Journal Name
http://ojs.ukscip.com/index.php/xxxx

Article Type (Article, Review, Communication, etc.)

Raspberry-PI based design of an interactive Smart Mirror for daily life

Joe Reginald Lyons [1], Ogbonnaya Anicho [1] and Emanuele Lindo Secco [1,*]

[1] 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: Day Month Year; **Accepted**: Day Month Year; **Published**: Day Month Year

Abstract: The Internet of Things and spatial computing are increasingly becoming pervasive in today's technological landscape. These devices can sometimes offer the counter effect of complicating interaction between non-expert end-users and the device itself. In this paper, we propose a simple, user-friendly, cost-effective configurable smart mirror able of displaying useful, relevant real-time information. This system is designed around a low-cost Raspberry PI, paired with an LCD screen the system can be connected to a PC via the IEEE 802.15 wireless communication protocol. Preliminary results showed the intuitive usability of the device in a daily life context.

Keywords: internet of things; low-cost interactive design; intuitive design; user-friendly design.

1. Introduction

The smart home industry is an ever-growing industry [1]. Smart mirrors are a type of home technology, they display relevant useful information, offering applications in health and energy efficiency [2].

Currently, there is a limited market for the device, hobbyists primarily make them, and it is “almost impossible to acquire one” [3], therefore it can be deduced that there is a lack of customisable software, a deficit we aim to fulfil within this project. Like smart mirrors, *smart displays* display relevant information visually. Evaluated at $3.78 billion in 2020 the global smart display market continues to rise, signifying the increasing demand for smart home technologies [4]. This rise in popularity is partly due to an increase in functionality and configurability, with voice assistants now being integrated into the technology, further improving functionality.

Homes are bespoke 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 are not being adopted, we think this is due to a lack of configurability and cost. Some smart mirrors do offer 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:

- *Low-cost* smart mirror design
- *Configurable* smart mirror design
- *Energy-efficient* smart mirror design
- Overview of the *deficits* within 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 aims and objectives: the device will be used frequently, it will be left on for prolonged periods and be part of the user's home, therefore the proposed design should be *robust*, *aesthetically pleasing*, and *functional*.

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

- User data is kept secure.
- The application and device display data graphically.
- Up-to-date relevant information is displayed.
- The user can configure where and what data is displayed.
- The interface is easy to understand and use.
- There is a range of widget options
- The device can connect to the internet wirelessly.
- The device can connect to devices 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 an evaluation and conclusion of the system.

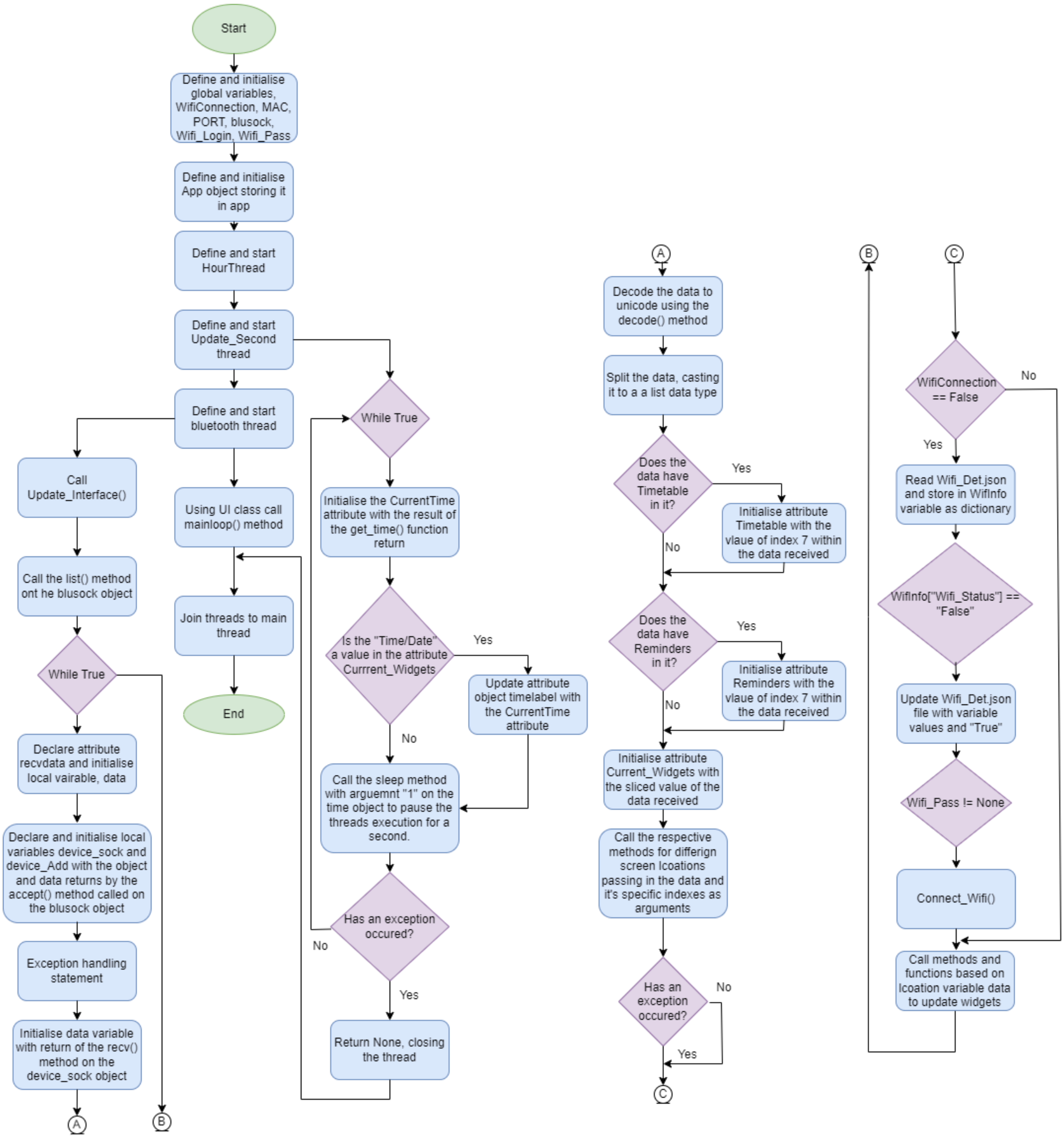
Two applications are developed using Python 3.12.2, one for configuration, running on a Windows machine and one running the mirror’s interface on a Raspberry Pi 3B. Python is a high-performance, portable language supporting rapid development, this allows for ease of development reducing development time and multiple iterations of the device being made across a range of operating systems [6].

2. Materials and Methods

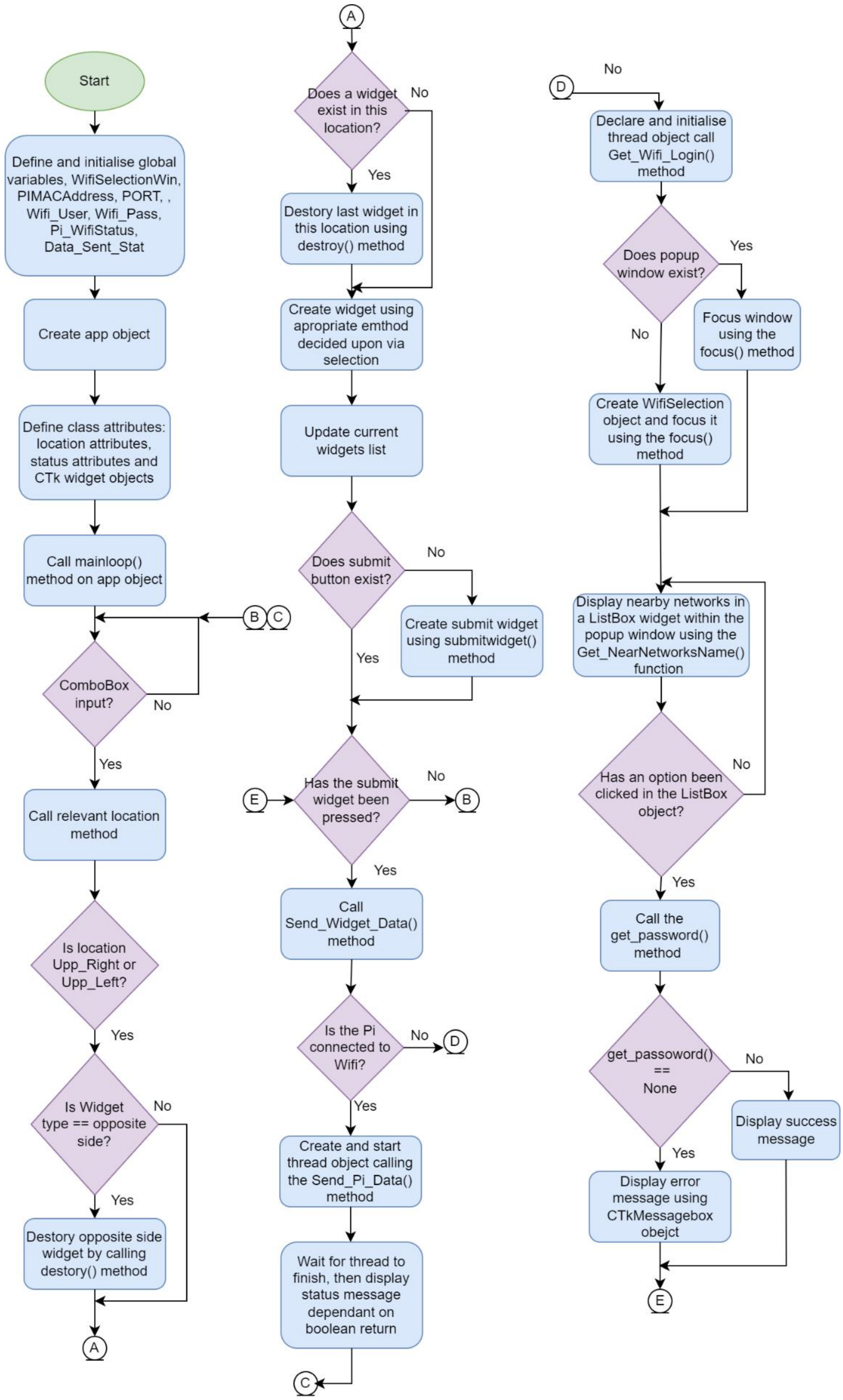
The device will need access to the internet for API requests, a Wi-Fi network is ideal for this, however, the mirror cannot connect to a Wi-Fi Network due to a lack of credentials and input. Instead, the application will communicate via IEEE 802.15 protocol, namely Bluetooth PAN initially, receiving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

A set of algorithms need to be scheduled and planned: precisely, two differing flowcharts detail the execution path of the applications, helping application development, decreasing development time, and ensuring project aims are met by aiding algorithm comprehension [7]. An overview of these algorithms is reported in Figures 1 and 2.

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



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



73

2.1. Software

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

- The requests library is required to retrieve API information that the application displays.
- *CustomTkinter* is used to create the graphical user interface, it adds methods and classes that can be used to create custom graphical interfaces.
- *Pillow* is an image library used to load and format images so they can be displayed graphically.
- The subprocess library allows terminal commands to be made from within the program, this is required to fetch nearby network credentials which are sent to the Pi and used to connect.
- Threading is required for simultaneous multi-threading; this can lower program execution time allowing processes requiring constant CPU attention to be executed [8].
- The *socket library* is used to connect and send data via Bluetooth to and from the mirror.
- *Datetime* is required to retrieve the system time and date, this is parsed and the widget updated as necessary.
- *CTkListBox* is a small library built on top of "CustomTkinter", it adds functionality for another type of graphical widget.
- *CTkMessageBox* is another small library built on "CustomTkinter", it adds functionality for pop-up message windows.

2.2. Hardware

As well as having a significant software aspect, the project also has a significant hardware aspect; I have listed the hardware and the rationale for these choices below:

LCD - the project requires an LCD screen to display the graphical user interface, this screen sits behind a transparent two-way mirrored Perspex sheet. The LCD is 1024X600, a 7" Inch HDMI display, the right size for the planned GUI.

The *Raspberry Pi 3B* is a single-board computer, this model boasts four 1.2GHz cores with 1GB RAM. This is sufficient for my application and for the physical design of the device due to its small size and energy efficiency [9].

Transparent mirror, this material is a two-way see-through mirror, it is acrylic Perspex, therefore won't smash making the device safer. The material allows light to pass through while maintaining its mirror-like appearance.

Cables - a 12V power supply is required for the Raspberry PI 3B and a USB A to Micro USB with a HDMI for the screen.

When selecting the hardware, the cost was taken into consideration. Each component and its respective cost are reported in Table 1. A comparison vs other commercial products is also reported in Figure 3.

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 1024X600 display	54.99	Amazon
Two-way reflective Perspex (A5)	10.00	Ebay
12V micro USB power supply	5.30	Amazon
HDMI cable	3.46	Amazon
Wooden frame	9.67	B&Q
		Total Cost: 113.51

Figure 3. Graph comparison between the cost of similar devices

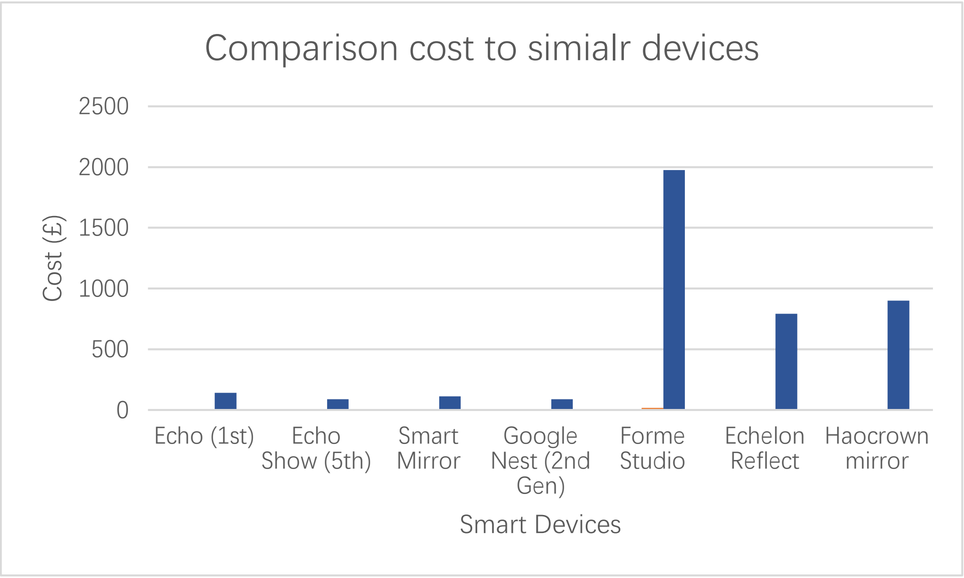


Table 1 and Figure 3 show the low-cost nature of the proposed device vs similar systems in the market[1]. The device's overall cost is similar to devices of less functionality such as the Amazon Echo, not the more expensive similar smart mirror devices, showing it is possible to create a configurable low-cost smart mirror.

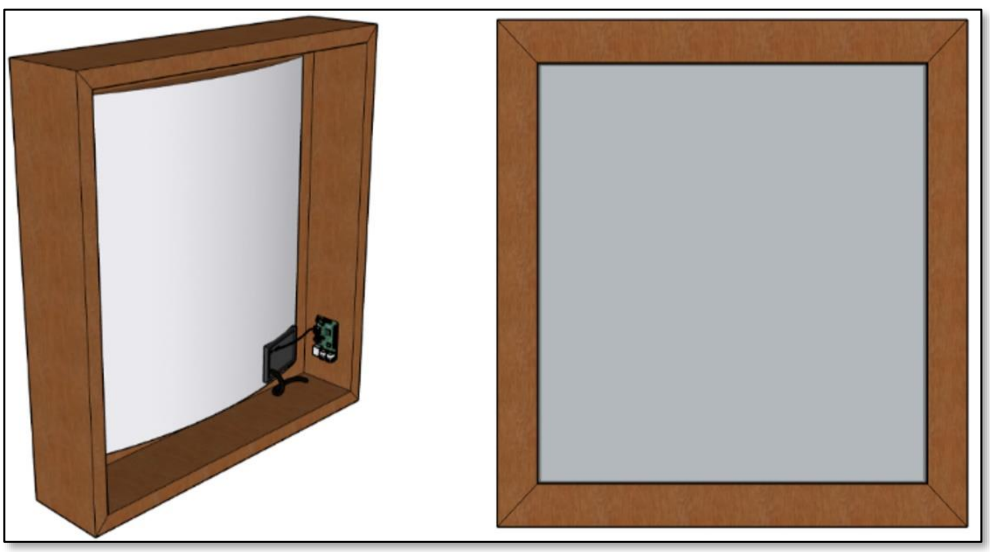
Power efficiency was another important factor when considering the hardware components, to comply with energy efficiency regulations and compete with similar smart devices, energy certifications can impact product sales significantly, such as the widely recognised and adopted Energy Star rating program [10].

The Raspberry Pi 3B uses a RISC ARM processor, which is more efficient than CISC x86 processors as less heat is produced [11], the device also uses a small LCD. Housed within a thin wooden picture frame, with a perspex two-way mirrored front, heat can easily dissipate, therefore, further cooling was not required, as passive cooling sufficed, reducing energy consumption further.

[1] 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.

Figure 4. The project hardware layout

Figure 5. Initial Sketch Up design



2.3. Design of the User Interface

Due to the applications differing functionality, two graphical user interface designs are required. The interface designs should be easy to use, display relevant information and be easily understood as outlined within the project aims.

Mirror Interface

This Interface is displayed on the mirror, it is used frequently and constantly, displaying data varying in size and type.

First Iteration - This design has had a focus on readability, to achieve this, information and interface widgets are spaced, and a gap in the middle has been left for the mirror's reflection. The bold large titles ensure that data is easily identifiable and understandable. This design includes a quote which is generated daily, a time greeting, the date, reminders, current Spotify song, and the top news headlines (Figure 6, left panel).

Figure 6. First and second mirror interface designs, panel left and right, respectively

Second Iteration - This design is a revamped version of the first iteration, it contains less information, however, it is easier to read and understand due to the increased spacing, the more readable "Arial" font and the increase in font size [12]. A weather widget has been added to the design replacing Spotify which requires a paid account; these widgets can be swapped and customized to user preference (Figure 6, right panel).

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

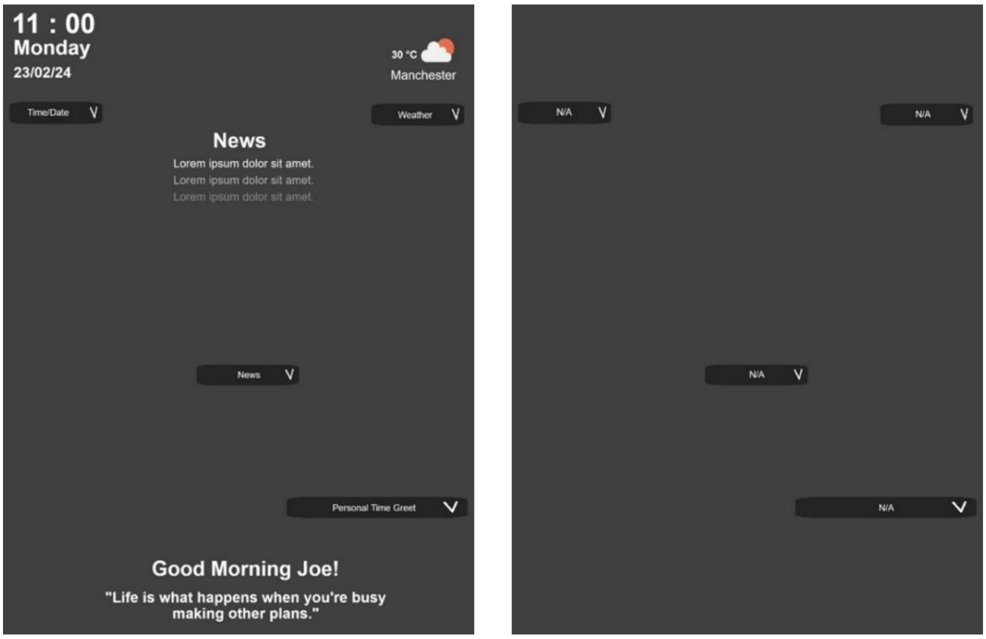
- Widget interface locations
- Real-time weather data
- Time and Date
- No data
- Real-time news
- Time-table
- Reminders
- Updating complements

- Greeting based on time e.g. “Good Morning {name}!”
- Generic time-based greeting

Configuration and Setting

The configuration application will run on a Windows machine, its interface will be similar to the mirror’s interface, 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).

Figure 7. Configuration interface design



Dropdown elements are typical for other UIs making them intuitive, they offer a dynamic solution to widget choices as they can be appended with more options without significantly changing the UI [13].

Dropdown menus have been coloured to contrast the background and other UI colours so that they stand out, improving the interface’s accessibility.

2.4. Development of the Algorithm

The applications are graphically user-centred and, therefore programmed modularly. Modularity provides the opportunity to reuse program parts in other applications, allowing multiple UI instances to be created, each with its own attributes. The code can be found here at the following link: [System source code.]

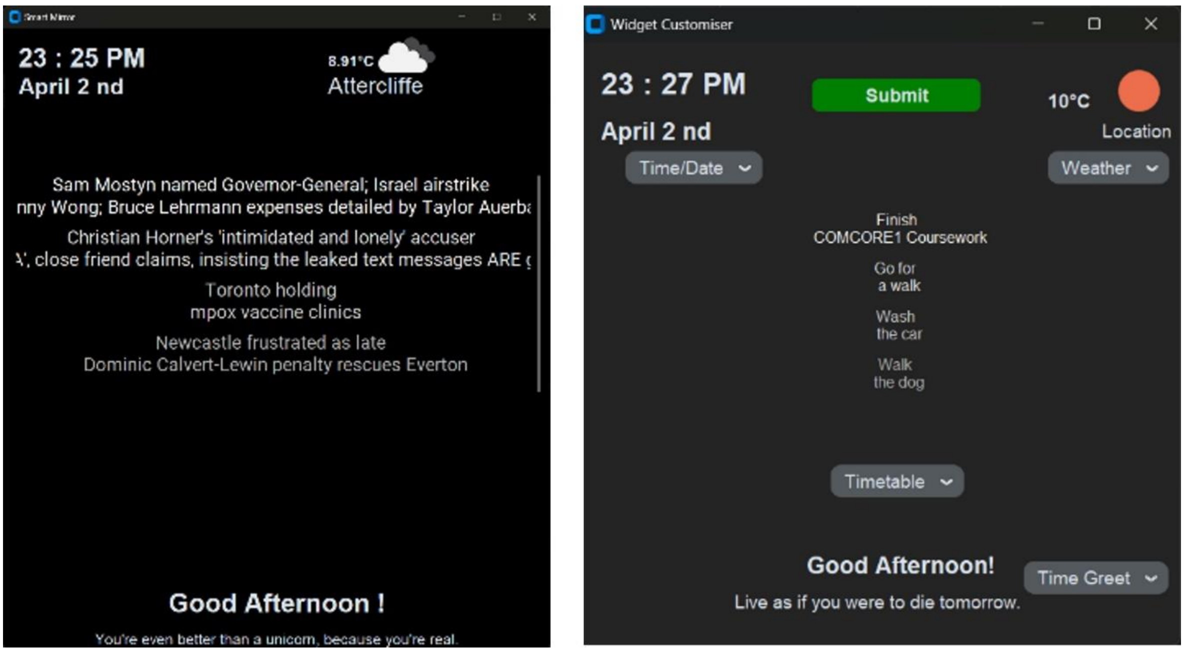
A. Configuration Algorithms

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

Send_Pi_Data() - This function is called by the “Send_Widget_Data” method within the “App” class, it is called by a separate thread enabling the UI to keep updating. The function connects to the Pi via Bluetooth, formatted parameters and variable data are then sent in a specific order and unpackaged relative to this. A Bluetooth socket object is initialised using the socket library, relevant methods are then called to connect and send encoded byte data to the application’s port and the Pi’s Mac Address. If the Pi’s Wi-Fi status stored as a Boolean value within a text file is false, the program awaits another thread's execution before continuing. This is required as the awaiting thread will be gathering Wi-Fi credential inputs, which will be sent to the Pi via Bluetooth.

Get_Near_NetworksName() – This function is called within the WI-FI Selection class's constructor method, this function is responsible for fetching network credentials enabling the Pi to connect via Wi-Fi. The function uses the subprocess library to fetch nearby network data. Fetched data is formatted to Unicode, whitespace is removed and nearby network SSIDS are appended to the local SSID list and returned, if an exception occurs "Absent" is returned.

Figure 8. The mirror interface and the configuration interface on the left and right panels, respectively



Get_NetworkPass() - Taking a network's SSID as parameter, this function uses the SSID to execute a terminal command requesting the saved 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 popup window interface, nearby network SSIDs are displayed using a "ListBox" widget. Once the appropriate network is selected the respective password is fetched, and the data is sent to the Pi using the "Send_Pi_Data()" function.

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

Update_ConnectionsList() – This set up 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 object's text color, this creates a unique aesthetic and improves readability. The value list passed in by the "values" parameter is looped by a for loop. Each value is formatted and appended to the widgets options 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. Once stripped the "Get_NetworkPass" method is called with the "SSID" parameter. Using selection, a status message is displayed via a "CTKMessageBox" object, dependent on the result returned by the "Get_network" function call. If "None" has been returned, a button object is created prompting the user to continue with Bluetooth. The variables "Wifi_Pass" and Pi Wi-Fi status are updated.

ContinueBlue() - This method is called by a button widget, the object offers the option to continue with a Bluetooth connection, only created if the network password can't be attained. The method destroys the "WifiSelection" object instance using 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's version of the class uses "Combobox" widgets offering configuration input. Example data is used where possible instead of making API requests as this is not necessary to represent the configuration. As well as "ComboBox" widgets, the class fetches input using "CTkInputDialog" objects which produce pop-up windows. These objects prompt Timetable and Reminder input, this is sent to the Pi after submission.

- **Widget placements -** Due to screen size differences and "ComboBox" widget requirements, interface elements are placed in differing locations using screen coordinates, anchors aren't used as data is pre-defined. Widget placement is representative of the mirror interface.

B. Mirror Algorithms

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

Get_IP_Location() – This function is called to fetch the Pi's public IP address location, this function's return is used to gain weather information. The function will make an API request using the requests library, this is returned in a JSON format.
The JSON format will be decoded into a dictionary data type and relevant information fetched to be 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 is held and returned, if an exception occurs, the image path of the universal no Wi-Fi icon is returned.

Get_Date_prefix() – The function takes an integer value as a parameter representing the date. A local dictionary is defined and initialised with the relevant date postfixes, using selection the relevant postfix is returned.

Get_quote() - This function makes 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, returning 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 per user preference.

App class methods - This class has been designed to differ in method functionality, the differing types being location, widget updating and widget creation.

Location methods

- **Location methods** act as controllers for specific areas of the screen, they call methods dependent on parameters, they can create or destroy widgets.

- **Upp_left(), Upp_Right(), Bott_Centre(), Centre() -** Sharing the same functionality, these methods using selection and appropriate method calls create the widget specified by the parameter "type". If the widget already exists, it is swapped from that location by updating that location's attributes and calling the "Widget_Destory()" method. This prevents errors from occurring due to multiple API requests and unsupported thread instances overloading the Pi.

Widget updating methods

- **Widget updating** methods are called to update specific widgets dependant on parameter values, they are typically called within the time updating threads.

- **Update_Compliment(), Update_Greeting(), Update_Quote(), Update_Day(), Update_Weather() and Update_Date() -** Sharing similar functionality these methods call necessary fetching methods and functions getting 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 called 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.

Widget creation methods

- **Widget creation methods** are called to create a new widget, the widget created is dependent on parameter value and other widgets.

- **WeatherWidget(), Create_Greeting_Widget(), Create_Subgreet_Widget(), Create_ListWidget() and Greeting_Widget() -** Sharing similar functionality these methods update the necessary widget objects, depending on parameter, attributes are updated and method calls made to fetching relevant information.

3. Results

An input/output table has been used to test both applications, the mirror application has no input relying on the configuration algorithm, therefore, this test will identify errors in both applications (Table 1).

Table 2. Results of the testing, where trials show that the applications can handle boundary, erroneous and normal input data

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 performed. This tested the hardware and the application's usability, the device and application stayed on for 24 hours without major errors, 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. A graph displaying the device's power consumption

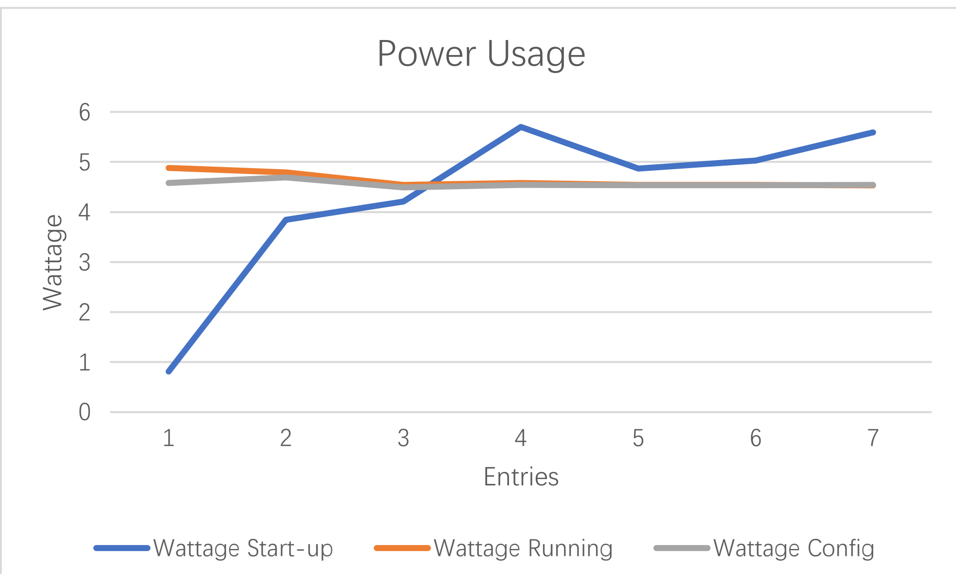
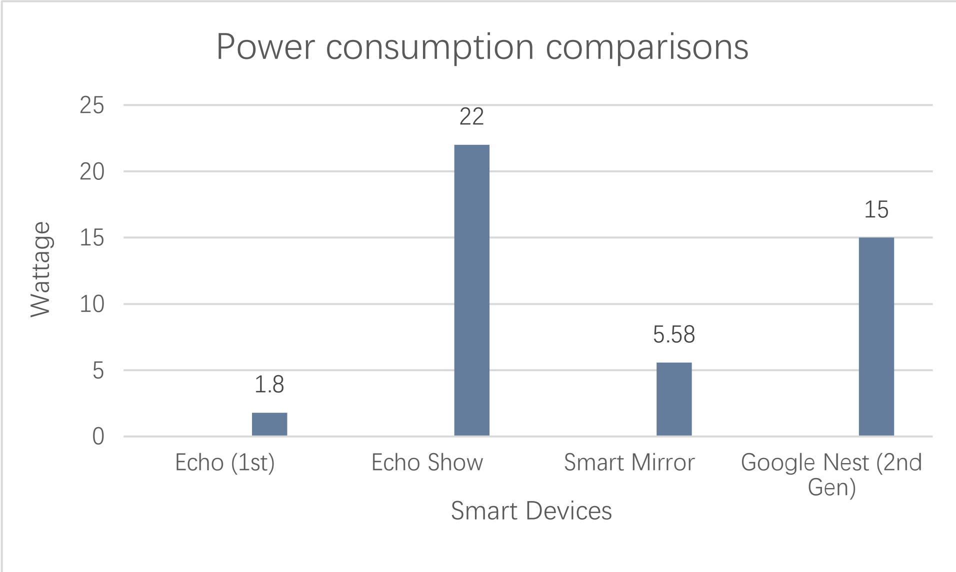
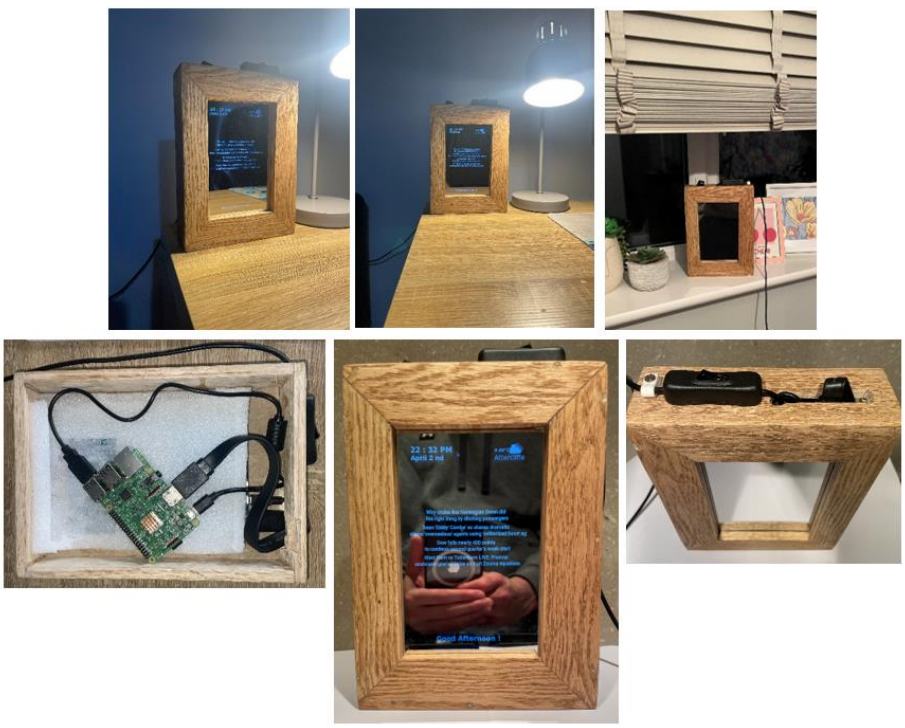


Figure 10. A graph displaying average power consumption for differing devices



A comparison vs the power consumption of other devices is also reported in Figure 10[2]. As shown in the Figures 9 and 10, the proposed smart device uses minimal power having a maximum of 5.58 W during the initial start-up of the system. Compared with similar products, the device outperforms them significantly with the only exception being the Amazon Echo. This device offers less functionality 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.

Figure 11. The Smart Mirror final iteration



[2] Power consumption reported on data sheet of the producst at th efolowing 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

4. Conclusion

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

- **User data is secure -** User data is kept secure, sensitive information is not recorded, or sent anywhere else other than the mirror via a secure 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, achieved by the various methods used to update the interface's widgets.
- **The user can configure where and what data is displayed -** Using the configuration application, widget location and widget 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 universal standards.
- **There is a range 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 differing windows devices via a Bluetooth connection.

In summary, the project has highlighted the deficits within home technologies and the benefits of innovation within this sector. This design can be made quickly at low cost, customised to user requirements, and used in a range of applications. The device can display relevant up-to-date information in an easily understandable format, that can be configured to user needs.

Clearly, a set of further work can be foreseen: the smart mirror proposed in this project can be easily improved, the software is modular, and designed to allow perfective maintenance. The next step for this device would be to gain further user feedback to add additional widget support and make interface changes, as well as to consider integrating machine learning and assistive technologies to further provide service and support to the 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, while Artificial Intelligence and Augmented reality technologies could be implemented using a camera. More importantly further support should be added for differing devices allowing for greater accessibility when configuring the device, the configuration application could be moved to the web using 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, JL and ES; methodology, JL; software, JL; validation, JL; supervision, AO and ES; writing—original draft preparation, JL and ES.

Funding

This work received no external funding.

Acknowledgements

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. et al. (2023) The digital harms of Smart Home Devices: A systematic literature review. Computers in Human Behavior, 145, pp.1–3.
2. Moris, M. E. et al. (2013) Smart-home technologies to assist older people to live well at home. Journal of Aging Science, 01(01).
3. Kulovic, S. and Ramic-Brkic, B. (2018) DIY smart mirror. Lecture Notes in Networks and Systems, pp.329–336.
4. Tewari, D., Jangra, H. and Mutreja, S. (2021) Smart Display Market Size, Share, Competitive Landscape and Trend Analysis Report by Type, Resolution, Display Size and End User : Global Opportunity Analysis and Industry Forecast, 2021-2028. rep. Display Technologies, pp. 0–332.
5. Guhr, N. et al. (2020) Privacy concerns in the smart home context. SN Applied Sciences, 2(2).
6. The Python Language Reference (2024) The python language reference, Python documentation [online]. Available from: <https://docs.python.org/3/reference/index.html> [accessed 7 April 2024].
7. Scanlan, D. A. (1989) Structured flowcharts outperform pseudocode: An experimental comparison. IEEE Software, 6(5), pp.28–36.
8. Mahmmod, B. M. et al. (2023) Performance enhancement of high order Hahn polynomials using multithreading. PLOS ONE, 18(10).
9. Gamess, E. and Hernandez, S. (2022) Performance evaluation of different Raspberry Pi models for a broad spectrum of interests. International Journal of Advanced Computer Science and Applications, 13(2), pp.819–828.
10. Brown, R., Webber, C. and Koomey, J. G. (2002) Status and future directions of the Energy Star Program. *Energy*, 27(5), pp.505–520.
11. Gupta, K. and Sharma, T. (2021) Changing trends in computer architecture : A comprehensive analysis of ARM and x86 processors. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pp.619–631.
12. Tullis, T. S., Boynton, J. L. and Hersh, H. (1995) Readability of fonts in the windows environment. Conference companion on Human factors in computing systems - CHI ’95, pp.127–128.
13. Raskin, J. (1994) Viewpoint: Intuitive equals familiar. Communications of the ACM, 37(9), pp.17–18.
14. M Innes, EL Secco, An Understanding of How Technology Can Assist in the Epidemic of Medicine Nonadherence with the Development of a Medicine Dispenser, European Journal of Applied Sciences, 11(3), 522-550, 2023, DOI: 10.14738/aivp.113.14878
15. VD Manolescu, EL Secco, Design of an Assistive Low-Cost 6 d.o.f. Robotic Arm with Gripper, 7th International Congress on Information and Communication Technology (ICICT 2022), Lecture Notes in Networks and Systems (ISSN: 2367-3370), 1, 39-56, 2022, DOI: 10.1007/978-981-19-1607-6
16. M Van Eker, EL Secco, Development of a low-cost portable device for the monitoring of air pollution, Acta Scientific Computer Sciences, 5(1), 2023
17. K Brown, EL Secco, AK Nagar, A Low-Cost Portable Health Platform for the Monitoring of Human Physiological Signals, The 1st EAI International Conference on Technology, Innovation, Entrepreneurship and Education, 2017, DOI 978-3-030-02242-6_16
18. D McHugh, N Buckley, EL Secco, A low-cost visual sensor for gesture recognition via AI CNNS, Intelligent Systems Conference (IntelliSys) 2020, Amsterdam, The Netherlands
19. Buckley N, Sherrett L, Secco EL, A CNN sign language recognition system with single & double-handed gestures, IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), 1250-1253, 2021 - 10.1109/COMPSAC51774.2021.00173