DOI. 10.1109/SACI55618.2022.9919550

Energy Consumption Analysis for Raspberry Pi Based Smart Mirrors and Proposed Solutions

Alexandru Rat
Faculty of Automation and Computers
Politechnica University of Timisoara
Timisoara, Romania
alexandru.rat2@student.upt.ro

Cristina Stangaciu
Computers and Information Technology Department
Politechnica University of Timisoara
Timisoara, Romania
cristina.stangaciu@cs.upt.ro

Mihai Micea
Computers and Information Technology Department
Politechnica University of Timisoara
Timisoara, Romania
mihai.micea@cs.upt.ro

***Abstract*—This article presents an analysis regarding energy consumption of a type of smart devices, namely smart mirrors using a reference implementation based on Raspberry Pi and providing features like Internet connectivity, access to different applications, voice command and geolocation. This paper includes a comparative study between different software implementations and configurations proposed for the smart mirror.**

***Index Terms*—IoT device, embedded system, energy consumption**

I. Introduction

In the era of technology and information, the Internet of Things (IoT) plays a huge role in the evolution of applications from various high demand fields like 5G, electrical industry, smart cities, healthcare or even education [1]. Smart sensors communicate wirelessly and via the internet without needed human intervention to serve automated intelligent applications. Home and industrial automation, healthcare, transportation and logistics are just a few of the domains that benefit from the existence of the interconnected devices. Our homes can be called smart when they have a range of smart devices that can be controlled remotely or work independently for automating house maintenance. The system becomes more complex when IoT is combined with those devices. The internet connectivity widens the possibilities of development and communication. Thus, the standard of living of the user is improved with the help of interacting computing systems and embedded systems [2].

Such is the case of smart mirrors. Every day we spend time in front of a mirror, so a mirror that can respond to our need for information can thrill anyone. [3] A normal mirror is improved into a smart mirror and can facilitate the awareness of the user towards various functions.

On the other hand, along with the start of the 2021 energy crisis, the problem of building energy efficient devices was given a higher importance [4], even though the endeavors for reducing energy consumption in smart homes are not new [5].

Now that the climatic changes start to become more apparent, people are starting to be more aware of buying energy efficient products. When choosing a product, the end user is also looking at reducing the long term cost alongside the initial cost of purchase.

Such is the case in the IoT domain. The choice of a user regarding the equipped features is influenced by the power consumption of those features. In the case of a smart mirror, users want to be able to control the IoT device remotely and without a remote controller. One obvious solution is using what nature gave us, our voice [6] [7].

The implementation of voice recognition comes at a price. This translates into increased demand for hardware and computational power, which further results in increased power consumption.

This research focuses on the analysis of power usage of a smart mirror when having implemented features that require speech recognition against the ones that do not have such features.

There is a need for a better understanding of what power consumption impact, the capability of speech recognition has on IoT devices. The conclusion of this research should mirror the balancing between the benefits added by speech recognition and the downsides of increased power consumption.

II. Related work

When referring to a smart mirror that belongs to the IoT domain, we think of a hardware-software bind that offers various functionalities, comes in different designs and is based on several technologies. The most frequently found implementation is done using a Raspberry Pi microcontroller and a LCD/LED monitor behind a two-way acrylic mirror. The purpose differs from mirror to mirror: the software can fetch information such as weather, time, date and location, process this information and format it in a user-friendly manner to display it on the monitor [8] or it can aim for greater scopes, such as medical or academic studies [9].

Some smart mirrors work in two modes: a normal mirror, which keeps the smart system into a standby mode, thus, when not used, it can conserve resources and energy; while in active mode, the mirror is updating and displaying relevant data to the user [10].

The functionalities of a smart mirror are defined by their capacity to interact with the environment around them, either by direct or indirect connection. The direct connection refers to a hard-wired connection through which the system can collect data or operate some actuators: microphone for voice instructions, humidity sensor for measuring humidity, IR receivers for short distance remote controlling. The indirect connection refers to an internet connection through which communication to the outside is done: most smart mirrors have a webpage approach based on different APIs used to fetch data [8]. Some types of smart mirror also have home assistance functions, being input controlled [9].

A study done for the IoT based smart mirrors [8] divided the smart mirrors into 5 major categories based on the field that they are used: general, medical, fashion, academic and sports. Each field has its own characteristics: energy-saving functionalities, facial and mood recognition, augmented reality etc.

Regarding the implementation of smart mirrors, a significant number of them are Raspberry Pi based systems [2], thus a study on this type of architecture seems to be highly relevant.

III. Proposed solution

The proposed solution for this research project is based on the development of a smart mirror device capable of the following tasks:

- show current date and time
- connect to the internet to a IP geolocation provider for retrieving the current location of the mirror
- based on the retrieved current location, retrieve the current weather status and weather forecast and display it in an intelligible manner
- provide the capability of voice recognition to support certain type of request (“tell time”, “tell date”, “tell weather”)

The interface is meant to be easy to understand and make its usage straightforward. It should be out-of-the box available and plug-and-play as in figure 1.

From the hardware point of view, the system runs on a Raspberry Pi 3B+ development board because it has a variety of advantages. The choice was made based on the state of the art and its capability to integrate all the requirements for the project.

First, it had to offer the processing power directly proportional with the needed specifications. It has to withstand consistent, close to real time, voice recognition along with constant weather status update.

Second, since approximately half of the functionality is to display an eye pleasing interface and most of the mirrors come in larger dimensions, it had to support larger resolution output.

Fig. 1. Smart mirror interface.

Thus, a HDMI port became a requirement. More than that, the interface had to be developed on a modern operating system compatible with the development board, either on Windows 10 IoT (here the programming language is C#, based on .Net Core) or on Raspberry Pi OS (former Raspbian with the main programming language being Python, or any other compatible Linux based OS). The decisive characteristic for the final choice of the OS was the already included voice recognition library integrated in the core of Windows.

IV. Experimental results

While the Raspberry Pi 3B+ can be powered via a MicroUSB cable, for the power consumption test the board has to be powered via an external power source, so that an ampere meter can be connected in series to measure the drawn current. The ampere meter is a KEITHLEY DMM7510 which is a high-resolution 7 digit precision graphical sampling digital multimeter. It collects samples at a very high sample rate (40Hz) and creates a CSV file that is later processed and interpreted to draw conclusions. The samples contain data regarding power consumption over time.

The +5V of the power source is connected to a 5V pin of the Raspberry Pi through the KEITHLEY ampere meter to measure the current flow. The GND pin from the power source is connected to a GND pin from the Raspberry Pi. The schematic from figure 2 and figure 3 shows how the connections are done.

After the system is completely assembled from hardware and software point of view, it is time for measurements to be done. There are 2 large test cases to be done that can be split into 4 smaller ones:

- original system runs on Windows 10 IoT and voice recognition feature is disabled
- original system runs on Windows 10 IoT and voice recognition feature is enabled

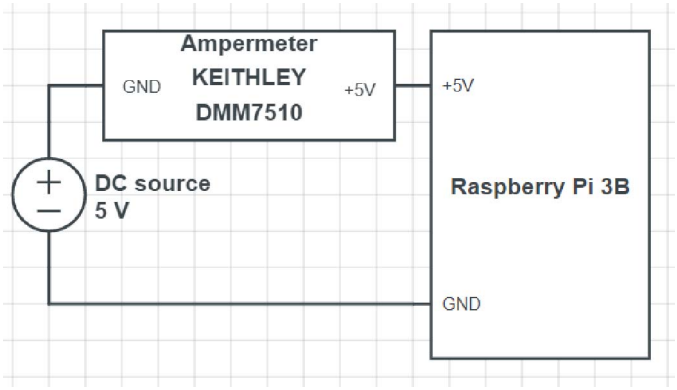

Fig. 2. Hardware schema for measuring power consumption.

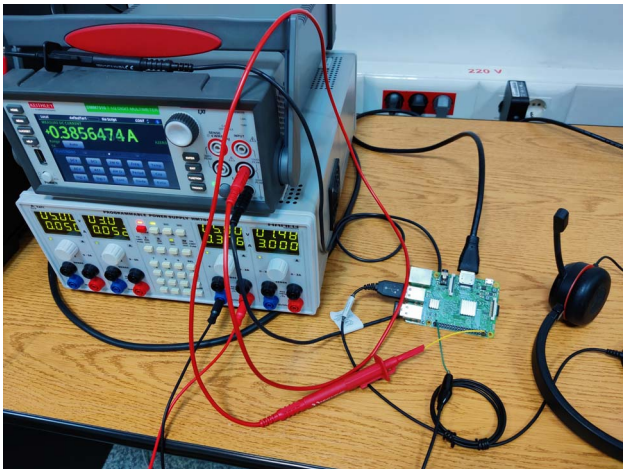

Fig. 3. Hardware photo for measuring power consumption.

- similar application runs on Raspberry Pi OS with voice recognition enabled
- similar application runs on Raspberry Pi OS with voice recognition disabeld

Looking at the experiment as a whole, it diverges into two systems: one running on a Linux based OS and the other on a OS offered by Microsoft. Each one has its advantages and disadvantages, aspects which are analyzed through the following paragraphs, from both hardware and software point of view individually.

Going to the software part of the system and considering the operating system on which the interface is running, we can distinguish two very popular choices for a Raspberry Pi development board: Windows 10 IoT and Raspberry Pi OS. The first one offered by Microsoft includes a voice recognition library which represents the main feature upon which this research diverges around. The interface was developed using C# for both frontend and backend. For the backend functional part of the project C# .Net Core 2.0 programming language was used. The main objective of the interface is to gather useful daily information from different vendors and display it in a user friendly manner. The frontend part was developed using XAML (Extensible Application Markup Language), based on the declarative format XML. To make accessing this information easier, the feature of voice recognition was added to recognize different voice commands and execute certain actions based on the recognized commands.

Regarding the voice recognition feature, Windows 10 IoT has an advantage over the Raspberry Pi OS because the voice recognition library is already included into the operating system, while for the later one it is not the case. In order to implement the voice recognition feature on the similar interface running on Raspberry Pi OS, an intermediate library had to be used. The SpeechRecognition library is a swiss army knife of voice recognition for Python based applications. It supports (as of now, but not limited to) CMU Sphinx, Google Speech Recognition, Microsoft Bing Voice Recognition, IBM Speech to Text. CMU Sphinx works offline, but has the disadvantage that its recognition accuracy is very low. Google Speech Recognition for online recognition has far higher accuracy but at the cost of constant required internet connection.

Back to the hardware point of view, we are going to focus on the power consumption in the 2 large scenarios. We make our analysis based on power consumption graphs with data recorded with the KEITHLEY advanced ampere meter, from which we should draw conclusions. As an input source, a voice recording with several commands was played back in loop for 5 minutes to provide the same testing conditions, allowing 25 repetitions. The microphone placement, playback volume and room background noise was also kept the same during the testing of scenarios. There are 4 actual scenarios to be tested as follows.

Considering the fact that there is no humanly visible variation in spikes height from the graph and in order to keep the power consumption graphics inteligible, they will only represent a one minute time interval capture. However, the average power consumption is calculated on the entire five minute time interval capture.

A. *Windows 10 IoT without voice recognition*

We first take a look at the power consumption graph from the system running on Windows 10 IoT, but without having the voice recognition feature. On figure 4 we can see that the power usage over time is constant, with small spikes specific to the period of data refresh. The important thing to notice is the average power usage that stays below 0.4 amps almost all the time (avg. 0.382 amps), ranging from around 0.36 amps to around 0.39 amps, with only the spikes that rise above for a very short period of time to maximum upwards variation of 0.528 amps. We set the 0.4 amps as a reference value in order to make the comparison between those scenarios easier.

B. *Windows 10 IoT with voice recognition*

Secondly we move on with the analysis to the scenario same as previous, but here having the voice recognition feature. By looking in comparison with the previous scenario, we can see that the average power consumption of the system has gone up by quite a bit, now ranging from around 0.38 amps to around 0.42 amps. This can be seen as 0.02 amps

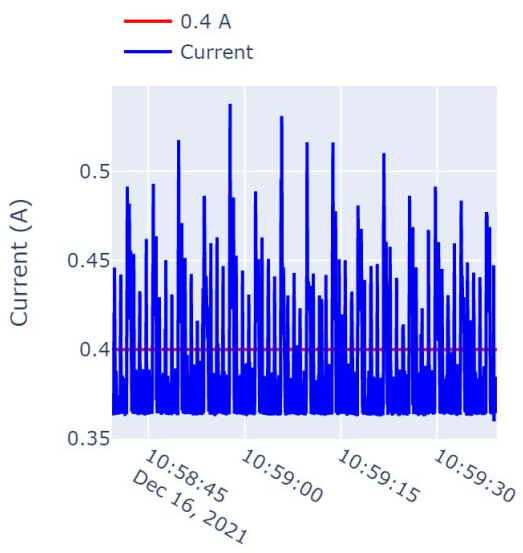

Fig. 4. Windows 10 IoT without voice recognition power consumption graph.

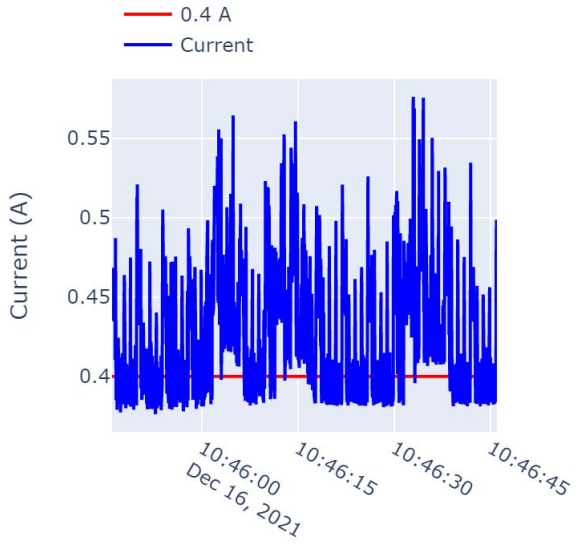

Fig. 5. Windows 10 IoT with voice recognition power consumption graph.

consumed more on average or as an increase of 5% just while idling. We can conclude that this increased idling power consumption is caused by the constant audio input analysis. The voice recognizer constantly listens for audio input. When input volume reaches above a certain threshold, the recognizer starts recording until input volume gets below the threshold for a longer period of time. After that the recorded audio is interpreted into text. If we take a closer look at the graph from 5 we can notice 4 periods of time with visible power consumption increase. Those 4 periods correspond with the moments when audio input was recorded and decoded into text. During those recording and decoding periods the increased processing power required translates into increased power consumption with calculated average being 0.421 amps with maximum upwards variation up to 0.532 amps. (See figure 5).

As a conclusion, there is a clear increase seen in power consumption when having the voice recognition feature implemented versus not having it. The exact increase percentage cannot be determined, as it is directly proportional to the amount of sound over time that exceeds the start threshold of the recording. In other words, the more noise there is in the room where the system is placed, the higher the occurrence rate of the recording and decoding is, which means the more power the system consumes.

C. Raspberry Pi OS without voice recognition

Moving on to the scenario where our system runs on Raspberry Pi OS and taking a closer look at the power consumption graph from figure 6, we can see that the current values only rise above 0.4 amps reference mark during the data refresh, same as the interface running on Windows 10 IoT. The calculated average power consumption is 0.375 amps with maximum upwards variation up to 0.457 amps. What is more important to notice are the lower current spikes drawn during the application idling. Here it barely goes above the reference value, while the Windows 10 IoT application has quite frequent spikes, some of them even rising above 0.55 amps. This might seem that it does impact the general power consumption by raising the average current drawn by quite a bit for the Windows system, but the reality is that those are happening for a very short period of time, so the impact is not that great.

D. Raspberry Pi OS with voice recognition

Last but not least, the scenario where the application running on Raspberry Pi OS has the voice recognition feature shows some variations from the one that does not have the voice recognition enabled. The recorded average value was 0.379 amps with maximum upwards variation up to 0.551 amps. By looking at the graph from figure 7 in comparison with the one from figure 6 the main differences that are noticeable are the current spikes from the moments the recordings have started and were sent for decoding. The important thing to mention is that those are only happening for a short period of time and do not impact the overall consumption by much. By just comparing the average power consumption over 5 minute for both scenarios running on Raspberry Pi OS, the value goes from 0.375 amps average without the voice recognition feature to 0.379 amps average with the feature.

V. Conclusions and Future Perspectives

After the analysis of the power usage over time of the 4 systems (two with voice recognition feature and the other two without it), there should be an overall power consumption

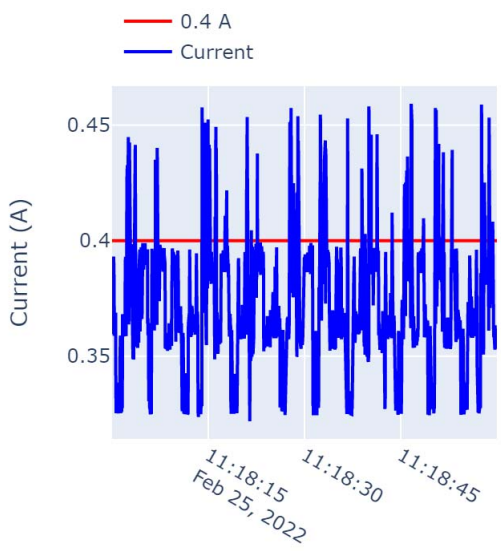

Fig. 6. Raspberry Pi OS without voice recognition power consumption graph.

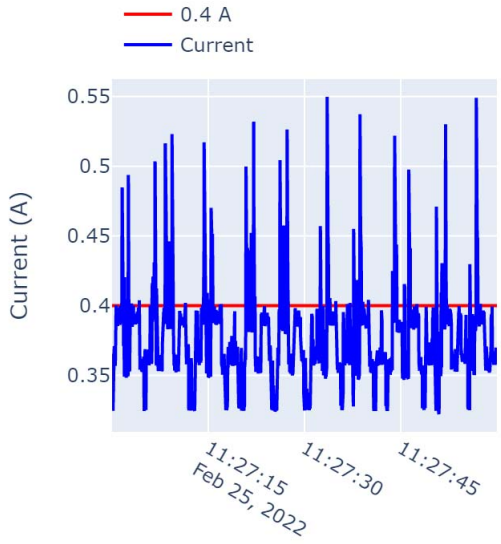

Fig. 7. Raspberry Pi OS with voice recognition power consumption graph.

difference. Variations during the recorded time frames are expected, but are not expected to influence the outcome of the result.

The obvious expected result would be that the power consumption of the system that has the voice recognition integrated is greater than the system that does not have it. The main cause of this result should be the increased processing power needed to continuously record and decode the recording in real time. Increased memory footprint is also expected for the first mentioned system.

Also, comparison with similar application running on Raspberry Pi OS are done from where conclusions regarding the efficiency of the operating system while running the application are to be drawn.

From the analysis done on the 4 scenarios, the conclusion is that the voice recognition feature increases the power consumption over the same systems, but without having this feature. Regarding the interface developed using C# running on Windows 10 IoT, the difference can be seen the easiest. From 0.382 amps to 0.421 amps, there is a increase in power draw by 10.2%. This is a direct result of all the processing involved on decoding the voice audio recorded that is done locally, on the board. The other interface developed using Python and a Python specific external library for voice recognition that runs on Raspberry Pi OS, does not illustrate such a big difference in current drawn because all the resource intensive decoding is happening on a Google server. The most resource intensive tasks happening on this interface are the voice recording, storing and sending to Google for decoding. As a result, there is a small increase in average power consumption, going from 0.375 amps without the voice recognition feature to 0.379 amps for the system with voice recognition. Here the increase in power draw is only 1%, so the conclusions cannot be safely made.

In shorter terms, from the power consumption point of view it is better to do the decoding on the cloud. The main disadvantage is the constant required internet connection, which in this case was a requirement for the interface to be able to fetch data relevant to the user.

There are several future perspectives for this project to be continued. One of them would be an analysis of the power consumption of voice recognition while a large quantity of audio constantly triggers the recording and decoding of audio. For example, a smart mirror similar to the one from this project, but placed into a crowded room with constant noise. Another perspective would be to find other systems, more energy efficent, that can incorporate a set of requirements similar to the ones from this project.

REFERENCES

[1] Bansal, S., & Kumar, D. (2020). IoT ecosystem: A survey on devices, gateways, operating systems, middleware and communication. International Journal of Wireless Information Networks, 27(3), 340-364.

[2] Sahana, S., Shraddha, M., Phalguni, M. P., Shashank, R. K., Aditya, C. R., & Lavanya, M. C. (2021, April). Smart Mirror using Raspberry Pi: A Survey. In 2021 5th International Conference on Computing Methodologies and Communication (ICCMC) (pp. 634-637). IEEE.

[3] Johri, A., Jafri, S., Wahi, R. N., & Pandey, D. (2018, December). Smart mirror: A time-saving and affordable assistant. In 2018 4th International Conference on Computing Communication and Automation (ICCCA) (pp. 1-4). IEEE.

[4] Hamdan, Y. B. (2021). Smart home environment future challenges and issues-a survey. Journal of Electronics, 3(01), 239-246.

[5] Jensen, R. H., Strengers, Y., Kjeldskov, J., Nicholls, L., & Skov, M. B. (2018, April). Designing the desirable smart home: A study of household experiences and energy consumption impacts. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (pp. 1-14).

[6] Venkatraman, S., Overmars, A., & Thong, M. (2021). Smart Home Automation—Use Cases of a Secure and Integrated Voice-Control System. Systems, 9(4), 77.
[7] Ruslan, A. H., Jusoh, A. Z., Asnawi, A. L., Othman, M. R., & Razak, N. A. (2021, May). Development of multilanguage voice control for smart home with IoT. In Journal of Physics: Conference Series (Vol. 1921, No. 1, p. 012069). IOP Publishing.
[8] Maitreyee Vaidya, Shantanu Moraskar, L P Nikhade "SMART MIRROR USING RASPBERRY PI", International Research Journal of Engineering and Technology, IEEE 2019
[9] Muhammad Muizzudeen Yusri, Shahreen Kasim, Rohayant i Hassan, Zubaile Abdullah, Husni Ruslai, Kamaruzzaman Jahidin, et al., "Smart Mirror for Smart Life", International Journal of Advanced Research in Computer and Communication Engineering, 2017
[10] R. K. Meine, "System and method for displaying information on a mirror," May 6 2003. US Patent 6,560,027