SCIENTIFIC Publishing Limited

**Digital Technologies Research and Applications**

https://ojs.ukscip.com/journals/dtra

*Article*

# Raspberry-PI based Design of An Interactive Smart Mirror for Daily Life

**Joe Reginald Lyons, Ogbonnaya Anicho and Emanuele Lindo Secco** *

School of Mathematics, Computer Science and Engineering, Liverpool Hope University, Hope Park, L16 9JD, UK
* Correspondence: seccoe@hope.ac.uk; Tel.: +44 (0)151 291 3641

**Received**: 22 April 2024; **Revised:** 20 May 2024; **Accepted**: 24 May 2024; **Published**: 12 June 2024

**Abstract:** The Internet of Things and spatial computing are increasingly popular in today's technological environment. These devices can sometimes produce counterproductive effects, complicating the interaction between non-expert end users and the device itself. In this paper, we propose a simple, user-friendly, and cost-effective configurable smart mirror, which can display usefully relevant real-time information. This system is designed based on a low-cost Raspberry Pi paired with an LCD screen. The system can connect with a personal computer (PC) through the IEEE 802.15 wireless communication protocol. The preliminary results in this paper show the intuitive usability of the device in daily life.

**Keywords:** Internet of Things; low-cost interactive design; intuitive design; user-friendly design

## 1. Introduction

The smart home industry is continuously developing [1]. Smart mirrors, one kind of home technology, can display relevant useful information and provide applications in health and energy efficiency [2].

At present, the market for this device is limited, and it is produced mainly by hobbyists–almost impossible to acquire one [3]. Therefore, we can infer that there is a lack of customisable software, which is the gap we aim to fill in this project. Like smart mirrors, smart displays visualise relevant information. Having Reached $3.78 billion in 2020, the global smart display market continues to increase, signalling a growing demand for smart home technologies [4]. This growth in popularity is attributed to increased functionality and configurability, with voice assistants now integrated into the technology, further improving functionality.

Homes are customised according to user needs and differ greatly in requirements. Therefore, smart devices must be configurable to meet this range of requirements. Regardless of the growing industry, these devices have not been adopted, this is considered due to a lack of configurability and cost. Some smart mirrors do provide reconfigurability, however, these devices are on average more than double the cost of similar smart home technologies this price is not justified, an idea explored within this paper. In this context, the main contributions of this paper are:

- A smart mirror design that combines the advantages of low cost, configurability, and energy efficiency;
- An overview of the deficits in the smart mirror industry;
- Improvements that can be made to current smart mirror technologies.

We want to design a novel smart mirror with the following objectives: the device will be used frequently, retained for a long time and be a part of the user's home. Therefore, it is recommended that the design should be robust, aesthetically pleasing, and functional.

The proposed device must also maintain user privacy, a growing concern in the home technology industry [5]. In order to achieve these goals, the development will be user-centred, based on user feedback and testing, namely:

- User data remains secure.
- The application and device display data graphically.
- Up-to-date relevant information is displayed.
- The user can configure the location and content of displayed data.
- The interface is easy to understand and use.
- There is a range of gadget options.
- The device can be connected to the internet wirelessly.
- The device can be connected to the device via Bluetooth.

The paper is organised as follows: Section 2 presents the hardware and software of the system, Section 3 explores system testing and corrective maintenance, and Section 4 contains the evaluation and conclusion of the system.

Two applications are developed using Python 3.12.2, one for configuration, running on a Windows machine and the other running the mirror's interface on a Raspberry Pi 3B. Python is a high-performance, portable language that supports rapid development, which makes development easier, and reduces developing time and multiple iterations of the device being made across a range of operating systems [6].

## 2. Materials and Methods

The device will need to access the internet for API requests, and a Wi-Fi network is ideal for this. However, the mirror cannot connect to a Wi-Fi network due to a lack of credentials and input. Instead, the application will communicate via IEEE 802.15 protocol, namely Bluetooth PAN initially, and receive the Wi-Fi network credentials later via this protocol.

A set of algorithms needs to be designed: precisely, two differing flowcharts detail the execution path of the applications, which helps to develop the application, decrease development time, and ensure the realisation of project aims by aiding algorithm comprehension [7]. An overview of these algorithms is reported in Figures 1 and 2.

### 2.1. Software

The applications use various libraries to improve software functionality and help to meet the success criteria. Libraries required and the justification of each:

- The Requests library is required to retrieve the API information displayed by the application.
- CustomTkinter is used to create the graphical user interface (UI). It adds methods and classes that can be used to create custom graphical interfaces.
- Pillow is an image library for loading and formatting images so they can be displayed graphically.
- The subprocess library allows terminal commands to be issued from the program, which is necessary to fetch the nearby network credentials sent to the Pi and used for connection.
- The Threading library offers support for simultaneous multi-threading; this can reduce program execution time by allowing multiple processes requiring CPU attention to be executed simultaneously, this is used for updating the user interface [8].
- The socket library is used to connect and send data via Bluetooth from and to the mirror.
- Datetime is required to retrieve the system time and date, which will be parsed and update the widget as needed.
- CTkListBox is a small library built on “CustomTkinter”, which adds functionality for another type of graphical widget.
- CTkMessageBox is another small library built on “CustomTkinter”, which adds functionality for pop-up message windows.

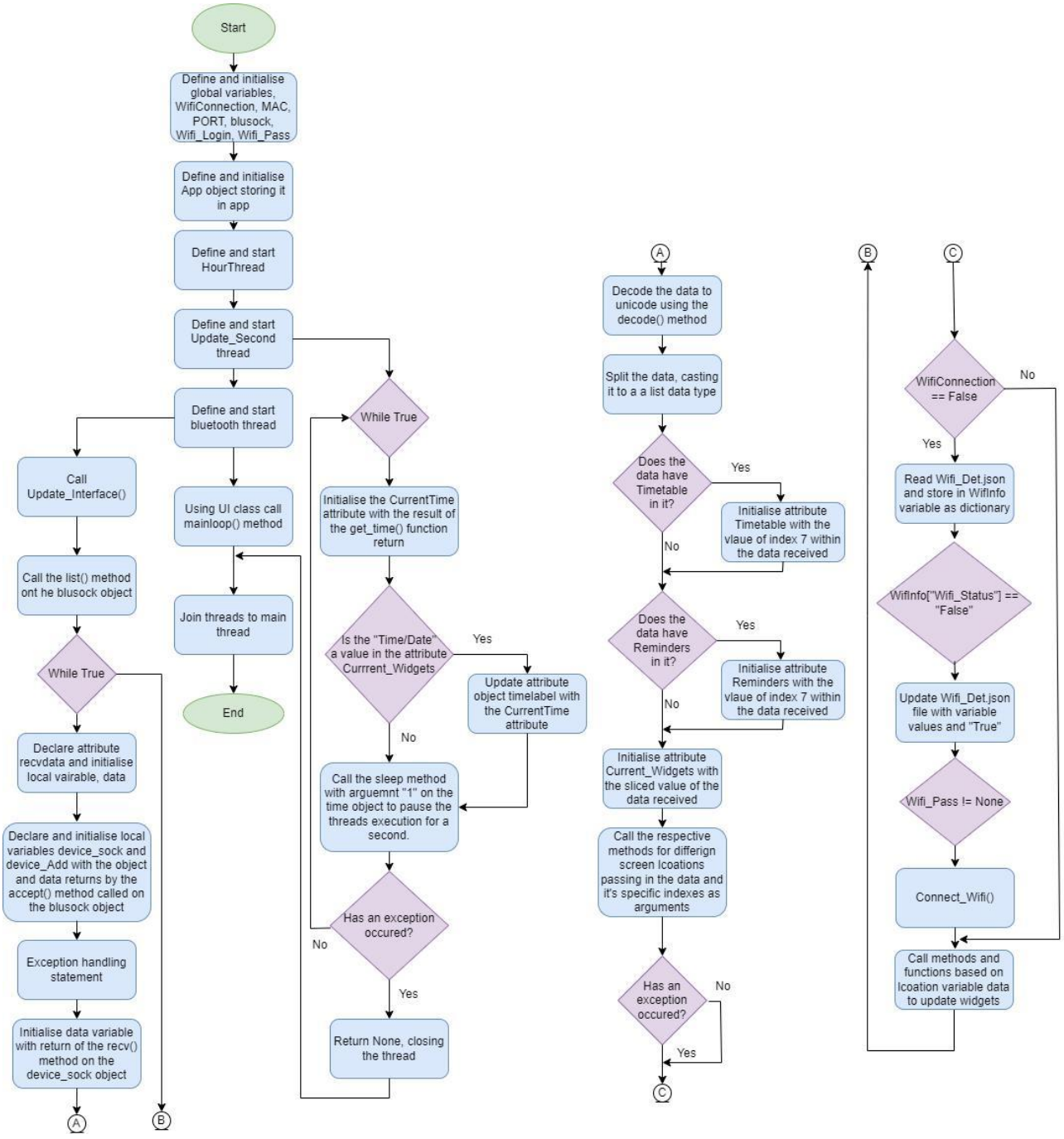



**Figure 1.** The mirror algorithm flow chart.

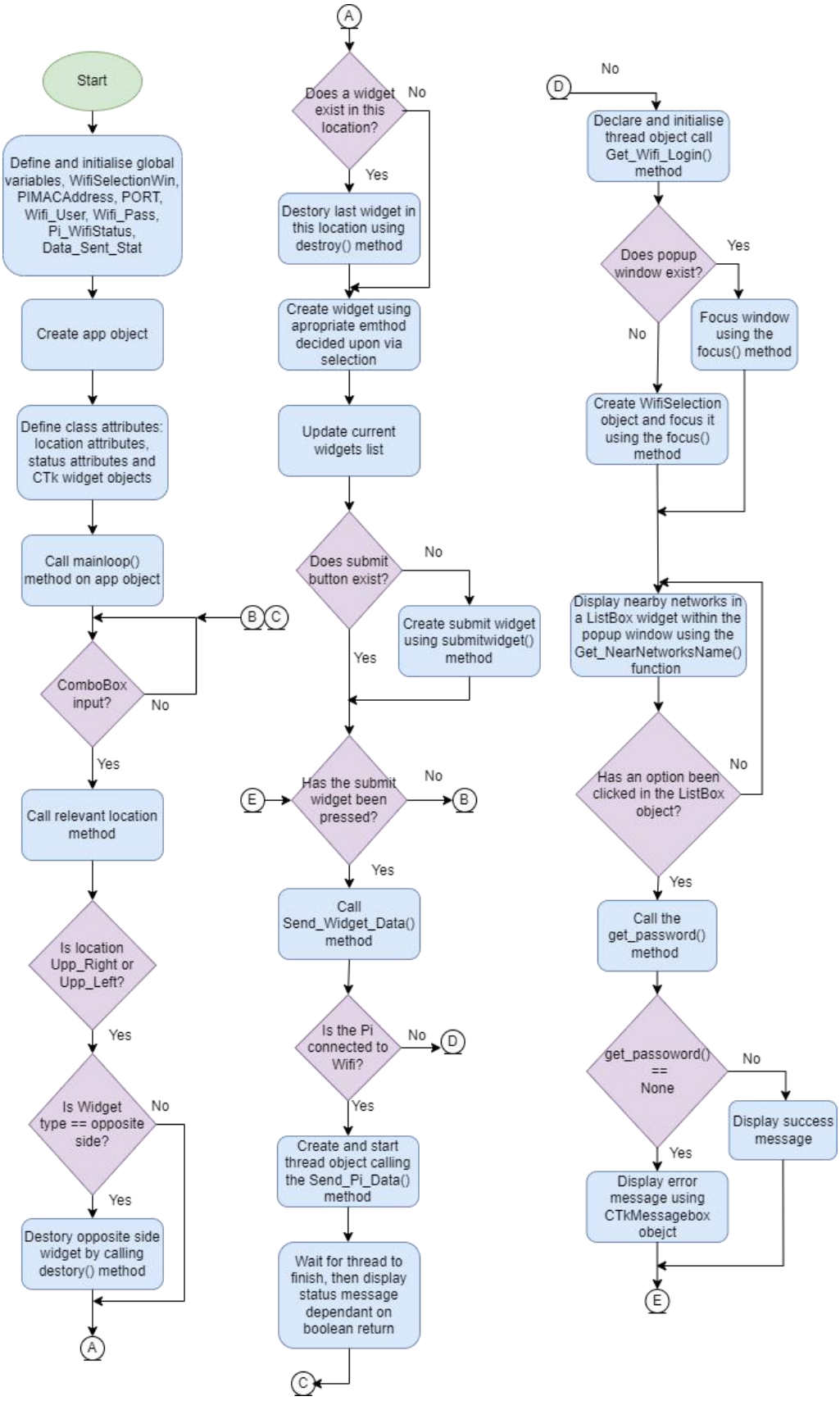



**Figure 2.** The configuration algorithm flow chart.

## 2.2. Hardware

The project involves not only software but also major hardware. The basic principles for selecting the hardware are listed below:

- LCD: The project requires an LCD screen to display the graphical user interface. This screen is located behind a transparent two-way mirrored Perspex sheet. The LCD is a 1024 × 600, a 7 Inch HDMI display, suitable for the planned Graphical User Interface (GUI).
- Raspberry Pi 3B: The Raspberry Pi 3B is a single-board computer with four 1.2 GHz cores with 1 GB RAM. This is sufficient for the application and the physical design of the device due to its small size and high energy efficiency [9].
- Transparent mirror: This material is a two-way mirror. It is acrylic Perspex, so it will not shatter making the device safer. The material allows light to pass through while maintaining its mirror-like appearance.
- Cables: A 12 V power supply is needed for the Raspberry Pi 3B, a USB A to Micro USB cable for display power, and a High-Definition-Multimedia (HDMI) cable connecting the Liquid Crystal Diode (LCD) display to send visual data.

When selecting the hardware, the cost was taken into account. Each component and its respective cost are listed in Table 1. A comparison with other commercial products is also reported in Figure 3. This hardware is displayed in Figure 4 displaying the size and limited hardware used. Figure 5 details the overall proposed design of the system combining all hardware displayed in Figure 4.

**Table 1.** Components and their respective costs.

| Hardware | Cost (£) | Supplier |
|---|---|---|
| Raspberry Pi Model 3B | 27.89 | Amazon |
| Raspberry Pi heat sync | 2.20 | Amazon |
| LCD 1024 × 600 display | 54.99 | Amazon |
| Two-way reflective Perspex (A5) | 10.00 | Ebay |
| 12 V micro USB power supply | 5.30 | Amazon |
| HDMI cable | 3.46 | Amazon |
| Wooden frame | 9.67 | B&Q |
| Total Cost | 113.51 | - |

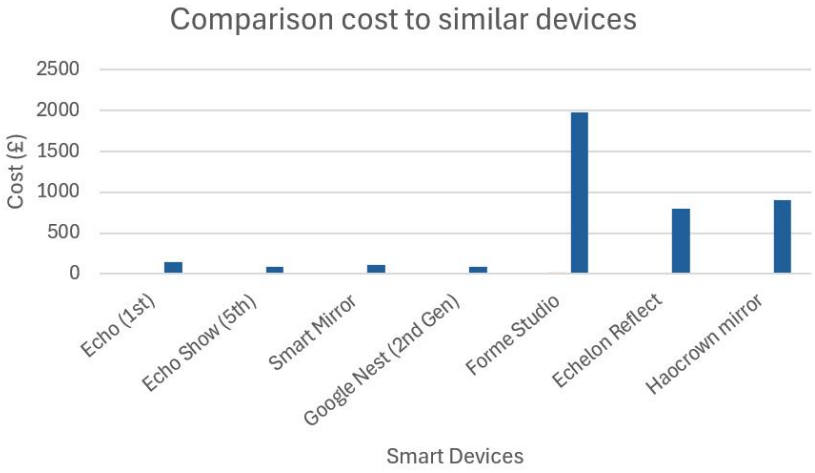



**Figure 3.** Comparison between the cost of similar devices.

Table 1 and Figure 3 show the low-cost nature of the device compared to similar systems in the market (Costs taken on May 2024 from: https://uk.pcmag.com/smart-home/39701/amazon-echo; https://formelife.com/pages/hardware?sscid=51k8_hgutk; https://www.amazon.co.uk/dp/B086MBPXWJ?ascsubtag=&linkCode=gs2&tag=hearstmagazin-21). The device's overall cost is similar to devices with less functionality such as the Amazon Echo, rather than similar smart mirror devices at a higher price, indicating that it is possible to create a configurable low-cost smart mirror.

Power efficiency is another important factor when considering hardware components, and energy certifications can have a significant impact on product sales in order to comply with energy efficiency regulations and to compete with similar smart devices, with Energy Star ratings being a widely recognised program [10].

The Raspberry Pi 3B uses a Reduced Instruction Set Computer (RISC) processor, which is more efficient than a Complex Instruction Set Computer (CISC) processor as less heat is produced [11]. The device also uses a small LCD. With it mounted in a thin wooden frame, with a two-way mirrored Perspex on the front, heat can easily dissipate, so no further cooling is required, because passive cooling is sufficient, reducing energy consumption further.

**Figure 4.** The project hardware layout.

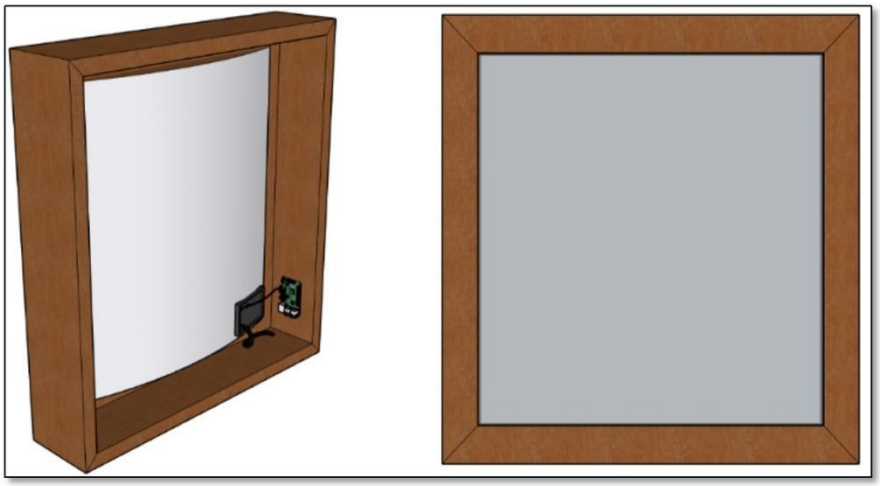


**Figure 5.** Initial sketch design.

## 2.3. Design of the User Interface

Due to the different functionalities of applications, two types of graphical designs of the user interface are required. The interface designs should be easy to use, which can display relevant information and be easily understood to achieve project aims.

### 2.3.1. The Mirror Interface

This interface is displayed on the mirror. It is used frequently, to display data of different quantities and types.

First Iteration - The focus of this design is readability. To achieve this, the information and interface widgets are separated, leaving a gap for the mirror's reflection. The bold large titles ensure that the data is easy to identify and understand. This design includes a quote that generates the daily time greeting, the date, the current Spotify song, and reminders and headlines news on the left top (Figure 6a).

Second Iteration - This design is an improved version of the first iteration which contains less information. However, it is easier to read and understand due to the increased spacing, the more readable "Arial" font and the increased font size [12]. A weather widget has been added to the design to replace Spotify, which requires a paid account; these widgets can be swapped and customised according to user preferences (Figure 6b).


(**a**) First mirror interface design. (**b**) Second mirror interface design.

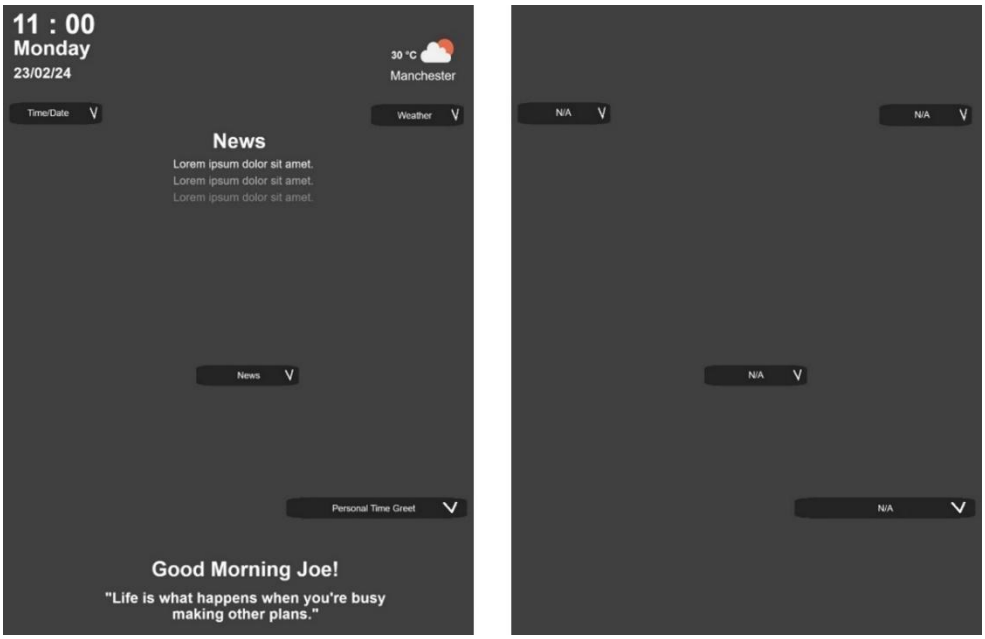
**Figure 6.** First and second mirror interface designs.

This interface will be configurable regarding the displayed information and the location of the data on the interface. Possible configuration changes are listed below:

- Widget interface locations
- Real-time weather data
- Time and Date
- No data
- Real-time news
- Time table
- Reminders
- Updating complements
- Greeting based on time e.g., "Good Morning {name}!"
- Generic time-based greeting

### 2.3.2. Configuration and Setting

The configuration application will run on a Windows machine. Its interface will be like a mirror, simulating the mirror. The interface needs to handle user input, Wi-Fi information and UI options; therefore, the interface will need easily interactable elements (Figure 7).




(**a**) Configuration Interface showcasing widget options.

(**b**) Configuration Interface showcasing no widget options chosen.

**Figure 7.** Configuration interface design.

Dropdown elements are typical for other UIs, making them intuitive and providing a dynamic solution for widget choices as they can add more options without significantly changing the UI [13].

The color of dropdown menus should be great different from that of the background and other UI colors to make it stand out and improve the interface's accessibility. The implementation of this interface can be seen in Figure 8a. This user interface design can be seen implemented into the solution in Figure 8b with differing customisable options.

## 2.4. Development of the Algorithm

The applications are graphically user-centered, therefore programmed modularly. Modularity provides an opportunity to reuse program components in other applications, allowing multiple UI instances to be created, each with its own attributes. The code can be found in the following link: System source code. Figure 7 displays an overview of the resulting interfaces' aspect.

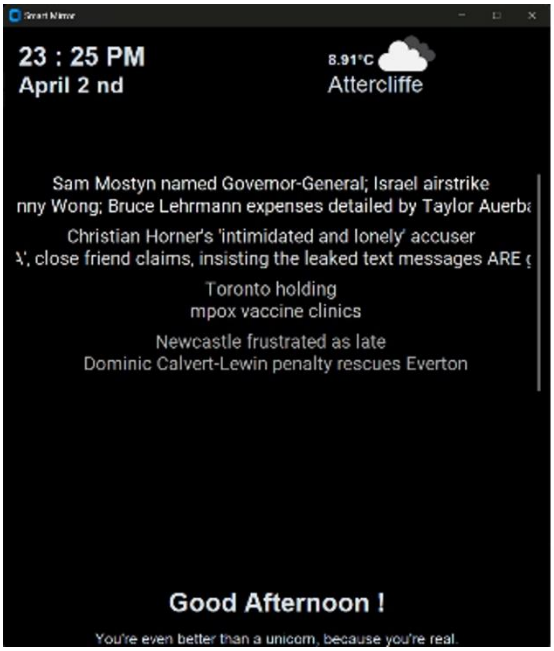
### 2.4.1 Configuration Algorithms

The following functions and methods have been integrated into the algorithm.

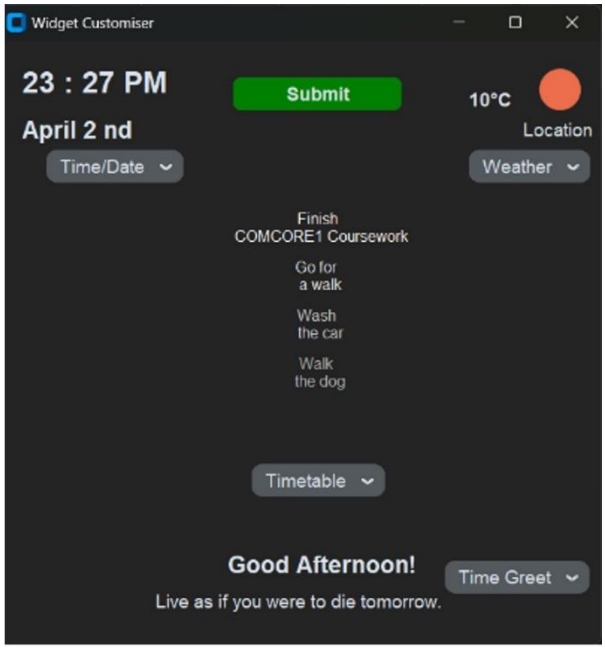
**Send_Pi_Data () -** This function is used in the "Send_Widget_Data" method in the App class. It is used by a separate thread, so that UI can be updated continuously. The function connects to the Pi through Bluetooth, and then formatted parameters and variable data are sent in a specific order and unpackaged relative to this. A Bluetooth socket object is initialised through the socket library, and relevant methods are then used to connect and send encoded byte data to the application's port and the Pi's Mac Address. If the Pi Wi-Fi status stored as a Boolean value in a text file is false, the program will wait for another thread's execution before continuing. This is necessary because the waiting thread will be gathering Wi-Fi credential inputs, which will be sent to the Pi via Bluetooth.

**Get_Near_NetworksName () –** This function is used in the constructor method of the Wi-Fi Selection class. This function is responsible for fetching network credentials so that Pi can connect through Wi-Fi. The function

uses the subprocess library to fetch nearby network data. The extracted data is formatted to Unicode, whitespace is removed and nearby network SSIDS is appended to the local SSID list and returned. If an exception occurs, "Absent" is returned.




(**a**) Mirror interface, displayed on the device.




(**b**) Configuration interface, used to configure the device, used by a separate machine.

**Figure 8.** The mirror interface and the configuration interface.

**Get_NetworkPass() -** Taking a network's SSID as a parameter, this function uses the SSID to execute a terminal command requesting to save network password. The return is decoded and formatted, enabling the password to be fetched and returned. If the password cannot be attained or an exception occurs the local variable "Network_Pass" is initialized to None.

**WifiSelection Class –** This function is used to create a pop-up window interface. Nearby network SSIDs are displayed through the usage of a "ListBox" widget. Once the appropriate network is selected the respective password will be fetched, and the data will be sent to the Pi through the "Send_Pi_Data()" function.

**Methods -** The class's methods are designed to fetch data, process inputs and update interface widgets.

**Update_ConnectionsList() –** This setup is used to update the "ListBox" widget displaying nearby networks. This method first assigns a local RGB variable. This is decremented per appended value and used to change the text color of the object, creating a unique aesthetic and improving readability. The value list passed in by the "values" parameter is looped by a for loop. Each value is formatted and appended to the widget option attribute; a break line character is added to the middle of each value to ensure readability.

**Get_password()** – The function is responsible for fetching network credentials indicated by the SSID parameter and updating necessary variables. This function first strips the SSID to avoid errors. After stripping, the "Get_NetworkPass" method will be used with the SSID parameter. Through selection, a status message is displayed via a "CTKMessageBox" object, dependent on the result returned by the "Get_network" function use. If "None" has been returned, a button object will be created to prompt the user to continue to use Bluetooth. The variables "Wifi_Pass" and Pi Wi-Fi status will be updated.

**ContinueBlue() -** This method is used by a button widget, the object offers the option to continue with a Bluetooth connection. And it will only be created if the network password can't be attained. The method destroys the "WifiSelection" object instance through the usage of the "destroy()" method.

**App class –** This class defines the main window display. Due to the similarity to the App class of the Mirror script, the differences have been listed:

**ComboBoxes and Inputs -** This script version of the class uses "Combobox" widgets offering configuration input. Example data is used whenever possible instead of making API requests, as this is not necessary to represent the configuration. In addition to "ComboBox" widgets, the class fetches input through the usage of "CTkInputDialog" objects which produce pop-up windows. These objects prompt Timetable and Reminder input, which is sent to the Pi after submission.

**Widget placements -** Due to differences between screen size and of "ComboBox" widget requirements, interface elements are placed in different locations through screen coordinates without using anchors as the data is predefined. Widget placement is representative of the mirror interface.

### 2.4.2 Mirror Algorithms

The Raspberry Pi 3B has the least resources and is passively cooled, so in order to avoid overheating, the program must be time-saving and memory-efficient. The mirror will only have network input and will likely be left on for days. Therefore, it is essential for the program to detect and handle errors efficiently.

**Get_IP_Location() –** This function is used to fetch the Pi's public IP address location. And its return is used to gain weather information. The function will make an API request through the usage of the requests library, this is returned in a JSON format.

The JSON format will be decoded into a dictionary data type and relevant information will be fetched and returned. If an exception occurs the longitude and latitude for Manchester will be returned.

**Get_Weather_Data() –** This class is responsible for fetching weather data. This function takes latitude and longitude as parameters which are used to make an API request. The returned JSON data is formatted to a data type dictionary, and the relevant data will be saved and returned. If an exception occurs, the universal image path of no Wi-Fi icon will be returned.

**Get_Date_prefix() –** The function takes an integer value as a parameter to represent the date. A local dictionary is defined and initialised with the relevant date postfixes, through the selection of relevant postfix to be returned.

**Get_quote() -** This function produces an API request, fetching a random quote. The request return is cast from JSON to dictionary data type and the quote is fetched. If the quote's length is greater than eight or less than five characters, recursion is used to request another, otherwise the quote is returned.

Quote length is limited to maintain usability and readability.

**Get_time()** - The result of the "now()" method on the "DateTime" object is returned, providing the current date and time.

**Class App -** This class is very similar to the class used in the configuration application. It is used to create a main window instance. This window acts as the primary user interface displaying widgets according to the user preference.

**App class methods -** This class has been designed to differ in method functionality, including different location and the updating and creation of widgets.

### 2.4.3 Location Methods

**Location methods** act as controllers for specific areas of the screen. They use methods dependent on parameters to create or destroy widgets.

**Upp_left(), Upp_Right(), Bott_Centre(), Centre() -** Sharing the same functionality,hese methods create the widget specified by the parameter "type" through the selection of appropriate method use. If the widget already exists, it will be swapped from that location by updating its attribute and using the "Widget_Destory()" method. This prevents errors from occurring due to multiple API requests and unsupported thread instances overloading the Pi.

### 2.4.4 Widget Updating Methods

**Widget updating** methods are used to update specific widgets based on parameter values. They are typically used within the time updating threads.

**Update_Compliment(), Update_Greeting(), Update_Quote(), Update_Day(), Update_Weather() and Update_Date() -** Sharing similar functionality, these methods are necessary to obtain up-to-date relevant information to update widgets. Widgets are updated using the "configure()" method.

**Update_Interface() and Update_Hour() -** Responsible for updating widgets at specific times, these methods are calld in separate threads. The methods require constant CPU attention as "while true" loops are used to avoid recursion depth limits. Widgets are updated using the sleep() method of the "time" class and selection specifying which widgets to update.

Instead of using time the "Update_Interface()" method acts upon a Bluetooth connection. When data is received it is decoded, unpackaged, and the relevant widget objects and variables updated.

### 2.4.5 Widget Creation Methods

**Widget creation methods** are called to create a new widget, which depends on parameter value and other widgets.

**WeatherWidget(), Create_Greeting_Widget(), Create_Subgreet_Widget(), Create_ListWidget() and Greeting_Widget() -** Sharing similar functionality, these methods update necessary widget objects, depending on parameter and attributes, by fetching relevant information.

## 3. Results

An input/output table has been used to test both applications. The mirror application do not rely on input from the configuration algorithm. Therefore, this test will identify errors in both applications (Table 1). Table 2 shows that the applications can handle boundary, erroneous and normal input data.

**Table 2.** Results of the testing.

| Data Type | Input | Output |
|---|---|---|
| Normal | Upper Right Interface combo box choices | Expected, interface updated |
| Normal | Upper Left Interface combo box choices | Expected, interface updated |
| Normal | Centre Interface combo box choices | Expected, interface updated |
| Normal | Bottom Centre Interface combo box choices | Expected, interface updated |
| Erroneous | Upper Right Interface choice is equal to Upper left Interface choice | Expected, error handled, the widgets were swapped |
| Erroneous | The network chosen has no network password saved on the system | Expected, error handled, the continue Bluetooth button was displayed |
| Boundary | The submit button is pressed multiple times while data is sent | Expected output, the configuration data sent as normal |
| Erroneous | The submit button is pressed while the mirror is not connected | Expected output, error handled, warning message displayed notifying the user to "try again" |

A 24-hour use test has been conducted, including the hardware and the application's usability. The device and application stayed on for 24 hours without major error. However, a logical error did occur. The date did not update, after examination, it was found the "Update_Hour()" method was comparing the "date" attribute to the starting date local variable that wasn't being updated, this resulted in the "Update_day()" method not being called. Figure 9 displays the results of this test.

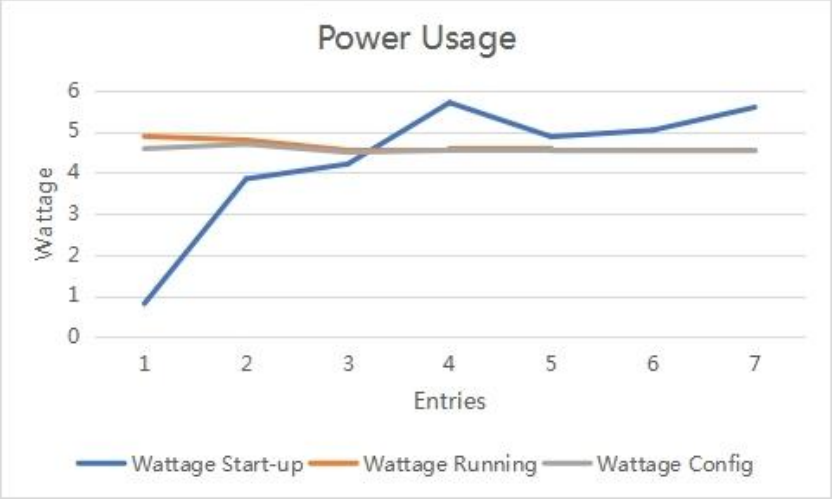



**Figure 9.** The device's power consumption.

A comparison with other devices in the power consumption is also reported in Figure 10 (Power consumption reported on data sheet of the products at the following links: howtogeek.com (How Much Electricity Does the Amazon Echo Use?); https://www.argos.co.uk/product/9325195?clickSR=slp:term:smart%20home:5:566:2). As shown in the Figures 9 and 10, the smart device uses minimal power during the initial start-up of the system with a maximum power of 5.58 W. Compared to similar products, the device outperforms them significantly with the only exception being the Amazon Echo. This device provides fewer functions and lacks a screen. This comparison shows that this design's power consumption is closer to a device offering significantly less functionality than a similar device that displays data visually. The final system can be seen in Figure 11.

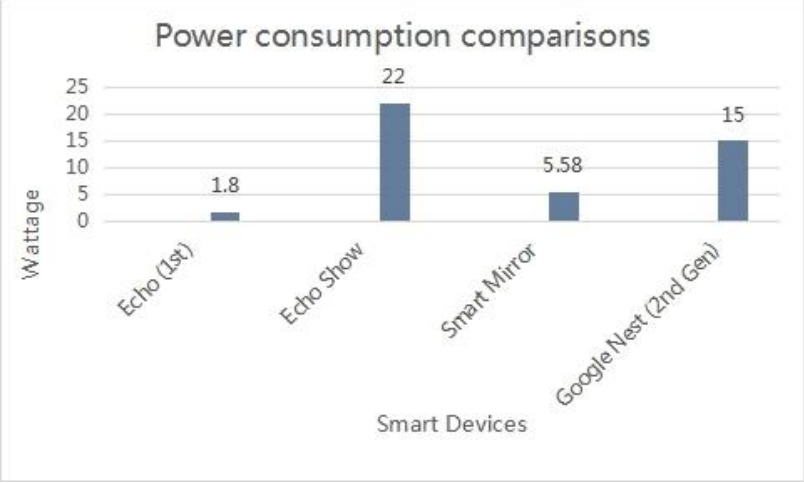



**Figure 10.** The average power consumption of different devices.

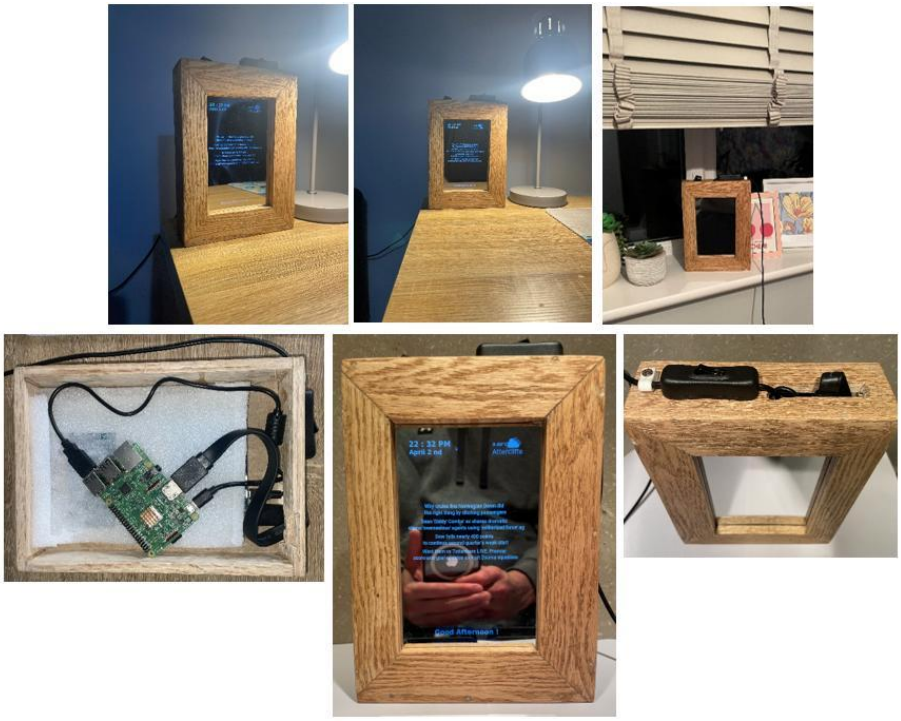


**Figure 11.** The Smart Mirror final iteration.

## 4. Conclusions

The following aims and objectives have been covered in the project:

- **User data is secure -** User data is kept secure, and sensitive information is not recorded, or sent anywhere else other than the mirror via the Bluetooth connection.
- **The application displays data graphically -** This has been met, as seen by the images below. The application displays data graphically via a graphical user interface.
- **The application displays up-to-date relevant information -** The application does display relevant up-to-date information by using various methods and update the interface's widgets.
- **The user can configure where and what data is displayed -** Through the configuration application, widget location and type can be changed.
- **The interface is easy to understand and use -** The interface has been designed to be intuitive by increasing readability and following common standards.
- **There are a series of widget options -** The configuration application offers multiple widget options supported by the mirror.
- **The device can connect to the internet wirelessly -** The device can connect to the internet via WI-FI with the application making API requests.
- **The device can connect to devices via Bluetooth -** The device is able to connect to different windows devices via Bluetooth.

In summary, the project has highlighted the shortcomings of home technologies and the benefits of innovation within this sector. This design can be produced quickly at a low cost, customised according to users' requirements, and used in a range of applications. The device can display the latest relevant information in an easily understandable way, which can be configured to user needs.

Clearly, a series of further work can be foreseen: the smart mirror proposed in this project can be easily improved, the software is modularized, and designed to allow perfect maintenance. The next step for this device is to gain further user feedback to add additional widgets to support interface changes, and consider integrating machine learning and assistive technologies to further provide service and support for end-users [14,15]. Such a system could also be integrated with a set of sensors and other smart home devices or connected to an Ambient

Assisted Living or medical system [16,17]. In this context, it is reasonable to also foresee the integration with gesture recognition systems making the mirror more intuitive when interacting with the end-user [18,19]. Hardware could be changed to increase size and reduce depth; increasing usability and functionality while reducing noticeability. And Artificial Intelligence and augmented reality could be implemented by using a camera. More importantly further support should be added for different devices to achieve greater accessibility when configuring the device. The configuration application could be moved to the web through the usage of Javascript, HTML and CSS to increase the application's portability and ultimately its accessibility.

## Supplementary Materials

The project code is reported on the following GITHUB repository – System source code.

## Author Contributions

Conceptualisation, J.R.L. and E.L.S.; methodology, J.R.L.; software, J.R.L.; validation, J.R.L.; supervision, O.A. and E.L.S.; writing—original draft preparation, J.R.L. and E.L.S. All authors have read and agreed to the published version of the manuscript.

## Funding

This work received no external funding.

## Institutional Review Board Statement

Not applicable.

## Informed Consent Statement

Not applicable.

## Data Availability Statement

We encourage all authors of articles published in our journals to share their research data. In this section, please provide details regarding where data supporting reported results can be found, including links to publicly archived datasets analyzed or generated during the study. Where no new data were created, or where data is unavailable due to privacy or ethical restrictions, a statement is still required.

## Acknowledgments

This work was completed by Joe Lyons as part of his coursework requirements for the BSHH in Computer Science at Liverpool Hope University's within the School of Mathematics, Computer Science, and Engineering. We thank Mr I Steel for his support.

## Conflicts of Interest

The authors declare no conflict of interest.

## References

1. Buil-Gil, D.; Kemp, S.; Kuenzel, S.; Coventry, L.; Zakhary, S.; Tilley, D.; Nicholson, J. The Digital Harms of Smart Home Devices: A Systematic Literature Review. *Comput. Hum. Behav.* **2023**, *145*, 1–3. [CrossRef]
2. Morris, M.; Adair, B.; Miller, K.; Ozanne, E.; Hansen, R.; Pearce, A.; Santamaria, N.; Viega, L.; Long, M.; Saidet, C. Smart-Home Technologies to Assist Older People to Live Well at Home. *J. Aging Sci.* **2013**, *1*, 1–9. [CrossRef]
3. Kulovic, S.; Ramic-Brkic, B. DIY Smart Mirror. In Proceedings of the International Symposium on Innovative and Interdisciplinary Applications of Advanced Technologies, Jahorina, Bosnia and Herzegovina, 21–24 June 2018. [CrossRef]
4. Smart Display Market Size, Share and Industry Analysis | 2028 (alliedmarketresearch.com). Available online: https://www.alliedmarketresearch.com/smart-display-market-A11780#:~:text=The%20smart%20display%20market%20size%20was%20valued%20at,a%20CAGR%20of%2021.6%25%20from%202021%20to%202028 (accessed on 7 June 2024).

5. Guhr, N.; Werth, O.; Blacha, P.P.H.; Breitner, M.H. Privacy Concerns in the Smart Home Context. *SN Appl. Sci.* **2020**, *2*, 1–12. [CrossRef]
6. The Python Language Reference. Available online: https://docs.python.org/3/reference/index.html (accessed on 7 April 2024).
7. Scanlan, D.A. Structured Flowcharts Outperform Pseudocode: An Experimental Comparison. *IEEE Software* **1989**, *6*, 28–36. [CrossRef]
8. Mahmmod, B.M.; Flayyih, W.N.; Fakhri, Z.H.; Abdulhussain, S.H.; Khan, W.; Hussain, A. Performance Enhancement of High Order Hahn Polynomials Using Multithreading. *PLoS One* **2023**, *18*. [CrossRef]
9. Gamess, E.; Hernandez, S. Performance Evaluation of Different Raspberry Pi Models for a Broad Spectrum of Interests. *Int. J. Adv. Comput. Sci. Appl.* **2022**, *13*, 819–828. [CrossRef]
10. Brown, R.; Webber, C.; Koomey, J.G. Status and Future Directions of the Energy Star Program. *Energy* **2002**, *27*, 505–520. [CrossRef]
11. Gupta, K.; Sharma, T. Changing Trends in Computer Architecture: A Comprehensive Analysis of ARM and x86 Processors. *Int. J. Sci. Res. Comput. Sci., Eng. Inf. Technol.* **2021**, *7*, 619–631. [CrossRef]
12. Tullis, T.S.; Boynton, J.L.; Hersh, H. Readability of Fonts in the Windows Environment. In Proceedings of the Human Factors in Computing Systems, CHI '95 Conference Companion: Mosaic of Creativity, Boston, USA, 7–11 May 1995. [CrossRef]
13. Raskin, J. Viewpoint: Intuitive Equals Familiar. *Commun. ACM* **1994**, *37*, 17–18. [CrossRef]
14. Innes, M.; Secco, E.L. An Understanding of How Technology Can Assist in the Epidemic of Medicine Nonadherence with the Development of a Medicine Dispenser. *Eur. J. Appl. Sci.* **2023**, *11*, 522–550. [CrossRef]
15. Manolescu, V.D.; Secco, E.L. Design of an Assistive Low-Cost 6 d.o.f. Robotic Arm with Gripper. In Proceedings of the 7th International Congress on Information and Communication Technology, London, UK, 21–24 February 2022. [CrossRef]
16. Van Eker, M.; Secco, E.L. Development of a Low-cost Portable Device for the Monitoring of Air Pollution. *Acta Sci. Comput. Sci.* **2023**, *5*. [CrossRef]
17. Brown, K.; Secco, E.L.; Nagar, A.K. A Low-Cost Portable Health Platform for the Monitoring of Human Physiological Signals. In *EAI International Conference on Technology, Innovation, Entrepreneurship and Education*, 1st ed.; Reyes-Munoz, A., Zheng, P., Crawford, D., Callaghan, V., Eds.; Springer: Canterbury, Great Britain, 2016; Volume 532, pp. 221–224. [CrossRef]
18. McHugh, D.; Buckley, N.; Secco, E.L. A Low-cost Visual Sensor For Gesture Recognition via AI CNNS. In Proceedings of the Intelligent Systems Conference, Amsterdam, Netherlands, 3–4 September 2020.
19. Buckley, N.; Sherrett, L.; Secco, E.L. A CNN Sign Language Recognition System With Single & Double-handed Gestures. In Proceedinggs of IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), Madrid, Spain, 12–16 July 2021. [CrossRef]