6-14-2018

SmartSys: Energy Conservation of Buildings Using IoT Devices

Jun Chang
Santa Clara University, jchang2@scu.edu

Darence Lim
Santa Clara University, dlim1@scu.edu

Tracy Sun
Santa Clara University, gsun@scu.edu

Patrick Wu
Santa Clara University, pwu1@scu.edu

SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

Date: June 13, 2018

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Jun Chang
Darence Lim
Tracy Sun
Patrick Wu

ENTITLED

SmartSys: Energy Conservation of Buildings Using IoT Devices

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Science

BACHELOR OF SCIENCE IN COMPUTER ENGINEERING

B. Dezfouli

Thesis Advisor

N. Liu

Department Chair

SmartSys: Energy Conservation of Buildings Using IoT Devices

by

Jun Chang
Darence Lim
Tracy Sun
Patrick Wu

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 14, 2018

SmartSys: Energy Conservation of Buildings Using IoT Devices

Jun Chang
Darence Lim
Tracy Sun
Patrick Wu

Department of Computer Engineering
Santa Clara University
June 14, 2018

ABSTRACT

The majority of the energy within the United States is wasted. In addition, buildings such as apartment complexes and high-rises consume large amounts of energy, with commercial buildings wasting, on average, 30% of the energy that they consume. This issue leads to drastic consequences such as an increase in carbon footprint and high energy costs. With this in mind, our team was inspired to create a solution that decreases energy consumption and cost. Our project achieves this goal with a scalable and personalized smart home system that caters to individual users' needs while conserving energy on a large scale. Our solution, SmartSys, cuts energy consumption as well as energy costs through interaction with IoT devices, an architecture that includes a combination of database-centric and event-driven data flows, and various technologies including sensors and machine learning. As a result of single room testing, we estimate that SmartSys will help indivdual users save over $1000 over a time period of 20 years in addition to saving a city with 20 apartment complexes over 150 million KWh after 20 years. For future work, we hope to decrease the fixed cost of SmartSys to make our solution have an even greater impact on energy cost savings while maintaining its energy saving performance. In addition, we hope to engage in multiple room testing as well as scale SmartSys to function throughout a large building.

SmartSys: Energy Conservation of Buildings Using IoT Devices

Jun Chang
Darence Lim
Tracy Sun
Patrick Wu

Department of Computer Engineering
Santa Clara University
June 14, 2018

ACKNOWLEDGEMENTS

This project would not have been possible without the help and guidance of our Senior Design advisor, Professor Behnam Dezfouli. His expertise on computer networking, Internet of Things, and machine learning has been a tremendous help to the development of SmartSys from the beginning to end. We also give thanks to Salma Abdel Magid for advising and mentoring us on deep learning model concepts. Finally, our team would like to thank the Santa Clara University School of Engineering for sponsoring the Senior Design Conference and giving us the opportunity to work on and present our project to industry experts, faculty advisors, and fellow students.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem and Motivation

The U.S. wastes 61%-86% of the energy that runs through the country's economy [19]. An average commercial building wastes 30% of the energy that it consumes [7]. High energy consumption leads to high costs paid by homeowners and renters, which pose potential financial burdens. The average American making over $50,000 spent 5% of their income on utilities in 2001 as opposed to 9% in 2012. Americans in a lower income bracket (making less than $10,000 per year) face even more impact on their income spent on utilities, from spending 36% of their income on utilities in 2001 to 78% in 2012 [19]. Other detrimental consequences include the contribution to climate change, low energy supply, and an increase in carbon footprint [14]. In addition, the use of electricity has dramatically increased in the most recent decades. The United States went from using 1,747 billion KWh in 1975 to 3,833 billion KWh in 2012 [22].

1.2 Solution

In this section, we explore the current solutions on the market today as well as provide insight into our contribution.

1.2.1 Current Solutions

Current solutions include the Nest thermostat, CURB, and Lucid. On one end of the spectrum of smart home technology, Nest [9] is personalized to the individual user with its functionalities meeting personal needs in terms of temperature.

CURB [23] monitors energy within individual homes with functionalities including sending users consumption reports and electricity expenditure reports in addition to controlling smart home devices. CURB hopes, with these functionalities, to allow residents to better understand how much energy they are consuming, and help residents decide where in their homes they should cut back on energy usage.

Lucid [4], on the other end of the spectrum, targets energy management in terms of buildings and campuses.

Features include dashboards with reports and graphs detailing building energy consumption in addition to performance comparisons between buildings.

1.2.2 Contribution

The marketplace lacks a solution that provides personalized energy management that is also scalable and can be maintained throughout a large building. Our solution is a hierarchal system that supports scalability, personalization, and energy conservation. First, SmartSys is able to provide preset brightness levels for different user activities at home. In addition, personalization is achieved through SmartSys incorporating a deep learning model, specifically Convolutional Neural Network (CNN), to classify user behavior. This information and the current light level reading help SmartSys determine what light level the connected light bulbs should produce. SmartSys can handle interoperability with other IoT devices such as Philips Hue light bulbs.

SmartSys achieves scalability through its modular design. SmartSys consists of several gateways such as light sensors, Arduinos, and Raspberry Pis to provide modularization. From personal homes to large commercial buildings, SmartSys' modular characteristics allow for flexibility in regards to the size of the various entities that it is implemented in.

SmartSys achieves energy conservation by adjusting light levels based on current light level readings and classification of user activity. Activities such as sleeping and watching TV usually require low light levels. When users engage in these activities, SmartSys will adjust light levels to match these low light level activities. This reduces the amount of energy consumed by the user.

Chapter 2

Requirements

2.1 Description

In this chapter, the list of functional and nonfunctional requirements is prioritized from top to bottom. Requirements define and qualify what the system must do. In section 2.2, the list of functional requirements are categorized into *required*, *suggested*, and *recommended*. The nonfunctional requirements are considered in terms of scalability, performance, and reliability.

2.2 Functional Requirements

Functional requirements explain what the system must do. They specify a behavior or function. A function is described as a set of inputs, the behavior, and outputs.

- Required:
  - Provide preset brightness levels [13] for different user activities at home.
  - Incorporate deep learning model to track user behavior to create a better user experience.
  - Classify user activities.
  - Automatically adjust the brightness level based on the current brightness reading from the light sensor and user activity.
  - Handle interoperability with other IoT devices.
- Suggested:
  - Expand the system to work with temperature sensors.
  - Create a user interface to allow the user to adjust brightness levels for different activities.

- Recommended
  - Network with other individual systems to create efficient strategies for energy conservation.
  - Utilize an energy conservation algorithm to calculate the amount of energy saved by SmartSys.

2.3 Nonfunctional Requirements

Nonfunctional requirements describe the manner in which the functional requirements must be achieved. Nonfunctional requirements define how a system is supposed to be. Implementing nonfunctional requirements is detailed in the system architecture in Chapter 5.

- Scalability
  - The system must be able to scale up or down based on the size of the residence.
- Performance
  - The system must provide accurate classification of user activities.
  - The system must have the ability to prioritize the most recent user activity.
  - User interface must be user-friendly.
- Reliability
  - The system must protect privacy, confidentiality, and integrity of user data.
  - The system must use secure protocols for sending data.

Chapter 3

Use Case Diagram

3.1 Description

Use case diagrams are usually referred to as behavior diagrams to describe a set of actions (use cases) that the system should or can perform in collaboration with one or more external users of the system (actors). Each use case should have some valuable result for the actors of the system.

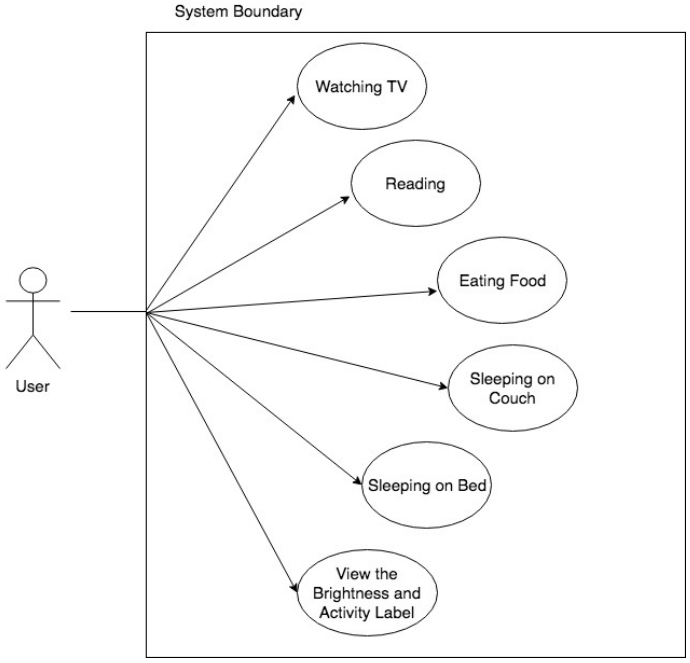

Figure 3.1: Use Case Diagram

3.2 Functionality

3.2.1 Users

Users interact with SmartSys by performing the following four activities:

- Watching TV
- Reading
- Eating Food
- Sleeping

Users can also view the brightness level and its classified activity label on the iOS application.

3.2.2 SmartSys

SmartSys classifies user behavior by using Convolutional Neural Network (CNN) so that once activities are classified, the model helps adjust the brightness level into the desired range based on the preset brightness level of user activities [13] and the brightness reading from the light sensor.

Actor: Residents

Steps:

1. The camera captures an image of the user performing the activity.
2. The system uses Convolutional Neural Network (CNN) to classify user activity.
3. The light sensor reads the current lux reading and sends the reading to SmartSys.
4. The system automatically adjusts the brightness level into the desired range based on preset brightness level for the user activity and the light sensor reading.

3.2.3 View Brightness Level

The iOS app provides residents with the classified activity label and a clear view of the current brightness reading.

Actor: Residents

Steps:

1. Click ”Connect” to build the connection between SmartSys and the iOS app.
2. Click ”Refresh” to get the activity label and readings of the current brightness level.
3. Click ”Disconnect” if the user wants to discontinue the connection between SmartSys and the iOS app.

Chapter 4

Activity Diagram

4.1 Description

Activity diagrams show the flow of actions of a user when interacting with a part of a system. Our activity diagram (Figure 4.1) shows when a user interacts with the SmartSys iOS mobile application. When the user opens the mobile application, the iOS device is connected to the apartment gateway. The user can then view the brightness level of their Philips Hue light bulbs and can press the refresh button to get the latest brightness level as well as the corresponding activity of the user [3].

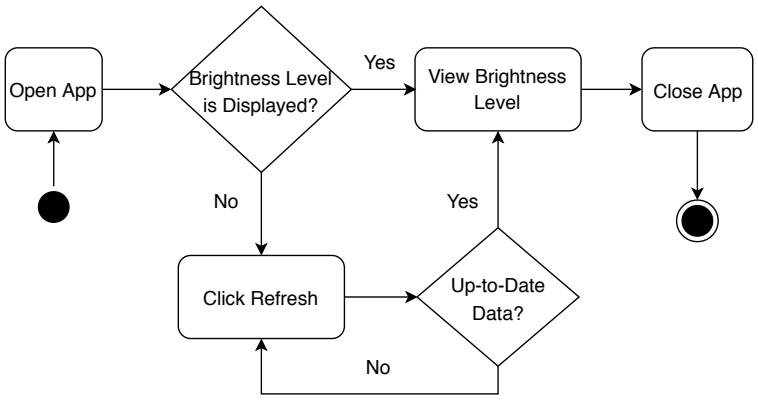

Figure 4.1: Activity Diagram

Chapter 5

Architecture Diagram

5.1 Description

Architecture diagrams model the structure and the behavior of a system. Each element on the diagram represents a modularized part of the architecture, and they are tied together using arrows to show the interaction between each part (Figure 5.1).

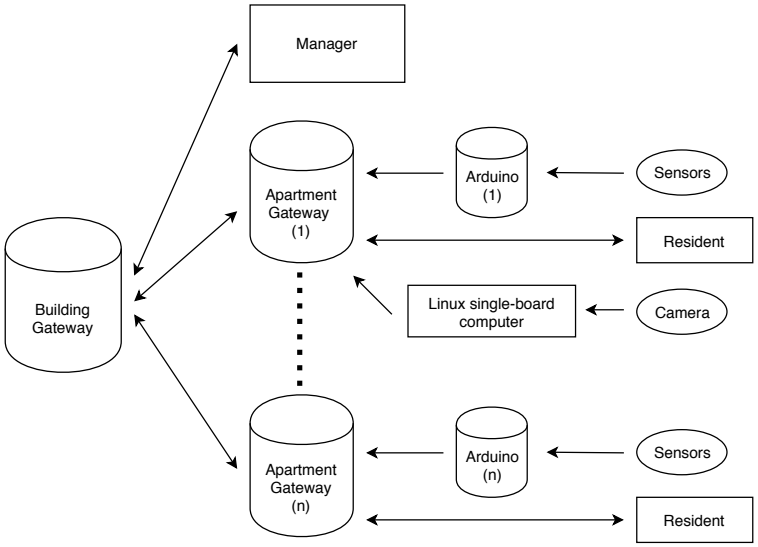

Figure 5.1: Architecture Diagram

5.2 Structure

To make our system expandable and responsive, the architecture includes a combination of database-centric and event-driven data flows.

5.2.1 Database-centric Data Flow

Due to Smartsys' need of data collection from its surroundings and the need to report data to various users, Smartsys incorporates the database-centric data flow to its system as a way to transfer data. Smartsys' database-centric data flow has multiple tiers of microcontrollers, each acting as a data storage device serving different levels of clients such as the apartment residents and apartment managers. The current brightness of a particular room is recorded by the light sensor, which is transmitted to the Arduino through a wired I2C connection. From there, the Arduino transmits the brightness data to the apartment gateway using a Bluetooth connection. The apartment gateway processes that brightness data to adjust the appropriate smart devices to maintain the user-desired brightness level in the room. The apartment gateway also sends that data to the building gateway using a Bluetooth connection while the data is being processed. For future work, we hope that SmartSys will be able to record and calculate other types of data such as energy consumption in each of the gateways. We hope to send these types of additional information from the apartment gateways to the building gateways. This allows the building managers to view the overall energy consumption of the building which can help create energy conservation strategies.

The advantages of this structure make it easy for the apartment resident and the apartment manager to interact with the brightness data that are being collected. In addition, each of the gateways can be easily replaced because each individual gateway acts as an independent module. This makes the process of replacing and upgrading a gateway simpler when said gateway can no longer support the addition of new edge devices.

5.2.2 Event-driven Data Flow

A part of SmartSys' architecture is event-driven due to a collection of sensors that are outputting data such as brightness level to the Arduino. Once the brightness data collected from the sensors are uploaded from the Arduino to the apartment gateway, the resident of the apartment will be able to access the collected data through an iOS application that is connected to the apartment gateway. The apartment gateway also automatically sends out control messages to appropriate IoT devices.

The advantages of this architecture make the system responsive to the smallest changes in the environment. The data collected from the sensors will have an immediate impact on the way the IoT devices are being controlled as the data is uploaded to the apartment gateway for processing. In this case, the apartment gateway acts as both a data storage and a data processing point. The apartment gateway processes the data received from the edge devices and uses that information to control the connected IoT devices. For example, if the room is too dark after sunset, the

light sensor sends the low brightness signal to the Arduino, which will transfer the data to the apartment gateway. After processing the data, the apartment gateway sends a message to the Philips Hue LED light bulbs to increase their brightness until the room is at a desired brightness.

Chapter 6

Technology Used and Design Rationale

6.1 Description

In this section, we will be discussing the technologies that are used throughout SmartSys such as the embedded system, lighting system, deep learning model, and the iOS app.

6.2 Embedded System

The embedded system is divided into two categories: edge devices and gateways. The edge devices are responsible for interfacing with various sensors that are attached to them. In SmartSys, the edge device collects data from the light sensor and reports them to the gateways. The gateways are responsible for receiving data from multiple edge devices and using the aggregated data to control the connected IoT devices such as the Philips Hue LED light bulbs. The gateway receives the light sensor data from the edge devices and processes the image of the user activity from the camera to classify user activity. The apartment gateway uses the aggregated data to change the brightness of the light bulbs in order to maintain an optimal light level in the room.

6.2.1 Edge Devices

For the edge devices, we chose to use micro-controllers that are small in size, so they can be placed anywhere without being intrusive to the user's environment. For this particular piece of hardware, we considered the Arduino Uno and the Arduino Trinket. Although an Arduino Trinket is small in size (31mm x 15.5mm), it does not have enough memory to install both the Bluetooth library and the light sensor library. Additionally, the board only has five pins which are not enough to connect multiple sensors. The Arduino Uno, on the other hand, is optimal for the job even though it is slightly bigger in size (68.6mm x 53.4mm). The Arduino Uno has 20 pins and sufficient flash memory (32 kB) to support all the sensors along with the Bluetooth chip. However, for edge devices that require a camera, we are using a Raspberry Pi 3 instead of the Arduino Uno due to Arduino Uno's lack of power for image processing with CNN.

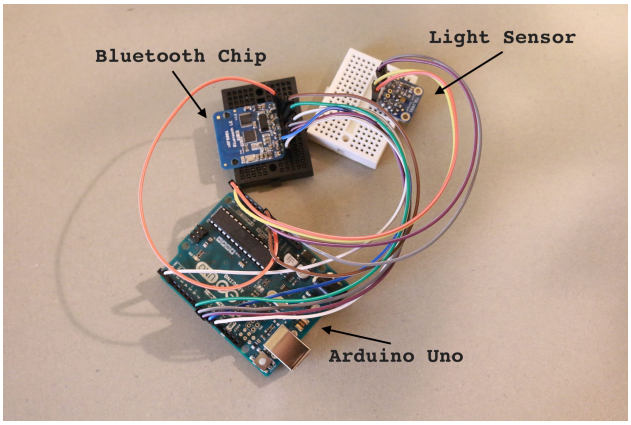

Figure 6.1: Light Sensor

6.2.2 Gateways

In our design, the hardware used as the gateways are flexible as long as they support the number of edge devices that are connected. In our case, we chose to use the Raspberry Pi 3 over the Samsung Artik even though the Samsung Artik has better specifications overall such as a better CPU and GPU. Compared to Samsung Artik, Raspberry Pi 3 is cheaper and has a friendlier user interface. Furthermore, the Raspberry Pi 3 has enough memory (1GB) and computing power (4 x ARM Cortex-A53) to run Convolutional Neural Network (CNN), and it also has built-in Wi-Fi and Bluetooth capabilities which are required for communication with other gateways, edge devices, and the iOS application.

6.3 Lighting System

6.3.1 Light Sensors

Although there are many different light sensors on the market, there are only a few that can fit into SmartSys because we need the light sensor to be small, calibratable and have a high range of detection. We ultimately settled on the TSL2591 Light Sensor as it is a good mix of all the qualities we needed. The light sensor is connected to the Arduino through wired connection and communicates using an I2C protocol with the Arduino using a preset library. The software written on the Arduino is coded in a subset of C/C++. The software does two things: one is to read the brightness level collected from the wired sensors, and the second is to send those data to the apartment gateway using the Bluetooth chip. Figure 6.1 shows the prototype edge device which consists of the Arduino Uno, the Bluetooth chip, and the light sensor all connected together.

6.3.2 Smart Lights

For the smart light system, we chose to use the Philips Hue lighting system [24] because it uses the Zigbee which is a low-power communication protocol. Additionally, Philips Hue light system uses JSON as its messaging format.

JSON is a lightweight data-interchange format designed to be easy for humans to read and write. The software used to control the light system is written in python and resides in the apartment gateway. The python code does three things: one is to read in the brightness level received from the edge devices, the second is to send out JSON messages to the light system using the Zigbee protocol to adjust the brightness of the light bulbs, and finally the code can also request the brightness of the light bulbs from the light system.

6.4 Deep Learning

SmartSys is able to adjust light levels based on current light levels and user activity. For example, if someone is reading, SmartSys determines the current light levels, classifies the user's activity, and takes the preset brightness range for reading into consideration. With this, SmartSys adjusts the light level until the light sensor records a light level within the preset brightness range for reading. In order to differentiate and classify user activity, SmartSys uses a deep learning model, specifically Convolutional Neural Network (CNN), to classify various user activities such as reading, eating, sleeping, and watching TV. Our team also utilized the Bing Search API to gather training data in order to train CNN.

6.4.1 Bing Search API

The Bing Search API allowed us to scrape the Web for images pertaining to various human activities such as eating, sleeping, reading, and watching TV. This was done by sending query requests for images of these activities. The results came back in JSON format. In batches, images were saved and downloaded into a training dataset folder. After gathering the images for each activity, we used these retrieved images to train our deep learning model [15]. We preferred this method as opposed to manual data collection for efficiency purposes.

Pseudocode

This section consists of the pseudocode for the Bing Search API. Please refer to the source code in Appendix B.2.1 for complete documentation.

- Search Web given the query.

```
term = args["query"]
headers = {"Ocp-Apim-Subscription-Key" :  API_KEY}
params = {"q":  term, ‘‘offset":  0, ‘‘count":  GROUP_SIZE}
print("[INFO] searching Bing API for '{}'".format(term))
search = requests.get(URL, headers=headers, params=params)
search.raise_for_status()
```

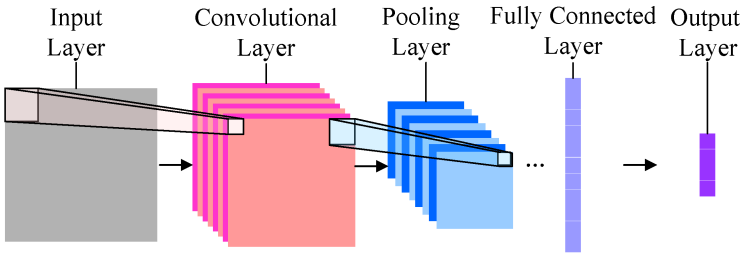

Figure 6.2: Convolutional Neural Network

- Results are saved in JSON format.

```
results = search.json()
estNumResults = min(results["totalEstimatedMatches"], MAX_RESULTS)
print("[INFO] {} total results for '{}'".format(estNumResults,term))
```

- Images (in batches) are iteratively saved and downloaded into a training dataset folder.

6.4.2 Convolutional Neural Network (CNN)

Background

The modern CNN is inspired by a well-known research paper written by Yann LeCun and Leon Bottou in 1998 [8]. In this research paper, LeCun and Bottou introduced a neural architecture, LeNet5, that is used for recognizing hand-written digits and words and established the classification accuracy of 99.2% on the MNIST dataset. The LeNet5 architecture was fundamental in deep learning. An important takeaway from the research paper is that image features are distributed across the entire image, so convolutions with learnable parameters are an effective way to extract similar features at multiple locations with few parameters [17]. At that time, there was no GPU to help training, so being able to use fewer parameters and less computation power was a key advantage.

In addition, CNNs are inspired by biological models. The research done by D.H.Hubel and T.N Wiesel [5] proposed an explanation for the way that mammals visually perceive the world around them using a layered architecture of neurons in the brain. This inspired engineers to develop similar pattern recognition mechanisms in computer vision.

CNN is a deep learning model that is made up of neurons that have learnable weights and biases. Each neuron receives some inputs, learns directly from the image data, processes and transforms the input to produce an output label. CNN is one of the most popular techniques used in image recognition and computer vision systems today.

CNN Architecture

The architecture of a typical CNN (Figure 6.2) is composed of multiple layers where each layer performs a specific function of transforming its input into a useful representation. There are four major hidden layers that are commonly used which are listed as the following:

1. The Convolutional Layer

   The convolutional layer forms the basis of the CNN and performs the core operations for training the model. It seeks out certain patterns in the image that forms a filter to allow neurons to look for the same patterns in different regions of the image.

   The convolutional layer applies a convolution operation to the input, passing the result to the next layer. For mathematical purposes, a convolution is the integral measure of how much two functions overlap as one passes over the other. The convolutional layer perceives images as volumes, i.e., 3D object rather than 2D. Instead of focusing on one pixel at a time, the convolutional layer takes in square patches of pixels and passes them through a filter. That filter is also a square matrix smaller than the image itself, and the job of the filter is to find patterns in the pixels [17].

   One of the main problems with image analysis is due to its high dimensional characteristic. Therefore, processing images has a high cost in time and computing power. The convolutional layer is designed to reduce the dimensionality of images.

2. The ReLU Layer (Rectified Linear Unit Layer)

   ReLU is the most commonly deployed activation function for the outputs of the CNN neurons. ReLU allows the network to be properly trained without slowdowns from the early layers. Mathematically, it is described as: $f(x) = max(0, x)$, where x is the input to a neuron. This is also known as a ramp function and is analogous to half-wave rectification in electrical engineering. It has been demonstrated for the first time in 2011 to enable better training of deep network compared to the widely used activation functions such as logistic sigmoid and the hyperbolic tangent. The rectifier is the most popular activation function for deep neural network.

3. The Pooling Layer

   The pooling layer reduces the spatial dimensions of the input volume for the next convolutional layer. The pooling layer ensures that the network focuses on the most relevant patterns discovered by the convolutional and ReLU layer.

   The operation performed by this layer is called down-sampling. Size reduction leads to a loss of information. However, the loss is beneficial for the network for two reasons. First, the decrease in size leads to less computational overhead for the upcoming layers. And secondly, it prevents over-fitting the training model.

   The transformation is either performed by max pooling, which takes the maximum value from the values observable in the window or by taking the average of the values. Max pooling has better performance over taking the average.

The operation is performed for each depth slice and then for each color channel. The values will be down-sampled to their representative maximum value if we perform the max pooling operation.

4. The Fully Connected Layer

   The fully connected layer is configured to connect with the output of the previous layer. They are typically attached to the very end of the network to connect to the output layer and construct the desired outputs.

Libraries Used

In order to train our CNN model for image recognition, the following libraries are used and listed as follows:

- Keras [2]: Neural Network API works with TensorFlow and provides an easy and fast experimentation.
- TensorFlow [20]: The library provides functions for CNN layers and runs faster than pure Python code.
- OpenCV2 [10]: An open source computer vision library that performs image processing.

Advantages

CNN has several unique features that outperform non-neural networks because of the following reasons:

- CNN has weight-sharing. It is more efficient in terms of memory and complexity.
- CNN has good feature extractors.
- CNN is efficient in pre-training because the new classifier can be added at the end of the labels.

CNN Used in SmartSys

After the Raspberry Pi camera captures the image of users' activity, the data is stored in the apartment gateway. Then the image data is processed by the CNN model to produce an activity label. Meanwhile, the light sensor is taking the current light reading of the environment. Then SmartSys will consider the final brightness level based on the user activity, preset brightness range, and the current light reading. Finally, the output of CNN (activity label and the suggested brightness level) is sent to the iOS application, so the users can view the modified brightness level.

Pseudocode

In this section, we demonstrate the pseudocode for CNN classification. Please refer to the source code in Appendix B.4.1 for complete documentation.

- Load the user activity image that is taken by the camera to be pre-processed.

```
image = cv2.imread(args["image"])
```

- Load the trained convolutional neural network and the label.

```
model = load_model(args["model"])
lb = pickle.loads(open(args["labelbin"], "rb").read())
```

- Classify the input image.

```
proba = model.predict(image)[0]
idx = np.argmax(proba)
label = lb.classes_[idx]
```

- Set brightness level to the recommended light level by National Optical Astronomy Observatory.

```
watch TV range:  125 - 175
eating range:  200 - 300
reading range:  450 - 550
sleeping turns off the lights
```

- Build the label and draw the label on the image.

```
label = ":  :.2f%".format(label, proba[idx] * 100)
```

- Write the activity label and brightness into text file to be send to iOS app.

```
f.write(label)
f.write(str(brightness))
```

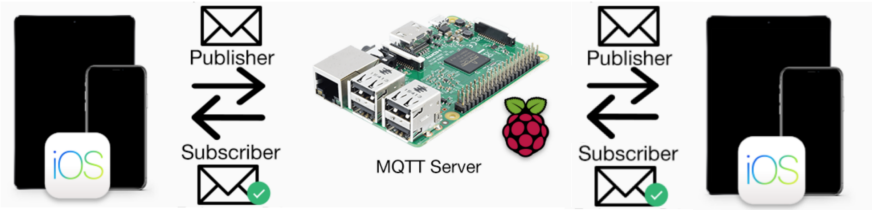

Figure 6.3: MQTT Protocol

6.5 iOS Mobile App

The purpose of this mobile app is to give homeowners a sense of how much energy is being consumed with SmartSys. The app shows the measurement of the Philips Hue brightness which range from 0 to 254 with its own unit of measurement, as well as the percentage of the brightness compared to the max output. For future work, we hope the iOS app will allow the user to be able to adjust brightness level to further personalization as well as display energy conservation statistics.

6.5.1 Xcode

We used Xcode as the integrated development environment (IDE) to develop the mobile application. Xcode contains many software development tools by Apple for iOS. We used Swift as the primary language for iOS app development in Xcode.

6.5.2 Message Queuing Telemetry Transport (MQTT)

When deciding which messaging protocol to use for communication between our iOS mobile app and SmartSys, our group chose between MQTT and HTTP [18]. Even though HTTP is the most popular and widely used protocol, MQTT has rapidly gained traction over recent years. In terms of the design and messaging of the two protocols, MQTT is data-centric whereas HTTP is document-centric. Since HTTP is a request-response protocol for client-server computing, it is not necessarily optimized for mobile devices. MQTT, on the other hand, is a light-weight messaging protocol that is very popular among IoT devices. It follows a publish/subscribe model that makes it very suitable for conserving battery power on devices such as the iPhone. Since we are focusing on the transmission of data and not documents in our app, we opted for MQTT. In addition, the publish/subscribe model works independently of different clients, which enhances the reliability of the system as a whole. If one client is unavailable, it does not affect the rest of the system.

Here is an explanation of how devices connect and communicate with each other in MQTT using the publish/subscribe model. A client that sends messages is known as a publisher, and a client that receives messages is known as a subscriber. Each message sent by a publisher contains a topic, and a subscriber that is subscribed to that topic will receive that message. Let us assume that an iOS device is the publisher and a Raspberry Pi is a subscriber. A publisher sends a message with a topic of ”get energy level”, for example. This message is received by the the Raspberry Pi MQTT server, and the MQTT server will search for the subscriber that is subscribed to the same topic ”get energy level.” If the Raspberry Pi is subscribed to the topic ”get energy level”, the MQTT server will send the message to the Raspberry Pi itself and the message is successfully delivered.

In order for the iOS app to connect to the Raspberry Pi 3, we set up the Raspberry Pi 3 to have a hostname, ”SmartSys.” This means that the iOS app will use hostname to retrieve the IP address of the gateway. For SmartSys, the iOS mobile app serves as the MQTT client that requests for data regarding the brightness level and the activity of the user [3]. The Raspberry Pi 3, which serves as the MQTT server, receives the request and sends a response message back to the app with the relevant data. Once the iOS app receives the message, it then displays the data on the user interface. When information is sent to the Raspberry Pi 3 gateway, MQTT is capable of distributing information from the Raspberry Pi 3 to multiple iOS devices. Therefore, MQTT is highly optimal in what our team hopes to achieve in a large-scale distribution system.

Libraries Used

Libraries used for iOS app development are listed as followed:

- CocoaMQTT [3]: The CocoaMQTT client library is an open source MQTT client library written in Swift for iOS applications. This library allows the iOS device to act as a client to communicate with an MQTT server.
- Eclipse Paho [3]: The Eclipse Paho library provides open-source client implementations of MQTT messaging protocols aimed at applications for IoT devices. This library allows the Rasbperry Pi to serve as the MQTT server that recieves and sends MQTT messages to various MQTT clients such as the iOS device.

6.5.3 Algorithm

- When the app opens, it utilizes the CocoaMQTT library's mqttClient function to connect to the Raspberry Pi. The MQTT Server is set up on the Raspberry Pi using the Eclipse Paho Library.
- After the iOS app is connected to the Raspberry Pi MQTT server, the app publishes an MQTT message with the topic of "Get Brightness" to the MQTT server, requesting data regarding the Philips Hue light bulb brightness.
- This message with the topic "Get Brightness" is received by the Raspberry Pi MQTT server. Since the Raspberry Pi is subscribed to the topic "Get Brightness", the message will be delivered from the iOS app to the Raspberry Pi. The Raspberry Pi now knows to communicate back to the iOS app by publishing another MQTT message with the topics "Send Brightness" and "Send Activity," and the corresponding data gets sent to the app.
- The iOS app calls the subscribe function to the Raspberry Pi with the topics "Send Brightness" and "Send Activity." The iOS app will then successfully receive the message from the Raspberry Pi, parse the message, and then display the relevant brightness and user activity data on the user interface.

6.5.4 Pseudocode

iOS MQTT Client

- `mqttClient = CocoaMQTT(client, host, port)`

  There are three parameters when we call the CocoaMQTT Library function to create an MQTT client. The first is Client ID, which we specify as the iOS device. Next, is the host that the host connects to, which is the hostname of the Raspberry Pi, "SmartSys". The third parameter is the TCP/IP port for MQTT, which is 1883.

- ```
  function connect(sender:  button) {
  mqttClient.connect()
  }
  ```
  ```

When the connect button on the app is pressed, the mqttClient connects to the host, which is the Raspberry Pi gateway.

- `function disconnect(sender: button) {`

  `mqttClient.disconnect()`

  `}`

  When the disconnect button on the app is pressed, the mqtt disconnects from the Raspberry Pi gateway.

- `function refresh(sender: button) {`

  `mqttClient.publish(topic, string)`

  `mqttClient.subscribe(string)`

  `mqttClient.didReceiveMessage = {mqtt, message}`

  `}`

  - mqttClient publishes a request with the topic ”Connect” with the string ”Get Brightness.”
  - mqttClient subscribes to the topics ”Send Brightness” and ”Send Activity”, which are sent by the Raspberry Pi MQTT server.
  - When the mqttClient recieves the messages from topics ”Send Brightness” and ”Send Activity”, the brightness and activity string gets parsed and displayed on the app user interface as labels to be viewed.

Raspberry Pi MQTT Server

- `connectionStatus(client, userdata, flags, rc):`

  `mqttClient.subscribe("Connect")`

  The Raspberry Pi MQTT server subscribes to the topic ”Connect”. If the iOS app publishes an MQTT message with the same topic, then an MQTT connection between the iOS device and the Raspberry Pi will be established.

- `messageDecoder(client, userdata, msg):`

  `if message == "Get Brightness":`

  `mqttClient.publish("Send Brightness", brightness)`

  `if message == "Get Activity":`

  `mqttClient.publish("Send Activity", activity)`

  - When the Raspberry Pi receives an MQTT message from the iOS app with the topic ”Connect”, it will check to see what the message is.

- If the message is ”Get Brightness”, the Raspberry Pi publishes a message with the topic ”Send Brightness” and the corresponding brightness of the Philips Hue light bulb to the iOS MQTT client.
- If the message is ”Get Activity”, the Raspberry Pi publishes a message with the topic ”Send Activity” and the corresponding activity of the user to the iOS MQTT client.

- `mqttClient.loop_forever()`

  The loop forever function monitors the MQTT client activity until the connection is disconnected by the user.

Chapter 7

Testing

7.1 Description

The following are the types of testing that we have completed for the project: unit, functional, component, system, regression, and client testing.

7.2 Unit Testing

Unit testing is a process to verify that a relatively small piece of code is doing what it is intended to do. An example of our engagement in unit testing includes testing the individual libraries for our deep learning model and updating any libraries if necessary.

7.3 Functional Testing

Functional testing is to check multiple units as a functional unit. An example of our engagement in functional testing includes testing the deep learning model with all libraries and functions working to accurately classify images.

7.4 Component

Component testing tests multiple functions. We integrated the light sensor with the deep learning model. From this step, we tested whether the light bulb would change light levels depending on the current light level (provided by the sensor) and the specific user activity (classified by the deep learning model).

7.5 System

System testing tests multiple components together. We integrated the light sensor, deep learning model, and iOS application to produce a fully-functional system. We tested the system as a whole to confirm that each component worked coherently with each other in reading the current light level, classifying human activity, communicating with the light bulbs, and displaying the brightness level to the user.

7.6 Regression Testing

Regression testing determines whether aspects of the system still work when changes are made to a particular component of the system. An example of our engagement in regression testing includes testing the deep learning model after system integration to make sure it was still properly classifying user activity.

7.7 Client Testing

Our team engaged in client testing with one of our members acting as the client. Our team member lived with SmartSys for three days and without SmartSys for another three days. SmartSys was placed in our team member's dorm room where he performed activities such as eating, sleeping, reading and watching TV. He used a lamp with a single light bulb and a power meter to measure energy consumption. The number of hours spent with the system equaled the number of hours spent without the system in terms of the light bulb being on. Pleases see Chapter 8 (Results) for the energy conservation and cost savings results associated with the completed client test.

Chapter 8

Results

8.1 Description

Client testing produced energy conservation and financial savings. With our test results, we estimated the savings at an individual, building, and city-wide level. To learn about our client testing methodology, please refer to Chapter 7.

8.2 Energy Savings

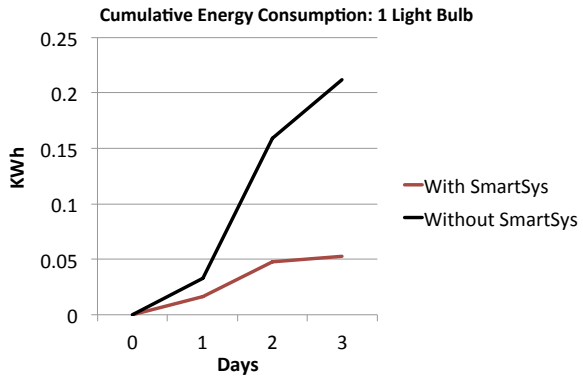

Figure 8.1: Cumulative Energy Consumption Using One Light Bulb

Client testing proved that SmartSys does indeed conserve energy for the user. Over the course of three days with SmartSys, our team member used 0.053 KWh with one light bulb. Over the course of three days without SmartSys, our team member used 0.212 KWh with one light bulb. As a result, with SmartSys, our team member used around four times less energy than he did without SmartSys. Please refer to Figure 8.1.

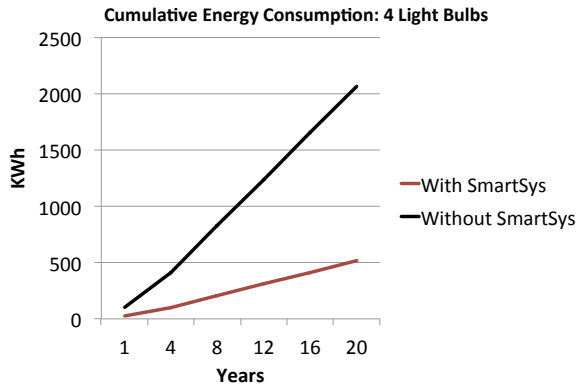

Figure 8.2: Cumulative Energy Consumption Using Four Light Bulbs

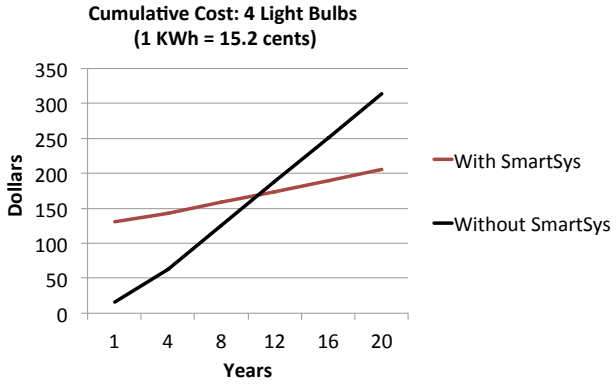

Figure 8.3: Cumulative Cost: 4 Light Bulbs

If these results are extended to 4 light bulbs with one system over 20 years, we find that with SmartSys, a user will use 516 KWh as opposed to 2064 KWh without SmartSys. Please refer to Figure 8.2.

8.3 Financial Savings

The fixed cost of our system totals to $126.88 including the Arduino Uno ($22.00), light sensor ($6.95), Bluetooth ($19.95), Raspberry Pi with power source ($49.99), and Raspberry Pi camera ($27.99).

1 KWh = 15.2 cents in California [21]. With the fixed cost and energy cost in mind, our system breaks even at around year 10-11. Please refer to Figure 8.3.

After 20 years, a resident residing in an individual living unit with ten rooms will save over $1000 while a city with 20 apartment complexes will save over 150 million KWh as seen in Table 8.1.

Entity	Energy Savings (after 20 years)	Financial Savings (after 20 years)
Individual Living Unit with 10 Rooms	15,476 KWh	$1,084
Apartment Complex with 500 Living Units	7,738,212 KWh	$541,808
City with 20 Apartment Complexes	154,764,240 KWh	$10,836,164

Table 8.1: Long Term Energy Conservation and Financial Savings

Chapter 9

Future Work

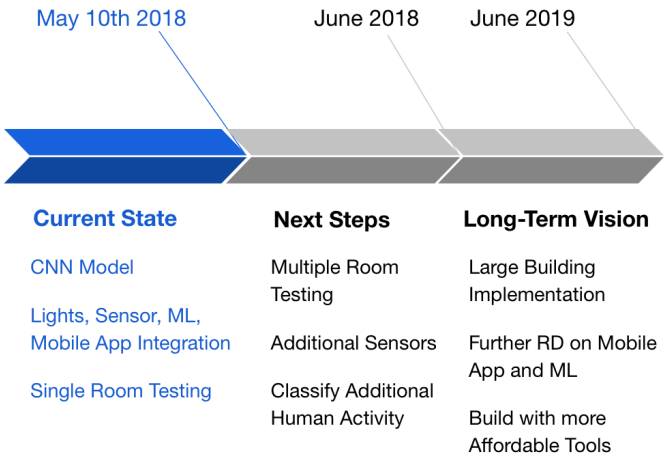

Figure 9.1: Remaining Tasks

9.1 Current State

The Convolutional Neural Network (CNN) model allows for image analysis and classification of user activity. When the Raspberry Pi camera captures the image of users' activity, the CNN model processes the image data to produce an activity label. The output of CNN (activity label and the suggested brightness level) then gets stored in the Raspberry Pi gateway. The Philips Hue light bulbs serve as the primary home appliance that we are adjusting through our smart home system. We have incorporated the TSL2591 light sensor to measure the brightness of the room. By implementing machine learning, we have been able to train our smart home system to accurately detect user activity, so that SmartSys knows when to change the brightness of the Phlips Hue light bulbs to maintain an optimal light level in the room. We have also developed an iOS mobile app integration with SmartSys to allow users to see the brightness level of their Philips Hue. So far, we have been able to successfully test SmartSys in a single room.

9.2 Next Steps and Long-Term Vision

If time permits, our team will integrate SmartSys into multiple rooms for testing. After testing SmartSys in multiple rooms, we hope to scale our system to accommodate large buildings such as apartments or schools. We hope to add additional sensors for temperature, and we hope to have SmartSys recognize additional human activities such as cooking and cleaning. We also hope to work more on the machine learning aspect of our home system to recognize human activity with increased accuracy and speed. Further developments will be done on the mobile app to fit cross-platform devices, such as Android mobile devices. Finally, we want to improve SmartSys by integrating more affordable tools to lower costs.

Chapter 10

Lessons Learned

We learned a lot of lessons during our year-long process of design and implementation. We learned to work with resource constrained devices such as the Raspberry Pi 3 to run appropriate deep learning models on them. We also researched reliable and energy efficient communication protocols to preserve energy. Finally, we learned different architecture designs to find the best fit for our IoT system.

Chapter 11

Obstacles Encountered

There were many obstacles that we encountered during the design and implementation process. One of the major ones includes the selection of a neural network. We had to carefully evaluate the advantages and disadvantages of each network in order to find the best fit. For SmartSys, we wanted our classification of the clients' actions to be as accurate as possible. However, due to our inexperience with machine learning models, the first model we implemented generated a low accuracy rating in terms of classifying user activity. Later, we mitigated this problem by applying CNN even though it does take longer than other networks at approximately 20 seconds on the Raspberry Pi 3.

Another challenge was to connect all of the modules together since all of the team members worked on different aspects of the project. It was difficult finding the appropriate communication protocols for the modules while having all the group members present. Nevertheless, we were able to effectively communicate the requirements we needed for each of our parts to work. In the end, we successfully combined all of our work and created SmartSys.

Lastly, since SmartSys takes in user data in the form of images, security is one of SmartSys' top priorities. After considering the software approach, which is using security protocols, and the hardware approach, which is storing all user data locally, our team decided to use the hardware approach. Since SmartSys works with edge devices, storing all user data on Raspberry Pi and regularly replacing the user image (SmartSys only stores one image at a time) with the most recent one achieve the nonfunctional requirements in terms of security and performance. In addition, storing user data locally instead of storing data in the cloud reduces the risk of data being hacked. When users integrate SmartSys into their environment through Wi-Fi, most of the Wi-Fi services today provide security protocols which add additional protection to SmartSys.

Chapter 12

Ethics

12.1 Description

As our team worked on SmartSys, we encountered prominent ethical issues. From an internal standpoint, the concern of security raised an ethical flag that affected our decision-making process in completing SmartSys. From an external, ethical view, our team made technical decisions to create a system that helps lower energy consumption, which affects both individuals and communities at large. This section explores the ethical aspects of SmartSys in regards to the environment and security.

12.2 Environmental Impact

One defining ethical aspect of our project is its focus on energy conservation. The U.S. wastes 61%-86% of the energy that runs through the countrys economy [19]. Detrimental consequences include the contribution to climate change, high energy costs, and an increase in carbon footprint [14]. Through two of the Markkula Center for Applied Ethics approaches, Utilitarian and Common Good [1], one can observe the ethical impact that these consequences evoke.

Through the Utilitarian approach, one can comprehend the enormous cost in regards to the overconsumption of energy. Increasing one's energy consumption increases one's energy costs. This can lead to critical burdens for individuals and families at financial risk. SmartSys tackles this potential burden through its machine learning model and integration with smart lights. SmartSys' machine learning model is able to detect human activity such as sitting and sleeping. If our system does detect sleeping, then the connected smart lights will adjust light levels accordingly, most likely dimming or shutting down completely. If, on the other hand, our system had no detection feature or smart device integration, then any lights turned on would still be active even if the resident is sleeping, resulting in the exertion of energy that has no use or value. Thus, the decision to integrate machine learning and smart home devices allows SmartSys to reduce energy consumption, which lowers the probability of increased energy costs for those utilizing the system. By lowering energy costs and energy consumption, we lower the costs associated with the Utilitarian approach of ethics.

Another approach that one can use to analyze our systems ethical impact is through the Common Good approach in regards to the change in carbon footprint and the implications it has on our communities. The increase in carbon emissions results in consequences such as reducing food and water supply, elevating sea levels, and increasing the probability of extreme weather conditions [16]. All of these outcomes affect individuals, families, and even broader communities. The reduction of food and water supply will affect food and nourishment attainment, especially for low-income families. The rise of sea levels will force whole communities to migrate to higher elevation or other regions, causing distraught and disruption. The increase in extreme weather scenarios will lead to the destruction of homes, communities, and the economy. All of these aspects pose detrimental results to society as one looks through the Common Good approach of ethics.

One focus of our project is to lower the carbon footprint of those utilizing SmartSys. As mentioned earlier, our decision and methodology of lowering energy consumption, thus lowering carbon emissions, is through the systems machine learning model and the integration of smart home devices such as smart lights. However, SmartSys' scalable feature further extends its positive environmental impact. Our team decided to modularize the system's various components to allow SmartSys to be easily extended, so that the system can cover not only a single living unit, but scale up to reduce the collective carbon footprint for large buildings such as apartment complexes. By utilizing a machine learning model, smart device integration, and scalability, the system strives to mitigate carbon emissions and thus, reduces the probability of negative consequences that could harm the Common Good.

12.3 Security

One of SmartSys' functional requirements is to automatically adjust energy settings based on event triggers, and this is completed by applying a machine learning methodology, convolutional neural network (CNN), to detect and classify human activities. However, the concern for our solution is data security because SmartSys includes cameras that will actively record the residents' daily activities. If security is not properly taken care of, SmartSys will be exposed to various vulnerabilities and will be open to malicious attacks, resulting in failure to protect user data. According to the first IEEE Code of Ethics, we are responsible in making decisions consistent with the safety, health, and welfare of the public, and disclose promptly factors that might endanger the public or the environment [6]. Therefore, if our system faces the failure to protect user privacy, we will have full responsibilities for our development decisions and face serious consequences.

The first decision made to ensure security for users is the use of cloud versus local storage. After cameras identify users' presence, the device will take a picture of the user and process the image for activity classification. We can either send the image data to the cloud or store them locally. Cloud storage adds benefits such as adequate memory space, accessibility, cost saving and backup recovery. On the other hand, local storage requires SmartSys to regularly

free up its memory in order to continuously save user data due to its limited space. Though cloud storage brings many advantages over local storage, one ethical issue with using cloud storage is data security. According to the Rights approach, the ethical action is the one that best protects and respects the moral rights of those affected. Our decision making is based on the protection of free choice and one's privacy. Though cloud storage encounters our dilemma with limited memory, in order to respect the safety and privacy of our users, our decision is to choose local storage. It may seem that local storage does not provide the best performance, but it is the optimal choice because there are too many risks such as hacking, the Middle Man attack, and corrupted data in terms of storing user data remotely. If there was a data breach, there will be too much damage to our users, and we will lose the credibility to those who initially trusted SmartSys. Though the decision to store image data on local devices affects SmartSys' overall performance, after careful evaluation of the potential risks, we chose the latter as it provides more protection for user privacy.

Chapter 13

Conclusion

As technological developments continue to improve people's daily life, our group considered ways to make use of these advancements to create a more sustainable way of living. Our project aims to tackle the problem of wasted energy through home appliances by implementing a scalable smart home system. In our database-centric architecture, data is gathered by sensors attached to Arduinos, and is then uploaded into the apartment gateway. Then the apartment gateway sends that data to the building gateway. As a result of our tests, a user living with SmartSys only uses 25% of the amount of energy as opposed to living without SmartSys.

For future work, our team will add additional types of sensors such as temperature to SmartSys, and we will work to have SmartSys recognize additional human activities such as cooking and cleaning. Given our system gateway setup, machine learning implementation, IoT device, and mobile app integration, our team has set a good foundation for further improvements and scalability to meet our goal of building a high performing, reliable, and secure smart home system. By focusing our attention on an architecture that is designed for large building complexes, we hope that SmartSys will mitigate the issue of energy waste on a larger scale.

Bibliography

[1] "Ethical Decision Making." *Markkula Center for Applied Ethics*. accessed April 10, 2018. https://www.scu.edu/ethics/ethics-resources/ethical-decision-making/.

[2] "Keras: The Python Deep Learning Library." *Keras*. accessed May 09, 2018. https://keras.io/.

[3] "Raspberry Pi Guide - MQTT Protocol - Raspberry Pi Forums." *Raspberry Pi*. 22 Jan. 2016. www.raspberrypi.org/forums/viewtopic.php?t=196010.

[4] "Harness the Power of Data to Improve Building Operations." *Lucid*. Accessed January 11, 2018. https://lucidconnects.com/.

[5] Hubel, David H., and Torsten N. Wiesel. "Receptive fields and functional architecture of monkey striate cortex." The Journal of physiology 195.1 (1968): 215-243.

[6] "IEEE Code of Ethics." *IEEE - Advancing Technology for Humanity*. accessed April 10, 2018. https://www.ieee.org/about/corporate/governance/p7-8.html

[7] "Improve Energy Use in Commercial Buildings." *ENERGY STAR Buildings and Plants — ENERGY STAR*. Accessed May 29, 2017. https://www.energystar.gov/buildings/about-us/how-can-we-help-you/improve-building-and-plant-performance/improve-energy-use-commercial.

[8] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

[9] "Learning Thermostat, 3rd Generation." *Home Depot*. Accessed January 11, 2018. https://www.homedepot.com/p/Nest-Learning-Thermostat-3rd-generation-T3007ES/206391087.

[10] OpenCV. Accessed May 29, 2018. https://opencv.org/.

[11] Peng, Min, Chongyang Wang, Tong Chen, and Guangyuan Liu. "NIRFaceNet: A Convolutional Neural Network for Near-Infrared Face Identification." *MDPI*. October 27, 2016. Accessed May 10, 2018. http://www.mdpi.com/2078-2489/7/4/61.

[12] "Privacy-preserving Image Processing with Binocular Thermal Cameras." *Communications of the ACM*. Accessed May 10, 2018. https://dl.acm.org/citation.cfm?id=3161198.

[13] Recommended Light Levels. PDF. National Optical Astronomy Observatory. https://www.noao.edu/education/QLTkit/ACTIVITY_Documents/Safety/LightLevels_outdoorindoor.pdf

[14] Rogers, Chris. "What Are the Effects of Overusing Energy?" *SFGate*. accessed January 11, 2018. http://homeguides.sfgate.com/effects-overusing-energy-78753.html.

[15] Rosebrock, Adrian. "How to (Quickly) Build a Deep Learning Image Dataset." *pyimagesearch*. accessed May 18, 2018. https://www.pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-image-dataset/.

[16] Caroli, Sarah. "Consequences of Carbon Emissions for Humans." *Seattle Post-Intelligencer*. accessed April 09, 2018. http://education.seattlepi.com/consequences-carbon-emissions-humans-4138.html

[17] Saxena, Abhineet. "Convolutional Neural Networks (CNNs): An Illustrated Explanation." XRDS. June 29, 2016. Accessed May 29, 2018. https://xrds.acm.org/blog/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/.

[18] Serozhenko, Marina. "MQTT vs. HTTP: Which One Is the Best for IoT? MQTT Buddy Medium." *Medium*. Augmenting Humanity. 20 Mar. 2017. medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-iot-c868169b3105.

[19] "Spooky Statistics About Energy And Water Waste." *Energy Resource Center*, last modified October 29, 2013, https://www.erc-co.org/spooky-statistics-about-energy-and-water-waste/.

[20] TensorFlow for R. Accessed May 29, 2018. urlhttps://tensorflow.rstudio.com/.

[21] "The Price Of Electricity In Your State." *npr*. accessed May 9, 2018. https://www.npr.org/sections/money/2011/10/27/141766341/the-price-of-electricity-in-your-state.

[22] "Total Electricity End Use in the U.S from 1975 to 2017 (in Billion Kilowatt Hours). *Statista*. Accessed May 28, 2018.
https://www.statista.com/statistics/201794/us-electricity-consumption-since-1975/.

[23] "What is CURB?" *CURB*. Accessed May 28, 2018.
http://energycurb.com/.

[24] "Wireless and Smart Lighting by Philips — Meet Hue." *Philips*. Accessed June 05, 2018.
https://www2.meethue.com/en-us.

Appendix A

User Manual

A.1 Update and Upgrade Raspberry Pi

The first step is to update and upgrade any existing packages:

1. `sudo apt-get update`
2. `sudo apt-get upgrade`

A.2 Set up for CNN Libraries

In this section, we are installing several libraries such as OpenCV2, Keras, and TensorFlow.

A.2.1 Install OpenCV2

1. `sudo apt-get install python-opencv`

A.2.2 Install Necessary Libraries

1. `install scipy`
2. `sudo apt-get install libblas-dev`
3. `sudo apt-get install liblapack-dev`
4. `sudo apt-get install python3-dev` (Possibly already installed)
5. `sudo apt-get install libatlas-base-dev` (Optional)
6. `sudo apt-get install gfortran`
7. `sudo apt-get install python3-setuptools` (Possibly already installed)
8. `sudo apt-get install python3-scipy`
9. `sudo apt-get install python-sklearn`
10. `sudo pip install pillow`
11. `sudo apt-get install python3-h5py`
12. `sudo pip install protobuf`
13. `sudo pip install imutils`

A.2.3 Install Keras

1. `sudo pip install keras`

A.2.4 Install TensorFlow1.5

1. Go to: `https://github.com/lhelontra/tensorflow-on-arm/releases`
2. Download tensorflow-1.5.0-cp27-none-linux_armv7l.whl
3. `sudo pip2 install tensorflow-1.5.0-cp27-none-linux_armv7l.whl`

A.3 Install Phue Library

In this section, we are downloading and installing the Phue library source code from GitHub.

1. `git clone https://github.com/studioimaginaire/phue.git`
2. `sudo easy install phue` OR `pip install phue`

A.4 Install Bluetooth

A.4.1 Set up UART on Raspberry Pi 3 terminal

1. `sudo hciconfig hci0 up`

A.5 Install MQTT Protocol

In this section, we are installing and setting up the MQTT protocol for iOS client and Raspberry Pi server.

A.5.1 Installing CocoaMQTT Library on iOS Application

1. `sudo gem install cocoapods`
2. `pod setup --verbose`
3. `cd Desktop/nameofproject`
4. `pod init`
5. `open -a Xcode Podfile`
6. Edit the pod file to include the CocoaMQTT external library:
```
platform :ios, '10.0'
target 'SmartSys iOS App' do
use_frameworks!
pod 'CocoaMQTT'
end
```
7. `pod install`

A.5.2 Set up the Mosquitto MQTT Serve on Raspberry Pi

1. `wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key`
2. `sudo apt-key add mosquitto-repo.gpg.key`
3. `cd /etc/apt/sources.list.d/`
4. `sudo wget http://repo.mosquitto.org/debian/mosquitto-stretch.list`
5. `sudo apt-get install mosquitto`

A.5.3 Install the Eclipse Paho Library

1. `sudo apt-get install python-dev`
2. `sudo apt-get install python-pip`
3. `pip install paho-mqtt`

A.5.4 Testing the Mosquitto MQTT Server Installation

1. `mosquitto -v`

A.6 Set up Static IP Address in Raspberry Pi

1. `sudo nano /etc/dhcpcd.conf`
2. Add the following lines and then save/exit:
```
interface eth0
static ip_address=192.168.0.2/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1
interface wlan0
static ip_address=192.168.0.2/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1
```
3. `sudo reboot`
4. `ip a`

A.7 Download and Run Source File

1. `git clone https://github.com/tracys9522/SeniorDesign_ioT.git`

Appendix B

Source Code

B.1 Bluetooth

This section contains the code for the light sensor and the code for establishing Bluetooth connection with the Raspberry Pi from the Arduino Uno.

B.1.1 light_sensor.ino

```
 /*
Source credit to:
https://github.com/adafruit/Adafruit_nRF8001/blob/master/examples/echoDemo/echoDemo.ino
https://github.com/adafruit/Adafruit_TSL2591_Library
*/
#include <SPI.h>
#include <SoftwareSerial.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BLE_UART.h"
#include "Adafruit_TSL2591.h"

// Connect CLK/MISO/MOSI to hardware SPI
// e.g.  On UNO & compatible:  CLK = 13, MISO = 12, MOSI = 11
// connect SCL to analog 5
// connect SDA to analog 4
// connect Vin to 3.3-5V DC
// connect GROUND to common ground

#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RDY      // This should be an interrupt pin, on Uno thats 2 or 3
#define ADAFRUITBLE_RST 9
Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ, ADAFRUITBLE_RDY, ADAFRUITBLE_RST);
Adafruit_TSL2591 tsl= Adafruit_TSL2591(2591); //pass in a number for the sensor identifier (for
your use later)

/*
* BLUETOOTH SETUP
*/
void setup(void)
{
Serial.begin(9600);
```

```
while(!Serial); // Leonardo/Micro should wait for serial init
      Serial.println(F("Adafruit Bluefruit Low Energy nRF8001 Print echo demo"));
      BTLEserial.setDeviceName("Alexa"); /* 7 characters max!  */
      BTLEserial.begin();
      if (tsl.begin())
      {
            Serial.println(F("Found a TSL2591 sensor"));
      }
      else
      {
            Serial.println(F("No sensor found ...  check your wiring?"));
            while (1);
      }
      /* Configure the sensor */
      configureSensor();
}
/*!
      Constantly checks for new events on the nRF8001
*/
aci_evt_opcode_t laststatus = ACI_EVT_DISCONNECTED;
void loop()
{
      // Tell the nRF8001 to do whatever it should be working on.
      BTLEserial.pollACI();

      // Ask what is our current status
      aci_evt_opcode_t status = BTLEserial.getState();
      Serial.println(status);
      // Pring if the status changed
      if (status != laststatus) {
            if (status == ACI_STATUS_ERROR_BUSY){
                  Serial.println(F("BUSY"));
            }
            if (status == ACI_EVT_DEVICE_STARTED) {
                  Serial.println(F("* Advertising started"));
            }
            if (status == ACI_EVT_CONNECTED) {
                  Serial.println(F("* Connected!"));
            }
            if (status == ACI_EVT_DISCONNECTED) {
                  Serial.println(F("* Disconnected or advertising timed out"));
            }
            // OK set the last status change to this one
            laststatus = status;
      }
      if (status == ACI_EVT_CONNECTED)
            Serial.println(F("* Data Sent!"));
            broadcastRead();
            delay(1000);
      }
      if (status == ACI_EVT_CONNECTED) {
            // Check for data...
            if (BTLEserial.available()) {
                  Serial.print("* ");
```

```
            Serial.print(BTLEserial.available());
            Serial.println(F(" bytes available from BTLE"));
        }
        // OK while we still have something to read, get a character and print it out
        while (BTLEserial.available()) {
            char c = BTLEserial.read();
            Serial.print(c);
        }

        // Next up, see if we have any data to get from the Serial console
        if (Serial.available()) {
            // Read a line from Serial
            Serial.setTimeout(100); // 100 millisecond timeout
            String s = Serial.readString();

            // We need to convert the line to bytes, no more than 20 at this time
            uint8_t sendbuffer[20];
            s.getBytes(sendbuffer, 20);
            char sendbuffersize = min(20, s.length());

            Serial.print(F("Sending ->"));
            Serial.print((char *)sendbuffer);
            Serial.println("¨");

            // write the data
            BTLEserial.write(sendbuffer, sendbuffersize);
        }
    }
}

/*
* LIGHT SENSOR
*/
void simpleRead(void)
{
    // Simple data read example.  Just read the infrared, fullspecrtrum diode
    // or 'visible' (difference between the two) channels.
    // This can take 100-600 milliseconds!  Uncomment whichever of the following you want
to read
    uint16_t x = tsl.getLuminosity(TSL2591_VISIBLE);
    //uint16_t x = tsl.getLuminosity(TSL2591_FULLSPECTRUM);
    //uint16_t x = tsl.getLuminosity(TSL2591_INFRARED);
    Serial.print(F("[ ")); Serial.print(millis()); Serial.print(F(" ms ] "));
    Serial.print(F("Luminosity:  "));
    Serial.println(x, DEC);
}

/*
* Show how to read IR and Full Spectrum at once and convert to lux
*/

void advancedRead(void)
{
    // More advanced data read example.  Read 32 bits with top 16 bits IR, bottom 16 bits
```

```
full spectrum
      // That way you can do whatever math and comparisons you want!
      uint32_t lum = tsl.getFullLuminosity();
      uint16_t ir, full;
      ir = lum >> 16;
      full = lum  0xFFFF;
      Serial.print(F("[ ")); Serial.print(millis()); Serial.print(F(" ms ] "));
      Serial.print(F("IR: ")); Serial.print(ir); Serial.print(F(" "));
      Serial.print(F("Full:  ")); Serial.print(full); Serial.print(F(" "));
      Serial.print(F("Visible:  ")); Serial.print(full - ir); Serial.print(F(" "));
      Serial.print(F("Lux:  ")); Serial.println(tsl.calculateLux(full, ir));
}
void broadcastRead(void)
{
      // Broadcast the read data through the bluetooth connection
      uint32_t lum = tsl.getFullLuminosity();
      uint16_t ir, full;
      ir = lum >> 16;
      full = lum  0xFFFF;
      BTLEserial.print(tsl.calculateLux(full, ir));
      Serial.print(tsl.calculateLux(full, ir));
}
void configureSensor(void)
{
      // You can change the gain on the fly, to adapt to brighter/dimmer light situations
      //tsl.setGain(TSL2591_GAIN_LOW); // 1x gain (bright light)
      tsl.setGain(TSL2591_GAIN_MED); // 25x gain
      // tsl.setGain(TSL2591_GAIN_HIGH); // 428x gain
      // Changing the integration time gives you a longer time over which to sense light
      // longer timelines are slower, but are good in very low light situtations!
      tsl.setTiming(TSL2591_INTEGRATIONTIME_100MS); // shortest integration time (bright light)
      // tsl.setTiming(TSL2591_INTEGRATIONTIME_200MS);
      // tsl.setTiming(TSL2591_INTEGRATIONTIME_300MS);
      // tsl.setTiming(TSL2591_INTEGRATIONTIME_400MS);
      // tsl.setTiming(TSL2591_INTEGRATIONTIME_500MS);
      // tsl.setTiming(TSL2591_INTEGRATIONTIME_600MS); // longest integration time (dim light)
      /* Display the gain and integration time for reference sake */
      //Serial.println(F("------------------------------------"));
      //Serial.print (F("Gain:  "));
      tsl2591Gain_t gain = tsl.getGain();
      switch(gain)
      {
            case TSL2591_GAIN_LOW:
                  //Serial.println(F("1x (Low)"));
                  break;
            case TSL2591_GAIN_MED:
                  //Serial.println(F("25x (Medium)"));
                  break;
            case TSL2591_GAIN_HIGH:
                  //Serial.println(F("428x (High)"));
                  break;
            case TSL2591_GAIN_MAX:
                  //Serial.println(F("9876x (Max)"));
                  break;
```

```
    }
    //Serial.print (F("Timing:  "));
    //Serial.print((tsl.getTiming() + 1) * 100, DEC);
    //Serial.println(F(" ms"));
    //Serial.println(F("------------------------------------"));
    //Serial.println(F(""));
}
```

B.2 Bing Search API

This section consists of the Bing Search API, which was used to aggregate training data for the deep learning model.

B.2.1 search_bing_api.py

```
'''
    Source credit to:
https://www.pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-image-dataset/
'''
# USAGE
# python search_bing_api.py --query "sleeping in bed" --output dataset/sleep
# python search_bing_api.py --query "eating food" --output dataset/eat
# python search_bing_api.py --query "reading on chair" --output dataset/read
# python search_bing_api.py --query "watching TV" --output dataset/watchTV

# import the necessary packages
from requests import exceptions
import argparse
eimport requests
import cv2
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-q", "--query", required=True,
          help="search query to search Bing Image API for")
ap.add_argument("-o", "--output", required=True,
          help="path to output directory of images")
args = vars(ap.parse_args())

# set your Microsoft Cognitive Services API key along with (1) the
# maximum number of results for a given search and (2) the group size
# for results (maximum of 50 per request)
API_KEY = "#####PUT YOUR API KEY HERE#######"
MAX_RESULTS = 2000
GROUP_SIZE = 50

# set the endpoint API URL
URL = ‘‘https://api.cognitive.microsoft.com/bing/v7.0/images/search"

# when attemping to download images from the web both the Python
# programming language and the requests library have a number of
# exceptions that can be thrown so let’s build a list of them now
# so we can filter on them
```

```
#EXCEPTIONS = set([IOError, FileNotFoundError,
#            exceptions.RequestException, exceptions.HTTPError,
#            exceptions.ConnectionError, exceptions.Timeout])

# store the search term in a convenience variable then set the
# headers and search parameters
term = args["query"]
headers = {"Ocp-Apim-Subscription-Key" :  API_KEY}
params = {"q":  term, ‘‘offset":  0, ‘‘count":  GROUP_SIZE}

# make the search
print("[INFO] searching Bing API for ’{}’".format(term))
search = requests.get(URL, headers=headers, params=params)
search.raise_for_status()

# grab the results from the search, including the total number of
# estimated results returned by the Bing API
results = search.json()
estNumResults = min(results["totalEstimatedMatches"], MAX_RESULTS)
print("[INFO] {} total results for ’{}’".format(estNumResults, term))

# initialize the total number of images downloaded thus far
total = 0

# loop over the estimated number of results in ‘GROUP_SIZE‘ groups
for offset in range(0, estNumResults, GROUP_SIZE):
          # update the search parameters using the current offset, then
          # make the request to fetch the results
          print("[INFO] making request for group - of ...".format(
                    offset, offset + GROUP_SIZE, estNumResults))
          params["offset"] = offset
          search = requests.get(URL, headers=headers, params=params)
          search.raise_for_status()
          results = search.json()
          print("[INFO] saving images for group {}-{} of {}...".format(
                    offset, offset + GROUP_SIZE, estNumResults))

          # loop over the results
          for v in results["value"]:
                    # try to download the image
                    try:
                              # make a request to download the image
                              print("[INFO] fetching:  {}".format(v["contentUrl"]))
                              r = requests.get(v["contentUrl"], timeout=30)

                              # build the path to the output image
                              ext = v["contentUrl"][v["contentUrl"].rfind("."):]
                              p = os.path.sep.join([args["output"], "{}{}".format(
                                        str(total).zfill(8), ext)])

                              # write the image to disk
                              f = open(p, "wb")
                              f.write(r.content)
                              f.close()
```

```
                    # catch any errors that would not unable us to download the
                    # image

                    except Exception as e:
                              print("exception")
                              continue
                              # check to see if our exception is in our list of
                              # exceptions to check for
                              #if type(e) in EXCEPTIONS:
                                        # print("[INFO] skipping:  {}".format(v["contentUrl"]))
                                        # continue
                    # try to load the image from disk

                    image = cv2.imread(p)

                    # if the image is `None` then we could not properly load the
                    # image from disk (so it should be ignored)

                    if image is None:
                              print("[INFO] deleting:  {}".format(p))
                              os.remove(p)
                              continue

                    # update the counter
                    total += 1
```

B.3 CNN

In this section, we first used the trained CNN model by using the images from the Search-Bing API, then we classify the activity label. In terms of how to train a CNN model, please refer to Appendix A.7 to download the complete source files.

B.3.1 classify.py

```
'''
     Credit Source for basic program layout:
     https://www.pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnns/
     Author:  Darence Lim and Tracy Sun
     USAGE
     python classify.py --model pokedex.model --labelbin lb.pickle --image examples/eating.png
'''

# import the necessary packages
from phue import Bridge
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import numpy as np
import argparse
import imutils
import pickle
import cv2
```

```
import os
import time

# Connect to phue bridge
b = Bridge('192.168.1.2')
b.connect()
b.get_api()

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True, help="path to trained model model")
ap.add_argument("-l", "--labelbin", required=True, help="path to label binarizer")
ap.add_argument("-i", "--image", required=True, help="path to input image")
args = vars(ap.parse_args())

# load the image
image = cv2.imread(args["image"])
output = image.copy()

# pre-process the image for classification
image = cv2.resize(image, (96, 96))
image = image.astype("float") / 255.0
image = img_to_array(image)
image = np.expand_dims(image, axis=0)

# load the trained convolutional neural network and the label
print("[INFO] loading network...")
model = load_model(args["model"])
lb = pickle.loads(open(args["labelbin"], "rb").read())

# classify the input image
print("[INFO] classifying image...")
proba = model.predict(image)[0]
idx = np.argmax(proba)
label = lb.classes_[idx]

# we'll mark our prediction as "correct" of the input image filename
# contains the predicted label text (obviously this makes the
# assumption that you have named your testing image files this way)
filename = args["image"][args["image"].rfind(os.path.sep) + 1:]

# set brightness level to the recommended light level by National Optical Astronomy Observatory
f = open("lux.txt", "r")
currentLux = int(f.read())
f.close()

# watch TV range between 125 - 175
if(label=="watchTV"):
      desiredLux = 150
      while (currentLux > 175 or currentLux < 125):
            temp1 = brightness = b.get_light(5,'bri')
            factor = 0
      if (currentLux < 125):
            b.set_light(5, 'on', True)
```

```
            if(desiredLux - currentLux > 1000):
                  factor = 35
            if(desiredLux - currentLux > 500):
                  factor = 23
            if(desiredLux - currentLux > 200):
                  factor = 15
            if(desiredLux - currentLux > 100):
                  factor = 10
            if(desiredLux - currentLux <= 100):
                  factor = 7
            temp1 = temp1 + factor
            if (temp1 > 255):
                  temp1 = 255
            b.set_light(5, 'bri', temp1)
      if (currentLux > 175):
            if(currentLux - desiredLux > 1000):
                  factor = 35
            if(currentLux - desiredLux > 500):
                  factor = 23
            if(currentLux - desiredLux > 200):
                  factor = 15
            if(currentLux - desiredLux > 100):
                  factor = 10
            if(currentLux - desiredLux <= 100):
                  factor = 7
            temp1 = temp1 - factor
            if (temp1 <= 0 ):
                  temp1 = 0
                  b.set_light(5, 'on', False)
            b.set_light(5, 'bri', temp1)
      time.sleep(1)
      f = open("lux.txt", "r")
      currentLux = int(f.read())
      f.close()

# eating range 200 - 300
if(label=="eating"):
      desiredLux = 250
      while (currentLux > 300 or currentLux < 200):
            temp1 = brightness = b.get_light(5,'bri')
            factor = 0
            if (currentLux < 200):
                  b.set_light(5, 'on', True)
                  if(desiredLux - currentLux > 1000):
                        factor = 35
                  if(desiredLux - currentLux > 500):
                        factor = 23
                  if(desiredLux - currentLux > 200):
                        factor = 15
                  if(desiredLux - currentLux > 100):
                        factor = 10
                  if(desiredLux - currentLux <= 100):
                        factor = 7
                  temp1 = temp1 + factor
```

```
                if (temp1 > 255):
                        temp1 = 255
                b.set_light(5, 'bri', temp1)
        if (currentLux > 300):
                if(currentLux - desiredLux > 1000):
                        factor = 35
                if(currentLux - desiredLux > 500):
                        factor = 23
                if(currentLux - desiredLux > 200):
                        factor = 15
                if(currentLux - desiredLux > 100):
                        factor = 10
                if(currentLux - desiredLux <= 100):
                        factor = 7
                temp1 = temp1 - factor
                if (temp1 <= 0 ):
                        temp1 = 0
                        b.set_light(5, 'on', False)
                b.set_light(5, 'bri', temp1)
        time.sleep(1)
        f = open("lux.txt", "r")
        currentLux = int(f.read())
        f.close()

# reading range between 450 - 550
if(label=="readingChair"):
      desiredLux = 500
      while (currentLux > 550 or currentLux < 450):
            temp1 = brightness = b.get_light(5,'bri')
            factor = 0
            if (currentLux < 450):
                  b.set_light(5, 'on', True)
                  if(desiredLux - currentLux > 1000):
                        factor = 35
                  if(desiredLux - currentLux > 500):
                        factor = 23
                  if(desiredLux - currentLux > 200):
                        factor = 15
                  if(desiredLux - currentLux > 100):
                        factor = 10
                  if(desiredLux - currentLux <= 100):
                        factor = 7
                  temp1 = temp1 + factor
                  if (temp1 > 255):
                        temp1 = 255
                  b.set_light(5, 'bri', temp1)
            if (currentLux > 550):
                  if(currentLux - desiredLux > 1000):
                        factor = 35
                  if(currentLux - desiredLux > 500):
                        factor = 23
                  if(currentLux - desiredLux > 200):
                        factor = 15
                  if(currentLux - desiredLux > 100):
```

```
                    factor = 10
               if(currentLux - desiredLux <= 100):
                    factor = 7
               temp1 = temp1 - factor
               if (temp1 <= 0 ):
                    temp1 = 0
                    b.set_light(5, 'on', False)
               b.set_light(5, 'bri', temp1)
          time.sleep(1)
          f = open("lux.txt", "r")
          currentLux = int(f.read())
          f.close()

# sleeping turn off the lights
if(label=="sleepingBed" or label=="sleepingCouch"):
     b.set_light(5, 'on', False)

# build the label and draw the label on the image
activity = label
label = ":  :.2f%".format(label, proba[idx] * 100)
output = imutils.resize(output, width=400)
cv2.putText(output, label, (10, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

# show the output image
print("[INFO] ".format(label))
brightness = b.get_light(5,'bri')
print("[BRIGHTNESS] ".format(brightness))

# write label and brightness into text file
f = open("result.txt","wb")
f.write(activity)
f.write(" ")
f.write(str(brightness))
```

B.4 MQTT

In this section, we included the source code of the Python script to run the MQTT server on the Raspberry Pi, as well as the source code to the iOS app.

B.4.1 mqtt.py

```
import paho.mqtt.client as mqtt
import paho.mqtt.publish as publish

def connectionStatus(client, userdata, flags, rc):
     mqttClient.subscribe("connect")
def messageDecoder(client, userdata, msg):
message = msg.payload.decode(encoding='UTF-8')
     if message == "get brightness":
          mqttClient.publish("to iOS", brightness)
          print("Brightness sent to iOS device!")
     if message == "get activity":
```

```
            mqttClient.publish("to iOS", label)
            print("Activity sent to iOS device!")
      else:
            print("Unknown message!")

#Instantiate Eclipse Paho as mqttClient
mqttClient = mqtt.Client("RPI")

#Set calling function functions to mqttClient
mqttClient.on_connect = connectionStatus
mqttClient.on_message = messageDecoder

#Connect client to server
mqttClient.connect("SmartSys")

#Monitor client activity forever
mqttClient.loop_forever()
```

B.4.2 SmartSys iOS App

```
import UIKit
import CocoaMQTT

class ViewController:  UIViewController {

      @IBOutlet weak var brightnessPercentage:  UILabel!
      @IBOutlet weak var brightnessLabel:  UILabel!
      @IBOutlet weak var activityLabel:  UILabel!

      var mqttClient = CocoaMQTT(clientID: "iOS Device", host:  "SmartSys", port:  1883)

      override func viewDidLoad() {
            super.viewDidLoad()
      }

      override func didReceiveMemoryWarning() {
            super.didReceiveMemoryWarning()
      }

      @IBAction func connect(_ sender:  UIButton) {
            mqttClient.connect()
      }

      @IBAction func disconnect(_ sender:  UIButton) {
            mqttClient.disconnect()
      }

      @IBAction func refreshBrightness(_ sender:  UIButton) {
            mqttClient.publish("connect", withString:  "get brightness")
            mqttClient.subscribe("to iOS")
            mqttClient.didReceiveMessage = { mqtt, message, id in
                  let brightness = String(Int(message.string!)!  * 100 / 256)
                  self.brightnessPercentage.text = brightness + "
```

```
                self.brightnessLabel.text = "Philips Hue:  " + (message.string!)  + "/256"
            }
        }

        @IBAction func refreshActivity(_ sender:  UIButton) {
            mqttClient.publish("connect", withString:  "get activity")
                mqttClient.subscribe("to iOS")
                mqttClient.didReceiveMessage = { mqtt, message, id in
                self.activityLabel.text = "Activity:  " + (message.string!)
            }
        }
}
```