MSE 4499 - Mechatronic Design Project

Smart Home Window

by

Douglas Cheng

Xingmin Zheng

Pahuldeep Mangat

Keeshigan Pirabaharan

Final Report

Faculty Advisor: Dr. Ladak

Date Submitted: April 15th, 2022

Word Count (excluding front matter, figures, tables and references: 7453/7500

Executive Summary

Current conventional windows for living spaces must be opened and closed manually which can be a very tedious low-level task. Home windows along with its coverings such as blinders and curtains allow occupants to control the lighting and temperature of the home using natural factors. This is a much more cost-effective and eco-friendly way to do so rather than using HVAC systems to heat/cool the home and light bulbs to illuminate the home. Although these devices use more energy and cost more, they are more convenient as they can be controlled with a click of a button. For example, an occupant may rather turn on the AC to cool the home instead of opening each individual window in a home on a summer evening or turn on the lights in a room instead of drawing each window curtain in the morning. If operating windows were as convenient and automated as modern HVAC systems, it could lead to potential energy and cost savings. Occupants also often forget to close windows once they have opened them. Leaving the window open during unwanted weather conditions can cause damage to items in the home that are close to windows and also creates a security risk

Due to the tedious nature of opening and closing windows along with their coverings, there exists a need to develop a device or system that intelligently automates the operation of home windows and its coverings to provide convenience for the occupant while preventing the consequences of leaving them in an unwanted state. This need inspired the concept of *Smart Home Window*.

The goal of *Smart Home Window* is to be a fully automated device or system that can accurately obtain weather data and make the appropriate decisions to control the window and its coverings along with the option of remote control settings for the user. The selected concept to achieve this goal is a motorised sliding window with motorised blinders equipped with various sensors to detect weather changes along with a mobile app for user control. The goal was broken down to multiple subtasks, each with their own concept generation. The design validation proved the selected concept for each subtask is able to achieve the overall goal.

This project resulted in a prototype that demonstrated the functionality of the selected design and a final design that exhibits more of the practicality and marketability of the product. The final design went through many iterations after receiving comments on previous reports and review meetings. Major feedback required us to change the emphasis of the need for this product, change selected concepts and add extra features.

This report further explains the concepts that were generated and eventually selected, the validation and refinement of the design to address the problem, the development, testing and improvements of the prototype, as well as conclusion and recommendations following the completion of the project.

Table of Contents

Introduction

Opening and closing windows along with its coverings such as blinders and curtains allow occupants to control the lighting and temperature of the home using natural factors such as outside temperature, sunlight and wind. This a much more cost-effective and eco-friendly way to do so rather than using HVAC systems to heat/cool the home and lightbulbs to illuminate the home. Although these devices use more energy and cost more, they are more convenient as they can be controlled with a click of a button. For example, an occupant may rather turn on the AC to cool the home instead of opening each individual window in a home on a summer evening or turn on the lights in a room instead of drawing each window curtain in the morning. Current HVAC systems can be connected to thermostats to turn on these systems automatically to achieve the desired home temperature. If operating windows were as convenient and automated as modern HVAC systems, it could lead to potential energy and cost savings.

Occupants also often forget to close windows once they have opened them. Leaving the window open during unwanted weather conditions can cause damage to items in the home that are close to windows. For example, A computer on a desk next to a window can be damaged by rain coming through an opened window. Leaving windows open while no one is home also creates a security risk. ADT Security Services found in a study that ¼ of home break-ins are through the windows of a home [1]. Failing to close windows when HVAC systems are running can also increase energy use for that area by as much as 30% [2].

In conclusion, the use of conventional windows along with their coverings can be a tedious low level task. Leaving windows and window coverings in the incorrect mode can also lead to property damage, safety risks, energy loss, and economic waste. As such there is a need to address this problem using smart technologies to operate windows conveniently and intelligently.

Market

The consumers for this solution are homeowners and home builders. About 69% of Canadians are homeowners [3]. This product would target the homeowners that would like to invest in their homes by replacing their existing windows and reduce their energy reliance. This behaviour is evident in the increasing popularity of solar panel installations. In competitive housing markets, such as in Canada and USA, it would also be a great selling point for real estate developers that would like to market homes with the latest home automation technology.

Problem Definition

Due to the tedious nature of opening and closing windows along with their coverings, there exists a need to develop a device or system that intelligently automates the operation of home windows and its coverings to provide convenience for the occupant while preventing the consequences of leaving them in an unwanted state.

Background Information

Trend of Home Automation Devices

Home automation devices have been becoming increasingly popular within the past decade. These days, there's a smart version of pretty much every home device you can think of. Examples of these devices are robotic vacuums, Nest learning Thermostat, Ring video doorbell, smart locks and many more. In general, these products connect to the internet, so you can control them from your phone. The main objectives of home automation are convenience, safety and energy/cost efficiency. The home automation market was worth US$5.77 billion in 2013, predicted to reach a market value of US$12.81 billion by the year 2020 [4]. This trend is due to the smart technologies made possible by mechatronic engineering advances.

Current Solutions

There are smart home products for different areas and functions around the home, but what devices/systems have addressed the need for intelligent automation for windows and their coverings? Currently, there is no product on the market that controls both the functions of windows and its coverings. Existing products in this realm are simple automated versions of the open/close operation of either a window or window covering. These devices also usually only rely on human input rather than intelligent automation based on sunlight, weather conditions, etc. This addresses the need for convenience of opening and closing a window as pressing a button is much easier than rolling a knob or pushing with force. However, these products do not operate intelligently which would address the need for better safety, energy and economic savings, and better quality of sleep. Below are examples of existing products in the home automation market along with their strengths and shortcomings.

Motorised Blinds from SelectBlinds [5]

Motorised blinds from SelectBlinds allow you to control your window coverings via remote control for fun & simple operation, an example can be seen in Figure 1. SelectBlinds offers their motorised blinds in many style options to choose from. SelectBlinds standard blinds range from CAD$44.99-$629.98 with the motorization add-on feature being CAD$99.99-$224.99 on top of that. Most customers might opt out of getting the morized version of their blinds due to the high price point. This product allows you to pre-set your motorised blinds to open on their own at a designated time in the morning, and close at a pre-set time at night. This product does not offer blind operation based on sunlight or other weather conditions outside. The product also does not operate in accordance with a window. For example, if an occupant wants to open the window and blinds to let in air, they would be able to open the blinds but have to manually open the windows. Most motorised blinds offered by other companies also have the same shortcoming.

Figure 1: Motorised Blinds

Olide Automatic Window Openers [6]

Olideauto has a wide variety of automatic window openers depending on the type of window, size of window, and other features. Prices can vary from CAD$183.40 to CAD$1,128.54. The motorised window openers feature stainless steel chains, which makes the device stronger and prevents oxidation and water damage. In addition to that, most models are low-noise, which means you won’t be distirubed from the sounds of a window being opened or closed. The basic models are more affordable, but can only be operated with a remote control. The advanced models can be controlled via phone and tablet. Once connected to a mobile device, Alexa and Google Assistant can be used to adjust the windows voice control. Similar to the motorised blinds from SelectBlinds, this product does not provide autonomy based on the weather and doesn’t work alongside window coverings. Another flaw of this product is it comes in many parts and can be complicated to install, according to its reviews, and might require a technician for installation. An example of this product can be seen in Figure 2.

Figure 2: Olide Automatic Window Openers

Bridging the Gap

The common technological gap in existing products is the automation element. These products are purely mechanical and electrical systems that do not provide intelligent decisions. Motorised windows and blinds provide convenience over conventional windows and blinds but do not address the potential energy and cost savings, and safety features an automated product would.

In order to bridge this gap, a microcontroller with sensors and actuators can be implemented to these devices. Using the team's strength in mechatronic systems engineering to implement this smart aspect to windows. Strengths in microcontroller programming, sensor and actuator selection and fundamental mechanical and engineering design principles can help bridge this technological gap.

Engineering Tools/Techniques

A 3D assembly model of the final design will be created using Solidworks. Solidworks will also allow us to do static, thermal, and finite element analysis to validate the mechanical design. A mobile app will be made using the Kotlin programming language on Android Studio. A Raspberry Pi will be used to interface the mobile app, sensors and actuators of the device. A software and power analysis will be conducted to ensure the microcomputer and mobile app are feasible. A power cost analysis will be conducted on the device to ensure it achieves the goal of providing energy savings over HVAC systems and electrical lighting. These engineering tools and techniques utilised for design analysis are shown in the Design Validation section.

Scope

The goal of the project is to design and build a fully automated device or system that can accurately obtain input data from various weather sensors and make the appropriate decisions to control the window and its coverings along with the option of remote control settings for the user. This device will eliminate the daily manual labour required to adjust the window and its coverings by automatically opening and closing based on the information extracted from the sensors and online weather data. A prototype was built within the budget ($500 provided + $100 for electrical shop + personal investment) to demonstrate the product's function and form by the end of the academic year.

Objectives and Constraints

To achieve our goal of creating this product, key subtasks were generated in which fulfilling them would result in a fully-functional completed product.

Table 1: Subtasks of Design

Subtask #	Subtask
1	Select a microcontroller/microcomputer that can perform the electrical/software subtasks listed below
2	Determining when sunlight is or is not available to light up the room and adjusting the blinds to the desired position
3	Get temperature data to adjust window position to meet user-specified conditions
4	Get rain data to open and close window
5	Detect when the user is away from the house and lock windows and shut coverings
6	Control the device using a method that is fully functional and accessible
7	Motorizing a sliding window to allow automated adjustments based on decisions made from data inputs
8	Motorizing roller blinds to allow automated adjustments based on decisions made from data inputs

9	Ensure the external sensors work in any weather condition
10	Ensure Fail safe for sliding window to prevent user injuries
11	Emergency override for scenarios that require rapid evacuation such as fires

High-Level Design: Objectives and Constraints

First, high-level objectives and constraints are developed for *Smart Home Windows* (SHW). Listed in Table 2 are the objectives that the high-level design will meet to eliminate the need to manually adjust the window and its coverings to the user's desired state.

Table 2: High-Level Objectives

Objective	Functions
Device must be easy to install and use	● Device should be lightweight ● One-time setup and calibration
Device should be able to operate for a long period of time before replacement	● Actuators used should have a long life cycle ● Gears and other components used should have high fatigue life
Device should look aesthetically pleasing	● Device components should fit within window frame or encased unobtrusively ● App interface to control the device is user friendly and fully functional
Device should not intrude occupants	● Device must operate with minimal noise ● Device should occupy minimal space ● No loose electronics/mechanical components
Device should be cheap	● Device materials should be chosen where there is a balance between function and cost ● Device should be competitive in pricing with regular window and coverings
Device should operate to save money for the user	● Cut down on need of AC/Heating ● Reduce need for electric lighting

Constraints are developed from assessing ways in which the fundamental purpose of the design could not be met. Numerical calculations were done by doing preliminary research to find out the weights of the window and its coverings the device must support. The constraints of the high-level design are listed in Table 3.

Table 3: High-Level Constraints

Category	Constraint
Function/Form	● Device must autonomously adjust window and coverings based on input data received ● Device must have an automatic mode where user presets how windows and coverings are controlled ● Device must have a manual mode for user to control ● Device must be able to open the average home window (sliding window, size: 120 cm x 90 cm) [7] ● Device must be able to partially open the average large window (sliding window, size: 215 cm x 150 cm) [7] ● Parts of the device that are fixed outside the home should survive the harsh weather conditions such as hail, high wind and heavy rain/snow etc. and be waterproof as its intended to be used in any household in the world ● Must meet CSA enclosure Type 4 [8] ● Power supply should have enough power to separately adjust max window load (up to 25 kg) and max covering load (up to 4.5 kg) [8] ● Windows and coverings should not adjust faster than 5 cm/s to ensure safety of occupants near the device (observed maximum speed of competing blinds and windows) [9] ● Device should connect to the internet to allow for connection to user's mobile device ● Device (sensors, actuators and material) should be able to operate within -40°C to 40°C ● Device must be able to be controlled by the user within a 30 m radius of the device
Economic	● Device must not cost more than $200 of the current window and coverings option
Environmental	● Ensure device does not contain or produce hazardous materials ● Ensure device will not create debris or pollutants
Ethical and Legal	● Ensure that the design does not copy any patented designs

Health and Safety	● Device must lock windows to prevent break-ins while occupants are away from the home or asleep / Device should not be overpowered by intruders ● Device must not injure occupants in the vicinity while in operation ● Must be of IEC 61140 Class II [10] ● Meet Canada's Electrical Safety Standard (ESA) ● Window should be able to be manually closed by the user in case of malfunction of the device or power outage (Unnecessary for blinds as malfunction does not impact occupant safety ● Blinds motor has torque under 5 Nm at less than 40 RPM ● Sliding window to be translated at constant speed

Subtasks: Objectives and Constraints

Objectives for each subtask are shown below (Table 4). Constraints for each subtask are shown in Table 5.

Table 4: Subtask Objectives

#	Subtask	Objectives
1	Microcontroller/ Microcomputer	● Fast computing time ● Low Cost ● Long-life ● Power efficient
2	Sunlight Data Input	● Fast response time ● Durable ● Repeatable ● Accurate ● Low Cost ● Long-life
3	Temperature Data Input (Indoor/Outdoor)	● Fast response time ● Durable ● Repeatable ● Accurate ● Low Cost ● Long-life

4	Rain Data Input	● Fast response time ● Durable ● Repeatable ● Accurate ● Low Cost ● Long-life
5	House Occupied/Unoccupied	● Fast response time ● Accurate ● Reliable ● Long-life ● Low Cost
6	Device Control	● Convenient ● User friendly interface ● Long life
7	Motorizing Sliding Window	● Fast response ● Reliable ● Long-life ● Quiet ● Simple Movement ● Inexpensive
8	Motorizing Roller Blinds	● Fast response ● Reliable ● long-life ● Quiet ● Simple movement ● Inexpensive
9	Protection of external sensors	● Long-life ● Reliable ● Accurate ● Inexpensive ● Protection Performance
10	Ensuring Failsafe for Sliding Window	● Fast response ● Reliable ● Robust ● Low Cost
11	Manual Override for Emergency Scenarios	● Fast response ● Reliable ● Robust ● Inexpensive

Table 5: Subtask Constraints

#	Subtask	Constraint
1	Microcontroller/ Microcomputer	● Wi-Fi compatibility ● Have at least 10 data pins for sensor input/ actuator output ● Operate under 12 V to meet IEC 61140 Class II compliance ● No larger than 10 cm x 10 cm x 5 cm
2	Sunlight Data Input	● Detect light intensity ● Sunlight intensity range from 0 to 120 000 lux ● Compatible with the selected microcontroller/ microcomputer ● Accuracy of 10 lux
3	Temperature Data Input (Indoor)	● Temperature range of 10°C to 35°C ● Compatible with the selected microcontroller/ microcomputer ● Temperature accuracy of ≤ 1°C
	Temperature Data Input (Outdoor)	● Temperature range of -40°C to 40°C ● Waterproof ● Compatible with the selected microcontroller/ microcomputer ● Temperature accuracy of ≤ 1°C
4	Rain Data Input	● Works independently of user ● Waterproof ● Compatible with the selected microcontroller/ microcomputer
5	House Occupied/Unoccupied	● Works independent of occupants ● Irrespective of occupants' location in the house ● Works for multiple occupants ● Compatible with the selected microcontroller/ microcomputer
6	Device Control	● Must be able to control the device anywhere within a 10 m radius (average remote control range) [11] ● Compatible with the selected microcontroller/ microcomputer
7	Motorizing Sliding Window	● Must close window securely ● Autonomous movement ● Does not cause damage to window ● Does not obscure window ● Does not close faster than 5 cm/s

8	Motorizing Roller Blinds	• Must close blinds securely • Autonomous movement • Does not affect the shading performance • Does not cause any damage • Does not close faster than 5 cm/s
9	Protection of External sensors	• Does not affect the collection of input data • Ease to produce • Waterproof, dirt resistant and anti-corrosion.
10	Ensuring Failsafe for Sliding Window	• Does not change original function • Does not obstruct user's view of window • Does not require additional power source
11	Manual Override for Emergency Scenarios	• Inaccessible from outside • Does not obstruct user's view of window • Ensures Safe Operation

Concept Generation and Selection

Multiple concepts were developed to achieve each sub-task. Concepts were generated from existing solutions, brain-stormed with inspiration from the content learned during our mechatronics engineering undergraduate and background research done to complete the subtask.

Subtask 1: Microcontroller/Microcomputer

Table 6: Concepts for Subtask 1

Concept	Description
Raspberry Pi 4B	Technical Specifications include: • Built-in Wi-Fi • Quad core ARM v8 @ 1.5GHz • 4GB of RAM • 4 USB ports • 2 HDMIs • 40 GPIO pins

Arduino Mega	Technical Specifications include: ● Wi-Fi shield available ● Clock speed: 16 MHz ● 8KB of RAM ● 56 digital pins ● 16 analog pins

Generated concepts are first compared to input constraints it has to meet in a Go/No-Go matrix (Table 7). Concepts that meet all constraints pass while concepts that do to meet all constraints fail.

Table 7: Go/No-Go Matrix for Subtask 1

Constraints					
Concept	Wi-Fi compatibility	10 Data Pins	12 Volts Limit	Size ≤ 10 cm x 10 cm x 5 cm	Result
Raspberry Pi 4B	Go	Go	Go	Go	Pass
Arduino Mega	Go	Go	Go	Go	Pass

If multiple concepts remain after Go/No-Go screening, they are compared against each other using the objectives the generated concept has to meet. First, each objective is compared against each other using the pairwise comparison method to generate a weighted scale (Table 8). Then the best concept is selected by comparing remaining concepts against the weighted objective (Table 9).

Table 8: Pairwise Comparison for Subtask 1

	Fast Computing Time	**Cost**	**Long-life**	**Power Efficient**	**Total**	**Weight**
Fast Computing Time	1.00	3.00	5.00	3.00	12.00	49.59%
Cost	0.33	1.00	0.33	0.50	2.16	8.94%
Long-life	0.20	3.00	1.00	0.50	4.70	19.42%
Power Efficient	0.33	2.00	2.00	1.00	5.33	22.04%
					24.20	100.00%

Table 9: Decision Matrix for Subtask 1

	Weight	Raspberry Pi	Arduino Mega
Fast Computing Time	49.59%	1	-1
Cost	8.94%	0	1
Long-life	19.42%	1	1
Power Efficient	22.04%	1	1
Total:	100.00%	0.9105937457	0.0081278413

A Raspberry Pi 4B was chosen as the main microcomputer for the SHW device because of the advantage in computing power it has over Arduino. Additional parts such as a Wi-Fi shield has to be purchased to make the Arduino viable for our product which makes it undesirable over the Raspberry Pi.

Subtask 2: Sunlight Level

Table 10: Concepts for Subtask 2

Concept	Description
TSL25911FN Digital Light Intensity Sensor	● Analog Sensor ● Photoresistor based sensor that detects light as well as its intensity up to 120 000 lux
Robojax Light Sensor	● Digital sensor ● Photoresistor with potentiometer to adjust the threshold at which light can be sensed
Online weather API	● Get live and predictive weather data from the internet ● Data obtainable includes: ○ Sunrise/sunset times ○ Light Intensity ○ UV Report ○ Time for which direct sun is out

Table 11: Go/No-Go Matrix for Subtask 2

Constraints					
Concept	Sunlight intensity	Range 0 to 120000 lux *	Raspberry Pi compatible	Accuracy of 10 lux	Result

TSL25911FN Ambient Light Intensity Sensor	Go	Go	Go	Go	Pass
Robojax Light Sensor	No-Go	No-Go	Go	Go	Fail
Online Weather API	Go	Go	Go	Go	Pass

*Selected concepts must detect sunlight to a maximum sunlight brightness of 120000 lux [10]

Table 12: Pairwise Comparison for Subtask 2

	Fast response	Durable	Repeatable	Accurate	Long-life	Cost	Total	Weight
Fast response	1.00	0.33	0.33	0.25	0.50	0.50	2.92	5.76%
Durable	3.00	1.00	2.00	0.50	3.00	2.00	11.50	22.70%
Repeatable	3.00	0.50	1.00	0.33	2.00	2.00	8.83	17.43%
Accurate	4.00	2.00	3.00	1.00	4.00	3.00	17.00	33.55%
Long-life	2.00	0.33	0.50	0.25	1.00	1.00	5.08	10.03%
Cost	2.00	0.50	0.50	0.33	1.00	1.00	5.33	10.53%
							50.67	100.00%

Table 13: Decision Matrix for Subtask 2

	Weight	TSL25911FN Sensor	Online Weather API
Fast response	5.76%	1	1
Durable	22.70%	1	1
Repeatable	17.43%	1	1
Accurate	33.55%	1	0
Long-life	10.03%	1	1
Cost	10.53%	1	0
Total:	100.00%	1	0.5592105263

Hence, the TSL25911FN Ambient Light Intensity Sensor is selected to retrieve sunlight data for the device. This sensor provides more accurate results specific to the house in comparison with online weather API while being able to read light intensity, which is a requirement to adjust blind positions.

Subtask 3: Ambient Temperature Level

Table 14: Concepts for Subtask 3

Concept	Description
DHT 22 (temperature sensor)	● Thermistor based temperature sensor that can detect ambient temperatures from -40°C to 80°C
MLX90614ESF Contactless IR Temperature Sensor	● Contactless IR temperature sensor that reads temperature from -40°C to 85°C in a short distance
Smartphone temperature sensor	● Use temperature sensors installed in the smartphone to send temperature data to the device
Online weather API	● Get local temperature data from online weather APIs

The limiting constraints between the indoor and outdoor temperature sensors were used as the selected sensor would be used to detect indoor and outdoor temperatures to make prototyping simpler.

Table 15: Go/No-Go Matrix for Subtask 3

Constraints					
Concept	Temperature range of -40°C to 40°C	Raspberry Pi compatible	Temperature accuracy of ≤ 1°C	Waterproof	Result
DHT 22 (temperature sensor)	Go	Go	Go	Go	Pass

MLX90614ESF Contactless IR Temperature Sensor	Go	Go	Go	Go	Pass
Smartphone temperature sensor	No-Go	Go	Go	Go	Fail
Online weather API	Go	Go	Go	Go	Pass

Table 16: Pairwise Comparison for Subtask 3

	Fast response	Durable	Repeatable	Accurate	Long-life	Cost	Total	Weight
Fast response	1.00	0.50	0.50	0.33	2.00	2.00	6.33	13.17%
Durable	2.00	1.00	2.00	0.50	3.00	2.00	10.50	21.84%
Repeatable	2.00	0.50	1.00	0.33	2.00	2.00	7.83	16.29%
Accurate	3.00	2.00	3.00	1.00	4.00	3.00	16.00	33.28%
Long-life	0.50	0.33	0.50	0.25	1.00	1.00	3.58	7.45%
Cost	0.50	0.50	0.50	0.33	1.00	1.00	3.83	7.97%
							48.08	100.00%

Table 17: Decision Matrix for Subtask 3

	Weight	DHT 22 Sensor	IR Temperature Sensor	Online Weather API
Fast response	13.17%	1	1	0
Durable	21.84%	1	1	1
Repeatable	16.29%	1	1	1
Accurate	33.28%	1	1	0
Long-life	7.45%	1	1	1
Cost	7.97%	1	0	0
Total:	100.00%	1	0.9202772964	0.4558058925

A temperature sensor outside the window to measure the environment’s ambient temperature and a temperature sensor on the inside to measure house/room temperature are required. The DHT 22 sensor provides the same accuracy as the IR temperature sensor while being cheaper. This is to be used as both the inside and outside temperature sensor.

Subtask 4: Rainfall Level

Table 18: Concepts for Subtask 4

Concept	Description
ASHATA Rain Relay Control Module	● Rain sensor that works by running a voltage through a plate, ● While there is no rain on the plate there is a voltage of 5 V running through the plate ● The plate is capacitive so there is no safety risk of running 5 V through the plate. ● As raindrops hit the plate, the current running through the plate is cut off and voltage levels go down (voltage goes down with amount of rain on the plate)
Rain Gauge	● A device that collects and measures the amount of rain that falls ● This device will need to be emptied in between rainfalls or be designed in a way to empty itself in order to get accurate rainfall data
Online Weather API	● Get live weather forecast data of local area from online weather APIs

Table 19: Go/No-Go Matrix for Subtask 4

Constraints				
Concept	Works independently of user	Waterproof	Raspberry Pi compatible	Result
ASHATA Rain Relay Control Module	Go	Go	Go	Pass
Rain Gauge	No-Go	Go	Go	Fail

Online Weather API	Go	Go	Go	Pass

Table 20: Pairwise Comparison for Subtask 4

	Fast response	Durable	Repeatable	Accurate	Long-life	Cost	Total	Weight
Fast response	1.00	1.00	2.00	0.33	3.00	3.00	10.33	20.98%
Durable	1.00	1.00	2.00	0.50	3.00	2.00	9.50	19.29%
Repeatable	0.50	0.50	1.00	0.33	2.00	2.00	6.33	12.86%
Accurate	3.00	2.00	3.00	1.00	4.00	3.00	16.00	32.49%
Long-life	0.33	0.33	0.50	0.25	1.00	1.00	3.42	6.94%
Cost	0.33	0.50	0.50	0.33	1.00	1.00	3.67	7.45%
							49.25	100.00%

Table 21: Decision Matrix for Subtask 4

	Weight	Rain Sensor	Online Weather API
Fast response	20.98%	1	0
Durable	19.29%	1	0
Repeatable	12.86%	1	1
Accurate	32.49%	1	0
Long-life	6.94%	1	1
Cost	7.45%	1	0
Total:	100.00%	1	0.1979695431

The ASHATA Rain Relay Control Module is selected to retrieve rain data for the device. This rain sensor will provide accurate and reliable results as soon as it starts raining unlike the online weather API that only gives the chance it will precipitate.

Subtask 5: House Occupancy

Table 22: Concepts for Subtask 5

Concept	Description

Location Data on Phone	• App has a location tracker to check if occupants are inside the house or outside the house in order determine if windows should be locked or not
User-Controlled Anti Break-In Mode	• User puts the device into "Anti Break-In" mode and device locks all windows until user turns the mode off
Motion Sensors at the Door of the House	• Motion sensors at the main entrances of the house toggles the device's "Anti Break-In" mode to either lock the windows or not
Motion Detectors on the Blinds	• Motion detecting sensors can be installed on the blinds to detect if any users are within the vicinity to function normally, otherwise it locks the windows

Table 23: Go/No-Go Matrix for Subtask 5

Constraints					
Concept	Works Independent of Occupants	Irrespective of Occupants' Location in the House	Works for Multiple Occupants	Raspberry Pi Compatible	Result
Location Data on Phone	Go	Go	Go	Go	Pass
User-Controlled Anti Break-In Mode	No-Go	No-Go	Go	Go	Fail
Motion Sensors	Go	Go	Go	Go	Pass
Motion Detectors on the Blinds	Go	No-Go	Go	Go	Fail

Table 24: Pairwise Comparison for Subtask 5

	Fast	**Reliable**	**Accuracy**	**Long-life**	**Cost**	**Total**	**Weight**

	response						
Fast response	1.00	0.33	0.50	2.00	2.00	5.83	17.33%
Reliable	3.00	1.00	2.00	3.00	3.00	12.00	35.64%
Accuracy	2.00	0.50	1.00	3.00	3.00	9.50	28.22%
Long-life	0.50	0.33	0.33	1.00	1.00	3.17	9.41%
Cost	0.50	0.33	0.33	1.00	1.00	3.17	9.41%
						33.67	100.00%

Table 25: Decision Matrix for Subtask 5

	Weight	**Location Data**	**Motion Sensors**
Fast response	19.71%	1	1
Reliable	34.62%	1	0
Accuracy	27.40%	1	1
Long-life	9.13%	1	1
Cost	9.13%	1	1
Total:	100.00%	1	0.6538461538

Location data through the smartphone app is selected to let the device know when the house is occupied. Location data is more accurate than other generated concepts while being cheaper to implement and less likely to fail since it is purely software without the need of sensors.

Subtask 6: Device Control

Table 26: Subtask 6 Concepts

Concept	Description
Device Embedded Interface	• Interface on device with screen to display information and buttons for user input
Mobile App	• Mobile app able to be downloaded on IOS and Android devices in order to control the device
Remote Control	• Portable control device with buttons to take user input

Table 27: Subtask 6 Go/No-Go Matrix

Constraints			
Concept	Controllable within 30m radius of device	Raspberry Pi compatible	Result
Device Embedded Interface	No-Go	Go	Fail
Mobile App	Go	Go	Pass
Remote Control	Go	Go	Pass

Table 28: Pairwise Comparison for Subtask 6

	Convenient	**User Friendly Interface**	**Long life**	**Total**	**Weight**
Convenient	1.00	0.50	2.0	3.50	33.33%
User friendly interface	2.00	1.00	2.00	5.00	47.62%
Long life	0.5	0.50	1.00	2.00	19.05%
				10.5	100.00%

Table 29: Decision Matrix for Subtask 6

	Weight	**Mobile App**	**Remote Control**
Convenient	33.33%	1	1
User friendly interface	47.62%	1	0
Long-life	19.05%	1	0
Total:	100.00%	1	0.33

It was determined that the device embedded interface was not a viable interface as it does not allow the user to operate within a 10m radius. A mobile app was chosen because it would have a more user friendly interface and have a longer life as there is no additional hardware component.

Subtask 7: Motorizing Sliding Window

Table 30: Concepts for Subtask 7

Concept	Description
Linear Actuator with Absolute Encoder	• Linear actuator to translate sliding window • Absolute encoder for position control

Linear Actuator with Ultrasonic Feedback	● Linear actuator to translate sliding window ● Ultrasonic Sensor for position control
Linear Actuator with Hall Sensor	● Linear actuator to translate sliding window ● Hall Sensor for closed position
Servo Motor with Lead Screw with Absolute Encoder	● Servo Motor with Lead Screw to translate sliding window ● Absolute encoder for position control
Servo Motor with Lead Screw with Ultrasonic Feedback	● Servo Motor with Lead Screw to translate sliding window ● Absolute encoder for position control
Servo Motor with Lead Screw with Hall Sensor	● Servo Motor with Lead Screw to translate sliding window ● Hall Sensor for closed position

Table 31: Go/No-Go Chart for Subtask 7

Constraints							
Concept	Must close window securely	Auto-nomous movement	Does not cause damage to window	Does not obscure window	5 cm/s speed limit	Position control	Result
Linear Actuator with Absolute Encoder	Go	Go	Go	Go	Go	Go	Pass
Linear Actuator with Ultrasonic Feedback	Go	Go	Go	Go	Go	Go	Pass
Linear Actuator with Hall Sensor	Go	Go	Go	Go	Go	No-Go	Fail
Servo Motor with Lead Screw with Absolute Encoder	Go	Go	Go	Go	Go	Go	🟩
Servo Motor with	Go	Go	Go	Go	Go	Go	🟩

Lead Screw with Ultrasonic Feedback							🟩
Servo Motor with Lead Screw with Hall Sensor	Go	Go	Go	Go	Go	No-Go	Fail

Table 32: Pairwise Comparison for Subtask 7

	Fast response	Reliable	Long-life	Simple Motion	Quiet	Cost	Total	Weight
Fast response	1.00	0.33	0.50	1.00	0.50	1.00	4.33	10.35%
Reliable	3.00	1.00	1.00	2.00	2.00	2.00	11.00	26.30%
Long-Life	2.00	1.00	1.00	2.00	2.00	1.00	9.00	21.51%
Simple Motion	1.00	0.50	0.50	1.00	1.00	0.50	4.50	10.76%
Quiet	2.00	0.50	0.50	1.00	1.00	0.50	5.50	13.15%
Cost	1.00	0.50	1.00	2.00	2.00	1.00	7.50	17.93%
							41.83	100.00%

Table 33: Decision Matrix for Subtask 7

	Weight	Linear Actuator with Absolute Encoder	Linear Actuator with Ultrasonic Sensor	Servo-Lead Screw with Absolute Encoder	Servo-Lead Screw with Ultrasonic Sensor
Fast response	10.35%	1.00	0.50	1.00	0.50
Reliable	26.30%	1.00	0.75	0.75	0.75
Long-life	21.51%	1.00	0.75	1.00	0.75
Simple Motion	10.76%	1.00	1.00	1.00	1.00
Quiet	13.15%	0.50	0.50	0.50	0.50
Cost	17.93%	1.00	0.50	0.50	0.50
Total:	100.00%	0.9343	0.6733	0.8118	0.6733

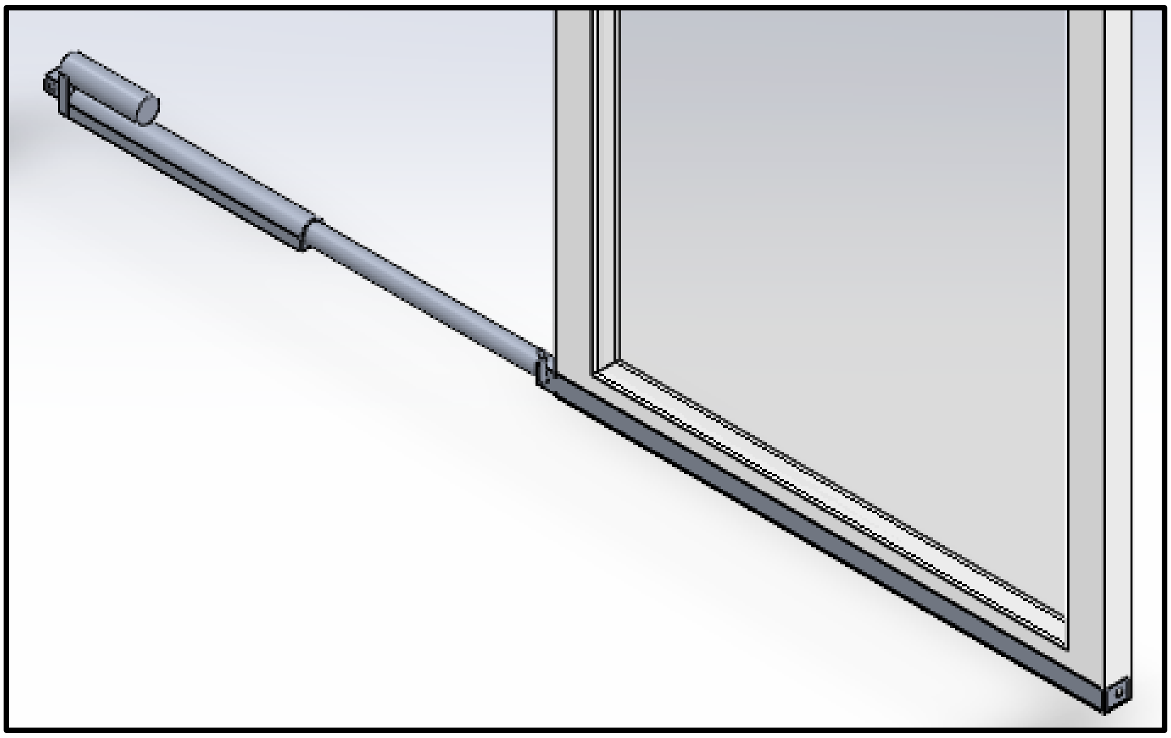


Figure 3: Linear Actuator Method for Translating Sliding Window

The linear actuator using absolute encoders for position control was selected after consulting the decision matrix. The main benefit of using the linear actuator over the servo with lead screw has to do with the reliability and cost of the design. The linear actuator is an enclosed design that does not require an external housing to shield rotating components. This is why the cost and reliability for this component ranks better with respect to the servo-lead screw method. The benefit of using absolute encoders over ultrasonic sensors is because ultrasonic sensors are more complex and can experience unexpected disturbances and interferences. This decreases the response time and reliability of the device. Thus, the linear actuator with absolute encoders should be used for this method.

Subtask 8: Motorizing Shades

Table 34: Concepts for Subtask 8

Concept	Description
Linear Actuator with Absolute Encoder	● Linear actuator to translate blind along fixed track ● Position Control using absolute encoder
Linear Actuator with Ultrasonic Sensor	● Linear actuator to translate blind along fixed track ● Position Control using ultrasonic sensor feedback
Servo Motor with Absolute encoder	● Servo to translate blind via shaft ● Position Control using absolute encoder
Servo Motor with Ultrasonic Sensor	● Servo to translate blind via shaft ● Position Control using ultrasonic sensor feedback

Table 35: Go/No-Go Chart for Subtask 8

Constraints						
Concept	Must close shades securely	Allows for Autonomous movement	Does not cause shading damage	5 cm/s speed limit	Does not require additional feedback	Result
Linear Actuator with Absolute Encoder	Go	Go	Go	Go	Go	Pass
Linear Actuator with Ultrasonic Sensor	Go	Go	Go	Go	No-Go	Fail
Servo Motor with Absolute encoder	Go	Go	Go	Go	Go	Pass
Servo Motor with Ultrasonic Sensor	Go	Go	Go	Go	No-Go	Fail

Table 36: Pairwise Comparison for Subtask 8

	Fast response	**Reliable**	**Long-life**	**Simple Motion**	**Quiet**	**Cost**	**Total**	**Weight**
Fast response	1.00	0.33	0.50	0.8	0.50	0.6	3.73	8.90%
Reliable	3.00	1.00	1.00	2.00	2.00	2.00	11.00	26.25%
Long-life	2.00	1.00	1.00	2.00	2.00	1.00	9.00	21.48%
Simple Motion	1.25	0.50	0.50	1.00	1.00	0.50	4.50	10.74%
Quiet	2.00	0.50	0.50	1.00	1.00	0.50	5.50	13.13%
Cost	1.67	0.50	1.00	2.00	2.00	1.00	8.17	19.50%
							41.9	100.00%

Table 37: Decision Matrix for Subtask 8

	Weight	Linear Actuator with Absolute Encoder	Servo Motor with Absolute Encoder
Fast response	8.90%	1.00	1.00
Reliable	26.25%	0.5	1.00
Long-life	21.48%	1.00	1.00
Simple Motion	10.74%	0.5	1.00
Quiet	13.13%	0.50	0.50
Cost	19.50%	0.50	1.00
Total:	100.00%	0.6520	0.93435

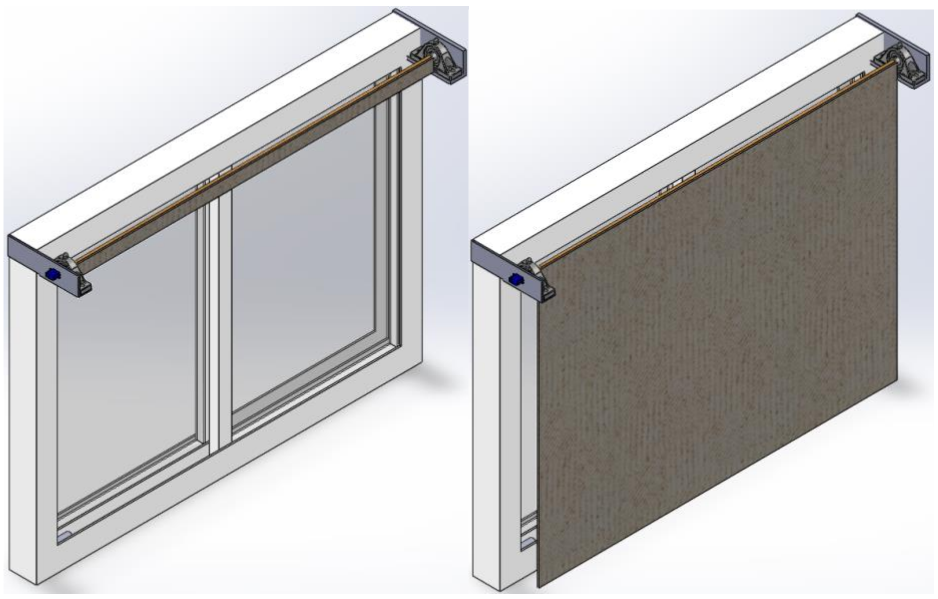


Figure 4: Servo Motor Method with Absolute Encoder for Position Control

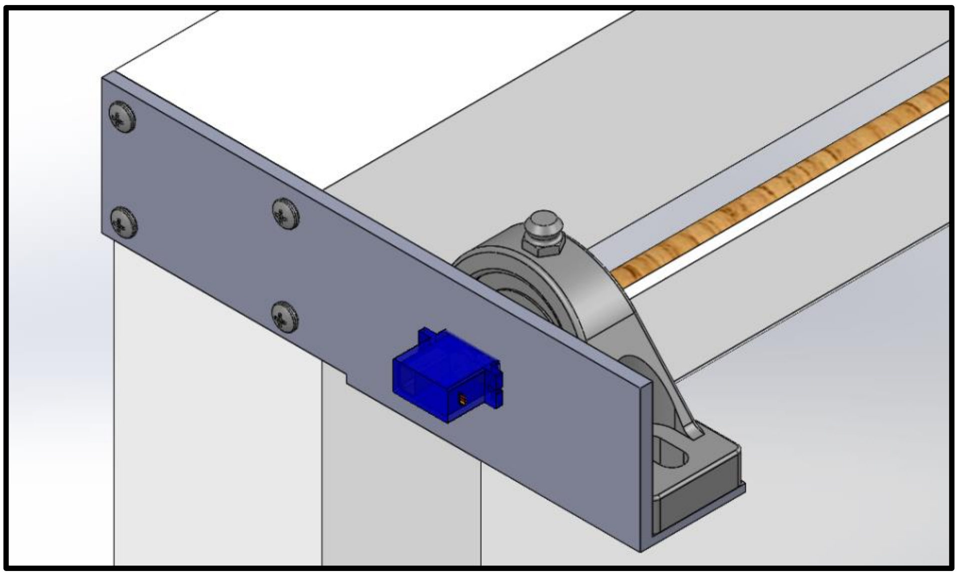


Figure 5: Section View of Servo Motor Method with Absolute Encoder

The methods using ultrasonic sensors for feedback were eliminated because implementing this would require unnecessary feedback logic that could be easily replaced by an absolute encoder. Of all the objectives, reliability and long product life were weighed the heaviest. The decision matrix determined that the Servo Method with an absolute encoder was more ideal for the design. Another reason why the Linear Actuator method was not pursued was because this would require a redesign to incorporate a track for the blinds. This would increase costs in manufacturing and create a more complex design than necessary. A SolidWorks model was made for the Servo motor method shown above.

Subtask 9: Protection of external sensors

Table 38: Concepts for Subtask 9

Concept	Description
Potting glue method	● Wrapped all external sensors with epoxy resin potting
Mechanical housing with nano coating	● Design a housing with slots to store all sensors.Nano coating is applied to the housing surface and voids.
Conformal coating	● Coat all external sensors with conformal coating

Table 39: Go/No-Go Chart for Subtask 9

Constraints				
Concept	Does not affect the collection of input data	Ease to produce	Waterproof, dirt resistant and anti-corrosion	Result

Potting glue method	Go	Go	Go	Pass
Mechanical housing with nano coating	Go	Go	Go	Pass
Conformal coating	Go	Go	NO-Go	Fail

Table 40: Pairwise Comparison for Subtask 9

	Long-life	Reliable	Protection Performance	Accurate	Cost	Total	Weight
Long-life	1.00	0.50	0.50	3	2.00	7	20.9%
Reliable	2.00	1.00	0.5	0.50	2.00	6	17.91%
Protection Performance	2.00	2	1.00	3	2.00	11	32.8%
Accurate	0.33	2.00	0.33	1.00	3.00	6.66	19.9%
Cost	0.50	0.50	0.50	0.33	1.00	2.83	8.45%
						33.49	100.00%

Table 41: Decision Matrix for Subtask 9

	Weight	Potting glue method	Mechanical housing with nano coating	Conformal coating
Long-life	20.9%	1	1.00	0.25
Reliable	17.91%	1	1.00	0.25
Protection Performance	32.8%	1	1.00	0.5
Accurate	19.9%	0.50	0.50	1.00
Cost	8.45%	0.25	1.00	1.00
Total:	100%	0.672725	0.9001	0.80555

It was decided that the mechanical housing with nano-coating would be more suitable for this task because its protection performance is better, and the cost is less. Although using the potting glue method has good performance, it costs more money to purchase the appropriate abrasives. Conformal coating cannot meet the requirements of long life and reliability, so it is not considered.

Subtask 10: Sliding Window Failsafe

Table 42: Concepts for Subtask 10

Concept	Description
Touch Sensor	● Uses touch sensor that is attached to the side of sliding window which stops the window when touch is sensed
Ultrasonic Sensor	● Uses sensors to determine normal distances that the window should be and compare with actual position to ensure no obstruction
Infrared Sensor	● Uses sensors to ensure that path is not obstructed
Mechanical Redesign	● Redesign window so that user cannot access or obstruct the window's line of motion. Perhaps a mesh screen in front of the sliding window to prevent access.

Table 43: Go/No-Go Chart for Subtask 10

Constraints				
Concept	Does not change original function	Does not obstruct user's view of window	Does not require additional power source	Result
Touch Sensor	Go	Go	Go	Pass
Ultrasonic Sensor	Go	Go	Go	Pass
Infrared Sensor	Go	Go	Go	Pass
Mechanical Redesign	Go	No-Go	Go	Fail

Table 44: Pairwise Comparison for Subtask 10

	Fast response	**Reliable**	**Repeatabl e**	**Accura te**	**Long-life**	**Cost**	**Total**
Fast response	1.00	0.50	0.50	0.33	2.00	2.00	6.33
Reliable	2.00	1.00	2.00	0.50	3.00	2.00	10.50
Repeatable	2.00	0.50	1.00	0.33	2.00	2.00	7.83
Accurate	3.00	2.00	3.00	1.00	4.00	3.00	16.00
Long-life	0.50	0.33	0.50	0.25	1.00	1.00	3.58
Cost	0.50	0.50	0.50	0.33	1.00	1.00	3.83

Table 45: Decision Matrix for Subtask 11

	Weight	Infrared Sensor	Ultrasonic Sensor	Touch Sensor
Fast response	13.51%	0.50	0.50	1.00
Reliable	32.43%	0.50	1.00	1.00
Robust	32.43%	0.50	1.00	1.00
Cost	21.62%	0.50	0.50	0.50
Total:	100.00%	0.5000	0.8243	0.8919

It was decided that the touch sensor would be more suitable for this task because it is more robust and reliable. Although, using an ultrasonic sensor can detect the normal ranges of positions that the window should be in and detect abnormal distances caused by human disturbances, the touch sensor allows for immediate changes since it can stop the device as soon as there is contact. This allows for less computation and creates a simpler design. Infrared sensors also cannot be used in the sunlight and have difficulty detecting obstacles in outdoor or dark indoor applications.

Subtask 11: Emergency Evacuation Override

Table 46: Concepts for Subtask 11

Concept	Description
Pull Pin Mechanism	● Use of pull-pin on bracket for easy removal
Button Mechanism	● New Mechanism that will disconnect sliding window with push of a button

Table 47: Go/No-Go Chart for Subtask 11

Constraints				
Concept	Inaccessible from outside	Does not obstruct user's view of window	Ensures safe operation	Result
Pull Pin Mechanism	Go	Go	Go	Pass
Button Mechanism	Go	Go	Go	Pass

Table 48: Pairwise Comparison for Subtask 11

	Aesthetic	Reliable	Fast response	Cost	Total	Weight
Aesthetic	1.00	0.50	0.50	0.50	2.50	13.51%
Reliable	2.00	1.00	1.00	2.00	6.00	32.43%
Fast response	2.00	1.00	1.00	2.00	6.00	32.43%

Cost	2.00	0.50	0.50	1.00	4.00	21.62%
					18.50	100.00%

Table 49: Decision Matrix for Subtask 11

	Weight	Button Mechanism	Pull Pin
Aesthetic	13.51%	1.00	0.50
Safe	32.43%	1.00	1.00
Fast response	32.43%	1.00	1.00
Cost	21.62%	0.50	1.00
Total:	100.00%	0.8919	0.9325

Although the Button concept is more aesthetic, the pull pin mechanism would be significantly cheaper, which is weighted more heavily than the button mechanism. Since both have the same function, the pull pin design will be used.

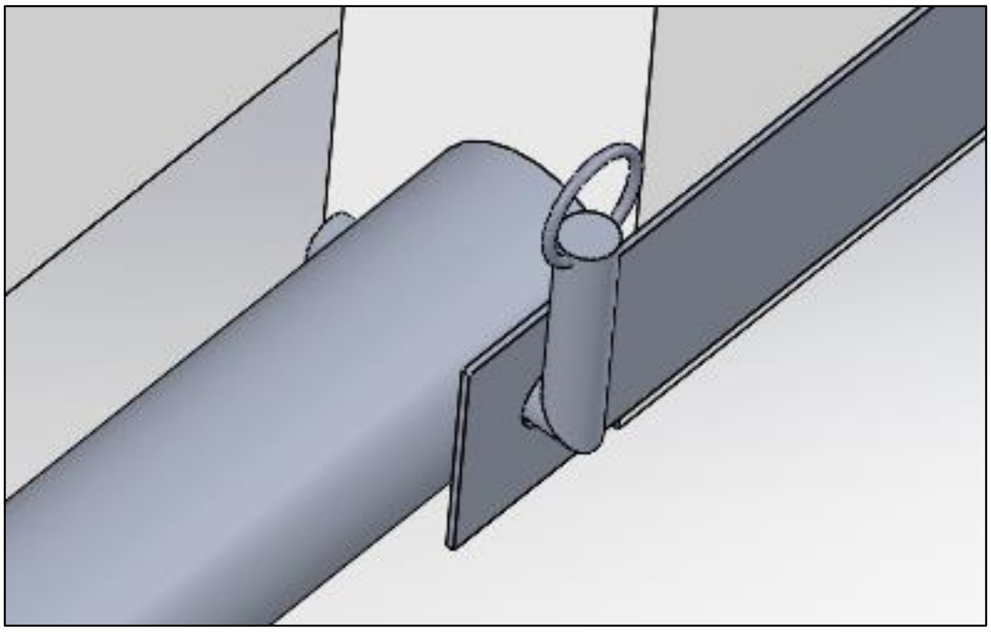


Figure 6: Section View of Pull Pin Mechanism.

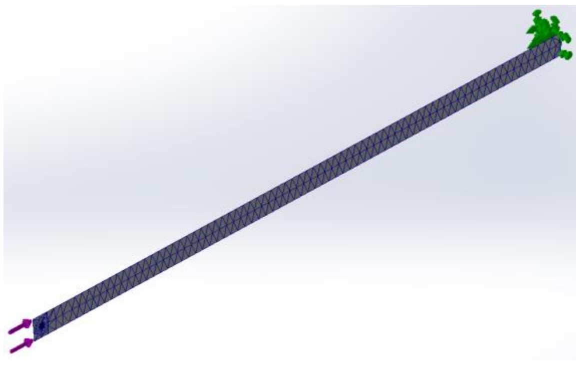
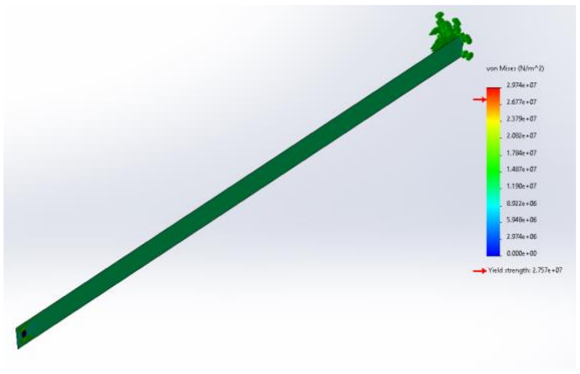
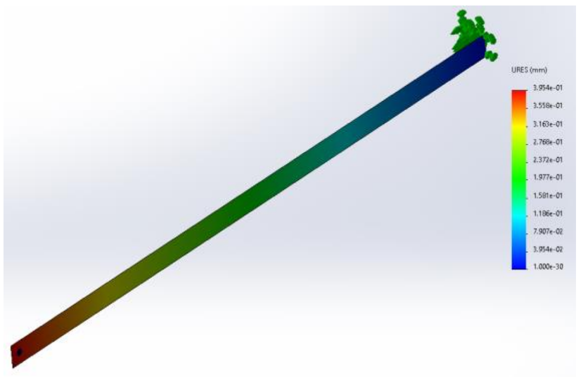
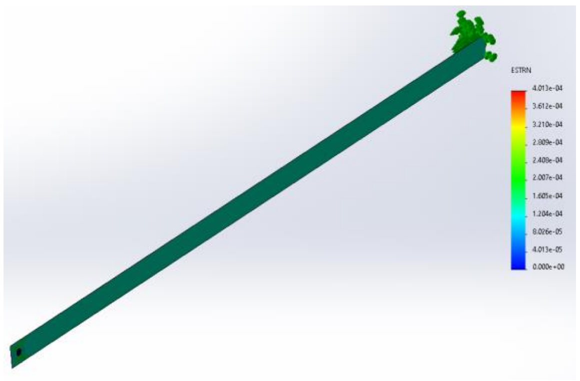
Design Validation

Mechanical Validation

Linear Actuator and Sliding Window Method

The translational speed of the linear actuator is 5.7mm/s. This means that it would take around 79 seconds to cover the 450mm stroke. This is reasonable as weather conditions would not change drastically within this time period.

The linear actuator also has a maximum force of 750N, which is more than what is sufficient to close the window sufficiently. However, it was crucial to ensure that this force can be safely applied to the linear actuator bracket and sliding window. Although the total force of 750N would only be applied to the bracket and window in its extreme condition, it was decided that a static study should be conducted to ensure that the design was safe. The results are shown below.

	 von Mises (N/m^2) 2.974e+07 2.677e+07 2.379e+07 2.082e+07 1.784e+07 1.487e+07 1.190e+07 8.922e+06 5.948e+06 2.974e+06 0.000e+00 Yield strength: 2.757e+07
Figure 7: Mesh of Linear Actuator Bracket	Figure 8: Static Stress Study of Linear Actuator Bracket
 URES (mm) 3.954e-01 3.558e-01 3.163e-01 2.768e-01 2.372e-01 1.977e-01 1.581e-01 1.186e-01 7.907e-02 3.954e-02 1.000e-30	 ESTRN 4.013e-04 3.612e-04 3.210e-04 2.809e-04 2.408e-04 2.007e-04 1.605e-04 1.204e-04 8.026e-05 4.013e-05 0.000e+00
Figure 9: Static Displacement Study of Linear Actuator Bracket	Figure 10: Static Displacement Study of Linear Actuator Bracket

From the static studies for the linear actuator bracket, the maximum von Mises stress is under the yield stress of the material, the maximum displacement is less than 0.4 mm, and the maximum equivalent strain is less than 4.013e-4. This shows that the bracket is well equipped to handle the 750N load that is applied by the linear actuator.

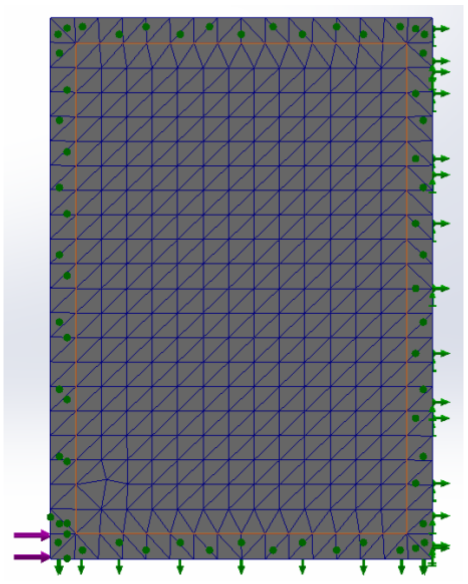


Figure 11: Mesh of Sliding Window

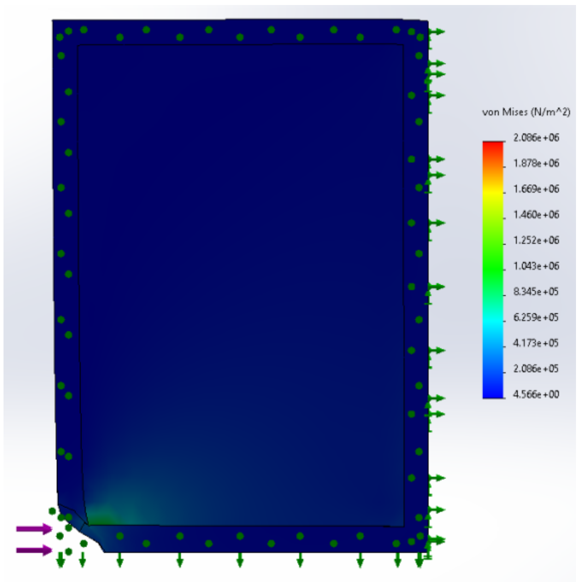


Figure 12: Static Stress Study of Sliding Window

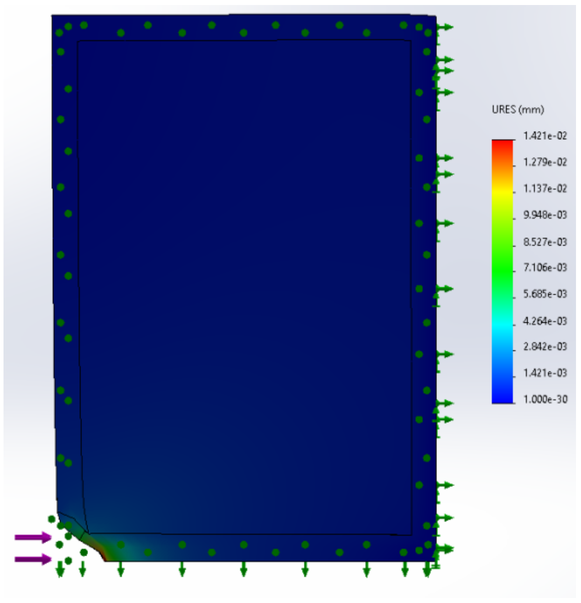


Figure 13: Static Strain Study of Sliding Window

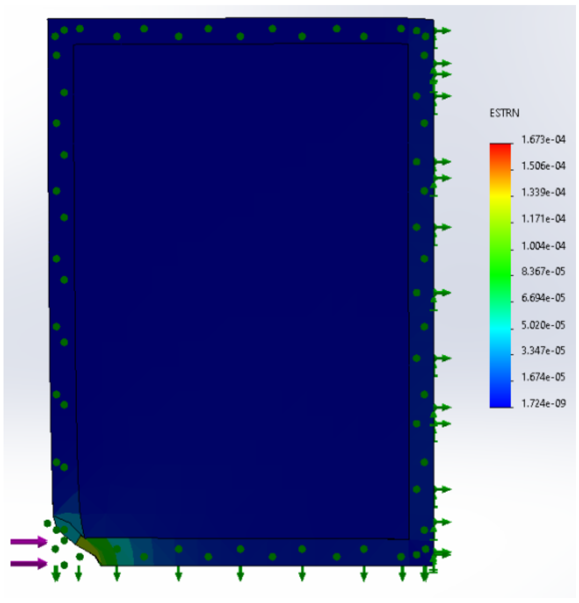


Figure 14: Static Displacement Study of Sliding Window

The static studies for the window show that it cannot handle the 750N force. Thus, we must ensure that the force from the linear actuator does not get forced onto the window. The design must never have a condition where the linear actuator is extended past the window's closed position. To mitigate this, the linear actuator will be placed in a position where the fully extended position lies at the closed state. This means that the sliding window will never be jammed onto the frame of the device.

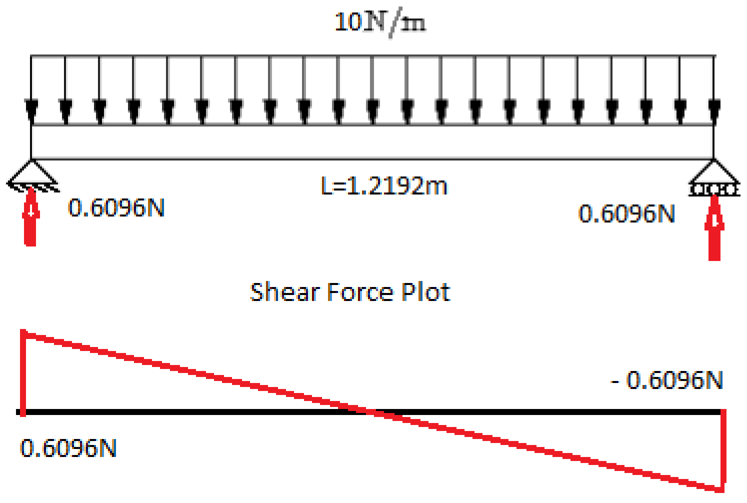
Servo Motor and Blind Translation Method

Test for Shear Failure

It was decided that the main failure of the shaft was due to shear forces. A Shear Force calculation was done to ensure that the material of the dowel was suitable for the design.

Shear Strength: 3-15 MPA [12] (Use 3 MPa to be conservative)
Diameter: 0.03175 m
Length: 1.2192 m.
Area: 0.00025201562 m^2
Assumptions:

- Uniform distribution of weight (10 N/m)



Max Shear Force = 0.6096 N
Shear Stress = V/A = 0.6096 N//0.00025201562 m^2 = 2.418 kPa
Design Factor: 1240.69

Servo Motor Validation

Calculations to validate the torque and speed requirements of the servo motor were conducted to validate the servo use in this mechanism as shown below.

Use of Feetech FS90R Continuous Servo

- Torque: 1.47Nm [12]
- Shaft Diameter: 0.03175 m
- Weight of blinds and shaft: 1.2192 N
- Assumption:
  - Weight is concentrated at edge of shaft (equal to the radius of shaft)
  - Ball Bearing handle all the axial loads (servo only has to handle rotation of shaft)
- Torque Required: (1.2192 N)*(9.8 N/m^2)*(0.03175 m/2) = 0.1898 Nm
- Design Factor: 7.74

- No Load Speed: 130 RPM. [12]
- Desired translational speed: 0.05 m/s or slower

- Circumference of shaft: 3.14159*(0.03175 m) = 0.09974 m
- Required speed: (60 s * 0.05 m)/(0.09974 m) = 30.07 RPM
- Speed can be reduced from 130 RPM to around 30 RPM using the Arduino

Encoder Validation

A calculation was conducted to determine how many revolutions are required to translate the complete blind system

Distance covered required: 0.9144 m
Shaft Diameter: 0.03175 m
Circumference of shaft: 3.14159*(0.03175 m) = 0.09974 m
Required revolutions: 0.9144 m/0.09974 m = 9.167 Revolutions
Verdict: Use an absolute multi-turn encoder that can handle 10 rotations for position control.

Mechanical Housing

The housing consists of the base and the cover .The base and the cover are connected and sealed by a gasket. The following are the 3D printing cost calculation, Transmittance Validation, and Waterproof Validation. The housing is used to store various types of electrical components including rain sensor, temperature and humidity sensor and light sensor. These sensors are used to detect external environment parameters such as rainfall, temperature, and light intensity(Figure 15).

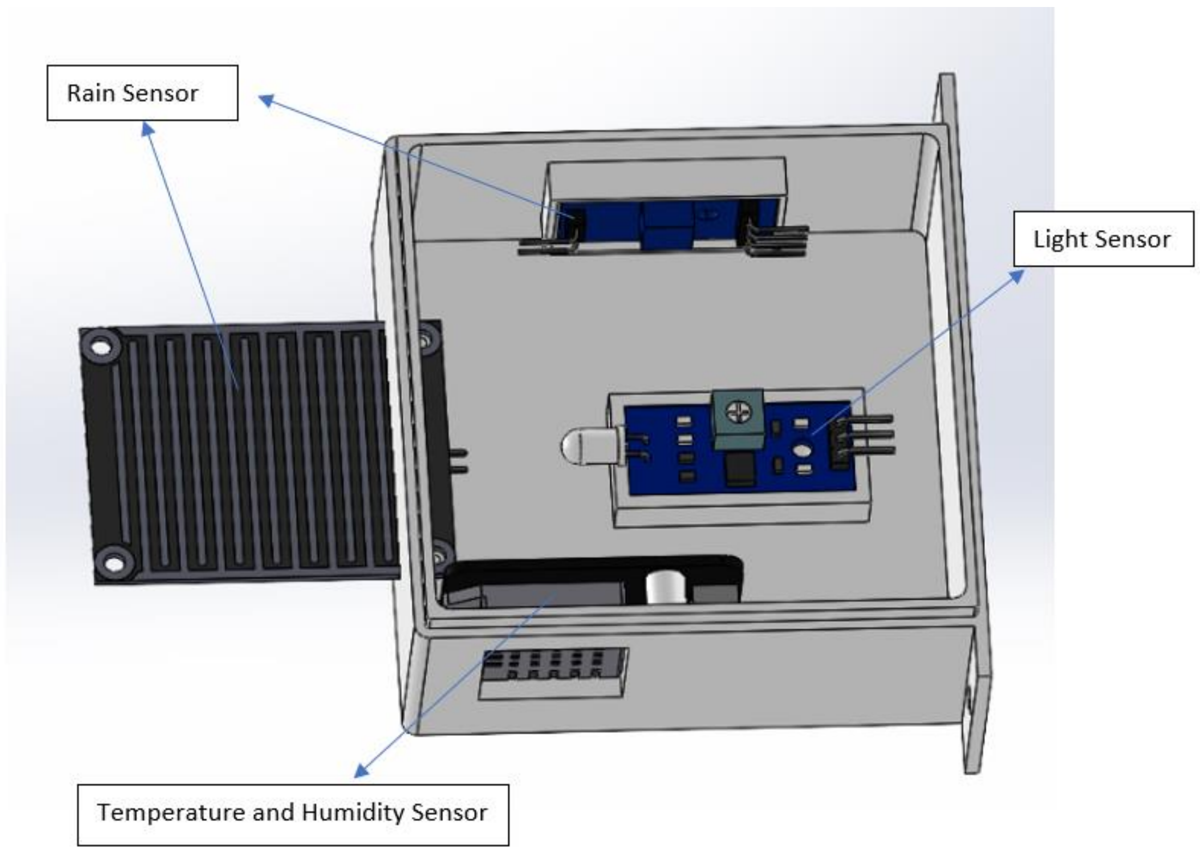


Figure 15: CAD of mechanical housing

Considering the accuracy of the measurement and the need for aesthetics, a light sensor, which was originally placed indoors, was decided to be placed inside the housing. This change also indicates the replacement of the original material from opaque ABS to semi- translucent ABSI. ABSI is similar to ABS and has two additional properties: semi-transparent and high impact resistance.The main cost of this mechanical housing is the consumption of ABS plastic.

ABSI is relatively inexpensive (prices: currently around $45 per kg[13]).   p=45 CAD/kg
Through Solidworks mass property, the Mass of the housing base and cover is determined to be 83.73grams.
The cost of housing is calculated as p*M=83.73grams/1000*$45(CAD/ kg)=$3.768 per housing.

The calculations above show that the material cost is in a reasonable range. Moreover, a smart home window requires only one mechanical housing. Therefore, 3D printing is suitable for producing both prototypes and final product.

Transmittance Validation

To ensure that the light sensor measurement is not overly influenced, the transmittance of housing is calculated as follows.

The parameters and assumptions used in the calculation are defined below.
Intensity of sunlight at the surface of the earth is: I_0=1 kW/m2 [14]
The spectrum of the Sun is from 296 to 1300nm, So 296 nm <λ <1300nm [15]
Attenuation coefficient of ABSI: when λ=1300nm u=0.0018m^{-1}=0.18cm^{-1}
when λ=296nm u=0.0083m^{-1}=0.83cm^{-1} [16]

Thickness of housing cover: x=3mm=0.3cm

According to Beer- Lambert law:

$$I_1= I_0*e^{-ux}$$ [17]

Light intensity after passing the housing I_1=1*e^(-0.18*0.3)=0.5827 kW/m2 (λ=1300 nm)
I_1=1*e^(-0.83*0.3)=0.7796 kW/m2 (λ=296 nm)

Transmittance of the housing is T=I_1/I_0=0.5827/1=58.27% (λ=1300nm)
T=I_1/I_0=0.5827/1=77.96% (λ=296 nm)

Therefore, the transmittance of the housing is between 58.27% and 77.96%. This transmittance is acceptable. Although the light intensity becomes approximately ⅔ of the original, the light sensor can be adjusted to redefine the dark and light levels by adjusting the internal code.

Waterproof Validation

Due to long-exposure to the external environment and non-watertight internal wiring and sensor, the permeability of the housing is very important. To examine the airtightness of the house, it is placed in water. The following is a pressure test prior to the airtightness test.

When water completely submerges the housing
Height of the housing h=50mm
Density of water ρ = 1000 kg.m-3
Pressure= ρ g h =1000*9.8*0.05=490 pa =0.0710684915 psi

In the SolidWorks model, the mounting holes on both sides are fixed. All materials are subjected to the influence of gravity (g=9.81m*s^-2). The housing is distributed with a pressure greater than 0.071 psi in all directions to simulate the extreme situation where the housing is completely submerged by water. The result of the static analysis (Figure 16) has a factor of safety of 120, which indicates a sufficiently overdesigned level of safety. It also shows that the design of the housing resists damage by water pressure perfectly. It also passes the pressure test before the airtightness test. Afterward, the housing was pressed into the water in an airtightness test. This housing shows good air tightness.

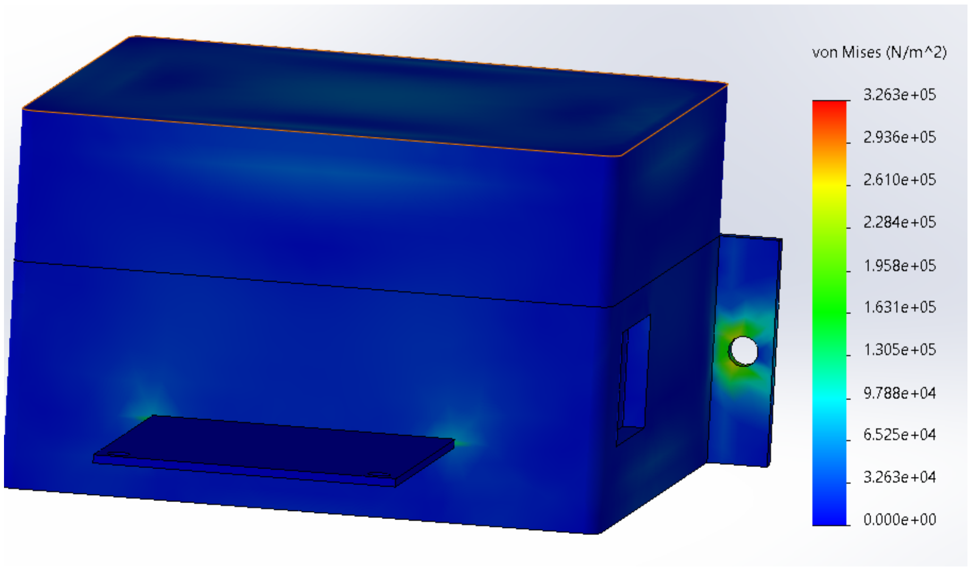


Figure 16: Results of static analysis of housing

Electrical Validation

Power Usage Analysis

$$Total\ Power\ =\ Max\ Voltage\ *\ Max\ Current$$
$$Max\ Voltage\ = \text{5V for Sensors and 12V for Motors}$$

Table 50: Electrical Component Source Maximum Listed Current (mA)

Electrical Component	Source	Operating Current (mA)
Raspberry Pi	Website	600
Arduino Mega 2560	Website	100
Temperature Sensor	Datasheet	1.5
Raindrop Sensor	Website	15
Light Sensor	Datasheet	20
Touch Sensor	Website	0.3
H-Bridge Motor Driver	Datasheet	0.1*

Linear Actuator	Amazon Website	1500**
Blinds Motor	Datasheet	550

*Power dissipation caused by Motor Driver is 100µA for a motor
**Conservate operating current considering window weighs 10lbs

$$\mathit{Total\ Max\ Power} = \sum \mathit{Max\ current\ draw\ from\ each\ component}$$
$$\mathit{Total\ Max\ Power} = 5 * \sum\ (\mathit{Raspberry\ Pi} + \mathit{Arduino\ Mega}\ +\ 2 * \mathit{Temperature\ Sensor}\ + \mathit{Raindrop\ Sensor}\ + \mathit{Light\ Sensor} + \mathit{Touch\ Sensor} + \mathit{Blinds\ Motor})\ + 12 * \sum (\mathit{Linear\ Actuator} + \mathit{Motor\ Driver})$$
$$\mathit{Total\ Max\ Power} = 5 * 1.288\ A + 12 * 1.5A\ = 24.44\ W$$

Conservative operating currents were used to calculate total power required to run the blinds and windows system as shown in Table 50. If all sensors and actuators are being used at the same time, a 25+ W power supply is required.

Power Cost Analysis

During the Power Analysis, it was determined the system's max power is 24.44 W if all sensors and actuators are being used at the same time. In practice, the Raspberry Pi and sensors will be continuously running while the actuators only need to run when opening/closing the windows and blinds which should not exceed several times a day on average. For the cost analysis, a very conservative assumption will be made of actuators running 5% of the time while the Raspberry Pi and sensors are running all of the time. This results in a power consumption of 4.515 W. Over the course of a year, the system's energy usage will be 39.55 kWh.

The average Canadian household uses 11,135 kWh of electricity per year [18]. Air conditioning, heating and lighting contribute up to 55% of the energy usage in a typical home [19]. Considering the average residential cost of electricity in Canada is 0.179 $/kWh [20], this means the average Canadian household spends $1,096.24 on regulating the house's temperature and lighting.

The *Smart Home Window* isn't able to completely replace the central heating and cooling system and lightbulbs of the house. However, it would only need to replace 0.65% of the energy usage of air conditioning, heating and lighting to breakeven.

Calculations:

Average Device Power Consumption:

$$P_{avg} = 100\% \times (\ P_{Rasp\ Pi} + \sum P_{Sensors}) + 5\% \times \sum P_{Actuators}$$
$$P_{avg} = 100\% \times (\ 5\ V \times 0.668\ A) + 5\% \times ((5\ V \times 1.1\ A) + (12\ V \times 1.5\ A))$$
$$P_{avg} = 4.515\ W$$

Device Energy Usage in a year: = (4.515 W × 8,760 hrs/yr)/1000 = 39.55 kWh

Cost of device in a year: 235.12 kWh × 0.179 $/kWh = $7.08

Average Household Energy (Air conditioning, heating and lighting) Cost: $(55\% \times 11{,}135\ kWh) \times 0.179\ \$/kWh = \$1{,}096.24$

Software Validation

Microcontroller Analysis

Wi-Fi Connectivity

In the concept generation phase, the two options for microcontrollers that were considered for the prototype were the Arduino and Raspberry Pi. The Raspberry Pi was chosen over the Arduino due to it being easily connected to the internet using Wi-Fi while Arduino would need an additional module or shield to connect to the internet. This was an important constraint as our design consisted of a Mobile App which allowed the user to control and interact with the device over Wi-Fi and for the device to connect to an Online Weather API. The Raspberry Pi is able to retrieve data from the OpenWeatherMap API and communicate with the mobile app using a Firebase server.

Runtime

Since the device needs to be prepared for user inputs and weather changes at any time, it was important for the program on the Raspberry Pi to be running continuously. Fortunately, the OS of a Raspberry Pi is extremely lightweight, and the power usage is very low. The prototype is required to be plugged in a power source rather than running on batteries for this reason. Since the hardware was capable of running the program continuously, all exceptions in the code should be handled correctly to allow for continuous runtime. Even though the latest iteration of the main program has not run into any runtime errors, it is hard to determine over a long time period as the expected device lifetime is several years. Although it is safe to assume the program can run this long, several unexpected cases were considered and mitigated. In case of a power outage, the system configuration of the device was set to reboot and run the main program once it receives power. In case of a Wi-Fi outage the device will switch to auto mode and use the latest inputs as the device will not be able to communicate with the Firebase setting or the user through the app. In both cases, the user will be notified the device is offline for troubleshooting purposes.

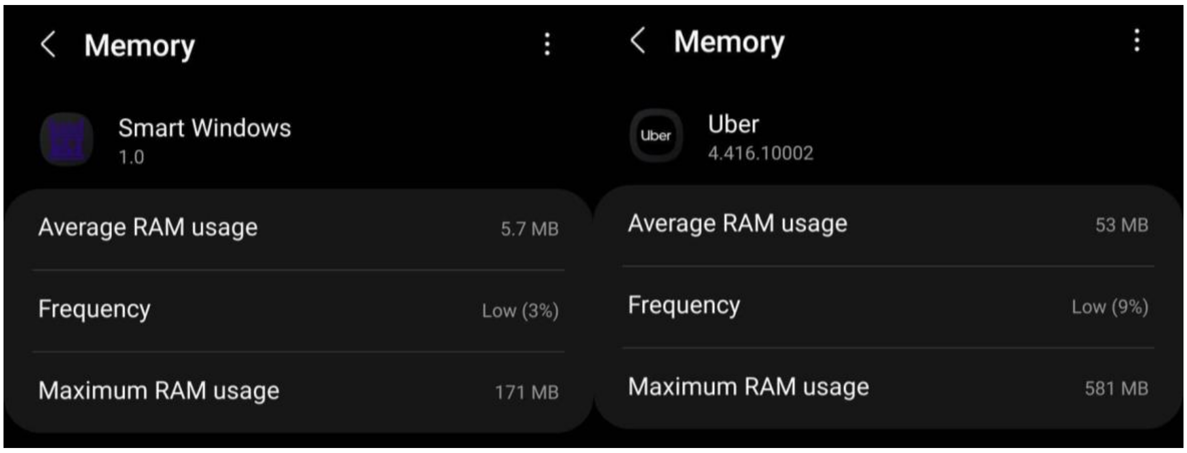
Android Application

Some objectives the Android application has to meet is a low memory usage, low background memory usage and low battery usage. The android application should work in unison with other instances of the application where mode settings (common to the device) are updated for every user. Moreover, the application should also be compatible with 95% of the current Android Market share which means it has to be compatible with Android OS Version 7 and up [21].

Computational Efficiency

Most Android phones available today have a minimum physical RAM of 4GB [22]. The goal of the SHW app is to use less than 400MB of RAM as that is the higher end of what non-media intensive applications use [23]. The SHW App was tested against the Uber app as they both require background location access and have customizable user settings changed through a user's account.

The conducted test includes both the SHW app and the Uber application being run for 1.5hrs in foreground with moderate usage including changing settings and updating inputs and also run for 1.5hrs in the background including a 45 minute drive across London to see how much maximum RAM and average RAM was used.


Memory

Smart Windows
1.0

Average RAM usage	5.7 MB
Frequency	Low (3%)
Maximum RAM usage	171 MB

Memory

Uber
4.416.10002

Average RAM usage	53 MB
Frequency	Low (9%)
Maximum RAM usage	581 MB

Figure 17: Application Memory Comparison Test

As shown in Figure 17, the SHW app had a maximum RAM usage of 171 MB and maintained an average RAM usage of 5.7 MB. By using minimal widgets and sending only bits of information to Firebase, RAM usage was minimised. Comparatively, the Uber app used an average RAM of 53 MB and a maximum RAM of 581 MB.

Power Efficiency

To implement a security feature where the blinds and windows close while all occupants are outside the home, all SHW app users' location must be retrieved (while stationary and moving). Three APIs were tested to see which was most power efficient while being able to update each user's location in a minute's interval (see Design Refinement for the updated test). To maintain the house's security, the blinds and windows must close within one minute of leaving the house. When implementing the LocationManager and FusedLocationProvider APIs, the user's location will have to be manually pinged every minute in order to obtain reliable location information. Geofencing however, intelligently pings for updates using the most power efficient location provider available (ex. Wi-FI, GPS or Network) while the user is moving. While the mobile phone is stationary (determined by the phone's accelerometer), location updates are not being retrieved. Through testing, geofences were observed to be updating at a maximum of 30 seconds after leaving or entering the geofence zone.

Control Validation

The main components of this automated window design depend on the control systems to accurately function. Without a feedback system, the blind system and the linear actuator system would solely rely on timing parameters in order to translate the blind and sliding window to desired positions. This would not be ideal since it would require timing to be perfect and assumes that everything is calibrated with tight tolerances. To mitigate this, a feedback system is implemented to validate the position of each respective subsystem.

In the earlier stages of the prototyping, we decided to use ultrasonic sensors for position feedback. Some issues involving the ultrasonic sensor include poor value readings and misalignment of the blind. Sometimes the ultrasonic sensor would read random high or low values that were outside the normal range of the readings. To mitigate this, a bandpass filter was introduced in the software to ignore readings below a certain threshold. A median filter was also used to only account for the median value of 5 readings to isolate abnormally high or low values. To mitigate the misalignment of the blinds,a dowel was added to the bottom of the blind to weigh down the blinds, ensuring some tension was present in the fabric. The dowel also widened the cross sectional area that would be read by the ultrasonic sensor, which improved the performance.

The same approach to calculate distance was used for the linear actuator method. An ultrasonic sensor was placed on the side of the frame facing the window that would be actuated. The program would adjust the window to the user's desired position based on the readings of the sensor. This was the main method of feedback for distance.

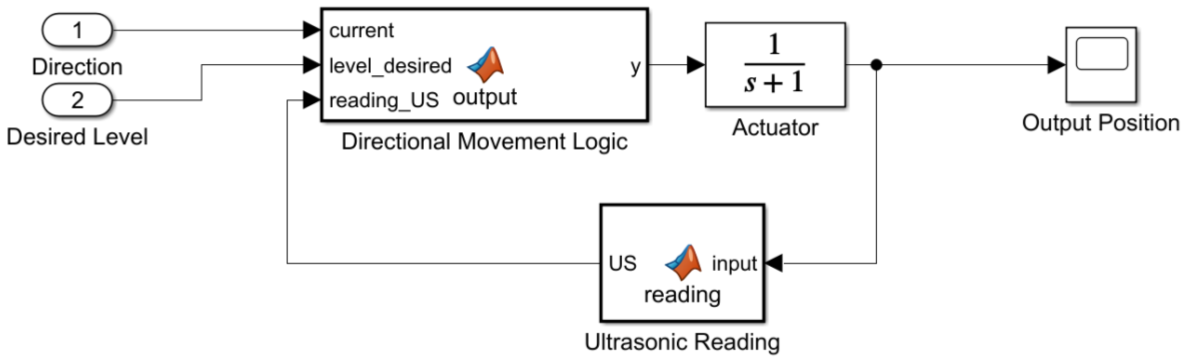


Figure 18: Control Schematic of System

Some limitations of this control method is that it assumes that there is no interference of the ultrasonic sensors. If unintended objects are found between the ultrasonic sensor and the window or blinds, the program will think that the position is in a different place. This would cause the program to think that the blinds are closer to the closed position and that the sliding window is closer to the open position. To mitigate this risk, the path the ultrasonic sensor senses needs to be closed off. However, doing so may affect the aesthetics of the design.

Thermal Validation

From thermal calculations (Table 51), we found that it would take around 40 kJ to increase the room temperature by 1°C. Given that there are no losses, it would take around 35.87 seconds to heat up the room by 1°C. This is not a realistic scenario, but it shows that solar radiation provides heat at around 1.11kW, which can save the users costs for heating. The same can be argued for actuating the blinds so that this heat does not enter the room in cases where the goal is to lower the internal temperature.

Table 51: Heat Calculations

Assumptions:

(1) No losses
(2) Room filled with just air

Given:

(1) Average bedroom size: 12’x12’x8’: 1152 cu ft= 32.62101m^3
(2) m_air=1.222(kg/m^3)* 32.621(m^3)=39.798kg
(3) c_air=1005J/C
(4) Q_room=mc(tf-ti)
(5) Φ_sun=1000w/m^2
(6) A_window=36”x48” or 1.11m^2

Calculations:

(1) Q_room=40(tf-ti)
(2) P_sun=Φ_sun*A_window = 1.11kW
(3) Q_sun=P_sun*time
(4) Q_sun+Q_room=0
(5) 1.115W(time)=40(tf-ti)
(6) Let (tf-ti)=1
(7) time=40/1.115
(8) time=35.87s.

Design Refinement

Control Refinements

After further iterations, it was decided that the use of an absolute encoder for the position control of both the blinds and sliding window would be more effective than using feedback from an ultrasonic sensor. When using the ultrasonic sensor, the accuracy and precision of the positions were directly affected by the ultrasonic readings. This causes a reliability issue since any disturbance to the ultrasonic sensor readings would affect the position of the device. Although bandpass and median filters could be used to ignore bad values, the use of these filters require more readings, which significantly lowers the response time of the system. With a longer response time, the device also tends to overshoot the desired value. Although this could be mitigated by increasing the tolerance range that is accepted for each position level, this directly lowers the precision of the device.

The use of an absolute encoder speeds up the response time and avoids complex computations that are associated with ultrasonic sensor feedback. Although overshoot may still occur, the faster response time of this method will significantly lower the overshoot, since the program can poll for the encoder position at a higher frequency.

Android Application Refinements

Power consumption of each location-provider API was obtained by determining the drop in battery percentage over the duration of the test. This test included power consumed by the screen, the operating system and other programs. This test was re-done to obtain just the power consumed from the application to see if an Android phone with the SHW App running can get through the day on one charge.

LocationManager and FusedLocationProvider APIs for power usage using an independent application (see Appendix 1 for location provider code). This application was run for one hour for both APIs where the button was clicked (through software) to determine how much battery was lost over the timespan. Geofencing was also tested for one hour (stationary, driving) using another app that sets up and runs geofences at a selected location (see Appendix 2 for geofencing code).

Table 52: Power Consumed by Location Providers

Location Service Provider API	LocationManager	FusedLocationProvider	Geofencing (Stationary)	Geofencing (Moving)
Battery Consumed (excluding screen time)	8%	6%	0.2%	1%
Updates Within a Minute	Yes	Yes	Yes	Yes
Location is Accurate	Yes	Yes	Yes	Yes

It was observed that geofences used 5% less battery than FusedLocationProvider and 7% less than LocationManager while providing accurate and timely location updates. Over a 16-hour work day, an application running a geofence is expected to use an additional 4.8% battery life if the user spends two hours a day on the move. This is reasonable power consumption and would not impact the user as long as they finish the day with more than 5% battery left on their phone.

Safety Stop Feature

For the safety stop feature, capacitive touch sensor strips were mounted to the side of the translating sliding window. When this sensor senses a disturbance, an interrupt would be sent to the Raspberry Pi, and translation of the sliding window will be halted until no touch is read by the sensor.

Originally, the safety stop fail safe was to be implemented via ultrasonic feedback by placing the ultrasonic sensor at the bottom and facing upwards. The device would then send an interrupt when the sensor senses any value that is significantly closer to the reference, which would be the top of the frame. It was decided that the touch sensor is more optimal for this function since the response time is much quicker, which is crucial for emergency stop mechanisms.

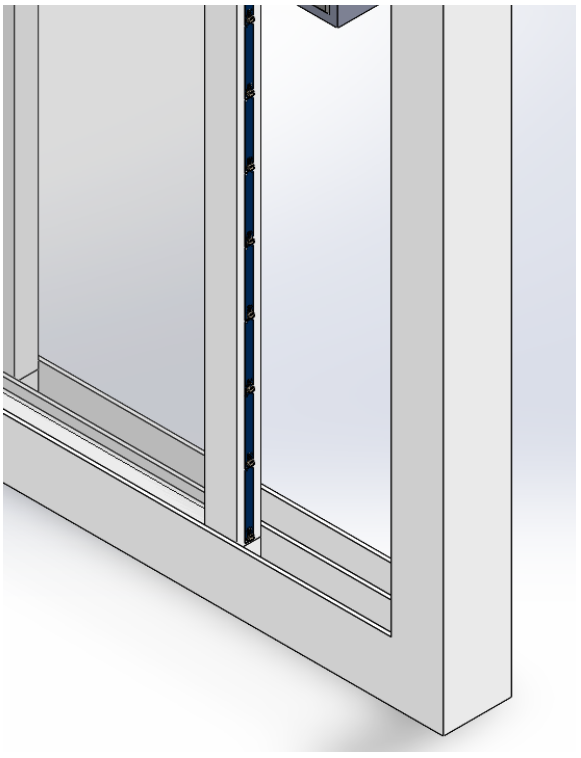
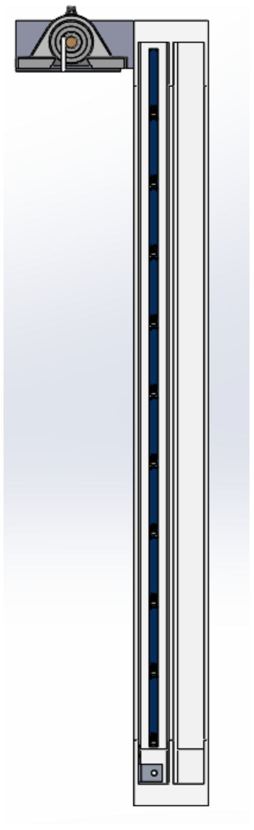
	

Figure 19: Isometric View of Touch Sensor	Figure 20: Section View of touch sensor

Prototype Planning and Development

Mechanical Development

The main mechanical subsystems involved in the prototype include the methods of translation for the sliding window, method of translation of the blinds, housing for the electronics. Some things that are in the final design that are not in the prototype include the pull-pin method to ensure rapid evacuation in emergency scenarios. The primary reason for this is because the housing used for the prototype has slightly different dimensions than in the final product. This caused the group to use different brackets that did not have the same dimensions that would allow for the use of the intended pull pin. An exploded view for the intended assembly is shown below.

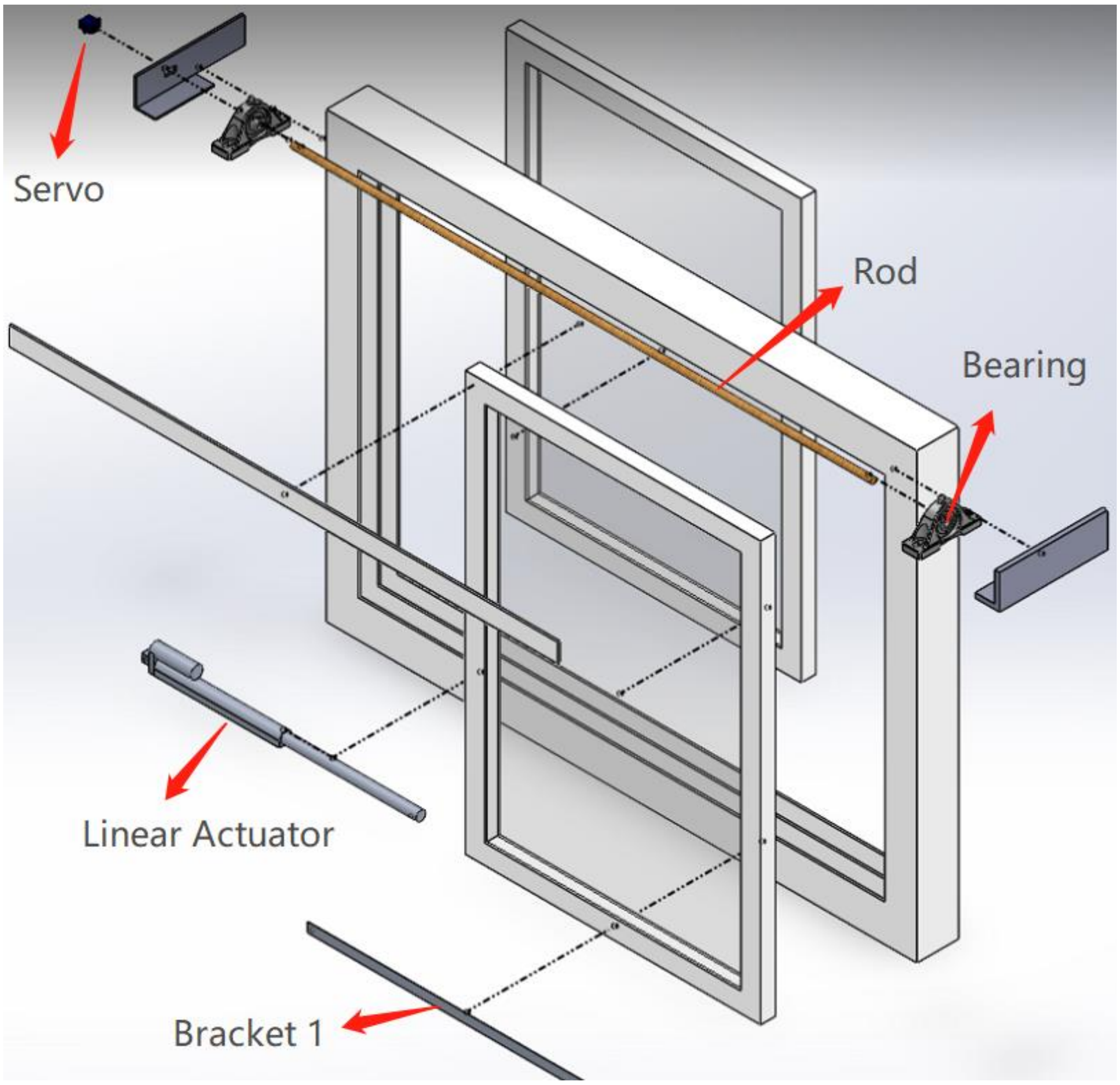


Figure 21: Exploded View of Window

Sliding Window Translation Method

The sliding window translation method for the prototype uses a linear actuator that does not have a built-in absolute encoder. For demonstration purposes, the device will control the position via feedback from the ultrasonic sensors. The assembly steps are noted below.

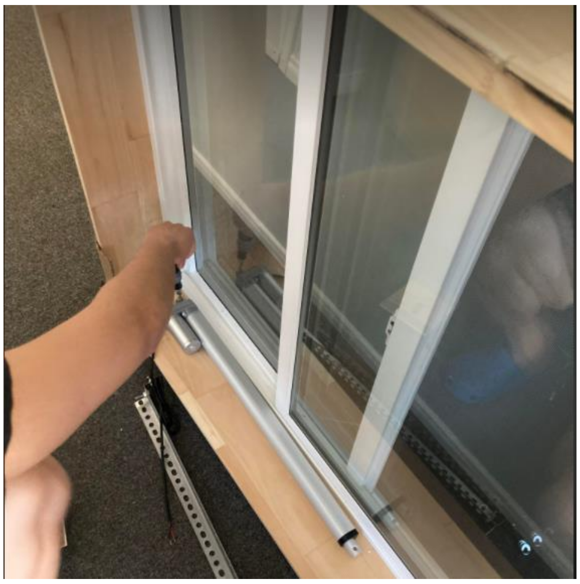


Figure 22: Fully extend the linear actuator and determine desired position with respect to window position. Mount Linear Actuator to desired position.

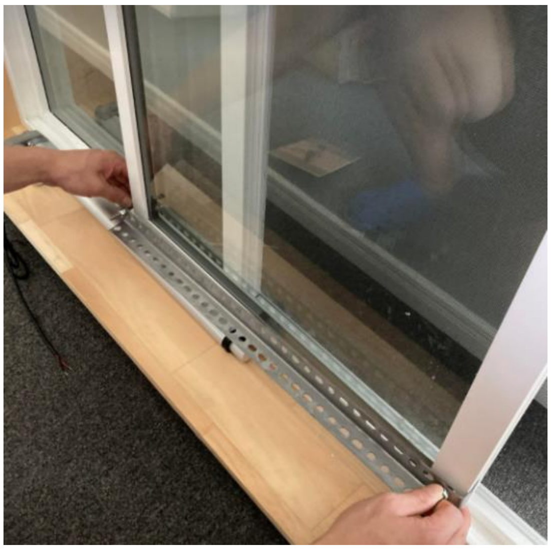
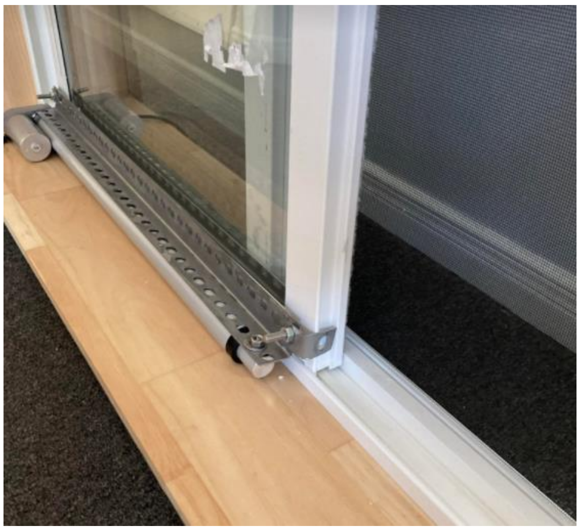
	

Figure 23: Mount the linear actuator retrofit bracket on the lower part of the removable glass side	Figure 24: Connect linear actuator to the linear actuator retrofit bracket by screws

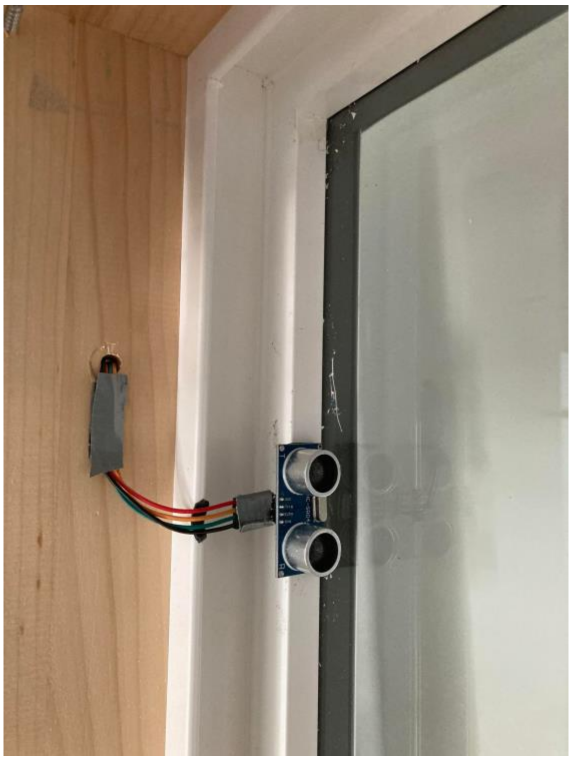
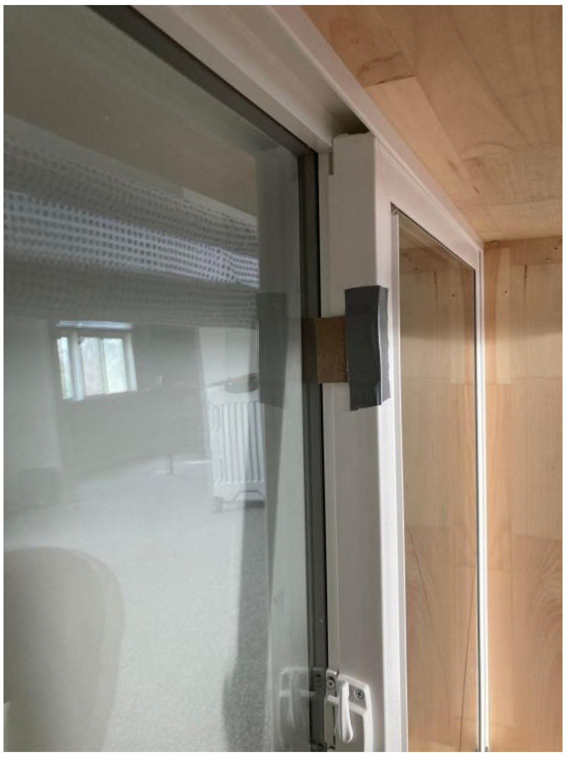
	

Figure 25: Mount ultrasonic sensor on window frame facing the sliding window to be translated	Figure 26: Ensure there is a dark flat surface that is aligned to ultrasonic line of sight to ensure proper reading

Blind Translation Method

The blind translation method for the prototype uses the GB37Y3530-13 12V DC motor with ultrasonic sensor feedback for position control instead of using the Feetech FS90R Servo with an absolute encoder. This change is primarily due to defects in the absolute encoder that was purchased and the inability to properly mount the ball bearing to the wooden shaft. The assembly instructions are indicated below.

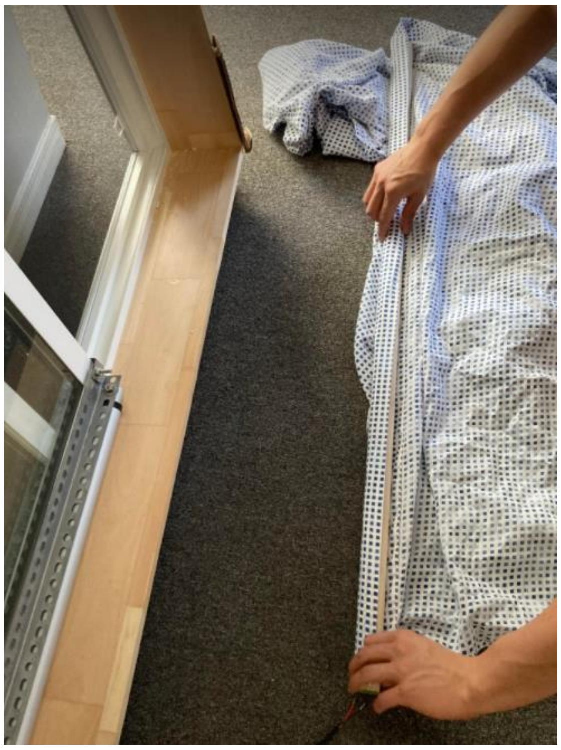
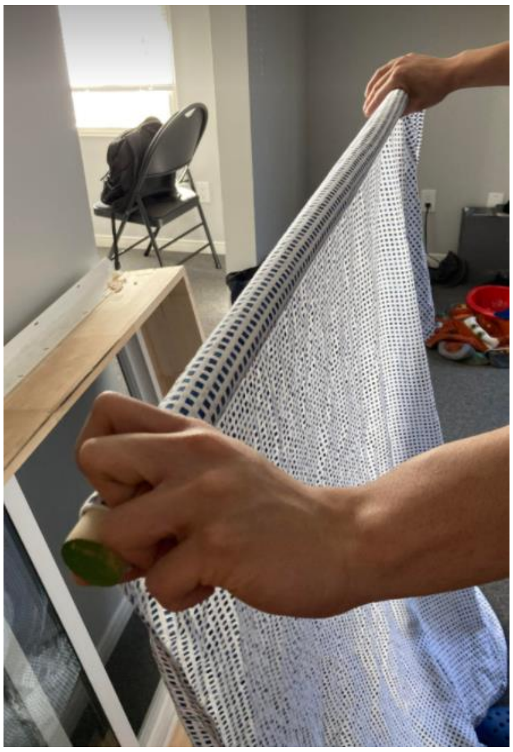
	

Figure 27: Attach the wooden dowel to the end of the curtain.	Figure 28: Attach the wooden shaft to the top of the curtain.

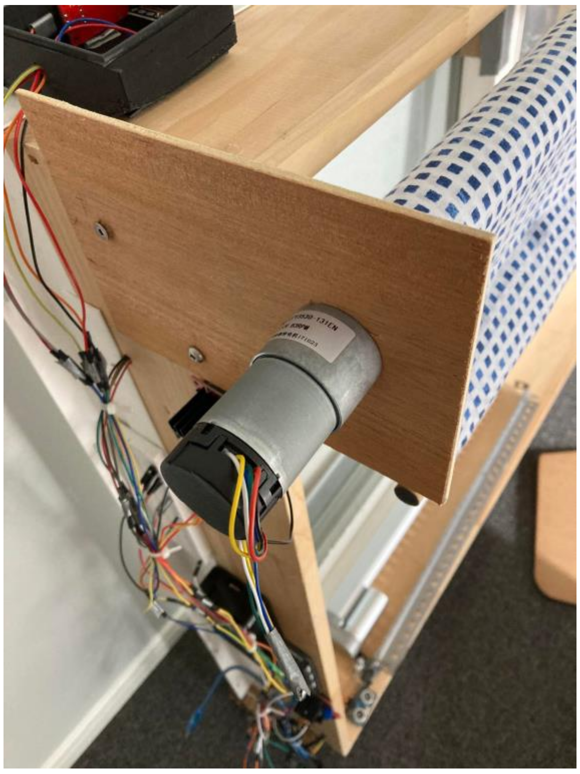
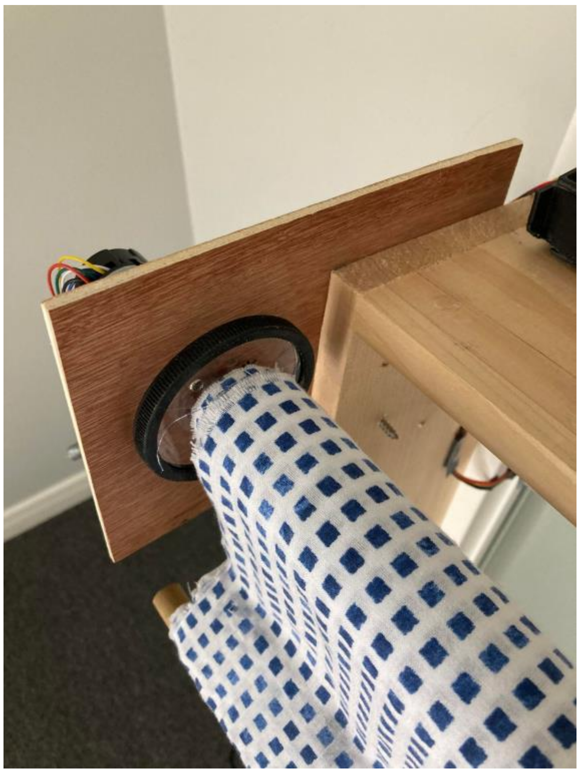
	

Figure 29: Mount DC Motor on the Housing of the window.	Figure 30: Secure DC motor with the coupling and the wooden shaft.

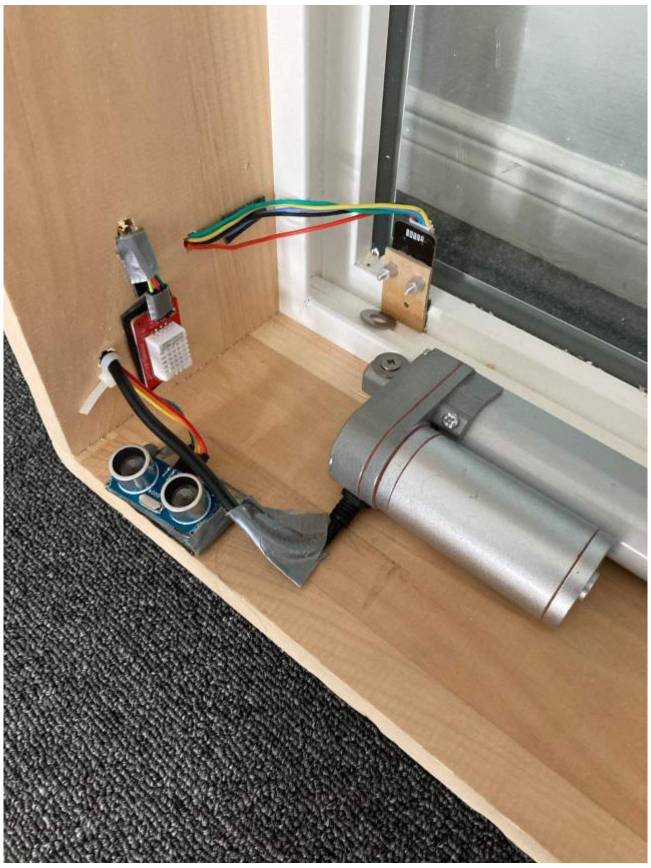


Figure 31: Mount the Ultrasonic Sensor directly below the blinds for position feedback

Housing for sensors

The housing consists of two parts: the base and cover, both of which are 3D printed from ABSI. The base has slots for each sensor which are mounted on the corresponding slot and fixed with screws. Connections between wires and Arduino are released by pulling the wires through the small hole at the rear of the housing. The base and cover should be installed with the corresponding gasket. The contact surfaces of the rain sensor, temp sensor and housing base and the small hole should be coated with nano-coating to ensure the airtightness of the housing. Afterward, the housing is fixed on the outside of the *smart home window* through the mounting holes on both sides of the housing.

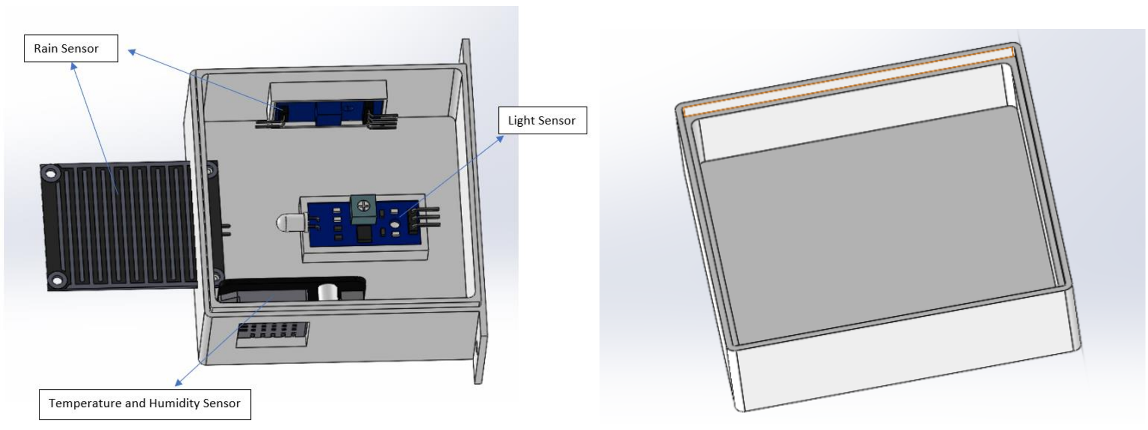


Figure 32: housing base and cover

Emergency Failsafe Method

Although the Pull-pin Failsafe was not implemented in the prototype, the sliding window failsafe was still implemented. However, ultrasonic sensors were used to demonstrate this in place of touch sensors due to the inability to receive them within the timeframe of the project. This ultrasonic sensor was mounted directly at the bottom of the housing and pointed to the top of the housing. When no interference is detected, the length that the sensor should read should be the height of the window. When the distance read is shorter, it indicates that there is a disturbance. The ultrasonic sensor is placed in front of the moving sliding window as shown below.

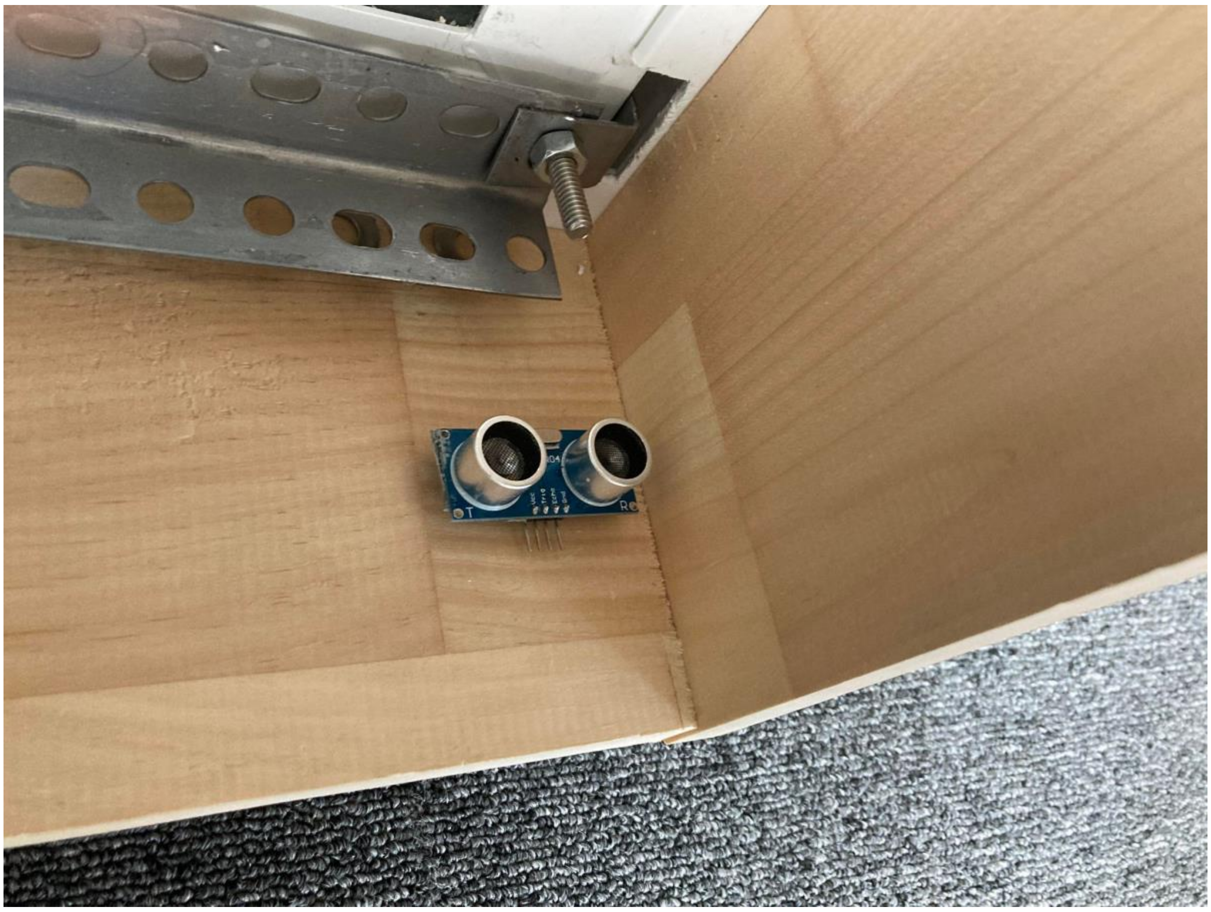


Figure 33: Ultrasonic sensor placement for sliding window failsafe

Electrical Development

Electrical Schematic

The initial pin layout for the Raspberry Pi 4B microcomputer is shown below in Figure 34. All sensors, actuators and motor drivers are connected to the Raspberry Pi and powered by the 12 V power supply. The 12 V power supply is converted to 5 V through the motor driver.

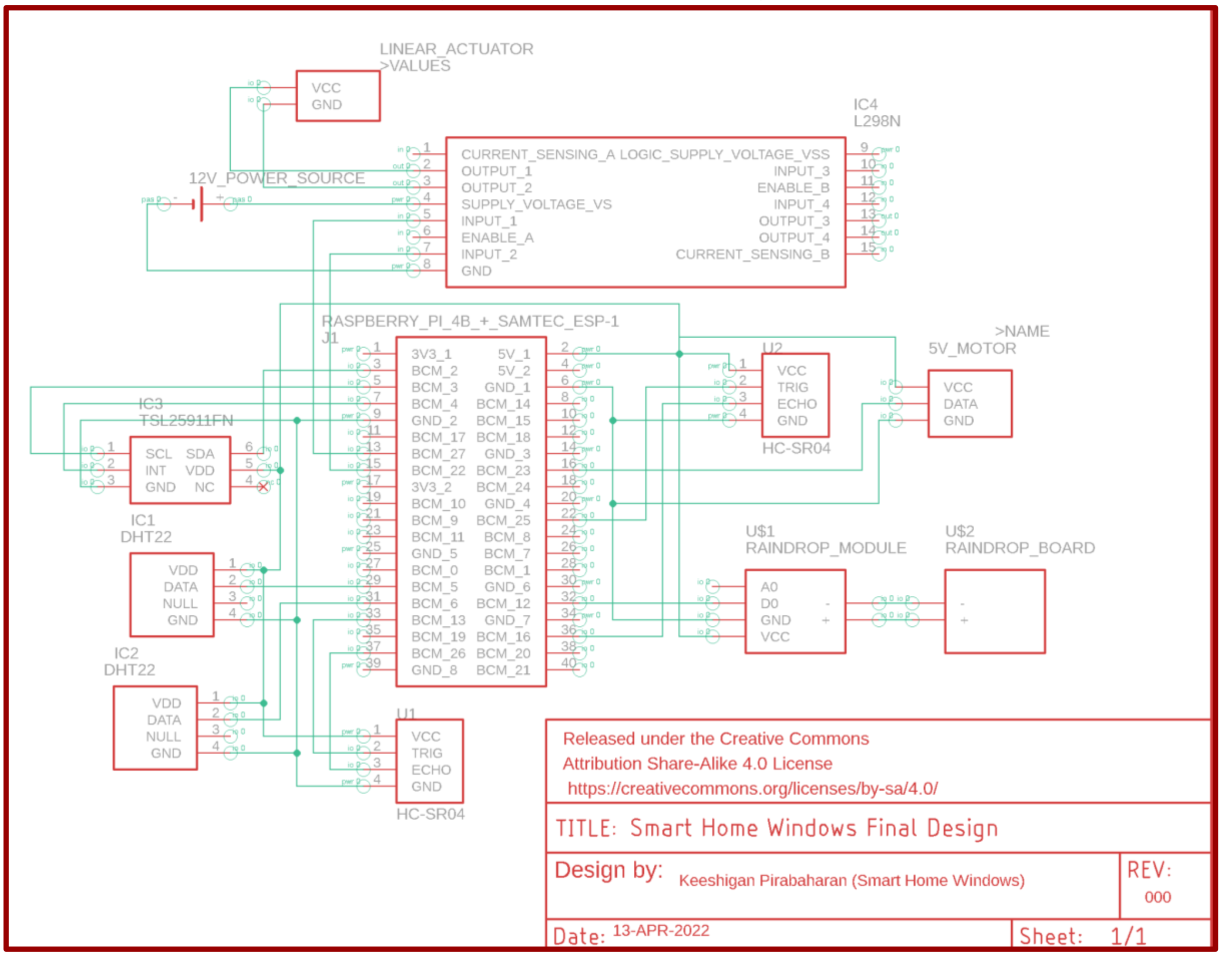


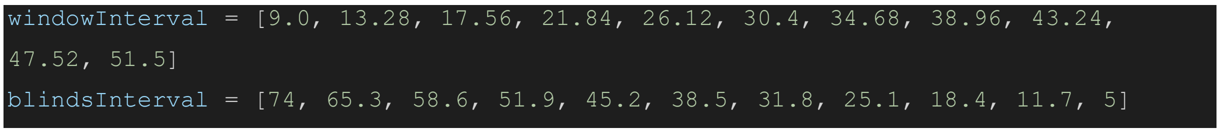
Figure 34: Final Electrical Schematic of Prototype

Control System Implementation

In order to set the blinds and windows to incremental positions, the maximum and minimum distance observed by the ultrasonic sensor was obtained for linear interpolation.

$$\begin{aligned} &\mathit{Distance\ at\ Increment\ N} \\ &\quad = \mathit{Minimum\ Distance} + \frac{(\mathit{Maximum\ Distance} - \mathit{Minimum\ Distance})}{11} * N \end{aligned}$$

Using the equation above, an array was created to store the distance values at each increment for both blinds and windows (shown in Figure 33). As the window is moving further from the sensor when being closed the distance that accounts for 100% of the window position is 51.5 cm. The blinds are moving closer to the sensor when being closed so the distance for 100% of the blinds position is 5 cm.



```
windowInterval = [9.0, 13.28, 17.56, 21.84, 26.12, 30.4, 34.68, 38.96, 43.24,
47.52, 51.5]
blindsInterval = [74, 65.3, 58.6, 51.9, 45.2, 38.5, 31.8, 25.1, 18.4, 11.7, 5]
```

Figure 35: Array of Distance Values to be Checked by Ultrasonic Sensor

Safety Stop Failsafe Method

In the prototype, the safety stop fail safe method for the sliding window translation was implemented using an ultrasonic sensor instead of a touch sensor as previously stated. The logic behind the failsafe was to declare a normal distance value which corresponds to the height of the window and poll ultrasonic sensor reading cyclically to ensure that the distance remains relatively the same. When a smaller distance is detected, this indicates that there is an obstruction of the linear actuator path, which could potentially be a person’s hand or something that should not be in contact with the window. This then disables the window from closing. This does not happen when opening because it is assumed that this safety stop fail safe only protects the user from the sliding window closing on them.

Software Development

Device Main Program

The device’s main program is the code that will run on start-up and control the program flow and operation of the device. The final iteration of the device's main program (Main.py) can be seen in *Appendix 3*. The program flowchart of the Main program can be seen in Figure 36.

The main program is set to run on the boot up of the device. It will first require the pair of the device with a smartphone. This will be done through the mobile app where the user will create an account. The program will initialise all the components it needs to run. These components include the sensors, actuators, app connection, and python libraries. Once a user has paired the device and all components have been initialised, the device can operate the window and blinds.

In order to provide the user with a variety of options to control, the device offers 3 modes: Manual, Auto, Smart. How each mode operates is explained further in the *Device Modes* subsection. The device will run a continuous loop of checking inputs and produce a value of what level the blinds and window should be. The device will then adjust the blinds and window if needed.

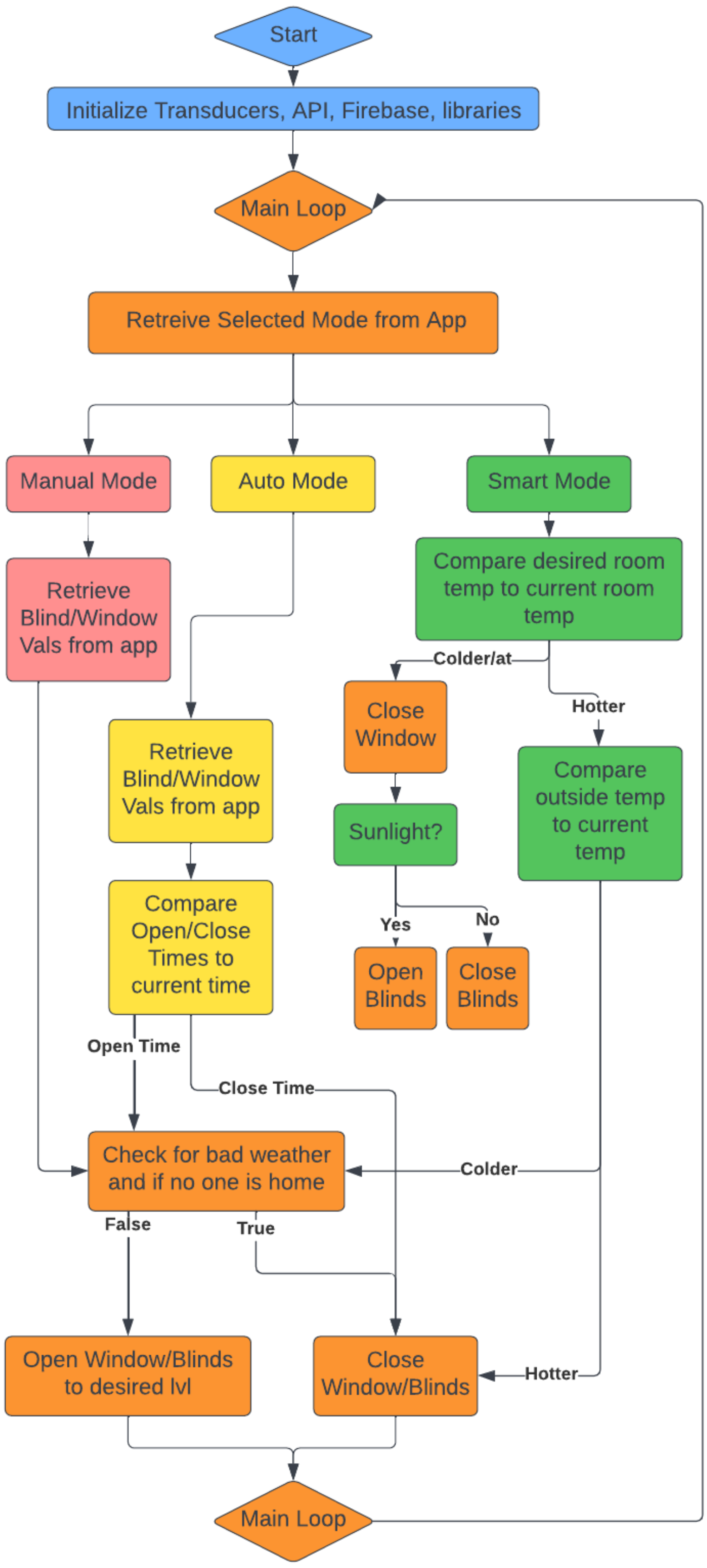

Start
Initialize Transducers, API, Firebase, libraries
Main Loop
Retreive Selected Mode from App
Manual Mode
Auto Mode
Smart Mode
Retrieve Blind/Window Vals from app
Compare desired room temp to current room temp
Colder/at
Hotter
Close Window
Retrieve Blind/Window Vals from app
Sunlight?
Compare outside temp to current temp
Yes
No
Open Blinds
Close Blinds
Compare Open/Close Times to current time
Open Time
Close Time
Check for bad weather and if no one is home
Colder
False
True
Open Window/Blinds to desired lvl
Close Window/Blinds
Hotter
Main Loop

Figure 36: Main Program Flowchart

Device Modes

Manual mode allows the device to be controlled by the user. The user can simply set the window and blinds to a specific level through the mobile app. For example, the user can choose to fully open the window and halfway open the blinds through the app. This mode allows the user to operate the windows and blinds how they normally are with the added convenience of remote control anywhere in the house.

In auto mode, the device operates automatically at set times determined by the user. The user inputs specific times they want the windows and blinds to be open. For example, the user can set the blinds and window open at 8 am and close at 5 pm every day.

In smart mode, the device operates on its own with only 1 user input. This mode is meant to require minimal user input and operate autonomously without the user. The user provides the desired room temperature. The device uses this along with the sensor to determine the optimal times to open and close the window and blinds. This mode helps to regulate the temperature and lighting inside the home, allowing less dependency on HVAC systems and electric lighting, respectively, while also protecting the home by closing windows and blinds during bad weather and when no one is home. The decisions based on the sensor conditions can be seen in figure 36.

Finally, the device checks the geofence feature. This feature makes sure the windows and blinds are closed even though the decision is to open the windows. The geofence feature is an optional safety feature that prevents break-ins when no one is home. IT is discussed in detail in the Android Application section.

Android Application

To implement each device mode, the most appropriate built-in Android widget was used to retrieve values from the user. For smart mode, two NumberPickers were used to retrieve the desired temperature and unit (see Figure 37 for smart mode layout). In automatic mode four TimePickers were used to retrieve open and close times for both the blinds and windows (see Figure 38 for automatic mode layout). For manual mode two SeekBars were used to retrieve blinds and windows position in 11 increments (0% to 100%) (see Figure 39 for manual mode layout).

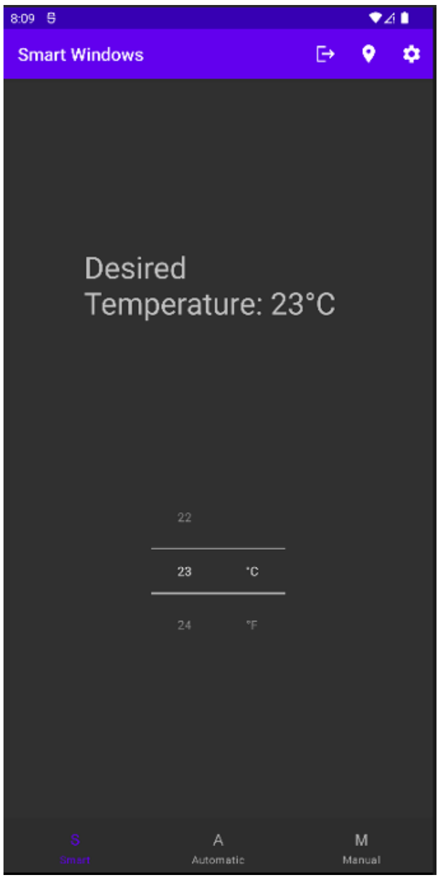


Figure 37: Smart Mode with Two NumberPickers

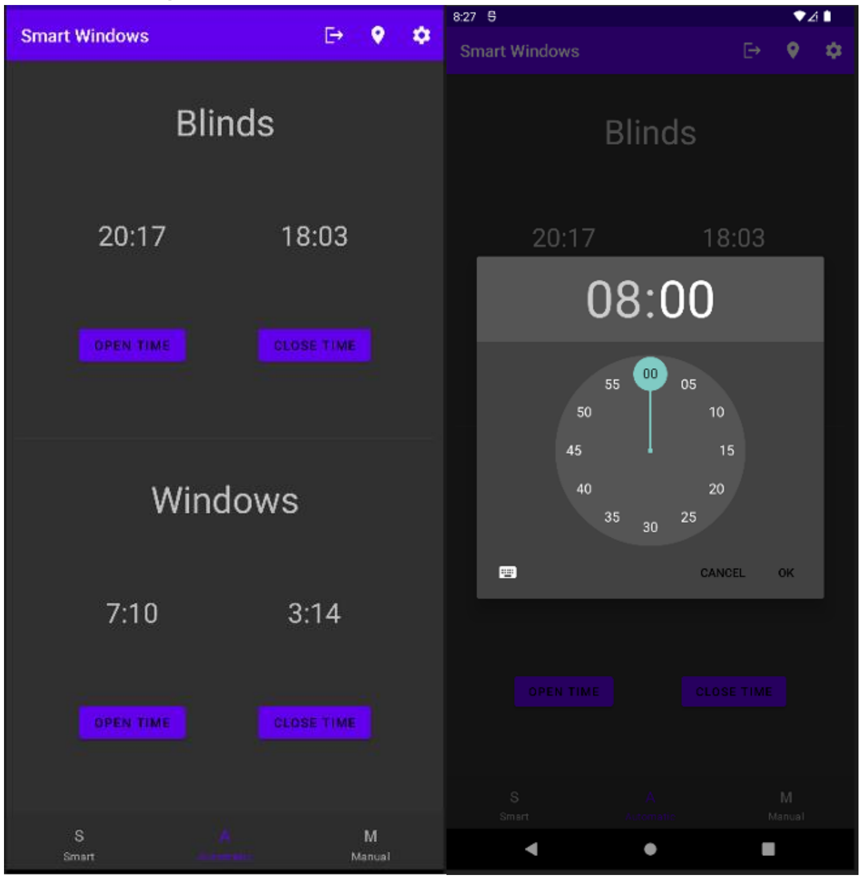


Figure 38: Automatic Mode with Four TimePickers (left) TimePicker View (right)

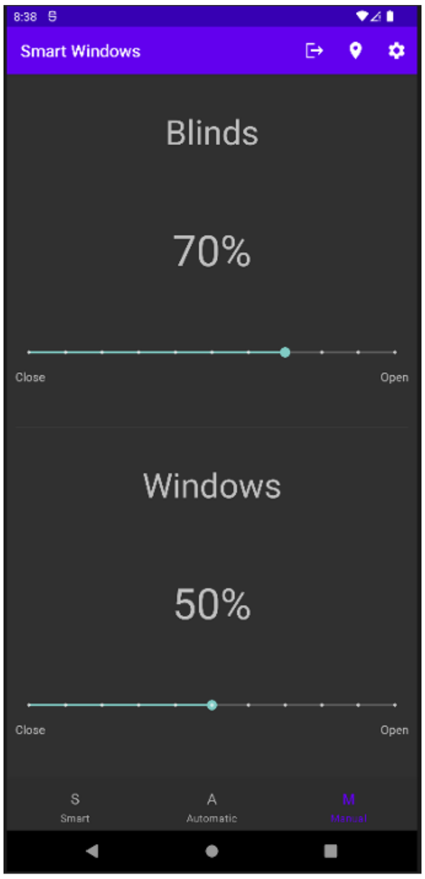


Figure 39: Manual Mode with Two Seekbars

Geofence

In the location settings of the SHW app, a geofence with a radius of 75 m can be set around the setup location of the SHW device with a button click (see Figure 40). A range of 75 m was chosen as geofences work best if the minimum radius is 100 m unless there is Wi-Fi in the area where minimum distance can be 50 m [24]. The average was taken to be the geofence radius as it is greater than the radius of the average home and can trigger enter/exit events accurately. Entering or exiting the geofence would trigger an event that pushes a Boolean value of true if the user is in the geofence or false if outside the geofence to Firebase under their encrypted user id (see Figure 41). Geofences were implemented to work while the application is not running with the use of PendingIntents. A PendingIntent is a token that gives a foreign application permission to execute predefined code.

By keeping location information on the phone and only transferring Boolean values, privacy leaks can be mitigated through transfer. Values stored on Firebase are encrypted using AES-256 which means time to crack using a quantum computer is 2.29*10^32 years [25]. The only method of breaking into the house would be by stealing a household member's phone and going near the house. However, like in any other situation where a phone is stolen it's advised that the user immediately locks their phone using their Google Account at which point location permissions to the application is denied and geofences are disabled. Geofences are implemented to work with multiple users and are fully functional for the final design of the product and this feature can be turned on or off as the users' desire.

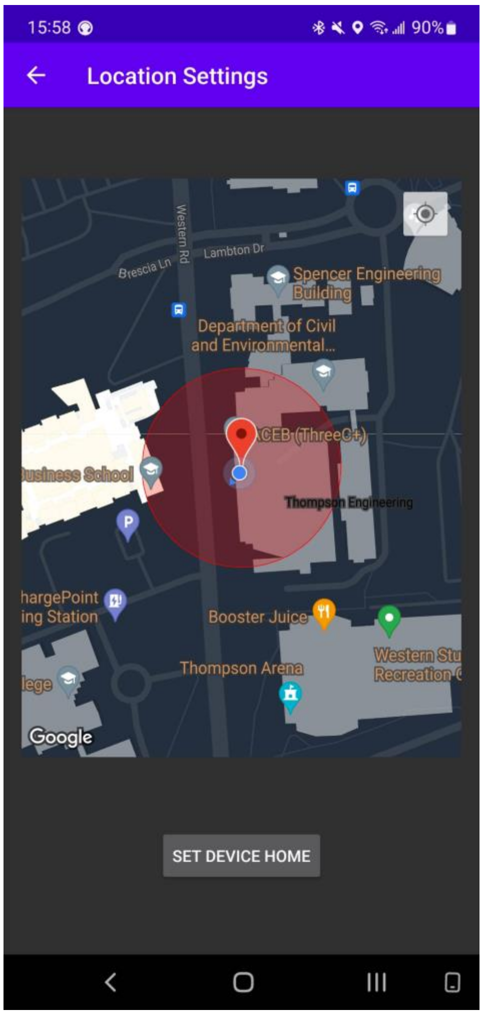


Figure 40*:* Geofence Setup Screen

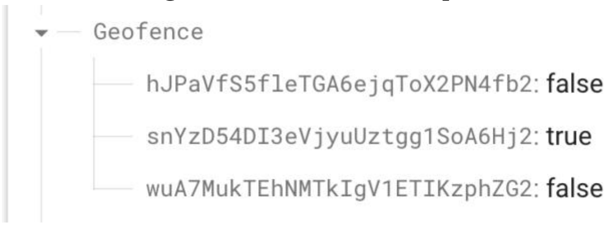

```
Geofence
    hJPaVfS5fleTGA6ejqToX2PN4fb2: false
    snYzD54DI3eVjyuUztgg1SoA6Hj2: true
    wuA7MukTEhNMTkIgV1ETIKzphZG2: false
```

Figure 39: Firebase Geofence Values

Prototype Testing and Evaluation

Mechanical Testing

The three main components that were tested in the prototype were the sliding window method, the blinds method, and the housing method.

For the sliding window translation method, the main concern was that the sliding window would tilt because the linear actuator applies a force that is significantly lower than its centre of gravity. Theoretically, this would cause a moment and the window would rotate about the centre of gravity. However, through testing, we ensured that the linear actuator was positioned perpendicular to the side of the window through attaching it with a steel bracket. The linear actuator was also secured to the frame of the window so that it was in a fixed position. This ensured that the window had a fixed plane of motion and was unable to rotate.

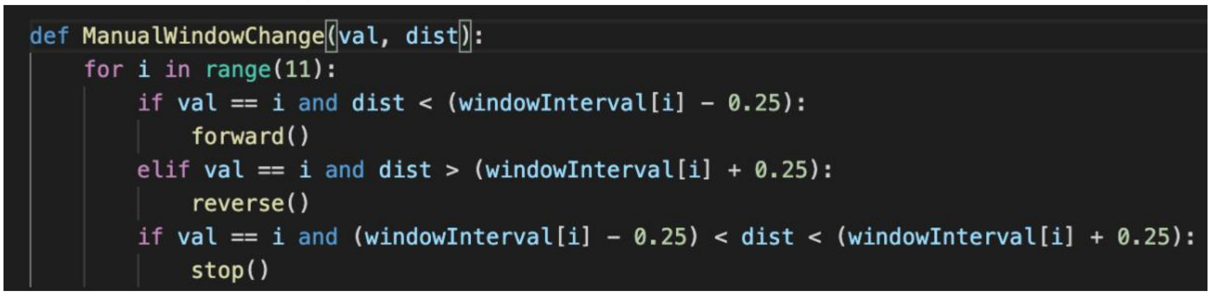
For the blind's method, there were two primary concerns. The first concern involved the inability to mount the bearings to the servo motor and the shaft. This caused the servo motor to handle axial loads which were unintended in the original design. Through testing and calculations, it was found that the FS90R servo would not be suitable without the use of bearings to handle the axial loads. To mitigate this, we decided to use a stronger motor that could handle the torsional load along with axial loads of the shaft.

The second concern involved the blinds being bunched up along the shaft. Through testing, it was discovered that there were three primary reasons for this: an uneven shaft due to manufacturing issues, misalignment of the shaft, and the lack of constant tension to ensure that the blinds were contracted evenly. To mitigate this, the shaft was smoothened and realigned, and a hardwood dowel was mounted at the bottom of the curtain to ensure there was a weight that ensured constant tension on the shaft.

For the housing, the primary tasks were waterproofing and transmittance tests. For the waterproofing test, the gaskets were compressed so that water could not enter. From the Solidworks simulation, this housing could withstand a certain degree of water pressure. As a result, only the airtightness of the housing needs to be considered. By pressing the housing into the water, the quality sealing of the housing was assured. To test transmittance, values from the sensors in the housing were recorded frequently. The values with the values from the sensor placed outside the housing were compared. The internal value was about ¾ of the external values. However, when amplifying these values and inputting them to the Raspberry Pi, the impact was minimal. it was decided that this housing design was satisfactory for the intended purpose.

Controls Testing

During testing it was observed that ultrasonic sensor values are refreshed at a rate of 0.25 cm/pulse when blinds and windows are opening/closing. To account for the overshoot that will occur, the blinds and windows are set to stop 0.25 cm before the desired position is reached. This is important for the extreme cases of fully open or closed where the blinds should not roll over itself and the window should not crash into the frame (see Figure 42 for control implementation of windows in code, blinds implementation similar). Blinds and windows were opening/closing at a speed of 2.5 cm/s. A speed test was done by measuring the time it took to cover 30 cm with a stopwatch. This met our safety constraint of not closing faster than 5 cm/s.



```python
def ManualWindowChange(val, dist):
    for i in range(11):
        if val == i and dist < (windowInterval[i] - 0.25):
            forward()
        elif val == i and dist > (windowInterval[i] + 0.25):
            reverse()
        if val == i and (windowInterval[i] - 0.25) < dist < (windowInterval[i] + 0.25):
            stop()
```

Figure 42: Code Implementation of Windows Position Adjustment

Geofence

The geofence API was tested by walking and driving out of the geofence zone. It was observed that when walking out of the geofence zone it took the application 30 seconds to trigger blinds and windows closed event and driving took 10 seconds to trigger the event. This is well within the constraint of having the device close within a minute of the last person leaving the house and would not give burglars enough time to enter the user's house. A notification was appropriately pushed to the user to ensure the triggering event occurred and blinds and windows were closed. Similar results were given for users entering the geofence (see Figure 43).

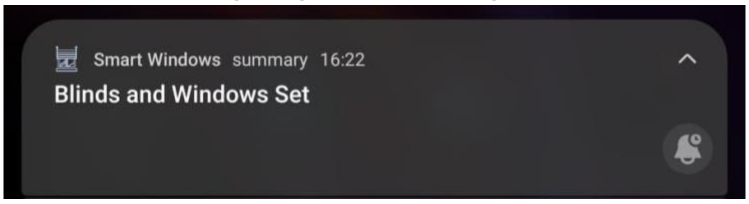

Smart Windows summary 16:22
Blinds and Windows Set

Figure 43: Notification for Blinds and Windows Being Set on Entering Geofence

Weather Sensors

The light sensor was tested by placing it against an existing window and observing the value of sunlight at different times. During hours of sunlight the range of lux values read by the light sensor was from 106-197. During nighttime the lux values ranged from 15-58. A threshold value of 100 was used to differentiate between daytime and nighttime. In order to test different lighting environments, the light sensor was moved to different spots on multiple windows around the house.

The rain sensor was tested by placing water droplets on the sensor to simulate rainfall. A completely dry sensor produces a value of 1.0. Placing water droplets on the sensor produces values ranging from 0.25-0.55. A threshold value of 0.75 was used to differentiate between rain and no rain.

To evaluate the temperature sensors, The outside temperature sensor was placed outside the window exposed to the outside environment and the inside temperature was placed inside the home next to the window. The outside temperature was compared to the online weather data while the inside temperature sensor was compared to the thermostat inside the home. Both sensors were within 1°C of their respective comparisons. To ensure the inside temperature sensor was not being influenced by the outside and vice versa, in the prototype, the outside temperature sensor was placed in a housing outside the window and the inside sensor was placed on the side of the window opposite of where it opened. To test this a heat source was placed next to a single sensor. It was observed that only the sensor exposed to the heat source would change in reading.

Prototype Modification and Improvements

Actuator Implementation

For the linear actuator method, no modifications were made to improve the prototype. However, the GB37Y3530-131 motor was used in place of the FS90R due to its stronger stall torque. One problem was that this motor has a no-load speed of 83 RPM, which translates to a vertical translation speed of 13.79 cm/s, In the constraints, it is indicated that the speed should be less than 5cm/s. To mitigate this, the input voltage was reduced by using resistors until it met the constraints.

Housing Implementation

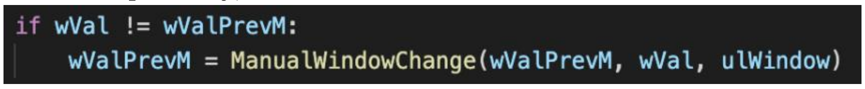
The housing prototype has some small cracks at the rear mounting hole during installation. In response to this problem, the thickness of the rear baffle has been increased by 1 mm while shims have been added during installation. In addition, there is some extra unnecessary ABSI material in the slot of the rain sensor that needs to be cleaned manually. The main reason for this is that the slot is designed on the side of the housing whereas the excess material is what keeps this part supported during the 3D printing process. The solution is to set all the slots at the bottom of the housing base, which not only eliminates the need for extra materials, but also saves printing time.

Arduino Implementation

From concept selection, the plan was to connect all sensors and actuators to a Raspberry Pi 4B as the sole microcomputer of the SHW device. However, during implementation of the rain sensor, it was found that due to the lack of knowledge of working with sensors using Python, a voltage reading could not be read. Furthermore, the linear actuator kept stalling when controlled by Raspberry Pi. To show full functionality of the SHW device, an Arduino was used synchronously with Raspberry Pi using Firmata, an intermediate protocol that connects an embedded system to a host computer [26]. Using PyFirmata, the Arduino can be directly controlled through a USB connection by Raspberry Pi using Python without the need of the Arduino IDE (see Appendix 4 for Firmata Code ran on Arduino).

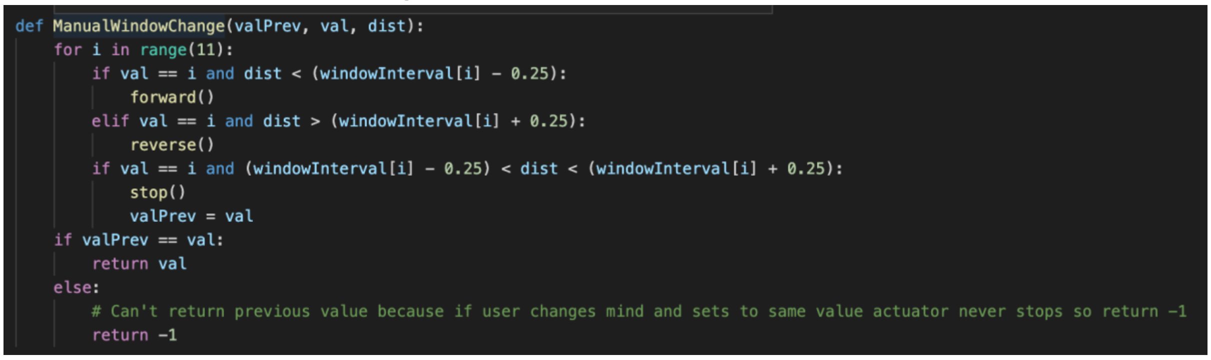
Controls Modification

During prototyping it was noticed that the blinds and window would occasionally adjust itself after being set because the ultrasonic reading would fall out of the range of the conditions that were set for it to stop. To prevent this, blinds and windows operation is to only occur when a change in value from Firebase was observed and after blinds and windows position was reached the previous value variable is set to the current variable (see Figure 44 and 45 for the main code and function call of windows operation respectively).



```python
if wVal != wValPrevM:
    wValPrevM = ManualWindowChange(wValPrevM, wVal, ulWindow)
```

Figure 44: Main Code Call of Function



```python
def ManualWindowChange(valPrev, val, dist):
    for i in range(11):
        if val == i and dist < (windowInterval[i] - 0.25):
            forward()
        elif val == i and dist > (windowInterval[i] + 0.25):
            reverse()
        if val == i and (windowInterval[i] - 0.25) < dist < (windowInterval[i] + 0.25):
            stop()
            valPrev = val
    if valPrev == val:
        return val
    else:
        # Can't return previous value because if user changes mind and sets to same value actuator never stops so return -1
        return -1
```

Figure 45: Windows Operation Function

Resources and Budget

Team Skills

Table 53: Strength and weakness

Member	Strengths	Weaknesses
Keeshigan	● Programming (Python) ● Android App Development ● Sensor Selection ● Web App Programming ● Documentation	● SolidWorks ● Hands-on Construction
Douglas	● SolidWorks ● Actuator Selection ● Material Selection ● Programming	● Electronics
Pahuldeep	● Programming (Python) ● Sensor Selection ● Code Documentation	● SolidWorks ● Hands-on Construction
Xingmin	● SolidWorks ● Electronics ● Housing Design	● Programming

External Support

The people that provided support and knowledge include Eugen Porter, and Dr. Ladak, our faculty advisor. Their design opinions and expertise proved to be of great help.

Funding

The budget limit for the prototype was $125 for each member with an additional $100 for machining. This puts the budget for the prototype at a combined total of $600.

Time

In the time since Detailed Design Documentation, the team spent most of the time completing the prototype to be fully functional for the showcase.

Equipment

The team had access to the prototyping lab and machine shop in the engineering buildings where we could use soldering irons, saws, drill presses, sanders, and more to create all necessary parts in the assembly.

Materials

The full bill of materials for the prototype and the final product are listed in Table 54 and Table 55. The primary difference in materials used between these two designs is that the method of position feedback are different. In the intended design, the position control of the linear actuator and the servo motor are done using absolute encoders. However, in the prototype, ultrasonic sensors were used for this. Also, the safety feature in the intended design relies on capacitive touch sensors, whereas a similarly functioning method was implemented using an ultrasonic sensor in the prototype.

Table 54: Prototype Bill of Materials

Item	**Description**	**Supplier**	**Price ($)**	**Quantity**	**Total Price ($)**
JELD-WEN Windows & Doors 36-inch x 48-inch 5000 Series Single Hung Vinyl Window with 4 9/16-inch Frame	Window	Home Depot	$268.68	1	$268.68
Linear Actuator Electric Linear Telescopic Rod Linear Motion 12V DC 450mm Silver	Linear Actuator	Amazon	$90.34	1	$90.34
Feetech FS90R 360 Degree Continuous Rotation Micro Servo Motor + RC Tire Wheel for Arduino (Pack of 2)	Servo Motor	Amazon	$18.95	0.5	$9.48
CQRobot Ambient Light Sensor	Light Sensor	Amazon	$19.99	1	$19.99
Elegoo HC-SR04 Ultrasonic Module Distance Sensor for Arduino (Set of 5)	Ultrasonic Sensor	Amazon	$13.99	0.6	$8.40
AM2302 DHT22 Digital Temperature and Humidity Measurement Sensor Pack of 2	Temperature sensor	Amazon	$15.88	0.5	$7.44
Fielect Raindrops Module Raindrop Detection Sensor Humidity Rain Weather Sensitivity Monitoring for Arduino 1pcs	Rain Sensor	Amazon	$6.39	1	$6.39
Raspberry Pi 4	Raspberry-Pi	Raspberry -Pi	$35	1	$35
L298N Dual H-Bridge Motor Drive Controller for Arduino Smart Car Robot Power Stepper Motor Driver Module	Power Stepper Driver	Amazon	$16.51	1	$16.51
Belker 36W Universal 3-12V AC DC Adapter Power Supply 1-3A, 3000mA Amp Max.	Power Supply	Amazon	$27.90	1	$27.90
Paulin 1-inch x 1/4-inch x 96-inch Steel Slotted Angle - Zinc	Brackets	Home Depot	$16.87	1	$16.87

Alexandria Molding 1/8-inch x 24-inch x 24-inch White Melamine Handy Panel	Wood Panel	Home Depot	$5.25	1	$5.25
Ball Bearing Open, Trade Number R2, for 1/8" Shaft Diameter	Bearing	McMaster	$6.57	2	$13.14
Bearing Housing for 3/8" Wide and 0.375" OD Bearing	Bearing Housing	McMaster	$9.76	2	$19.52
Alexandria Molding Hardwood Dowel 1-1/4 In. x 48 In. Avocado	Wooden Dowel	Home Depot	$14.67	1	$14.67
Passivated 18-8 Stainless Steel Pan Head Phillips Screw 10-24 Thread, 1" Long (50 pack)	10-24, 1” Long Screw	McMaster	$10.49	0.08	$0.84
18-8 Stainless Steel Washer for No. 10 Screw Size, 0.203" ID, 0.438" OD (100 Pack)	No. 10 Washer	McMaster	$2.40	0.1	$0.24
Steel Hex Nut Zinc-Plated, 10-24 Thread Size (100 Pack)	No. 10 Bolt	McMaster	$2.09	0.04	$0.09
Brass Phillips Flat Head Screws for Wood No. 5 Size, 1" Long (50 pack)	Screw For Dowel	McMaster	$15.35	0.04	$0.62
				TOTAL:	**$561.37**

Table 55: Final Product Bill of Materials

Item	**Description**	**Supplier**	**Price ($)**	**Quantity**	**Total Price ($)**
JELD-WEN Windows & Doors 36-inch x 48-inch 5000 Series Single Hung Vinyl Window with 4 9/16-inch Frame	Window	Home Depot	$268.68	1	$268.68
Linear Actuator Electric Linear Telescopic Rod Linear Motion 12V DC 450mm Silver	Linear Actuator	Amazon	$90.34	1	$90.34
Feetech FS90R 360 Degree Continuous Rotation Micro Servo Motor + RC Tire Wheel for Arduino (Pack of 2)	Servo Motor	Amazon	$18.95	0.5	$9.48
Multiturn absolute encoder BSM58	Absolute Encoder	Pepperl-Fuchs	$68.95	2	$137.90
CQRobot Ambient Light Sensor	Light Sensor	Amazon	$19.99	1	$19.99
Capacitive Touch Slider Wireling	Touch Sensor	Tiny-Circuits	$4.95	10	$49.50

AM2302 DHT22 Digital Temperature and Humidity Measurement Sensor Pack of 2	Temperature sensor	Amazon	$15.88	0.5	$7.44
Fielect Raindrops Module Raindrop Detection Sensor Humidity Rain Weather Sensitivity Monitoring for Arduino 1pcs	Rain Sensor	Amazon	$6.39	1	$6.39
Raspberry Pi 4	Raspberry-Pi	Raspberry-Pi	$35	1	$35
L298N Dual H-Bridge Motor Drive Controller for Arduino Smart Car Robot Power Stepper Motor Driver Module	Power Stepper Driver	Amazon	$16.51	1	$16.51
Belker 36W Universal 3-12V AC DC Adapter Power Supply 1-3A, 3000mA Amp Max.	Power Supply	Amazon	$27.90	1	$27.90
Paulin 1-inch x 1/4-inch x 96-inch Steel Slotted Angle - Zinc	Brackets	Home Depot	$16.87	1	$16.87
Alexandria Molding 1/8-inch x 24-inch x 24-inch White Melamine Handy Panel	Wood Panel	Home Depot	$5.25	1	$5.25
Ball Bearing Open, Trade Number R2, for 1/8" Shaft Diameter	Bearing	McMaster	$6.57	2	$13.14
Bearing Housing for 3/8" Wide and 0.375" OD Bearing	Bearing Housing	McMaster	$9.76	2	$19.52
Alexandria Molding Hardwood Dowel 1-1/4 In. x 48 In. Avocado	Wooden Dowel	Home Depot	$14.67	1	$14.67
Passivated 18-8 Stainless Steel Pan Head Phillips Screw 10-24 Thread, 1" Long (50 pack)	10-24, 1" Long Screw	McMaster	$10.49	0.08	$0.84
18-8 Stainless Steel Washer for No. 10 Screw Size, 0.203" ID, 0.438" OD (100 Pack)	No. 10 Washer	McMaster	$2.40	0.1	$0.24
Steel Hex Nut Zinc-Plated, 10-24 Thread Size (100 Pack)	No. 10 Bolt	McMaster	$2.09	0.04	$0.09
Brass Phillips Flat Head Screws for Wood No. 5 Size, 1" Long (50 pack)	Screw For Dowel	McMaster	$15.35	0.04	$0.62
				TOTAL:	**$730.90**

Conclusions

Mechanical Conclusions

One conclusion that can be made about the mechanical design is that component selection is very important when it comes to competitively pricing the product. One example of this is shown by how an existing off-the-shelf window was used as the housing for prototyping. When designing a product, it is crucial to have a good idea about the quantity of the products that would be made. If this product undergoes large-scale manufacturing, it would be best to look into manufacturers to assemble the intended design instead of the off-the-shelf solution. This would also be beneficial, since other parts like the linear actuator bracket could be freely designed and do not have to worry about unusual dimensions of existing products, which occurred with the off-the-shelf window.

Another example of this is in the housing design. For prototyping, it was decided that the housing would be 3D-printed. However, this would not be feasible in large-scale productions because it would take too much time and it would be expensive to manufacture. The solution for this is to create this part via plastic moulding, which could produce a large number of parts in less time.

Electrical Conclusions

Overall, the only issue with electrical implementation was the lack of knowledge in implementing the rain sensor and the linear actuator with Raspberry Pi. Unfortunately, the goal of having one microcomputer for the prototype was not me, however with the use of an Arduino full functionality was shown through the prototype.

Controls Conclusions

From this project, it was realised that a simpler feedback system is often more robust and avoids unnecessary computations and complications. This was evident through using absolute encoders instead of using the ultrasonic sensors for position control. When using the ultrasonic sensors, there were challenges such as avoiding interferences and disturbances, validating sensor readings, and overshoot. All of these issues were mitigated when using the absolute encoder in its place.

Software Conclusions

It was discovered that when making a simple device “smart”, some of the decisions the device makes are not so straightforward. When using technology to automate tasks done by humans, behavioural aspects have been considered as well. For example, in the case of an empty home, the Smart Home Window will close the windows and blinds to prevent intruders from entering the house. However, a behavioural aspect that was not considered is that closed blinds and windows may give off a signal that the house is empty, making it more vulnerable to break-ins. Humans also may make different ways to operate their windows and blinds given the same conditions. Since humans make decisions based on past experiences and knowledge, there isn't a “one size fit all” algorithm. In order to mitigate this, the device should learn the behaviour and patterns of the user to provide the best automation experience.

Recommendations

Mechanical Recommendations

A recommendation that could be made for mechanical design involves determining realistic quantities that are to be produced. In the early stages of the design, it may be best to do rapid prototyping, which involves 3D printing and potentially using existing products in the bill of materials. However, as production becomes larger, it is crucial to explore different means of production that have larger batch sizes, quicker production and lower costs.

Electrical Recommendations

Since there was a lack of expertise in getting readings from the rain sensor onto the Raspberry Pi, research into libraries and posting on help forums should have been done early to ensure there is time to develop functional code.

Controls Recommendations

One of the main reasons ultrasonic sensors provide poor feedback is the high downtime between each pulse feedback. If the goal of the system is to reach critical damping, each electrical component that makes up the system needs to be selected where the refresh frequency meets a set constraint.

Software Recommendations

The “smart” mode of the product makes a single decision based on all the inputs it receives. These decisions are not decided by the device itself but by the programmer. In order for the device to truly make decisions, the device algorithm must learn the user’s preferences and behaviour. If this device were to improve, a deep learning algorithm could be used to accomplish this. For example, a user keeps their windows closed when working because they live next to a very busy and loud street. Instead of the user switching to manual mode to close the windows, the device can learn the user’s behaviour and store this information. This would allow the device to be truly autonomous. However, this also requires the introduction of many more parameters and very complex code. The labour and cost of this should also be considered if this option is pursued.

References

[1] “How do burglars break into houses?,” ADT Security Systems. [Online]. Available: https://www.adt.com/resources/how-do-burglars-break-into-houses. [Accessed: 02-Oct-2021].

[2] “Office design,” *Sustainable Buildings Initiative*. [Online]. Available: https://challenge.abettercity.org/toolkits/emissions-reduction-toolkits/energy-efficiency/office-design/?toolkit=223. [Accessed: 02-Oct-2021].

[3] "How long do homeowners stay in their homes?," *www.nar.realtor*. [Online]. Available: https://www.nar.realtor/blogs/economists-outlook/how-long-do-homeowners-stay-in-their-homes. [Accessed: 12-Apr-2022].

[4]"Research and Markets: Global Home Automation and Control Market 2014-2020 - Lighting Control, Security & Access Control, HVAC Control Analysis of the $5.77 Billion Industry | Reuters," *web.archive.org*, May 05, 2016. [Accessed: 02-Oct-2021].

[5] "Motorized blinds - automatic blinds - remote control shades: Select blinds canada," *Select Blinds Canada - Custom Blinds Shipped Free to your door*. [Online]. Available:

[6] "Olideauto Intelligent Smart AC Motor Automatic Window opener with wired US standard WIFI push panel phone app control,compatible with Alexa&Google Assistance," *Amazon.ca: Home*. [Online]. Available: https://www.amazon.ca/dp/B0834XTF45?psc=1&th=1&linkCode=gs2&tag=smarthomepe01-20. [Accessed: 02-Oct-2021].

[7] Feldco, "What are standard window sizes? helpful guide for homeowners," Feldco Factory Direct, 15-Aug-2019. [Online]. Available: https://www.4feldco.com/articles/standard-window-sizes/. [Accessed: 22-Nov-2021].

[8]"Canadian Standards Association (CSA) Enclosure Rating Definitions," *Integrated Rectifier Technologies Inc.* https://irtrectifier.com/technical-info/enclosure-rating-definitions/canadian-standards-association/ [accessed Apr. 12, 2022].

[9]Audio Advice, "What are the Best Motorized Shade Options?," 25-Jun-2018. [Online]. Available: https://www.youtube.com/watch?v=4k4w7qbct5w. [Accessed: 12-Apr-2022].

[10]M. Hess, "IEC 61140 - safety classes description," *Intouch-quality.com*. [Online]. Available: https://www.intouch-quality.com/blog/iec-61140-safety-classes-descriptions. [Accessed: 12-Apr-2022].

[11]"How does a remote control work?," *Wonderopolis.org*. [Online]. Available: https://wonderopolis.org/wonder/how-does-a-remote-control-work. [Accessed: 12-Apr-2022].

[12]"American White Oak wood," *Matweb.com*. [Online]. Available: https://www.matweb.com/search/datasheet_print.aspx?matguid=44cdf6b01d004baaa7e95105 75891dc3&n=1. [Accessed: 14-Apr-2022].

[13]"ABSi," *Materialise*. [Online]. Available: https://www.materialise.com/en/manufacturing/materials/absi. [Accessed: 14-Apr-2022].

[14]"explain how variations in the intensity of sunlight - Lisbdnet.com." https://lisbdnet.com/explain-how-variations-in-the-intensity-of-sunlight/ (accessed Apr. 12, 2022).

[15]B. W. Jones, "Wavelengths of light from our sun," *Utah.edu*. [Online]. Available: https://webvision.med.utah.edu/2013/10/wavelengths-of-light-from-our-sun/. [Accessed: 14-Apr-2022].

[16]G. R. Veneziani, E. L. Corrêa, M. P. A. Potiens, L. L. Campos, "Attenuation coefficient determination of printed ABS and PLA samples in diagnostic radiology standard beams," 2015.

[17]Wikipedia contributors, "Beer–Lambert law," *Wikipedia, The Free Encyclopedia*, 05-Apr-2022. [Online]. Available:

https://en.wikipedia.org/w/index.php?title=Beer%E2%80%93Lambert_law&oldid=1081160265.

[18]“Electricity prices in Canada 2021,” energyhub.org, 26-Dec-2021. [Online]. Available: https://www.energyhub.org/electricity-prices/#:~:text=the%20two%20seasons.-,Ontario,%24125%20per%20month%20in%202020. [Accessed: 06-Mar-2022].

[19]“What uses the most electricity in my home?,” – Biggest Electricity Hogs | Direct Energy. [Online]. Available: https://www.directenergy.com/learning-center/what-uses-most-electricity-in-myhome. [Accessed: 06-Mar-2022].

[20]“Residential Electricity and Natural Gas Plans,” EnergyRates.ca, 01-Sep-2020. [Online]. Available: https://energyrates.ca/residential-electricity-natural-gas/. [Accessed: 06-Mar-2022].

[21] S. O'Dea, “Mobile android version share worldwide 2018-2021,” *Statista*, 23-Feb-2022. [Online]. Available: https://www.statista.com/statistics/921152/mobile-android-version-share-worldwide/. [Accessed: 06-Mar-2022].

[22] “How much ram does your Android Phone Really Need in 2022?,” *Android Authority*, 14-Jan-2022. [Online]. Available: https://www.androidauthority.com/how-much-ram-do-i-need-phone-3086661/. [Accessed: 06-Mar-2022].

[23] “Gary explains: How much ram does your phone really need in 2019?,” *Android Authority*, 28-Jan-2019. [Online]. Available: https://www.androidauthority.com/how-much-ram-do-you-need-in-smartphone-2019-944920/. [Accessed: 06-Mar-2022].

[24]“Create and monitor geofences,” *Android Developers*. [Online]. Available: https://developer.android.com/training/location/geofencing. [Accessed: 14-Apr-2022].

[25]“128 or 256 bit Encryption: Which Should I Use?,” *Ubiq*, 15-Feb-2021. [Online]. Available: https://www.ubiqsecurity.com/128bit-or-256bit-encryption-which-to-use/. [Accessed: 14-Apr-2022].

[26]B. Zuo, “Firmata tutorial - how to use firmata on arduino compatible boards - seeed wiki,” *Seeedstudio.com*. [Online]. Available: https://wiki.seeedstudio.com/ODYSSEY-X86J4105-Firmata/. [Accessed: 14-Apr-2022].

Appendices

Appendix 1: Get User Location Code

```
package com.example.get_location
// https://www.youtube.com/watch?v=Iq9yQmVOThE&ab_channel=CodingAdventure

import android.content.pm.PackageManager
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.Toast
import androidx.core.app.ActivityCompat
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationServices
import android.Manifest
import android.location.LocationManager
import com.google.android.gms.location.LocationRequest

class MainActivity : AppCompatActivity() {

   private lateinit var fusedLocationProviderClient: FusedLocationProviderClient
   private lateinit var locationManager: LocationManager

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)

      fusedLocationProviderClient = LocationServices.getFusedLocationProviderClient(this)

      findViewById<Button>(R.id.btn_get_location).setOnClickListener{
         fetchLocation()
      }
   }

   private fun fetchLocation() {
      if(ActivityCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_FINE_LOCATION)
!=PackageManager.PERMISSION_GRANTED
         && ActivityCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED
      ) {
         ActivityCompat.requestPermissions(
            this,
            arrayOf(android.Manifest.permission.ACCESS_FINE_LOCATION),
            101
         )
         return
      }
      // Obtain Location using LocationManager
      var location = locationManager.getLastKnownLocation("GPS")
      Toast.makeText(applicationContext, "$location", Toast.LENGTH_LONG).show()

      // Obtain Location using FusedLocationProvider
      fusedLocationProviderClient.lastLocation.addOnSuccessListener {
      if(it != null){
         Toast.makeText(applicationContext, "${it.latitude} ${it.longitude}",
Toast.LENGTH_LONG).show()
      }
      }
   }
}
```

Appendix 2: Geofencing Code

```
//Main Activity Code

package com.example.maps

import android.Manifest
import android.app.PendingIntent
import android.content.pm.PackageManager
import android.graphics.Color
import android.os.Build
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import androidx.annotation.NonNull
import androidx.core.app.ActivityCompat

import androidx.core.content.ContextCompat

import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.OnMapReadyCallback
import com.google.android.gms.maps.SupportMapFragment
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.MarkerOptions
import com.example.maps.databinding.ActivityMapsBinding
import com.google.android.gms.location.Geofence
import com.google.android.gms.location.GeofencingClient
import com.google.android.gms.location.LocationServices
import com.google.android.gms.maps.model.Circle
import com.google.android.gms.maps.model.CircleOptions
import com.google.android.gms.tasks.OnFailureListener
import java.lang.Exception
import java.security.acl.Permission

const val GEOFENCE_ID = "GEOFENCE_ID"
const val TAG = "MapsActivity"
const val GEOFENCE_RADIUS = 200.0
const val CAMERA_ZOOM_LEVEL = 17f
const val LOCATION_ACCESS_REQUEST_CODE = 10001

class MapsActivity : AppCompatActivity(), OnMapReadyCallback,
GoogleMap.OnMapLongClickListener {

    private lateinit var map: GoogleMap
    private lateinit var binding: ActivityMapsBinding
    private lateinit var geofencingClient: GeofencingClient
    private lateinit var geofenceHelper: GeofenceHelper

    var circle: Circle?= null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMapsBinding.inflate(layoutInflater)
        geofencingClient = LocationServices.getGeofencingClient(this)
        geofenceHelper = GeofenceHelper(this)

        setContentView(binding.root)

        // Obtain the SupportMapFragment and get notified when the map is ready to be used.
        val mapFragment = supportFragmentManager
            .findFragmentById(R.id.map) as SupportMapFragment
        mapFragment.getMapAsync(this)
    }

    override fun onMapReady(googleMap: GoogleMap) {
        map = googleMap
        map.uiSettings.isZoomControlsEnabled = true
        // Add a marker in Sydney and move the camera
        val Home = LatLng(43.8817753, -79.0549579)
        with(map){
            //addMarker(MarkerOptions().position(Home).title("Marker at Home"))
            moveCamera(CameraUpdateFactory.newLatLngZoom(Home,
CAMERA_ZOOM_LEVEL))
        }
        //addCircle(map)
        enableUserLocation()
        map.setOnMapLongClickListener(this)
    }

    private fun enableUserLocation(){
        if(ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
            ==PackageManager.PERMISSION_GRANTED||ContextCompat.checkSelfPermission
                (this, Manifest.permission.ACCESS_COARSE_LOCATION)
            ==PackageManager.PERMISSION_GRANTED){
            map.isMyLocationEnabled = true
        }
        else{
            // Ask for Permission
                // FIX, ASK FOR PERMISSIONS BETTER, NO DIALOG SHOWING
                var permissions = mutableListOf<String>()
            if(ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
                !=PackageManager.PERMISSION_GRANTED||
                ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION)
                !=PackageManager.PERMISSION_GRANTED){
                permissions.add(Manifest.permission.ACCESS_FINE_LOCATION)
                permissions.add(Manifest.permission.ACCESS_COARSE_LOCATION)
            }
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
                permissions.add(Manifest.permission.ACCESS_BACKGROUND_LOCATION)
            }
            if(ActivityCompat.shouldShowRequestPermissionRationale(this,
                    Manifest.permission.ACCESS_FINE_LOCATION)){
                // We need to show user a dialog for displaying why the permission is needed and then
            // ask for the permission
                ActivityCompat.requestPermissions(this, permissions.toTypedArray(),
                    LOCATION_ACCESS_REQUEST_CODE)
            }else {
                ActivityCompat.requestPermissions(this, permissions.toTypedArray(),
                    LOCATION_ACCESS_REQUEST_CODE)
            }
        }
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        if(requestCode == LOCATION_ACCESS_REQUEST_CODE){
            if(grantResults.isNotEmpty() &&grantResults[0] ==
PackageManager.PERMISSION_GRANTED){
                // We have the permission
                map.isMyLocationEnabled = true
            }else{
                // We do not have the permission
            }
            super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        }
    }

    override fun onMapLongClick(latLng: LatLng) {
        map.clear()
        addMarker(latLng)
        addCircle(latLng, GEOFENCE_RADIUS)
        addGeofence(latLng, GEOFENCE_RADIUS.toFloat())
    }

    // Add geofence
    private fun addGeofence(latLng: LatLng, radius: Float){
        val geofence = geofenceHelper.getGeofence(GEOFENCE_ID, latLng, radius,
Geofence.GEOFENCE_TRANSITION_ENTER or
Geofence.GEOFENCE_TRANSITION_DWELL or Geofence.GEOFENCE_TRANSITION_EXIT)
// How do I pass multiple transitions?
        val geofencingRequest = geofenceHelper.getGeofencingRequest(geofence)
        Log.d(TAG, "Checkpoint 1")
        val pendingIntent = geofenceHelper.getPendingIntent()
        Log.d(TAG, "Checkpoint 4")

        geofencingClient.addGeofences(geofencingRequest, pendingIntent)
            .addOnSuccessListener {
                Log.d(TAG, "onSuccess: Geofence Added...")
            }
            .addOnFailureListener { e:Exception ->
                val errorMessage = geofenceHelper.getErrorString(e)
                Log.d(TAG, "Failed to make Geofence")
                Log.d(TAG, "onFailure$errorMessage")
            }
    }

    private fun addMarker(latLng: LatLng){
        map.addMarker(MarkerOptions().position(latLng).title("Marker"))
    }
    private fun addCircle(latLng: LatLng, radius: Double){
        circle = map.addCircle(
            CircleOptions()
                .center(latLng)
                .radius(radius)
                .strokeColor(Color.argb(255, 255, 0, 0))
                .fillColor(Color.argb(64, 255, 0, 0))
        )
    }
}
```

```
//GeofenceHelper Code

package com.example.maps

import android.app.PendingIntent
import android.content.Context
import android.content.ContextWrapper
import android.content.Intent
import android.util.Log
import com.google.android.gms.common.api.ApiException
import com.google.android.gms.location.Geofence
import com.google.android.gms.location.GeofenceStatusCodes
import com.google.android.gms.location.GeofencingRequest
import com.google.android.gms.maps.model.LatLng


class GeofenceHelper(base: Context?) : ContextWrapper(base) {

    private val TAG = "GeofenceHelper"
    //var pendingIntent: PendingIntent

    fun getGeofencingRequest(geofence: Geofence): GeofencingRequest {
        return GeofencingRequest.Builder()
            .addGeofence(geofence)
            .setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER)
            .build()
    }

    fun getGeofence(id: String, latLng: LatLng, radius: Float, transitionTypes: Int): Geofence {
        // Create Geofence
        return Geofence.Builder()
            .setCircularRegion(latLng.latitude, latLng.longitude, radius)
            .setRequestId(id)
            .setTransitionTypes(transitionTypes)
            .setLoiteringDelay(5000)
            .setExpirationDuration(Geofence.NEVER_EXPIRE)
            .build()
    }

    //@JvmName("getPendingIntent1")
    fun getPendingIntent(): PendingIntent {
        var pendingIntent: PendingIntent
//        if(pendingIntent!= null){
//            return pendingIntent
//        }
        val intent = Intent(this, GeofenceBroadcastReceiver::class.java)
        Log.d(TAG, "Checkpoint 2")
        pendingIntent = PendingIntent.getBroadcast(this, 2607, intent,
            PendingIntent.FLAG_MUTABLE or PendingIntent.FLAG_UPDATE_CURRENT)
        Log.d(TAG, "Checkpoint 3")
            return pendingIntent
    }

    fun getErrorString(e: Exception): String? {
        if (e is ApiException) {
            when (e.statusCode) {
                GeofenceStatusCodes.GEOFENCE_NOT_AVAILABLE -> return
"GEOFENCE_NOT_AVAILABLE"
                GeofenceStatusCodes.GEOFENCE_TOO_MANY_GEOFENCES -> return
"TOO_MANY_GEOFENCES"
                GeofenceStatusCodes.GEOFENCE_TOO_MANY_PENDING_INTENTS -> return
"TOO_MANY_PENDING_INTENTS"
            }
        }
        return e.localizedMessage
    }
}
```

```
//Geofence Broadcast Receiver Code
package com.example.maps

import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.util.Log
import android.widget.Toast
import com.google.android.gms.location.Geofence
import com.google.android.gms.location.GeofencingEvent

class GeofenceBroadcastReceiver : BroadcastReceiver() {

    private val TAG = "GeofenceBroadcastReceiver"

    override fun onReceive(context: Context, intent: Intent) {
        Toast.makeText(context, "Geofence triggered...", Toast.LENGTH_SHORT).show()

        val geofencingEvent = GeofencingEvent.fromIntent(intent)

        if(geofencingEvent.hasError()){
            Log.d(TAG, "onReceive: Error receiving geofence even...")
            return
        }
        var geofenceList = listOf<Geofence>()
        geofenceList = geofencingEvent.triggeringGeofences
        //val location = geofencingEvent.triggeringLocation
        val transitionType = geofencingEvent.geofenceTransition

        // More to be added
        when(transitionType){

        }
    }
}
```

Appendix 3: Device Main Program

```
#import libraries
from datetime import datetime #Date and time library
import time #time-related function library
import sys
import os
import board
import RPi.GPIO as GPIO
import adafruit_dht
from pyrebase import pyrebase
from pyfirmata import Arduino, util

#initialise firebase
config = {"apiKey": "AIzaSyDIx4wG2woFuKsk64nPpu7BnKE9CawnMDo",
  "authDomain": "smart-windows-app-edc8a.firebaseapp.com",
  "databaseURL": "https://smart-windows-app-edc8a-default-rtdb.firebaseio.com",
  "projectId": "smart-windows-app-edc8a",
  "storageBucket": "smart-windows-app-edc8a.appspot.com",
  "messagingSenderId": "1035393408535",
  "appId": "1:1035393408535:web:3a24ebae2282d1d7f40652",
  "measurementId": "G-CXZQXB2ZGV"}
firebase = pyrebase.initialize_app(config)
db = firebase.database() # Get a reference to the auth service

# Initial both temp sensors
tempSensorIn = adafruit_dht.DHT22(board.D21,use_pulseio=False) #inside temp sensor
tempSensorOut = adafruit_dht.DHT22(board.D12,use_pulseio=False) #outside temp sensor
currentRoomTemp, outsideTemp = None, None #initialise temp variables

# set up arduino
board = Arduino('/dev/ttyACM0')
pin = board.get_pin('a:0:i')
it = util.Iterator(board)
it.start()

#initialise servo motor
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
pwm=GPIO.PWM(17, 50)
pwm.start(0)
#initialise blinder level values
bValPrev = -1
bVal = 0

#######initialise lin actuator
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
#initialise window level values
wValPrev = -1
wVal = 0

def ultrasonicW(): # Window Ultrasonic function
    TRIG, ECHO = 17, 27
    GPIO.setup(TRIG, GPIO.OUT)
    GPIO.setup(ECHO, GPIO.IN)
    GPIO.output(TRIG, False)
    #time.sleep(0.2)
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)
    pulse_start = 0
    pulse_end = 0
    while GPIO.input(ECHO) == 0:
        pulse_start = time.time()
    while GPIO.input(ECHO) == 1:
        pulse_end = time.time()
    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150
    distance = round(distance, 2)
    #print("distance:", distance, "cm")
    return distance

def ultrasonicB():  # Blind Ultrasonic function
    TRIG, ECHO = 23, 24
    GPIO.setup(TRIG, GPIO.OUT)
    GPIO.setup(ECHO, GPIO.IN)
    GPIO.output(TRIG, False)
    #time.sleep(0.2)
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)
    pulse_start = 0
    pulse_end = 0
    while GPIO.input(ECHO) == 0:
        pulse_start = time.time()
    while GPIO.input(ECHO) == 1:
        pulse_end = time.time()
    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150
    distance = round(distance, 2)
    #print("distance:", distance, "cm")
    return distance

def lightSensor(): # light sensor measurement function
    # initialise light sensor
    import TSL2591
    lightSensor = TSL2591.TSL2591()
    T = 50 # threshold value

    if lightSensor.Lux > T:
        sunlight = True
        print('Lux: %d'%lightSensor.Lux)
        #print("Day time")
        lightSensor.TSL2591_SET_LuxInterrupt(50, 200)
    else:
        sunlight = False
        print('Lux: %d'%lightSensor.Lux)
        #print("Night time")
        lightSensor.TSL2591_SET_LuxInterrupt(50, 200)
    return sunlight


def Rain(): # Rain Sensor function
    analog_value = pin.read()
    while analog_value is None:
        analog_value = pin.read()
    T = 0.75 # Threshold value

    if analog_value > T:
        rain = False
    elif analog_value < T:
        rain = True

    return rain

def operateBlinders(valPrev, val): # operate Blinders Function
    distInterval = [74, 65.3, 58.6, 51.9, 45.2, 38.5, 31.8, 25.1, 18.4, 11.7, 5]
    dist = ultraSonicB()
    A, B = 6, 7 # Lin Actuator Pins on Arduino

    for i in range(11):
        if val == i and dist < (distInterval[i] - 2):
            board.digital[A].write(1)
            board.digital[B].write(0)
        elif val == i and dist > (distInterval[i] + 2):
            board.digital[A].write(0)
            board.digital[B].write(1)
        if val == i and dist > (distInterval[i] - 2) and dist < (distInterval[i] + 2):
            board.digital[A].write(1)
            board.digital[B].write(1)
            valPrev = val

    if valPrev == val:
        return val
    else:
        return -1 # Can't return previous value

def operateWindow(valPrev, val): # operate window function
    distInterval = [9.0, 13.28, 17.56, 21.84, 26.12, 30.4, 34.68, 38.96, 43.24, 47.52, 51.5]
    dist = ultraSonicW()
    A, B = 8, 9 # Lin Actuator Pins on Arduino

    for i in range(11):
        if val == i and dist < (distInterval[i] - 0.25):
            board.digital[A].write(0)
            board.digital[B].write(1)
        elif val == i and dist > (distInterval[i] + 0.25):
            board.digital[A].write(1)
            board.digital[B].write(0)
        if val == i and dist > (distInterval[i] - 0.25) and dist < (distInterval[i] + 0.25):
            board.digital[A].write(1)
            board.digital[B].write(1)
            valPrev = val

    if valPrev == val:
        return val
    else:
        return -1 # Can't return previous value

def Geofence():
    geofence = False
    cO = db.child("Settings").child("closeOption").get().val() #check if geofence is enabled

    if cO == False:
        geofence = True
    else:
        location = db.child("Geofence").get().val()
        for k, v in location.items(): #iterate through every user
            if v is True:
                geofence = True
    return geofence

def Temp(crt, ot): #retrieve temp values
    dRoomTemp = (db.child("Smart").child("temp").get()).val() # get desired room temp from app

    try:
        cRoomTemp = tempSensorIn.temperature # get temp reading from inside temp sensor
        outsideTemp = tempSensorOut.temperature # get temp reading from outside temp sensor
    except RuntimeError as error:
        cRoomTemp = crt
        outsideTemp = ot

    return dRoomTemp, cRoomTemp, outsideTemp

if __name__ == "__main__":
    while True:
        mode = db.child("SelectedMode").get().val() # get selected mode from app

        if mode == 1:
```

```
        print("Smart", mode)
        sunlight = lightSensor()
        desiredRoomTemp, currentRoomTemp, outsideTemp = Temp(currentRoomTemp,
outsideTemp)

        if (currentRoomTemp > (desiredRoomTemp + 2)): #room is hotter than desired
          print("room is hotter")
          if (outsideTemp > (desiredRoomTemp + 2)):
            print("outside is hotter")
            print("closing window and blinds")
            wVal = 10 #close windows
            bVal = 10 # close blinds
          elif (outsideTemp < (desiredRoomTemp + 2)):
            print("outside is colder/desired")
            print("opening window and blinds")

            wVal = db.child("Settings").child("autoWindowsLevel").get().val()
            bVal = db.child("Settings").child("autoBlindsLevel").get().val()
        elif (currentRoomTemp < (desiredRoomTemp + 2)): #room is colder or at desired
          wVal = 10 # close window to prevent heat from escaping home
          print("room is colder")
          if sunlight == True:
            print("opening blinds bc sun")
            bVal = db.child("Settings").child("autoBlindsLevel").get().val() # allow sunlight to
heat home
          else:
            print("closing blinds bc no sun")
            bVal = 10 # close blinds

        #Rain Override
        rain = Rain()
        print("rain: ", rain)
        if rain == True:
          print("closing window bc of rain")
          wVal = 10 # close windows

      if mode == 2:
        print("Auto", mode)

        # Window Open Time
        wOpenHour = db.child("Automatic").child("Windows").child("wOpenHour").get().val()
        wOpenMinute =
db.child("Automatic").child("Windows").child("wOpenMinute").get().val()
        print("Window Open Time: ", wOpenHour,":", wOpenMinute)
        wOpenTime = wOpenHour * 60 + wOpenMinute
        # Blinds Open Time
        bOpenHour = db.child("Automatic").child("Blinds").child("bOpenHour").get().val()
        bOpenMinute = db.child("Automatic").child("Blinds").child("bOpenMinute").get().val()
        print("Blinds Open Time: ", bOpenHour,":", bOpenMinute)
        bOpenTime = bOpenHour * 60 + bOpenMinute

        # Window Close Time
        wCloseHour = db.child("Automatic").child("Windows").child("wCloseHour").get().val()
        wCloseMinute =
db.child("Automatic").child("Windows").child("wCloseMinute").get().val()
      print("Window Close Time: ", wCloseHour,":", wCloseMinute)
      wCloseTime = wCloseHour * 60 + wCloseMinute
      # Blinds Close Time
      bCloseHour = db.child("Automatic").child("Blinds").child("bCloseHour").get().val()
      bCloseMinute = db.child("Automatic").child("Blinds").child("bCloseMinute").get().val()
      print("Window Open Time: ", bCloseHour,":", bCloseMinute)
      bCloseTime = bCloseHour * 60 + bCloseMinute

      #current
      cH = datetime.now().hour  # Current Hour as an int
      cM = datetime.now().minute
      print("Current Time: ", cH,":", cM)
      currentTime = cH * 60 + cM

      #Auto window Val
      if wOpenTime < wCloseTime:
        if wOpenTime <= currentTime < wCloseTime:
          print("Open Window")
          wVal = db.child("Settings").child("autoWindowsLevel").get().val()
        else:
          print("Close Window")
          wVal = 10
      else:
        if currentTime >= wOpenTime or currentTime <= wCloseTime:
          print("Open Window")
          wVal = db.child("Settings").child("autoWindowsLevel").get().val()
        else:
          print("Close Window")
          wVal = 10
      #Auto Blind Val
      if bOpenTime < bCloseTime:
        if bOpenTime <= currentTime < bCloseTime:
          print("Open Blinds")
          bVal = db.child("Settings").child("autoBlindsLevel").get().val()
        else:
          print("Close Blinds")
          bVal = 10
      else:
        if currentTime >= bOpenTime or currentTime <= bCloseTime:
          print("Open Blinds")
          bVal = db.child("Settings").child("autoBlindsLevel").get().val()
        else:
          print("Close Blinds")
          bVal = 10

    if mode == 3:
      print("Manual", mode)
      wVal = db.child("Manual").child("windowsVal").get().val()
      bVal = db.child("Manual").child("blindsVal").get().val()

    #Geofence override
    geofence = Geofence()
    if geofence == False:
            #print("closing window and blinds bc no one is home")
            wVal = 10
            bVal = 10

        if wVal != wValPrev: #adjust window if needed
            wValPrev = operateWindow(wValPrev, wVal)
            print("Previous Window Value: ", wValPrev)
            print("Current Window Value: ", wVal)
        if bVal != bValPrev: #adjust blinds if needed
            bValPrev = operateBlinders(bValPrev, bVal)
            print("Previous Blinders Value: ", bValPrev)
            print("Current Blinders Value: ", bVal)
```

Appendix 4: Firmata Code

```

  Firmata is a generic protocol for communicating with microcontrollers
  from software on a host computer. It is intended to work with
  any host computer software package.

  To download a host software package, please click on the following link
  to open the list of Firmata client libraries in your default browser.

  https://github.com/firmata/arduino#firmata-client-libraries

  Copyright (C) 2006-2008 Hans-Christoph Steiner.  All rights reserved.
  Copyright (C) 2010-2011 Paul Stoffregen.  All rights reserved.
  Copyright (C) 2009 Shigeru Kobayashi.  All rights reserved.
  Copyright (C) 2009-2016 Jeff Hoefs.  All rights reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  Licence as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  See file LICENSE.txt for further informations on licensing terms.

  Last updated August 17th, 2017
*/

#include <Servo.h>
#include <Wire.h>
#include <Firmata.h>

#define I2C_WRITE                   B00000000
#define I2C_READ                    B00001000
#define I2C_READ_CONTINUOUSLY       B00010000
#define I2C_STOP_READING            B00011000
#define I2C_READ_WRITE_MODE_MASK    B00011000
#define I2C_10BIT_ADDRESS_MODE_MASK B00100000
#define I2C_END_TX_MASK             B01000000
#define I2C_STOP_TX                 1
```

```
#define I2C_RESTART_TX             0
#define I2C_MAX_QUERIES            8
#define I2C_REGISTER_NOT_SPECIFIED  -1

// the minimum interval for sampling analog input
#define MINIMUM_SAMPLING_INTERVAL   1


/*==============================================================================
=========
 * GLOBAL VARIABLES

*==============================================================================
======*/

#ifdef FIRMATA_SERIAL_FEATURE
SerialFirmata serialFeature;
#endif

/* analog inputs */
int analogInputsToReport = 0; // bitwise array to store pin reporting

/* digital input ports */
byte reportPINs[TOTAL_PORTS];       // 1 = report this port, 0 = silence
byte previousPINs[TOTAL_PORTS];     // previous 8 bits sent

/* pins configuration */
byte portConfigInputs[TOTAL_PORTS]; // each bit: 1 = pin in INPUT, 0 = anything else

/* timer variables */
unsigned long currentMillis;        // store the current value from millis()
unsigned long previousMillis;       // for comparison with currentMillis
unsigned int samplingInterval = 19; // how often to run the main loop (in ms)

/* i2c data */
struct i2c_device_info {
  byte addr;
  int reg;
  byte bytes;
  byte stopTX;
};

/* for i2c read continuous more */
i2c_device_info query[I2C_MAX_QUERIES];

byte i2cRxData[64];
boolean isI2CEnabled = false;
signed char queryIndex = -1;
// default delay time between i2c read request and Wire.requestFrom()
unsigned int i2cReadDelayTime = 0;

Servo servos[MAX_SERVOS];
byte servoPinMap[TOTAL_PINS];
byte detachedServos[MAX_SERVOS];
byte detachedServoCount = 0;
byte servoCount = 0;

boolean isResetting = false;

// Forward declare a few functions to avoid compiler errors with older versions
// of the Arduino IDE.
void setPinModeCallback(byte, int);
void reportAnalogCallback(byte analogPin, int value);
void sysexCallback(byte, byte, byte*);

/* utility functions */
void wireWrite(byte data)
{
#if ARDUINO >= 100
  Wire.write((byte)data);
#else
  Wire.send(data);
#endif
}

byte wireRead(void)
{
#if ARDUINO >= 100
  return Wire.read();
#else
  return Wire.receive();
#endif
}

/*==============================================================================
=========
 * FUNCTIONS

*=============================================================================
======*/

void attachServo(byte pin, int minPulse, int maxPulse)
{
  if (servoCount < MAX_SERVOS) {
    // reuse indexes of detached servos until all have been reallocated
    if (detachedServoCount > 0) {
      servoPinMap[pin] = detachedServos[detachedServoCount - 1];
      if (detachedServoCount > 0) detachedServoCount--;
    } else {
      servoPinMap[pin] = servoCount;
      servoCount++;
    }
    if (minPulse > 0 && maxPulse > 0) {
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin), minPulse, maxPulse);
    } else {
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin));
    }
  } else {
    Firmata.sendString("Max servos attached");
  }
}

void detachServo(byte pin)
{
  servos[servoPinMap[pin]].detach();
  // if we're detaching the last servo, decrement the count
  // otherwise store the index of the detached servo
  if (servoPinMap[pin] == servoCount && servoCount > 0) {
    servoCount--;
  } else if (servoCount > 0) {
    // keep track of detached servos because we want to reuse their indexes
    // before incrementing the count of attached servos
    detachedServoCount++;
    detachedServos[detachedServoCount - 1] = servoPinMap[pin];
  }

  servoPinMap[pin] = 255;
}

void enableI2CPins()
{
  byte i;
  // is there a faster way to do this? would probably require importing
  // Arduino.h to get SCL and SDA pins
  for (i = 0; i < TOTAL_PINS; i++) {
    if (IS_PIN_I2C(i)) {
      // mark pins as i2c so they are ignored in non i2c data requests
      setPinModeCallback(i, PIN_MODE_I2C);
    }
  }

  isI2CEnabled = true;

  Wire.begin();
}

/* disable the i2c pins so they can be used for other functions */
void disableI2CPins() {
  isI2CEnabled = false;
  // disable read continuous mode for all devices
  queryIndex = -1;
}

void readAndReportData(byte address, int theRegister, byte numBytes, byte stopTX) {
  // allow I2C requests that don't require a register read
  // for example, some devices using an interrupt pin to signify new data available
  // do not always require the register read so upon interrupt you call Wire.requestFrom()
  if (theRegister != I2C_REGISTER_NOT_SPECIFIED) {
    Wire.beginTransmission(address);
    wireWrite((byte)theRegister);
    Wire.endTransmission(stopTX); // default = true
    // do not set a value of 0
    if (i2cReadDelayTime > 0) {
      // delay is necessary for some devices such as WiiNunchuck
      delayMicroseconds(i2cReadDelayTime);
    }
  } else {
    theRegister = 0;  // fill the register with a dummy value
  }

  Wire.requestFrom(address, numBytes);  // all bytes are returned in requestFrom

  // check to be sure correct number of bytes were returned by slave
  if (numBytes < Wire.available()) {
    Firmata.sendString("I2C: Too many bytes received");
  } else if (numBytes > Wire.available()) {
    Firmata.sendString("I2C: Too few bytes received");
  }

  i2cRxData[0] = address;
  i2cRxData[1] = theRegister;

  for (int i = 0; i < numBytes && Wire.available(); i++) {
    i2cRxData[2 + i] = wireRead();
  }

  // send slave address, register and received bytes
  Firmata.sendSysex(SYSEX_I2C_REPLY, numBytes + 2, i2cRxData);
}

void outputPort(byte portNumber, byte portValue, byte forceSend)
{
  // pins not configured as INPUT are cleared to zeros
  portValue = portValue & portConfigInputs[portNumber];
  // only send if the value is different than previously sent
  if (forceSend || previousPINs[portNumber] != portValue) {
    Firmata.sendDigitalPort(portNumber, portValue);
    previousPINs[portNumber] = portValue;
  }
}

/* -----------------------------------------------------------------------------
 * check all the active digital inputs for change of state, then add any events
 * to the Serial output queue using Serial.print() */
void checkDigitalInputs(void)
{
  /* Using non-looping code allows constants to be given to readPort().
   * The compiler will apply substantial optimizations if the inputs
   * to readPort() are compile-time constants. */
  if (TOTAL_PORTS > 0 && reportPINs[0]) outputPort(0, readPort(0, portConfigInputs[0]), false);
  if (TOTAL_PORTS > 1 && reportPINs[1]) outputPort(1, readPort(1, portConfigInputs[1]), false);
  if (TOTAL_PORTS > 2 && reportPINs[2]) outputPort(2, readPort(2, portConfigInputs[2]), false);
  if (TOTAL_PORTS > 3 && reportPINs[3]) outputPort(3, readPort(3, portConfigInputs[3]), false);
  if (TOTAL_PORTS > 4 && reportPINs[4]) outputPort(4, readPort(4, portConfigInputs[4]), false);
  if (TOTAL_PORTS > 5 && reportPINs[5]) outputPort(5, readPort(5, portConfigInputs[5]), false);
  if (TOTAL_PORTS > 6 && reportPINs[6]) outputPort(6, readPort(6, portConfigInputs[6]), false);
  if (TOTAL_PORTS > 7 && reportPINs[7]) outputPort(7, readPort(7, portConfigInputs[7]), false);
  if (TOTAL_PORTS > 8 && reportPINs[8]) outputPort(8, readPort(8, portConfigInputs[8]), false);
  if (TOTAL_PORTS > 9 && reportPINs[9]) outputPort(9, readPort(9, portConfigInputs[9]), false);
  if (TOTAL_PORTS > 10 && reportPINs[10]) outputPort(10, readPort(10, portConfigInputs[10]),
false);
  if (TOTAL_PORTS > 11 && reportPINs[11]) outputPort(11, readPort(11, portConfigInputs[11]),
false);
  if (TOTAL_PORTS > 12 && reportPINs[12]) outputPort(12, readPort(12, portConfigInputs[12]),
false);
  if (TOTAL_PORTS > 13 && reportPINs[13]) outputPort(13, readPort(13, portConfigInputs[13]),
false);
  if (TOTAL_PORTS > 14 && reportPINs[14]) outputPort(14, readPort(14, portConfigInputs[14]),
false);
  if (TOTAL_PORTS > 15 && reportPINs[15]) outputPort(15, readPort(15, portConfigInputs[15]),
false);
}

// -----------------------------------------------------------------------------
/* sets the pin mode to the correct state and sets the relevant bits in the
 * two bit-arrays that track Digital I/O and PWM status
 */
void setPinModeCallback(byte pin, int mode)
{
  if (Firmata.getPinMode(pin) == PIN_MODE_IGNORE)
    return;

  if (Firmata.getPinMode(pin) == PIN_MODE_I2C && isI2CEnabled && mode !=
PIN_MODE_I2C) {
    // disable i2c so pins can be used for other functions
    // the following if statements should reconfigure the pins properly
    disableI2CPins();
  }
  if (IS_PIN_DIGITAL(pin) && mode != PIN_MODE_SERVO) {
    if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
      detachServo(pin);
    }
  }
  if (IS_PIN_ANALOG(pin)) {
    reportAnalogCallback(PIN_TO_ANALOG(pin), mode == PIN_MODE_ANALOG ? 1 : 0); //
turn on/off reporting
  }
  if (IS_PIN_DIGITAL(pin)) {
    if (mode == INPUT || mode == PIN_MODE_PULLUP) {
      portConfigInputs[pin / 8] |= (1 << (pin & 7));
    } else {
      portConfigInputs[pin / 8] &= ~(1 << (pin & 7));
    }
  }
  Firmata.setPinState(pin, 0);
  switch (mode) {
    case PIN_MODE_ANALOG:
      if (IS_PIN_ANALOG(pin)) {
        if (IS_PIN_DIGITAL(pin)) {
          pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
#if ARDUINO <= 100
          // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
          digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
#endif
        }
        Firmata.setPinMode(pin, PIN_MODE_ANALOG);
      }
      break;
    case INPUT:
      if (IS_PIN_DIGITAL(pin)) {
        pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
#if ARDUINO <= 100
        // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
        digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
#endif
        Firmata.setPinMode(pin, INPUT);
      }
      break;
    case PIN_MODE_PULLUP:
      if (IS_PIN_DIGITAL(pin)) {
        pinMode(PIN_TO_DIGITAL(pin), INPUT_PULLUP);
        Firmata.setPinMode(pin, PIN_MODE_PULLUP);
        Firmata.setPinState(pin, 1);
      }
      break;
    case OUTPUT:
      if (IS_PIN_DIGITAL(pin)) {
        if (Firmata.getPinMode(pin) == PIN_MODE_PWM) {
          // Disable PWM if pin mode was previously set to PWM.
          digitalWrite(PIN_TO_DIGITAL(pin), LOW);
        }
        pinMode(PIN_TO_DIGITAL(pin), OUTPUT);
        Firmata.setPinMode(pin, OUTPUT);
      }
      break;
    case PIN_MODE_PWM:
      if (IS_PIN_PWM(pin)) {
        pinMode(PIN_TO_PWM(pin), OUTPUT);
        analogWrite(PIN_TO_PWM(pin), 0);
        Firmata.setPinMode(pin, PIN_MODE_PWM);
      }
      break;
    case PIN_MODE_SERVO:
      if (IS_PIN_DIGITAL(pin)) {
        Firmata.setPinMode(pin, PIN_MODE_SERVO);
        if (servoPinMap[pin] == 255 || !servos[servoPinMap[pin]].attached()) {
          // pass -1 for min and max pulse values to use default values set
          // by Servo library
          attachServo(pin, -1, -1);
        }
      }
      break;
    case PIN_MODE_I2C:
      if (IS_PIN_I2C(pin)) {
        // mark the pin as i2c
        // the user must call I2C_CONFIG to enable I2C for a device
        Firmata.setPinMode(pin, PIN_MODE_I2C);
      }
      break;
    case PIN_MODE_SERIAL:
#ifdef FIRMATA_SERIAL_FEATURE
      serialFeature.handlePinMode(pin, PIN_MODE_SERIAL);
#endif
      break;
    default:
      Firmata.sendString("Unknown pin mode"); // TODO: put error msgs in EEPROM
  }
  // TODO: save status to EEPROM here, if changed
}

/*
 * Sets the value of an individual pin. Useful if you want to set a pin value but
 * are not tracking the digital port state.
 * Can only be used on pins configured as OUTPUT.
 * Cannot be used to enable pull-ups on Digital INPUT pins.
 */
void setPinValueCallback(byte pin, int value)
{
  if (pin < TOTAL_PINS && IS_PIN_DIGITAL(pin)) {
    if (Firmata.getPinMode(pin) == OUTPUT) {
      Firmata.setPinState(pin, value);
      digitalWrite(PIN_TO_DIGITAL(pin), value);
    }
  }
}

void analogWriteCallback(byte pin, int value)
{
  if (pin < TOTAL_PINS) {
    switch (Firmata.getPinMode(pin)) {
      case PIN_MODE_SERVO:
        if (IS_PIN_DIGITAL(pin))
          servos[servoPinMap[pin]].write(value);
        Firmata.setPinState(pin, value);
        break;
      case PIN_MODE_PWM:
        if (IS_PIN_PWM(pin))
          analogWrite(PIN_TO_PWM(pin), value);
        Firmata.setPinState(pin, value);
        break;
    }
  }
}

void digitalWriteCallback(byte port, int value)
{
  byte pin, lastPin, pinValue, mask = 1, pinWriteMask = 0;
  if (port < TOTAL_PORTS) {
    // create a mask of the pins on this port that are writable.
    lastPin = port * 8 + 8;
    if (lastPin > TOTAL_PINS) lastPin = TOTAL_PINS;
    for (pin = port * 8; pin < lastPin; pin++) {
      // do not disturb non-digital pins (eg, Rx & Tx)
      if (IS_PIN_DIGITAL(pin)) {
        // do not touch pins in PWM, ANALOG, SERVO or other modes
        if (Firmata.getPinMode(pin) == OUTPUT || Firmata.getPinMode(pin) == INPUT) {
          pinValue = ((byte)value & mask) ? 1 : 0;
          if (Firmata.getPinMode(pin) == OUTPUT) {
            pinWriteMask |= mask;
          } else if (Firmata.getPinMode(pin) == INPUT && pinValue == 1 &&
Firmata.getPinState(pin) != 1) {
            // only handle INPUT here for backwards compatibility
#if ARDUINO > 100
            pinMode(pin, INPUT_PULLUP);
#else
            // only write to the INPUT pin to enable pullups if Arduino v1.0.0 or earlier
            pinWriteMask |= mask;
#endif
          }
          Firmata.setPinState(pin, pinValue);
        }
      }
      mask = mask << 1;
    }
    writePort(port, (byte)value, pinWriteMask);
  }
}


// -----------------------------------------------------------------------------
/* sets bits in a bit array (int) to toggle the reporting of the analogIns
 */
//void FirmataClass::setAnalogPinReporting(byte pin, byte state) {
//}
void reportAnalogCallback(byte analogPin, int value)
{
  if (analogPin < TOTAL_ANALOG_PINS) {
    if (value == 0) {
      analogInputsToReport = analogInputsToReport & ~ (1 << analogPin);
    } else {
      analogInputsToReport = analogInputsToReport | (1 << analogPin);
      // prevent during system reset or all analog pin values will be reported
      // which may report noise for unconnected analog pins
      if (!isResetting) {
        // Send pin value immediately. This is helpful when connected via
        // ethernet, wi-fi or bluetooth so pin states can be known upon
        // reconnecting.
        Firmata.sendAnalog(analogPin, analogRead(analogPin));
      }
    }
  }
  // TODO: save status to EEPROM here, if changed
}

void reportDigitalCallback(byte port, int value)
{
  if (port < TOTAL_PORTS) {
    reportPINs[port] = (byte)value;
    // Send port value immediately. This is helpful when connected via
    // ethernet, wi-fi or bluetooth so pin states can be known upon
    // reconnecting.
    if (value) outputPort(port, readPort(port, portConfigInputs[port]), true);
  }
  // do not disable analog reporting on these 8 pins, to allow some
  // pins used for digital, others analog.  Instead, allow both types
  // of reporting to be enabled, but check if the pin is configured
  // as analog when sampling the analog inputs.  Likewise, while
  // scanning digital pins, portConfigInputs will mask off values from any
  // pins configured as analog
}

/*==============================================================
=========
 * SYSEX-BASED commands

*==============================================================
======*/

void sysexCallback(byte command, byte argc, byte *argv)
{
  byte mode;
  byte stopTX;
  byte slaveAddress;
  byte data;
  int slaveRegister;
  unsigned int delayTime;

  switch (command) {
    case I2C_REQUEST:
      mode = argv[1] & I2C_READ_WRITE_MODE_MASK;
      if (argv[1] & I2C_10BIT_ADDRESS_MODE_MASK) {
        Firmata.sendString("10-bit addressing not supported");
        return;
      }
      else {
        slaveAddress = argv[0];
      }

      // need to invert the logic here since 0 will be default for client
      // libraries that have not updated to add support for restart tx
      if (argv[1] & I2C_END_TX_MASK) {
        stopTX = I2C_RESTART_TX;
      }
      else {
        stopTX = I2C_STOP_TX; // default
    }

    switch (mode) {
      case I2C_WRITE:
        Wire.beginTransmission(slaveAddress);
        for (byte i = 2; i < argc; i += 2) {
          data = argv[i] + (argv[i + 1] << 7);
          wireWrite(data);
        }
        Wire.endTransmission();
        delayMicroseconds(70);
        break;
      case I2C_READ:
        if (argc == 6) {
          // a slave register is specified
          slaveRegister = argv[2] + (argv[3] << 7);
          data = argv[4] + (argv[5] << 7);  // bytes to read
        }
        else {
          // a slave register is NOT specified
          slaveRegister = I2C_REGISTER_NOT_SPECIFIED;
          data = argv[2] + (argv[3] << 7);  // bytes to read
        }
        readAndReportData(slaveAddress, (int)slaveRegister, data, stopTX);
        break;
      case I2C_READ_CONTINUOUSLY:
        if ((queryIndex + 1) >= I2C_MAX_QUERIES) {
          // too many queries, just ignore
          Firmata.sendString("too many queries");
          break;
        }
        if (argc == 6) {
          // a slave register is specified
          slaveRegister = argv[2] + (argv[3] << 7);
          data = argv[4] + (argv[5] << 7);  // bytes to read
        }
        else {
          // a slave register is NOT specified
          slaveRegister = (int)I2C_REGISTER_NOT_SPECIFIED;
          data = argv[2] + (argv[3] << 7);  // bytes to read
        }
        queryIndex++;
        query[queryIndex].addr = slaveAddress;
        query[queryIndex].reg = slaveRegister;
        query[queryIndex].bytes = data;
        query[queryIndex].stopTX = stopTX;
        break;
      case I2C_STOP_READING:
        byte queryIndexToSkip;
        // if read continuous mode is enabled for only 1 i2c device, disable
        // read continuous reporting for that device
        if (queryIndex <= 0) {
          queryIndex = -1;
        } else {
          queryIndexToSkip = 0;
          // if read continuous mode is enabled for multiple devices,
          // determine which device to stop reading and remove its data from
          // the array, shifting other array data to fill the space
          for (byte i = 0; i < queryIndex + 1; i++) {
            if (query[i].addr == slaveAddress) {
              queryIndexToSkip = i;
              break;
            }
          }

          for (byte i = queryIndexToSkip; i < queryIndex + 1; i++) {
            if (i < I2C_MAX_QUERIES) {
              query[i].addr = query[i + 1].addr;
              query[i].reg = query[i + 1].reg;
              query[i].bytes = query[i + 1].bytes;
              query[i].stopTX = query[i + 1].stopTX;
            }
          }
          queryIndex--;
        }
        break;
      default:
        break;
    }
    break;
  case I2C_CONFIG:
    delayTime = (argv[0] + (argv[1] << 7));

    if (argc > 1 && delayTime > 0) {
      i2cReadDelayTime = delayTime;
    }

    if (!isI2CEnabled) {
      enableI2CPins();
    }

    break;
  case SERVO_CONFIG:
    if (argc > 4) {
      // these vars are here for clarity, they'll be optimized away by the compiler
      byte pin = argv[0];
      int minPulse = argv[1] + (argv[2] << 7);
      int maxPulse = argv[3] + (argv[4] << 7);

      if (IS_PIN_DIGITAL(pin)) {
        if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
          detachServo(pin);
        }
        attachServo(pin, minPulse, maxPulse);
        setPinModeCallback(pin, PIN_MODE_SERVO);
      }
    }
    break;
```

```
   case SAMPLING_INTERVAL:
     if (argc > 1) {
       samplingInterval = argv[0] + (argv[1] << 7);
       if (samplingInterval < MINIMUM_SAMPLING_INTERVAL) {
         samplingInterval = MINIMUM_SAMPLING_INTERVAL;
       }
     } else {
       //Firmata.sendString("Not enough data");
     }
     break;
   case EXTENDED_ANALOG:
     if (argc > 1) {
       int val = argv[1];
       if (argc > 2) val |= (argv[2] << 7);
       if (argc > 3) val |= (argv[3] << 14);
       analogWriteCallback(argv[0], val);
     }
     break;
   case CAPABILITY_QUERY:
     Firmata.write(START_SYSEX);
     Firmata.write(CAPABILITY_RESPONSE);
     for (byte pin = 0; pin < TOTAL_PINS; pin++) {
       if (IS_PIN_DIGITAL(pin)) {
         Firmata.write((byte)INPUT);
         Firmata.write(1);
         Firmata.write((byte)PIN_MODE_PULLUP);
         Firmata.write(1);
         Firmata.write((byte)OUTPUT);
         Firmata.write(1);
       }
       if (IS_PIN_ANALOG(pin)) {
         Firmata.write(PIN_MODE_ANALOG);
         Firmata.write(10); // 10 = 10-bit resolution
       }
       if (IS_PIN_PWM(pin)) {
         Firmata.write(PIN_MODE_PWM);
         Firmata.write(DEFAULT_PWM_RESOLUTION);
       }
       if (IS_PIN_DIGITAL(pin)) {
         Firmata.write(PIN_MODE_SERVO);
         Firmata.write(14);
       }
       if (IS_PIN_I2C(pin)) {
         Firmata.write(PIN_MODE_I2C);
         Firmata.write(1);  // TODO: could assign a number to map to SCL or SDA
       }
#ifdef FIRMATA_SERIAL_FEATURE
       serialFeature.handleCapability(pin);
#endif
       Firmata.write(127);
     }
     Firmata.write(END_SYSEX);
     break;
   case PIN_STATE_QUERY:
     if (argc > 0) {
       byte pin = argv[0];
       Firmata.write(START_SYSEX);
       Firmata.write(PIN_STATE_RESPONSE);
       Firmata.write(pin);
       if (pin < TOTAL_PINS) {
         Firmata.write(Firmata.getPinMode(pin));
         Firmata.write((byte)Firmata.getPinState(pin) & 0x7F);
         if (Firmata.getPinState(pin) & 0xFF80) Firmata.write((byte)(Firmata.getPinState(pin) >> 7)
& 0x7F);
         if (Firmata.getPinState(pin) & 0xC000) Firmata.write((byte)(Firmata.getPinState(pin) >> 14)
& 0x7F);
       }
       Firmata.write(END_SYSEX);
     }
     break;
   case ANALOG_MAPPING_QUERY:
     Firmata.write(START_SYSEX);
     Firmata.write(ANALOG_MAPPING_RESPONSE);
     for (byte pin = 0; pin < TOTAL_PINS; pin++) {
       Firmata.write(IS_PIN_ANALOG(pin) ? PIN_TO_ANALOG(pin) : 127);
     }
     Firmata.write(END_SYSEX);
     break;

   case SERIAL_MESSAGE:
#ifdef FIRMATA_SERIAL_FEATURE
     serialFeature.handleSysex(command, argc, argv);
#endif
     break;
 }
}

/*==============================================================================
=========
 * SETUP()

*==============================================================================
======*/

void systemResetCallback()
{
 isResetting = true;

 // initialize a default state
 // TODO: option to load config from EEPROM instead of default

#ifdef FIRMATA_SERIAL_FEATURE
 serialFeature.reset();
#endif

 if (isI2CEnabled) {
   disableI2CPins();
 }
 for (byte i = 0; i < TOTAL_PORTS; i++) {
   reportPINs[i] = false;    // by default, reporting off
   portConfigInputs[i] = 0;  // until activated
   previousPINs[i] = 0;
 }

 for (byte i = 0; i < TOTAL_PINS; i++) {
   // pins with analog capability default to analog input
   // otherwise, pins default to digital output
   if (IS_PIN_ANALOG(i)) {
     // turns off pullup, configures everything
     setPinModeCallback(i, PIN_MODE_ANALOG);
   } else if (IS_PIN_DIGITAL(i)) {
     // sets the output to 0, configures portConfigInputs
     setPinModeCallback(i, OUTPUT);
   }

   servoPinMap[i] = 255;
 }
 // by default, do not report any analog inputs
 analogInputsToReport = 0;

 detachedServoCount = 0;
 servoCount = 0;

 /* send digital inputs to set the initial state on the host computer,
  * since once in the loop(), this firmware will only send on change */
 /*
 TODO: this can never execute, since no pins default to digital input
       but it will be needed when/if we support EEPROM stored config
 for (byte i=0; i < TOTAL_PORTS; i++) {
   outputPort(i, readPort(i, portConfigInputs[i]), true);
 }
 */
 isResetting = false;
}

void setup()
{
 Firmata.setFirmwareVersion(FIRMATA_FIRMWARE_MAJOR_VERSION, 
FIRMATA_FIRMWARE_MINOR_VERSION);

 Firmata.attach(ANALOG_MESSAGE, analogWriteCallback);
 Firmata.attach(DIGITAL_MESSAGE, digitalWriteCallback);
 Firmata.attach(REPORT_ANALOG, reportAnalogCallback);
 Firmata.attach(REPORT_DIGITAL, reportDigitalCallback);
 Firmata.attach(SET_PIN_MODE, setPinModeCallback);
 Firmata.attach(SET_DIGITAL_PIN_VALUE, setPinValueCallback);
 Firmata.attach(START_SYSEX, sysexCallback);
 Firmata.attach(SYSTEM_RESET, systemResetCallback);

 // to use a port other than Serial, such as Serial1 on an Arduino Leonardo or Mega,
 // Call begin(baud) on the alternate serial port and pass it to Firmata to begin like this:
```

```
  // Serial1.begin(57600);
  // Firmata.begin(Serial1);
  // However do not do this if you are using SERIAL_MESSAGE

  Firmata.begin(57600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for ATmega32u4-based boards and Arduino 101
  }

  systemResetCallback();  // reset to default config
}

/*==============================================================================
=========
 * LOOP()

*==============================================================================
======*/
void loop()
{
  byte pin, analogPin;

  /* DIGITALREAD - as fast as possible, check for changes and output them to the
   * FTDI buffer using Serial.print()  */
  checkDigitalInputs();

  /* STREAMREAD - processing incoming messagse as soon as possible, while still
   * checking digital inputs.  */
  while (Firmata.available())
    Firmata.processInput();

  // TODO - ensure that Stream buffer doesn't go over 60 bytes

  currentMillis = millis();
  if (currentMillis - previousMillis > samplingInterval) {
    previousMillis += samplingInterval;
    /* ANALOGREAD - do all analogReads() at the configured sampling interval */
    for (pin = 0; pin < TOTAL_PINS; pin++) {
      if (IS_PIN_ANALOG(pin) && Firmata.getPinMode(pin) == PIN_MODE_ANALOG) {
        analogPin = PIN_TO_ANALOG(pin);
        if (analogInputsToReport & (1 << analogPin)) {
          Firmata.sendAnalog(analogPin, analogRead(analogPin));
        }
      }
    }
    // report i2c data for all device with read continuous mode enabled
    if (queryIndex > -1) {
      for (byte i = 0; i < queryIndex + 1; i++) {
        readAndReportData(query[i].addr, query[i].reg, query[i].bytes, query[i].stopTX);
      }
    }
  }

#ifdef FIRMATA_SERIAL_FEATURE
```

```
  serialFeature.update();
#endif
}
```