University of Tennessee, Knoxville
Trace: Tennessee Research and Creative Exchange

University of Tennessee Honors Thesis Projects | University of Tennessee Honors Program

5-2016

Motion-Activated Infrared Security System

Christopher H. Skelton
University of Tennessee, Knoxville, cskelton@utk.edu

Daniel Graves
University of Tennessee, Knoxville, xzs919@utk.edu

Kenton Culbertson
University of Tennessee, Knoxville, kentonculbertson@gmail.com

Joseph B. Burke
University of Tennessee, Knoxville, whysojoe@gmail.com

Edward Hockaday
University of Tennessee, Knoxville, edhockvols@me.com

Follow this and additional works at: http://trace.tennessee.edu/utk_chanhonoproj

 Part of the Electrical and Electronics Commons

Recommended Citation

Skelton, Christopher H.; Graves, Daniel; Culbertson, Kenton; Burke, Joseph B.; and Hockaday, Edward, "Motion-Activated Infrared Security System" (2016). *University of Tennessee Honors Thesis Projects.*
http://trace.tennessee.edu/utk_chanhonoproj/2137

This Dissertation/Thesis is brought to you for free and open access by the University of Tennessee Honors Program at Trace: Tennessee Research and Creative Exchange. It has been accepted for inclusion in University of Tennessee Honors Thesis Projects by an authorized administrator of Trace: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

Motion-Activated Infrared Security System

April 26, 2016

Team 14

Heath Skelton

Kenton Culbertson

Joseph Burke

Daniel Graves

Edward Hockaday

Customer: Dr. Shelley McCoy

Table of Contents

Executive Summary

The goal of our project was to design and build a security system which takes still photos when motion is detected. These photos would be sent to the user where they could view the captured image as well as previously captured images. The camera was also to be usable at night, which necessitated the use of an infrared camera. An additional requirement was to supply backup power in the case of an outage through the form of an uninterruptible power supply. The device was to be mountable outdoors, which meant the enclosure needed to be waterproof to withstand rain. The price constraint was $300.

We believe we have met these goals. We selected components that fit the requirements specification and that stayed well under budget at about $145. We have implemented a security system capable of detecting motion that can operate both day and night. A battery is used to act as the uninterruptible power supply, satisfying the continuous operation requirement. An infrared camera and infrared LEDs are used to enable night-time security photos. Using bash scripts and Google Apps Scripts, we are able to notify the user whenever a still photo has been captured. Additionally, all captured images are stored on cloud storage, so historical photos can also be viewed.

This project was not without challenges, however. Many design modifications have been made throughout the semester to arrive at the current implementation. One of these first such decisions was to use only a single infrared camera instead of a dual-camera system. Another challenge was the glare from the infrared lights inside the enclosure. For this reason, we needed to house the infrared LEDs outside of the waterproof enclosure. There are many other challenges and changes made, which will be discussed in greater detail in later sections.

Requirements

Hardware Requirements

1.1 Enclosure

1.1.1 The device must be contained in one single enclosure

1.1.2 The enclosure should be the size of a Raspberry Pi

1.1.3 The enclosure must be weather-proof

1.2 Optical

1.2.1 The device must contain a single camera

1.2.2 The camera must have infrared capabilities

1.2.3 The device must be capable of taking still photos

1.3 Power

1.3.1 The device must be constantly powered

1.3.2 The device must be powered with 120V AC power

1.3.3 The device must contain a battery system for backup power

1.4 Connectivity

1.4.1 The device must be internet connected

1.4.2 The device must use WiFi for internet communication

Software Requirements

2.1 Data Storage and Access

2.1.1 All images must be stored with a cloud storage system

2.1.2 Images must be accessible remotely through the internet

2.1.3 All images must be viewable from the user's phone

2.1.4 User must receive a phone notification when an image is taken

2.2 Motion Detection

2.2.1 Camera must have motion sensing capabilities

2.2.2 Images must be taken whenever motion is sensed

Budget Requirements

3.1 Device must cost less than $300

Change Log

Throughout the process, we changed many aspects of our design. Some of these changes were major and some were minor tweaks to ensure smooth operation. They are listed below along with the reasoning behind the changes, in chronological order.

1. Performing motion detection in software.

   One of the first major decisions we made was how to detect motion. Ultrasonic sensors were initially considered, but we decided that a software algorithm could be more precise and less buggy. Physical sensors also cannot detect the full field of view of a camera, providing our system with greater coverage.

2. Changing from a dual-camera system to a single infrared camera setup.

   Initially, we had intended to use both a standard webcam and an infrared camera. The infrared camera would be used for all motion detection, and would capture images at night. The webcam would be used when ample light was available. When we received the infrared camera, we found it to be satisfactory in daylight conditions. Additionally,

the enclosure is quite tight with space and fitting both cameras inside the enclosure would have been much more difficult.

3. Using email notification instead of building an application.

   We originally planned to build an Android application, but upon our customer's suggestion, we decided to implement the notification system simply using email. This way, she will be able to access the images from a phone or computer. It also does not require her to have a dedicated application on her phone.

4. Using an enclosure larger than a Raspberry Pi

   This change was necessary in order to satisfy the requirement of an uninterruptible power supply. Since we are using a Raspberry Pi as our microcontroller, the enclosure would have to be larger to fit a battery in. We found an alternative in a dummy security camera shell. This is an empty security camera, which a cable-way to route power in, that can be mounted externally due to its waterproofing. We communicated this change to our customer, which she approved.

5. Housing the infrared LEDs externally instead of in the enclosure.

   This change was one out of necessity. The infrared light reflects off the transparent lens in the enclosure back into the camera lens, causing significant glare. We are mounting the infrared LEDs under the camera in a separate enclosure in order to get rid of this glare and provide better night-time images.

Documentation of the Design Process

We began the project with a few design ideas that were necessary and unchangeable for the project. These mainly stem from the requirements. These included the need for a microcontroller, an infrared camera, infrared LEDs, an uninterruptible power supply, a waterproof enclosure, and wireless connectivity. From here, we had freedom to determine which components to buy and how to assemble them to reach a desirable end goal. After components had been sourced and acquired, the software design could begin. We also had many options here for how to proceed. Many aspects of the design depended on software and could be solved in many ways. The basic software components come down to the choice of operating system, motion detection, image capture, image sending, and image storage. We will go into each of these hardware and software components in depth and explain the various options we considered, why we chose the design we chose for each portion, and how we were able to verify its outcome.

Microcontroller

This was one of the most critical choices in the whole project, and was the first choice we had to make as a team, since the project could not proceed without a chosen microcontroller. The options considered were the Raspberry Pi, the Arduino Uno, and the BeagleBone Black. We needed something compact that could be connected to a camera and to the internet. It needed to have processing fast enough to perform adequate image processing. Each of these boards would have worked for this project, but we found the Raspberry Pi to be the best choice. It has tons of resources online to help with projects, is inexpensive, and has hardware designed specifically to interface with it. Additionally, operating systems with a

graphical user interface can be installed to make programming and testing easier. One additional advantage of the Raspberry Pi is that we were able to obtain a Raspberry Pi Model 3, which has built-in WiFi communications, something none of the other boards support. This allows us to not rely on a questionable USB WiFi adapter which may not have sufficient range for our needs. The Raspberry Pi also has a MicroSD card slot, which allowed us to provide about 14GB of storage for images. The Raspberry Pi is powered via a MicroUSB port, so it could easily be interfaced with a USB battery backup system to act as a UPS. The Raspberry Pi Model 3 perfectly served its purpose in this project and we would certainly recommend it to any future projects, due to its ease of use and variety of applications.

Infrared Camera

This decision and all other decisions could be made with the Raspberry Pi in mind, as it was our first design decision. The infrared camera was a relatively simple decisions since Raspberry Pi makes an infrared camera specifically designed to interface with the board's Camera Serial Interface port. The Raspberry Pi NoIR camera is a 5 Megapixel camera with no infrared filter - allowing it to be used in conjunction with infrared lighting. Due to the lack of an infrared filter, we were worried that daytime photos would be of poor quality. During initial testing, we were satisfied with the appearance. There is slight discoloration, but image quality is still high. One advantage of using this camera over a USB webcam is the use of the ribbon cable. This allows extremely fast data transmission, and makes the camera's positioning very flexible. Since we were dealing with a very tight space constraint inside the enclosure, this proved to be very beneficial. One other option we considered was purchasing a USB webcam and removing the infrared filter. Since webcams are generally of a much larger form factor than

the NoIR camera board, however, this idea did not pan out. Modifying the camera is also much more risky than using a camera designed for the specific purpose of being using at night, which leaves a greater room for error. Additionally, the Raspberry Pi NoIR camera is autofocusing, which is very important for this application. A focus shift occurs when a camera switches from high to low light with the use of infrared lighting. Without autofocus, the nightvision photos would always be blurry and useless. Along with using an infrared camera comes the need of infrared LEDs, discussed in the next section.

Infrared LEDs

Infrared cameras can only "see" at night if the area is illuminated by infrared light. This can be achieved with the use of a bank of infrared LEDs. While other options are available to generate light, LEDs are among the most efficient sources of light, they are inexpensive, and they are very long lasting. Once we knew we would need these as a "light" source (in quotes because the light is not visible to the naked eye), we needed to know how many we would need to illuminate our customer's desired area. Since she requested the camera to be used to light up her driveway area, the light would need to be sufficient to illuminate the entire field of view. Without a fully illuminated field of view, motion could not be detected in far-reaching areas, in addition to not being able to fully see the area at night. For this reason, we decided to go big rather than to be conservative, using the better safe than sorry philosophy. While a bank of 48 LEDs may have been overkill, we are confident that the full area will be illuminated. Most CCTV security systems mount infrared LEDs around the camera lens, usually 15 or less. One unforeseen problem we ran into here was the infrared light glare. We were forced to mount to infrared LED bank outside of the enclosure in order to prevent glare. This is not a huge issue,

but does slightly increase the size of the device. We will mount the lights under the camera to shield it from the elements. During testing we found the lighting to be far beyond sufficient and were satisfied with our product choice.

Uninterruptible Power Supply

This was one of the more debated design decisions throughout our process. Many UPS systems rely on physical batteries, like AA or AAA, or large capacitor banks, but we wanted something more robust. The Raspberry Pi is powered through a 5V USB connection, so a typical USB battery bank, like the ones used to charge a phone on the go, would be a good solution. There is a large potential problem in choosing one, however. The battery must be able to simultaneously charge and discharge, and there must be no interruption of service when power goes out. We scoured internet reviews and forums trying to find a battery that met these specifications and found only a couple. One was a product designed specifically for the Raspberry Pi, but at a cost of $60, it was far too expensive. We found an older battery by TeckNet that could allegedly simultaneously charge and discharge and that had been used in similar applications. We ordered the battery and were surprised when we received an entirely different battery. The product we provided a link to in our order form had changed in the few days between when we submitted the order and when it was shipped. This meant that we got a newer, better product, but we did not know whether it retained the much needed simultaneous charging and discharging capabilities. We nervously tested the device and were relieved to find out that it would work for our product. The device is very powerful, at 10,000mAh, and will last several hours continuously before requiring charge. External power is

routed into the battery, then out into the Raspberry Pi, so power is always being supplied to the microcontroller.

Waterproof Enclosure

This was one design decision where it was necessary to deviate from the requirements specification. In order to fit both a microcontroller and backup battery inside an enclosure, it would have to be bigger than a Raspberry Pi board. We told our customer of the design change, which she approved. We found a dummy security camera shell -- essentially an empty security camera which could house other components inside. We removed the fake camera lens and fake LEDs in order to install our own parts. The shell is waterproof, and has external cable routing. We enlarged the cable routing hole to fit a MicroUSB cable through. The enclosure's interior is just large enough to fit the battery and microcontroller. We had to make some modifications to the interior of the shell in order to fit our parts, but in the end it is adequate. We have test to ensure that the camera's view is not blocked by any part of the camera shell, and it performed perfectly. The camera can be mounted to a wall and adjusted to face any way it is needed. Power will be supplied into the housing from a cable which will connect to external power outlets. We have purchased an adapter which allows us to tap off light sockets to provide 120V AC power.

Wireless Connectivity

This is a very critical aspect of the project, which helped us to determine which microcontroller we would use. Without wireless internet, the product is essentially useless, as it would be unable to relay images to the customer. Once the device is on a network, we can remote into to perform any setup or maintenance needed. During the initial setup, we will

connect the device to our customer's wireless network and then work remotely to set up the necessary image masks. For wireless connection, we essentially had two choices: a USB WiFi adapter or an onboard wireless chip. We believe the second option to be a better solution for several reasons. First, an onboard chip will be well documented and is guaranteed to interface well with the device. Its range and compatibility will also greater than a cheap USB adapter. Additionally, it will not require any additional space or power draw as it already on the Raspberry Pi board. We initially purchased a wireless adapter before we had obtained a Raspberry Pi Model 3. In testing, we found the Raspberry Pi Model 3 with onboard WiFi communications performed much better than a Raspberry Pi Model 2 with a USB WiFi adapter.

Operating System

The Raspberry Pi Model 3 is running Raspbian. We chose this ARM-based operating system because it is developed by the creators of the Raspberry Pi and integrates well with all of our hardware components. It has built in WiFi connectivity options within the graphical user interface that allows the user to easily change the WiFi network without having to use the command line. We feel that using an operating system with a graphical user interface is a huge advantage to the average user that does not know how to use Linux and command line tools. Raspbian is based on Debian Jessie and comes standard with Python, Java, and many other useful built-in resources. Since many of our scripts are written in Python, this makes our device much more seamless for the user.

Motion Detection

Due to widespread usage and readily accessible documentation, we decided to use OpenCV for our motion detection algorithms. Our alternatives were limited due to the Pi's

computational and CPU constraints as well as our lack of access to proprietary material. One such alternative was to write our own image processing and computer vision algorithms. Benefits of this approach would have included better control over optimization (for example, using a compiled language instead of a scripted one) as well as finer integration with the GPU, where applicable. However, the additional time required for programming these methods and algorithms ourselves was unacceptable for the timeframe of our project and unnecessary for the ultimate goal of our project. Since our goal was one of motion detection and not high-quality streaming, measures such as frames-per-second (FPS) were flexible to provide ample time for computation to take place. Some tuning parameters, such as the number of consecutive motion frames required for an event to be considered, were tweaked to account for this FPS flexibility. The algorithm uses absolute image difference to determine if a motion event has occurred. The difference images consist of the current frame and an average of recent frames. Once a motion frame is detected, the average frame is tested against subsequent frames to determine that motion is still occurring. After a configuration-set number of consecutive motion frames has occurred, an image of the current frame is sent to the image dump file where it will be emailed to the owner. This configuration number is set on-site to decrease motion miss rate to a reasonable level while avoiding unnecessary spam from motion glitches. To avoid image spam from a single motion event occurring over several minutes (or hours), another configuration parameter is defined to avoid sending multiple images within a specified timeframe.

Image Capture

The Raspberry Pi provides an official camera capture interface in Python called *python-picamera*. This is a library that can be imported in a script and used to access camera frames in a convenient manner. Although our original plans called for the use of C++ and a third-party library, we decided that it was best to use the official interface for its widespread usage and the availability of its documentation. Additionally, we decided that the potential benefit from using C++ (a compiled language) over Python (a scripting language) was irrelevant as frame rate and fine computational optimization is not required for our algorithms. Using the motion detection algorithms, a motion frame is chosen as the last frame in a sequence of consecutive motion that differs from an average frame. This ensures that the frame sent contains a subject of interest and that the frame sent is one after the subject has exited the view of the camera.

Image Storage and Sending

Since our resources are limited on the Raspberry Pi to 16GB of flash storage, we decided to store our images remotely via Google Drive. When motion is detected in the customer's driveway, a Python script takes a still image and stores it in a directory on the Raspberry Pi. There is a bash script running continously that searches for files with the .jpg extension. The bash script then separately emails each .jpg file to an email account setup specifically for this purpose. The image file is then deleted from the Raspberry Pi to conserve storage. Using Google's built in Apps Scripts, we automatically add this file to Google Drive and sort it by date and time. The user is then notified via email that there has been motion detected, and a screenshot of the detected motion is attached to the email. The user also has access to the Google Drive folder that contains all previously captured images.

Each of these design decisions had implications for other choices, and we needed to ensure that each change made would not harm other components of the project. For this reason, we approached the design and testing phases in a piece by piece way. We began with hardware assembly and verified that all of the hardware communicated with each other properly and would properly function. This phase included testing the wireless capabilities of the Raspberry Pi Model 3 with onboard WiFi versus the Raspberry Pi Model 2 with a USB WiFi adapter. We were then able to settle on the Raspberry Pi Model 3 and proceed with writing code in order to implement the software aspects of the design. We carefully considered that our design met our agreed upon and modified requirements at each step of the way. We have sufficiently satisfied each requirement, and included additional functionality. Testing for some phases was more laborious and time intensive than others. The hardware testing was by far the simplest and most pain-free of the design stages. Our hardware components simply worked correctly out of the box and required little to no additional modifications. The mounting of the components inside the enclosure was the most difficult and frustrating portion of this stage, since the space is tight and some modifications were made to the case to fit our pieces. After the hardware was assembled and verified to work, we could move on to fully focusing on the software aspects.

We began the software implementation by writing scripts that would automate the necessary processes. First, a Google Apps Script was written to sort captured images in a Google Drive cloud storage solution, sorted by date and time. Each day has its own separate folder, and the images are sorted chronologically within the folder. After this sorting procedure

was verified by using dummy test images, we moved on to the email and notification system. This is accomplished with Python and bash scripts. These were also tested with dummy images and email addresses to verify that it functioned properly before tackling the biggest task of motion detection. Since all the other tasks had been tested and confirmed, we could focus entirely on creating an accurate motion detection program that would not spam our customer with unnecessary images. This is accomplished by setting a sleep timer to not send images constantly, and by applying image masks over unwanted detection areas. The image mask uses an AND operator to prevent the camera from recognizing motion in a portion of the field of view, such as a distant road or tree tops moving in the wind.

Since we divided the work responsibility among team members in a way that suited each person's strong suits, we were able to work on the project efficiently and were able to gain valuable knowledge in each other related aspects of the design. Through testing and team member communication, we were able to ensure that all portions of the design properly interface with each other. It was important to us to approach the project piece by piece instead of diving into all of it at once. This enabled us to verify consistency and compatibility at each step of the way, rather than doing everything separately and hoping the interfacing performs correctly.

The final design can be decomposed into the hardware interface and the software interface. Beginning with the hardware interface, the device flow is as follows: External power is connected via MicroUSB cable to the USB battery. A USB cable connects this battery to the Raspberry Pi's MicroUSB port. The Raspberry Pi NoIR camera is connected to the Raspberry Pi via the Camera Serial Interface port. These components are all housed inside the waterproof

dummy security camera shell. Mounted underneath the enclosure is the infrared LED bank, which contain a photocell so they are only drawing power at night. The camera lens is mounted facing outward through a glass window on the security camera shell. Power will be connected into the system through external wall sockets. The software interface is implemented as follows: the Raspberry Pi boots in Raspbian, where scripts automatically start to begin looking for new .jpg files. The motion detection software constantly scans for motion and takes still images when motion has been detected. These images are then found by the Python script, which sends an email with the file attached to an email account specifically created for this purpose. The file is then deleted from the Raspberry Pi's storage to save space on the device. A Google Apps Script scans for emails in the setup account, and uploads the attached files to Google Drive, where they are sorted by date and time. An email is then sent to the user, notifying her that motion has been detected and her attention to the matter is needed. The customer can view the captured image, and view previously captured images if she wants. The scripts have a sleep timer embedded in them so that notifications are not continually sent. Additionally, the motion detection software uses size difference techniques to determine if the motion was "large" enough to warrant a notification, reducing the annoyance of something like a squirrel running through the yard.

Lessons Learned

Throughout the process of designing our security system, we learned many lessons about the design process and what it takes to be a good engineer. We experienced both failures and successes throughout the entire semester, but ended up with a good final product that we

are proud to present our customer. Some aspects of a project like this that we had never encountered before include customer communication, agreeing upon and modifying a requirements specification, and tackling a project of this scope from start to finish.

One of the first issues we ran into was how to power our security system. The requirements made it clear that the security system needed to be mounted on the outside of a house which would give us limited power source options. We had to consult with our customer about options for this. We came to the conclusion that the best option would be to plug into a traditional outlet on the outside of the house. We are displeased that our system won't be able to provide a cleaner looks since wires will be exposed, but since external power is needed it is the only available solution. Through this we learned that sometimes you have to sacrifice aesthetics for functionality, especially when you are on a strict budget. We also had a conversation about the importance of considering early on the small things, like security systems, in a large scale design project, such as the power layout for a house.

We also faced the decision of choosing a microcontroller. We were pretty certain from the beginning that we wanted to use a Raspberry Pi. At the start of the design process the available model was the Raspberry Pi Model 2. We were aware that the Raspberry Pi Model 3 was going to be available shortly which would include built in WiFi. It was tempting to hold off and wait to try and order a Raspberry Pi Model 3 in order to save money and use the final collection of parts for the whole design process. We made the decision to order the Raspberry Pi Model 2 and go ahead with the design process until the Raspberry Pi Model 3 became available. This, in retrospect, was the correct decision because it allowed us to proceed and test other aspects of our design. Once we were able to obtain a Raspberry Pi Model 3 it took a slight

change in programming to accommodate the built in WiFi and to change the physical design to accommodate the removal of the WiFi dongle. It was certainly worth the added expense of purchasing the second microcontroller to be able to proceed in the design process at an earlier time and to still have a more powerful and connectible device.

One very valuable lesson learned was to test the product in phases. By verifying step-by-step that each portion performs and interfaces correctly, we lose the risk of catastrophic failure at the end when something does not work properly. It also enabled us to pinpoint failures and fix them before they ballooned and affected other portions of the design. One regret we had in this process is that we were unable to test our components before ordering them. This resulted in us purchasing a USB webcam which would go unused, due to space constraints and lack of necessity.

We were able to experience how to divide up project portions between team members to fit their expertise. For example, Daniel has a background in image processing, so he was tasked with implementing the motion detection software. This allowed each member to focus on portions that they understood and had either experience in or the capability to tackle. We found this approach to work well, as well have a variety of interests and experience. As a result, we were each able to gain valuable experience related to our fields in addition to learning how the other portions work during the testing phases.

One final lesson we learned was how to communicate between team members and the customer. We experienced a few times when team members would go on a rabbit trail of their own ideas without consulting another team member. This occurrence ended up wasting time and the decreasing the overall productivity of the team. As the design process progressed, we

learned to better discuss and agree on how to go about meeting the requirements of the customer in our design. It saved us time and gave us more manpower when we are all openly communicating and agreed on the direction of the project.

One thing that all team members agreed upon is that we would have liked to have done a 2-semester project. We feel that we could have gotten much more done and undertaken a more complicated project, and learned more along the way. Writing the papers and doing the presentations in ECE 401 was valuable experience, but we feel that it would have been more valuable if those writing and presenting tasks were related to a project which we were actually attempting. As a suggestion for future projects, we think the course would be more fulfilling if all projects were 2-semester in order to provide more time to attempt larger and more interesting projects.

Team Member Contributions

Heath Skelton

Responsibilities: Team Leader, Report Preparation, Hardware Component Decisions

Contributions: Sourced all hardware components, prepared purchase form, edited all reports, MBO report preparation and meetings, set up team meetings, set up Raspberry Pi and wireless network for testing.

Daniel Graves

Responsibilites: Image Processing, Motion Detection, Image Capture

Contributions: Applied and tested image processing and computer vision algorithms, installed libraries for computer vision and image capture, assisted with general Linux knowledge and command line interfaces.

Joseph Burke

Responsibilities: Customer Communication, Bash Scripting, Python Scripting

Contributions: Python and Bash Scripting, Networking, and communication with customer.

Kenton Culbertson

Responsibilites: Google Apps Scripting, File Sorting

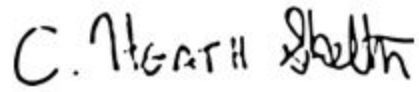
Contributions: Wrote scripts that retrieved, sorted, and automatically backed up all motion detected images to the Google Drive storage.

Edward Hockaday

Responsibilities: Presentation Preparation, Hardware Assembly

Contributions: Battery knowledge, presentation editing, arrangement of components inside casing.

Signatures of Team Members and Customer



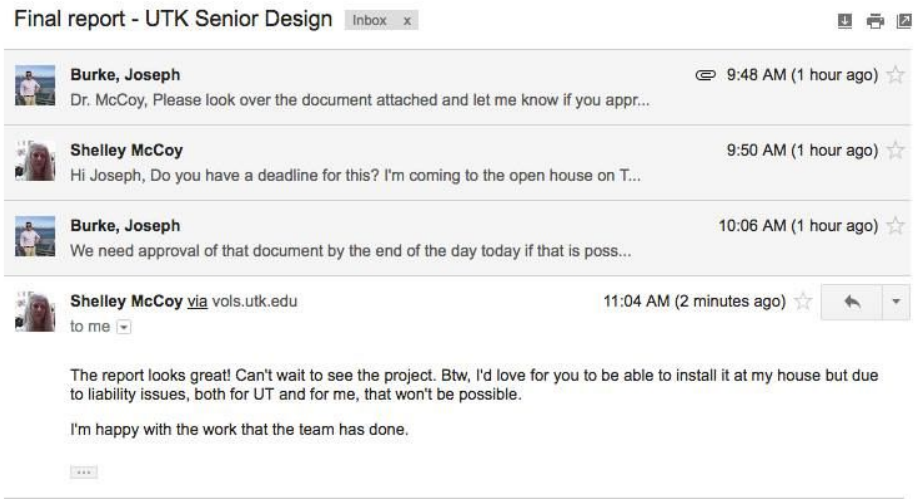
C. Heath Skelton

Joseph Burke

Kenton Culbertson

Daniel Graves

Edward Hockaday



Customer: *Dr. Shelley McCoy*

Dissenting Statements: None