CANOPY

Michael Chapman, CSE, John Curci, EE, Justin Thibodeau, CSE, and Zachary Windoloski, CSE

***Abstract* — Two of the largest contributors to home energy costs in the United States are heating and cooling [1]. The Canopy automated shade system helps consumers cut down on energy use by controlling the amount of sunlight that enters a home. The system integrates with the Nest automated thermostat over WiFi, in order to determine whether the home is currently being heated or cooled. This information is combined with input from a light sensor at each shade unit, in order to determine the position of the shade. Nest integration allows Canopy to take advantage of Nest's machine learning, which improves the system's heating and cooling efficiency. Another major focus of the system is convenience. Through a custom Android application, users can control their shades remotely, and view the light levels at each window equipped with a Canopy shade system. Settings controlled through the app determine the behavior of the shades, and a switch on the shade unit allows for direct control of the shades.**

I. Introduction

THE motivation for our project came from the high cost associated with heating and cooling homes and buildings. In 2013, the average American with natural gas spent $680 per year on residential heating [2], and those with air conditioners spent an additional $280 per year in 2010 [3]. Considering the fact that 26% of American adults don't have any money saved, and approximately 38 million households live paycheck to paycheck, this cost of about $1000 per year is a significant burden on many people [4]. However, heating and cooling buildings is far more than just a financial problem. It poses a problem on the global scale as well, as the use of HVAC systems takes a direct toll on the environment. It is estimated that air conditioners release approximately 100 million tons of carbon dioxide every year [5], and in 2006, approximately 8% of US Carbon Dioxide emissions came from residential HVAC [6].

There are several existing solutions that attempt to solve this problem by making HVAC systems more efficient. The Nest thermostat, which has received significant publicity in the past few years, saves energy by providing more intelligent temperature control compared to a standard thermostat [7]. According to a recent report, the Nest thermostat reduces homeowners' costs by an average of 12% on their heating bill and 15% on their cooling bill [7]. Ceiling fans made by the company Big Ass Fans provide another solution to the problem. These fans integrate with the Nest thermostat, and make Nest more effective by allowing the system to control the airflow in a building [8]. The energy savings are most significant in the summer, when the added air movement allows a user to raise the temperature on their thermostat without feeling warmer [8]. Big Ass Fans claims that for every degree the Nest thermostat is raised in the summer, customers will save 5% on their energy bill [8]. A third solution is provided by the company Lutron. Lutron makes an automated shades system that saves energy by providing season-based settings: "winter warm" keeps the shades open to let in light during the winter, while "summer cool" closes the shades to keep out light during the summer [9]. Since sunlight is a significant source of heat gain in a home, controlling the light entering through windows depending on the season helps make air conditioners more efficient in the summer and heaters more efficient in the winter [10].

Like Lutron, we created an automated window shades system that controls indoor temperature by changing the amount of sunlight coming in through the windows. However, we added customization beyond Lutron's two energy-based settings. In addition, like Big Ass Fans, we integrated with the Nest thermostat to provide additional intelligence and energy savings. The specifications for our design and our implementation to meet them are shown in Table I.

TABLE I
SPECIFICATIONS

Goal	Implementation
Outdoor Light Level Sensor With +/-50% precision over 0.5-120k lux	BH1620FVC-TR Photodiode
Stable Shade to Internet Communication	IEEE 802.11
Motor Producing > 4 kg*cm	Dynamixel AX-12A
User to Device Communication Security	PSK Encryption
Web Based Data Storage	AWS
Data Integrity & Security	AWS Encrypted Signatures
Continuous Power Source	Standard US 120V Wall AC
Remote Control	Android App
Manual Control	Physical Switch on Device

II. Design

A. Overview

Figure 1 shows the design of our automated shade unit. The main control unit is a WiFi integrated microcontroller which continually communicates an Amazon Web Services database and the user. This control unit serves as a processing and

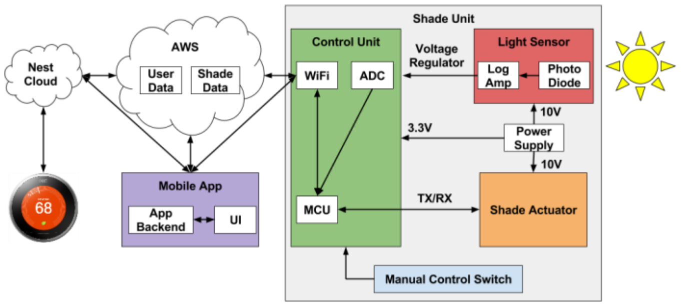

Figure 1: Canopy Block Diagram

communication hub, with inputs originating from the Nest cloud, the user via Android app through the database, a light sensor, and a physical control switch. For our control unit, we considered three ideas: a centralized hub controlling multiple shades, an external server acting as the hub, and distributed control units built into each shade unit. Distributed units were chosen since each device would require a microcontroller and some form of wireless communication regardless of our approach. Considering that our computation requirements are relative low, microcontrollers placed within each unit are sufficiently powerful for this application. We use a photodiode-based light sensor to measure outdoor light levels. This sensor element was chosen for its ability to detect a wide range of light levels including darkness and bright sunlight. The Canopy Android application allows the user to control his or her shades remotely, and to view the light levels at each window in their home as measured by our light sensors. The output of our system is a shade actuator which opens and closes the window shade. For our actuator, we considered a stepper motor but decided on a servo motor because it is able to provide more precise position control. For a power supply, we considered battery or solar powered approaches, but determined that using wall AC would be more convenient for the user and would allow for continuous operation in all weather conditions. We use a switched-mode design to provide a compact and energy-efficient power source to provide the necessary voltages and currents to each system.

B. Block 1: Light Sensor

One of the core functions of Canopy is to provide an additional input for Nest's intelligent heating and cooling system. This is achieved using a light sensor, which measures the brightness level at each window in the user's home, and feeds that information into the Canopy control unit. This serves the dual purpose of improving the overall efficiency of the Nest system, and helping Canopy's algorithms balance the energy savings associated with opening or closing the shade against the user's light level preferences.

The sensor is based around the BH1620FVC-TR light sensing IC from Rohm semiconductors, which provides a linear output current across brightness levels spanning five orders of magnitude [11]. This relationship is shown in Figure 3. The LOG101AID logarithmic transimpedance amplifier from Texas Instruments was used to convert this signal, which included currents in the range of nanoamperes at the low end, into a voltage that could be read in by our microcontroller [12]. The light sensing element is able to produce a distinct output across a wide range of illuminance levels. Its measurement coverage includes the brightness range of outdoor light from 0.5 lux (bright moonlight) to 120,000 lux (very bright sunlight) [13].

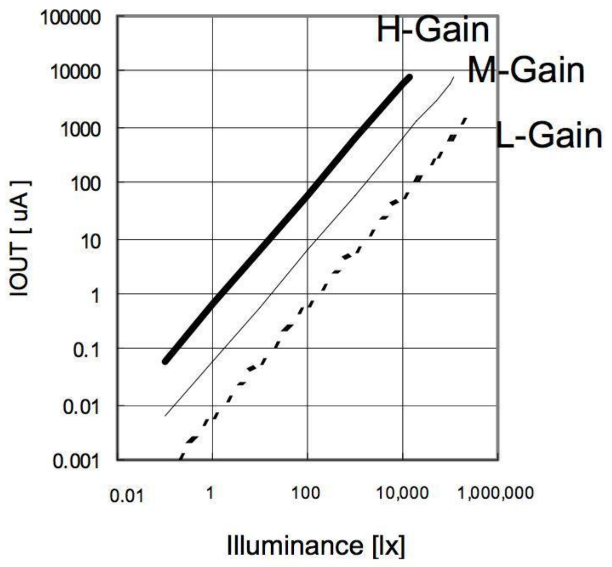

Figure 2: Gain vs Light Level BH1620FVC-TR

C. Block 2: Shade Actuator

The shade actuator is a key component of the system that is relied upon to adjust the angle of the horizontal blinds. For our project, the team chose Achim Home Furnishing's brand of venetian blinds to allow for granular light levels [14]. The actuator needed to meet a few general requirements: first of all, it needed to have enough torque to rotate the shaft that turns the blinds. We calculated that the blinds require a maximum of 4 kg-cm to rotate, so the actuator must be rated for at least 4 kg-cm of standing torque. In addition, the blinds move from -90 degrees (closed, pointing down) to 0 degrees (fully open) to 90 degrees (closed, pointing up) for a full rotation of 180 degrees. Therefore, our actuator must be capable of rotating at least 180 degrees.

Having considered these requirements, the Dynamical AX-12A servo was chosen as our actuator [15]. The AX-12A provides a maximum of 10kg-cm of torque at 10V, which is more than sufficient for our purpose [15]. In addition, the AX-12A provides 300 degrees of rotation [15]. This allows us to achieve the full 180-degree rotation without any modifications. In order to perform the rotation, the motor is coupled to the rotating shaft inside the top of the shade. As the motor turns the shaft, the threads running through horizontal blinds move up or down, which changes their angle of the horizontal blinds.

Unlike most servos that are controlled with a PWM signal, the communication with the AX-12A is done via half duplex UART, where one wire is used for sending and receiving data [15]. Two 0xFF bytes indicate the start of the signal, followed by two bytes for the ID of the AX-12A unit, one byte for the length of the signal, and bytes for the instruction or error message. In order to allow for both sending and receiving, the motor's half duplex UART must be converted into the full duplex used by our microcontroller, where separate ports are used for transmitting (TX) and receiving (Rx). A simple tri-state buffer circuit shown below in Figure 3 was built to do this conversion [15].

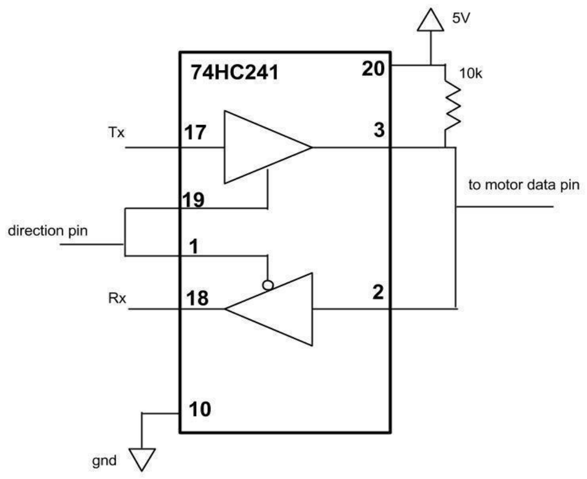

Figure 3: Full Duplex to Half Duplex

In addition to the actuator, a SPDT rocker switch was added to the frame of the shade in order to provide manual control over the angle of the shade. Each of the throws connects to an input pin on our microcontroller, to allow either clockwise or counterclockwise rotation of the blinds depending on the direction that the switch was pressed.

D. *Block 3: Control Unit*

Our control unit is the brains behind the entire canopy device. It needed to be able to take in an analog voltage signal from our light sensor and translate this into a usable light level value. It needed to be able to send and receive bytes of data over a TX/RX connection to our motor to set shade positions and receive feedback from the motor. It needed to be capable of communicating to the internet in order to access user defined settings held in an Amazon Web Services (AWS) database [16], Nest thermostat information from Nest's own data cloud [17]. Finally, it needed to be able to communicate directly with a user's phone in order to be set up on their in-home network.

To handle the required data storage, and model user to device relationships, we created a DynamoDB instance within AWS [16]. AWS provides us with a database that requires pre-shared credentials in order to access data, allowing us to ensure that all data is coming from a trusted place. It is also scalable, meaning that while testing we are only using a very small database, but when this project would eventually grow to multiple shades and users, the database will scale up to match demand.

The Adafruit Huzzah board [18] was chosen to perform our computation tasks for several reasons. The onboard ESP8266 processor provides us a microcontroller with full WiFi support to act as both an access point and as a client. There are low power modes available that allow the board to disable WiFi communication in order to save power, which will allow the system to sense data more frequently than it communicates over the internet. Also included is a watchdog timer, which allows the board to be put into a very low power state when not actively sensing or communicating, while still able to receive an interrupt signal to wake up at specific times or when the user interacts with the device.

When acting in access point mode, the board can be directly connected to and will respond to POST requests containing an SSID, password, and AWS id. When the device receives one of these requests, it sends an OK response back, stores the given network information in its EEPROM, and switches to its client mode. We encrypt the network password using a pre-shared key so that it is never sent as raw text. When the board is in client mode, it directly connects to the network stored in its EEPROM. Upon entering this mode, the board performs two initial tasks. First, it sends an NTP request via UDP to get the current UTC time. This should occur on a daily basis and any time the device is restarted to ensure that the correct time is always set. The board then sends an HTTP request to ip-api.com/xml, in order to find its current latitude and longitude value. This information is combined with the current time to determine the approximate angle of elevation of the sun. Once these two tasks are complete, the board sends queries to the database to get all behavior data for the Shade entry using the id that was given to it during setup, and to update the current state of the device. This is done through a series of state machines so that we can send a request and continue doing other work until a response comes back or the request times out and we try again.

If the shade is set to be in user or room mode, a second query is made to get the behavior from the appropriate object. If the shade is set to be in schedule mode, it queries for all schedule items linked to the shade and does a 2-way search to decide which item to follow. Schedules only have a notion of weekday, not date, so to decide which to follow it needs to either find the closest item to the current time that occurred earlier in the week, or the item furthest away later in the week since this would wrap around in a circle.

In order to access Nest thermostat information, we need an SSL connection, which we cannot open directly from the Huzzah board. To get around this, we leverage AWS Lambda [21] which can run JavaScript code when it detects changes in DynamoDB. Through this, we detect the device updating its entry in the database then send queries to Nest using their API to get temperature data from the linked thermostat. Nest throttles queries to any given thermostat that occur more than once per minute, so we also store a timestamp of the last update to ensure our requests never exceed this rate.

During normal operation, the device reads in sunlight data being provided from our sensor on its ADC connection as a value between 0 and 1023. We map this value onto a set of experimentally chosen brightness windows including bright sun, cloudy, and dark. These windows are shown in Table 2. The sunlight level is used in many of our behavior modes, and is also pushed to AWS every 15 minutes so that a visual graph of light levels can be viewed from within the app.

TABLE II
LIGHT LEVEL THRESHOLDS

Approximate Light Level in Lux	Log of Light Level	Description
10,000 - 100,000	>700	Bright Sun
1,000 - 10,000	400 - 700	Cloudy
100 - 1,000	200 - 400	Very Cloudy
0.1 - 100	<200	Dark

The behavior modes we have implemented are: opened, closed, manual, preservation, convenience, and energy efficiency. When set to open or closed, the shade will stay at that position until it is changed to something else through the app or scheduling. The manual behavior keeps the shade at its current position only allowing the physical switch to change it. Preservation mode is used when the user wants to protect items from the sun. The shade will be open at a low enough light level, but when it detects direct sunlight above the "Bright Sun" threshold, the shade will close, preserving light sensitive valuables. The convenience mode is a standard use case where the shade will open at sunrise and stay open until the sun sets. Sunrise and sunset times are calculated using the latitude and longitude values gathered earlier.

Energy efficiency is the most prominent mode of the device and tells the shade to act in whichever way will help make heating or cooling more efficient. To do this, the controller first uses the temperature data provided from Nest to see if the thermostat is attempting to heat or cool the room. It then compares this with the current light level. If the thermostat is trying to heat the home, and there is ample sunlight above the "Bright Sun" threshold, the shades open to let it in. If it is too dark, below the "Very Cloudy" threshold, the shades close to insulate and keep in heat. Conversely, if the thermostat is trying to cool the home, and there is bright sunlight, the shades close to block it out. If it becomes dark, the shades open to try and minimize their insulating effect.

At any point, the user has the ability to manually override the behavior of the shade using the rocker switch on the side of the frame. When a manual override is detected, a button will appear in the mobile application allowing the user to cancel that manual override on the application, and reset the shade to its programmed behavior. If the user doesn't press this cancel button, the shade will reset automatically after a grace period. After the first manual override is detected, the microcontroller will give the user a 15-minute grace period before resetting itself. However, if the microcontroller continues to receive overrides after it has reset, this grace period will increase from 15 minutes, to 30 minutes, to 1 hour, and then to 24 hours, in order to respond intelligently to the user's behavior. If the user stops manually overriding the behavior, the grace period will drop back to 15 minutes.

Once everything above has occurred, the microcontroller sends a packet of data over its TX connection to the shade actuator telling it to set the shade slats to the specific angle defined by the behavior or manual override.

E. Block 4: Mobile Application

The mobile application is the interface for users to communicates with their shades and the overall system. In order to save settings and shade information, and to share that information between the application and the control unit, we needed a common database. We chose to use AWS's DynamoDB [16]. The application is able to push and pull data from this database by using provided libraries created for Android by Amazon.

Within the app, users set up a username and password to uniquely identify their account which links all of their shades, rooms, and schedules together. The password they provide is hashed using an md5 hash so no plaintext passwords are stored within AWS. Users are able to link this account with their Nest account by opening an internal browser which loads an authorization page and asks the user for their consent. After this is done, they can add or remove shades from their system, group shades together by rooms, link shades to a specific Nest thermostat, and define the shades behavior. Figure 4 shows an example of a shade page.

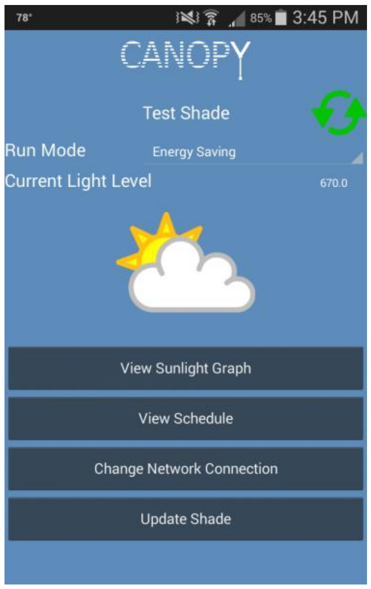

Figure 4: Shade View

A user can create a schedule for their shade, room, or home through the app which allows them to pick any of the behaviors described in Section D and place them at a start time and day of the week. When in schedule mode, the shade will follow the most recent behavior until another scheduled behavior comes up. Within the app there is a schedule view, shown in Figure 5, which allows the user to view their schedule in a pictorial format over time, with different images denoting different behaviors. This schedule setup is very similar to how the Nest thermostat itself handles schedules, making it easy for the user to transfer knowledge from their system to ours, or vice versa.

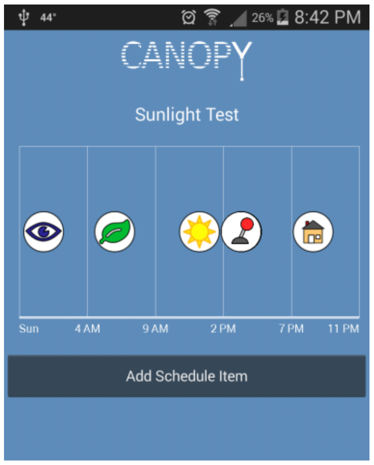

Figure 5: Schedule View

Another feature present for each shade is the ability to view a sunlight graph within the app. Every 15 minutes the shade sends its current sunlight reading to AWS. We then use all of those data points to create a graph through time of the light level that the shade has recorded. This gives the user an easy way to see the changes in light outside of that window and allow them to make smarter choices about their shade layouts. An example of this is shown in Figure 5.

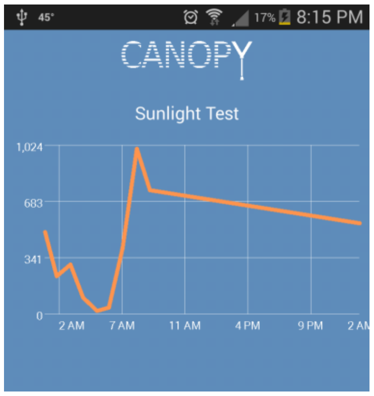

Figure 6: Sunlight Graph

When a shade is added to a system, the application allows the user to connect the shade to their local wireless network. To do this, the application first creates a shade object in AWS and then prompts the user to: select their network from a list of those visible to the app, enter the password for the network, and enter an id that would be printed on the shade in order to distinguish it from any other shades the user may own. The app then disconnects itself from its current WiFi network (if any) and connects directly to the shade's network card which is acting as an access point. After connecting, the app pushes the network SSID, a hashed version of the password, and the id of the shade object it created in AWS. It then waits several seconds to see if the shade was able to successfully connect and communicate with AWS before alerting the user of the success or failure.

When deciding which mobile platform to develop in, there were three main choices, Android, iOS, and Windows. Windows had limited libraries and tutorials available at the time of design so it was ruled out. The two remaining were very similar in their offered functionality, but we decided on Android since the majority of our own physical devices ran Android which would make testing much easier.

Because mobile application development is not taught in any of our courses, we learned almost everything this year We used android's multiple tutorials and code samples as our primary sources of information [19]. Since the application integrates with Amazon Web Services (AWS) for data storage, tutorials and code samples provided by Amazon were also used during development.

Development for this application was done through Android Studio [20]. It allowed for the easiest way to create and test the application. The application can be built through their IDE using the Java programming language. Android Studio also offers an emulator which was used to test the application without having to load it onto a physical device. This allowed for testing of the application and testing between the application and AWS to occur easily. To ensure that data was being sent correctly to AWS we used Amazon's developer console to directly view the tables.

III. Project Management

From MDR to FPR we were able to meet all of the deadlines that we had set for ourselves using a Gantt chart, as well as some additional functionality that we had not initially planned out. All of the timelines assigned were completed by the person that they were intended for. Continuing to use our weekly team meetings to ensure each member was on pace to finish played a crucial role in the success of our project. We also used that time to work together for parts of the project that required communication with each other so that everyone was always on the same page moving forward and to ensure there was nothing blocking progress.

Our team was comprised of one EE and three CSE majors. This worked well since our project had a significant software focus. We tasked the EE in our group, John, with the light sensor and power systems since that is where his strengths lie. He designed our light sensor circuit, our PCB to house all electronics, and assembled these electronics. Michael, a CSE, worked with the shade actuator which is mechanically heavy and integrated that subsystem with the control unit. He also took control of the manual override and finalizing the shade behavior logic. Zachary, also a CSE, was responsible for the mobile application and collaborated with Justin to coordinate AWS database communication and setup. He put the bulk of the app together, figured out user login, and created our shade, room, and schedule management systems. Justin, the third CSE and Team Manager was responsible for the control unit's interfaces, WiFi setup and internet communication, as well as keeping the team on track. Towards the end of the project, Justin was able to help improve the usability of the app by adding a visual interface for schedules, light level graphs, and optimizing the overall app flow.

IV. Results

We performed an analysis of our system while it was running in order to determine its power usage. When idle, the system uses ~85mW, which remains unchanged during the device's WiFi setup. While the motor was running, there was a ~145mW peak, however the average power consumption remains very close to 85mW since the motor is only running for a few seconds at a time, not constantly. From this, we calculated that one shade uses roughly 2Wh/day when operating nonstop. For comparison, a 15W CFL bulb uses 30Wh when used for only 2 hours/day. Our devices usage is also comparable to running a 1500W space heater for ~5 seconds. For an average household which may use up to 20 shades, the overall power usage of the system would be just 40Wh/day.

We also did a cost analysis for the system. Our calculated development cost for a single unit and estimated production costs per unit, when producing 1000, are shown below in Table 3. At production, we are currently still accounting for the high cost of 3D printing a case, however we believe that this cost would be drastically minimized if we used an injection molding process, or adapted the system to be usable within any existing shades.

TABLE III
Cost Analysis

Part	Development Price	Production Price
PCB	$141.51	$1.52
Motor	$44.90	$37.50
Adafruit Huzzah	$9.95	$7.96
Light Sensor	$12.32	$6.05
Logarithmic Amp	$16.42	$9.77
Voltage Regulator	$2.76	$2.03
Power Adaptor	$19.55	$11.98
Resistors/Capacitors	$1.00	$1.00
AWS*	$0.00	$30.60
Case	$53.43	$53.43
Total	**$331.13**	**$179.23**

V. Conclusion

For FPR we had proposed: the electronics would be entirely self-contained on a PCB with no connections to a variable power supply or computer necessary, Nest integration would be complete, device to network setup would be achievable from within the app, shade behavior modes would be fully fleshed out and working, manual control would be fully implemented, and the app would contain all required functionality to set behaviors and schedules, organize objects, and have user authentication to allow for a multi-user system. All of these goals have been achieved with the additions of our sunlight data collection, updated app graphics and flow, automatic return from manual override, a housing to protect the electronics, and an analysis of the system's power consumption. Overall, we are very proud of what we have managed to accomplish across these 2 semesters and are very glad to have had this opportunity.

Acknowledgment

We would like to thank our Faculty Advisor, Professor Csaba Andras Moritz, for being a guiding hand as we made design decisions for our project and for helping us refine our ideas. We would also like to thank our faculty evaluators, Professor T. Baird Soules and Professor Zlatan Aksamija, for giving us useful feedback and criticism at various stages of development. Regarding technical advice, we would like to thank Professor Joseph Bardin for his advice on the amplifier stage of the light sensor, Professor Charles Malloch for his advice on implementing WiFi capabilities, and Daniel Bergman for his help with the motor setup and the associated machining.

References

[1] Energy.gov, 'Tips: Your Home's Energy Use | Department of Energy', 2015. [Online]. Available: http://www.energy.gov/energysaver/tips-your-homes-energy-use. [Accessed: 09- Dec- 2015].

[2] Eia.gov, 'Heating costs for most households are forecast to rise from last winter's level - Today in Energy - U.S. Energy Information Administration (EIA)', 2013. [Online]. Available: http://www.eia.gov/todayinenergy/detail.cfm?id=13311#tabs_SpotPriceSlider-3. [Accessed: 09- Dec- 2015].

[3] Carbonrally.com, 'Carbonrally – air conditioner costs', 2015. [Online]. Available: http://www.carbonrally.com/challenges/22-air-conditioner-costs. [Accessed: 09- Dec- 2015].

[4] CreditDonkey, "23 Dizzying Average American Savings Statistics", 2016. [Online]. Available: https://www.creditdonkey.com/average-american-savings-statistics.html. [Accessed: 03- Jan- 2016].

[5] Energy.gov, "Air Conditioning | Department of Energy", 2016. [Online]. Available: http://energy.gov/energysaver/air-conditioning. [Accessed: 03- Jan- 2016].

[6] C2es.org, 'Buildings Overview | Center for Climate and Energy Solutions', 2015. [Online]. Available: http://www.c2es.org/technology/overview/buildings. [Accessed: 09- Dec- 2015].

[7] J. McGrath, "Nest studied customers' heating and cooling savings, and the results are in", Digital Trends, 2015. [Online]. Available: http://www.digitaltrends.com/home/how-much-money-does-nests-smart-thermostat-save/. [Accessed: 02- Jan- 2016].

[8] Big Ass Fans Introduces Integration with Nest Learning Thermostat » Big Ass Fans, "Big Ass Fans Introduces Integration with Nest Learning Thermostat » Big Ass Fans", 2014. [Online]. Available: http://www.bigassfans.com/big-ass-fans-introduces-integration-nest-learning-thermostat/. [Accessed: 02- Jan- 2016].

[9] Lutron.com, "Shading Solutions from Lutron Provide Energy Saving Light Control", 2016. [Online]. Available: http://www.lutron.com/en-US/Residential-Commercial-Solutions/Pages/Residential-Solutions/ShadingSolutions.aspx. [Accessed: 03- Jan- 2016].

[10] Savewithsrp.com, "SRP: Home inspection to reduce heat gain", 2016. [Online]. Available: http://www.savewithsrp.com/DIY/heatgain.aspx. [Accessed: 03- Jan- 2016].

[11] Rhom Semiconductor. (2012, Feb.). “BH1620FVC Ambient Light Sensor IC” [Online]. Available: http://rohm fs.rohm.com /en/products/databook/datasheet/ic/sensor/light/ bh1620fvc-e.pdf [Dec 28, 2015].
[12] Texas Instruments. (2002, May.). “LOG101 Logarithmic and Log Ratio Amplifier” [Online]. Available: http://www.ti.com/lit/ds/symlink/log101.pdf [Dec 28, 2015].
[13] M. Burke, Handbook of machine vision engineering. London: Chapman & Hall, 1996.
[14] Amazon.com, "Amazon.com - Achim Home Furnishings Luna 2-Inch Vinyl Blind, 23 by 64-Inch, Mahogany - Window Treatment Vertical Blinds", 2016. [Online]. Available: http://www.amazon.com/Achim-Home-Furnishings-64-Inch-Mahogany/dp/B00FJDVDU4. [Accessed: 02- Jan- 2016].
[15] Support.robotis.com, "AX-12/AX-12+/AX-12A", 2016. [Online]. Available: http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actua tor.htm. [Accessed: 09- Dec- 2016].
[16] Amazon Web Services, Inc., 'AWS | Amazon DynamoDB - NoSQL Cloud Database Service', 2015. [Online]. Available: https://aws.amazon.com/dynamodb/?hp=tile. [Accessed: 09- Dec-2015].
[17] Developer.nest.com, "Nest Developers", 2015. [Online]. Available: https://developer.nest.com/. [Accessed: 30- Dec- 2015].
[18] Learn.adafruit.com, 'Overview | Adafruit HUZZAH ESP8266 breakout | Adafruit Learning System', 2015. [Online]. Available: https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout/overview. [Accessed: 09- Dec- 2015].
[19] Developer.android.com, "Getting Started | Android Developers", 2016. [Online]. Available: http://developer.android.com/training/index.html. [Accessed: 25- Jan- 2016].
[20] A. Overview, "Android Studio Overview | Android Developers", Developer.android.com, 2016. [Online]. Available: http://developer.android.com/tools/studio/index.html. [Accessed: 25- Jan- 2016].
[21] Amazon Web Services, Inc., 'AWS Lambda - Serverless Compute', 2016. [Online]. Available: https://aws.amazon.com/lambda. [Accessed: 02-May-2016].