e-mail and the date/time it was sent. These values were concatenated into a single string and then written to a text file stored in the root directory of the Raspberry Pi (Figure 6). To prevent duplicate events from being listed, the text file was overwritten each time the script was executed.



```
$user = 'raspberrypifsu@gmail.com'; //Username for the RPi's gmail account.
$pass = 'CLASSIFIED'; //Password for RPi's e-mail account.
$hostname = '{imap.gmail.com:993/imap/ssl}INBOX'; //Hostname that uses IMAP to connect to t

#Use imap_open to the RPi's e-mail account. Terminate the program and display the imap errc
$inbox = imap_open($hostname,$user,$pass) or die('Unable to connect to RPi g-mail account!

/*Use the imap search function to retrieve all of the e-mails and store them in a variable.
imap_search returns an array, argument #1 takes an imap_open()variable. The second argument
This will use REGEX to only take e-mails containing RPi in the subject*/
$mail = imap_search($inbox,'SUBJECT "RPi::"'); //Retrieve emails that contains "RPi:: in th
```

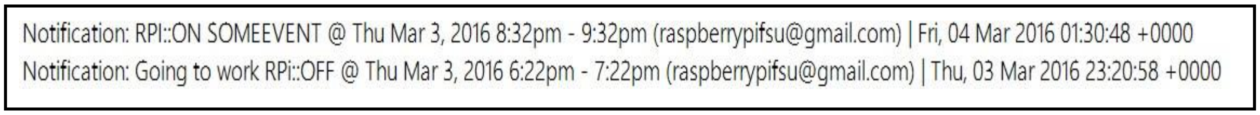
Figure 4. Gathering Google Calendar Events - PHP code executed by the RPi to grant access to its Gmail account containing user event alerts.



```php
//If the e-mails were successfully retrieved from the inbox...
if($mail) {

    //Use reverse sort to order the e-mails from newest to oldest. This will allow any newe
    rsort($mail);

    $messageInfo=''; //Make a blank string so that messageInfo can be continuously appendec

    /*Use a foreach loop to extract the date and subject line of each e-mail and stores it
    foreach($mail as $numMail) {
        //Use imap_fetch_overview to retrieve the subject line and date/time of the message
        $overview = imap_fetch_overview($inbox,$numMail,0);

        //Store the subject line and the date/time of the e-mail in a string. Separate the
        $messageInfo.= $overview[0]->subject." | ";
        $messageInfo.= $overview[0]->date.PHP_EOL; //PHP_EOL identifies the newline charact

    }

    $fileName = "/home/pi/project/RPiEmails.txt"; /*Open the file that will store the messa
    will create the file if does not exist.*/

    /*Write the string containing each of the messages to the file. file_put_contents acts
    It opens the file, overwrites it and then closes it when finished.*/
    file_put_contents($fileName, $messageInfo);
```

```php
else{
     die('Unable to extract e-mail messages!'. imap_last_error()); //Kill the script
}

// Close the IMAP connection
imap_close($inbox);
```

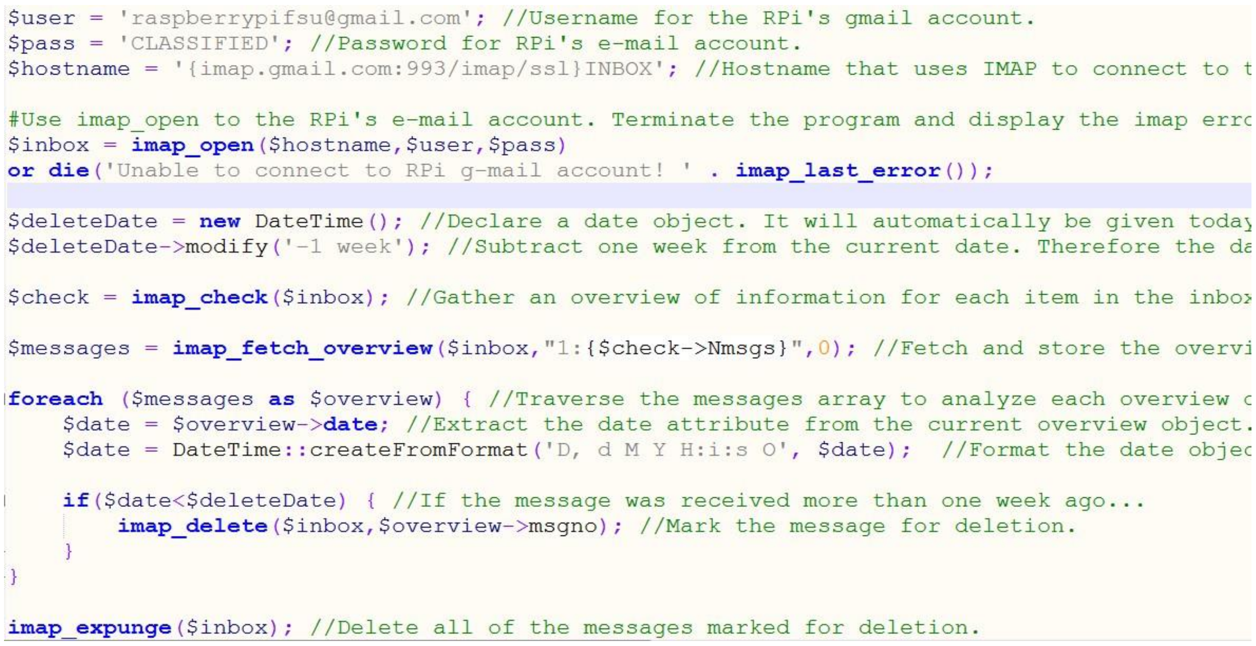
Figure 5. Extracting and Saving Event Information - PHP code responsible for extracting the subject and date information from e-mail event alerts and writing them to a text file.



```
Notification: RPI::ON SOMEEVENT @ Thu Mar 3, 2016 8:32pm - 9:32pm (raspberrypifsu@gmail.com) | Fri, 04 Mar 2016 01:30:48 +0000
Notification: Going to work RPi::OFF @ Thu Mar 3, 2016 6:22pm - 7:22pm (raspberrypifsu@gmail.com) | Thu, 03 Mar 2016 23:20:58 +0000
```

Figure 6. Appearance of Locally-stored Header Information - Samples of header information extracted from e-mail event alerts after being written to a text file.

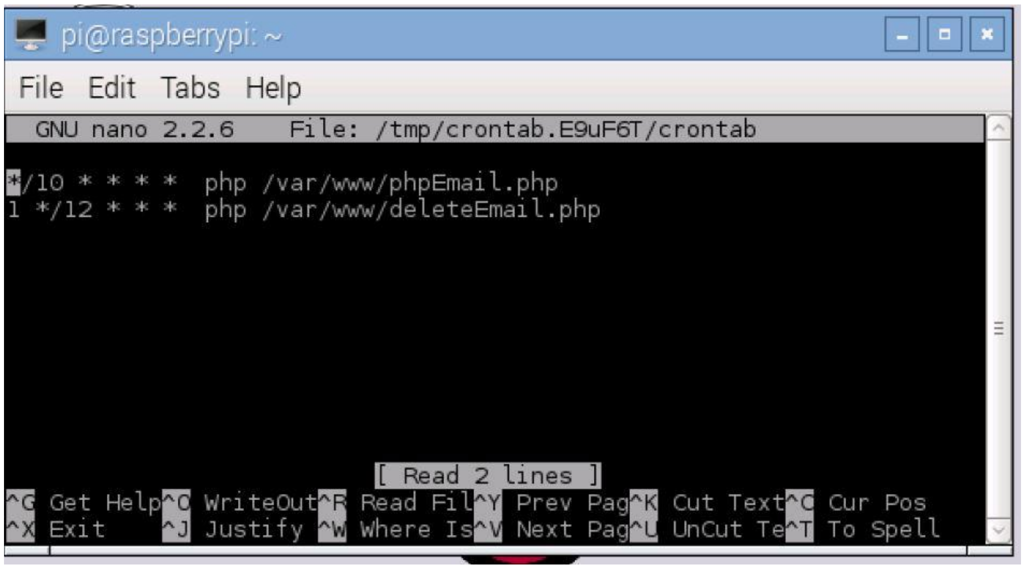
In order to assure that old e-mails weren't being extracted and the inbox never became full, a helper script was coded to delete unnecessary messages (Figure 7). This PHP script connected to the e-mail server and extracted information similar to the script in Figure 4. However, the date and time found in each header was compared to the current date and time indicated by the system. If a particular e-mail was a week or older, it was marked for deletion and then deleted from the server using the "imap_delete" and "imap_expunge" functions respectively.



```
$user = 'raspberrypifsu@gmail.com'; //Username for the RPi's gmail account.
$pass = 'CLASSIFIED'; //Password for RPi's e-mail account.
$hostname = '{imap.gmail.com:993/imap/ssl}INBOX'; //Hostname that uses IMAP to connect to t

#Use imap_open to the RPi's e-mail account. Terminate the program and display the imap erro
$inbox = imap_open($hostname,$user,$pass)
or die('Unable to connect to RPi g-mail account! ' . imap_last_error());

$deleteDate = new DateTime(); //Declare a date object. It will automatically be given today
$deleteDate->modify('-1 week'); //Subtract one week from the current date. Therefore the da

$check = imap_check($inbox); //Gather an overview of information for each item in the inbox

$messages = imap_fetch_overview($inbox,"1:{$check->Nmsgs}",0); //Fetch and store the overvi

foreach ($messages as $overview) { //Traverse the messages array to analyze each overview o
    $date = $overview->date; //Extract the date attribute from the current overview object.
    $date = DateTime::createFromFormat('D, d M Y H:i:s O', $date);  //Format the date objec

    if($date<$deleteDate) { //If the message was received more than one week ago...
        imap_delete($inbox,$overview->msgno); //Mark the message for deletion.
    }
}

imap_expunge($inbox); //Delete all of the messages marked for deletion.
```

Figure 7. Deleting Old Event Alerts - PHP script used to delete e-mails that are a week or older from the RPi's Gmail account.

Finally, the PHP scripts in Figure 4, 5, and 7 were set to run by the system automatically. This was done using a tool known as Cron. Found in virtually all Linux distributions, the embedded package references a locally-stored file ("crontab") that has a list of coded times and an associated shell command. When the system time matches a time indicated in the file, the associated shell command is executed in the background. Through the use of this tool, the PHP script that was used to retrieve the e-mails was set to execute every ten minutes (Figure 8). The script that deleted all of the old e-mails in the Raspberry Pi's e-mail inbox was set to execute every twelve hours. This provided a balance that would assure nearly real-time accuracy for the state of the lights while leaving notifications that are sent well in advance with time to be executed.

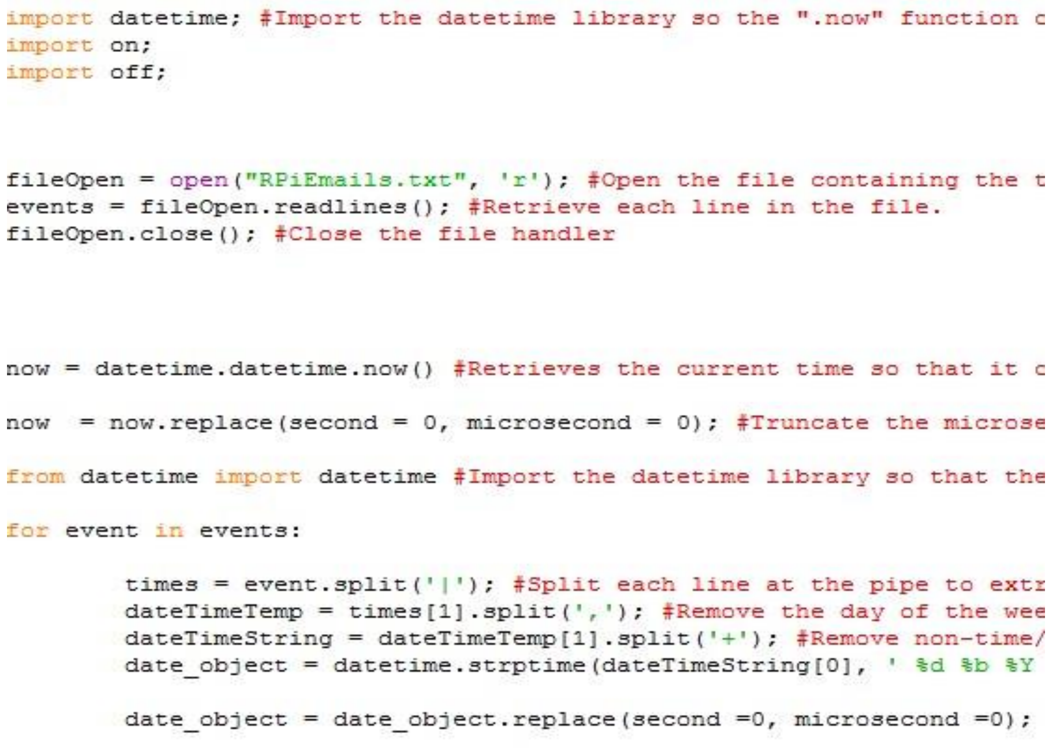


```
  GNU nano 2.2.6    File: /tmp/crontab.E9uF6T/crontab

*/10 * * * *  php /var/www/phpEmail.php
1 */12 * * *  php /var/www/deleteEmail.php




                         [ Read 2 lines ]
^G Get Help^O WriteOut^R Read Fil^Y Prev Pag^K Cut Text^C Cur Pos
^X Exit    ^J Justify ^W Where Is^V Next Pag^U UnCut Te^T To Spell
```

Figure 8. Scheduling E-mails to be Retrieved and Deleted -The two lines of code found in the RPi's crontab file to retrieve e-mail notifications every ten minutes and to delete old e-mails off of the Gmail account every 12 hours respectively.

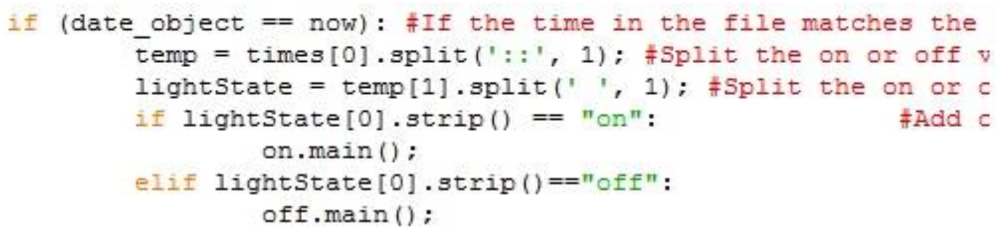
Implementation of Phase II

Part two of the implementation relied primarily on processing the resulting text file from the first phase of implementation. After e-mails have been retrieved from the Raspberry Pi's account, the text file contains the event name (containing "RPi::On" or "RPi::Off") and the date/time that the event begins. Both of these pieces of information now need to be processed to see if any action is necessary. This was made possible by utilizing a Python script to extract these strings from the text file and then make explicit comparisons. The date from each e-mail in the text file was converted to a Python "datetime" object, and then compared to the current date and time indicated by the system (Figure 9). If the date and times matched, then the "RPi::" statement found in the event name was compared to the current state of the lights (Figure 10). For instance, if the user indicated they needed the lights turned off (by using "RPi::Off" in the event name) and the lights were already off, then no action was taken. Conversely, if the lights were currently turned on, then a script would be executed to turn them off. It was decided that these parameters should be analyzed locally during phase two rather than on the e-mail server during phase one. The primary reason for this was that the coded scripts that alter the state of the lights had already been written in Python. Integrating their execution into another Python script would prevent more potential issues than having the PHP script for phase one execute them. Moreover, processing the data locally allows for the Raspberry Pi to interact with the e-mail sever minimally. This not only prevents potential security errors, but also allows for the system to continue to operate if Internet connection is temporarily lost. Similar to the aforementioned PHP scripts, this Python script was set to be executed automatically via CRON. Considering the amount of time an event notification is received prior to its start is unpredictable, it was decided that this script should execute every minute.



```
import datetime; #Import the datetime library so the ".now" function c
import on;
import off;



fileOpen = open("RPiEmails.txt", 'r'); #Open the file containing the t
events = fileOpen.readlines(); #Retrieve each line in the file.
fileOpen.close(); #Close the file handler



now = datetime.datetime.now() #Retrieves the current time so that it c

now  = now.replace(second = 0, microsecond = 0); #Truncate the microse

from datetime import datetime #Import the datetime library so that the

for event in events:

        times = event.split('|'); #Split each line at the pipe to extr
        dateTimeTemp = times[1].split(','); #Remove the day of the wee
        dateTimeString = dateTimeTemp[1].split('+'); #Remove non-time/
        date_object = datetime.strptime(dateTimeString[0], ' %d %b %Y

        date_object = date_object.replace(second =0, microsecond =0);
```

Figure 9. Preparing Event Times for Comparison - The part of the Phase II Python script responsible for converting the time of calendar events and the current time to datetime objects for comparison.

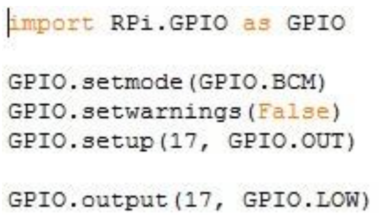


```
if (date_object == now): #If the time in the file matches the
        temp = times[0].split('::', 1); #Split the on or off v
        lightState = temp[1].split(' ', 1); #Split the on or c
        if lightState[0].strip() == "on":              #Add c
                on.main();
        elif lightState[0].strip()=="off":
                off.main();
```

Figure 10. Adjusting the Current State of the Lights - Python code that compares the current and calendar event times and changes the state of the lights if deemed necessary.

Implementation of Phase III

With a suitable method to determine what state the light should be in, the third and final step was to code scripts to change the state of the lights. This required using the aforementioned Python GPIO (General Purpose Input/Output) library to code two separate scripts: one to turn the lights off (Figure 11) and another to turn them on (Figure 12). Both of these scripts utilized the GPIO library's output function in order to adjust the status of the lights. At any given the moment, the GPIO pin located in a particular position on the breadboard is either in a "high" or "low" state. A pin that is in a "high" state will allow an electric current to pass through to its attached device (which in this case, is an LED light). Conversely, a pin in a "low" state will allow no current to pass through, thus rendering the attached device off. These high and low states can be controlled programmatically through the output function. Implementing state alterations in this way was not only simple, but also served as a practical means for ease of future expansions. Since the state of the light is controlled by restricting or allowing the flow of an electrical current, utilizing the system to control other electronics such as an air conditioning unit could be seemingly integrated with little to no changes.



```
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(17, GPIO.OUT)

GPIO.output(17, GPIO.LOW)
```

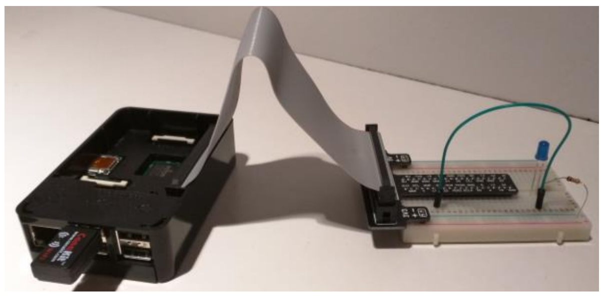
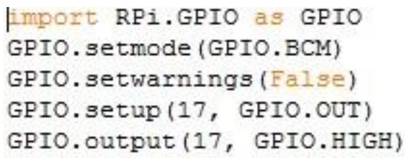


Figure 11. Turning off the Lights - Python code found in the "off" script and the appearance of the prototype when the script is executed.



```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(17, GPIO.OUT)
GPIO.output(17, GPIO.HIGH)
```

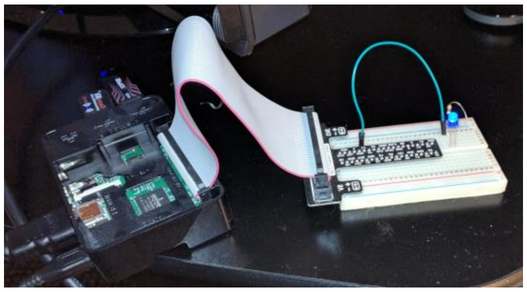


Figure 12. Turning on the Lights - Python code found in the “on” script and the appearance of the prototype when the script is executed.

Conclusions and Future Work

Although the fully implemented system is only a prototype, its effectiveness in minimizing light pollution can ultimately contribute to conserving our planet’s dwindling resources. While the system’s autonomy is granted through a variety of algorithms, this would not be possible without utilizing the Raspberry Pi 2 and Google Calendar’s capabilities. The Raspberry Pi 2 allows the system to behave as an object within the Internet of Things and served as a means of communication for scripts coded in different languages. Google Calendar’s extremely popular online tool allowed for a seamless extraction of all necessary data while minimizing the inconvenience users would have to face for the system to be successful. The prototype’s three phases of implementation: I) having the RPi gather data from the user’s Google calendar, II) assessing the data to determine the ideal state of the lights, and III) then adjusting the state of the lights based on this determination, are divided in such a way that epitomizes the value of the project. By isolating most of the system’s autonomy in the first phase of implementation, new algorithms can be added to the latter two phases without compromising what makes the prototype unique. In other words, the prototype serves as a base for a system that can become increasingly efficient in preserving energy without sacrificing autonomy.

When thought of as a metaphorical building block, it appears that there are a multitude of future projects that can be undertaken to use the system to its potential. For example, up to this point, the system has been verified to be working with a single LED light attached to the Raspberry Pi’s breadboard. Logically, the next step would be to test this autonomous lighting system on a commonly used light source, such as a lamp. From here, the system could be expanded to support several light sources. By prohibiting or allowing electrical currents to multiple GPIO pins simultaneously, the number of lights that could be controlled at the same time would only be limited only by the electrical power distribution of the Raspberry Pi. As was previously mentioned, there is room for algorithmic improvements in regards to the system’s autonomic nature. The system currently relies on searching for calendar events containing the string “RPi::On” or “RPi::Off” to make decisions. However, if enough data is collected after the system has been tested by several people, it may be possible to improve the algorithm so that this step is no longer needed. For instance, the system could be coded to detect which events consistently contained “RPi::Off” its subject line and keep record of this on a user-by-user basis. At this juncture, it is unclear as to what algorithm would produce a higher level of autonomy without a tradeoff in accuracy. Regardless, the best potential algorithm(s) for enhanced autonomy will become lucid as the system is used more frequently by users.

References

[1] Dykstra, P. (2008) “History of environmental movement full of twists, turns.” *Cable News Network*. Cable News Network. http://www.cnn.com/2008/TECH/science/12/10/history.environmental.movement/index.html. (Accessed 09/26/2016).

[2] Zanella, A., Bui, N., Zorzi, M., Castellani, A., and Vangelista, L. (2014) “Internet of Things for Smart Cities” *IEEE INTERNET OF THINGS JOURNAL*. 1.1. Pg 22-32.

[3] Klosowski, T. (2013) “Use a Raspberry Pi to Automate Your Blinds and Air Conditioner.” *Lifehacker*. Univision Communications. https://lifehacker.com/use-a-raspberry-pi-to-automate-your-blinds-and-air-cond-1479598714. (Accessed 09/28/2016).

[4] Hartley, T. (2013) “What is the AirPi?” *AirPi*. http://airpi.esphp/whatisthis. (Accessed 09/28/2016).

[5] N.A. (2012) “Flood/Water Presence Sensor.” *PrivateEyePi*. http://projects.privateeyepi.com/home/home-alarm-system-project/wireless-projects/flood-water-presence-sensor. (Accessed 09/27/2016).

[6] N.A. (2013) “Create your own smart light using Raspberry Pi.” *Lelylan Lab*. Lelylan. https://lelylan.github.io/lab-projects/raspberry-pi-light/. (Accessed 09/27/2016).

[7] Minardi, J. (2013) “Make an Internet Controlled Lamp with a Raspberry Pi and Flask.” *Jack Minardi*. http://jack.minardi.org/raspberry_pi/make-an-internet-controlled-lamp-with-a-raspberry-pi-and-flask/. (Accessed 09/27/2016).

[8] N.A. (2009) “What is Light Pollution?” *Globe at Night*. Association of Universities for Research in Astronomy. https://www.globeatnight.org/light-pollution.php. (Accessed 11/09/2015).

[9] N.A. (2012) “Light Pollution Hurts Our Economy and Our Resources.” *FAU Department of Physics*. Florida Atlantic University. http://physics.fau.edu/observatory/lightpol-econ.html. (Accessed 11/09/2015).

[10] N.A. (2013) “About Raspbian.” *Raspbian*. https://www.raspbian.org/RaspbianAbout. (Accessed 08/20/2016).

[11] N.A. (2015) “Raspberry Pi 2 Model B.” *Canada Robotix*. Carobotix, INC. http://www.canadarobotix.com/image/cache/data/products/1400/1426-2-800x800.jpg. (Accessed 10/31/2016).

[12] N.A. (2015) “Raspberry Pi 2.” *Revolvy*. Revolvy, LLC. https://www.revolvy.com/main/index.php?s=Raspberry%20Pi%202&item_type=topic. (Accessed 10/31/2016).