Smart Mirror

A smart home solution for increased productivity during those busy mornings

COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Group: K

Senior Design I Project Documentation

04 / 28 / 2016

Members

Justin Gentry Michael Trivelli Hector Zacarias

Table of Contents

1. Executive Summary

Technological integration into homes, so called “Smart Home Technology” is becoming increasingly popular in our society today. The primary benefit of smart home technology is to simplify our day to day lives in any way possible. Some benefits include saving time, relieving stress, or even saving money. These benefits can be accomplished in a number of ways including automation of tasks, improved access to media and information, as well as varying degrees of personal comfort.

A Smart Mirror is meant as a smart home technology to allow convenient access to media and information that people might be interested in on a daily basis. The mirror will display customized information to the user to allow users to get some basic daily information at a glance. This will save users time in their busy morning routine which will, as a result, relieve stress and help people get out the door on time in the morning. The mirror will display information users may access in the morning via their phone or other technology in a way that is readily available as soon as they walk into the bathroom. This will allow the user to plan for their day while simultaneously completing their daily routine in front of the mirror.

Some information that will be provided to users includes; the time and date, the weather for the day, a daily schedule, a to-do list, the users preferred news source, traffic delays to expect on the way to work, and so much more. The Smart Mirror will improve people’s lives by integrating information they consume into a location that they visit multiple times daily in the most convenient format possible. The mirror will be hands free, turning itself on and off automatically when someone enters and exits the room. Due to the fact that mirrors are generally used by multiple people, facial recognition will be implemented and the mirror will display custom information per user, with a basic guest account for faces not recognized by the mirror. User interaction will be achieved through voice commands, keeping the interface hands free for convenience and ease of use.

One of the driving design principles behind this project is to create a platform for open source development. This will allow an abundance of application to be developed by multiple people, expanding this project beyond what one team can easily achieve. Due to this, this project will be implemented on a platform such that any user can design and implement their own widgets, allowing ultimate customizability to tech savvy users. Though it offers unlimited capability to technologically intelligent users it will have a simple basic interface with the most critical functions implemented in a way that is beneficial to any user. This will allow the mirror to appeal to casual users while also offering advanced users any potential features they can imagine and create. Customizability is an important aspect of anything that goes into people’s homes as these type of items tend to cost a lot of money. With the level of flexibility provided by the mirror, users will be able to set the mirror up exactly as they personally see fit.

2. Project Description

2.1 Project Motivation & Goals

As technology advances, we continue to find more and more uses for it that would previously be inconceivable. Originally, technology was primarily useful for performing tasks humans struggle with, but today it is used in even the most mundane tasks in an attempt to simplify our lives. With the technological revolution, we have been able to save time in a number of ways; however, as media consumption has increased, we also lose time. Due to this, saving time in our daily routines is always helpful. One way technology has been implemented to save time is by integrating computers into numerous elements in our home, thus creating “Smart Home” devices. The “Smart Mirror” project is based upon this concept.

The Smart Mirror will merge technology with a mirror to provide users information while they use their mirror. The primary motivation behind the smart mirror is to improve quality of life. Providing information to users in the most convenient way possible is a driving motivation behind the majority of technological development for smartphones and tablets. The smart mirror will provide convenient information to users on their mirror every day. Allowing the user to multitask by consuming media while preparing for the day will save people time nationwide. The goal of the mirror is to provide people with information they may require in the morning while getting ready for the day or at night before going to bed. This will save users time every day and help to ensure they are aware of important details for their day. A user will be able to check their calendar for any upcoming events, peek at the weather forecast, and not to mention, consult the mirror for traditional personal appearance adjustments.

Motivation for this project stems from multiple sources. In the Iron Man films, the main character utilizes holographic displays around the home to perform a number of activities. A couple years back, Corning released a video about their product called Glass which is intended to allow a smart surface anywhere in the home. While these examples, and a multitude of others, are well beyond the scope of this mirror, their realization also seems to be well into the future. One benefit to the smart mirror is that, while it does not provide the advanced capabilities of these examples, it is readily feasible. Another driving factor in this project is the fact that smart home technology has been developed for many parts of the home but smart mirrors are lacking. While there are plenty of tinkerers projects posted around the web, no fully realized implementation has been marketed to users thus far.

2.2 Objectives

A staple furniture piece found in every bedroom and bathroom, the mirror has provided a means for effective personal grooming for thousands of years. Our team has brought it into the 21st century with Smart Mirror. While the user is preparing themselves for the day, they will be able to glance at their mirror and instantly retrieve important bits of information such as the current time, date, and weather conditions. These simple bits of information are commonly sought after in the morning while getting ready for the day or at night before going to bed. Our Smart Mirror ensures that you remain at the forefront of the emerging smart-home revolution. You will take advantage of every opportunity to look at yourself in the mirror in order to be filled with weather information and today's date. With a brief look at the mirror, users will be able to check the current time, peek at the weather forecast, and not to mention, consult the mirror for traditional personal appearance adjustments. While the user is preparing themselves for the day, they will be able to talk to their smart mirror to see what reminders they had set for the up and coming week, what the weather will be like so they can dress accordingly, and set any calendar events they need to remember all while being able to listen to their favorite songs via Pandora Internet Radio.

The Smart Mirror will be a smart home implementation in the bathroom which is a room currently lacking technological innovations. Smart Home technology has been integrated to a number of rooms and interfaces throughout the home however the bathroom has been left mostly untouched. Almost every person spends some portion of their time daily in the bathroom. As a result, there is ample opportunity to present users with information that could improve their daily life.

The mirror will be able to present personalized information to users every morning as they prepare for their day. Ideally it will save users time by displaying information they would likely check in the morning. This will include information such as weather, daily schedule, news, and also the time so the user can keep on schedule. The information will be provided on the mirror in an unobtrusive manner, leaving the majority of the prime mirror real estate unaltered. This will allow the user to easily absorb the displayed information while going about their normal routine.

One of the prime objectives of the mirror is to be as user friendly as possible and to provide different options and customizability to different users. Due to this, all applications will be customizable by the user to present information in the way they deem best. Also, all applications will be able to be deactivated if the user deems them useless for their day to day life. This will allow the users to use the mirror in a way that fits their own unique circumstances without being bothered by features that they will not use. If the user ever changes their mind about a feature they can simply enable it and continue use as normal with the feature included.

Since this project is designed to be as unobtrusive and as user friendly as possible, all control of the mirror will be accomplished via voice commands. This has been decided due to the fact that voice commands will be the least intrusive on a user's daily routine. As you go about your morning or evening in the bathroom, you tend to utilize your hands a lot, thus, touch controls or gesture controls would simply inconvenience users. Due to this observation, voice controls seem the best fit to allow user interaction with the mirror. One issue with voice controls we will attempt to avoid is the necessity for a strict list of commands. The voice controls will be implemented in a way such that interacting with the mirror seems like talking to an intelligent agent rather than rattling off a list of scripted commands.

The Smart Mirror is designed to utilize a persistent internet connection. The mirror will fetch the weather conditions and news headlines, for instance, periodically. The time and date are set according to the Raspberry Pi's internal clock which is synced upon connecting to the internet. The voice commands do not rely upon the internet connection. While the Smart Mirror relies on a constant internet connection to fully function, a loss of internet connectivity is not fatal to the mirror's operation. In the event that the internet connection is lost during the moment when the internet-dependent data are fetched, such information on the mirror will be hidden to signify that the internet connection has been lost. The time and date do not require a persistent internet connection after the initial synchronization has occurred.

Lastly, the mirror will need to include safety measures to ensure use in a bathroom does not cause irreversible damage to the mirror itself. The mirror will monitor the temperature and humidity of the internals of the mirror housing and, if these levels hit certain thresholds, the mirror will act to avoid damage. Under unsafe situations, the mirror will either attempt to bring the temperature or humidity down by turning on an air movement system such as a pair of fans or, if the temperature and humidity levels run too high, the mirror will simply shut down.

2.3 Project Specifications

The mirror will be developed with specifications based upon devices people use every day; tablets, smartphones, and PCs. The software implementation specifically will be implemented in a way where specifications for future hardware implementations can be as flexible as possible. However, during the early development phase, strict specifications will be set to ensure all goals and objectives mentioned previously are met.

2.3.1 PC Specifications

For the smart mirror, a computer will be required to process and display all information to the user. The software for the mirror will be implemented via Universal Windows Platform programs which means they will function on any computer running Windows 10 or Windows 10 IoT. However, during development, the mirror will utilize a Raspberry Pi 2 Model B as the primary computer. All sensor components will be run through an MCU and fed into the Raspberry Pi. The hardware specifications of the Raspberry Pi 2 Model B are shown in Table 2.3.1.1.

Raspberry Pi 2 Model B	
CPU	Broadcom Quad-Core ARM7 900MHz
Memory	1GB SDRAM
Storage	8GB microSD
Power Supply	5V microUSB
Wi-Fi Module	802.11b/g/n
Video	HDMI 1.4
Audio	3.5mm Audio Port
USB	4x USB 2.0
GPIO	40 pin extended GPIO

Table 2.3.1.1 – PC Hardware Specifications

2.3.2 Video and Audio Specifications

For display purposes, a thirty-two inch television will be utilized. The constraints on the television are flexible, requiring simply a single HDMI input to display the information presented by the Raspberry Pi. For audio implementation, there will be three primary options: the first will be speakers via the television, the second would be internal speakers, while the third would be external speakers. The first two options would be housed within the mirror itself. Speakers in the television would receive their signal from the HDMI which provides video. Internal speakers not connected directly to the television can receive audio via the 3.5mm audio jack on the Raspberry Pi. Finally, external speakers may be utilized but would require a third party Bluetooth dongle connected to on the Raspberry Pi’s USB slots.

2.4 Required Features

2.4.1 Required Features

The features outlined below were selected to highlight the project’s design and objectives. They are vital to the underlying functionality and operation of the mirror. These features implement objectives and specifications related to both hardware and software.

- The smart mirror shall be powered by a small computer housed within the mirror itself.

- The smart mirror is shall connected to a thirty two inch display, measured diagonally set in a portrait orientation. This display shall be mounted behind a one way mirror.
- The user shall be able to interact with the smart mirror via voice commands thus the mirror shall implement a form of voice recognition software which will act as the primary means of interaction.
- While it is running, the smart mirror shall display persistent applications at all times. The options shall include a clock, calendar, news feed, weather, and music player.
- The mirror shall allow the user to customize which of the persistent applications are present when the mirror is on.
- The smart mirror shall have the option to provide audio output via speakers housed within the frame or external speakers to provide audio output.
- The smart mirror shall utilize a temperature sensor and a humidity sensor, to prevent any potential damage to the mirror under high temperature and humidity conditions.
- The smart mirror shall utilize a light sensor and a motion detection system to implement an auto on/off feature. The mirror will turn on at motion recognition and will turn off if no motion occurs for a specified time or if the light in the room is turned off.

2.4.2 Additional Features

The features included in this section detail objectives and specifications that are not vital to the overall design. These features will be considered extras and will be implemented only once all required features have been met. These are features that could be nice to have but would not hamper the entire design if they are lacking.

- The mirror shall implement an LED lighting system on the housing which will be controllable via voice commands.
- The mirror shall implement a user account feature so different information can be presented to different users. The accounts would be accessible via simple voice commands to change users thus secure information will not be displayed.
- The mirror shall implement Microsoft’s smart assistant, Cortana, to aid users in daily tasks such as creating reminders or searching locations such as restaurants or entertainment.

3. Initial Research & Design

3.1 Similar Projects & Products

3.1.1 Projects

The idea of a Smart Mirror has been floating around on the internet for a couple years now and there are a number of well-developed projects available across the internet. Some of these projects have inspired features of our design and some have raised red flags allowing us to identify potential features that could improve the practicality and functionality of the mirror. Listed in this section are three of the more popular, polished projects that we found during our research. All projects selected for this section were required to be done as open source software by individuals who are not attempting to market their software.

Smart Mirror, the fairest of them all, started by developer Evan Cohen is a project currently under development by the open source community on GitHub. It is being developed using JavaScript and is available on a number of platforms. The project currently has a number of features implemented including voice commands to control the mirror. This project also includes the ability to control external lighting via voice commands. This mirror has some of the functionality we desire but it is not flushed out to the extent we are hoping for. While some lighting control will be implemented directly on the mirror itself, the functionality to control external lighting in this project is an option that will not be implemented on this project. This decision is due to the fact that controlling systems external to the mirror are beyond what has been deemed useful or important for the scope and goals of this project.

Magic Mirror is a project created by Michael Teeuw which was hosted on Raspberry Pi’s website. The project was created using JavaScript and, according to the GitHub, runs as a php script on a web server with little external dependency. This mirror implements some basic functionality but it does not have any user interaction capabilities. Details must be specified in a config file which specifies information such as current locations for weather and preferences for news. The mirror is a great implementation, from which we were inspired to add a news feature. However, the lack of user interaction makes it seem lacking for a device people might desire in their homes for daily use. This enforced the idea of voice control for this project. Another issue with this project is the requirement of users to edit a config file to set basic information such as location and news preferences. In designing this smart mirror we intend for all user interaction to be as simple as possible so any user, no matter the level of technical skill, could set it up and use it. This project gave us the idea to create a mobile application that will be allow user to input more complex information than the voice commands on the mirror would be convenient for.

Another mirror project is MirrorMirror, originally posted on Imgur by user ctraltdylan. Dylan Pierce created this mirror as a simple home project but it evolved into a project being modified and worked on by a small community on a website he set up. This project is a simple implementation including features such as the date and time, the weather, and a random daily compliment. This mirror is one among many that is floating around the internet with some simple applications implemented. These simple mirror projects are great for an amateur who is tinkering with software implementation and design. However, if Smart Mirrors are going to take off one day, a lot of thought and time needs to be put into developing solutions to display as much useful information as possible in the most user friendly way possible. This goal is something that is going to be pursued specifically in this project.

These existing projects inspired this smart mirror project. The currently existing implementations posted around the web are generally simple and lacking in depth features. The goal for this mirror is to create in depth features to encourage a larger community of developers to begin implementing their own features to extend this product. One of the primary features that was inspired by these products was voice control. Two of these three projects have no user interaction which is an important aspect if smart mirrors are to become an item used in people's daily lives. Due to this, voice control is at the top of the list for this project.

3.1.2 Products

As far as actual production quality smart mirrors go, there is not much available on the market currently. This is due partly to the fact that smart mirrors are a relatively new idea but it is primarily due to the cost. The cost to produce a well-made, marketable smart mirror is high due to the cost of each component. One way mirrors start at about $150 for a thirty two inch diagonal mirror and only increase from there with size. Adding in the cost of a thirty two inch television and a small computer or raspberry pi and you are left with around $500 just for the cost of materials. This is not even including the amount required to pay developers for quality software development and maintenance. Due to these issues and potentially more, most of the items on the market seem to still be in production to some extent.

One of the primary motivators behind the smart mirror is Corning's Glass. Back in 2011, Corning released a video entitled "A Day Made of Glass…" this video paints an amazing image of smart technology on surfaces throughout our entire day. This video was a large motivator in interest in this smart mirror device. Later in 2012 Corning released another video, "A Day Made of Glass 2…" which explores even more possibilities of smart home technology. As time has gone on, Corning has turned more towards implementing specialized displays for televisions including paper thin displays, flexible displays, and specialized glass to withstand the wear and tear of daily use for smart phones and tablets. Corning paints an interesting image of being connected no matter where we are in a future they call "The Glass

Age." This is an exciting idea and had a huge impact on our decision to implement this project.

Another smart mirror currently in production is a product called SenseMi, or Sense Mirror. It is implemented as a large display mirror and it utilizes touch screen technology for interaction. Some of its main features include a Virtual Closet, where you are able to enter your clothes into a database in the mirror then the mirror can display the clothes on you. It also offers a Virtual Makeover feature for women and enables you to try out makeup before actually applying it. Another primary feature is the home integration feature which allows the mirror to act as a centralized controller to control systems around the home including air conditioning, home security, or lighting. You cannot currently purchase an implementation directly from their site, they currently only have a 'Get a Quote' button which gets you in touch with their sales team.

Another company, Tech20, markets mirror based displays for use throughout the home. These include mirror based televisions, the television works as normal but when it is off, the display appears as a mirror as well as specialized weather resistant televisions for outdoors. In 2012, the company released a video marketing an interactive bathroom mirror which would monitor health as well as daily information. This is a great idea, to incorporate health information such as weight, heart rate, blood pressure, and BMI onto a mirror which can easily track information over time. Unfortunately it seems nothing ever came of the idea and on the website currently the closest thing is a specialized mirror and television setup to display television through the mirror.

Aside from these three smart mirror implementations, a number of major television companies, including LG and Toshiba, have done small marketing ventures into smart mirrors but nothing is available on the market yet. The smart mirror is likely to hit markets at some point in the future, the question is, how far out is it? Though this technology receives a lot of interest from the public, it is not quite time for a marketable product due to issues primarily associated with cost. Though people may be interested in the product it is unlikely they would be willing to pay for a well-designed smart mirror as the cost could be upwards of $500. This leaves this item in the realm of tinkerers for the time being. This project is intended to be a proof of concept of the idea to show what could potentially be done with this type of technology.

3.2 Relevant Technologies

3.2.1 Operating System

The first decision to be made, which most decisions in the project will be built upon, is what operating system will be used to power the mirror's software. It is important to choose an operating system that is readily available for as many users as possible while also providing support for developers to create their own programs and functions to extend this project's functionality. It is also important that the chosen operating system is as lightweight as possible to allow the implementation to run on as many devices as possible. If the operating system is lightweight, then the mirror will be capable of running on small embedded systems as well as full powered custom computers depending on the desired applications implemented.

When considering an operating system, three main contenders come to mind, Mac OS X, Windows, and Linux. Due to the fact that Mac OS X does not meet any of the specifications listed previously, it will not be considered. This leaves us with the primary operating systems Windows and Linux. However, we can also consider mobile operating systems for this project which allows us to include Android. While there are a number of Linux distributions we will consider Linux as one category. For Windows, the regular user distribution will be considered as well as Windows 10 IoT. This brings us to four primary options, listed in Table 3.2.1.1.

Operating System Options	Windows 10	Windows 10 IoT	Linux	Android
Open Source Community	Some	Large	Large	Some
Availability on Compact Devices	Limited	Yes	Yes	Yes
Cost	\$119-\$199	Free	Free	Free

Table 3.2.1.1: Operating System Pros and Cons

Though each platform has its own advantages and disadvantages, the Windows platform was chosen primarily due to the introduction of the Universal Windows Platform (UWP) discussed in a later section. This platform is important because it means any software implemented will have the capability to run perfectly on any Windows device. In the end, the Windows 10 IoT platform was chosen due to its versatility and ability to work across a large number of platforms. Although Android and Linux offer some of the same capabilities, Windows 10 also allows for easy cross platform development. This will allow the mirror to easily be changed to different types of hardware at any point as desired. One unique element of the Windows 10 IoT platform is that there is no default GUI implementation. This is not an issue to the nature of this product, as it is preferred that the GUI is designed from scratch. This will allow us to create an application which will simply run on startup of the Raspberry Pi running Windows 10 IoT. The application will manage all elements of the display and no desktop environment will be required. Even if this application were to be utilized on a Windows system with a desktop environment, the application would simply need to be launched and it would work as intended.

3.2.2 Speech Recognition Software

It was decided that user interaction with the smart mirror will be handled mostly via voice interaction. Due to the model we are working with, there will be limited mouse and keyboard interaction and, due to the choice of computer platform, gesture control systems will also be unavailable. Fortunately, for this project and its intended uses, voice controls are the most convenient and useful form of interaction. The purpose of the smart mirror is to provide users with concise information as they go about their busy days. Utilizing voice controls allows users to use their hands for whatever it is they may be doing while simultaneously controlling the mirror. Aside from setup, sensory input, and a couple buttons, all day to day interaction with the mirror will be controlled via voice commands.

Due to the complexity of processing audio signals, voice control will be implemented using an existing speech to text API. There are a number of options available for speech to text however the three primary options considered were Google Speech API, Wit.AI, and Microsoft Speech Recognition. The final choice for voice recognition software was narrowed down to Microsoft Speech Recognition for a number of reasons. While all option were viable, Microsoft's API seemed the most straightforward while also providing plenty of power and flexibility for our requirements. The Google API was a close second though it offered capabilities beyond our requirements and, when programming on a relatively weak computer like the raspberry pi, advanced features simply cost performance. Lastly, Wit.AI seems to be an extremely powerful tool, however it utilizes standalone software for set up and command management and there is currently little information on utilization on small platforms such as Windows 10 IoT Core.

Once the Windows Speech Recognition platform was decided upon, the next step was to determine the required voice controls. While every voice command will start with a keyword to ensure the mirror knows you are talking to it, each application running on the mirror will have its own set of voice controls. The Windows Speech Recognition API allows specification of multiple command lists which can be loaded or unloaded at any time. This functionality allows for a smaller number of commands to be active at any given time, reducing the amount of potential mistakes for the speech to text software to make. Each application will have its own file of commands which will be loaded only when that application is running or, for certain applications, currently focused on the mirror.

The Windows Speech Platform uses what is called a SRGS, Speech Recognition Grammar Specification, file which specifies what commands are acceptable. The SRGS Grammar File utilizes keywords to determine the command the user is trying to give. If the grammar file is robust, it allows users to speak commands in a number of ways as they see fit. The software listens for keywords specified in the grammar to indicate the user's intent. For example, if a grammar is used to control a weather application, a number of keywords can be specified into groups which are assigned by the software. A simple example of this is shown in Figure 3.2.2.1.

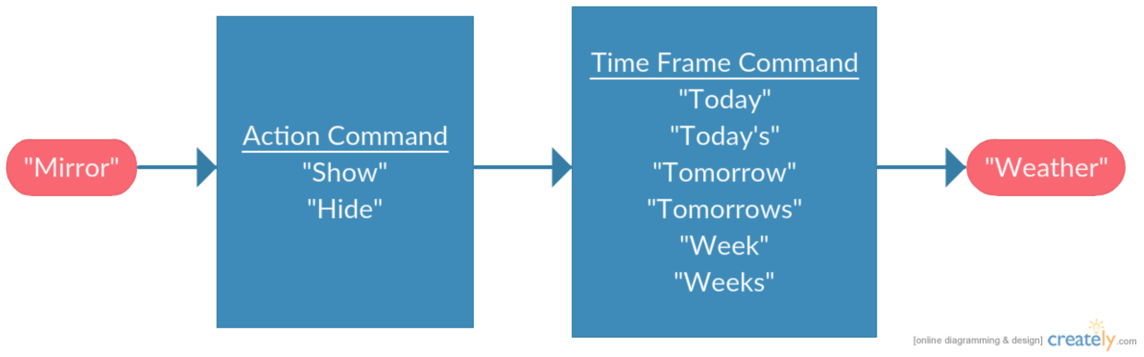


Figure 3.2.2.1: SRGS File Abstraction

This simple grammar setup allows users to utilize a number of commands to tell the weather application what to show. It allows users some freedom to speak in their own manner rather than requiring strict verbal commands. For instance, this example would allow a user to utilize the command “Mirror show me tomorrow’s weather” while another user might prefer “Mirror show me what to expect for tomorrow’s weather.” The user is able to use their own style of speech which the software filters for specific keywords. The grammar file can also set up a number of rules so that this sentence could be structured differently as well. A user may say “Mirror what is the weather like tomorrow” which, though asking the same information, since the keywords are ordered differently this could cause an issue however the grammar file allows designers to account for this by specifying different phrase orders utilizing the same keywords. This is an important feature as utilizing the voice commands on the mirror should feel comfortable and natural as if talking to something intelligent rather than memorizing a list of commands to recite.

Although one of the goals of this project is to make all commands as intuitive as possible, if the users is at a loss they will always be able to ask for a list of commands which will then be displayed upon the mirror in a specified location. The list of commands given when the users asks will be context sensitive so if the user is currently utilizing some application, the commands given in the list will be relative to that application specifically. The user will also be able to ask for general commands at any time to ensure they are able to return to the main display if need be. This will allow the user the ability to always be aware of all the possible functionality of the mirror which is important for the user to use the mirror efficiently.

3.2.3 Universal Windows Platform

Microsoft introduced the Universal Windows Platform (UWP) with Windows 10 to be a platform-homogeneous application architecture. The purpose of UWP is to allow developers to create apps that run on all Windows 10 platforms without the need to be re-written for each version of the operating system. Similar to how iOS apps are compatible across all iOS devices, the UWP allows a single application to run on multiple types of devices. When designing within the UWP, your target becomes device families rather than operating systems. This allows you to take advantage of the fact that there is a common API surface across device families. The core APIs of the UWP are identical; thus, apps that rely solely on these core APIs are able to run on any Windows 10 device. With our Smart Mirror, we are targeting the IoT device family. As a result, the APIs guaranteed to be available to our app include the APIs inherited from the universal device family in addition to the APIs that are particular to the IoT device family [1]. The device family tree is illustrated in Figure 3.2.3.1.

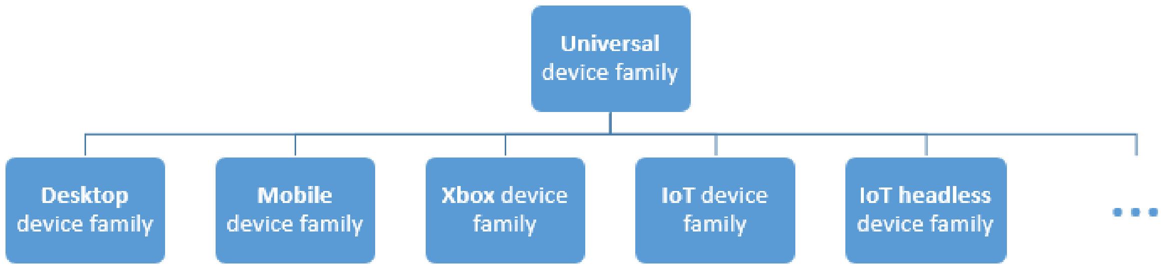


Figure 3.2.3.1 – Microsoft UWP Device Family Tree (Reprinted with permission from Microsoft)

The UI elements utilize effective pixels which are able to automatically adapt to various screen resolutions that can be found within the device family. The UWP exercises not only these adaptive controls but adaptive inputs as well. Native support exists for various keyboards, mice, touch controls, pen inputs, game controllers, and microphones. New layout panels and tooling facilitate the task of tailoring the UI to a specific screen size or device. The native development environment for UWP apps is Microsoft Visual Studio 2015, a comprehensive IDE with all of the necessary programming, debugging, compiling, building, and deploying tools. All phases of project development, from conception to deployment, can be realized within Visual Studio 2015. A handful of programming languages are supported including C++ with DirectX and/or Extensible Application Markup Language (XAML), JavaScript with HTML, and C# or Visual Basic with XAML. Implemented into each of these languages is the Windows Runtime, a native API built into the operating system [2].

3.2.4 Windows Cortana

Windows 10 also introduced Cortana, Microsoft's new personal assistant to rival the functionality of Apple's Siri and Google Now. The voice-activated Cortana allows you to search for files on your computer or the internet, manage your calendar, tell jokes, and much more. Cortana is heavily implemented into Windows 10 and, as a result, is aware of files and information on your computer to provide context-aware interaction. This results in a smarter and more efficient means to accomplish tasks such as setting reminders, checking the weather, or tracking a package. Because Cortana is part of the Universal Windows Platform, apps can be launched using your voice, and speech-to-text input is also supported within the app.

Cortana could be a useful application on the smart mirror due to her ability to communicate with the user. She is able to search things throughout the user’s file system, OneDrive, and even the internet. If implemented onto the smart mirror she can provide a wealth of information and accessibility in an instant that could take months to recreate. As a result, implementing Cortana on the mirror is a feature that would add great value to our project; however, though this feature could bring a lot of functionality to the mirror, it is not one of the main requirements because Cortana is not necessary to achieve our goals for the basic aspects of the mirror. The incorporation of Cortana would be placed towards the end of the list of priorities; though if implemented, could prove to be extremely useful.

3.2.5 Applications

Each software feature of the mirror will be designed from scratch and placed appropriately within the UI. Rather than executing individual widgets or gadgets, the different software features will be self-contained and built into a single UWP application that is executed on boot in order to provide a seamless user experience. The consolidation of each software feature into a single persistent application simplifies the design architecture and provides a unified experience.

The layout of the applications is actually one of the more important aspects of this project. The reason for this is that, if the mirror is not well thought out, easy to use, and does not intrude on daily use of a mirror, then users are unlikely to enjoy the product or utilize it daily. Due to this, much thought has gone into the layout of the applications on the mirror and more changes will likely be made as prototyping continues. Currently it is planned for all mirror applications to remain around the edges of the mirror and be presented in the smallest ways possible unless the users asks for more information. This format will allow a wealth of different information to be presented but not in much depth. Fortunately, all the user will have to do is ask for more information to replace that application with an in depth version of whatever they asked for. For the calendar this will mean if an event is selected, the calendar will be replaced with the details of the event. This same idea will hold for all other applications as well. Each element of the application will have its own block of mirror space where all its information will be presented. These locations will be customizable by the user to ensure the user can set up the layout to their liking as some users may prefer the time on the bottom center while others may like it in the top left. This customizability and layout design is what will set out mirror apart from other such projects and ensure ease of use for all users.

Shown in Figure 3.2.5.1 is the default layout of the mirror blocks. Since there is not enough space to provide all applications in a clean manner, some will have to be requested to be displayed. Any the user requests will show up in the ‘Extras’ section on the mirror. This block is slightly larger than all the rest, this is simply to ensure that there is always enough space to provide the applications the users deems as not important enough to have permanent mirror real estate. All applications displayed within the ‘Extras’ block will use their normal profiles as close to the edge of the mirror as possible. Certain items, including the To-Do list, the music information, and possible commands, will only be displayed under certain conditions as listed in their subsections.

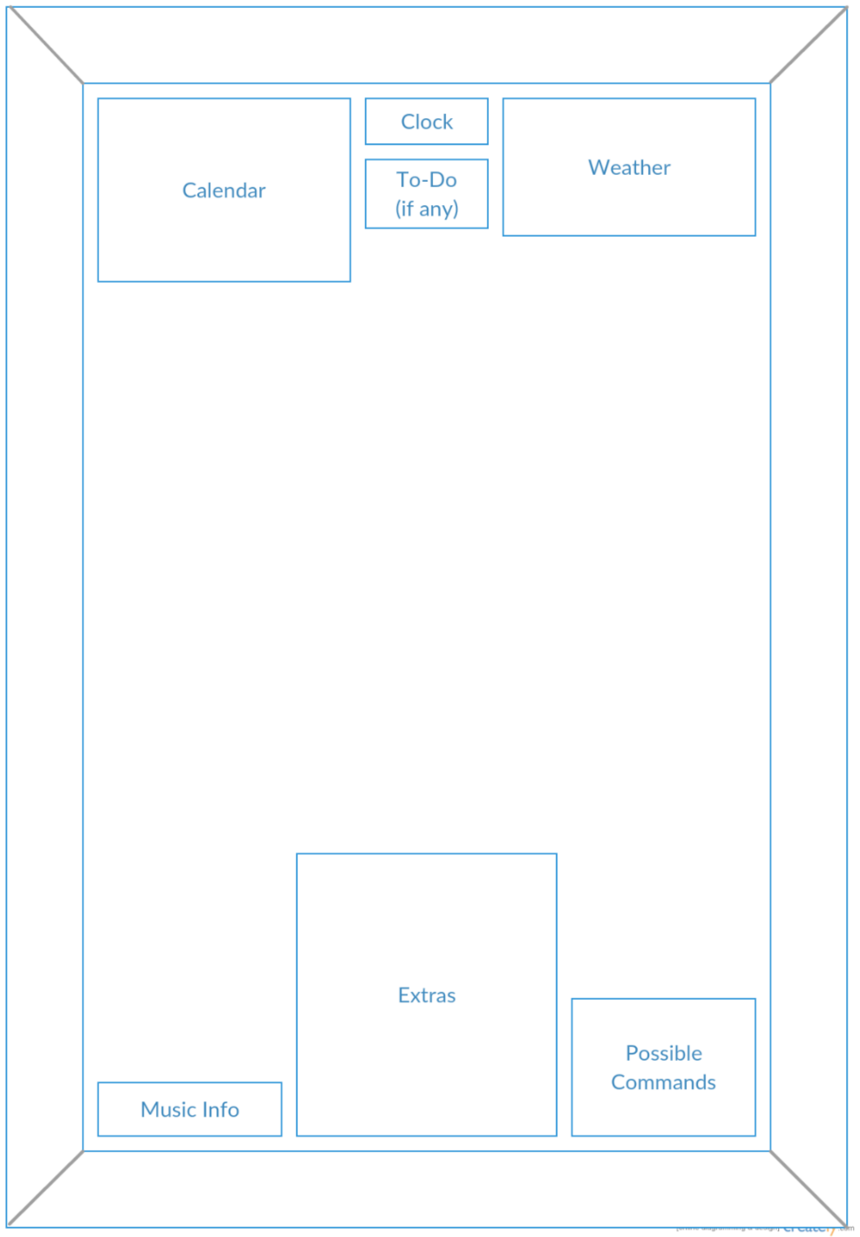


Figure 3.2.5.1: Default Smart Mirror Display Layout

3.2.5.1 Clock

The clock is one of the simpler components of this project and we want it to be as customizable as possible. The user will be able to choose the location of the clock to match their needs and desires. Users will also be able to select from a list of fonts and colors for the clock display, allowing them to make the mirror individual to their personal tastes. The clock will be customizable and it will be possible to set the clock as a digital or analog clock depending upon the user's preference. For the digital clock, the time will be able to be set as 24 hour time or 12 hour time and the user will also be able to select whether the clock precision is displayed to seconds or minutes. The analog clock will be implemented with only an hour hand and a minute hand; seconds precision will not be offered on this element. The user will have the option to display the numbers as roman numerals or standard hindu-arabic numerals.

The clock implementation, by default, will only display the time. However, for users who wish to see the date as well, there will be an option to show the date in a number of formats. The user will be able to select all the formatting options utilizing the created external application to manage settings. The time is one of the core elements of the mirror and it is important in helping people keep track of time as they prepare themselves for the day in the morning. Due to this, it is important for the clock to be one of the more customizable components of the mirror.

3.2.5.2 Weather

The weather app will display basic weather information including the temperature at your current location along with the high, low, and chance of precipitation. A simple graphic, such as a sun or raincloud, representing current weather conditions will be displayed as well. This information will be updated every hour, providing the user with an accurate estimation of what to expect at any time of the day. This is an important feature in assisting the user in planning for their day as they get ready in the morning. This feature will be displayed on the mirror at all times.

The weather application will also offer forecast options for the user which, via voice command, will be able to be toggled on or off. These forecast options will replace the current weather display for a set period of time before fading back to the current weather. The user will be able to glean information about the next days weather as well as about the weather forecast for the upcoming week. Each display element will be customized in a way to display as much information as possible in the smallest space to ensure there is still space around the mirror for its general use, reflection. This feature is important in saving a user time in the morning that they may utilize to check their computer, smartphone, or television to determine how they should dress or whether to bring an umbrella for the day.

3.2.5.3 Calendar

The calendar will be implemented as a list of events for the current day. An example of this can be found in the Day view of Google Calendar. The events on the calendar will be updated hourly to ensure the user is always up to date on their events. As the calendar will implement a day view of events, the current time will be indicated by highlighting the hourly row that corresponds to the current time. The calendar will display 8 hours at a time and the current time will be the 2nd hour on the calendar and the display will be updated hourly to shift the view bringing the next hour into view. There will be voice commands to control the calendar by opening events at specific times as well as to scroll the calendar to the next hours or to see tomorrow's information.

The calendar will incorporate events from Google Calendar and the user will be required to log in to their google account on the external application developed. Once the user logs in, the application will ask for verification that the users information may be displayed as this is required by all Google APIs. At the start, only Google Calendar will be implemented however, if possible, later on Outlook calendar may be included as well so that users are able to see information from whatever calendar application they use in their day to day life.

The calendar is an important application for the mirror as it allows the user to see events they have planned for the upcoming day at a glance. This will help users ensure they do not miss any events planned for that day. The ability to check the calendar for tomorrow allows users to get an idea of what their day will be like the night before. This is a great way for users to prepare for their day the night before by ensuring they remember everything they have coming up the next day.

The calendar application will also be interactive with the user. If the user is in front of the mirror and realizes they need to add an event, they will be able to command the mirror to create a new event at a specified time. The mirror will then open a new event window which will allow the user to enter information such location, people, and reminders to the event.

Another important aspect of the Google calendar that will be brought to the smart mirror is Reminders. Reminders help a number of people in their daily life and are actually one of the easier items to set on an Android device. Due to this, reminders will be brought to the smart mirror to ensure users notice their reminders in the morning. If users grow accustomed to utilizing the smart mirror in the mornings and don't check their phone, they could potentially miss reminders which is a consequence that would be ideal to avoid. By allowing reminders to be utilized on the mirror we allow users another step in the direction of saving time and increasing productivity and efficiency.

Reminders will be displayed differently from other calendar elements and could almost be implemented as a separate section however since they are tied directly to google calendar they are being included here. Since reminders are meant to appear at certain times only, the reminder will only appear on the mirror if it is currently active and it has not been dismissed on any other devices. The reminders will appear directly above the calendar display keeping all event based information in one location on the mirror. This is an important simplification as users will always be looking to that area of the mirror to check on their events anyways.

3.2.5.4 To-Do List

A to-do list is a helpful item in any person’s life. A to-do list connected to all your devices is even better. The intention of adding a to-do list to the smart mirror is to offer users the ability to keep track of items even beyond their daily calendar that they have ahead of them. This will allow users to be reminded daily of potentially time sensitive tasks that didn’t seem to fit their calendar specifically. This will help users stay on top of all the important things in their life.

For the to-do list implementation on the smart mirror we will be using the popular to-do list application called Todoist. It is an established to-do list service with applications already developed on Windows, Mac OSX, Android, and Linux making it an already versatile tool. The company also offers an open API for developer use which will allow us to seamlessly implement this application into the smart mirror.

This application will not be one of the ever present staples on the smart mirror and will only be utilized when the user asks for it. When the user commands the mirror to show the to-do list, it will be toggled on in place of the calendar application. The to-do list will show all the main to-do items the user has while the user will be required to select a to-do list item to expand it to see any subtasks within. This will allow us to present as many to-do items possible on the mirror with the least space. By default the to-do list will show only the current day’s items but at a command from the user the list will be able to be expanded to include the next day or even the upcoming week’s items.

3.2.5.5 Music

In order to provide another step of convenience and functionality to the user, music playback will be created to allow the user to listen to their music while they utilize the mirror. The music playback will be controllable via voice commands as everything else on the mirror is. The mirror display will house a small player that shows the current song name, artist name, and album name of the current song as well as a slider to indicate the current position within the playing song. The location of this information by default will be located in the lower left corner of the mirror. One of the commands will allow the user to see the names of all playlists they have currently stored on the device, this will be shown in the Extras location on the mirror and the user will then be able to select one of the playlists.

The music application will be implemented utilizing an online streaming service or music stored directly on the Raspberry Pi. The original idea, to use the Spotify web API is unfortunately not possible as it does not allow full audio playback. A number of other options were looked into and the only one that could possible work is Microsoft Groove. Due to this only being a possibility, it may be required for users to select music from their own personal collection to store on the Raspberry Pi. More details on these issues are discussed in the design discussion in section 5.4.6.

3.2.5.6 Travel Time

The mirror will allow a user to enter locations on their calendar events and it will utilize the calendar to provide users with travel time to their events for the day. This will be implemented as a simple travel time rather than an entire map. This feature is extremely useful as it will let the user know what to expect when traveling to their destination that day. If the user wakes up and sees that the travel time to their destination is twice that which is normal, they can ensure they get ready faster and leave in time.

The mirror will also allow the user to enter a work location and hours via the external application. If this is done, then for the hour prior to the time the user needs to leave to arrive on time, the mirror will display the travel time to the location specified for work. This will allow users who work every day to set reminders of different types of events but still ensure their travel time to work shows up at the appropriate time.

A full map display will not be offered as this would take up a significant portion of the mirror to implement and it is not necessary during normal daily routines. The user will be expected to use external devices to plan out an external route or see exactly where the delay is located. For this part of the application to function, it will be required for the user to enter their home location via the external application at set up. If a home location is not set then this portion of the mirror will be disabled by default as there will be no way to calculate the time to the location.

3.2.5.7 News

In order to provide the user with a few news events, a simple news feed will be displayed to provide three headlines from world news. After researching various different news feed APIs, we have decided to use CNN’s World News RSS feed. The other news outlets we considered included USA Today, The New York Times, and Associated Press. Our decision was based on simplicity of use and straightforward parsing ability. The alternatives were either down for maintenance, required complex API structures, or didn’t provide the scope of news stories we sought.

The news content will be centered near the bottom portion of the display in a bulleted-list fashion. The RSS feed will be converted to a JSON format which can be easily parsed and refreshed every hour. We will extract the headline as well as a teaser thumbnail image that is provided by the RSS feed. If no thumbnail is provided, a simple graphic representing the newspaper will be displayed in its place. CNN provides many different RSS feeds to cover different sections of the news such as Top Stories, World, U.S., Money, Technology, etc. Because all of the different feeds share the same internal format, we can easily change between them by simply exchanging the source link from one category to another. The news feed category we have decided to implement is World News.

3.2.6 Temperature Sensor

As it was mentioned previously, the temperature sensor will be needed to maintain the inside of the mirror cool enough to keep the component operational. The temperature sensor will be a component in the temperature control mechanism that will be discuss on the later sections of the project. There are multiple types of temperature sensors in the industry such as thermocouples, Resistance Temperature Detectors (RTDs), Thermistors, Infrared, and Integrated Circuit Sensors.

Thermistors
Thermistors temperature sensors are resistors that change their physical resistance depending on the changes in temperature. Thermistors are manufactured with two different set ups, Negative Temperature Coefficient (NTC) or in Positive Temperature Coefficient (PTC), for Negative Temperature Coefficient the value of the resistor decreases as the temperature increases and for the Positive Temperature Coefficient the resistor's value increases as temperature increases [3]. Since Thermistor sensors are resistors that change their values depending on the temperature, we would need to pass a current to measure the difference in voltage. Figure 3.2.6.1 illustrates the proper connecting for a Negative Temperature Coefficient Thermistor Temperature Sensor. According to the source, "The Thermistor, have an exponential change with temperature…", therefore are a non-linear device unless, it's used in a voltage divider network, if so the output voltage becomes linear with the temperature.

Resistive Temperature Detectors
Resistive Temperature Detectors (RTD) sensors are similar to the thermistors since they also change their resistor values depending on the temperature. RTD sensors are made with a higher purity conducting metal, which makes them a precision temperature sensor. The coefficient values for the resistance are positive (PTC). When comparing the RTD sensor with the Thermistor, the output from the RTD is extremely linear which makes a more accurate temperature, the downfall of the RTD's is that they do not have much thermal sensitivity [3]. Since the RTD sensor are also a linear device, we are going to need to apply a current

to create a voltage change which behaves linearly depending on the temperature, in general the resistance value at the freezing point of water is 100 Ω, and the operating temperatures are from -200ºC to +600 ºC. In general when using the RTD sensors in the design, the sensor is connected into a Whetstone Bridge network to avoid any inaccurate readings due to any self-heating from the device [3].

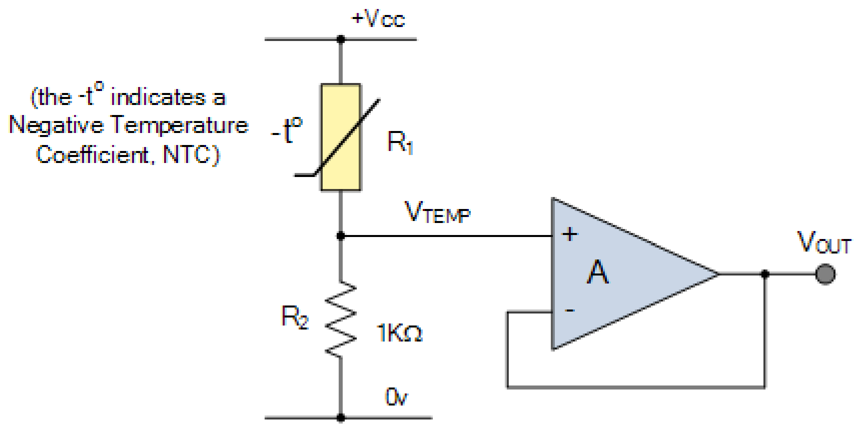
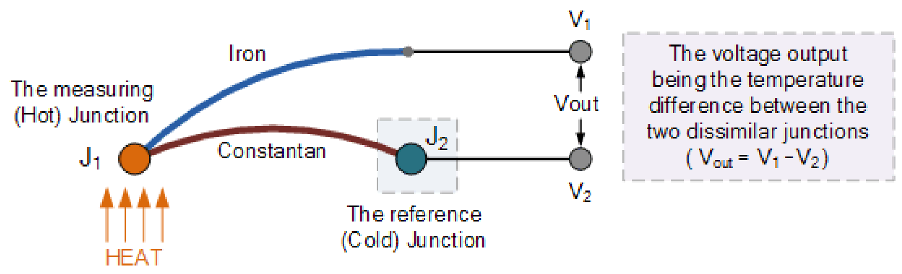


Figure 3.2.6.1: Proper Connection of a Thermistor- Recreated with data provided from Electronics Tutorial

Thermocouple

Thermocouple are thermoelectric sensors which use two junctions of different metals, one of the junction is used as a reference temperature usually as the lower temperature one, and the second junction measures the larger temperature. The sensor creates a voltage across both junctions as long as there is a difference in temperature between them, this voltage is very small (just a few millivolts for a 10ºC change which means an amplifier is required [3]. The Thermocouple sensors have a fast response time when measuring temperature, these are the most common temperature sensor used. Figure 3.2.6.2 shows the construction of the Thermocouple sensors.



V1
Iron
The measuring (Hot) Junction
Vout
The voltage output being the temperature difference between the two dissimilar junctions (Vout = V1 - V2)
J2
J1
Constantan
V2
HEAT
The reference (Cold) Junction

Figure 3.2.6.2: Construction of the Thermocouple- Recreated with data provided from Electronic Tutorials

Integrated Circuit Temperature Sensors
Integrated Circuit (IC) Temperature Sensors use two terminal integrated circuit temperature transducer, which depending on the temperature produce an output current [4]. Integrated Sensors can have different types of outputs such as voltage, current and digital. These sensors have a quick response time and a low thermal mass, the common temperature range are usually from 55 to 150 Degrees Celsius. The IC sensors that have a digital output have a built in analog to digital converters, and depending on the numbers of Bits in the A-D converter, it provides the resolution (10 Bit provides temperature increments of 0.25 C and for a 12 Bit is 0.0625 C) [4]. The advantages of using the IC sensors are that they are a low cost component, with the A-D converter the output can be either analog or digital with no additional circuitry, and they have a linear output. The disadvantages of IC sensors, the temperature can only range between -55 to 150 degree Celsius.

3.2.7 Light Sensor

Light sensors are a passive device which change their energy from the light into electricity, usually referred to as Photoelectric Devices or Photo Sensors. There are two classifications for the light sensors; Photo-voltaic, which creates electricity when the sensor is illuminated, or Photo resistors which change their electrical properties.

Light Dependent Resistor
Lights Dependent Resistor (LDR) is made of a semiconductor material which changes its electrical resistance depending on the amount of light that the sensor is expose to [5]. The LDR sensor have a long response time for the change in the light, the sensor reduces its resistivity when it's exposed to light which creates a better conductivity. Figure 3.2.7.1 illustrates the change in resistance depending on the illumination.

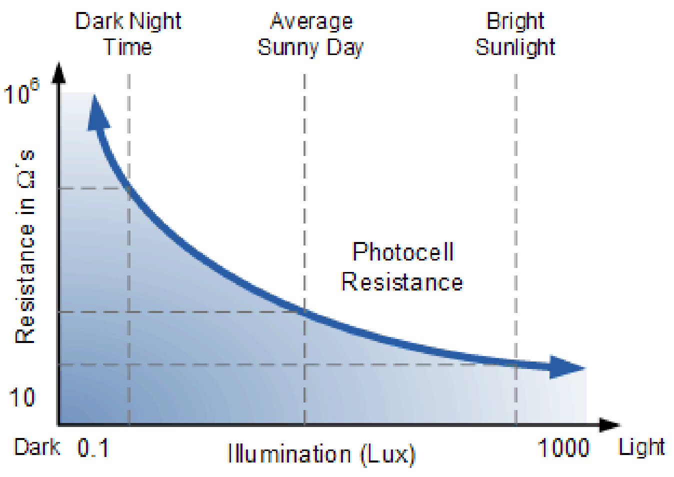


Figure 3.2.7.1: Resistance response - Recreated with data provided from Electronics Tutorials

The most commonly used photo resistive light sensor is the ORP12, this sensor increases its resistance value when there is an absence in light, this causes low or no current to flow through the sensor [5]. The top of the sensor has a zigzag pattern which creates the higher resistivity, commonly in the Mega ohms. If the LDR is in series with another resistor, we can apply a voltage divider and determine the voltage across the second resistor, the higher the resistivity on the LDR the lower the voltage across the second resistor [5].

Photodiode Light Sensor
The Photodiode light sensor is similar to the conventional PN- junction diode, the main difference for the photodiode the outer casing is transparent to let light into the PN junction to increase the sensitivity. One of the main advantages of using a Photodiode is its fast response to the change in illumination, but the disadvantage is that the current flow produced is very low even when the Photodiode is fully illuminated [5].

Phototransistor Light Sensor
The phototransistor light sensor is the same as a photodiode light sensor, the difference is that the phototransistor has its collector-base PN junction reverse biased which is expose to the light source. Figure 3.2.7.2 displays this characteristic.

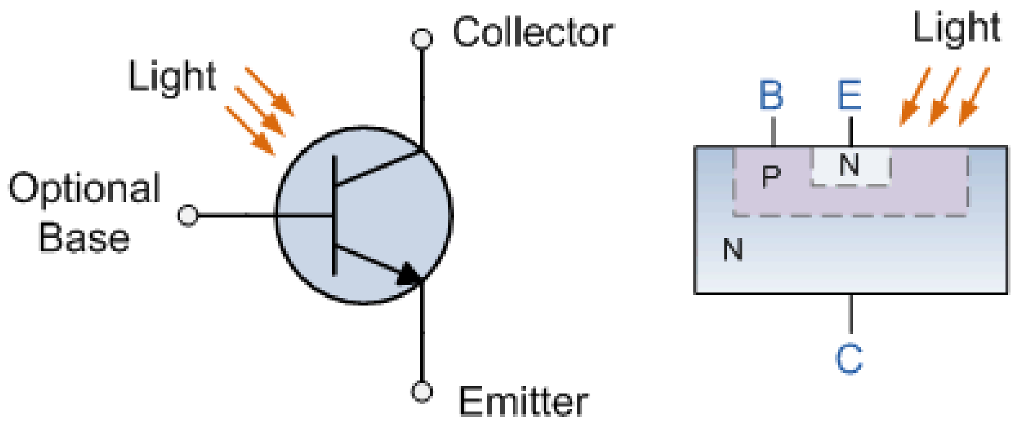


Figure 3.2.7.2: Phototransistor Characteristics - Recreated with data provided from Electronic Tutorials

Since the collector-base junction is reverse biased, there is a very small current going through when there is no light presence but when the light shines on the base of the transistor a higher current is produced and then amplified by the transistor [5]. When a second NPN bipolar transistor is introduced then the transistors become a Photo-Darlington, which is used to amplify the current even further which creates more sensitivity, the downfall when using this set up is that the response time is slower when is compared with a phototransistor [5]. Figure

3.2.7.3 demonstrates the setup of the Photo-Darlington. The typical applications of a Phototransistor are in opto-isolators, slotted opto switches, light beam, sensors, fiber optics and TV type remote controls.

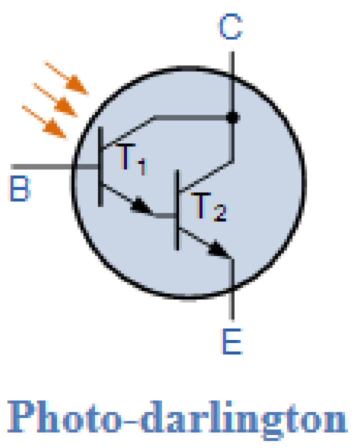


Figure 3.2.7.3: Photo-Darlington Setup - Recreated with data provided from Electronic Tutorials

Photovoltaic Cells

The most common type of photovoltaic light sensor is the Solar Cell. This type of sensor directly converts light into a DC electrical energy. This type of sensor is best used under direct sunlight. Solar Cells are used as an alternative source of power for batteries, this offers a renewable source of power [5]. Photovoltaic Cells are like the Photodiode in the sense that it uses a PN junction, but the main difference is that the Photovoltaic cells do not use a reverse bias. Which has the same characteristics as a very large photodiode in the dark. Figure 3.2.7.4 shows the relation between the current and voltage depending on the amount of light being exposed to the cell.

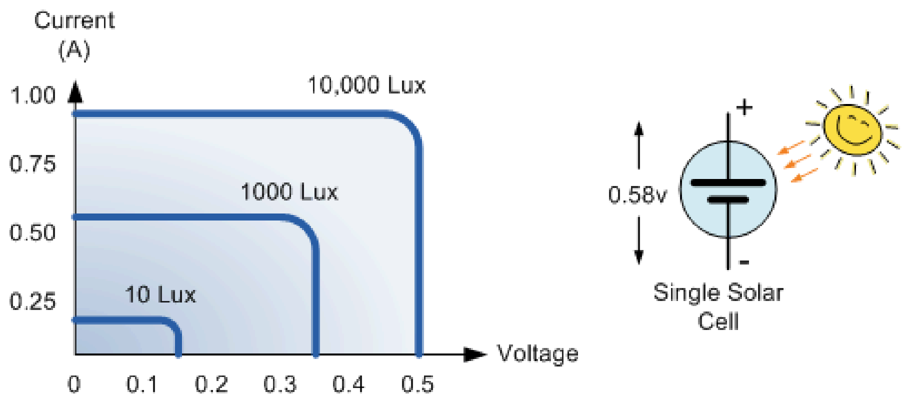


Figure 3.2.7.4: Current response with to voltage - Recreated with data provided from Electronic Tutorials

3.2.8 Motion Detection

In the previous Smart Mirror project, a gesture sensor was used as a motion control. One of the differences between our project and previous one is that we are only going to include a motion sensor to determine when the Mirror will turn on. There are multiple proximity sensors such as inductive proximity sensors, capacitive sensors, Photoelectric, Ultrasonic sensors.

Inductive Proximity Sensors
Inductive Proximity Sensors respond to ferrous and nonferrous metal objects [6]. This proximity sensor is commonly used for modern high speed process control for detection, also for counting and positioning any objects with any ferrous and nonferrous metals. Usually used to upgrade the speed and reliability of existing machinery resulting in replacing limit switches.

Capacitive Sensors
Capacitive sensors respond to any substance with a high dielectric constant with necessarily making any physical contact. Capacitive sensor are less suitable for any substances that have low densities [6]. This type of sensor is responding to the change in a dielectric medium that is around the active face. Capacitive sensors are typically used for level control of non-conductive liquids, granular substances, and substances through a protective layer.

Photoelectric or Opto-electronic Sensors:
Photoelectric sensors consist of a light source and a detector. The light source send either an infra-red or a visible light energy to an object which reflects back the send energy to a detector. This non-contact sensor is able to sense objects up to 10 meters away from it. This type of sensor is increasingly used since they can detect over a greater distance than previously mention sensors. Photoelectric sensor have multiple modes such as Infrared Proximity, Transmitted Beam, Retroreflective, Polarized Retroreflective, Fiber optic, and Background Rejection [6]. The Transmitted Beam and the Retroreflective will not be considered for this project, since we only want to detect whether the user is in the room when the Smart Mirror is being used and these type of sensors use a beam that need to be interrupted to activate. For this project the only photoelectric sensor that can be best implemented is the Infrared Proximity, since this sensor detects the light being reflected back to the detector. We might keep the Background Rejection Sensor as a backup, since this type of sensor is mainly used to ignore any movement outside a range, for example we can have a range where the sensor actives if there is movement within a foot or two from it.

Ultrasonic Sensors
Ultrasonic Sensors are uses a high frequency waves to detect objects or distances to the object. There are two basic type of sensors, Electrostatic and Piezoelectric.

Since this sensor uses high frequency this sensor will not be considered for the project.

3.2.9 Humidity Sensor

A key component to keep the electronics inside the Smart Mirror is a Humidity Sensor. when selecting a humidity sensor is important to keep these consider these specifications; accuracy, repeatability, interchangeability, long-term stability, ability to recover from condensation, resistance to chemical and physical contaminates, size, packaging, cost effectiveness [7]. Other specifications worth to note are; cost associated with sensor replacement, field and in-house calibrations, and the complexity and reliability of the signal conditioning and data acquisition circuitry.

Capacitive Humidity Sensors:
Capacitive Relative Humidity sensors are commonly used for commercial, industrial and weather telemetry applications. These sensors are made using a metal oxide that is in between two conductive electrodes [7]. When comparing the change in the dielectric constant of the capacitive sensor with the relative humidity (RH) of the environment, it noticeable the near proportional relation, Figure 3.2.9.1 illustrates said relation. The response time is generally elapses between 30 to 60 seconds for a 63% HR step change. One of the limitations for the RH sensor is distance away from the signal conditioning circuitry, the practical limit is less the 10 feet [7].

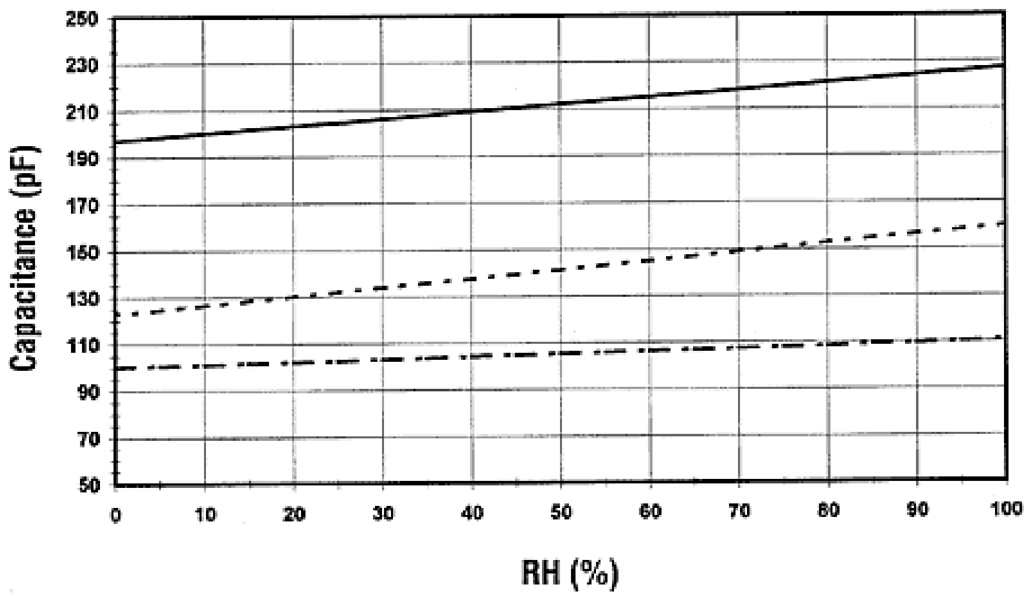


Figure 3.2.9.1: Dielectric Response to Relative Humidity - Recreated with data provided from Sensors Online

Resistive Humidity Sensors:
The resistive humidity sensors usually consist of noble metal electrodes either deposited on a substrate by photoresist techniques or wire-bound electrodes on a plastic or glass cylinder [7]. The substrate is coated with a salt or conductive polymer. This result in the sensor measuring the change in impedance of the hygroscopic medium. The sensor absorbs the water vapor and ionic functional groups are dissociated, which results in an increment in electrical conductivity. The general lapsed time for these type of sensors is from 10 to 30 seconds for a 63% step change, and the impedance ranges from 1 kΩ to 100 MΩ, Figure 3.2.9.2 illustrates the relation between the output DC voltage and the RH %.

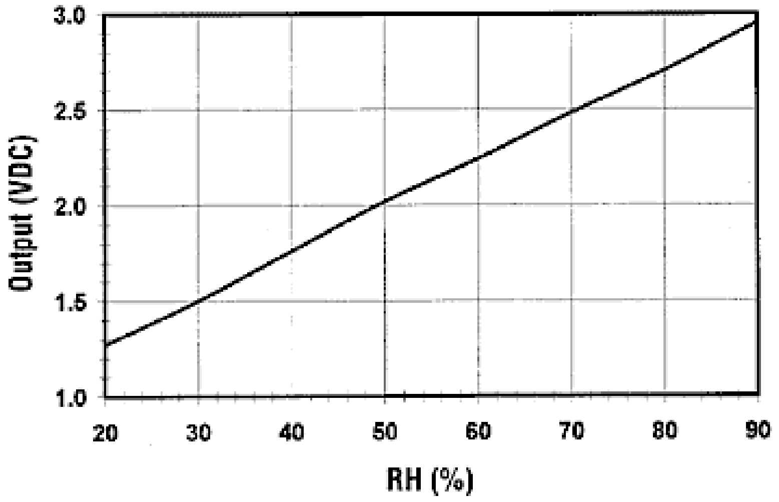


Figure 3.2.9.2: Voltage Response to Relative Humidity - Recreated with data provided from Sensors Online

In general these sensors have a life expectancy of more than 5 years as long as they are not exposed to chemical vapors or any other contaminants. When resistive sensors are installed in environments that have a large temperature fluctuation, it has a lot of temperature dependencies.

Thermal Conductivity Humidity Sensors
Thermal Conductivity Humidity Sensors measure the absolute humidity by quantifying the difference between the thermal conductivity of dry air and that air containing water vapor [7]. These sensors are made up of two negative temperature coefficient (NTC) thermistors that are matched, one of NTC is expose to the environment and the other one is encapsulated by dry nitrogen. Since the thermistors change their resistivity depending on their temperature, the sensor is creating a proportional relation between the voltage (once we apply a current to the thermistor’s resistance) and the absolute humidity presence, Figure 3.2.9.3 illustrates this relation. The advantages of using thermal conductivity sensors are that they are durable, they operate at high temperatures (up to 300 degrees C) and they are resistant to chemical vapors.

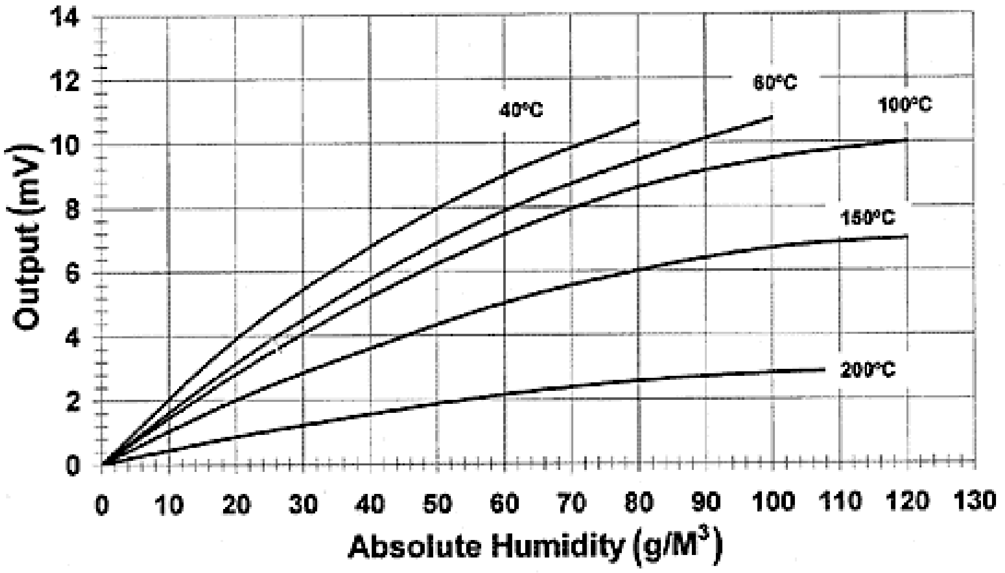


Figure 3.2.9.3: Output Voltage Response to Absolute Humidity - Recreated with data provided from Sensors Online

3.2.10 Audio Capture

Audio input will be provided via USB microphone. The primary purpose for audio capture is to receive speech recognition requests; thus, it is an extremely important component of the mirror, serving as the sole bridge between the user and the mirror software. Should the microphone fail to perform its function, the user would have no way to interact with the mirror. It is important to place the microphone in an effective location, taking into consideration the possibility that music may be playing from the internal speakers that can interfere with any speech recognition requests. A small hole will be cut into the front of the mirror frame towards the bottom to allow for a clear path for voices to be picked up. This location is ideal because it is far away from the internal speakers which are mounted high on the sides of the mirror. It is also worth noting that the microphone must be sensitive enough to sense faint voices. The USB microphone we have elected to implement into our mirror boasts an impedance of 2.2KΩ and its sensitive to -58dB ± 1dB [8]. Erroneous speech inputs can be tuned out through software thresholds.

3.2.11 Microcontroller

Microcontroller Units (MCU) are a self-contained system with peripherals, memory and a processor that can be used as an embedded system. In the modern era most MCUs are embedded in phones, cars, house appliances and many other consumer products. MCUs can be a sophisticated systems regarding programing and memory which become more complex, or they can be very minimal in terms of programming and memory capacity. The MCU will be used to process the data from each of the sensors mentioned above and send it to the Raspberry Pi. The programmable MCUs are usually categorized by several parameters such as Flash size, RAM size, number of input/ output lines, packaging type, supply voltage and speed. Microcontrollers are also categorized on their processing bits, 4-bit, 8-bit, 16-bit, and 32-bit. The 4-bit microcontrollers are commonly used for electronic toys [9]. 8-bits microcontrollers are generally used for control applications like position control, speed control and many other process control system. The 16-bits microcontroller are developed for a higher speed control application than the 8-bit one, an example of a higher speed control would be robotics. The 32-bits typically used for very high speed operations in robotics, image processing, telecommunications, and intelligent control system.

4. Hardware Design

4.1 Design Discussion

This project has two major components that will be working together to accomplish the tasks required. The Raspberry Pi 2 will control all software components and handle all aspects of displaying information to the user. The MCU will handle all sensory inputs excluding audio to interact with the environment in the most helpful and convenient way possible. Shown in Figure 4.1.1 is a block diagram of the overall system indicating which member is responsible for which aspect of the design.

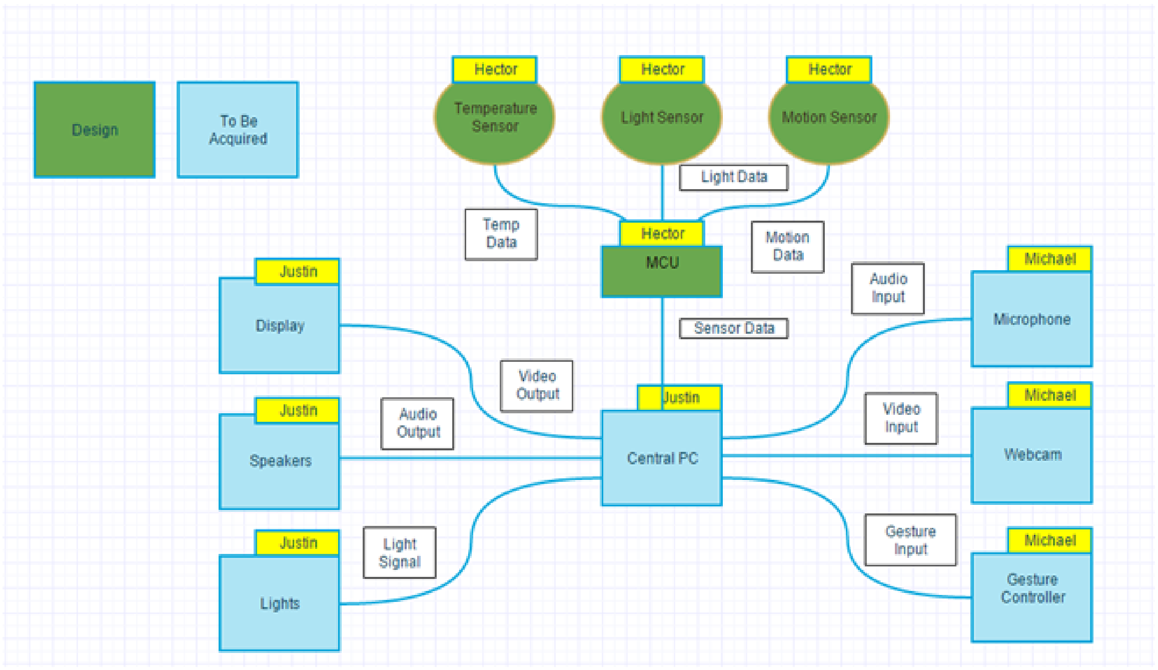


Figure 4.1.1: Overall Hardware Block Diagram

As shown, the MCU will interface with a number of sensors to monitor the environment around the mirror including temperature, light, and motion. Each sensor will be mounted onto the mirror appropriately to record information that will be passed back to the MCU which will then be able to communicate with the central computer as necessary. The central computer will be implemented using a Raspberry Pi and will interact with all specified components of the system in Figure 4.1.1. The individual components will be discussed and selected in the coming sections as well as details on their implementation.

4.2 Mirror Housing Construction

4.2.1 Requirements & Constrains

The housing of the Smart Mirror will be designed in a way that is cost effective but also displays the mirror and software in the clearest way possible. The housing will be built from wood with a face panel made of four miter cut boards as shown in Figure 4.2.1.1. The wood will be done with a burned finish then stained for a nice presentation that people might like to have in their homes. It is important that this is presented as a quality display piece otherwise most people would have little interest in having it in their home. All sensors and buttons will be hidden to present the image of a regular mirror.

The way the mirror works is by utilizing a one way mirror behind which is placed a television. This will allow light to shine through the mirror wherever the television is lit but it will simply reflect like a mirror wherever there are black pixels on the display. To accomplish this, the one way mirror will set into the face panel slightly and the television will be mounted directly behind it.

4.2.2 Housing Design

All bezeling and plastic housing will be removed from the television so it can sit as close to the mirror as possible. There will be a small rubber seal between the metal face of the television and the mirror to avoid any unnecessary pressure on the mirror. This configuration will create a dark environment behind the mirror, which is required for a one way mirror to work efficiently. Keeping the television as close to the mirror as possible is important because it will prevent light from bleeding across the mirror and will localize the light to the areas where the display elements are located.

Since this mirror is not quite like a regular mirror, there will need to be a housing built around the television which will extend 5 inches behind the face panel. The television will mount to the back of the face panel and will be held in using brackets and screws. The speakers from the television will be mounted to the sides of the mirror housing to allow music to be played via the television. The control buttons for the television will be mounted to the bottom of the mirror housing allowing manual control of the television if necessary. An opening for the Raspberry Pi will be located on one side of the mirror housing to allow access to the USB ports and the Ethernet port if needed as shown in Figure 4.2.2.1. If the mirror has internal speakers these will also be implemented on the sides of the mirror housing to output sound appropriately but to ensure a clean presentation from the front of the mirror. Lastly, a number of small holes will be made in the key locations on the face panel of the mirror to allow for placement of the sensors, including the motion detector, the light sensor, and a microphone to take user voice commands.

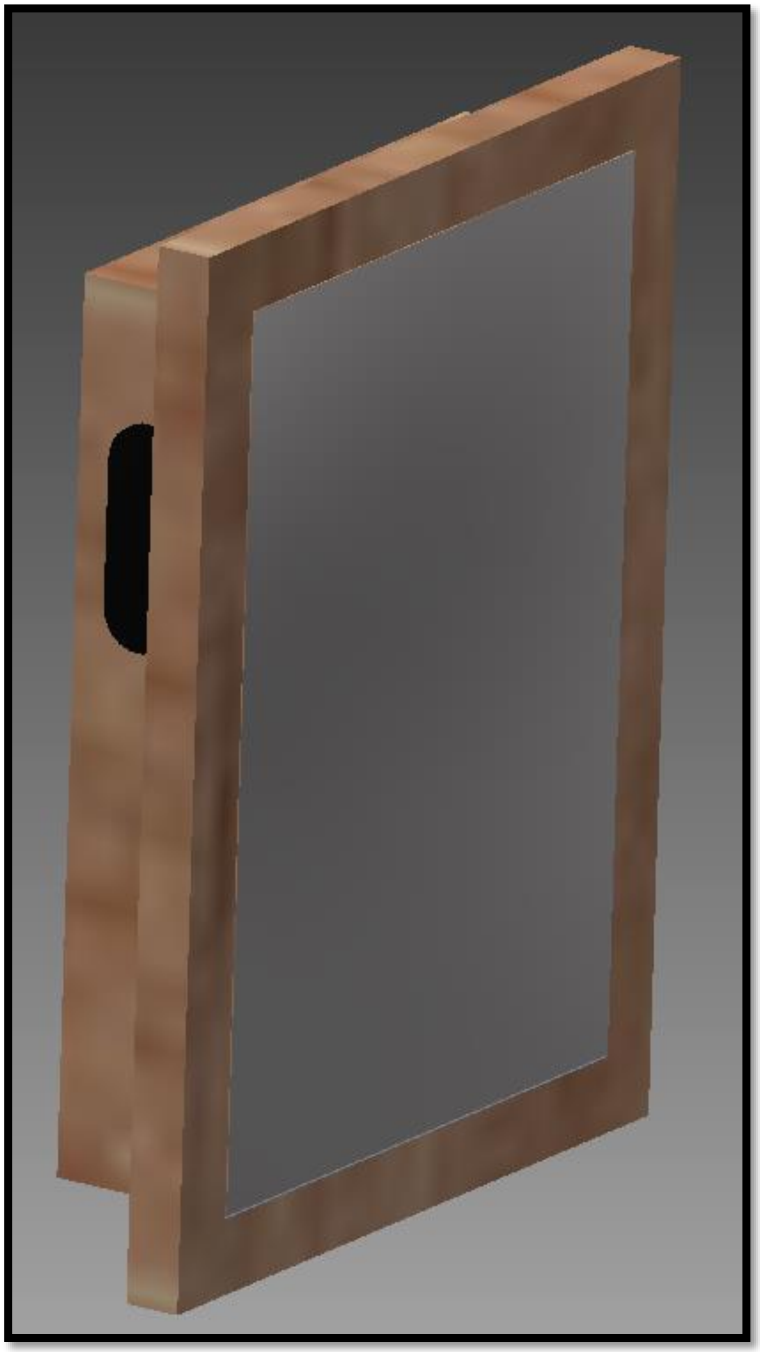


Figure 4.2.1.1: 3D Model of Mirror Housing Design

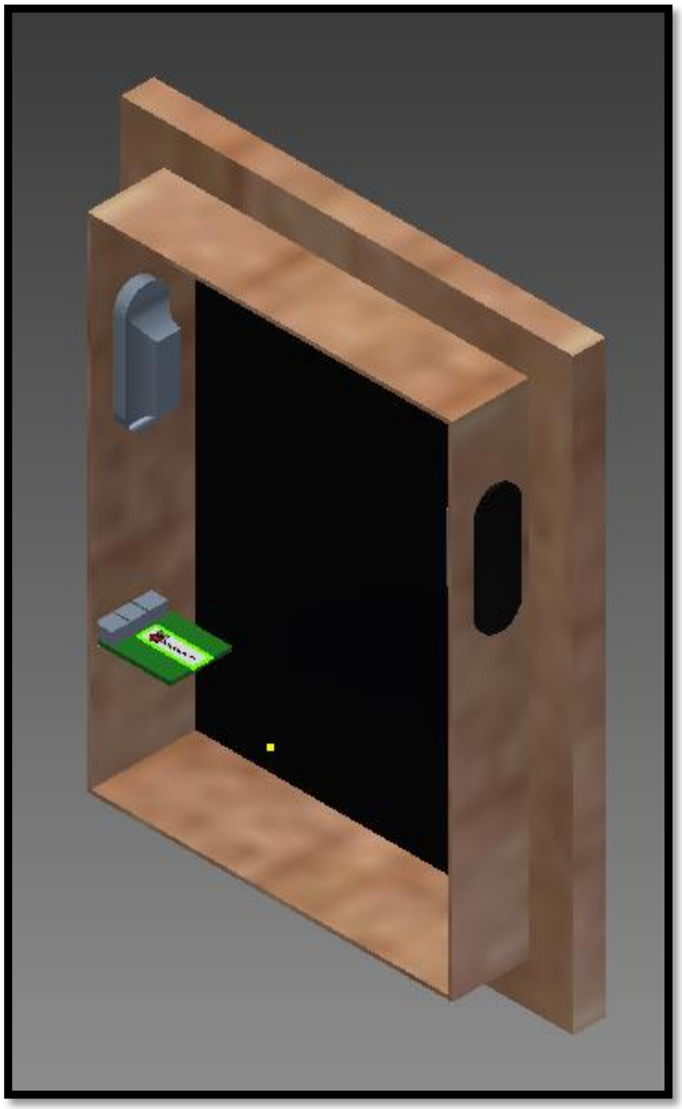


Figure 4.2.2.1: 3D Model of the Rear View of the Mirror Housing

The actual mirror user for the smart mirror project is an important aspect of the overall design. There are a number of options for exactly how to implement the mirror with a varying degree of price and functionality. There is a vendor who manufactures and sells one way mirrors specifically for this purpose, called Two Way Mirrors. Their mirrors are specialized for implementing televisions behind. The company offers a number of options at varying price points. Each option has different levels of light transmission and reflectivity as shown in Figure 4.2.2.2. While their mirrors would be a great option for implementation of a smart mirror, their mirrors are rather pricey and, for the intents and purposes of this project as a proof of concept, are out of budget. Two other options are presently available which are a regular one way mirror, or glass tinted with a special one way mirror tint. A regular one way mirror is somewhat costly, starting around $150 for a thirty two inch diagonal display. While this would be a better option, the price is again, somewhat out of budget for a proof of concept type project. Due to this, the project will be implemented using the final option, a pane of glass tinted with a specialized reflective tint. Though this option does not provide the highest level of clarity, reflectivity, or light transmission, it is the best decision for the goal this project is attempting to accomplish. If this project were to be taken a step farther to be implemented and marketed, the best option would likely be Two Way Mirrors Vanity Vision Glass which offers a seventy percent reflection and a forty percent light transmission.

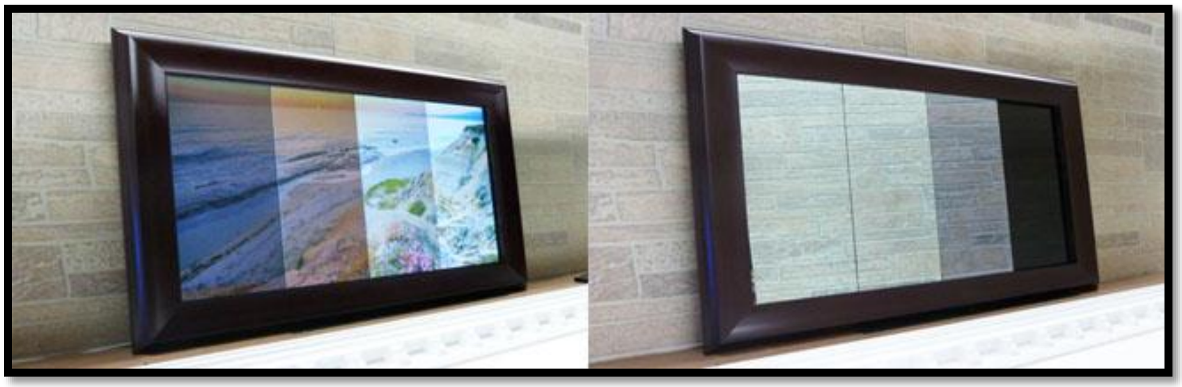


Figure 4.2.2.2: Specialized Two Way Glass (Left to right: Regular One Way mirror, VanityVision, Dielectric, Clear Glass) Permission Pending from Two Way Mirrors

In designing this product, one potential minor inconvenience is the necessity of power cables running into the mirror housing to power the television and the computer. In order to avoid having multiple cables running from the housing to an outlet, a C13 Power Connector will be set into the bottom of the mirror housing. From this C13 Power Connector, power will be run to the television as well as all elements of the computer and MCU. This will allow for a neat appearance of a single cable which plugs into the housing of the mirror.

4.3 Sensors

Probably the most important physical component for this experiment are going to be sensor. On this section we are going to break down all the different type of sensors that are going to be used for the Smart Mirror project. One of the possible locations for the Smart Mirror is going to be in a humid area, for example a bathroom, is very important to incorporate a humidity sensor in order to keep the electronics protected from any moisture. A compliment to the humidity sensor will be a temperature sensor, in order to prevent any component to be overheated inside the Smart Mirror, we are going to need to include way to measure the heat inside the mirror. A sensor that will help in term of conserving power being dissipated into the mirror is a light sensor. It’s important to incorporate a way to measure the light in the room to determine if the Mirror needs to be ON or OFF.

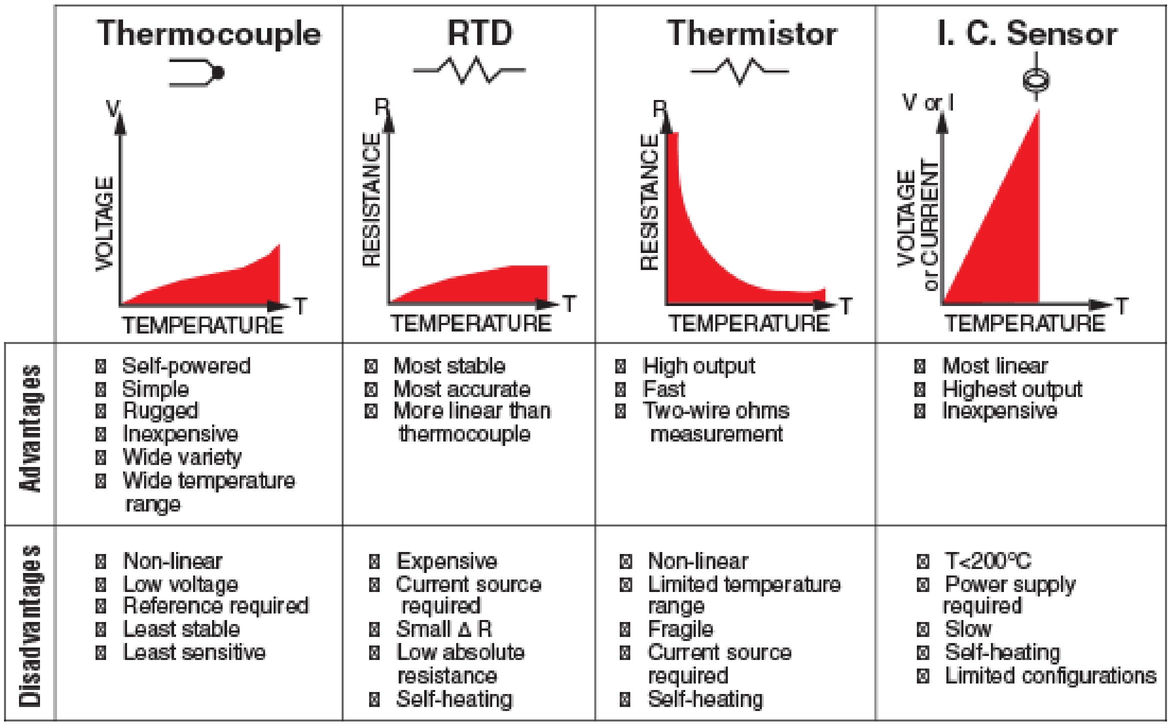
The compliment to the light sensor is the motion sensor, the mirror will need to determine whether there is a user present to turn ON or OFF. This section will focus on comparing relevant sensors for each of the task mentioned above.

4.3.1 Temperature Sensor

After the preliminary research, Table 4.3.1.1 was created to facilitate a comparison with key specifications that are important for the design of the final project. Besides Table 4.3.1.1, Figure 4.3.1.1 was found during the research and it will be used to select the component that will be used.

Type of Temperature Sensor				
	Thermocouple	RTD	Thermistor	I.C. Sensor
Temperature Range	-200C to 2000C	-200C to 600C	-75C to 260C	-40C to 125C
Interchange Ability	Good	Excellent	Poor to fair	Fair
Long-Term Stability	Poor to fair	Good	Poor to fair	Good
Accuracy	Medium	High	Medium	High
Repeatability	Poor to fair	Excellent	Fair to good	Excellent
Sensitivity (Output)	Low	Medium	Very high	High
Response	Medium to fast	Medium	Medium to fast	Fast
Linearity	Fair	Good	Poor to fair	Excellent
Self-Heating	No	Very low to low	High	Low
Point Sensitive	Excellent	Fair	Good	Excellent
Size Packaging	Small to large	Medium to Small	Small to medium	Small
Cost	Low	Low	Low	Low

Table 4.3.1.1: Comparison of Different Type of Temperature Sensor


VOLTAGE
V
T
TEMPERATURE
RESISTANCE
R
T
TEMPERATURE
RESISTANCE
R
T
TEMPERATURE
VOLTAGE or CURRENT
V or I
T
TEMPERATURE

	Thermocouple	RTD	Thermistor	I. C. Sensor
Advantages	⊠ Self-powered ⊠ Simple ⊠ Rugged ⊠ Inexpensive ⊠ Wide variety ⊠ Wide temperature range	⊠ Most stable ⊠ Most accurate ⊠ More linear than thermocouple	⊠ High output ⊠ Fast ⊠ Two-wire ohms measurement	⊠ Most linear ⊠ Highest output ⊠ Inexpensive
Disadvantages	⊠ Non-linear ⊠ Low voltage ⊠ Reference required ⊠ Least stable ⊠ Least sensitive	⊠ Expensive ⊠ Current source required ⊠ Small Δ R ⊠ Low absolute resistance ⊠ Self-heating	⊠ Non-linear ⊠ Limited temperature range ⊠ Fragile ⊠ Current source required ⊠ Self-heating	⊠ T<200°C ⊠ Power supply required ⊠ Slow ⊠ Self-heating ⊠ Limited configurations

Figure 4.3.1.1: Linear Comparison of Temperature Sensors - Recreated with data provided from Omega

After using Table 4.3.1.1 and Figure 4.3.1.1 to compare the four types of temperature sensors, it was determine that the integrated circuit sensor will be implemented in the smart mirror project. This sensor is the simplest to implement since it has the best linearity and it does not need any other circuitry to analyses the data. Since Integrated Circuit Sensors will be picked for the project, 3 individual sensors will be compared for the design of the project; LM35, TMP36, and TMP102. Table 4.3.1-2 compares all 3 components in multiple parameters.

The next parameter that will be compared will be the thermal response in still air, Figure 4.3.1.2 shows the percentage change of the final value changes over 8 minutes after activation for the LM35 Sensor. It can be determine that after 3 minutes the LM35 sensor reaches 100%.

Figure 4.3.1-3 illustrates the thermal response in still air for the TM36 temperature sensor. After inspecting the graph we can determine that the smaller the size of the PCB, the faster the response. We can see that comparing the smallest PCB with the response time of the LM35, it can determine that the TM36 sensor is a little slower than the LM35 by reaching 100 percent at 200 seconds (3 minutes and 20 seconds).

	LM35 Series	TMP36	TMP102
Output Type	Analog	Analog	Digital
Temperature Range	-55 C to 150 C	-40 C to 125 C	-40 C to 125 C
Accuracy (+/-)	0.5 C	1 C	0.5 C
Operating Temperature	-55 C to 150 C	-40 C to 150 C	-55 C to 150 C
Linear Temperature Slope	10 mV/ C	10 mV/ C	-
Supply Current	114 uA	50 uA (max)	-
Supply Voltage	4 V to 5.5 V	2.7 V to 5.5 V	1.4 V to 3.6 V
Output Voltage Range	-1V to 6 V	100 mV to 2 V	0.2 V to 4.0 V
Impedance Output	0.1 Ohm	Low	-
Sensor Gain	+10	-	-
Line Regulation (+/-)	0.02 C	0.5 C	-
Response Time	Fast	Medium	Fast
Long Term Stability (+/-)	0.08 C	0.4 C	-
Self-Heating	Low	Low	Low
Cost	$1.23	$1.77	$5.95

Table 4.3.1.2: Comparison of Temperature Sensors

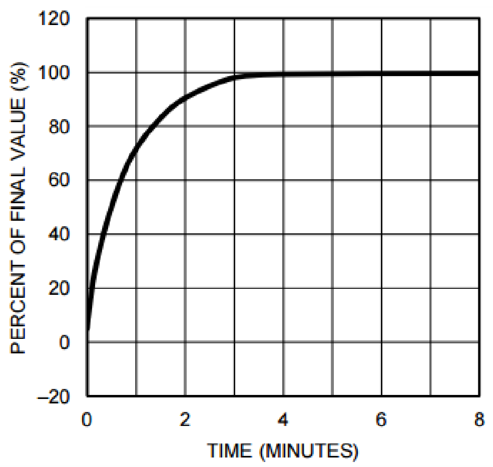


Figure 4.3.1.2: LM35 Percent of Final Value - Recreated with data provided from the datasheet

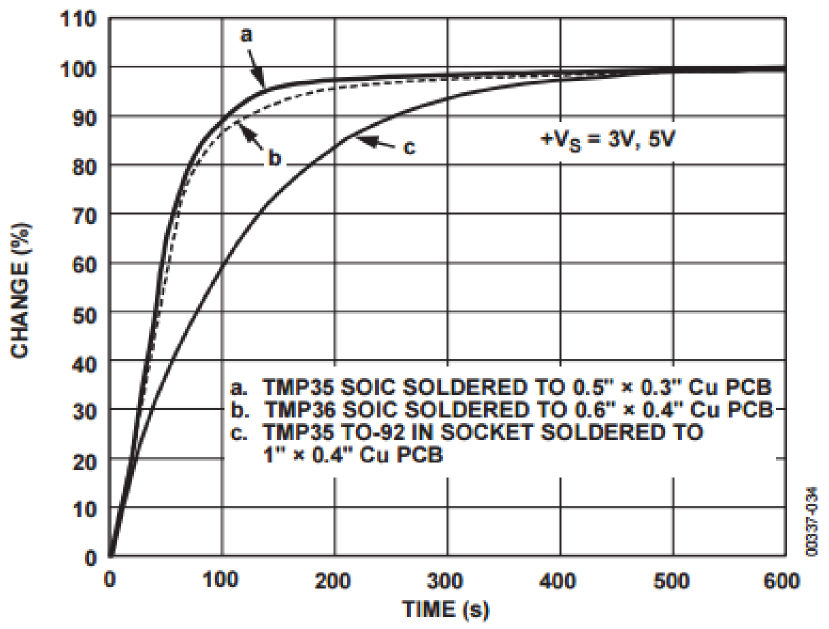


Figure 4.3.1.3: TM36 Change Response with Time - Recreated with data provided from the datasheet

Figure 4.3.1.4 shows the conversion time as the temperature increases of the TMP102 sensor. This graph is different from the previous one since this sensor's output is digital. But we can see the conversion time from -40 C to 140 C, when applying a 3.6 V supply, is about 1 microsecond.

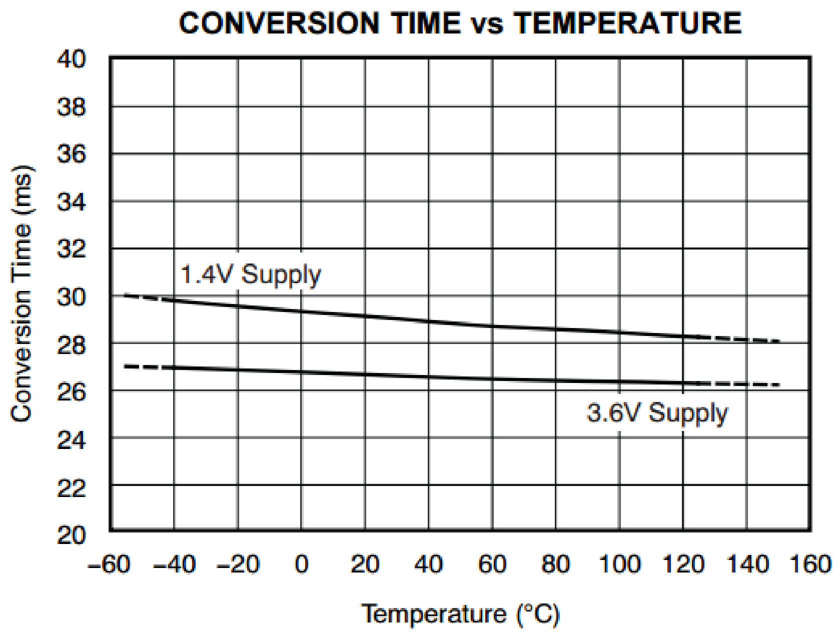


Figure 4.3.1.4: TMP102 Conversion Time Depending on Temperature - Recreated with data provided from the datasheet

After comparing all 3 graph, we can determine that the LM36 sensor has the quickest thermal response time in still air. The next parameter that will be compared will be the temperature accuracy. Figure 4.3.1-5 displays the temperature accuracy for the LM35 sensor, as well as Figure 4.3.1-6 for the TM36 sensor and Figure 4.3.1-7 for the TMP102 sensors. We can see that the temperature error are very small for all of them but the TM36 is more accurate.

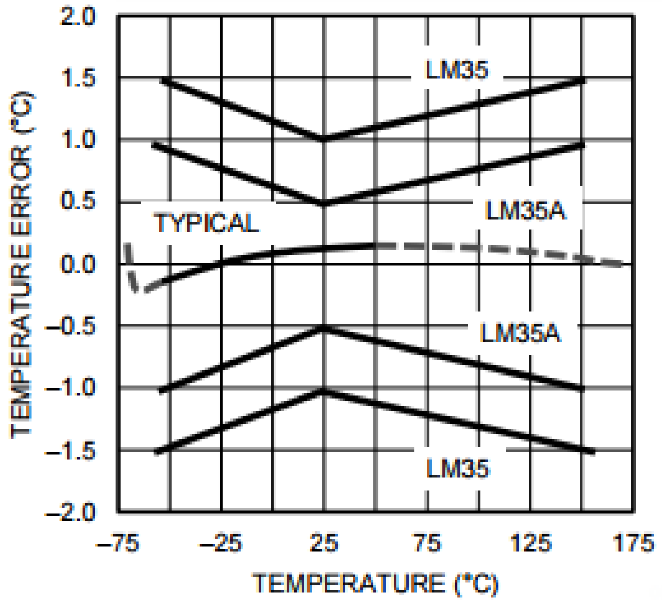


Figure 4.3.1.5: LM35 Error Depending on Temperature - Recreated with data provided from the datasheet

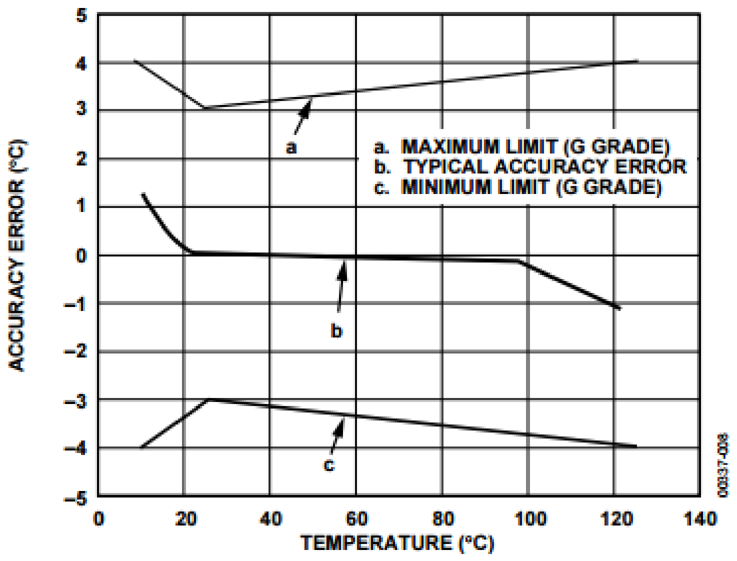


Figure 4.3.1.6: TM36 Error With Respect to Temperature - Recreated with data provided from the datasheet

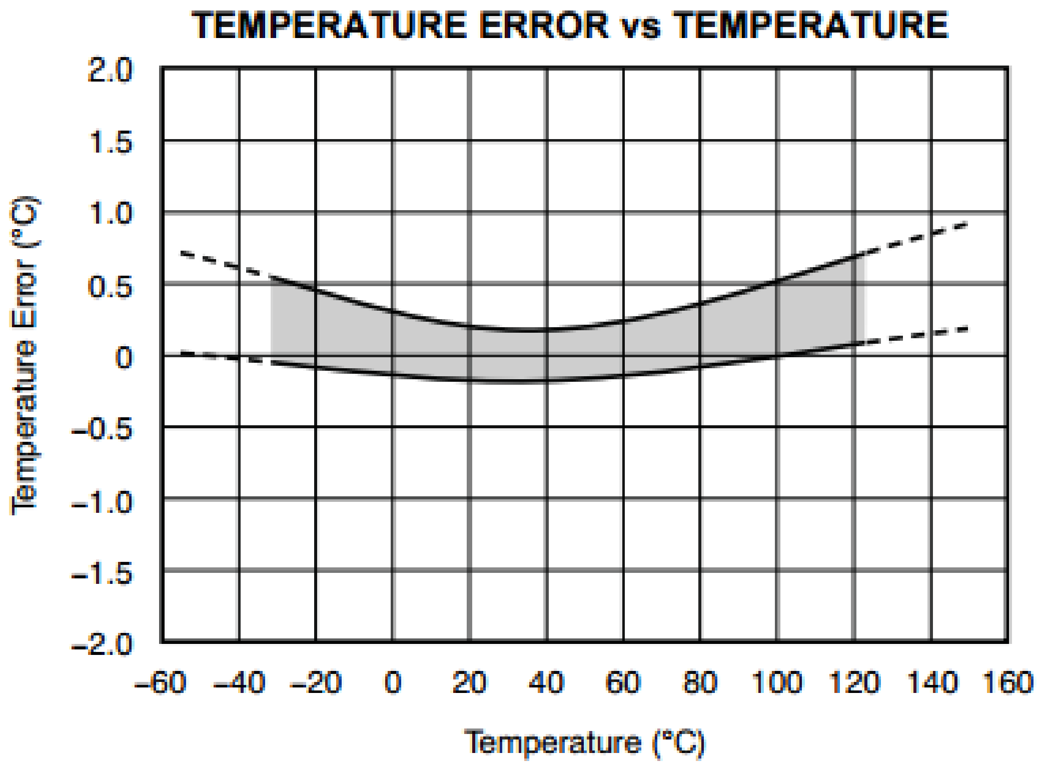


Figure 4.3.1.7: TMP102 Temperature Error with respect to Temperature - Recreated with data provided from the datasheet

After reviewing all the data, we decided to use LM35 for the design of the project since it has the best thermal response time and is not dependent of the size of the Printed Circuit Board as well as its accuracy. The TMP102 sensor was not considered since it the cost of the component is about 4 times more than the LM35.

For the Temperature Controller in the Smart Mirror we are going to need a voltage comparator to determine if the fan will turn ON in order to keep the internal temperature of the system in the operational range. In to implement this comparator we are going to need the temperature sensor, one Operational Amplifier, one NPN Bipolar Junction Transistor, one fan, three Diodes, and six Resistors. The components will needed to implement a Schmitt Trigger to evaluate the voltage. If the voltage goes reaches the upper bound of the Hysteresis Window, it will turn the fan on, acting as a switch. Figure 4.3.1.7 illustrates the schematic designed for the temperature control, and Figure 4.3.1.8 illustrates the Layout for the Printed Circuit Board (PCB). While designing the Printed Circuit Board Layout, Surfer Mounted components were not being considered, since this will facilitate the prototyping portion of the project.

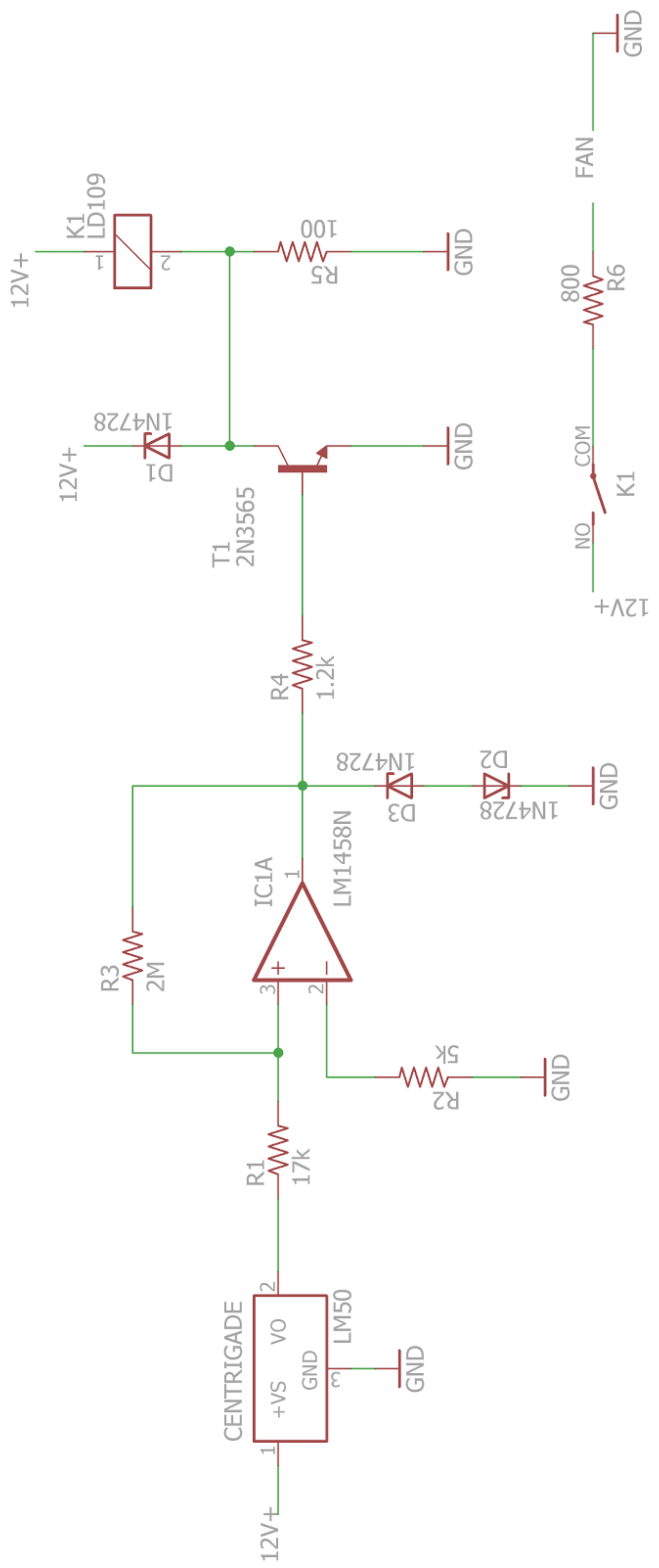


Figure 4.3.1.7: Temperature Control Schematic

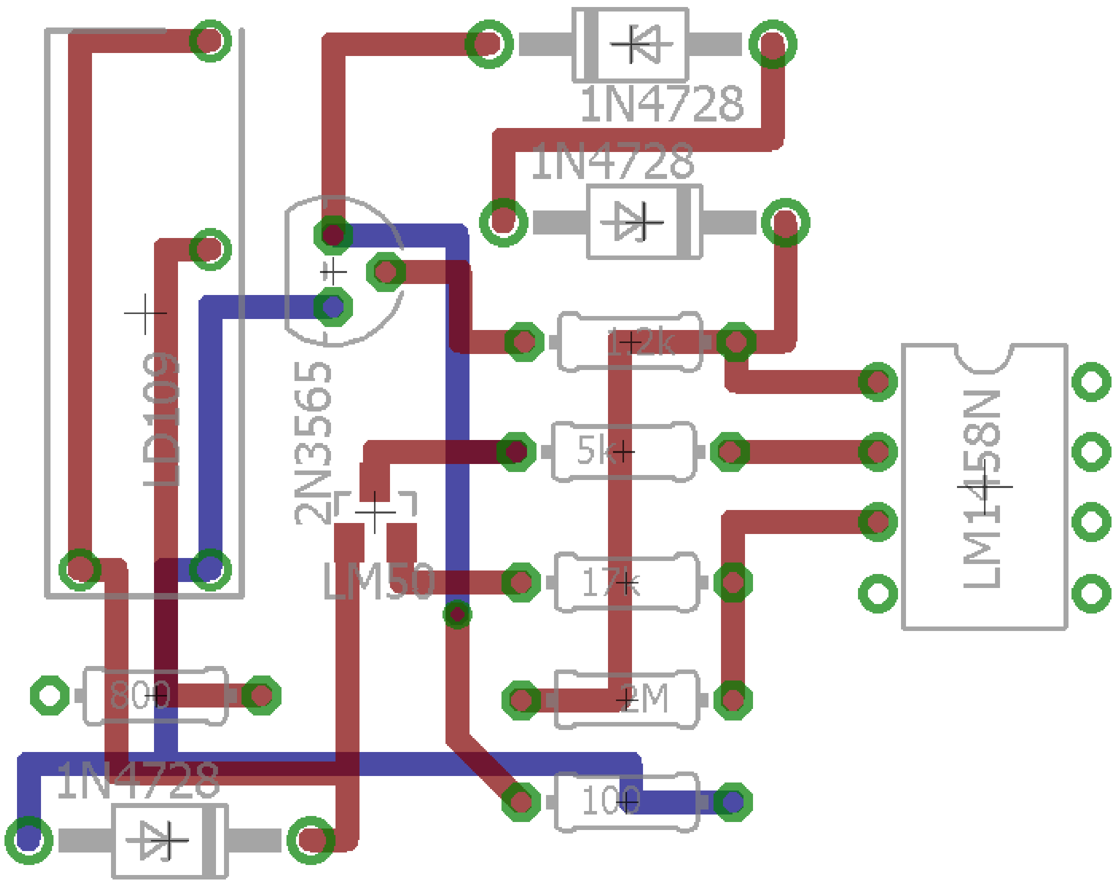


Figure 4.3.1.8: Printed Circuit Board (PCB) Layout

4.3.2 Motion Sensor

Since we are not looking for a lot of precision on the Motion sensor we have decided to select a motion sensor that can detect motion in a range of up to 150 cm, which is around 5 feet away from the mirror. While searching for a motion sensor it was determine that those sensors that had an analog output had a distance range with a longer range. Table 4.3.2.1 compares four sensors that meet the range of the distance that we are interested in: GP2Y0A02YK0F, GP2Y0A60SZxF, GP2Y0A700K0F, and GP2Y3A003K0F. It's important to note that sensors GP2Y0A700K0F and GP2Y3A003K0F are not in stock while the research portion of the paper was being completed. When comparing the four sensor using the table we can point out that they share the same range for the Operating Temperature as well as the Recommended Supply Voltage needed for proper operation.

	GP2Y0A02YK0F	GP2Y0A60SZxF	GP2Y0A700K0F	GP2Y3A003K0F
Output Type	Analog	Analog	Analog	Analog
Distance Range	20 cm to 150 cm	10 cm to 150 cm	100 cm to 550 cm	40 cm to 300 cm
Supply Voltage	-0.3 V to 7.0 V	-0.3 V to 5.5 V	-0.3 V to 7.0 V	-0.3 V to 7.0 V
Maximum Supply Current	50 mA	50 mA	50 mA	50 mA
Voltage Difference at a Distance	2.05V at Range	1.6V at Range	1.6V at Range	1.6V at 40C to 100C
Operational Temperature	-10C to 60C	-10C to 60C	-10C to 60C	-10C to 60C
Typical Output Voltage	.4V	.35V	2.7V	2.3V
Cost	$14.95	$8.94	Not Available	Not Available

Table 4.3.2.1: Motion Sensor Comparison

The following figures will compare the relationship of the output voltage that each one of the sensors produce depending on the distance from the reflective object. After comparing all four figures we can see that as the object moves away from the sensor, the lower the output voltage becomes. The motion sensor that would work better for our project would be the GP2Y0A700K0F, but at the moment this sensor is not available, therefore the second option for our project would need to be the GP2Y0A60SZxF since it's more cost effective and we could provide a smaller voltage supply.

The following figure corresponds to the GP2Y0A02YK sensor.

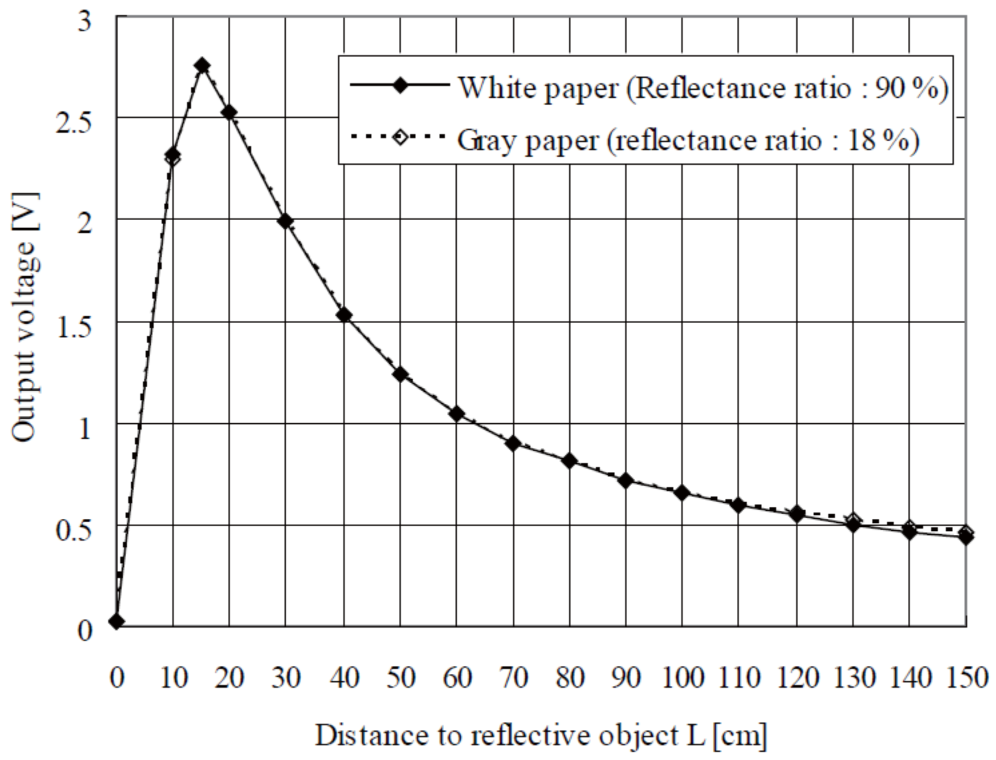


Figure 4.3.2.1: GP2Y0A02YK Sensor Output Voltage vs Distance to Object

The following figure corresponds to the GP2Y0A60SZ sensor

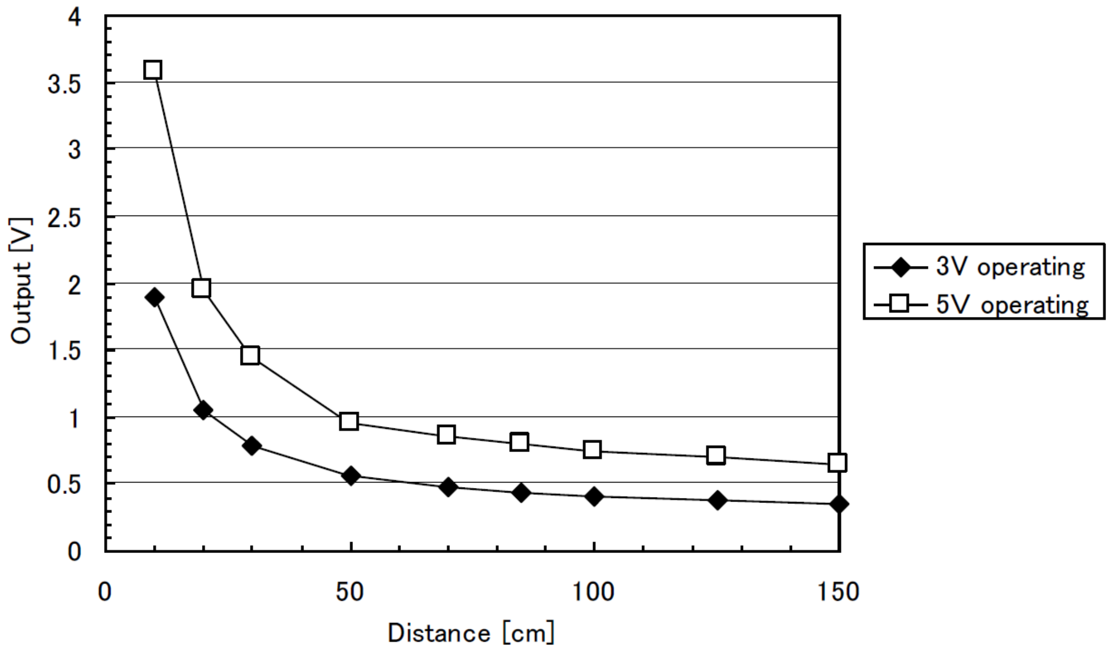


Figure 4.3.2.2: GP2Y0A60SZ Sensor Output Voltage vs Distance to Object

The following figure corresponds to the GP2Y0A700K0F sensor.

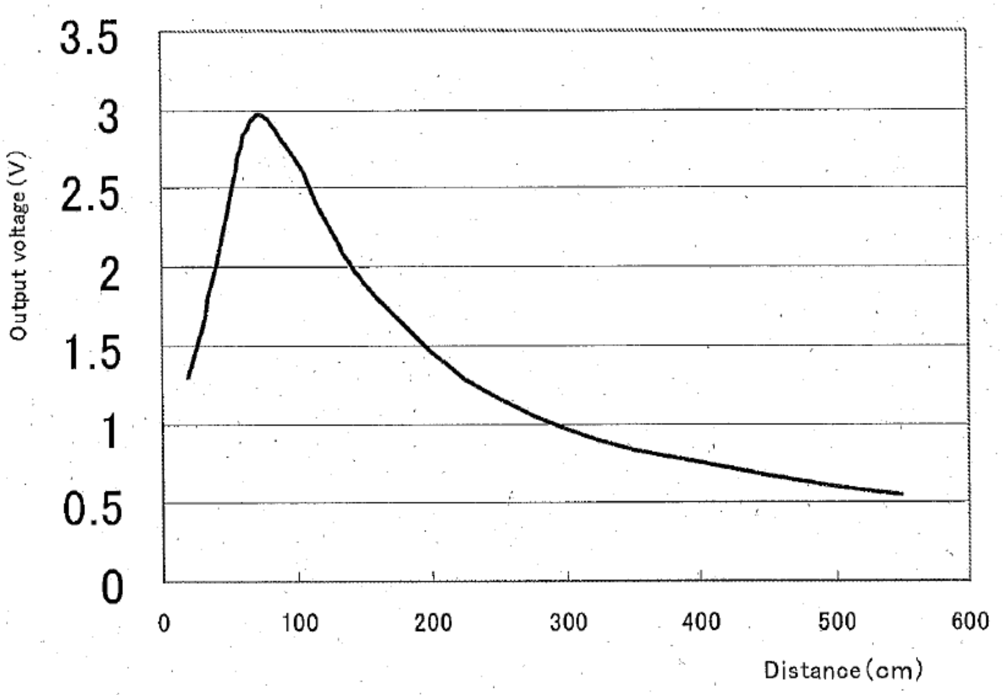


Figure 4.3.2.3: GP2Y0A700K0F Sensor Output Voltage vs Distance to Object

The following figure corresponds to the GP2Y3A003K sensor.

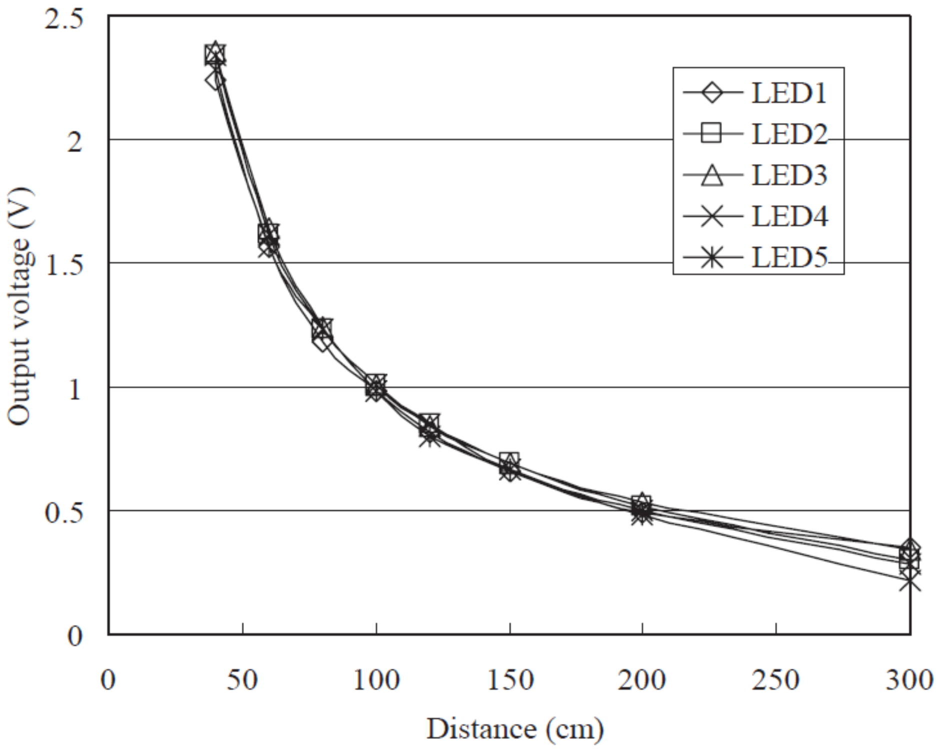


Figure 4.3.2.4: GP2Y3A003K Sensor Output Voltage vs Distance to Object

4.3.3 Light Sensor

After reviewing the different type of Light sensor is was determine that a highly precise sensor will not be needed, since the primary purpose of the sensor will be to determine if there is light in the room, if so it will “turn on” the mirror. The search was narrowed down to a photodiode and a phototransistor. Table 4.3.3.1 compares 3 commonly used sensors: Gl5528, NORP12, and TEMT6000. After reviewing the temperature, it was determine that the GL5528 sensor will be chosen for the design of our project since it has a low power dissipation and it has the lowest cost.

	GL5528	NORP12	TEMT6000
Light Resistance at 10 Lux (25 C)	8 kohms to 20 kohms	5.4 to 12.6 kohms	-
Collector Light Current	-	-	16 uA
Dark Resistance at 0 Lux	1.0 Mohms	1.0 Mohms	-
Collector Dark Current	-	-	50 nA
Power Dissipation (25C)	100mW	250mW	100mW
Max Voltage (25C)	150 V	250V	-
Operating Temperature	-30 C to +70 C	-60C to 75C	-40C to 85C
Spectral Response Peak (25C)	540 nm	550 nm	570 nm
Cost	$1.50	$6.66	$4.95

Table 4.3.3.1: Light Sensor Comparison

4.3.4 Humidity Sensor

After researching the different type of humidity sensors, we can compare the capacitive and resistive sensor, since the thermal conductivity humidity sensors will be more precise without the need to be and it will be more expensive than the Relative Humidity. Table 4.3.4.1 compares the Capacitive and Resistive Humidity sensors.

Type of Humidity Sensors			
	Capacitive	Resistive	Thermal Conductive
Cost	Low	Low	High
Size	Small	Small	Small
Accuracy (+/-)	1% RH	2% RH	+/- 5% RH at 40 C and +/- 0.5% at 100 C
Long-term Stability	Yes	Yes	Yes
Interchangeable	Yes (only if laser trimmed)	Yes	No
Response Time for a 63% RH step Change	30 to 60 seconds	10 to 30 seconds	20 seconds
Operating Temperature	Up to 200 C	-40C to 140C	Up to 300 C
Uncertainty (from 5% to 95% RH with two-point calibration)	+/- 2% RH	+/- 1 to 2 %	-
Linearity	Yes	No	-
Calibration Needed	Yes (Computer Based)	No	Yes
Sensitivity (for every 1% change in RH %)	0.2 to 0.5	4pf	-
Humidity Range	0 to 100% RH	5 to 95% RH	-
Resistance to Chemical Vapors	Reasonable	No	Yes
Resistance to temperature fluctuations	Yes	No	Yes

Table 4.3.4.1: Humidity Sensor Comparison

After reviewing previous projects and researching online, the most frequently used sensors for humidity are the innovative sensor technology sensors P-14 and MK33. Table 4.3.4.2 compares these two sensors.

Component Comparison		
	P-14	MK33
High Chemical Resistance	Yes	Yes
Humidity Operating Range	0 to 100% RH	0 to 100% RH
Operating Temperature Range	-50 C to 150 C	-40 C to 190 C
Capacitance	150 pF +/- 50 pF	300 pF +/- 40 pF
Sensitivity per RH%	0.25 pf	0.45 pf
Loss Factor	<0.01	<0.01
Linearity error	< 1.5% RH	< 2.0% RH
Hysteresis	< 1.5% RH%	< 2.0% RH%
Response Time	< 5 seconds	< 6 seconds
Frequency Range (KHz)	1 - 100	1 - 100
Maximum Operating Voltage	< 12 Vpp	< 12 Vpp

Table 4.3.4.2: Comparison between P-14 and MK33.

4.3.5 Temperature Controller

For the Temperature Controller in the Smart Mirror we are going to need a voltage comparator to determine if the fan will turn ON in order to keep the internal temperature of the system in the operational range. In to implement this comparator we are going to need the temperature sensor, one Operational Amplifier, one NPN Bipolar Junction Transistor, one fan, three Diodes, and six Resistors. The components will needed to implement a Schmitt Trigger to evaluate the voltage. If the voltage goes reaches the upper bound of the Hysteresis Window, it will turn the fan on, acting as a switch. Figures X illustrates the schematic designed for the temperature control, and Figure X illustrates the Layout for the Printed Circuit Board (PCB). While designing the Printed Circuit Board Layout, Surfer Mounted components were not being considered, since this will facilitate the prototyping portion of the project.

4.4 MCU

After researching microcontrollers and comparing previous projects regarding similar projects, the 3 microcontrollers that will be considered for this project will be the Arduino Uno, Arduino Due, and TI MSP430. While researching the microcontrollers units, it was difficult to find information on the Texas Instruments website. Table 4.4.1 compares the mentioned 3 microcontrollers.

	Arduino Uno	TI MSP 430	Arduino Due
Microcontroller	ATmega328P	MSP430FR5969	AT91SAM3X8E
Operating Voltage	5 V	1.8 V to 3.6 V	3.3 V
Input Voltage	7 to 12 V	-	7 to 12 V
Digital I/O Pins	14	-	54
PWM Digital i/O Pins	6	-	12
Analog Pins	6	15	14
DC Current Per I/O Pin	20 mA	+/- 2 mA	130 mA
DC Current for 3.3 V Pin	50 mA	-	800 mA
Flash Memory	32 KB	32 KB	512 KB
SRAM	2 KB	2 KB	96 KB
EEPROM	1 KB	-	-
Clock Speed	16 MHz	16 MHz	84 MHz

Table 4.4.1: Microcontroller Comparison

While researching the different type of microcontrollers, it was determine that the Arduino Uno was the best fit for our project. The Arduino Uno has a large open-sourced libraries, this will make the development process of the final product much easier. Besides the open-sourced, Arduino has multiple products that can be combined with the standard board, this will make the prototyping of the project simpler. Arduino also has a massive amount of community support online, it’s common to find threads of question and answers for common issues that arise when doing projects using the Arduino boards. Some of these threads include solutions for troubleshooting the board, and can even become more specific with ways to fix common errors made while using the I2C, SPI, and UART interfaces.

Part of the requirements for the project is to create a schematic that will be used for the final product. It was decided to create a PCB of the Microcontroller Unit along with voltage regulators and components needed to run the MCU at the desired clock rate since the sensors will be communicating with it and it’s necessary for it to be costumed. The layout used for this printed circuit board was also obtained from Arduino. Arduino is open source meaning their schematics and boards are available to the public, making prototyping and customizing of the board with the sensors more accessible.
The microprocessor that was chosen was from ATMEL, ATMega328P. Several other models of ATMEL well also compared as well as microprocessors from other providers such as Texas Instruments, and Raspberry Pi.
The specifications and reason why this processors was chosen are described in Table 4.4.2 below along with pictures of the schematics and board layouts of the Arduino board with the ATMEL processor.

	ATmega328P
Flash	32 KB
CPU	8-bit AVR
Max Frequency	20 MHz
Max I/O Pins	23
ADC Channels	8
ADC Resolution	10 bit
ADC Speed	15 ksps
Analog Comparators	1
I/O Supply Class	1.8 to 5.5
PWM Channels	6
SPI	2
TWI (I2C)	1
UART	1

Table 4.4.2: ATmega328P Components

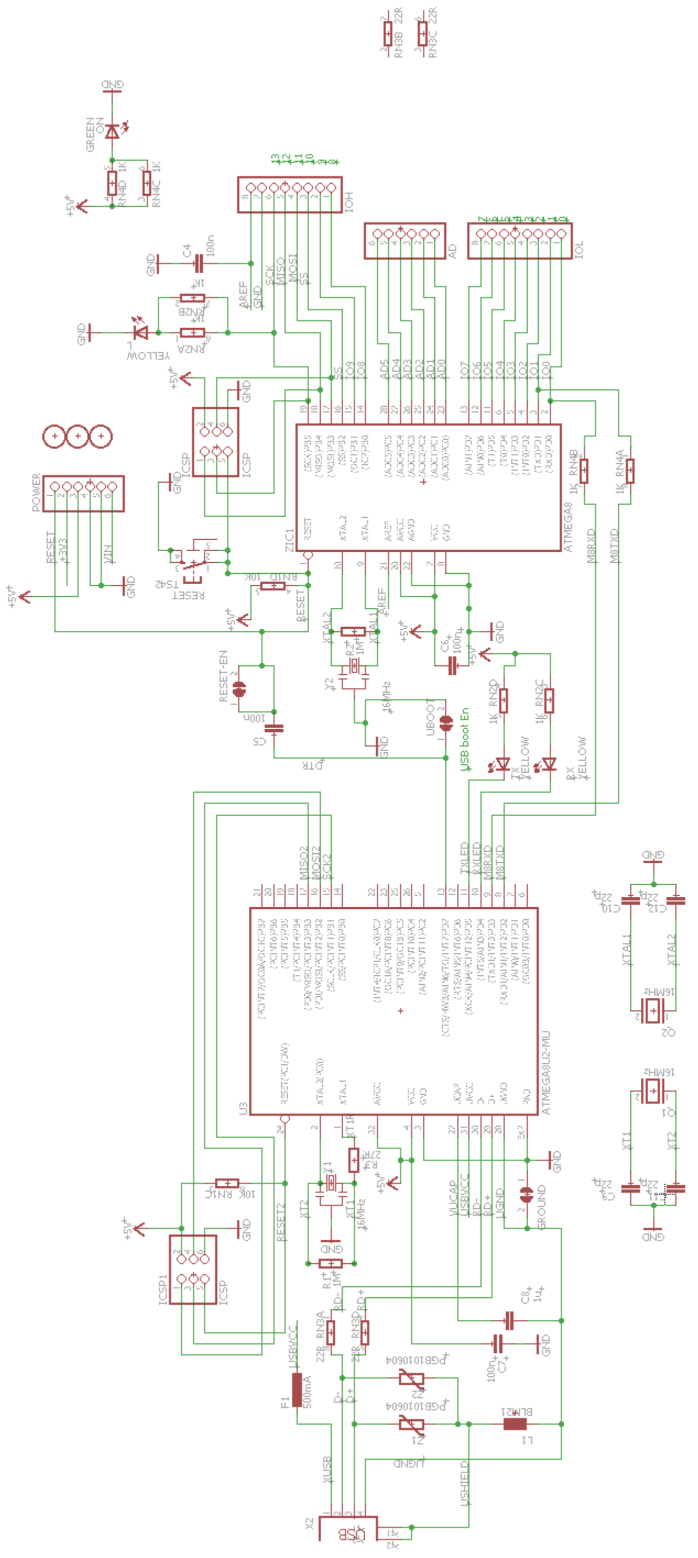


Figure 4.4.1: Board Layout of Arduino Board

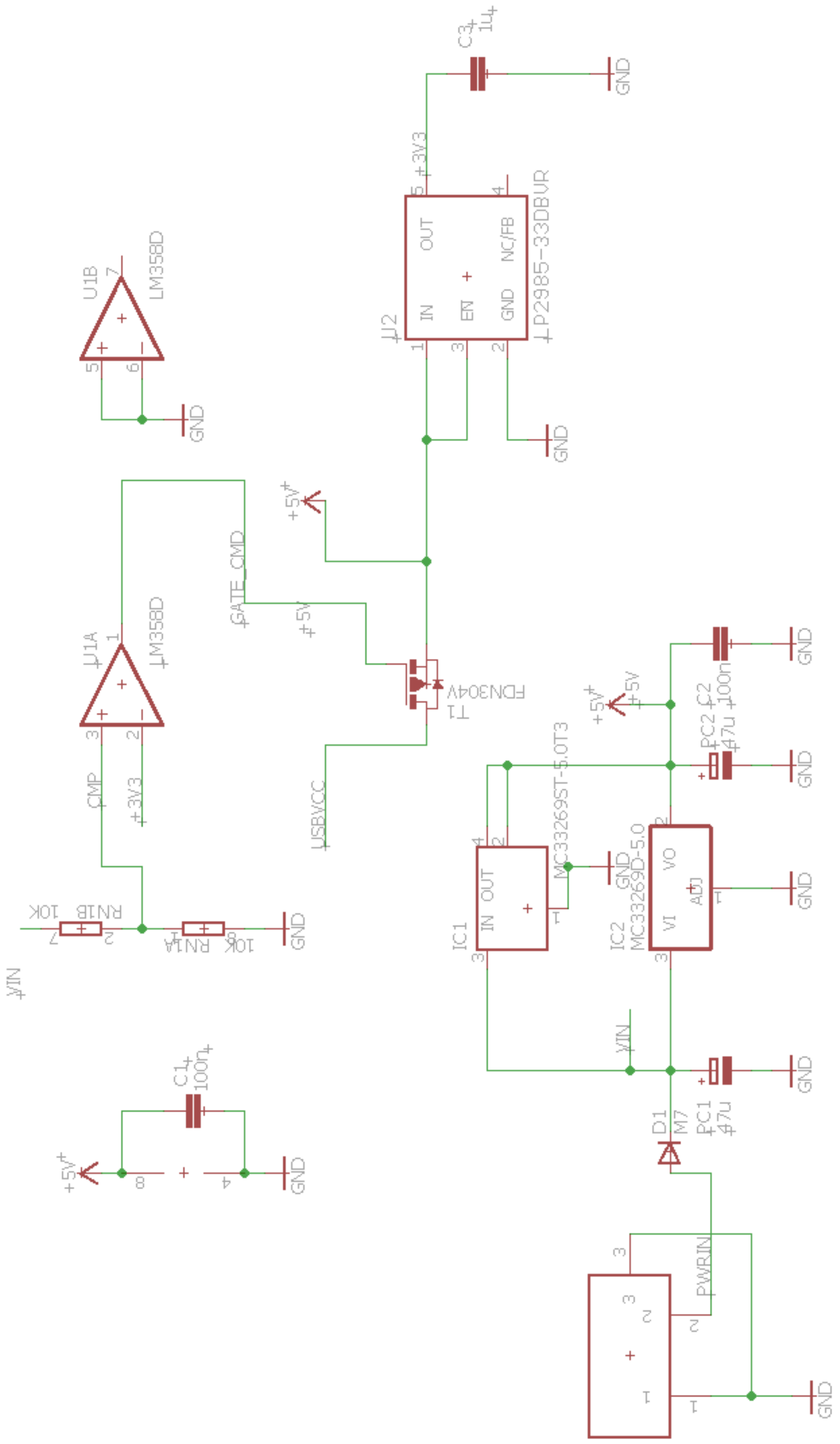


Figure 4.4.2: Schematic of Arduino Board

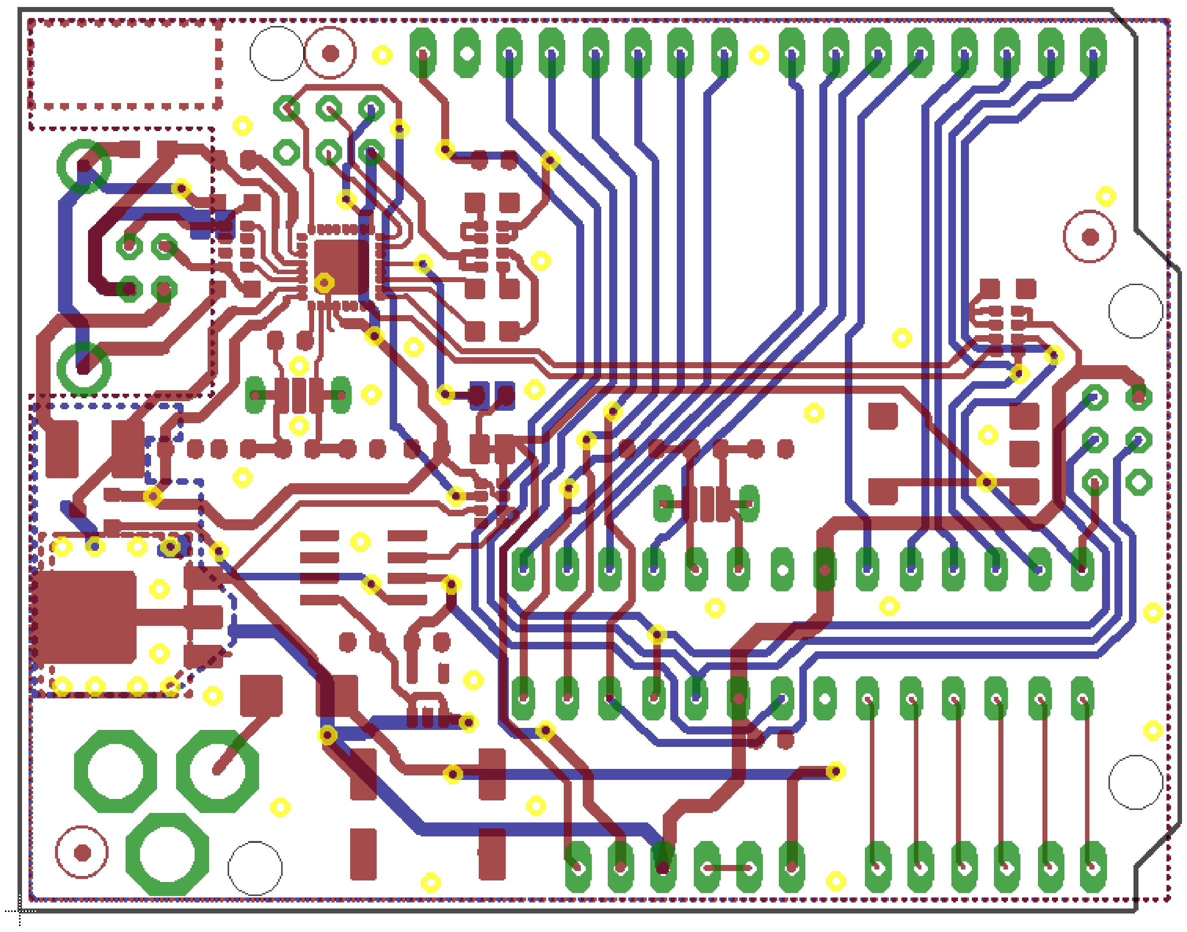


Figure 4.4.3: Printed Circuit Board Layout of Arduino Board

4.5 Audio System

The audio system on the Smart Mirror needs to include audio input as well as audio output. The audio input system will be utilized to recognize any commands that the user will instruct via voice control. While the audio output system will provide the user with any information they may want such as music, current temperature and number of events scheduled for the day.

4.5.1 Audio In

For the audio in, it has been decided to utilize a small, USB powered microphone which will receive all voice commands the user gives. The USB microphone will be selected based upon cost and quality of the unit. The unit needs to record quality sound but at the same time, needs to remain in budget. The microphone will be mounted into the face of the smart mirror to receive the best possible audio signal to ensure the voice commands are clear.

4.5.2 Audio Out

For an audio out system, there will be two possible options first is to utilize speakers within the housing of the mirror while the second is to utilize external speakers. The speakers within the housing may be connected to the television or the Raspberry Pi itself. The television purchased for this project has built in speakers which will be accommodated into the sides of the mirror. This will ensure that the audio is projected away from the microphone in an attempt to interfere as little as possible. External speakers could be utilized via a Bluetooth USB dongle for the Raspberry Pi however that could potentially cause interference with the microphone if the speaker’s face towards the mirror. Though all options will be supported, it will be encouraged for the speakers housed within the mirror to be utilized.

4.6 Power

For power for the smart mirror system, it will be ensured that the user only has one power cable that needs to connect to the wall. Unfortunately a battery is unfeasible for an item of this size that will be permanently affixed in a household. This requires a single power cable at the least. The 120V AC power will be brought into the mirror housing utilizing a C13 Connector. The power will then be split into two lines, one running directly to the television with the other intending to power the Raspberry Pi and the MCU. The power to the Raspberry Pi and MCU will be required to be converted to a 5V DC signal for the Pi and a 3.3V DC signal for the MCU. To do this, a power converter will be purchased to convert from 120V AC to 5V DC power and it will be implemented within the mirror. From there, power will be converted to a microUSB to power the Pi. Finally, power will be utilized from the 5V converter and passed through a step down circuit to step the power down to 3.3V for the MCU. This will provide power to all the necessary MCU systems including all the sensors.

5. Software Design

5.1 Design Discussion

Our user interface design will be simple and minimalistic, displaying the various bits of information in an elegant fashion. We will utilize the screen real estate to display the information in an optimal orientation that does not hinder the function of the mirror, such as in the top corners of the display. The font, text size, and color should be appropriate for its intended use location, accounting for the lighting conditions of the room in which it operates. For aesthetic purposes, a monochromatic theme is desirable but could become difficult to read in bright environments.

Due to implementation of the software on Windows 10 IoT using the Universal Windows Platform (UWP), the project will be constructed utilizing the standard UWP design architecture; MVVM. The MVVM, Model-view-viewmodel architecture is derived from the basic model-view-controller (MVC) pattern. One of the core ideas of the MVVM model is separation of development of the graphical user interface (GUI) and the the back end logic and data implementation. The 'View' component refers to the presentation layer and specifies the user interface. The 'Model' component of MVVM is the code that handles all logic and data required by the system. The 'View Model" component is used to convert information from the Model to a format presentable in the GUI then bind it to the view.

The majority of the program will be written utilizing the .NET framework and C# for the model layer of MVVM. UWP programs generally utilize XAML to describe the View with the thin view model layer behind it being written in the same language as the model, in this case C#. Due to the use of a number of web APIs for a majority of the applications, other languages will need to potentially need to be utilized implement RESTful services or other web based services.

The general display status and layout of the applications on the mirror display will be controlled by a single class which will hold the current state of the mirror at all times. This class, MirrorState will have a number of strings and booleans to indicate what is currently active as well as where it should be located on the display. It will also include information about the status of each application such was whether or not it is currently focused or running some specific functionality for the user. This class will be the heart of the application and will essentially manage everything going on. All sensor information will be fed into this class as well to be sorted out so action may be taken. An overall block diagram of the architecture of the application is shown in Figure 5.1.1.

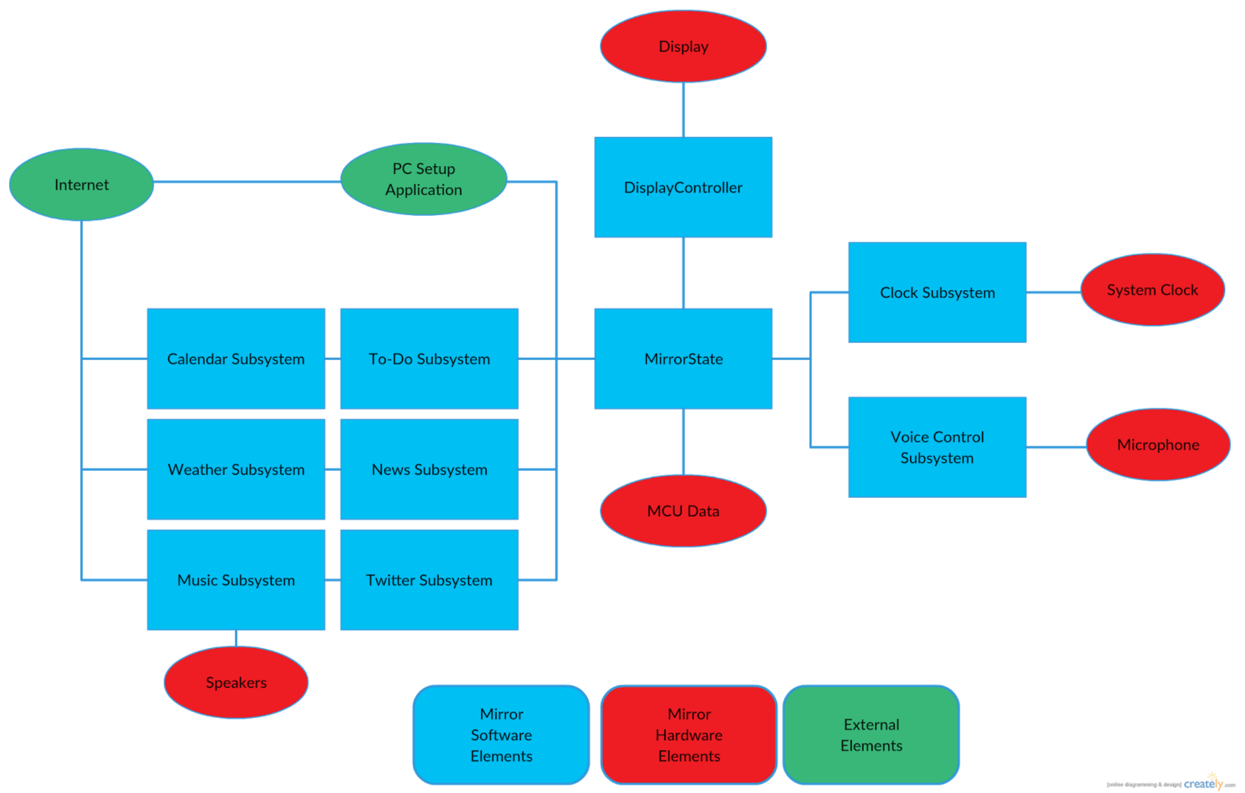


Figure 5.1.1: Application Block Diagram

5.2 Voice Control

5.2.1 Requirements & Constraints

The voice controller is the primary component used to interact with the smart mirror. Due to this, its implementation details are vital to the function of the mirror overall. The requirements of the mirror's voice commands need to be thoroughly considered and tested to ensure that they work as desired in an efficient, user friendly manner. Each command should be implemented in a number of ways to account for different types of speech and phrases used. This is to ensure that each user finds interaction with the mirror intuitive and unencumbered. To give a set list of commands required for each application will simply have users needing to continuously refer to the list of commands which will simply bog the user down in minutia making the mirror ineffective in its use of saving time and providing easy access to necessary information.

It is required that each application have a specific set of voice commands which can be listed to the user depending on the context of the mirror currently. This list will show the general commands the user may use however it should be possible for the user to say things similar to these commands to the same effect. With a relatively large set of voice commands to achieve this it will be necessary for some to be active at certain times and inactive at other times. The reason for this is due to potential errors in the voice recognition system if the user is unclear about their command or the audio isn't good enough. While the voice recognition software may still be confused by an unclear phrase it is better for nothing to happen than for an erroneous action to occur. Unloading certain grammar files based upon the current state of the mirror can alleviate this issue to some extent. For example, when no music is playing, then commands to change the volume, pause the music, or skip a song are entirely unnecessary and simply offer more phrases for the voice recognition software to be confused by. Setting the system up this way will offer great benefits but will require in depth testing to ensure all operations that should be possible are at any given moment while the only commands disabled are those that make little sense in the current context.

The primary constraint with a voice recognition system is the unfortunate issue with accents. Users with a heavy accent may have a hard time using the mirror and there is honestly little that can be done about this issue aside from within the voice recognition software itself. Since the voice recognition software for this project is from an outside source it will simply have to be taken as is with hopes that this issue is accounted for to some extent. Another constraint placed upon the system by utilizing voice recognition is a language barrier. This system will only be able to be utilized by people who speak English in its current state. That being said, Microsoft Speech Recognition does offer support for different languages thus it would simply be a matter of converting all text on screen as well as all commands in the grammar file to support other regions.

5.2.2 Error Handling

Utilization of voice recognition software will likely always come with some errors. Whether it be in the form of invalid commands, unclear speech, or bad audio reception, errors will happen. The goal is to limit these issues as much as possible. Two of these primary issues can be alleviated to some extent by developers while the third is on the user. The input of invalid commands can be entirely avoided by creating an extensive grammar file that accounts for all situations. Unfortunately this is an extremely difficult, time consuming task. However, as artificial intelligence progresses, computers should be able to learn new commands themselves making this issue an invalid point. At this point in time, for this project, computer learning will not be utilized to increase commands as users interact with the mirror, we will currently attempt to implement as many ways to phrase commands as possible in the grammar files. The second issue with voice recognition, bad audio reception, can be avoided by utilizing the highest quality microphone possible. There is a tradeoff on this solution however due to the fact that microphones can get expensive quickly as quality increases. For this project, a happy medium will be met between microphone quality and price to ensure the mirror works well while remaining in budget. The final issue, unclear speech, is simply an issue the user must be accountable for. At the current level of technology, clear speech is necessary when interacting with a computer via voice commands and fortunately most users are aware of this limitation and will likely ensure they speak clearly and concisely.

Acknowledging that there will be errors with voice commands, an effort will be made to handle them in the most efficient and helpful manner. All commands will begin with the keyword “Mirror” thus the mirror will know when users are attempting to begin a voice command. If a voice command is started but does not match any of the possible commands in the current context, the mirror will display on screen what command it received as well as offer the user a list of potential commands it believes they may have been attempting to use. It will then allow the user to repeat the command or select one of the options it presented to ensure the user’s intentions can be carried out. This incorrect command, along with the list of options, will be displayed by default in the bottom right corner of the mirror and if the layout is changed, will be displayed wherever the user set the location of possible voice commands to display.

5.2.3 Implementation

The implementation of the voice recognition software will utilize the Microsoft Speech Recognition API for all voice command processing. The object that listens for voice commands is an implementation of the SpeechRecognizer class. This class is run in a separate thread from the main program utilizing the C# async keyword. The object raises interrupt events when speech begins and it listens to the speech then determines if the commands that were spoken fit one of the given rules in any of the grammar constraint files currently loaded. If the command matches any of the rules in the constraint file then the SpeechRecognizer object raises an interrupt and passes arguments of what words were spoken in the parameter list to a method specified at the SpeechRecognizer initialization. It is then expected for the developer to write code to handle the interrupt and determine what speech command was given and execute the appropriate code.

Since a large number of commands will be implemented, all of which may not be needed at the same time, the SpeechRecognizer will load and unload certain grammar files as they become necessary to the program’s execution. Due to this fact, a number of different grammar files will need to be implemented and loaded and unloaded at appropriate times. To ensure modularity, the subsystems will not call into the voice recognition code to load or unload systems, rather the voice recognition system will have a class whose sole job is to monitor the state of the mirror and load or unload grammar files depending upon values stored in MirrorState. This will be implemented in the most minimalistic way possible to ensure a low resource cost of loading and unloading files.

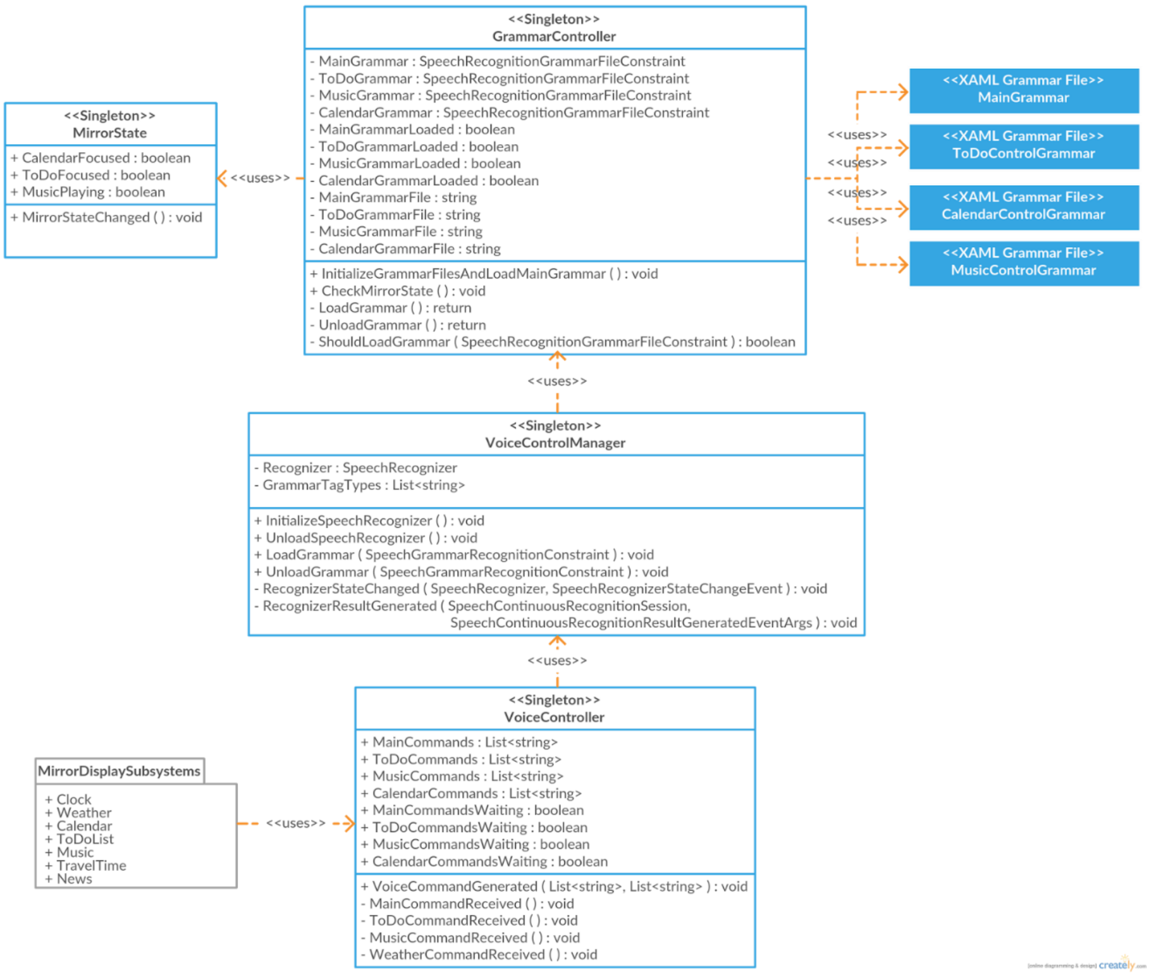

<<Singleton>>
GrammarController
- MainGrammar : SpeechRecognitionGrammarFileConstraint
- ToDoGrammar : SpeechRecognitionGrammarFileConstraint
- MusicGrammar : SpeechRecognitionGrammarFileConstraint
- CalendarGrammar : SpeechRecognitionGrammarFileConstraint
- MainGrammarLoaded : boolean
- ToDoGrammarLoaded : boolean
- MusicGrammarLoaded : boolean
- CalendarGrammarLoaded : boolean
- MainGrammarFile : string
- ToDoGrammarFile : string
- MusicGrammarFile : string
- CalendarGrammarFile : string
+ InitializeGrammarFilesAndLoadMainGrammar ( ) : void
+ CheckMirrorState ( ) : void
- LoadGrammar ( ) : return
- UnloadGrammar ( ) : return
- ShouldLoadGrammar (SpeechRecognitionGrammarFileConstraint) : boolean
<<Singleton>>
MirrorState
+ CalendarFocused : boolean
+ ToDoFocused : boolean
+ MusicPlaying : boolean
+ MirrorStateChanged ( ) : void
<<uses>>
<<XAML Grammar File>>
MainGrammar
<<XAML Grammar File>>
ToDoControlGrammar
<<XAML Grammar File>>
CalendarControlGrammar
<<XAML Grammar File>>
MusicControlGrammar
<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<Singleton>>
VoiceControlManager
- Recognizer : SpeechRecognizer
- GrammarTagTypes : List<string>
+ InitializeSpeechRecognizer ( ) : void
+ UnloadSpeechRecognizer ( ) : void
+ LoadGrammar (SpeechGrammarRecognitionConstraint) : void
+ UnloadGrammar (SpeechGrammarRecognitionConstraint) : void
- RecognizerStateChanged (SpeechRecognizer, SpeechRecognizerStateChangeEvent) : void
- RecognizerResultGenerated (SpeechContinuousRecognitionSession,
SpeechContinuousRecognitionResultGeneratedEventArgs) : void
<<uses>>
<<Singleton>>
VoiceController
+ MainCommands : List<string>
+ ToDoCommands : List<string>
+ MusicCommands : List<string>
+ CalendarCommands : List<string>
+ MainCommandsWaiting : boolean
+ ToDoCommandsWaiting : boolean
+ MusicCommandsWaiting : boolean
+ CalendarCommandsWaiting : boolean
+ VoiceCommandGenerated (List<string>, List<string>) : void
- MainCommandReceived ( ) : void
- ToDoCommandReceived ( ) : void
- MusicCommandReceived ( ) : void
- WeatherCommandReceived ( ) : void
MirrorDisplaySubsystems
+ Clock
+ Weather
+ Calendar
+ ToDoList
+ Music
+ TravelTime
+ News
<<uses>>
creately

Figure 5.2.3.1: Voice Recognition Class Diagram

The voice recognition software will contain three primary classes to handle all voice recognition events. The first will be a VoiceControlManager class which will handle implementation of the SpeechRecognizer and load and unload all grammar files as well as implementation of the method called at interrupt. This file will not call any of the subsystems itself rather it will delegate interaction with all subsystems to another class, the VoiceController class. When a voice command is given, the VoiceControlManager will call methods within the VoiceController to raise flags for the subsystems that a voice command was given. The subsystems will then check with the voice manager to find out what command was given and then act upon that command. This layout provides modularity between the subsystem classes and the voice manager. This would allow for another system to be swapped in place of, or supplement, the VoiceController if the need were ever to arise. The final class, the GrammarController, will interact with the MirrorState class to monitor the current state of the mirror. Depending on the state of the mirror, the GrammarController will pass specific grammar files to the VoiceControlManager to load or unload as needed. This information is summarized in a UML Class Diagram of the voice recognition system in Figure 5.2.3.1 as well as in a State diagram in 5.2.3.2.

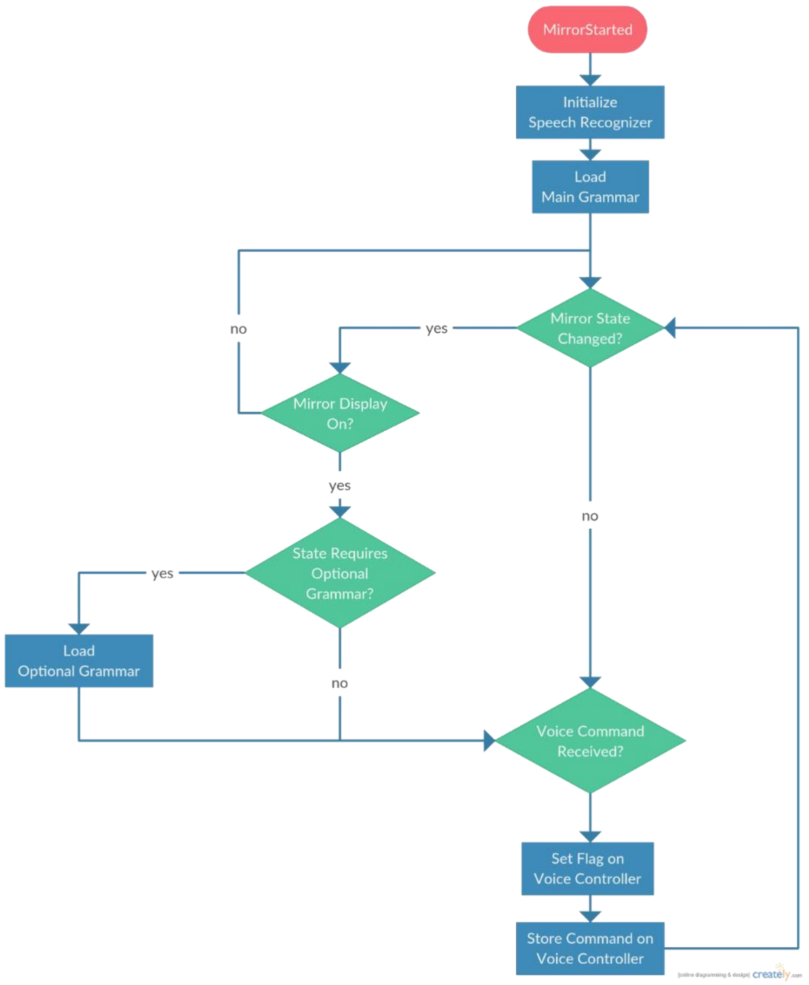


Figure 5.2.3.2: Voice Recognition State Diagram

5.3 Sensor Integration

5.3.1 Requirements & Constraints

For the implementation of the smart mirror, sensors will be required for certain functionality. The mirror needs to be able to monitor specific aspects about its environment to perform in the most suitable manner. There are a number of things that need to be monitored in the environment including temperature, humidity, motion, light, and audio. Each of these external stimuli are required for vital functionality of the smart mirror. Below, each sensor will be discussed in detail about requirements, use, and constraints.

The first requirement is that the mirror be able to sense temperature and humidity in the mirror housing. To meet this requirement, a temperature and humidity sensor need to be housed within the smart mirror. This will allow the mirror to protect itself from environmental factors that could damage the electrical components of the mirror. The temperature will be monitored to ensure the components of the mirror do not overheat. Two temperature thresholds will be set, at the first threshold a fan will be turned on to attempt to circulate air through the mirror housing to cool the internal temperature. At the second temperature threshold, the mirror will simply shut down to avoid overheating. The other sensor within the housing implemented to protect the mirror will be a humidity sensor. As this item will be used in a bathroom, humidity could present a real danger to the electrical components when a user showers and the bathroom becomes humid from the hot water and confined space. By monitoring the humidity in the housing of the mirror, it will be possible to shut off the mirror if necessary to avoid permanent damage to the electrical components.

The second requirement utilizing sensors is that the mirror will turn on and off if users enter the area. This will be implemented utilizing a motion detector which will send a signal if a user enters the room. This is an important utility to the mirror as it will allow the mirror to have an auto on/off feature which increases convenience for the user. The motion sensor will be able to sense movement up to a range of five feet to ensure the mirror turns on as soon as the user is near it. The sensor should have a field of view of, at minimum, ninety degrees. This will ensure that no matter the location of mirror in the room, it will activate when a user enters.

The third requirement is a light sensor. This sensor is also tied to the auto on/off functionality of the mirror. If the mirror detects motion and the light is off, the mirror should illuminate and remain on until a designated time period when no motion occurs. However, if the mirror senses motion and a light is on, then the mirror should turn off if the light in the room has turned off. Due to this requirement, a light sensor is required. The light sensor does not need to be able to sense an extremely accurate level of lighting simply whether or not a light in the room is on or off.

The final sensor implementation required for the implementation of the smart mirror is a microphone to record audio input for the voice recognition system of the mirror. As the voice recognition system is the primary means of interaction with the mirror, it is important that the microphone selected has good audio quality to ensure the voice recognition functionality works efficiently. While the other sensors will be implemented via a MCU which will process their data, this sensor will be connected directly to the Raspberry Pi via a USB connection.

5.3.2 MCU to PC Communications

As four of the five external sensory inputs will be implemented utilizing sensors connected to a MCU, it is important that the Raspberry Pi is able to communicate with the MCU selected. The Raspberry Pi offers I2C implementation which will be the most simple and efficient means of communication with the MCU.

The MCU will be required to transmit information to the Raspberry Pi on regular intervals to ensure that the information from the sensors is always processed in an efficient time period. Each sensor will implement different intervals at which the MCU needs to communicate with the Raspberry Pi. These intervals will be determined by the Pi which will act as the master in the I2C communication while the MCU will act as the slave. These intervals, discussed in the implementation section after this, will be determined based upon the importance of the information as well as the rate at which the information might change. Once the information is passed to the Raspberry Pi it will be utilized to perform specific tasks as deemed necessary by the software.

5.3.3 Implementation

The implementation of each sensor, excluding the microphone, will require microcontroller code, written in C, as well as general Object Oriented code implemented in the software of the mirror, written in C#. The C code should be relatively simple for each sensor as each will simply be evaluating a bit word given by the ADC implemented in the MCU. The C# code is where all the sensory information will be handled once it is acquired.

For the temperature and humidity sensor, data will be passed to the Raspberry Pi once every second. Temperature and humidity are unlikely to change much of the span of a second thus this, relatively slow, read rate is acceptable for the temperature and humidity sensor. The MCU will read more often than that and the most recent information will be passed to the Pi. Once the Pi has the information it will evaluate it with a simple if else block to determine if it is above the first or second threshold, if so the code will execute to act accordingly.

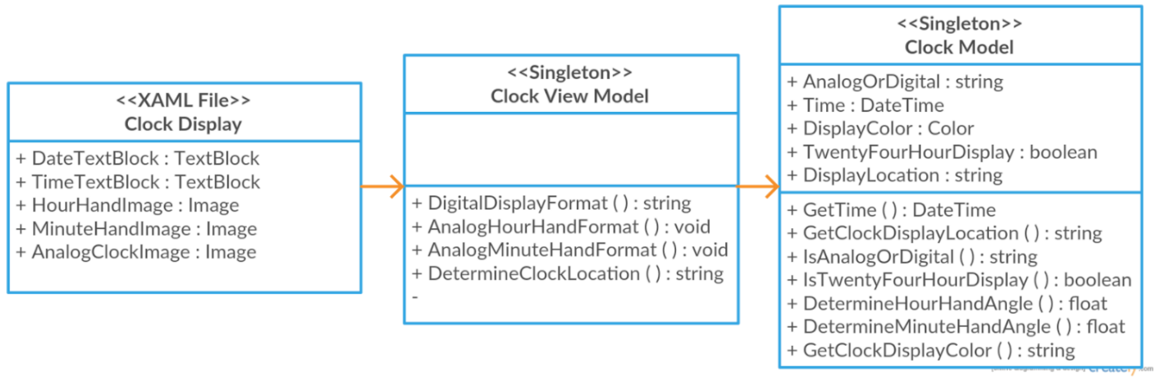
The implementation of the light sensor and the motion detector are tied to the same feature of the mirror, the auto on/off functionality. Thus they will be required to be read simultaneously. The information from these sensors will be passed to the Raspberry Pi five times a second to ensure that there is no lag between the time motion is detected and the mirror display is turned on. The information for the motion sensor will be passed to the Pi which will then activate the mirror display, if it is not already active, if there was motion, otherwise it will do nothing. Once the mirror is activated, the mirror will continue to check if a light is on in the room, if a light is on then the mirror will remain active unless the user tells it to turn off. If the light goes out after it was read as on by the sensor, then the mirror will wait a designated time, less than a minute, and shut off the display. This functionality serves a number of purposes. First, this implementation will be the most user friendly for the user, to not have to worry about turning the mirror on or off. Second, this will save power by ensuring the television is off when the user leaves the room, and lastly the television is the element that produces the most heat in the mirror housing, thus shutting it off whenever possible will help to keep the temperature down.

5.4 Applications

5.4.1 Clock

The clock will be implemented as a customizable display on the GUI and will be controlled by a simple C# class to update the time appropriately. It will utilize the C# DateTime class to update the time appropriately. The time will be checked on regular intervals, half a millisecond long, to ensure that the time is updated every second. The C# DateTime class uses the system time to get the date and time so the correct date and time will need to be set on the computer utilized for the project. This will be handled via the external application created for setup and configuration of the mirror.

The clock implementation for a digital display will be a simple matter of utilizing the built in DateTime element of C#. However, the analog clock implementation will be moderately more difficult and require some math to rotate the hands accordingly. The time and all settings will be implemented in a model layer class used to hold and manipulate the data appropriately. A view model class will be created on top of this to manipulate the data into a format for display via the view component. This design is shown as a UML class diagram in Figure 5.4.1.1.



<<XAML File>>
Clock Display
+ DateTextBlock : TextBlock
+ TimeTextBlock : TextBlock
+ HourHandImage : Image
+ MinuteHandImage : Image
+ AnalogClockImage : Image
<<Singleton>>
Clock View Model
+ DigitalDisplayFormat ( ) : string
+ AnalogHourHandFormat ( ) : void
+ AnalogMinuteHandFormat ( ) : void
+ DetermineClockLocation ( ) : string
-
<<Singleton>>
Clock Model
+ AnalogOrDigital : string
+ Time : DateTime
+ DisplayColor : Color
+ TwentyFourHourDisplay : boolean
+ DisplayLocation : string
+ GetTime ( ) : DateTime
+ GetClockDisplayLocation ( ) : string
+ IsAnalogOrDigital ( ) : string
+ IsTwentyFourHourDisplay ( ) : boolean
+ DetermineHourHandAngle ( ) : float
+ DetermineMinuteHandAngle ( ) : float
+ GetClockDisplayColor ( ) : string

Figure 5.4.1.1: Clock Class Diagram

5.4.2 Calendar

The calendar application will be created using the .NET framework written in C#. The application will utilize the Google Calendar API along with the main classes created by Google that go with the API. The API will provide calendar events to the application and the application will submit new events to the API to be stored as a google calendar event if a user creates a new event. The user will be required to login to their google account on the external application to allow authentication via a web browser for the mirror to access the user's calendar. The first calendar class, which will be a singleton, which will handle all interaction with the Google API. Aside from that, helper classes will be created to store and manipulate the data as appropriate for display on the mirror. The state of the calendar will be controlled by another class and its display information will be passed to a view model component which will bind all the given data to the view elements.

Voice commands will be created and stored in a grammar file to allow the user to interact with the calendar. Any time a user command is given, if the calendar has not been refreshed within the last five minutes it will be refreshed then the command will run. This is to ensure any changes are implemented before allowing the user to attempt to create or modify existing events. The user will have a number of options to control the calendar including scrolling the display to hours in the future, seeing the next day’s calendar, or creating or modifying calendar events as they see fit.

The google calendar API utilizes its own classes and objects to implement all interaction with the service. These will be used to push and pull information from the web. Once we have to objects, the necessary data will be extracted from the objects provided by the API to be converted into a format appropriate for the mirror display. Only the upcoming eight hours’ worth of events will be stored to save resources. The sacrifice this requires is in computation time when the user asks to see another portion of the calendar. To alleviate some of this cost, at a designated time if there are no events left on the calendar, the calendar will automatically shift to the next day’s view so that the user will not have to wait to see tomorrow’s calendar during the evening.

5.4.3 To-do List

The to-do list implementation on the smart mirror will be created utilizing the Todoist software currently available to users of multiple platforms, including Android, IOS, and Windows. Todoist will be extended to the smart mirror, enabling users another avenue to plan and prep for their day in the most efficient way possible. The main functionality of the Todoist application is to create reminders, show them at the appropriate times, and enable the user to delete them once they are completed. All this functionality will be extended to the mirror utilizing voice commands.

The to-do list application is one of the option applications in the mirror and thus, it will have its own set of extra voice commands to cut down on the total number of commands available at any given time. The user will be able to utilize simple voice commands such as show/hide to-do or clear to-do. The third default command for the to-do list will be “Open To-Do” which will open the to-do list into the ‘Extras’ section of the mirror where all extra voice commands such as creating, modifying and deleting to-do list items will be available

The to-do list utilizes HTTP Post functionality to retrieve information from the web API. Utilizing Post requires a specialized API token which will be passed to the mirror when the user logs in on the external setup application. Once the application has the API token tied to the user’s account, pulling information from and passing information to the web service is a matter of utilizing JSON with HTTP to pass data. Every time the to-do list software refreshes, the mirror will check all new to-do items against those currently saved on the mirror and update everything accordingly. This will ensure that items created, modified, or deleted on other devices are changed appropriately on the mirror as well.

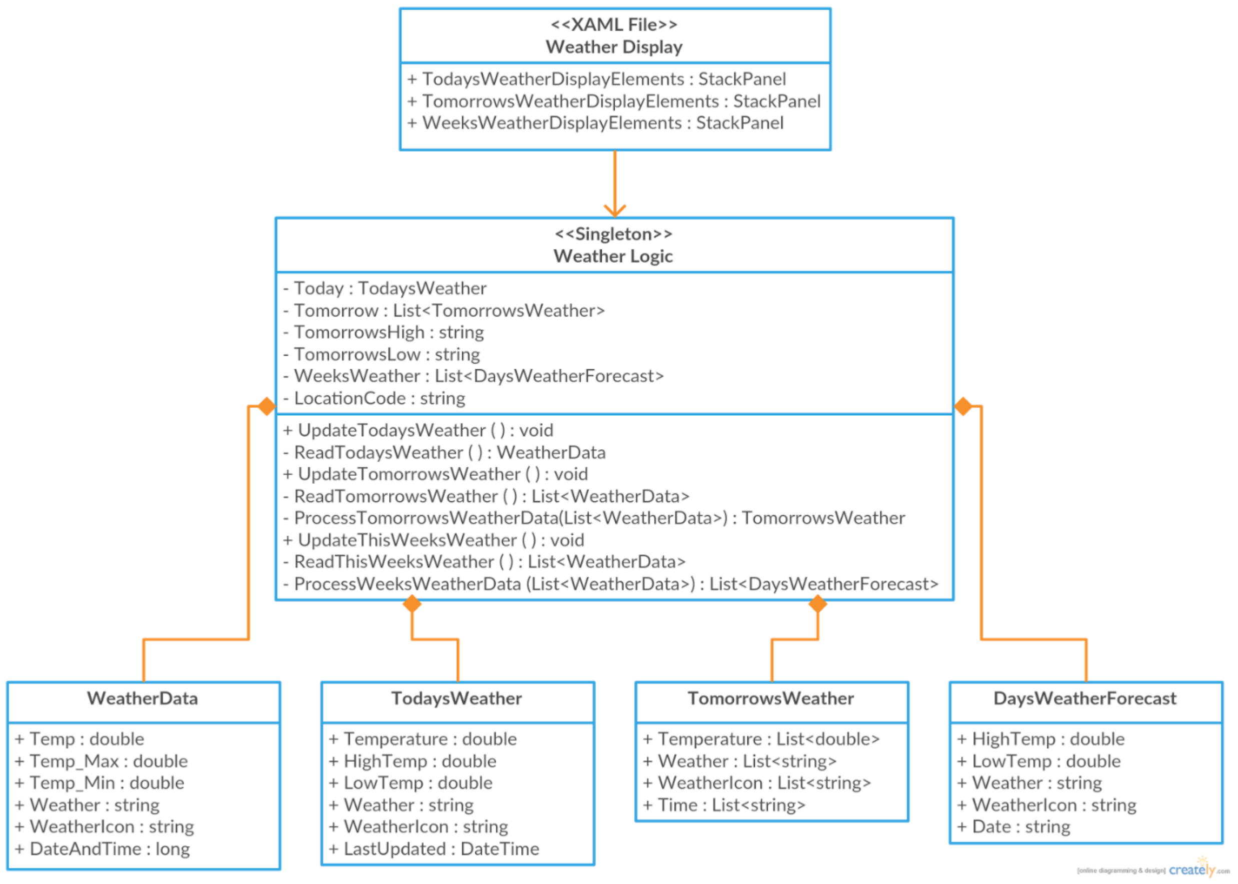
The to-do application will utilize a simple singleton class to handle all business logic tied to the to-do implementation. This singleton will store and manipulate information utilizing a simple POCO created specifically to store the to-do information required for efficient display on the mirror. A list of these to-do objects will be created every time the mirror receives information from the API and then it will be ensured that the contents of the current display are the same as what was received from the API. If not it will be updated accordingly.

5.4.4 Weather

The weather elements of the smart mirror will be implemented utilizing C# to manage all data and perform all required logic. The visual representation will be managed by an XAML view item which will specify all layout implementation details. All weather information will be provided by OpenWeatherMaps which offers a number of APIs with different weather information. Three APIs will be user from OpenWeatherMaps, the first is an API that gives the current weather. The second is an API that gives the upcoming five days weather information in three hour increments. From this second API, the next days weather will be reported in three hour increments. It will report six different sets of information varying over the hours of five A.M. to ten P.M. The final API will provide a weather summary of the next five days.

All weather information will be displayed in the same location on the mirror but only one type of information at a time. In the morning, the current days weather will be shown while in the evening, the next days weather will be shown. Voice commands will be created to allow users to allow the user to request that the mirror show any of the types of weather reports. This will allow the user to glean the most useful information at a glance while checking others with a simple command.

The weather information is retrieved from the website using a simple HTTP request and it reported back as a JSON file. This JSON file will be converted to C# objects with data which will be further converted into a custom C# object that is easier to work with. Once the weather information is obtained it is a simple matter of converting it to a viewable format and displaying it on the mirror. Error handling will be implemented in case the mirror loses connection to the internet in which case the weather information will simply be turned off entirely. The overall design of the mirror element is shown as a UML class diagram in Figure 5.4.4.1.



<<XAML File>>
Weather Display
+ TodaysWeatherDisplayElements : StackPanel
+ TomorrowsWeatherDisplayElements : StackPanel
+ WeeksWeatherDisplayElements : StackPanel
<<Singleton>>
Weather Logic
- Today : TodaysWeather
- Tomorrow : List<TomorrowsWeather>
- TomorrowsHigh : string
- TomorrowsLow : string
- WeeksWeather : List<DaysWeatherForecast>
- LocationCode : string
+ UpdateTodaysWeather ( ) : void
- ReadTodaysWeather ( ) : WeatherData
+ UpdateTomorrowsWeather ( ) : void
- ReadTomorrowsWeather ( ) : List<WeatherData>
- ProcessTomorrowsWeatherData(List<WeatherData>) : TomorrowsWeather
+ UpdateThisWeeksWeather ( ) : void
- ReadThisWeeksWeather ( ) : List<WeatherData>
- ProcessWeeksWeatherData (List<WeatherData>) : List<DaysWeatherForecast>
WeatherData
+ Temp : double
+ Temp_Max : double
+ Temp_Min : double
+ Weather : string
+ WeatherIcon : string
+ DateAndTime : long
TodaysWeather
+ Temperature : double
+ HighTemp : double
+ LowTemp : double
+ Weather : string
+ WeatherIcon : string
+ LastUpdated : DateTime
TomorrowsWeather
+ Temperature : List<double>
+ Weather : List<string>
+ WeatherIcon : List<string>
+ Time : List<string>
DaysWeatherForecast
+ HighTemp : double
+ LowTemp : double
+ Weather : string
+ WeatherIcon : string
+ Date : string
creately.com

Figure 5.4.4.1: Weather Class Diagram

5.4.5 News

In order to give the user a glimpse at what is going on in the world at that moment, our mirror will provide a section of the display dedicated to serving up world news headlines. The source of the news is CNN's World News RSS feed. In order to parse the data in the same fashion as the weather data, we will convert the RSS feed into a JSON format using the online API rss2json [10]. From a programming perspective, the news data is retrieved from the website using a simple HTTP request and it is reported back as a JSON file. Just as is done with the weather information, this JSON file will be converted to C# objects with data which will be further converted into a custom C# object that is more easily manipulated.

After the news information is obtained, our functions will parse the file for information to be displayed using the keywords "title" and "thumbnail" to fetch the news headline and image preview, respective. Afterwards, it is a simple matter of converting it to a viewable format and displaying it on the mirror programmatically by linking the objects to the Image panels in the XAML. We will set the news headlines to refresh every 30 minutes. Error handling will be implemented in the event of internet disconnectivity in which case the news information will simply be turned off completely. The overall design of the news element is shown as a UML class diagram in Figure 5.4.5.1.

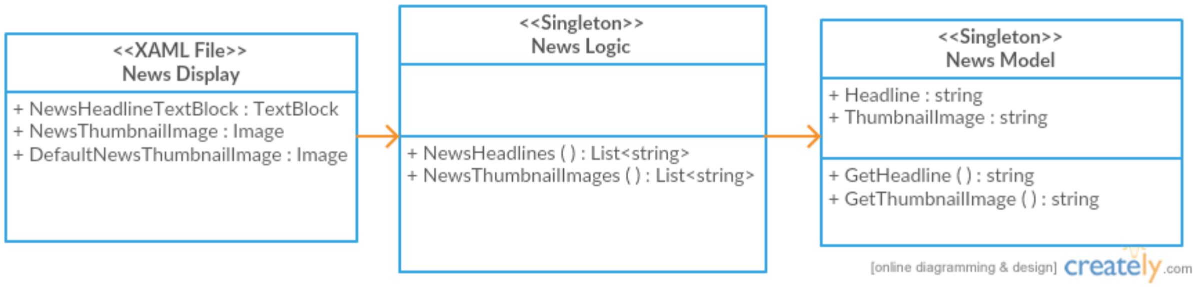

<<XAML File>>
News Display
+ NewsHeadlineTextBlock : TextBlock
+ NewsThumbnailImage : Image
+ DefaultNewsThumbnailImage : Image
<<Singleton>>
News Logic
+ NewsHeadlines ( ) : List<string>
+ NewsThumbnailImages ( ) : List<string>
<<Singleton>>
News Model
+ Headline : string
+ ThumbnailImage : string
+ GetHeadline ( ) : string
+ GetThumbnailImage ( ) : string
[online diagramming & design] creately.com

Figure 5.4.5.1: News Headline Class Diagram

5.4.6 Music

It will be required for the smart mirror to be able to play music specified by the user. Initially, the plan was to implement this utilizing one of the numerous music streaming software options. The options considered included Spotify, Google Play Music, Microsoft Groove, and Pandora. Unfortunately, none of these services offer an official web API that allows playback of music. Spotify offers full music playback for Android and iOS devices, Google Play Music and Pandora have no official API to speak of, and the Microsoft Groove API is still in the early stages and is in a closed beta. Permission may be requested to utilize the Microsoft Groove API but a system needs to be established with Microsoft Azure in order to request access. An attempt will be made to gain access to Microsoft Groove's API for web streaming playback however, if this is unable to be accomplished, it will be required to use a standard music playing application.

If a standard music playing application is utilized, users will be able to upload music from their computer to the mirror utilizing the external application used for set up. Then the mirror will implement a simple playback option for the user to play the music in the specified folder stored on the mirror. Unfortunately, the Raspberry Pi currently has an 8GB microSD card so only a small amount of the music will be able to be stored by default. It would be up to the user to purchase a larger microSD card, flash the Windows 10 IoT image to it, then install the smart mirror application. As most users would not want to be inconvenienced by such a task the goal will be to allow the user to create playlists which we can pass over internet connection to the Pi depending upon which playlist the user requests. To implement this smoothly, only the first couple songs of each playlist would be stored on the mirror while the rest would begin streaming to the mirror once playback of the selected playlist begins. Once the music stops, all but those few initial songs would be removed to keep as much space free as possible.

The code architecture design of this portion of the smart mirror will vary drastically depending upon which method is used. To do this, this portion of design will not be completed until it is determined that access to the Microsoft Groove API will not be possible. At which point the secondary option will be pursued. For now, only the general functionality of this portion will be established.

It will be required that the user is able to give voice commands to the mirror to tell it to play music, change volume, stop or pause music playback, or change the playlist. The mirror music playback functionality will utilize only playlists to simplify the amount of potential commands required as well as the potential for misunderstood voice commands related to unique artist, album, and song names. The playback of music is an added bonus for users of the smart mirror as a number of people enjoy listening to music in their morning and nightly routines. Incorporating this functionality into an existing element of their morning, their new smart mirror, offers them another step towards ease of use and functionality.

5.5 PC/Mobile App

All customizable components of the mirror will be configurable from a simple computer or mobile application. This will allow the user to interact with the mirror for simple tasks such as changing the time without over-encumbering them with too many voice commands. The voice commands will be utilized primarily for daily use to perform tasks such as expanding applications and getting more info. While any configuration tasks will be accomplished at setup, or any time thereafter, using an external application.

This external application will be written for Windows 10 which will be a required application alongside the mirror. In this application, users will set vital information such as location and time. Here the user will also be required to log in to all accounts tied to the mirror so that the mirror is able to pull information from the APIs.

This external application will also be created as a Universal Windows Platform program. This will allow the user to utilize the mirror as a means of configuring the device if they desire by connecting a mouse and keyboard. The external application will carry a number of features tied to the functionality of the mirror. The first requirement is that the application will be required to perform authorization of any account information required by the web APIs. The second is that it will allow users to set information about preferences as well as current location and any other information required for any part of the mirror application. Lastly, it will be required for the application to allow users to create accounts for any of the software tied to the mirror if they do not currently have one. This will allow the user to get the entire mirror up and running utilizing this single application rather than having to go to the companies' websites to create accounts.

6. Project Prototype Construction & Coding

6.1 Parts Acquisition & BoM

Part	Quantity	Unit Cost	Total Cost	Vendor	Acquired
Microsoft IoT Pack for Raspberry Pi 2 - w/ Raspberry Pi 2	1	$114.95	$114.95	Adafruit	Yes
Auria 32” 1080p TV	1	$100	$100	Craigslist	Yes
Select Pine Board 1” x 4” x 8’	2	$10.83	$21.66	Home Depot	Yes
Select Pine Board 1” x 5” x 8’	2	$14.82	$29.64	Home Depot	Yes
Glass Pane 3/32” x 30” x 24”	2	$11.48	$22.96	Lowes	Yes
BDF S05 Window Film One Way Mirror Silver	1	$19.99	$19.99	Amazon	Yes
Miscellaneous Wood and Hardware	1	$10.00	$10.00	Home Depot	Yes
Temperature Sensor	1	$18.19	$18.19	Texas Instruments	No
Humidity Sensor	1	$11.39	$11.39		No
Motion Sensor	1	$8.94	$8.94	Digi-Key	No
USB Microphone	1	$9.99	$10.64	Amazon	No
MCU	1	$9.99	$9.99	Amazon	No
USB Wifi Dongle	1	$0.00	$0.00	Adafruit	Yes

Table 6.1.1: Bill of Materials

Table 6.1.1 reports the names, quantities, individual cost, total cost, and vendor from which all parts for this project were purchased. The table also reports which parts have already been acquired. Anything that has not been purchased will be purchased as soon as it is needed for that parts implementation.

6.2 Final Coding Plan

The plan for coding is to begin by setting up a number of simple UWP programs that will allow prototyping of all the potential parts of the application. Once a simple form of each piece of the application has been implemented in its own program, then the process of bringing the project together into a single, cohesive unit will begin. Prototyping the different aspects of the application in separate projects at the beginning will be important as it will allow the developers to acquaint themselves with the external APIs in an environment where they can write ‘dirty’ code while they stumble around with the new API. Once the developers are familiar with the API, they can import it into the final project utilizing the cleanest, most efficient coding practices possible.

When the project is started, a skeleton will be set up including all classes and methods deemed important to the function of the project as a whole. Then, as the individual applications are prototyped, they will be transferred into this skeleton application created allowing all the parts to begin to work together. This will allow the developers to focus on separate tasks yet work on them simultaneously while causing the least amount of issue with other parts being worked on by other developers. By having all the vital methods in place at the start, the interface of each class will be known from the beginning and it will simply be a matter of implementing the correct information behind the interface in place of the stubbed values that the program skeleton will begin with.

One of the first aspects that will be implemented is the voice recognition. This will be an important aspect to have up and running at the beginning of the project so that the developers will be able to ensure all interaction with the mirror is suited to voice commands in the most seamless way possible. By implementing voice controls early on, the developers will also know exactly what to expect from the voice control interface as it will seem different from a normal user interaction interface with buttons and text boxes. Implementing these key aspects of software construction will allow the process to flow as smoothly as possible and allow multiple developers to work simultaneously on a single project.

The coding will be performed using Visual Studios 2015 as the IDE for this project. This is the required IDE for UWP applications and it offers a number of efficient tools for development. Resharper will be used as an extension to Visual Studios as it allows a simpler interface for interacting with Visual Studios including more user friendly shortcuts and better automation of tasks within the editor windows. A number of other tools will be utilized in addition to visual studios. The primary three items that will be included for development include a subversion software, a bug tracking software, a group messaging application specific to the project, and also a task tracking application. While the first two of these are specific to coding and will be discussed here, all are important aspects to aid the team in development of a project of this scale.

A subversion system will be utilized to keep track of revision history as well as to allow multiple users to work on code simultaneously without fear of messing things up to much as a reversion is always possible. The subversion system being looked at primarily is TortoiseSVN which offers an interesting feature of allowing multiple users to work on the same file simultaneously. Typical SVN systems require users to check out, and lock, a file they are working on while tortoise scans changes made to a file to verify if there were conflicting changes made since the last update. If no changes were made the lines the current developer edited, the changes can be committed while if a conflict is found, tortoise offers the option to choose which of the changes to utilize. This software allows the most seamless work between multiple developers possible, easing one potential issue and allowing the developers to focus on others.

A bug tracking software is an important tool when developing as it allows developers to keep track of all issues found thus far. This is important as occasionally, a bug is found but could be forgotten about later if it is not tracked efficiently. By utilizing a system that all developers can access, all issues can be stored in one location with their symptoms, notes, current state, and people working on it easily accessible. It is not clear which bug tracking software will be utilized at this point in time however it is likely an attempt will be made to find one that can interface with Visual Studios specifically to simplify the process of reporting and tracking bugs.

7. Project Prototype Testing

7.1 Hardware Testing

Testing the hardware is a crucial phase of project development that allows you to assess what you have accomplished and see what areas you need to improve upon. The testing process is what ensures that the final product performs to the specifications and requirements as they were drawn up before the design phase. It is critical to test the hardware because there are so many different components that must be compatible with each other and sometimes this is not apparent in the research. Each of the different hardware component systems will be subjected to different test criteria in order to determine that they perform their intended function and achieve a passing status.

7.1.1 Temperature / Humidity Sensors

We tested the temperature and humidity sensors in order to determine how accurately they perform in various conditions. The sensors must be able to perform accurately to a certain margin of error in order to be considered for our use in the project. The tests completed for temperature sensor are outlined in Table 7.2.1.1 below. The tests completed for the humidity sensor can be seen in Table 7.2.1.2 below.

Test	**Procedure**	**Expected Outcome**
Low temperature sensor accuracy	Acquire results from low temperatures (<10 °C)	+/- 1 °C error margin
Room temperature sensor accuracy	Acquire results from room temperatures (~25 °C)	+/- 1 °C error margin
High temperature sensor accuracy	Acquire results from high temperatures (>40 °C)	+/- 1 °C error margin
High humidity temperature sensor accuracy	Acquire results in a high-humidity environment (>90%)	+/- 1 °C error margin

Table 7.2.1.1: Temperature Sensor Tests

Test	**Procedure**	**Expected Outcome**
Low humidity sensor accuracy	Acquire results from area of low humidity (<40%)	+/- 2% error margin
Medium humidity sensor accuracy	Acquire results from area with moderate levels of humidity (40%-70%)	+/- 2% error margin
High humidity sensor accuracy	Acquire results from area with high levels of humidity (>70%)	+/- 2% error margin
High temperature humidity sensor accuracy	Acquire results in a high-temperature environment (>40 °C)	+/- 2% error margin

Table 7.2.1.2: Humidity Sensor Tests

7.1.2 Microcontroller Signal Control

The testing of the microcontroller unit (MCU) signal control is aimed at determining how it behaves with the temperature sensor from the PCB. The system is controlled by analog pins which allow the test to prove that the pins were being activated and produce the correct value. These tests are outlined in Table 7.2.2.1 below.

Test	**Procedure**	**Expected Outcome**
Specific Pin	Enable the highest active setting to the specific desired pin.	The sample serial output should demonstrate the value from the pin.
Pin Value	Connect the pin to the temperature sensor on the PCB in order to modulate its value.	The pin should correctly transmit its value to the receiver over a serial connection.
Code Behavior	Manually simulate temperature and humidity sensor readings.	The MCU should correctly transmit the temperature and humidity values to the receiver over a serial connection.

Table 7.2.2.1: MCU Signal Control Tests

7.2 Software Test Plan

The initial software tests will confirm that we will have a bug-free system before considering its release. The initial test will help identify any flaws with the smart mirror and help us make any improvements that we see necessary. Lastly, beta testing will allow us to determine if there are any features that could be implemented in different ways to improve functionality to the user.

There will be two testing processes, both done by the group members. First, we will test the individual features of the mirror for performance compliance using our development computers with Visual Studio 15 installed on Windows 10. These tests will be performed with mouse and keyboard connected so that set up of the mirror and manual refreshing can be tested. Second, there is one mirror prototype that will be used daily by a group member as a beta test to ensure the mirror provides the intended functionality. The environment for this second round of testing will be the same as the the live environment.

We will keep a running document of all bugs found thus far and as developers work they will select bugs from this document to fix. There are three members so if member A finds the bug in testing, member B fixes the bug, then member C will be the one to do testing after the fix to ensure any new bugs introduced in the fix will could be caught. As testing progresses, we will also maintain a document of tests to perform, this will allow us to run multiple test periods after every set of bug fixes to ensure no new bugs were introduced.
During testing we will rate the bugs on a level of impact to the system. They will be rated 1 - 5 where 1 is a critical level bug that breaks the system entirely and 5 is a minor cosmetic bug. Once there are no level 1 - 3 bugs we will consider the product “sufficient for user operation” but will continue to remove as many of the other bugs as possible and produce a polished finished product.

7.3 Software Testing

Throughout the development and testing phases, the smart mirror functionality was constantly being tested using the Debug features within Visual Studio in order to build the application for local execution on our machines. This was made possible by the fact that our smart mirror software is built on the Universal Windows Platform (UWP), allowing us to build a native version that is identical to the version loaded onto the mirror, apart from its processor architecture. A majority of the testing was performed on our local machines because compiling and building the project for the local machine is far quicker than remotely deploying it to the mirror; however, we deployed the project to the mirror when necessary in order to evaluate how the UI was displayed and to make adjustments. Once development was complete, all of the test cases were finally performed on the actual mirror in order to confirm proper functionality for the end user.

7.3.1 Graphical User Interface

The graphical user interface displays all of the various software features that are built into the smart mirror and must conform to a set of requirements on both a functional and nonfunctional level. The functional requirements are comprised of the specific feature set that we implemented into the mirror. Each of the requirements is expected pass an objectively determined goal in order to be considered functional. Conversely, the nonfunctional requirements demonstrate a level of quality assurance that must be subjectively assessed. The expected outcome of these requirements was the culmination of our discussions and experiences while using the smart mirror.

7.3.2 Functional

The functional tests of the graphical user interface encompasses all of the requirements that can be objectively determined as a success or failure. These tests cover proper mirror boot behavior, general user interface function as well as voice recognition behavior. The tests have been divided into the two tables below. Table 7.4.1.1.1 provides details on the tests and results for assorted functions while Table 7.4.1.1.2 demonstrates all of the speech recognition-related tests.

Test	**Procedure**	**Expected Outcome**
Smart Mirror GUI on boot	Power up the mirror by plugging in the power cable.	The Smart Mirror app should load upon boot rather than the Windows 10 IoT Core Dashboard utility.
Initial application data fetch	Boot up the mirror and ensure that the Smart Mirror app is booted.	The various software features should fetch their respective data such as the weather conditions and news headlines.
Application data refresh	Observe whether the various software features update their content according to their set refresh cycles. Reduce the refresh intervals in the code to expedite the testing.	Each software feature should update within 5 seconds after their specified refresh intervals have passed (given that the data from the API is different from the existing data).
Excessive Temperature/Humidity Warning	Manually initiate the code path that will trigger the warning associated with exceeding a favorable operating threshold.	The mirror will display a warning message to user to indicate that the humidity/temperature sensor has detected unfavorable operating conditions.
GUI becomes hidden automatically	With the GUI visible, step away from the motion sensors and wait a moment.	Once the mirror has not detected any motion after 60 seconds, the UI elements will become hidden.
Loss of Internet Connectivity Test	Remove the USB WiFi dongle to force a loss of internet connectivity.	In the event of internet connectivity disruption, the software features that rely on the internet connection should respond as programmed without compromising total system functionality.

Table 7.4.1.1.1: Assorted Functional Smart Mirror Tests

Test	**Procedure**	**Expected Outcome**
Different weather views	Use speech recognition to access the different weather view options. Available voice prompts: • Show/hide tomorrow's weather • Show/hide this week's weather • Show today's weather	The voice commands should successfully display the requested weather conditions in at least 75% of all attempts.
Show/Hide GUI	Use speech recognition to show or hide all of the software feature elements by saying "Mirron On" or "Mirror off" respectively.	The voice commands should show or hide the GUI successfully in at least 75% of all attempts.
Music Control	Use speech recognition to initiate music searches and control playback. Available voice prompts: • Search Pandora for '*Song, Artist, Genre'* • Pause music • Next track	The voice commands should successfully perform the music-related functions in at least 75% of all attempts.

Table 7.4.1.1.2: Speech Recognition Smart Mirror Tests

7.3.3 Non-Functional

The non-functional tests of the graphical user interface were determined in order to test the requirements that cannot be strictly determined as a success. These subjective tests were determined with our combined thoughts and discussions pertaining to our desired smart mirror behavior. Table 7.4.1.2.1 outlines all of these non-functional tests including the procedure for each as well as the "passing" criteria.

Test	**Procedure**	**Expected Outcome**
Boot-up time	Boot up the mirror by plugging in the power cable.	The Smart Mirror app should boot within 30 seconds of providing power.
Shutdown time	Shut down the mirror properly by initiating a shutdown with the companion app.	The Smart Mirror should shut down completely within 20 seconds of issuing the command.
Speaker Volume Level	Initiate Pandora music playback using speech recognition and assess the volume level.	All group members should agree that the volume level is appropriate and audible for the mirror.
Speech Recognition Consistency	Have all group members perform the multitude of speech recognition prompts, and variations of each, and observe the mirror's response.	The mirror should appropriately respond to the voice commands and variations with a 75% success rate.
DIsplay Brightness	Place the mirror in a bright environment and power it up.	The user interface elements should still be visible despite the unfavorable lighting conditions. The rest of the screen real estate should retain its mirror-like finish as provided by the display's reflective tint.
Motion Detector Reaction	Put the mirror into an inactive state with the user interface hidden and then trigger the motion sensor by walking in front of its line of sight. Repeat this process 10 times.	The mirror should become active with its user interface elements visible within 5 seconds of stepping into its line of sight.

Table 7.4.1.2.1: Non-Functional Smart Mirror Tests

8. Administrative Content

8.1 Milestones

When attempting to take on a project like our Smart Mirror, it is imperative to set realistic and attainable milestones in order to maintain an effective pace and accomplish all of our goals for the project. The majority of our time in Senior Design I has been spent researching and designing the various components that our mirror will consist of. There are hundreds of different sensors to choose from and we have to make sure we choose an MCU that is versatile enough to be compatible with all of the extra external hardware we intend to implement within the mirror. Table 8.1.1 lays out all of the milestones that we created for Senior Design I along with their start and end dates. Each task had varying degrees of intensity and naturally some tasks require more time than others to effectively complete. We followed this milestone table fairly close and were able to accomplish each task in a timely manner.

Task Name	**Duration**	**Start**	**Finish**
Display Research	1 week	Mon 2/8/16	Fri 2/12/16
Voice Recognition Software Research	2 weeks	Mon 2/8/16	Fri 2/19/16
Two-way mirror Research	2 weeks	Mon 2/15/16	Fri 2/26/16
Application Control Program Research	2 weeks	Mon 2/15/16	Fri 2/26/16
PC Component Research	2 weeks	Mon 2/22/16	Fri 3/4/16
Temperature, Light & Motion PCB Research	4 weeks	Mon 2/29/16	Fri 3/25/16
GUI Research	3 weeks	Mon 2/29/16	Fri 3/19/16
Application Control Program Design	6 weeks	Mon 3/7/16	Fri 4/15/16
Temperature, Light & Motion PCB Design	5 weeks	Mon 3/14/16	Fri 4/15/16
Lighting Control System Research	2 weeks	Mon 3/14/16	Fri 3/25/16
Voice Software & Webcam Design	6 weeks	Mon 3/21/16	Fri 4/29/16
Frame & Housing Research	2 weeks	Mon 3/21/16	Fri 4/1/16
GUI Design	7 weeks	Mon 3/21/16	Fri 4/29/16
Lighting Control System Design and Prototyping	4 weeks	Mon 3/28/16	Fri 4/22/16
Display and Mirror Design	2 weeks	Mon 3/28/16	4/8/16
Temperature, Light & Motion Processing Design	4 weeks	Mon 4/4/16	Wed 4/27/16

Table 8.1.1 - Senior Design I Milestone Schedule

Table 8.1.2 below outlines our prospective milestones for Senior Design II. Whereas Senior Design I focuses on the research and design aspects of the project, Senior Design II will consist primarily of the actual construction of our smart mirror. The culmination of twelve weeks' worth of research and design will allow us to begin putting the pieces together and fabricate a working prototype.

Task Name	Duration	Start	Finish
Voice Software & Webcam Prototyping	3 weeks	Mon 8/22/16	Fri 9/9/16
Display and Mirror Prototyping	2 weeks	Mon 8/29/16	Fri 9/16/16
Temperature, Light & Motion Processing Prototyping	4 weeks	Mon 8/29/16	Fri 9/23/16
PCB Prototyping	4 weeks	Mon 9/5/16	Fri 9/30/16
Housing & Frame Prototyping	2 weeks	Mon 9/5/16	Fri 9/16/16
Application Control Program Prototyping	6 weeks	Mon 9/12/16	Fri 10/21/16
Order PCB	1 week	Mon 9/12/16	Fri 9/16/16
PCB Testing	2 weeks	Mon 9/26/16	Fri 10/7/16
Mirror construction complete	1 week	Mon 10/3/16	Fri 10/7/16
Mirror debugging	10 weeks	Mon 10/3/16	Fri 12/9/16

Table 8.1.2 - Potential Senior Design II Milestone Schedule

8.2 Budget & Finances

The budget for our Smart Mirror was drawn out early on in the overall scheme of our project's development. Our original budget was slightly overestimated to allow for some leeway in each category and to move some of the funds around to different components. Compared to other smart mirror projects, our project takes advantage of the relatively low cost of the Raspberry Pi 2 as the main computer rather than building a diminutive, yet full-fledged desktop machine. We were also able to save costs by utilizing a reflective tint for the display rather than purchasing an expensive two-way glass mirror pane.

One of the primary reasons that there isn't a commercial smart mirror product on the market is because of the costs associated with the research & design, marketing, and materials to build such a product. We aimed to keep our product at a relatively low-cost. Because the software portion of our Smart Mirror was built as a UWP (Universal Windows Platform) for Windows 10, the source code can actually be compiled for any platform that support UWP apps such as Windows 10 x86/x64, Windows 10 Mobile, and of course Windows 10 IoT (Internet of Things) which is the system being employed on the Raspberry Pi 2. As a result, should someone acquire the source code to our Smart Mirror, or if we end up submitting it to the Windows Store, they would only need to construct the mirror portion of the project and our code would execute flawlessly on their version of the mirror.

With our original budget, we intentionally overestimated each of the categories as shown in Table 8.2.1 which compares our original and final budgets. It should be made clear that this isn't truly a final budget just yet as we won't know the final cost of some of the components or services until we undergo the processes in Senior Design II. As a result of overestimating each component, our final spending was less than we originally planned in every category. Overall, we have managed to spend just around half of the original budget. You can see that we were able to save the original $50 allocated for speakers by utilizing the built-in speakers from the TV.

Item	Original Budget	Final Budget	Difference
Auria 32" 1080p HDTV	$200	$100	$100
Microsoft IoT Pack for Raspberry Pi 2 - w/ Raspberry Pi 2	$150	$114.95	$35.05
Microphone	$50	$10.64	$39.36
Speakers	$50	$0	$50
Mirror Assembly/Frame	$200	$104.25	$95.75
MCU & Sensors	$100	$48.51	$51.49
Total:	$750	$378.35	$371.65

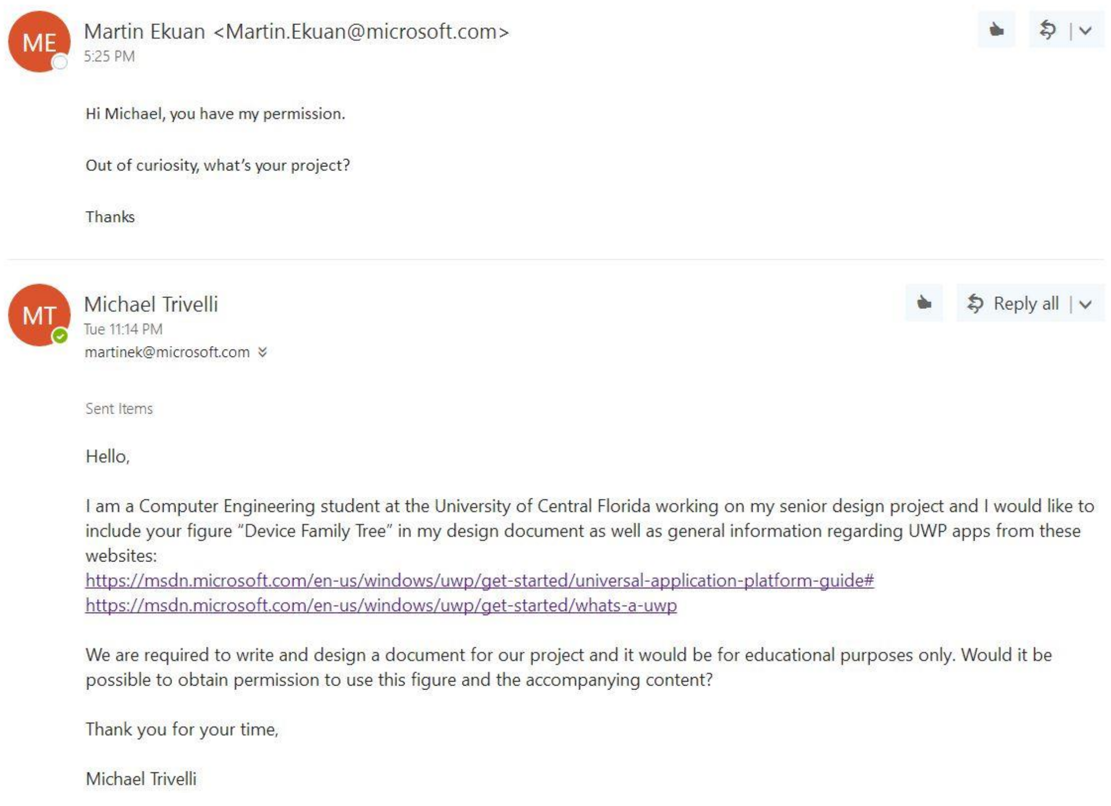
Table 8.2.1 - Original Budget vs Final Budget

8.3 Work Distribution

By following the milestone chart closely and with enough diligence, we have been able to make tremendous progress on our smart mirror project through the end of Senior Design I. The work distribution for this project has been split fairly amongst the three members with each member tasked with working the area that suits them best. With Hector being the sole electrical engineer, he focused hardware aspects of the mirror which included researching and designing the PCB and sensors. Both Justin and Michael focused on the software portion of the smart mirror, exercising their software engineering skills as computer engineer majors. All of the members worked closely together while implementing the sensors into the MCU and testing their functionality through the Raspberry Pi’s GPIO pins.

Appendix A: Copyright Permissions

Microsoft



Martin Ekuan <Martin.Ekuan@microsoft.com>
5:25 PM

Hi Michael, you have my permission.

Out of curiosity, what’s your project?

Thanks

Michael Trivelli
Tue 11:14 PM
martinek@microsoft.com

Sent Items

Hello,

I am a Computer Engineering student at the University of Central Florida working on my senior design project and I would like to include your figure “Device Family Tree” in my design document as well as general information regarding UWP apps from these websites:
https://msdn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide#
https://msdn.microsoft.com/en-us/windows/uwp/get-started/whats-a-uwp

We are required to write and design a document for our project and it would be for educational purposes only. Would it be possible to obtain permission to use this figure and the accompanying content?

Thank you for your time,

Michael Trivelli

Appendix B: References

[1] https://msdn.microsoft.com/en-us/windows/uwp/get-started/whats-a-uwp
[2] https://msdn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide
[3] http://www.electronics-tutorials.ws/io/io_3.html
[4] http://www.omega.com/prodinfo/Integrated-Circuit-Sensors.html
[5] http://www.electronics-tutorials.ws/io/io_4.html
[6] http://www.engineershandbook.com/Components/proximitysensors.htm
[7] http://www.sensorsmag.com/sensors/humidity-moisture/choosing-a-humidity-sensor-a-review-three-technologies-840

[8]http://www.amazon.com/VAlinks-Flexible-Microphone-Compatible-Recording/dp/B014MASID4?ie=UTF8&psc=1&redirect=true&ref_=ox_sc_act_title_1&smid=A3JUJR0KZ89KR2
[9] https://www.elprocus.com/microcontrollers-types-and-applications/
[10] rss2json.com