Fall 12-2022

# Measuring the Effectiveness of Light Concentration with the Catoptric Surface

Samatha Kodali
*Washington University in St. Louis*

WASHINGTON UNIVERSITY IN ST. LOUIS

McKelvey School of Engineering
Department of Computer Science and Engineering

Thesis Examination Committee:
Roger D. Chamberlain, Chair
Chandler Ahrens
Chris Gill

# Measuring the Effectiveness of Light Concentration with the Catoptric Surface

by
Samatha Kodali

A thesis presented to the
McKelvey School of Engineering
of Washington University in
partial fulfillment of the
requirements for the degree
of Master of Science

December 2022
St. Louis, Missouri

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgments

Thank you to the Washington University Department of Computer Science and Engineering. I would like to acknowledge Roger Chamberlain, Chris Gill, and Chandler Ahrens for their guidance and support throughout the entire experiment. I would also like to thank Scott Sirri for this work on the C++ code base and Bill Siever for his help choosing a micro-controller for the experiment.

Samatha Kodali

*Washington University in St. Louis*
*December 2022*

Dedicated to my parents.

ABSTRACT OF THE THESIS

Measuring the Effectiveness of Light Concentration with the Catoptric Surface
by
Samatha Kodali
Master of Science in Computer Engineering
Washington University in St. Louis, 2022
Professor Roger Chamberlain, Chair

The Catoptric Surface is an array of mirrors arranged on the interior of a large window which serves to manipulate the effects of natural lighting inside the room. The framework for moving the mirrors uses an interface which connects various Raspberry Pis and Arduino Unos to move stepper motors that connect to each mirror and move them as desired by the user. This kind of light manipulation allows the atmosphere of the room to be modified using natural lighting rather than artificial lighting and can be useful for varying the way people interact with a space. The intensity at which various configurations of the mirrors vary the lighting in the room becomes important for the user/controller of the Catoptric Surface to know in order to manipulate the mirrors in a fashion that will deliver the desired effect. Thus, in order to provide light concentration information to the user/controller, we have conducted a study in which we use a light sensor to gather the light concentration information (in lux) in the room with the mirrors in different configurations and discuss the implications of such measurements for future use of the Catoptric Surface.

# Chapter 1

# Controlling the Natural Lighting of Environments

## 1.1 The Importance of Natural Light in our Environments

The natural light emitted from the Sun is commonly utilized to illuminate buildings. The illumination of an environment with natural light provides tangible health benefits, economic benefits, and energy benefits [3, 4, 6, 7, 8, 9, 10, 12, 13, 14]. By exposing individuals to more natural light, they have lower levels of cortisol as well as higher levels of melatonin at 10pm, both of which result in lower levels of depressive symptoms and an increased quality of sleep [9]. Furthermore, the use of natural light can alter one's spatial perception in the environment [4]. The beneficial effects of natural light supplies designers with another mechanism of control through which to alter the experience of occupants in a particular environment.

In the context of this discussion/thesis, the environment in which we observe natural light consists of an atrium with glass windows comprising the north-facing and south-facing walls and no windows on the east-facing and west-facing walls. Furthermore the ceiling of our environment does not contain any windows/glass through which natural light could enter.

Thus, the natural light that enters through the north-facing and south-facing walls of the room combines to create the natural light for the context of our environment.

## 1.2 Challenges of Controlling the Natural Light in Our Environment

Currently, the presence of natural light in a room primarily depends on static features of buildings such as the quantity and location of windows as well as the quantity, location, and material of doors [4]. Orienting buildings east-to-west, using lighter colors on interior spaces, and allowing natural light from higher up to enter the building from multiple sides constitute other features that manipulate natural lighting inside rooms [12]. Unfortunately, changing these types of features involves altering fundamental aspects of the building which tend to be non-trivial and costly [11]. In addition to these features being expensive to modify, the time frame of their modifications can be quite long. Due to factors that affect the positioning and intensity of the Sun, such as date, time, and weather, the amount of natural light present in a room changes at a faster rate than it takes to modify the position of a window or the orientation of a building. Thus, in order to effectively possess the ability to manipulate the natural lighting in a room, smart, dynamic structures, such as automated blinds can be considered [11].

Dynamic structures are built into/on top of the existing building to provide a way to manipulate natural light in a room with the current building infrastructure intact. These dynamic structures must be intelligent enough to account for the factors of date, time, and weather that modify the positioning and intensity of the Sun. In addition, the system must be user-friendly enough to allow for a user to easily manipulate the natural light in a room as they desire.

Overall, manipulating natural lighting in a room by changing more fundamental features of the building such as window location and building orientation becomes too costly and slow to be useful. Therefore, we turn to dynamic structures to be able to modify the natural lighting while preserving the building's fundamental features. However, we have to be cognizant of the challenges of creating a dynamic structure that possesses enough intelligence to accurately and usefully manipulate natural light despite the factors that often change Sun positioning and intensity.

## 1.3 Current Approaches to Control the Natural Lighting in Environments

There are currently various methods of attempting to control the natural light within an environment. Two broad categories of natural light manipulation methods are:

(1) Systems that have shade in order to block direct sunlight and diffuse light.

(2) Systems that don't contain shading and instead redirect sunlight [11].

Four such examples are: manually controlled blinds/window shades, light shelves,, prismatic panels, and dynamically controlled mirrors.

### 1.3.1 Manually Controlled Blinds/Window Shades

Manually controlled blinds/window shades, as seen in Figure 1.1, consist of human-controlled, interior window-coverings whose movement will not be automated. They fall into the first category of methods for natural light manipulation because they contain shading to block natural light rather than allowing natural light in the room and then redirecting it. The horizontal slats of blinds/window shades on styles such as Venetian blinds provide the best utilization of natural light, however since humans control their movement, they can be left in a non-optimal position for long stretches of time [5, 11]. Therefore, an improvement to this system would be making it automated via a cyber-physical system so that the responsibility of shade/window blind movement no longer falls completely on the human.

### 1.3.2 Light Shelves

Light shelves, as seen in Figure 1.2, are horizontal panels that sit on the side of buildings (typically parallel to the ground) and work to reflect natural light deeper into the building/room for the windows above their top surface, and provide shade and/or glare reduction for the windows below their bottom surface. Since they contain shading to block natural light, they fall into the first category of methods for natural light manipulation. While light shelves

Figure 1.1: Manually controlled window blinds.

work well to protect from harmful/overly-harsh direct sunlight in lower latitude locations, they are not very effective in high latitude countries as they do not provide enough shading [11]. Additionally, their construction and use may be expensive and/or time-consuming depending on their materials and installation location and technique. An improvement for this system would involve researching the optimal size of the light shelf based on the latitude of the location in which they will be placed. An additional improvement would be creating a cyber-physical system to modify the angle/shape of the light shelves as needed.

### 1.3.3 Prismatic Panels

Prismatic panels, as seen in Figure 1.3, are panels that are shaped to redirect natural light. Since they do not contain shading, but instead redirect natural light, they fall into the second category of methods for natural light manipulation. Though in sunny and clear conditions

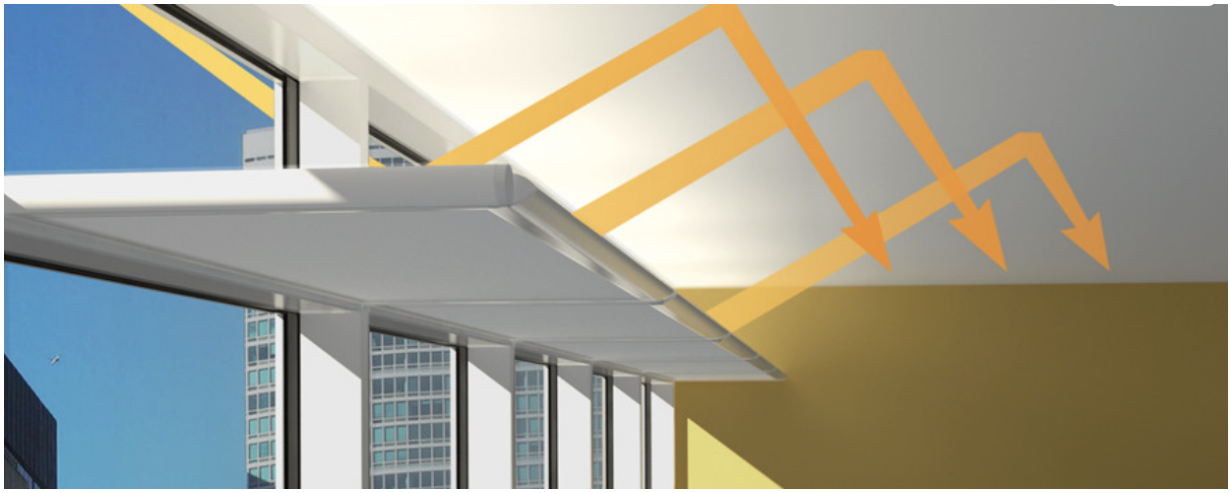

Figure 1.2: Light shelves.

prismatic panels allow for uniform natural light distribution, in cloudy conditions they do not have much of an effect, thereby reducing their practicality [11]. They also involve altering the structure of the building, and thus are not a simple or easy solution to implement.

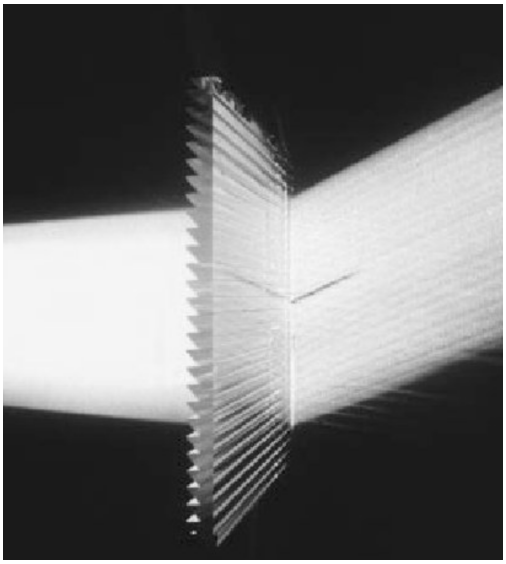

Figure 1.3: Prismatic panels.

### 1.3.4 Dynamically Controlled Mirrors

Dynamically controlled mirrors are a set of mirrors that can be moved which are used to reflect natural light in different patterns into an environment. The mirrors have a system through which their positioning can be controlled in order to change the amount and pattern of natural light entering the environment. These systems are effective because they can be added onto a building rather than needing to alter the structure of the building in order to be implemented.

An example installation with dynamically controlled mirrors is shown in Figure 1.4. This is the experimental system that is the subject of this thesis.

Figure 1.4: Example of dynamically controlled mirrors [1].

## 1.4 Contributions and Thesis Outline

The contributions of the thesis include the following:

- Maintenance of C++ Catoptric Surface code base
- Documentation, UML diagramming, and communication of the C++ Catoptric Surface code base

- Set up data collection framework for longer-term light sensor collection experiments
- Illuminance measurements with the Catoptric Surface mirrors in different configurations

This thesis covers current types of light manipulation systems to modify the natural lighting in a room in Chapter 1. It then discusses the light manipulation system we built, the Catoptric Surface, in Chapter 2. The experiment we ran to obtain light sensor measurements in order to assess the effectiveness of the Catoptric Surface at redirecting natural light is discussed in Chapter 3. In Chapter 4 we discuss the potential future work and conclusions of the thesis.

# Chapter 2

# The Catoptric Surface

In order to experiment with and observe the manipulation of natural light in an environment, we created our own dynamic structure for controlling natural light called the Catoptric Surface [1, 2]. The Catoptric Surface is composed of just under 650 round mirrors arranged in an array with 16 rows and 49 columns. Each mirror has 2 stepper motors to allow for movement. The motors can be controlled by a human via an arrangement of Arduino Unos and Raspberry Pis. The mirrors were placed on the interior of a south-facing glass wall of the Steinberg atrium and their positioning redirected the natural light that entered the Steinberg atrium.

## 2.1 Physical Structure

The physical structure of the Catoptric Surface is housed in Steinberg Hall. The mirrors are set up in an array alongside the South wall. Each row of mirrors is held up using wires and each mirror has two motors on it. The motors for each mirror are pan and tilt, allowing the mirrors two degrees of freedom. Each row of mirrors has an Arduino to control the mirrors in that particular row. Both the eastside section of the wall and the westside section of the wall have a Raspberry Pi to communicate to the Arduinos. These microcontrollers are housed in boxes mounted next to the Catoptric Surface.

## 2.2 Embedded System Structure

A vital part of the cyber-physical system we created was the software used to allow users to move the mirrors. The code that controlled the Catoptric Surface's stepper motors in order

to change the positioning of the mirrors were on the Raspberry Pis and Arduino Unos of the system. The Raspberry Pis had C++ code on them to take user-input. The user-input that the Raspberry Pis took were files with four comma-separated integer values (CSV files) or a string that represents one of the commands in the set that the system has outlined as acceptable. Those four values were used to identify which mirror to move, which direction to move the mirror in, and how much to move it. This user-input is parsed by the C++ code and communicated from the Raspberry Pis to the corresponding Arduino Uno for the mirror for which movement is desired. The communication from the Raspberry Pi to the Arduino Uno happens over Universal Serial Bus (USB) and has its own Finite State Machine (FSM) associated with it which outlines its process.

### 2.2.1 C++ Code Structure

There are four main pairs of header/source files in the C++ code:

(1) `CatoptricController.hpp` and `CatoptricController.cpp`,

(2) `CatoptricSurface.hpp` and `CatoptricSurface.cpp`,

(3) `CatoptricRow.hpp` and `CatoptricRow.cpp`, and

(4) `SerialFSM.hpp` and `SerialFSM.cpp`.

Overall, these groups of files contain the following classes/structs which allow for control in the system: `CatoptricController`, `CatoptricSurface`, `CatoptricRow`, `SurfaceDimensions`, `SerialPortDict`, `SerialPort`, `MotorState`, `Message`, `SerialFSM`, and `SerialComp`. The relationship between these classes and structs can be seen in the UML diagram in Figure 2.1 below.

The `CatoptricController` contains a `CatoptricSurface` which it is in charge of controlling. The `CatoptricSurface` contains $x$ `CatoptricRow` objects where $x$ is the number of rows of mirrors in the system. Since the Catoptric Surface in Steinberg contains 16 rows, there are 16 `CatoptricRow` objects contained in the `CatoptricSurface` object. The `CatoptricSurface` object also contains a `SurfaceDimensions` object in order to have the length of each row appropriately outlined and saved. The UML diagram for `SurfaceDimensions` can be seen in

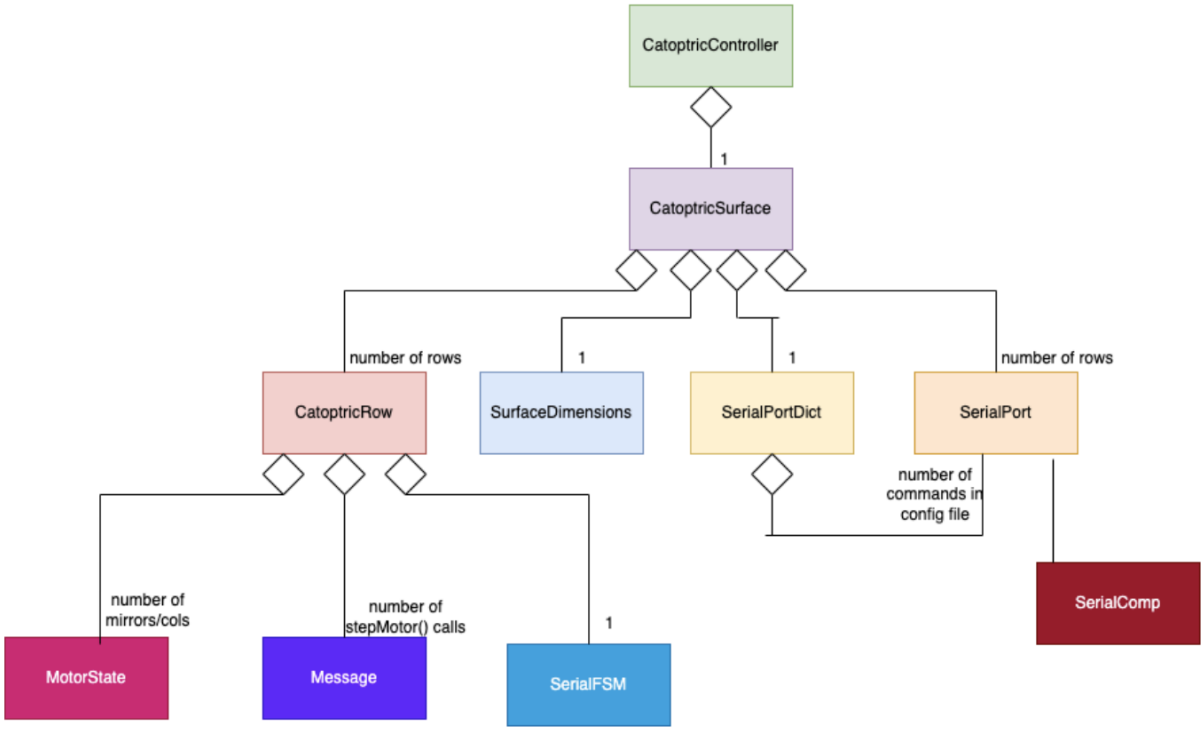


Figure 2.1: UML diagram with class/struct relationships.

Figure 2.7. Additionally, the `CatoptricSurface` class contains a `SerialPort` object for each row in the Catoptric Surface and a single `SerialPortDict` object. A `SerialPort` object can be used to represent the Arduino with which the Raspberry Pi is communicating by holding identifying information such as the serial number of the Arduino, the symlink through which it is available, and the row on the Catoptric Surface that the Arduino is associated with. The UML diagram for `SerialPort` can be seen in Figure 2.9. The `SerialPortDict` object consists of a vector of `SerialPort` objects in order to hold information about all the Arduino Unos with which the Raspberry Pis communicate. The UML diagram for `SerialPortDict` can be seen in Figure 2.8.

### CatoptricController Files

The `CatoptricController.hpp` and `CatoptricController.cpp` files outline the structure and actions of the `CatoptricController` class. The `CatoptricController` has the responsibility of ensuring that the user can interface with the `CatoptricSurface` object. Its UML diagram can be seen in Figure 2.2 below.

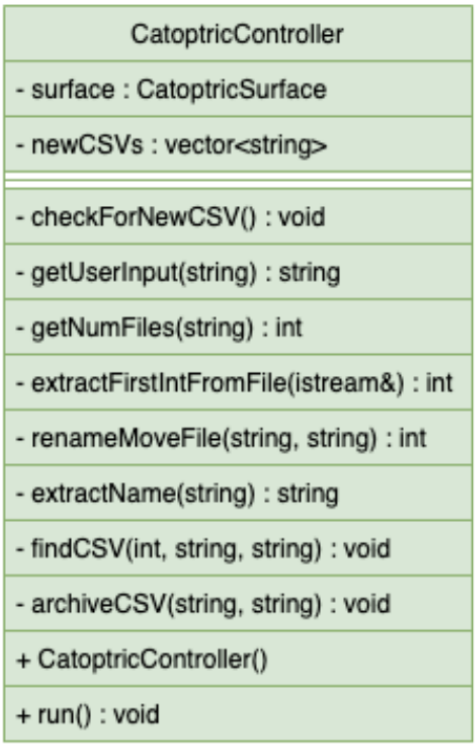


Figure 2.2: Catoptric controller.

The member variables present in the `CatoptricController` class are a `CatoptricSurface` object and a vector of strings called `newCSVs`. The `CatoptricSurface` object member variable is what the `CatoptricController` is in charge of interfacing with. The `newCSVs` vector of strings holds the names of any updated CSV files that the user has provided as input into the system.

The `CatoptricController.cpp` file contains the `main()` method where the control flow path starts for the system. Its `main()` function calls its `run()` member function on a `CatoptricController` object. This `run()` function continually (in an infinite `while()` loop) prints out the options that the user has for commands to enter, checks if there are any new CSV files with its `checkForNewCSV()` function, polls for which command the user enters, and calls any other needed methods in the program according to the user input provided.

The user input can take the form of a CSV file or of the following commands entered through the command line: `reset`, `test`, `check`, and `quit`. Additionally, if there are new CSV files detected in the file system, then the user can also enter the command `run` followed by the

name of an available CSV file to have the mirrors go through the movements configured in that named CSV file.

Since the `CatoptricController` has the responsibility of interfacing between the user and the `CatoptricSurface`, and because one of the forms of user input is CSV files that hold information on how the motors should be moved in order to manipulate the mirrors, the `CatoptricController` needs to be able to detect CSV files, determine which ones have not yet been used, and parse them so that the mirrors can be moved as the user desires. In order to do this, the `CatoptricController` needs to implement certain file features to detect and track the CSV files. To keep track of this information about the CSVs, the `CatoptricController.hpp` file specifies two important file paths. The first is "`csv/archive`" which it saves as the `#define ARCHIVE_DIR`. This is where CSVs that have already been executed are stored. The second path that the `CatoptricController` keeps track of is "`csv/new`" which it saves as `#define NEW_CSV_DIR`. This directory serves as the location where the user of the system can deposit new CSV files that they want to execute.

If the user input is through the form of a CSV file, then the CSV file must be in the form of four, comma-separated integer values per line. The first of these four integers represents the row number which is targeted. The second represents the column number of the mirror which we are targeting. The third of these integers represents the motor number that we would like to move for that mirror (this determines whether the mirror motion will be pan or tile). The fourth number in each row of a user-generated CSV will represent the number which we want to move the stepper motor.

When the user enters the `run` command followed by a filename which represents a CSV that they added to the system, the first thing that the `CatoptricController` does is ensure that the CSV name that they enter actually exists on the system. This is done by calling the `CatoptricController's findCSV()` function. The `findCSV()` function cycles through the vector of CSV files that reside on the system but have not yet been run. These files have been saved on the `newCSVs` member variable every time that `checkForNewCSV()` function was called.

After ensuring that the CSV exists on the system as a file that has not yet been run, the system constructs a C++ string with the file path for the CSV, uses this string to erase the CSV filename from the `newCSVs` vector member variable, and uses the `CatoptricController's CatoptricSurface` member variable to call the `CatoptricSurface` class's `updateByCSV()'s`

function. Once the `CatoptricSurface's updateByCSV()` has been called, the `CatoptricController's archiveCSV()` function is called. The `CatoptricController's archiveCSV()` function renames the CSV based on how many other files have already been archived, and moves the file into the `ARCHIVE_DIR` directory. This software functionality of interacting with new CSV files plays a crucial role in the design of the system because it ensures that the system can properly receive user configurations without repeating movements or accidentally trying to use an invalid CSV input.

When the user enters the string `quit` as a command for the system, the `CatoptricController's CatoptricSurface` member variable is used to call the `CatoptricSurface` class's `cleanup()` function. This functionality is important to ensure that the user has a way of closing out of the system.

When the user enters the string `check` as a command for the system, the `CatoptricController's updateByCSV()` function is called to see if any newer CSV files have been added to the system since the last call. This functionality gives the user of the system more control over when the system can discover the new CSV files that they have added, which then allows them more control over when they can run new CSV file configurations with the mirrors. Without this command, the user has to wait for the `checkNewCSV()` function to be called in the `CatoptricController's run()` function, something that is automatically handled by the C++ code on the system and thus the user doesn't posses any explicit control over.

When the user enters the `reset` command, the `CatoptricController's CatoptricSurface` member variable is used to call the `CatoptricSurface` class's `reset()` function with the `REGULAR_RESET` parameter. This allows the system to be reset to its default position. In addition to it being a useful benefit to the users of the system to have a default reset positioning for the system, it also plays a vital role in the correct functioning of the system.

Currently, there is no way for the user to know the positioning of the mirrors on the system. Since the movement of the mirrors is directed based on how many steps the stepper motor should take rather than a coordinate system that specifies absolute positioning, the initial positioning of the mirror impacts the commands that the user should be entering into the Catoptric Surface. By giving the user the option to use a `reset` command to move the mirrors to a default position, we allow the user to know the initial positioning of the mirrors, thereby allowing the user to know which commands they should use for running the motors to get the system into the desired position.

When the user enters the `test` command, the `CatoptricController's CatoptricSurface` member variable is used to call the `CatoptricSurface` class's `reset()` function with the `TEST_RESET` parameter. This allows the system to go through a particular sequence of movements which allows it to test all of its motors. This feature allows the user the ability to debug motor problems that could occur with the Catoptric Surface.

### CatoptricSurface Files

The `CatoptricSurface.hpp` and `CatoptricSurface.cpp` files outline the structure and actions of the `CatoptricSurface` class. The `CatoptricSurface` class represents the array of mirrors (and their accompanying motors) as well as the serial ports which are used to communicate with the Arduinos. The UML class diagram for the `CatoptricSurface` can be seen in Figure 2.3.

As seen from the UML class diagram of the `CatoptricSurface` class in Figure 2.3, the `CatoptricSurface` contains the following member variables: an `int` called `numRowsConnected`, a `SerialPortDict` object called `serialPortOrder`, a `SurfaceDimensions` object called dimensions, a vector of `CatoptricRow` objects called `rowInterfaces`, a vector of `SerialPort` objects called `serialPorts`, a vector of strings that holds the CSV file data called `csvData`, and a string called `SERIAL_INFO_PREFIX`.

In order to run commands from a CSV file, after the `run` command and its accompanying CSV filename is parsed out by the `CatoptricController`, control goes to `CatoptricSurface's updateByCSV()` function. The first thing the `updateByCSV()` function does is read in data from the CSV using the `CatoptricSurface's getCSV()` function which clears everything from the old read, reads data from a CSV line by line, and then adds all the input reads to the `csvData` vector. Next, the serial buffer is reset for each of the `CatoptricRows` in the `rowInterfaces` member variable, each line of the CSV is parsed, and a message object is constructed to represent the desired movement in each line. The UML diagram for `Message` is seen in Figure 2.4. Then the rows stored in `rowInterfaces` are iterated through, and the message is sent to the correct row by calling the `CatoptricRow's reorientMirrorAxis()` on the `CatoptricRow` with the message object as a parameter. If the appropriate row is found, the `CatoptricSurface` class's `run()` function can be called. The `CatoptricSurface's run()` function handles tracking information for each row's FSM

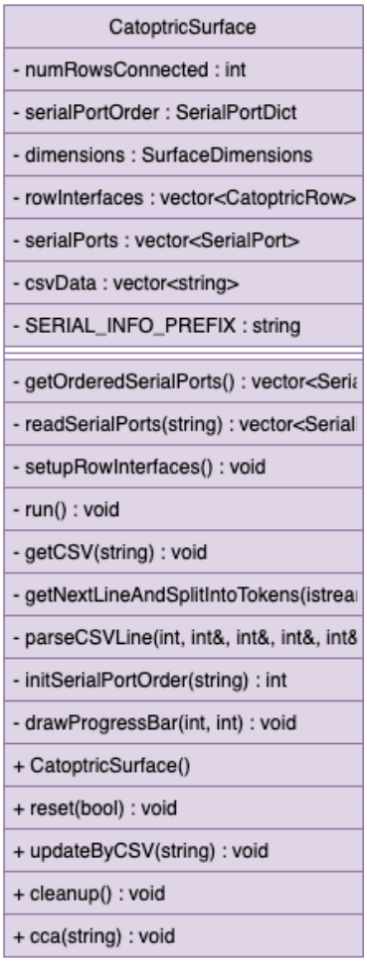


Figure 2.3: Catoptric surface.

and calls the `CatoptricRow's update()` function on each of the `CatoptricRow` objects in the `rowInterface` member variable.

If the user issues the `quit` command, control ends up at the `CatoptricSurface` class's `cleanup()` function. The `CatoptricSurface` class's `cleanup()` function calls the `CatoptricRow` class's `cleanup()` function on each of the `CatoptricRow` objects in the `rowInterface` member variable.

If the user issues the `reset` command, control ends up at the `CatoptricSurface` class's `reset()` function. The `CatoptricSurface` class's `reset()` function calls the `CatoptricRow` class's `reset()` function on each of the `CatoptricRow` objects in the `rowInterface` member variable and then calls the `CatoptricSurface` class's `run()` function which handles the


Figure 2.4: Message UML diagram.

process of keeping track of the FSM used for communicating with the Arduinos and calls `update()` for each of the `CatoptricRow` objects.

If the user issues the `test` command, the same actions occur in the `CatoptricSurface` class, but the `reset()` function is called with a different parameter that represents that the command entered was `test` rather than `reset`.

### CatoptricRow Files

The `CatoptricRow.hpp` and `CatoptricRow.cpp` files outline the structure and actions of the `CatoptricRow` class. The `CatoptricRow` class represents a row of mirrors in the Catoptric Surface (and their accompanying motors) as well as the `serialFSM` that it uses to communicate with its Arduino. The UML class diagram for the `CatoptricRow` can be seen in Figure 2.5.

As seen from the UML class diagram of the `CatoptricRow` class in Figure 2.5, the `CatoptricRow` contains the following member variables: an `int` called `serial_fd` to represent the file descriptor of the serial port written to for the row's Arduino, an `int` called `rowNumber` to represent the row number which the current `CatoptricRow` object corresponds to, an `int` called `numMirrors` which represents the number of columns/mirrors in the `CatoptricRow` object's row, a vector of `MotorState` objects called `motorStates` (whose UML diagram can be seen in Figure 2.6), a vector of `Message` objects called a `commandQueue` which represents

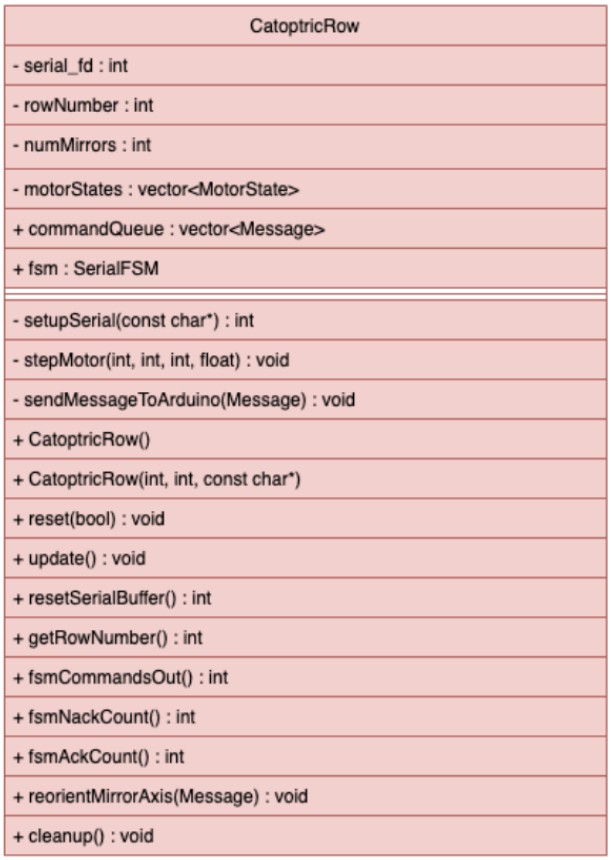


Figure 2.5: CatoptricRow UML diagram.

a list of the movements that the mirrors need to use, and a `serialFSM` object called `fsm` which represents the finite state machine used for communicating between the Raspberry Pi and this `CatoptricRow's` Arduino Uno.

In order to run commands from a CSV file, after the `run` command and its accompanying CSV filename is parsed out, control ends up in the `CatoptricRow's reorientMirrorAxis()` function where it updates some bookkeeping information about the current motor state and calls the `stepMotor()` function, where the command is pushed onto the `commandQueue`. The `commandQueue` has `Message` objects popped off and sent to the row's corresponding Arduino every time the `CatoptricRow's update()` function is called. Since the system loops through the rows and calls `update()` for every `CatoptricRow` whenever the `CatoptricSurface` runs, `update()` is able to execute those messages and move the mirrors. After sending the message to the Arduino, the `CatoptricRow's update()` function then waits to receive information

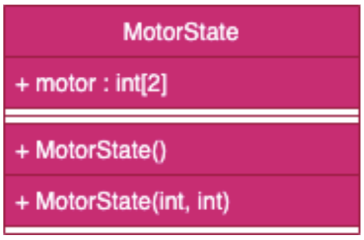


Figure 2.6: MotorState UML diagram.

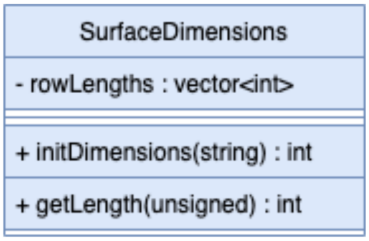


Figure 2.7: SurfaceDimensions UML diagram.

back from the Arduino so that it can update the row's FSM by calling `Execute()` on the `fsm` member variable.

When the user issues the `quit` command, control eventually ends up at the `CatoptricRow` class's `cleanup()` function. Here, the system calls `resetSerialBuffer()` to ensure that the serial buffer between the Raspberry Pi and the Arduino for which the particular row corresponds is clear for next time. The system also closes the `serial_fd` file descriptor and calls `clearMsg()` on the `fsm` member variable. This part of the system is vital for ensuring that things are always closed out smoothly so that old data will not cause issues in future runs of the system.

After issuing the `reset` command, control eventually ends up at the `CatopricRow` class's `reset()` function. The `CatoptricRow` `reset()` method calls `stepMotor()` with the appropriate parameters to move the mirror to a pre-programmed, default position. The `reset()` method takes a `bool` parameter whose value is dependent upon whether the user input the command reset or the command test into the system. This parameter is used to appropriately determine the default configuration that the program should be set to.

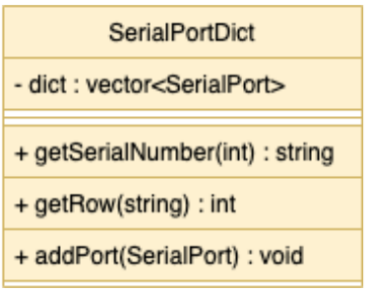


Figure 2.8: SerialPortDict UML diagram.

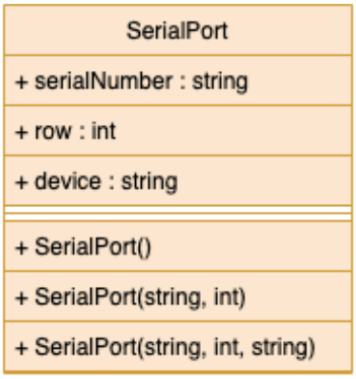


Figure 2.9: SerialPort UML diagram.

### 2.2.2 Arduino Communication

#### SerialFSM Files

In order for the Raspberry Pi to communicate with the Arduino, a Finite State Machine was implemented in order to track at which point in the process the Raspberry Pi is at sending and receiving the data. This Finite State Machine is comprised of the states: `GET_MAGIC_NUM`, `GET_KEY`, `GET_NUM_CHAR_HIGH`, `GET_NUM_CHAR_LOW`, `GET_CHAR`, `GET_ACK_KEY`, `GET_ACK_X`, `GET_ACK_Y`, `GET_ACK_M`, and `GET_NACK_KEY`. These states can be seen in the Finite State Machine diagram in Figure 2.10. The system moves between these states depending on if it gets certain pieces of data that compose the magic number, the key, and specific chars that tell the Raspberry Pi code to switch states.

The Arduino C code for each row in the Catoptric Surface was then responsible for receiving the messages sent by the Raspberry Pi about how to tell the mirrors to move. After receiving

the data, it parses out the information it needs about moving the mirrors, and sends back information to the Raspberry Pi so that the Raspberry Pi can update the Serial FSM as needed. The Arduinos were wired up to the stepper motors to move them appropriately.

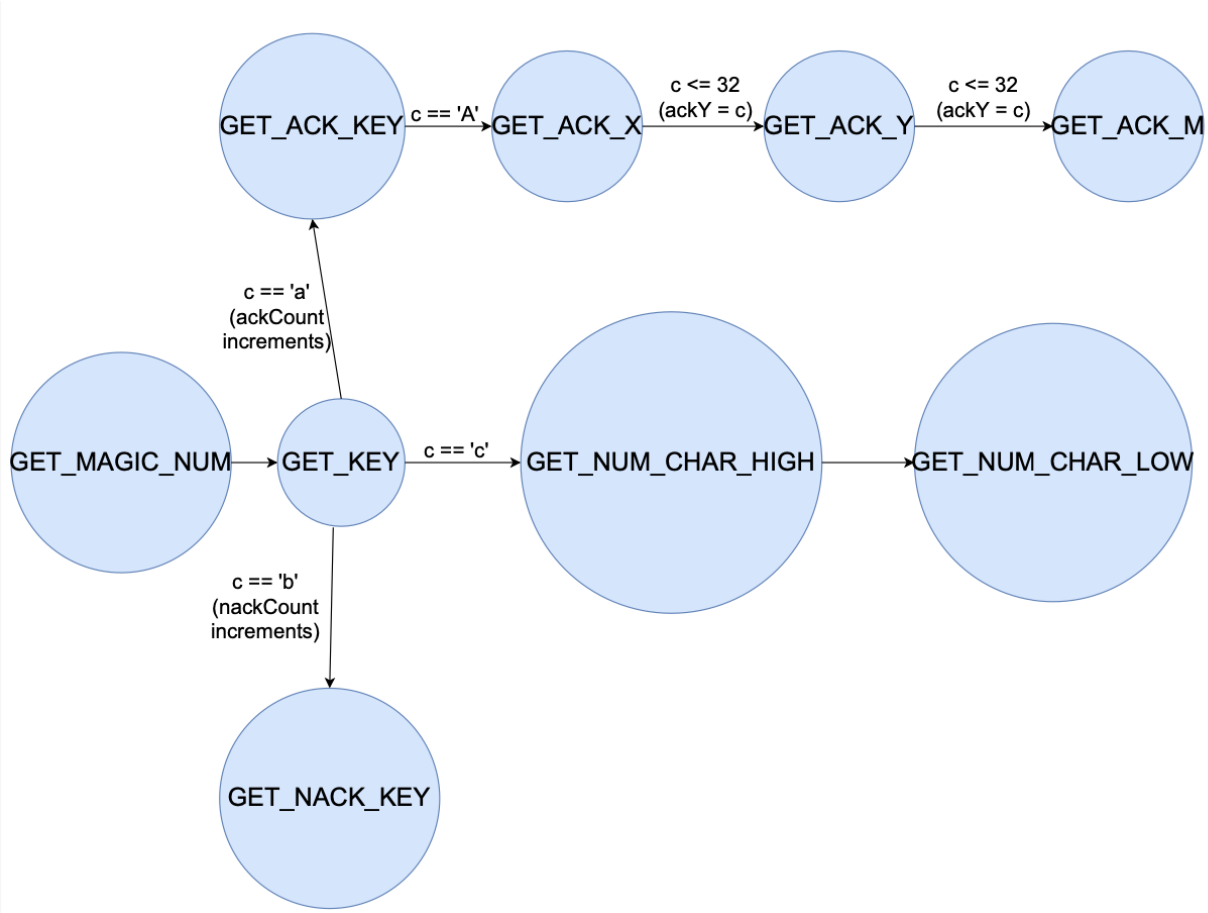


Figure 2.10: FSM diagram for the Catoptric Surface.

## 2.3 Working with a 2 x 2 Array of Mirrors

In order to get the C++ code on the Raspberry Pis working, we ran it on a test setup of 4 mirrors (see Figure 2.11). The 4 mirrors were in a 2 by 2 configuration with 2 Arduinos total (1 per row) and with 1 Raspberry Pi communicating to the Arduinos. In order to get the system working on the 2 by 2, a delay had to be added in between the `Message` objects that the Raspberry Pi sent to the Arduinos. This delay had to be added because without it, the first `Message` object would be sent and successfully move the first mirror, but

all the subsequent `Message` objects were not properly being received. Since the subsequent `Message` objects were not being received, only the first mirror was moving and the rest of the commands that the user requested were not being acted upon. From looking at the serial communication between the Raspberry Pi and the Arduinos with an oscilloscope, we saw that there was an issue with the messages being sent too close to each other, and so the second message onwards was not able to be properly processed. To remedy this, a for-loop style delay was added before the `Message` object is sent to the Arduino in the `CatoptricRow's update()` function. While this delay caused a noticeable wait time between mirror movements, it allowed for all the movements issued by the user to be sent and successfully move the mirrors on the 2 by 2 test setup.

Figure 2.11: 2 x 2 test setup.

# Chapter 3

# Experimental Setup and Results

## 3.1 Experimental Setup

To measure the illuminance from the Catoptric Surface, we took light sensor measurements in Steinberg with the mirrors of the Catoptric Surface in different configurations. In this experiment, the primary independent variable was the configuration of the mirrors, and the dependent variable of interest was the illuminance measurements from the light sensor. The measurements were taken on Friday, November 18th between 11 am and 12:30 pm. The weather during the time period of the experiment was partly cloudy which gave off enough natural light to carry out the experiment. During the actual measurements recorded below, the sun was not occluded at all (full sun, no clouds). This time of day was chosen because around noon is when the Sun is highest above the local meridian.

The light sensor used was the TSL2561 as seen in Figure 3.1. This was connected to a SparkFun ESP32 Thing Plus microcontroller in order to be able to read and log light sensor measurements periodically. The ESP32 was chosen due to several of its features that made it beneficial for this experiment. One of these features was the Qwiic connect system, a way of easily wiring the SparkFun ESP32 Thing Plus to sensors without having to solder the pins on the ESP32 board itself. Another benefit of choosing the ESP32 is that its code can be written in Arduino C, which makes it simple to program. The `SparkFunTSL2561.h` header file was added to the Arduino C file to allow the ESP32 to smoothly interface with the TSL2561 sensor.

In order to save the light sensor measurement data, the information was printed to the Serial Monitor in the Arduino IDE. Then, the application CoolTerm was used to listen at the serial port that was used to connect the ESP32 and obtain the data that was going to be printed to the Serial Monitor. Since CoolTerm uses the serial port to obtain this data, the ESP32

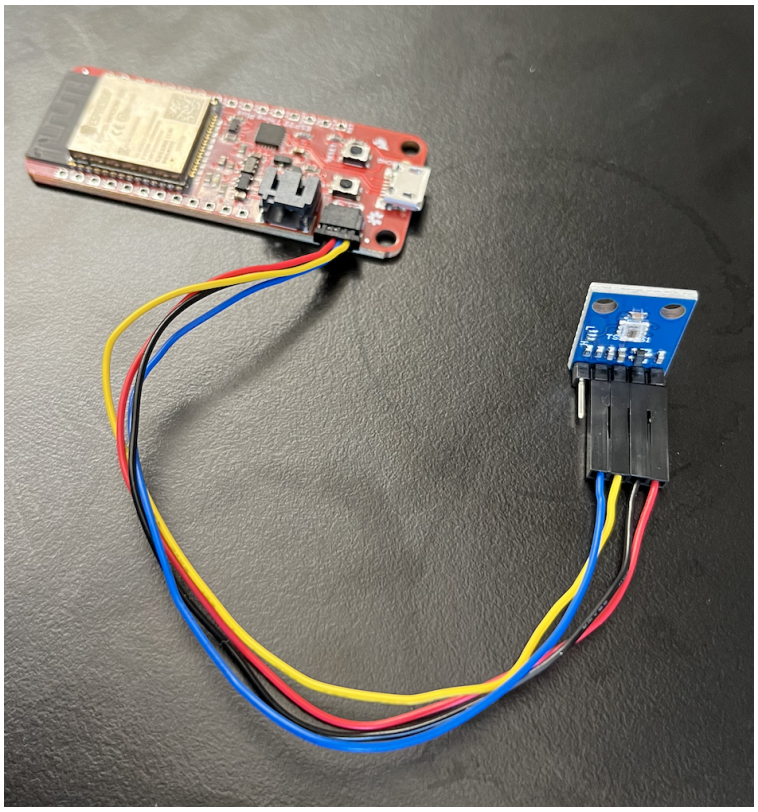

Figure 3.1: ESP32 and TSL2561.

needs to be plugged into the computer on which CoolTerm is running to be able to obtain the information. The settings in CoolTerm were also used to prepend the date and timestamp to the light sensor data in order to know when the light sensor measurements were taken with an increased level of precision. In order to save the data and its associated timestamps once it was available in CoolTerm, it was written to a text file.

While this setup was sufficient for the purposes of the experiment carried out, another data collection system was designed for a longer term experiment in which data collection could take place over a period of several days or weeks. For this setup, the application Node-Red was installed onto a Raspberry Pi. Node-Red is a web-browser application that the user can configure to allow the device on which it's running to perform different actions. In this setup, we configured Node-Red to be able to receive POST requests from the ESP32. Since the ESP32 has Bluetooth and WiFi capabilities, it is able to send POST requests. The POST requests that the ESP32 sends to the Raspberry Pi contain the timestamp and light sensor measurement information. Node-Red on the Raspberry Pi can then be used to automatically

write the received data to a text file so that it can be saved long-term. Since our experiment took light sensor measurements for approximately an hour and a half long timeframe, we were able to monitor the light sensor during the duration of the measurements and thus it did not warrant using this longer-term setup. However, an experiment performed in the future over an extended period of time where humans cannot monitor the setup during the entire duration of measurement can consider this approach.

## 3.2 Mirror Configurations

In order to move the mirrors, a group of individuals changed the positioning of the mirrors by hand because the code moving the mirrors on the 2 by 2 test-setup does not yet move the mirrors on the full Catoptric Surface in Steinberg. When running the actual experiment, illuminance measurements were taken with the light sensor when individuals were in the process of moving mirrors to a particular position, as well as once a particular configuration was set up. This allows us to capture not only a particular configuration's illuminance, but also to see how light changes as parts of the Catoptric Surface are transitioning.

The light sensor was placed on a table in the middle of the atrium in Steinberg (the building where the Catoptric Surface resides). It is important to note that disturbances to the light pattern may have occurred during portions of the experiment, such as the occasional cloud coverage or a person walking by the sensor and casting a shadow.

The configurations which we moved the mirrors to and took measurements at are as follows:

- All of the mirrors pointing upwards to try and concentrate the lights on a particular spot on the ceiling directly above the light sensor (Ceiling Config.). When in this configuration, the image of the ceiling of Steinberg is shown in Figure 3.2.
- All of the mirrors pointing downwards (Floor Config.). When in this configuration, the image of the ceiling of Steinberg is shown in Figure 3.3.

Figure 3.2: The Steinberg ceiling with the mirrors facing upwards.

Figure 3.3: The Steinberg ceiling with the mirrors facing downwards.

## 3.3 Results

Tables 3.1 through 3.6 and Figures 3.4 through 3.7 give the numerical measurement results of the experiment.

Table 3.1: Average Light Measurements for Various Configurations

| **Mirror Configuration** | **Average Illuminance Data (lux)** |
|---|---|
| Ceiling Config. | 3828.669 ± 8.395 |
| Floor Config. | 3131.178 ± 23.371 |

Table 3.2: Mode of Light Measurements for Various Configurations

| **Mirror Configuration** | **Mode of Illuminance Data (lux)** |
|---|---|
| Ceiling Config. | 3938.64 |
| Floor Config. | 3220.03 |

Table 3.3: Median of Light Measurements for Various Configurations

| **Mirror Configuration** | **Median of Illuminance Data (lux)** |
|---|---|
| Ceiling Config. | 3829.27 |
| Floor Config. | 3210.32 |

Table 3.4: Range of Light Measurements for Various Configurations

| **Mirror Configuration** | **Range of Illuminance Data (lux)** |
|---|---|
| Ceiling Config. | 417.92 |
| Floor Config. | 1209.92 |

Table 3.5: Variance of Light Measurements for Various Configurations

| **Mirror Configuration** | **Variance of Illuminance Data ($lux^2$)** |
|---|---|
| Ceiling Config. | 4766.3572 |
| Floor Config. | 36939.0284 |

Table 3.6: Standard Deviation of Light Measurements for Various Configurations

| **Mirror Configuration** | **Standard Deviation** |
|---|---|
| Ceiling Config. | 69.0388 |
| Floor Config. | 192.195 |

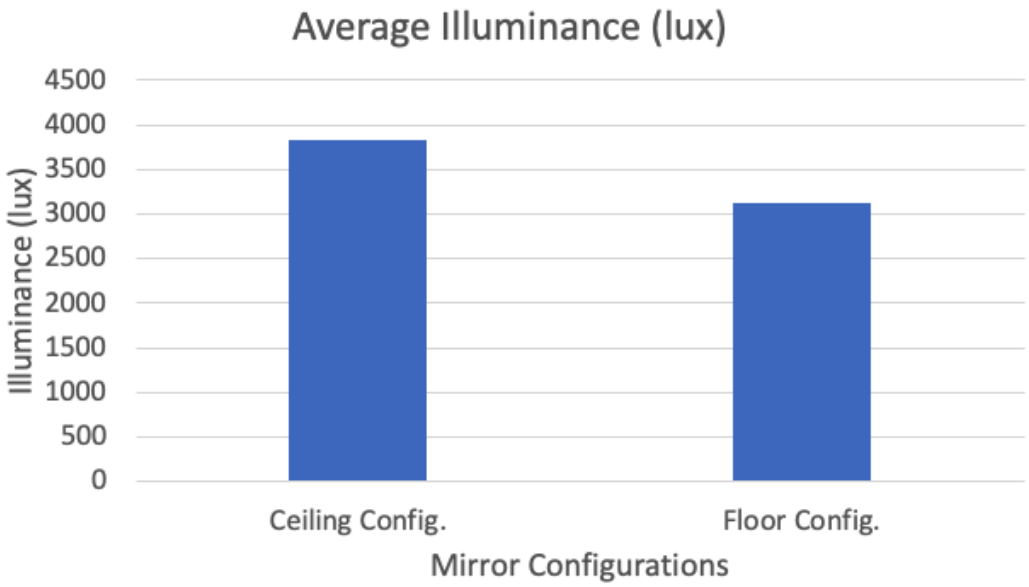


Figure 3.4: Average illuminance for the configurations.

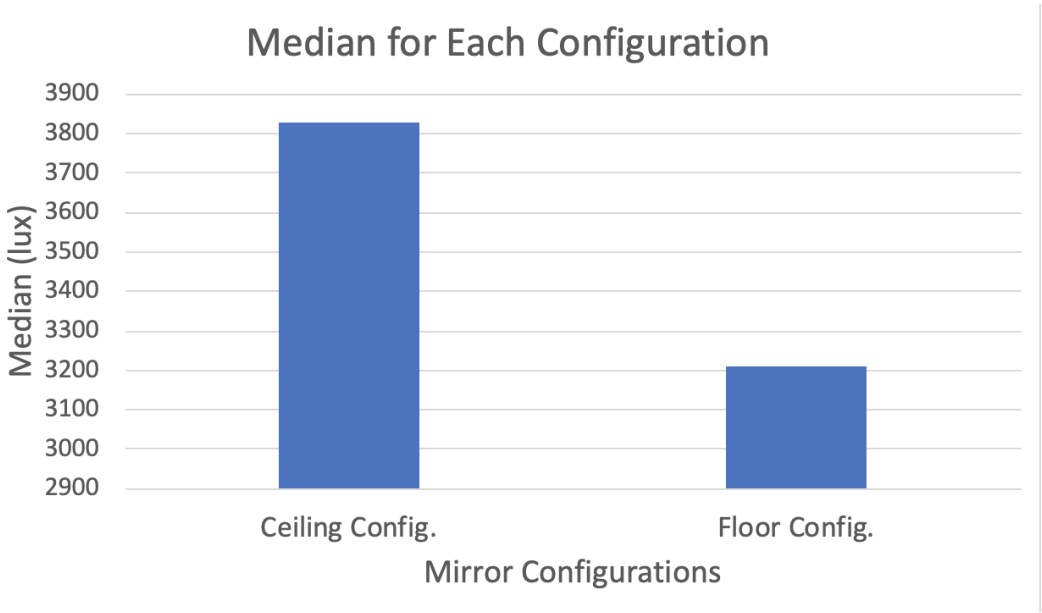


Figure 3.5: Median illuminance for the configurations.

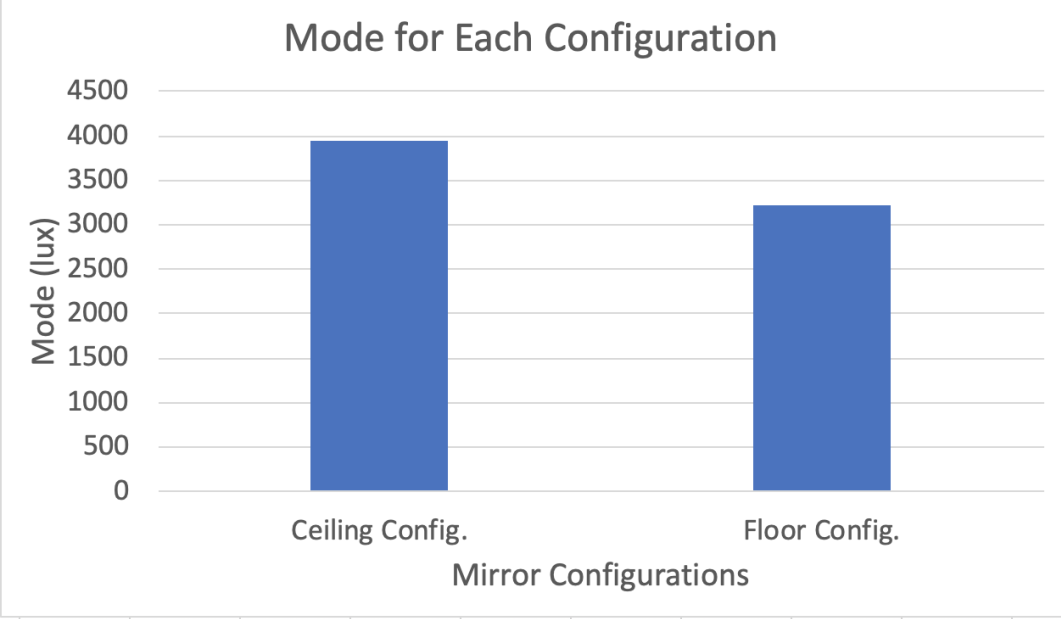


Figure 3.6: Mode illuminance for the configurations.

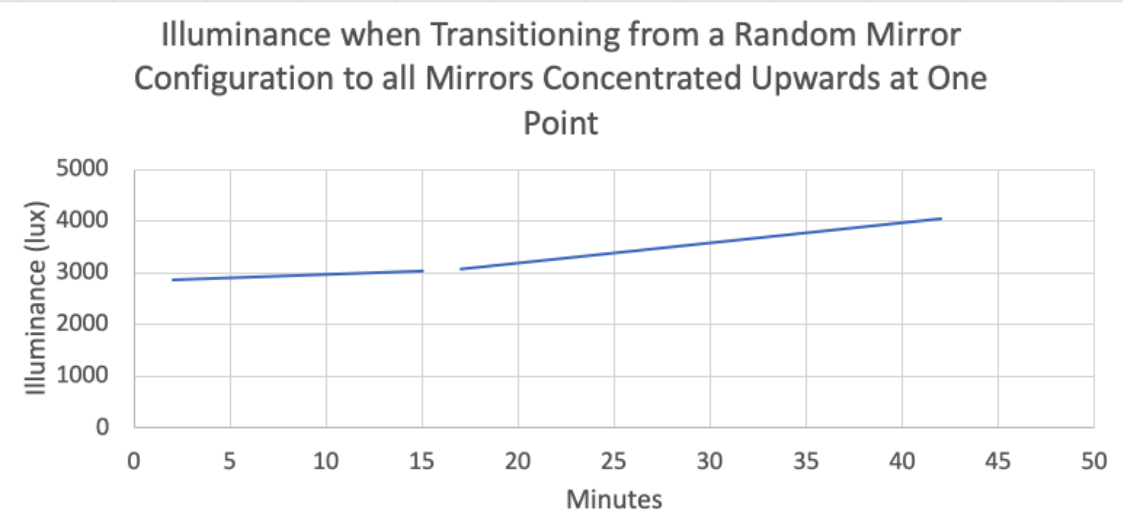


Figure 3.7: Illuminance when transitioning configurations.

## 3.4 Discussion of Results

The illuminance measurements from the time period in which the mirror configuration was being manually changed from its original position to the position in which the mirrors were pointed upwards to concentrate light at one spot on the ceiling demonstrated that as we moved more mirrors upwards the illuminance increased. This can be seen from the data in Figure 3.7.

The average illuminance has a higher value in the mirror configuration where all the mirrors are pointed to one spot on the ceiling above the light sensor than it does with the configuration where the mirrors are pointed towards the ground. It is important to note that disturbances to the light pattern may have occurred during portions of the experiment, such as the occasional cloud coverage or a person walking by the sensor and casting a shadow. In order to account for this we found the margin of error for the average illuminance from the Student's t distribution. This margin of error can be seen from the error bars in Figure 3.4.

The median, which is graphed in Figure 3.5, confirms this result as the median light sensor measurement for the configuration of mirrors pointed at the ceiling is higher than the median light sensor measurement for the configuration of mirrors pointed at the ground. The results of the mode for each configuration can be seen in Figure 3.6.

From looking at the standard deviation and variance for each of the configurations, we see that the standard deviation and variance for the configuration in which the mirrors are pointed at the ground is higher than it is for the configuration in which the mirrors are pointed upwards to concentrate light at one point on the ceiling. This demonstrates that there was more variability with the light sensor measurements for the configuration in which the mirrors are pointed at the ground. This increased variability could be caused by factors such as cloud coverage changing the amount of natural light entering Steinberg or people walking around the light sensor and casting a shadow. This higher variability results in a larger range for the configuration in which the mirrors are pointed at the ground. Overall, we see that the mirrors on the Catoptric Surface play an effective role in redirecting natural light to modify the illuminance of a space because the mirrors had a higher illuminance measurement when concentrated at one spot on the ceiling then when compared to being in a configuration in which all the mirrors are pointed at the ground.

# Chapter 4

# Conclusions and Future Work

Overall, throughout the lifetime of this project, we looked at an environment that we wanted to redirect light in and assessed which system we wanted to test out based on knowledge of existing light redirection systems as well as the requirements we had for manipulating light in the Steinberg atrium. We then built the Catoptric Surface to achieve these goals. Then, we developed C++ code, Arduino C code, and a system of communicating between those two microcontrollers to move the mirrors. This code facilitated breaking down the system into distinct parts. For example the CatoptricController was used for managing the relationship between the use and the system, the CatoptricSurface class was responsible for representing the Catoptric Surface as a whole, and the CatopricRow class was responsible for representing a single row on the Catoptric Surface.

We used this code to get a working 2 by 2 testing setup of 4 mirrors in the lab. This demonstrated that the Raspberry Pi and Arduino setup could be successful for controlling the mirrors on the Catoptric Surface. Then, we tested the effectiveness of the mirrors on the system at redirecting light by putting the mirrors on the CatoptricSurface in different configurations and measuring the resulting illuminance from the configurations as well as during the transitions to/from specific configurations. These measurements demonstrated that the mirrors on the Catoptric Surface were able to be positioned to redirect light as desired to increase illuminance in the desired environment.

Future work for the system as a whole could entail modifying the code on Raspberry Pis for the full Catoptric Surface to see if the delay added on the 2 x 2 has an impact on moving the Catoptric Surface in Steinberg. Additionally light sensor measurements could be taken over a longer period of time to obtain more data. Light sensor measurement over a longer period of time would also allow us to measure different variables such as the effects that varying weather patterns, different times of day, and different times of year have on illuminance. In order to obtain longer term light sensor measurements that require a measurement system

to be running without human presence, the data collection system that uses Node-Red as outlined in Chapter 3 can be utilized.

# References


[1] Chandler Ahrens, Roger Chamberlain, Scott Mitchell, Adam Barnstorff, and Joshua Gelbard. Controlling daylight reflectance with cyber-physical systems. In *Proc. of 24th International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA)*, volume 1, pages 433–442, April 2019.

[2] Chandler Ahrens, Roger D. Chamberlain, Scott Mitchell, and Adam Barnstorff. Catoptric surface. In *Proc. of 38th Conf. of Association for Computer Aided Design in Architecture (ACADIA)*, pages 216–225, October 2018.

[3] Magali Bodart and André De Herde. Global energy savings in offices buildings by the use of daylighting. *Energy and Buildings*, 34(5):421–429, 2002.

[4] Roger D. Chamberlain, Chandler Ahrens, Christopher Gill, and Scott A. Mitchell. Work-in-progress: Hierarchical control of a catoptric surface. In *Proc. of International Conference on Embedded Software (EMSOFT)*, October 2018.

[5] Jens Christoffersen and Kjeld Johnsen. An experimental evaluation of daylight systems and lighting control. In *Proceedings of Right Light 4, 4th European conference on energy-efficient lighting*, volume 2, pages 245–254, 1997.

[6] L Edwards and P Torcellini. Literature review of the effects of natural light on building occupants. Technical Report NREL/TP-550-30769, National Renewable Energy Lab., Golden, CO, USA, July 2002.

[7] MG Figueiro, M Kalsher, BC Steverson, J Heerwagen, K Kampschroer, and MS Rea. Circadian-effective light and its impact on alertness in office workers. *Lighting Research & Technology*, 51(2):171–183, 2019.

[8] MG Figueiro and MS Rea. Office lighting and personal light exposures in two seasons: Impact on sleep and mood. *Lighting Research & Technology*, 48(3):352–364, 2016.

[9] F. Harb, M. P. Hidalgo, and B. Martau. Lack of exposure to natural light in the worskpace is associated with physiological, sleep and depressive symptoms. In *Chronobiology International vol. 32, no. 3*, page 368–375, 2015.

[10] Pyonchan Ihm, Abderrezek Nemri, and Moncef Krarti. Estimation of lighting energy savings from daylighting. *Building and Environment*, 44(3):509–514, 2009.

[11] M. Konstantoglou and A. Tsangrassoulisd. Dynamic operation of daylighting and shading systems: A literature review. In *Renewable and Sustainable Energy Reviews*, volume 60, pages 268–283, 2016.

[12] F. Leslie. Capturing the daylight dividend in buildings: why and how? In *Building and Environment vol. 38*, page 381–385, 2003.

[13] Anna Pellegrino, Silvia Cammarano, Valerio RM Lo Verso, and Vincenzo Corrado. Impact of daylighting on total energy use in offices of varying architectural features in Italy: Results from a parametric study. *Building and Environment*, 113:151–162, 2017.

[14] Ayat Mohammadi Tabar, Mohammad Moradi, and Rima Fayaz. Effects of interior architecture for optimal use of natural light and electrical energy saving. *Energy Sources, Part A: Recovery, Utilization, and Environmental Effects*, 2019.

**Catoptric Surface Light Measurements, Kodali, M.S. 2022**