Volume 57, Number 2, 2016 ACTA TECHNICA NAPOCENSIS
Electronics and Telecommunications

AN INTELLIGENT EYE-DETECTION BASED, VOCAL E-BOOK READER FOR THE INTERNET OF THINGS

Anca APĂTEAN, Flaviu B. DUNCA
Technical University of Cluj-Napoca, Baritiu st., 26-28, Cluj-Napoca, Romania

Abstract: This paper presents an intelligent vocal e-book reader based on the Raspberry PI, that is remotely controllable through a web interface, being aimed for the Internet of Things. The software was mostly written in Python, but knowledge about HTML/CSS was also used. Images of the user are processed for face and eye detection by classification using LBP image features within OpenCV. To allow the system to accurately function under low light conditions, a light sensor is used in conjunction with a series of leds which are lit up when the lighting in the surrounding room is poor.

Keywords: *Internet of Things, machine learning, classifier, intelligent application, image features extraction, eye detection algorithm, Raspberry PI, web interface, Python programming.*

I. INTRODUCTION

The emerging Internet of Things (IoT) era comprises a multitude of devices developed for smart and innovative applications. Some of them may use machine learning concepts and algorithms, considering their significant contribution to equipping hardware with seeing capability, i.e. vision by computer vision, but also with communication skills, by language processing [1]. The IoT phenomenon promises a world with plenty of applications, e.g. to improve the health and well-being of children and elderly.

The vision is that smart environments and monitoring systems will be created by its large adoption, i.e. at an industrial scale [2]. Moreover, people in the business technology community, recently stated that tens of billions of IoT devices will invade the world by 2020. If this will stand and how intelligent these devices will truly become is only a matter of practice.

This paper briefly describes the implementation of an intelligent system for the IoT. The system is based on the Raspberry PI board, but other SBC (Single Board Computers) may be used. Few questions are foreseen here in order to explain the title of the paper: *Why is this system considered intelligent? Which is the purpose of it being remotely controlled? To whom this system mostly addresses?* Next, we try to give essential answers to these questions.

First, the system is called *intelligent* because it succeeds in detecting the face of a human user and then it establishes if the eyes of the subject are closed or not. Based on this information, it stops the reading process. To accomplish the reading part, it uses a voice synthesis module to produce human-like sounds. Second, it is *remotely controlled* due to the fact that it is desired to be a part of the IoT domain. In order to accomplish this, the Raspberry PI-based application also interacts with a smartphone, tablet, laptop, or any computer system connected to the Internet, with wireless networking capabilities. Such an application *may address to* visually impaired persons or to children, or to any person who would prefer to listen to books rather than read them. Being remotely controlled, it may also help an accompanying person to manage the situation from distance. The application comes with a web interface hosted on the Raspberry PI itself, which is accessible through a secured wireless network, from any authenticated device that has a basic web browser. Once authenticated, this device (e.g. smartphone, tablet, laptop, etc) can interact with the system through the web interface; this, among other facilities later exposed, allows the user to upload e-books (in PDF format). Once the user uploads a valid pdf text document, they can opt for the application to process the document. This will in turn lead to the contents of the document being read out loud, and also the corresponding transcription text being displayed in a designated place in the web interface. Using the web interface, the user can also control some parameters of the application, e.g. the volume of the playback, the language, the reading speed, and among others, the authentication credentials.

In order to make the system more comfortable to use, but also more energy-saving, an automatic power off feature was implemented. This will shut down the system in certain cases but not before it saves the reading progress. Such an operation appears when the system detects that the user has fallen asleep. This functionality is implemented by running a *closed eye detection algorithm* on the frames captured by a webcam connected to the Raspberry PI SBC. To make the detection mechanism even more reliable, thus to ensure that it remains well-functioning under poor lighting conditions, a series of leds are powered on automatically. This smart function is triggered based on the sensed intensity of the ambient lighting, which is registered using a conventional light sensor.

The main purpose of this paper is to briefly present our ABC-PY system, mostly the hardware part of it, as it follows in Section II. Some software aspects are also given in Section III. Section IV presents the main implementation information.

The system we developed is called *ABC-PY.* The first part of the name was so chosen, due to a two-fold interpretation: the fact that it can actually read letters/ text and the fact that basic electronics knowledge is required for constructing it. The last part of the name of the system is so because the software was programmed in Python.

Manuscript received February 19, 2016; revised June 12, 2016

II. HARDWARE ASPECTS

The devices from the IoT area are not only the ones with sensory capabilities, but also those providing actuation capabilities (e.g., bulbs or locks controlled through a network). Specific applications may contain various electronic components such as sensors, displays, resistors, capacitors, diodes, execution elements like motors, etc., so they are inclined to the micro controller applications area. Still, by inserting specific functions, like network connectivity, so they become network or even Internet controllable, they are generally classified as IoT applications. These devices will be proper to collect and exchange data with the world they interact with and even to create big data content.

On the market, there is an extensive number of commercial development boards, ranging from few to tens or even thousands of dollars, according to their capabilities and their brand. Some of them, even quite recent on the device market, are equipped with a lot of facilities as far as their sensors capability, network or Internet communication and SBC functionality are concerned. They allow different types of users, from hobbyists to more experimented ones to use them in the frame of their customizable applications.

II. A. Raspberry PI as a central component

Developed by the non-profit British organization with the same name, the Raspberry Pi is a smart, inexpensive and relatively easy to use mini-computer. It is *smart* because it supports a fully-fledged operating system (O.S.), it can run Python code and it may also run intelligent machine learning algorithms. The operating system is installed on SD or microSD memory cards.

The board can work with inputs/ outputs (via the pins available on the board) or it can manage peripherals (via USB interfaces). Even more, the Raspberry PI *can be used just like a PC*: it is able to process information from a camera / a microphone / sensors (or other inputs), to control output devices, e.g. motors (or other actuators), or it can be connected to a display or to the Internet. Similar products are the Galileo and the Edison boards developed by Intel.

The device we chose, is the Raspberry PI 2 released in early 2015. It has a processor supporting the ARM v7 instruction set, with 4 cores at 900 MHz each, it comes with 1GB of RAM, USB ports, HDMI, an audio socket, and many others.

Compared to a conventional computer, it also offers input/ output programmable pins, appointed as GPIO (General-Purpose Input/ Output) pins; from these, other devices can be supplied power at a voltage of about 3.3 volts [3]. In our project, we also used these pins to power on some leds, depending on the ambient light. It's good to know that, when the GPIO pins are used to supply power to components, it is recommended to limit the electrical current to about 16 mA per pin.

The GPIO pins may be controlled in applications written in Python using the RPi.GPIO library. There are two available modes to setup and identify the pins, the recommended one being the BOARD mode, which corresponds to the physical numbering on the board.

The DIY category of projects abounds nowadays in applications which combine inexpensive electronics with low-cost networked computers. Unlike proprietary commercial products, the projects you build match exactly what you need (or a user specific need). The advantages of DIY projects are multiple: you only have to respect a specific or your own design, and you have the entire control on both the additional hardware equipment and the software to use. Thus, your creativity, economic and nature protective spirit can be encouraged.

II. B. Other existing systems

Nowadays, the use of SBC-like devices is growing, especially in automation systems, for simplifying human life. In the field of document readers, two projects from the American company Dexter Industries, namely BookReader and BookReader version 2 inspired us. Both systems used in addition to a SBC, a popular product of the company, namely BrickPI. This is an Arduino-based platform that enables interaction between the Raspberry PI and various peripherals or shields, so the application may make use of various sensors and actuators [4].

The first version of the BookReader system read pages from a tablet PC. As shown in Figure 1 a), the unit in charge for processing the document is fixed on a support consisting of LEGO bricks. The system uses the Raspberry PI board to which a BrickPI-based page slider is connected. Once the system is functioning, by using a specific Raspberry PI camera module, the tablet is photographed while displaying a page of the document on the screen.

After taking the picture, the image is processed using a piece of software called Tesseract, which handles the optical character recognition task. When the Tesseract's task is finished, the result (which is the content of the current page represented in plain text) is applied to the input of the eSpeak voice/ speech synthesis application. Once the application is run, every word found in the input file will be played aloud at the output of the speakers connected to the Raspberry PI. After processing the full page, the motor driven arm slides on the tablet's screen, changing the page of the document; then the process starts all over again.

Although the software remained basically the same for the second version of the system, in Figure 1 b) one can notice the new mechanism used to interact with the document, which this time is a physical one, i.e. a printed book. The pages of the book first slide using a wheel (that lays on the paper), powered by a servo motor, and then these are turned using a mechanical arm [5], [6].

Consulting the available code on GitHub, we found out that none of the projects had any remote control facility at the time our project was developed. In addition, the systems did not have any mechanism for the detection of the tablet or of the physical book.

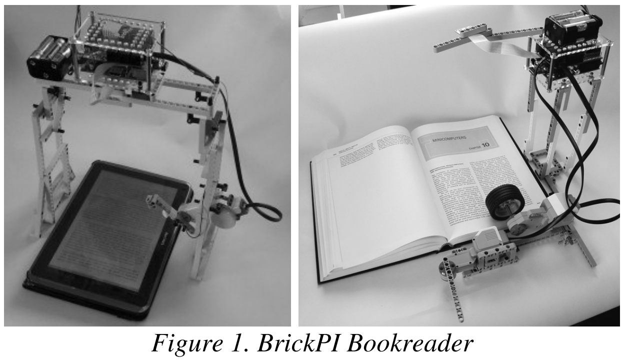


Figure 1. BrickPI Bookreader
a) original version [5]; b) second version [6]

Other problems may have occurred due to differently sized books, which would have probably involved the repositioning of the camera, and possibly changes to the position, the degree and the speed of the motor involved in turning the pages.

II. C. Our ABC-PY system

When implementing our system, we insisted less on the interaction with the document, and more on adding intelligent facilities. One of these is the remote control over a local wireless network.

The system uses the Raspberry PI mini-computer, but it also has a few peripherals, as it can be noticed in Figure 2. From left to right, the components of our system are: 1) a smartphone as the system controlling device, 2) a set of speakers connected via the Jack interface of the mini-computer, 3) a wireless network card with USB interface, 4) a light sensor and a lighting module, connected to the GPIO pins of the mini-computer and 5) a webcam, connected to the system via a USB port (needed in order to detect the face and the eyes of the subject).

A light detection sensor is used and a series of leds will be automatically lit when there is not enough light in the room. To power both the sensor and the leds, the GPIO pins of the board are used. As shown in Figure 3, the sensor has a potentiometer, from which the threshold voltage of the comparator is taken, and a voltage divider consisting of a conventional resistor and a photo-resistor (SEN), whose voltage is connected to the non-inverted input of the comparator. Thus, depending on the threshold voltage the potentiometer represents and on the voltage of the photo resistor [7], at the output of the comparator, the voltage will be 0V if the voltage of the photo resistor does not exceed the threshold; otherwise, the output will have a voltage equal to that of the power supply [8]. Further, based on the output voltage of the sensor, the state of the leds is easily controlled. Figure 4 shows the implementation diagram of the automatic lighting functionality based on the ambient lighting conditions. Knowing that the voltage drops between GPIO pins programmed in logic high mode and the ground pins of the board is approximately 3.3V, the values of the current limiting resistors were found. For the light sensor, it has been decided to limit the current to 5 mA, this being sufficient for the two leds in series with the power supply and the sensor's output to illuminate in order to signal that the sensor functions correctly.

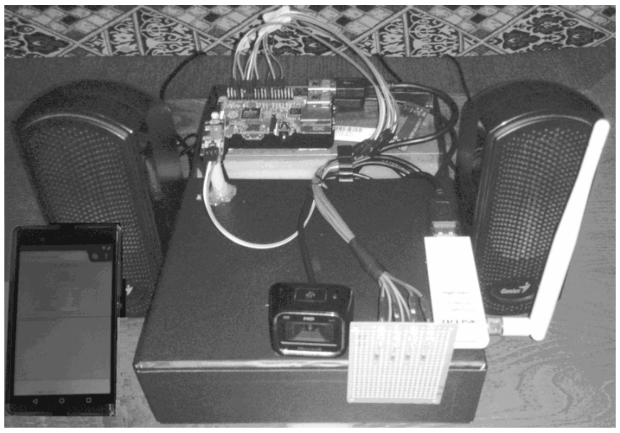


Figure 2. The hardware components of the ABC-PY

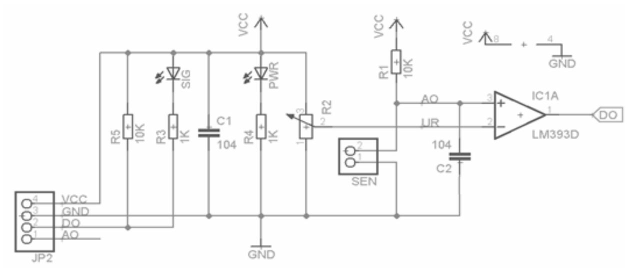


Figure 3. The schematic of the light sensor [7]

A resistance of 680Ω, i.e. R14, was chosen to be placed between the input pin of the sensor and the board's GPIO pin through which the sensor is powered on. Since the leds we used need approximately 3-3.2 volts, the rest of the permitted current was divided between the four groups of two leds in parallel. Each of the groups is powered by dedicated GPIO pins, and the current is limited to approximately 11 mA, through 300Ω resistors, maintaining in this way the limit of about 50 mA recommended for the Raspberry PI SBC.

III. SOFTWARE ASPECTS

The project also comes with a face and eye detection module, that allows the application to power off the whole system if it detects that the user has fallen asleep. For the system to properly function, the user needs to be just a few feet away from the webcam, and they need to pose a frontal view. The detection works best in daytime, but it also works during night-time, due to the leds, that automatically lit up when the ambient lighting is poor, supporting the camera.

III. A. Machine learning aspects

The face and eye detection feature was implemented using the OpenCV computer vision library. In OpenCV, the so called *classifiers* are used in order to be able to detect features in images. These classifiers contain information about the targeted feature, e.g. the human face, ears, hands and so on [9]. For face detection, a classifier that comes with OpenCV was used, i.e. the *lbpcascade_frontalface* classifier. The most popular feature descriptors in OpenCV are Haar and LBP; Haar typically offers lower error rates, but the classifier training process can last days, while with LBP it usually takes hours to train and the results are relatively good [10]; these estimates depend on the number of images used in the training process.

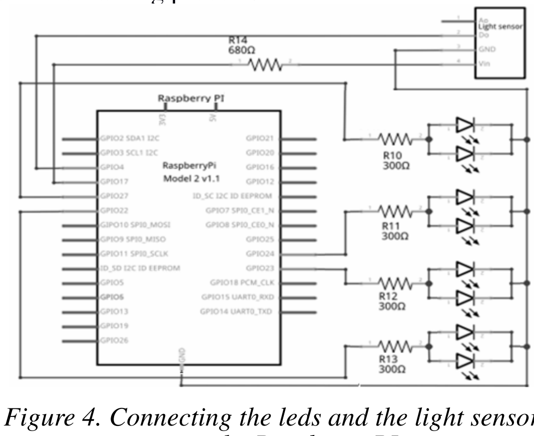


Figure 4. Connecting the leds and the light sensor to the Raspberry PI

The only required condition to succeed in detecting human faces is that the user should be located at a proper distance from the camera and they should be facing it.

The eye detection classifiers within OpenCV were not able to tell the difference between closed and opened eyes; thus, custom classifiers had to be trained. Two classifiers were trained for the closed eye detection, using the *opencv_traincascade* application that is bundled within the OpenCV library.

Each of *the positive image sets* that the training process requires, were made out of about 1100 gray-scale images of left closed eyes for the first set, and right closed eyes for the other. *The negative image set* comprised about 1200 gray-scale images, of human faces, having both eyes opened [11].

When the detection process starts, each frame captured by the system's webcam is first converted to gray-scale. *If features are detected*, their location is returned in the form of a Python list, as four coordinates [11]. In order to reduce false-positive detection results and to increase performance (faster features searching), from the face region, a loosely estimated region is extracted for each of the eyes. On these two regions, using the custom trained classifiers, the closed eye detection is tried. If the detection is successful for at least one eye, a time variable will be incremented.

If *the algorithm fails to detect the features*, it could either mean that the user is not facing the camera or that they have both the eyes opened. In this case, the time variable is cleared and the reading does not yet stop.

After the face and eye detection for the current frame finishes, if the time variable's value is under a threshold, the module will repeat its execution, by processing the next frame; on the contrary, the application will shut down all processes, will save the reading progress and then power off the system.

III. B. Python programming aspects

Due to the generous standard library and its continuous growth, the syntax that requires the production of readable, easy to follow and manage code, Python becomes even more popular for more types of users. Python is usually used in prototyping projects before their implementation in a compiled, thus faster programming language [12], for interconnecting different applications, for programming web applications, but also for programming in the scientific domain, rivaling MATLAB, Mathematica, etc.

Although Python's disadvantage is in its speed of execution, a strong point is the possibility of writing code about 3 times more compact than the equivalent written in Java, or even 15 times more compact compared to the equivalent written in C ++ [12].

III. C. Web programming aspects

In using the Python language, there are many helpful modules, called *web-frameworks*, which simplify the connection with the low-level details, such as HTTP protocol. We used the Flask web-framework to create the logic for the system's web interface.

As presented in more detail in section IV, the web interface is accessible through a password protected wireless access point, which is created every time the system boots. In Figure 5, a basic diagram illustrates how the user can interact with the system.

III. D. Other aspects of the application

For the voice synthesis, the eSpeak software that we used doesn't seem to support PDF files at the input, so the document needs to be first converted to plain text. A *pdf to text conversion* function transforms each page of the document in its own text file, using the free pdftotext command line utility.

Finally, the reading part is handled by a function that synthesizes the text content of each page into speech.

IV. IMPLEMENTATION ASPECTS

IV. A. The main configuration file

The main configuration file of the application contains options and their values, read in most of the program's modules. It holds credentials used to authenticate the controlling device to the system, the name of the currently selected book, but also options which dictate the state of the face and eye detection process, the reading speed, the reading language, and so on.

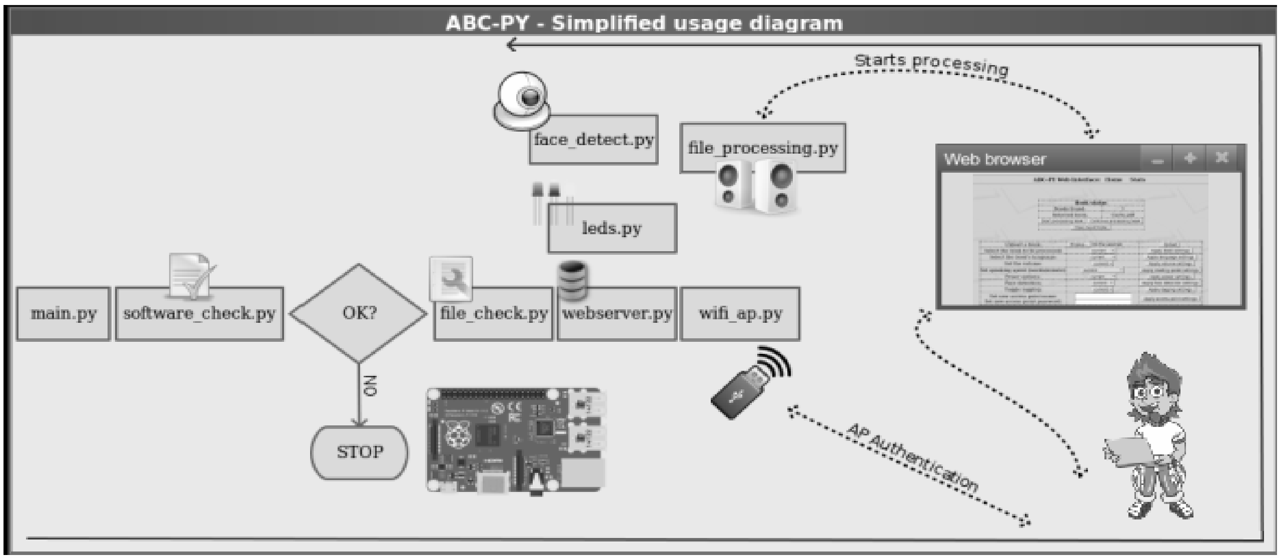


Figure 5. Basic system usage diagram

IV. B. The modules of the system

This part is made out mostly of modules which get called through the web interface, for tasks like starting to process a book, or clearing the books folder, but there are plenty of helper modules, like the configuration module which handles the project's customization, and the *wifi_ap module* which makes possible the connection between the system and the controlling device.

The config.py module is imported in most of the system's modules. It represents a bridge between the project's software and the main configuration file.

In order for the user to be able to communicate with the system, the system and the user's device need to share the same network. This is made possible through the *wifi_ap.py module*, through which, the USB wireless adapter connected to the board is first identified, then gets attributed a static IPv4 address, defined in the application's main configuration file. After that, the identifier of the detected wireless adapter and the authentication credentials are updated in a secondary configuration file. This configuration file contains the parameters used when running the *Hostapd console application*, through which the wireless access point is created. Both the system and the controlling device need to have IP addresses allocated in order to be able to communicate. The system, more specifically the wireless adapter connected to it, gets assigned an IP address, but the device also needs one. This is done using the *Dnsmasq DHCP client*, that automatically assigns IPs to devices as soon as the constantly running Hostapd program authenticates the device in the network.

The *main.py module* is run automatically when the board's (Linux based) O.S. boots, and it prepares the project for the user's interaction.

First, through the *software_check.py module*, it makes sure that all the necessary software the project relies on is installed. The project relies on third-party software like the previously mentioned Hostapd program, but also on Python modules like Flask, that was used to create the web interface's back end. Since different Linux distributions use different package managers and naming conventions for shipping software, on each of the targeted Linux distributions, namely Arch Linux and Debian the existence of the software dependencies was done using code specific to each of them.

As for the Python modules on which the project relies on, their existence is checked by importing them, which is not platform biased. If all the dependencies are satisfied, the *file_check.py module* is run, which assures that the project's configuration files are intact. After that, the *main.py module* calls the *webserver.py module*, which starts the web interface. In the end, the wireless access point is made available by running the *wifi_ap.py module*.

After the user authenticates using the default credentials to the running wireless access point, they will be able to access the web interface through the device's web browser that should be pointed to the IP address allocated by the *wifi_ap.py module* to the board's USB wireless adapter in the initialization sequence started by the *main.py module*.

The *file_processing.py module* is responsible for manipulating the selected book. Every time the web interface is requested, a function searches the books folder, for previously uploaded PDF documents. The interface dynamically updates the selection menu, where all the existing books are presented. The voice synthesis, by the eSpeak process, depending on the user's preference, starts either from the first page, or from the last page the reading process has been previously interrupted. Among the parameters used for executing eSpeak, there is the reading speed and the language of the audio playback, which should match the book's language. These values are read from the project's main configuration file, and they can be customized through the web interface.

The *face_detect.py module* usually runs whenever the user starts processing one of the uploaded books, but it can easily be disabled within the web interface. Inside the module, the webcam is first initialized, and if this succeeds, the face and eye detection classifiers get loaded and frames are captured by the webcam. On the contrary, if the initialization fails, the module will terminate. This module has *two working modes* that can be chosen from the web interface: a *realtime mode*, which allows the module to process the frames as fast as the system allows, and a *delayed mode* which will limit the processing to only one frame for every 5 seconds for performance reasons. In Figure 6, detection results are presented, when running a slightly modified version of the *face_detect.py module*, on a random image [13] in which the person has one eye open and one eye closed, the closed eye being successfully detected.

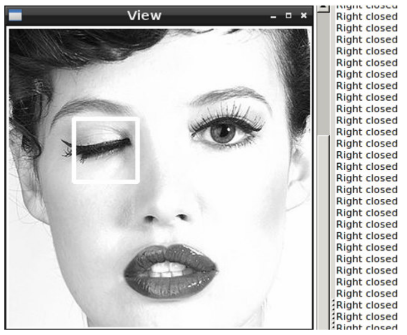
The *leds.py module* handles the leds that run alongside the face and eye detection feature. This module manipulates the light detection sensor, and the leds which will be automatically lit when there's not enough light in the room. The light detection sensor is powered through one of the board's pins and its output that is connected to another pin of the board, gets read. Based on these readings, the pins where the leds are connected, will be or will not be powered.

There is also a module which handles power off and system reboot actions, commanded by the user through the web interface. Before any of these actions, the application's processes that are opened, are safely closed, leading to the reading progress to be saved in the application's configuration file.

IV. C. The web interface

Within the web interface (presented in Figure 7), the user is able to see the contents of the page that is being read at a certain moment, as shown later, in Figure 8. That page is loaded as an HTML file in the interface, through an HTML iframe.

When the reading process is stopped, the application replaces whatever content there might be, with a standard message which marks the purpose of the iframe: "When reading, pages will show up...".



```
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
Right closed
```

Figure 6. Face detection: one closed eye detected

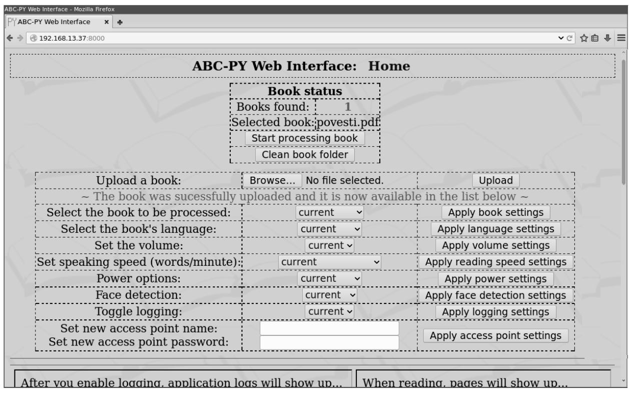
Whenever the user presses the *Clear book folder button*, available in the web interface, every saved book and every file and folder resulted during the books' conversion to plain text is removed.

The back-end of the web interface was written in Python with the help of the Flask web-framework, which makes the connection with lower level technologies, like the HTTP protocol. In this part of the web interface, the HTTP requests are evaluated. There's the regular GET request, made each time the user refreshes the page, and there's the POST request, made every time the user interacts with the interface's options. When POST requests are being made, the button the user presses is identified along with the input data, if any. *The front-end* is made mostly of the *index.html* file, that defines the design of the web interface. The Jinja2 templating language was used along with HTML/CSS in order to be able to provide a dynamic web interface, that changes along with the state of the rest of the application.

The interface, presented in Figure 7 comprises two HTML tables: 1) one for displaying information about the books, like the name of the selected one, the number of the available books, shown in red when there are no books, and in green on the contrary. Aside from these, this is where the buttons responsible for the book's processing are placed. When a book is selected, there's the usual *Start processing book button*, but also the *Continue processing book button*, if a previous interruption for the selected book was detected; 2) another table which hosts the input forms, like the one used for uploading books, the ones used for changing the speed and language parameters of the processing task, the authentication credentials, and so on.

When a reading process is running, some of these buttons will be disabled in order to prevent potential the breakage of the application, especially those buttons that control process parameters such as the reading language or the reading speed, that cannot simply be considered in real time.

Aside from the tables, there are two auto-loading HTML iframes: one of them is used to load the contents of the currently read page, and the other is used to present the user with the system's status logs. The content of both the iframes is refreshed every once in a while, using http-equiv="refresh" meta attribute that's built into HTML.



ABC-PY Web Interface: Home

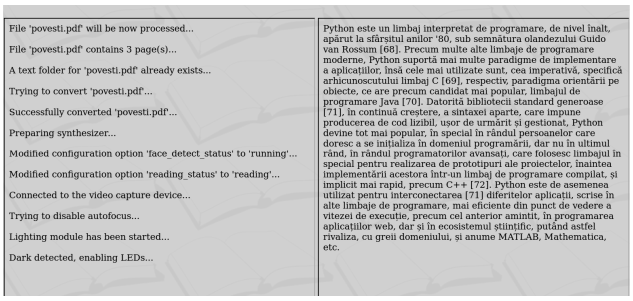
Book status	
Books found:	1
Selected book:	povesti.pdf
Start processing book	
Clean book folder	

Upload a book:	Browse... No file selected.	Upload
~ The book was sucessfully uploaded and it is now available in the list below ~		
Select the book to be processed:	current	Apply book settings
Select the book's language:	current	Apply language settings
Set the volume:	current	Apply volume settings
Set speaking speed (words/minute):	current	Apply reading speed settings
Power options:	current	Apply power settings
Face detection:	current	Apply face detection settings
Toggle logging:	current	Apply logging settings
Set new access point name: Set new access point password:		Apply access point settings

After you enable logging, application logs will show up...

When reading, pages will show up...

Figure 7. The web interface, after a document is loaded



```
File 'povesti.pdf' will be now processed...

File 'povesti.pdf' contains 3 page(s)...

A text folder for 'povesti.pdf' already exists...

Trying to convert 'povesti.pdf'...

Successfully converted 'povesti.pdf'...

Preparing synthesizer...

Modified configuration option 'face_detect_status' to 'running'...

Modified configuration option 'reading_status' to 'reading'...

Connected to the video capture device...

Trying to disable autofocus...

Lighting module has been started...

Dark detected, enabling LEDs...
```

Python este un limbaj interpretat de programare, de nivel înalt, apărut la sfârșitul anilor '80, sub semnătura olandezului Guido van Rossum [68]. Precum multe alte limbaje de programare moderne, Python suportă mai multe paradigme de implementare a aplicațiilor, însă cele mai utilizate sunt, cea imperativă, specifică arhicunoscutului limbaj C [69], respectiv, paradigma orientării pe obiecte, ce are precum candidat mai popular, limbajul de programare Java [70]. Datorită bibliotecii standard generoase [71], în continuă creștere, a sintaxei aparte, care impune producerea de cod lizibil, ușor de urmărit și gestionat, Python devine tot mai popular, în special în rândul persoanelor care doresc a se inițializa în domeniul programării, dar nu în ultimul rând, în rândul programatorilor avansați, care folosesc limbajul în special pentru realizarea de prototipuri ale proiectelor, înaintea implementării acestora într-un limbaj de programare compilat, și implicit mai rapid, precum C++ [72]. Python este de asemenea utilizat pentru interconectarea [71] diferitelor aplicații, scrise în alte limbaje de programare, mai eficiente din punct de vedere a vitezei de execuție, precum cel anterior amintit, în programarea aplicațiilor web, dar și în ecosistemul științific, putând astfel rivaliza, cu greii domeniului, și anume MATLAB, Mathematica, etc.

Figure 8. The web interface, after the reading process is initialized

CONCLUSION

Although for our system's implementation, a Raspberry PI board was chosen, the project should also be compatible with other boards alike, especially with the Banana PI, which was initially the main choice. This type of low-cost platform for real-time intelligent applications generally integrates multiple sensors and is able to adapt it's functioning to the user's need. Such system has to continuously monitor not only the surroundings, but also the user's or system's state. This is generally accomplished with different types of sensors, vision systems, microphones and so on, all needed to improve or generalize the IoT functionality.

REFERENCES

[1] Apatean, A., Rogozan, A., Bensrhair, A., "Image Features Extraction, Selection and Fusion for Computer Vision", in *Image Feature Detectors and Descriptors, Studies in Computational Intelligence*, Springer Switzerland A.I. Awad, M. Hassaballah (eds.), pp. 75-107, 2016

[2] Moscovciuc,M., Apătean,A., "Testing and Developing Intel Galileo Applications for Internet of Things", in *Novice Insights in Electronics, Communications and Information Technology*, Cluj Napoca, 2015

[3] M. Richardson, S. Wallace, *Getting Started with Raspberry PI*, ISBN: 978-1449344214, 176 pages, Dec. 2012.

[4] Dexter Industries, BrickPi, http://www.instructables.com id/BrickPi-Setup/ [Online, Accessed May 2015]

[5] Dexter Industries, BrickPi Bookreader, http://www. dexterindustries .com/BrickPi/projects/brickpi-bookreader/ [Online, Accessed May 2015]

[6] Dexter Industries – BrickPi Bookreader 2, http://www. dexterindustries.com/projects-2/brickpi-bookreader-2/ [Online, May 2015]

[7] CornerstoneRobotics – Cornerstone Electronics Technology and Robotics I Week 15, Voltage Comparators Tutorial, http://cornerstone robotics.org /curriculum/lessons_year1/ER%20Week15%20Compara tors. pdf, [Online, Accessed May 2015]

[8] Electrodragon– *Analog and Digital Sense of Sensors*, http://blog. Electrodragon.com/analog-and-digital-sense-of-sensors-lm393-the-voltage comparator/photocell-sensor-schematic/ [Online, Accessed May 2015]

[9] OpenCV-Python tutorials– *Face Detection using Haar Cascades*, https:// opencv -pythontutroals.readthedocs.org/en/latest/py_tutorials/py_obj detect/ py_face_detection/py_face_detection.html#face-detection, [Online, 2015]

[10] Tam Phuong Cao (editor), *Object recognition*, InTech, ISBN 978-953-307-222-7, 360 pages, 2011.

[11] S. Brahmbhatt, *Practical OpenCV (Technology in Action)*, Apres, 244 pages, ISBN: 1430260793, Nov. 2013.

[12] Python documentation – Comparing Python to Other Languages, https://www.python.org/doc/essays/comparisons/, [Online, May 2015]

[13] Wikimedia Commons – Cecilia Peckaitis, https://commons. wikimedia.org/wiki/File:Cecilia_peckaitis.jpg, [Online, May 2015]