EECS 149/249A: Smart AC Lighting

Milo Darling, Rohan Sagar, Henry Teng, Daniel Zhang
milodarling@berkeley.edu,rohansagar4@gmail.com,hteng1995,daniel.zhang@berkeley.edu

ABSTRACT

This report explains in detail the design, implementation, and evaluation of a "smart" AC lighting system. The system will employ a Bluetooth Low Energy network of Nordic NRF52 Development Boards linked to a central Raspberry Pi Zero to allow a user to control the dimness of any given lightbulb in the system. A website will input user commands, and various other sensors will provide the "smart" features of our design.

1 INTRODUCTION

Problem and Goals

Energy efficiency in the common household is a theme that has become strong in the past few decades, and the advent of smaller, cheaper, and powerful computing units have helped this field grow quickly. Our group wanted to approach a common problem seen in the household: how to easily control the brightness of each room in a house, beyond traditional light switches and slide controls.

We want a system that can be deployed in multiple rooms in a house, where the user has the ability to control each room from a central hub and enable "smart" features such as motion detection and light tracking to control lighting beyond direct human command. Our system will be judged according to three standards:

(1) Timeliness: Our system must have minimal latency, so that the user does not observe noticeable lag in system response to any user input
(2) Accuracy: Our system must not output unexpected response to user or sensor input
(3) Scalability: Our system should be easily deployable in multiple rooms to control multiple lights

Background: AC Dimming

The fundamental concept that underlies our project is AC phase control, which is used to control the amount of power going into an AC circuit (in this case, a light bulb). We refer to Figure 1 to illustrate how AC phase control works, and how we can use it in our project:

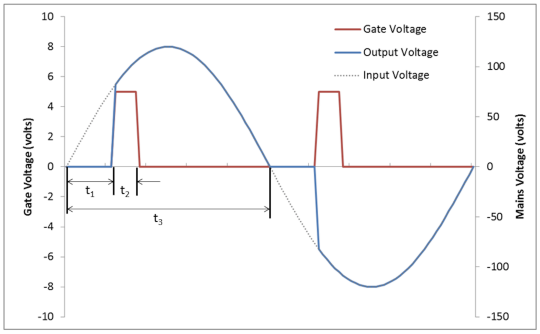

Figure 1: AC Phase Control *(Arduino Playground)*

A typical AC phase control module employs a Triac along with a zero-crossing detector. The zero-crossing detector will throw a pulse whenever the AC signal crosses zero, during which the Triac will wait a certain amount of time `t1` before opening its gate and letting the input voltage through for the rest of the period. The length of `t1` is inversely related to the brightness of the light bulb – the larger `t1`, the less averaged power passes to the bulb, and therefore the more dim the bulb is.

The microcontroller that we use will detect the zero crossing signal, and after a desired amount of time, will command the Triac gate to open.

Relevance to Course

Our system has to interface with various input/output peripherals, including a variety of sensors; both digital and analog. Not including the required use of interrupts and timers in actuating the AC dimming module, the microcontroller will have to implement a schedule of periodic tasks such as interfacing with sensors, sending information over BLE, and making decisions about how to control the lights. We also had to implement scheduled periodic tasks on the Raspberry Pi, avoiding deadlocks or multiple accesses of the same resource. The modelling of our system will be aided by finite state machine tools, as taught in class, and the system will be deployed across a wireless network, Bluetooth Low Energy, as well as WiFi.

2 DESIGN AND IMPLEMENTATION

System Architecture

To best illustrate the various components and connections in our system, we will refer to Figure 2 below, which illustrates our overall system architecture:

EECS 149/249A, December 14, 2018, UC Berkeley
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

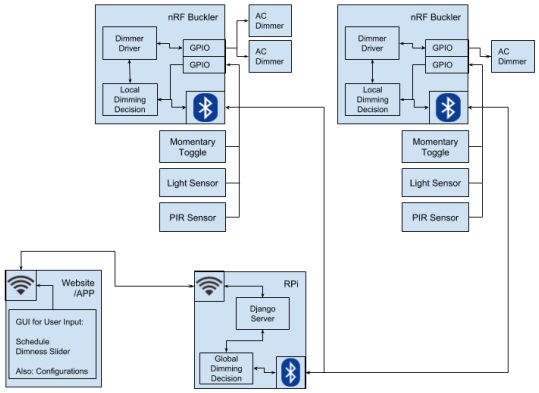

Figure 2: System Architecture

Our system will accept information from various user/sensor inputs:

(1) Schedule [Website]: Minimum/Maximum dimness in a time interval
(2) Virtual Slider [Website]: a two-way virtual slider, in-taking the desired dimness, but also displaying the current dimness of a given light
(3) Manual Toggle Switch [NRF52]: an analog sensor, similar to a car-window toggle switch with three states: up, down, and off, which increments/decrements the dimness of a given light
(4) Ambient Light Sensor [NRF52]: a digital sensor that outputs the ambient light in lux
(5) PIR Motion Detector [NRF52]: an analog infrared sensor that detects motion

In addition to the above Schedule and Virtual Slider, the website will allow users to toggle two "smart" features:

(1) Motion Detection: when the PIR sensor detects motion, the room will be lit up, and after a period of inactivity, will be shut down
(2) Ambient Light Tracking: a NRF52 will set an ambient light point, and will use the ambient light sensor to track that set-point. The set-point can be shifted after initial setting.

The AC Dimming module will be directly actuated by the Nordic NRF52 Development Board, as provided in the course. Each of these AC Dimming - NRF52 pairs acts as a peripheral in our BLE network; each reporting to a central device, a Raspberry Pi Zero, which hosts a server, written in Django. The RPi must also interface with a website, which will be the portal through which the user can input their commands.

The NRF52 must poll 3 sensors (see Figure 2) in order to generate its own "local" dimness command, which will be passed to the AC Dimming module. The NRF52 must also receive "global" commands from the Raspberry Pi. These global commands are sent to the NRF52, which then must resolve both commands to create a single command to actuate the AC dimmer with.

The RPi must resolve the user commands from the website to form a single global dimness command, which is sent to the NRF52, and must also update the website with the most current dimness setting at a NRF52.

Bluetooth LE. A custom BLE lighting service was created to hold all essential information needed to track user inputs, commands, and sources of dimness change (see Figure 3).

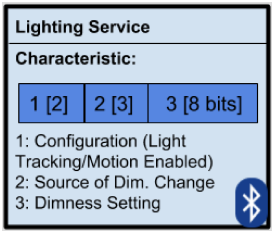

Figure 3: Custom Lighting Bluetooth LE Service

This service contains all information required for the NRF52 to respond to global command and/or perform local decisions and for the RPi to update the website and its server.

Design of NRF52 Machine. The design for the NRF52 is by far the most complicated for any component of the system. We detail its design with a FSM and a decision table:

In C code, all of the states of the FSM in Figure 4, except for `INITIALIZE` and `GLOBAL_DECISION` are implemented in a while-loop. The FSM is both timer and event-driven. The polling tasks that are abstracted into the state `POLL_SENSORS` are linked to software timers, in which timer interrupts will set condition variables in the main NRF52 code, signifying that a sensor should be polled. However, as seen by the preemptive transitions in Figure 4, the main NRF52 code is interrupted when the BLE service receives new data from the Raspberry Pi, and carries out that global command immediately.

After initialization, on each iteration of the main while loop, the NRF52 will check the sensor condition variables, and poll each "ready" sensor. After polling any "ready" sensors, the NRF52 will follow Table 1, using inputs from the BLE service including: a user configuration (turn on Motion Detection and/or Light Tracking), and the source of the change and the sensor value itself, to generate a decision. Table 1 states that global commands (preemptive in nature) have top priority, but the manual toggle, which will generate a local decision, is of equivalent priority due to high sensor polling frequency. This is an intentional design choice, as a user would expect a light switch to respond immediately to their toggling. More details on the implementation of Table 1 can be seen in *Challenges*.

Design of Raspberry Pi Server & Database. The backend of our project is a Django server with WebSockets and a RESTful API.

In order to change the dim level quickly, we use WebSockets. Upon loading the website, a WebSocket is established between the server and client, and any change of the slider causes the client socket to send the new light level to the server. When the server receives a new light level, it saves the new light level to a database,

Input: $\text{Dim}_{Global} \in \{Absent, 0-255\}$ **FROM BLE**, $(config_{light}, config_{motion}) \in \{Absent, Present\}$ **FROM BLE**
Output: $command \in \mathbb{Z} = \{0-255\}$ **TO AC DIMMER & BLE**, $source \in \{slider, schedule, light, manual, motion\}$* **TO BLE**
Variables: $Light_{Set\ Point} \in \mathbb{R}$, $Dim_{Current} \in Z = \{0-255\}$, $Increment \in \mathbb{Z}$, $Direction \in \{Up, Down\}$, $Dim_{Changed} = \{1,0\}$, $source$*
*global sources (slider & schedule) are only read from BLE, local sources (light, manual, motion) are generated in POLL SENSORS and written to BLE

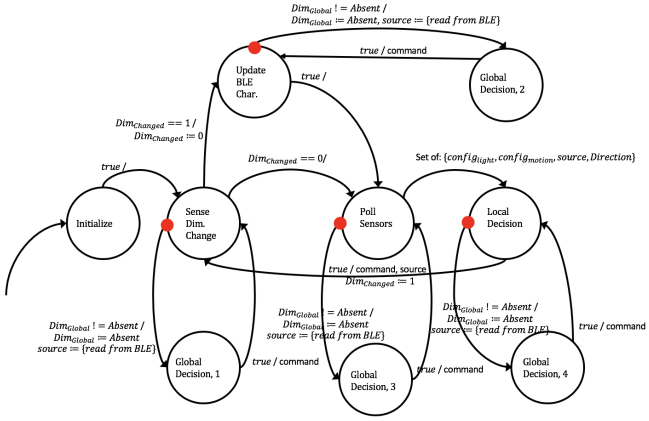

Figure 4: FSM for NRF52 Component

Table 1: Decision and Priority Table for NRF52

	Input				Output	
Priority	Configuration	Direction	Dim_{Global}	Source	Command	Source
1	N/A	N/A	$\mathbb{Z} = \{0-255\}$	App-Slider	Dimness = Global Set	App-Slider
1	N/A	N/A	$\mathbb{Z} = \{0-255\}$	Schedule	Dimness = Global Set	Schedule
1*	N/A	Up	N/A	Manual	Dimness = Dimness - Increment	Manual
1*	N/A	Down	N/A	Manual	Dimness = Dimness + Increment	Manual
2	Motion: ON, Light: N/A	N/A	N/A	Motion	Dimness = Prev. Dimness	Motion
3	Motion: N/A, Light: On	Up	N/A	Light	Dimness = Dimness - Increment	Light
3	Motion: N/A, Light: On	Down	N/A	Light	Dimness = Dimness + Increment	Light

to be sent to the Buckler by our BLE task (explained below in *Design of Communication Pathways*).

We also have RESTful endpoints for specific features, like scheduling. For these features, the information and data models are saved in a separate database from the bluetooth task database. This allows us to use these features without having to persist a connection and a separate websocket. The schedule endpoints allow for CRUD operations, and utilize a crontab library to schedule a python script that communicates via BLE to turn on a light.

Design of Website. The web app was designed to be the only hub an end user needs to access to control their lights. In the web app, users can add new Bucklers as soon as they are booted up, giving them a user-readable name like "Living Room." Users can also directly change the light level of any of their configured lights, set a dimming schedule for the lights, and can enable/disable the "smart" features for each light in their network.

For the website, we took advantage of free and open web design aids, utilizing Bootstrap to quickly create an attractive website, and DHTMLX Scheduler, a free HTML calendar library, to implement the scheduling feature. We used Django's HTML templating and a REST API to load the data about the current light on the website, and used WebSockets and AJAX requests to send new data. We used WebSockets for their speed when necessary, and AJAX elsewhere for its simplicity.

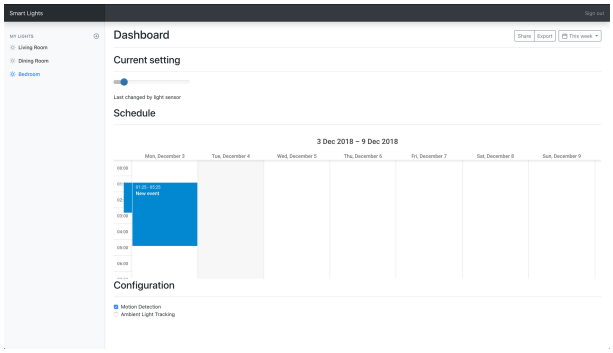

Figure 5: Screenshot of Website

Design of Communication Pathways. Initially, we had used Django endpoints to communicate with the lights. It was easy to integrate directly into the project; however, we found that repeated AJAX requests, in addition to having to fetch the correct BLE device, create

the characteristic, and write it in each request created too much latency, and not enough control. For example, if we received an excessive number of server requests, we could only in turn send an excessive number of BLE requests. In order to improve communication, we put BLE in a separate task on the Raspberry Pi. The BLE task would repeatedly check a SQLite database for new information, i.e. a modified brightness level. If a change was found, the task would write the new information to the corresponding Buckler repeatedly and quickly. While not writing, the BLE task scans and connects to bucklers, reads their light levels, and saves them to the database. With this design, the BLE connection is abstracted away from the server, and the server only handles reads and writes locally on the database. This also made it easy to implement scheduled dimming, as the scheduled task simply needs to write to the database, and the BLE task will handle the rest. Similarly, the server requests are abstracted away from the BLE communication. With BLE concentrated in one task, we were able to control the speed at which the Pi reads and writes from the Bucklers, improving overall latency significantly. This task also allowed us to continuously scan for and connect to new and previously connected Bucklers, making it resilient to random disconnects, power outages, and other issues that can arise with BLE.

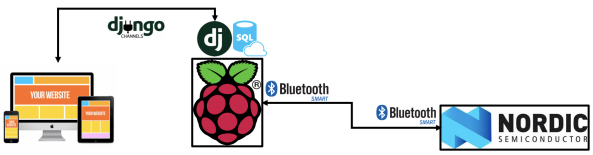

Figure 6: Communications Diagram

Hardware and Software

Hardware. The following hardware was used for our project:

(1) Raspberry Pi Zero
(2) NRF52 Development Boards + Berkeley Buckler
(3) MAX44009 I2C Ambient Light Sensor (Incl. on Buckler)
(4) Momentary Toggle Switch (Amazon Link)
(5) EKMB1101111 PIR Sensor (Supplied by Staff)
(6) AC Dimming Module (Amazon Link)
(7) Misc. cables, light bulbs, light sockets, etc.

Software. The first code written for the project was C code for the NRF52 Buckler to interface with the various sensors and peripherals. Much code was borrowed from lab work concerning the use of GPIOs, setting timers, and using interrupts, but all original work of our team. The only sensor that required additional work outside of the lab curricula was the I2C Ambient Light Sensor. For this, code was originally written based on the code written by staff for the I2C accelerometer on the Buckler, but we ultimately settled for the Staff-written MAX44009 library.

The Django server and SQLlite database were written by the team. The BLE implementation on the RPi was written by the team, using the BluePy Python library, and the task was run by systemd. The BLE implementation on the NRF52 was based extensively on the Staff example, but with slight adjustments to advertising, our own read/write functions and adding our own service.

Code to implement the Django Web Socket connection between the website and server was also written using the Django channels library.

Additional endpoints for certain features, such as scheduling, were implemented with the Django RESTful framework. Also utilized in implementing the schedules feature is the python-crontab library, used to programmatically create Cron jobs. These features were not required to have a persistent connection, and could afford a higher latency.

Photos. With reference to Figure 2, each peripheral of our network is comprised of a nRF52 + Buckler + AC Dimmer. For demonstration, this is packaged in an enclosure that exposes the sensors and protects the electronics, seen below in Figure 7:

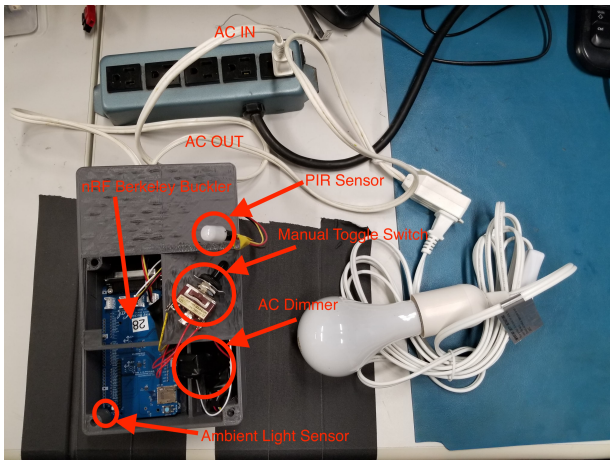

Figure 7: Peripheral Component in Enclosure (Opened Lid)

Our demo showcased a network of 3 peripheral, one central device, and 1 website:

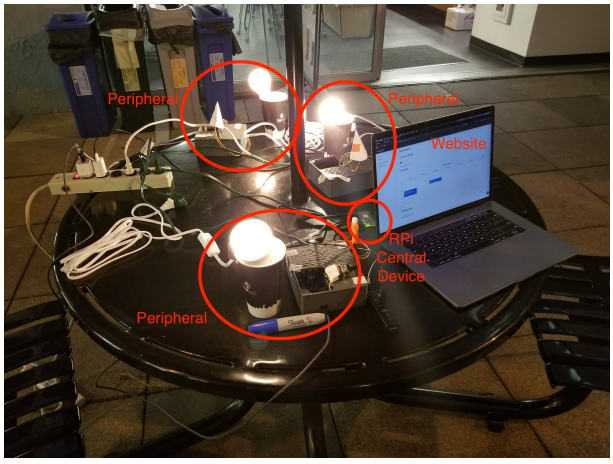

Figure 8: Full Demonstration (Raspberry Pi Server Obscured Behind Computer Screen

Challenges

NRF52 Priorities and Smart Features. The NRF52 must implement the priorities of Table 1 and also accommodate the two "smart" features listed in Page 2. We implemented the priorities tabulated in 1 by polling the sensors at different frequencies: the Manual Switch is polled the most frequently, followed by the motion sensor, followed by the light sensor. We chose these priorities with response time in mind – a manual switch when toggled should respond immediately. Motion detection, when enabled, should react reasonably quick to motion, and light tracking, when enabled, should react the slowest to avoid erratic oscillations in the open-loop tracking of the set-point.

However, issues arose in implementing the smart features. For one, we noted that the PIR sensor, due to its very wide field of view, would often give many "false positives" in our crowded lab setting. This would mean that when motion detection was activated, which turns on the lights upon any motion, the light would turn on even when someone did not walk directly in front of the sensor. We corrected this by adding a 8-bit buffer, which must read all "1"'s in order for the light to turn on. This, however, means that we have at a minimum $8 * t_{sample}$ ms latency in the motion detection reaction. As suggested by the GSIs during our demo, it would be helpful to correct our implementation, such that the light turns on more easily.

For both the light tracking and motion detection features, we had to determine what set-point each feature would either be tracking or setting the light to, respectively. For light tracking, we save an initial set-point to a static variable when the feature is activated on the website, and then update that variable with any new dimness commands that are not due to the light sensor. This way, we can account for a situation where the user, for example, increases the brightness in a room after engaging light tracking, and desires to keep that brightness. For motion tracking, we also use a static variable to keep track of any dimness changes attributed to not the motion sensor. This is then the value that the motion feature will set the lights to from a 0 brightness state when motion is sensed.

Disconnect between Ambient Light and Bulb Dimness. A concept that was not included in our design was this disconnect between the brightness of a light bulb and the ambient light in the room. This is a function that is critical to a more realistic light-tracking smart feature, but requires a model of the environment the system is deployed in. This is outside the scope of our project, but we remedy this problem during our evaluation trials by placing the bulb closer to the light sensor, so that a change in dimness of the bulb is easily caught by the ambient light sensor.

3 EVALUATION

We evaluate our project by the three metric introduced in the section: *Problem and Goals*, which are: Timeliness, Accuracy, and Scalability. For reference, our a video of our demo is available at: Link to Demo Video

Timeliness. This was a critical section of our project, as the brightness of a room is something that many people are accustomed to having direct and rapid control over in their households or workspaces. Since much of our design and implementation considered the user in mind, this standard was met rather well (for details, see the *Challenges* section). The manual switch on each peripheral reacts near instantaneously, with each toggle up or down reflected by the bulb immediately after the input is given. The smart features (a.k.a. the PIR sensors and light sensor) have latency on the order of seconds, but that is by design due to the nature of the feature itself, and is to be expected. On the website side, the schedule has latency on the order of a few seconds, with a command stored in the database being actuated on within seconds of the time stamp. The virtual slider was the greatest challenge with the least success. A slow press-and-slide of the slider results in reasonable latency (on the order of a second or two), as each value read from the slider having reasonable time to be sent to the NRF52 before the next value is read. This pathway becomes flooded if the slider is moved faster, but the results are still reasonable, with visible levels seen in the brightness of the bulb, but latency still on the order of a few seconds.

Accuracy. We validated the smart features independently. The motion detection feature triggering as expected (with a delay due to the 8-bit buffer required for the PIR sensor) and then turning off the lights after a duration of time of stillness. The light-tracking feature tracks set-point as expected, but a lacking is lack of sensitivity of the feature to the bulb brightness, which is due to the disconnect between ambient light and bulb brightness (as discussed in *Challenges*). However, this is acceptable, as we are more focused on the proper functioning of the logic of the local decisions. Both the motion detection and light tracking set points were also verified to be updated as expected.

When both the ambient light tracking and motion detection were enabled, the two features would sometimes conflict. For example, the motion tracking would turn off the light, and then the light tracking would start to turn it back up, because it sensed that the room got darker. Additionally, there were some cases where we were not sure the best user experience decision. These edge cases were not something we were able to extensively test and prevent in our time frame.

In addition, the NRF52 was verified to respond accurately to global decisions. The schedule was verified by setting schedule min/max values from the website, and the NRF52 reacted correctly. The virtual slider also responded correctly, with its range scaled to match the real limits of the dimmer.

Scalability. As seen in Figure 8, we can see that we easily accomodated 3 peripherals to one central device in our system. We could have easily added more, with each device simply requiring to be scanned and added into the database through the website. In addition, each NRF52 could potentially control more AC dimmer modules, and therefore could control more lights (albeit all with the same dimness), with the limitation being the hardware – how many GPIO pins there are on the Buckler.

4 DISCUSSION

More work needs to be done in improving the communication pathway. While we feel we came up with an elegant and relatively fast solution, we can continue to tweak the update intervals in order

to further improve communication speeds from the website all the way to the Buckler.

In addition, some more testing needs to be done to ensure the system response is accurate under extreme situations, and that the system is robust under those situations. While the demonstration was successful, more complex user input traces need to be evaluated. For instance, we did not test situations where all features and sensors were engaged at exactly the same point in time. We leverage the accuracy and robustness of our system on slight imperfections in the timing and scheduling of our tasks at all levels of the system (globally and locally).

This project, all in all, has great potential not simply in the consumer sphere. In initial scoping of our project, it became very clear that there exist more application of "smart lighting" in the public sphere, such as street lighting. Many cities across the US have investigated upgrading their their street lighting to be more energy efficient and also more "smart" in the way of creating safer communities. For example, one "smart" way of making a safer community is to make the street lights in the immediate vicinity around a person brighter than other lights farther from them. This sort of extension of our project sees more powerful implications than merely household deployment, and a toy-project like ours exposes the issues that may arise in deploying such a system, such as in the areas of timeliness, accuracy, and scalability; which will all have much greater implications at the public scale.