Multimodal Interaction project
Smart Lights

Project realized for the Multimodal Interaction course

Master degree in Computer Scienze – “La Sapienza” University of Rome - Academic Year 2015/16

Made by: Eugenio Nemmi, Carlo Maria Russo, Gabriele Saturni

Version: 1.0 beta

Summary

Introduction

The goal of the project was to realize an intelligent house's lighting system, whose role is to turn on and off all the lights in the house as a function of the human accidental movement and tuned by vocal commands in order to manipulate light's intensity: further more the system is designed to grant the energy saving and to simplify human interaction inside the house.

This project is meant to be used in the home-automation branch and exploits the usage of hardware devices commonly used in the domestic environment.

It is divided into three separate modules that interact each other to perform the final result:

1. Video module: it's used to catch the user's movements inside the house.
2. Audio module: it's used to understand user's requirements during software's execution;
3. Arduino module: it's the hardware part, responsible in performing the actions, analyzing all the input data coming from video and audio modules.

Both video and audio modules are realized with Python 2.7 programming language, instead the Arduino module has been programmed using the Java language: implementation details of these software parts will be discussed later on.

In order to complete the energy saving task, the system was designed to keep track of human accidental movement inside the house, and eventually to switch on only those lights which are closer to the user's position: in this way we decrease the energy consumption caused by lights left switched on in rooms the user is not interested in.

All the vocal commands are implemented such that the user can decide whether to turn on/off the whole system (that would keep running in background) or to adjust the intensity of the lights without the need of interacting with power switches. Moreover all the voice commands are meant to make the system live: this means that the user has the possibility to interact with the lights while he's walking inside the house.

Requirements' analysis

During the realization, we identified the following tasks the system had to perform:

1. User's movement recognition and management inside a room
   1.1 Background inizialization
   1.2 User recognition
2. User's vocal command recognition and management
   2.1 Audio input recognition
   2.2 Keyword recognition
3. Data catch and hardware execution on Arduino side

UML class diagram

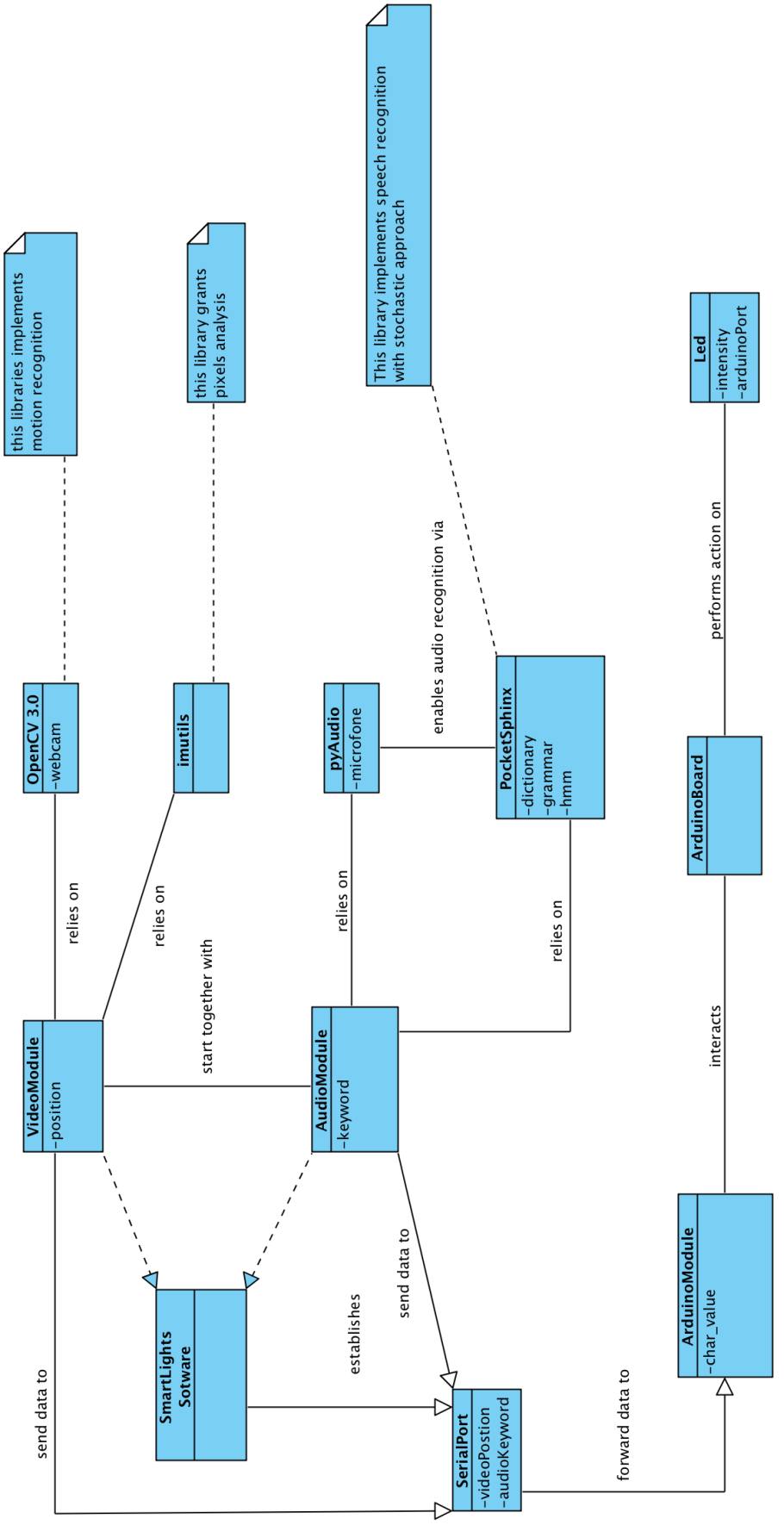


Figure #1 – UML class diagram

Use cases

System's use cases are the following:

1. Video managment: satisfies requirement 1
   1.1. Background management: satisfies requirement 1.1
   1.2. Movement management: satisfies requirement 1.2
2. Audio management: satisfies requirement 2
   2.1. Audio input recognition: satisfies requirement 2.1
   2.2. Keyword management: satisfies requirement 2.2
3. Data management: satisfies requirement 3

UML use cases' diagrams

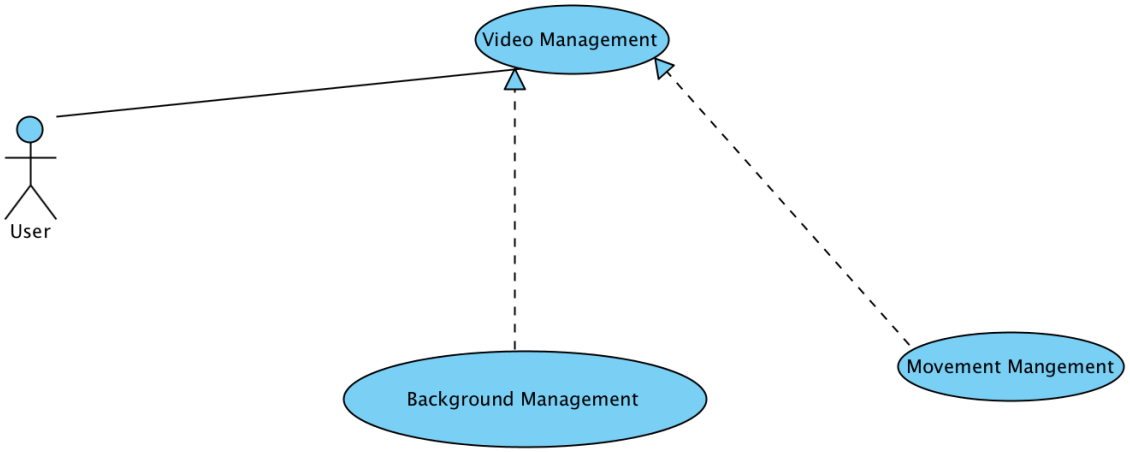


Figure #2 – Video Management UML diagram

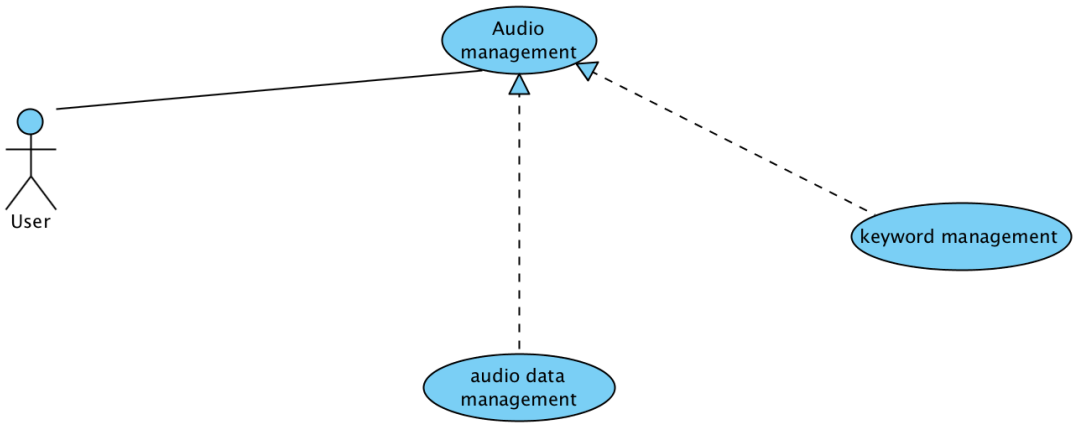


Figure #3 – Audio Management UML diagram

Implementation specifications

We decided to develop the whole project with the Python language, in particular with its 2.7 version, that grants a high level of scalability and provides a large set of native libraries, usefull for software's realization. We decided not to use the C++ language because of its low portability: in fact, what we wanted to achieve was a cross-platform software, runnable on the most used operative systems.
More details reguarding all the libraries and APIs used for the modules' implementation are going to be discussed in the proper modules' sections.

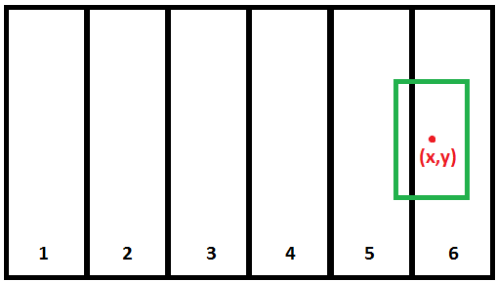
Video Module

The video module is directly responsible for the user's movements recognition and his position computing inside the room. The implementation was made using the OpenCV 3.0 library, implemented with Python 2.7: it includes several hundreds of computer vision algorithms, it features a variety of modules responsible of image processing, geometrical image transformations, video analysis such as motion estimation, background substraction and object tracking algorithms.
The goal of this module is to recognize and to keep track of user's movements via the webcam integrated in the laptop. Since we didn't have any external camera, this first version of the software is realized through the use of the webcam, but the possibility to use an another kind of video source has been implemented as well: future development takes into consideration the fact that CCTV camera will be used instead of a laptop.

Functionalities

Video module task is to satisfy the requisite 1.
The "imutils" library is a key component in this module, since it's directly responsible of the pixel analysis of the image captured by the camera: it does compute and recognize all the variations that use to happen in the image during software's execution. Video module functioning follows beyond.

Once the software is launched, a first background image is computed: this first image will be used to determine the pixel changes in order to recognize the user moving inside the room.
It is very important that there are no external subjects framed during the inizialization phase, or they will be identified as part of the background. To let the user know that system has been started correctly, the video captured by the webcam is directly projected to the laptop's screen: display screen is scalable and suits indipendently on which hardware it's running on.
From now on, the system is capable of recognizing a user simply analyzing pixel variations inside the camera action range.
When some change is detected, software enlights the area of interest by drawing a green bounding rectangle on the screen: user's position is gained by computing the center of this rectangle, which is supposed to be the mass center of the person. In order to avoid missdetections, we decided to fix a threshold based on some empirical experiments such that, if the area of the rectangle computed is not large enough, it will not be considered as a not negligible movement.



To determine user's position in the space we divided the camera screen in six[1] fractions, and we took the bounding rectangle's center as reference: we just analyzed movements along the x-axis in this first version of the software.

The computed position is transmitted to Arduino via serial port, and then it will be its responsiblity to turn on the correct lights in the room.

The functionalities this module implements are:

- Energy consumption reduction;
- Esemplification of the interaction between the user and the domestic environment.

We will now go deeper inside the audio module, analyzing parts of the source code:

- Threshold is set on a value that, as mentioned before, better approximates human size dimensions;

```
ap.add_argument("-a", "--min-area", type=int, default=5000, help="minimum area size")
```

- We both have the possibility to read the video source from the integrated webcam, or a cam connected to the laptop, or even a previously recorded video.

```
# if the video argument is None, then we are reading from webcam
if args.get("video", None) is None:
    camera = cv2.VideoCapture(0)
    time.sleep(0.25)
# otherwise, we are reading from a video file
else:
    camera = cv2.VideoCapture(args["video"])
```

- If there's some pixel change in the background not big enough to represent a human movement, software will ignore it […]

```
if cv2.contourArea(c) < args["min_area"]:
    continue
```

[…] otherwise it will compute the bounding rectangle and draw it on the scene.

```
# compute the bounding box for the contour, draw it on the frame
(x, y, w, h) = cv2.boundingRect(c)
rectangle = cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

[1] We have only six fractions since, when we implemented the software, we had only six leds to set up on the Arduino board

centroRectX = x+(w/2)

- If some not negligible movement is catched, then the actual position will be computed, considering the center of the bounding rectangle, and sent to Arduino. In order to avoid the serial port data overload, we decided to transmit the data to Arduino only when user moves from a sector to another. Further more the "break" command is necessary: in this way software will recognize only one person in the camera area.

```
if rectangle is not None:
    position = computePosition(camera.get(3), centroRectX)
    if position!= temp:
            temp=position
            arduino.write(temp)
    break
```

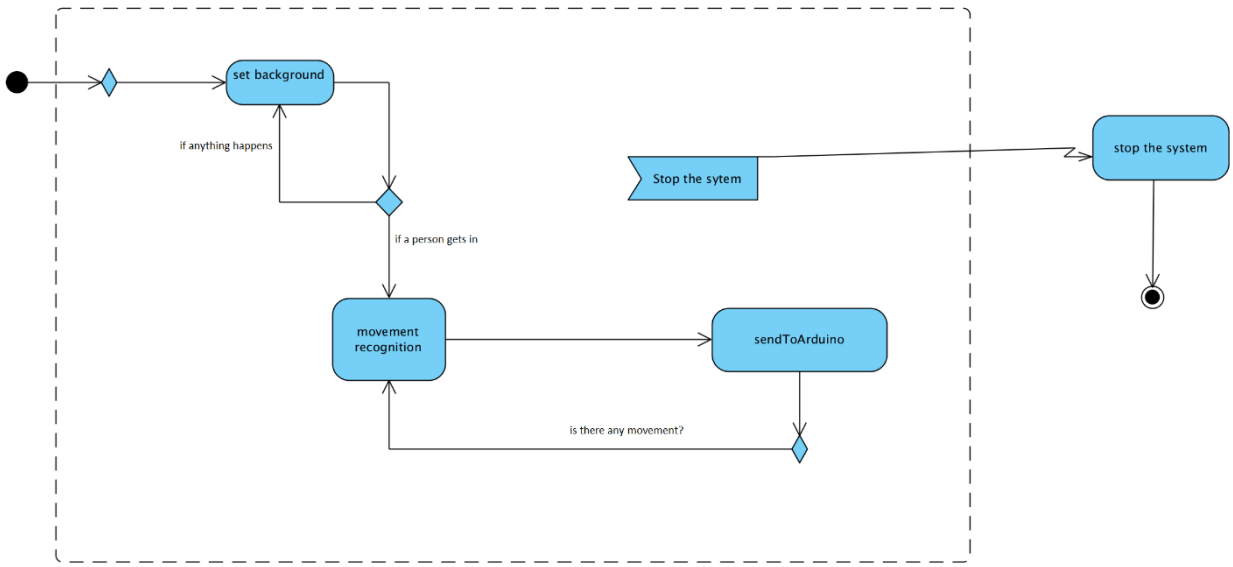


Figure #4 – Activity diagram for video module

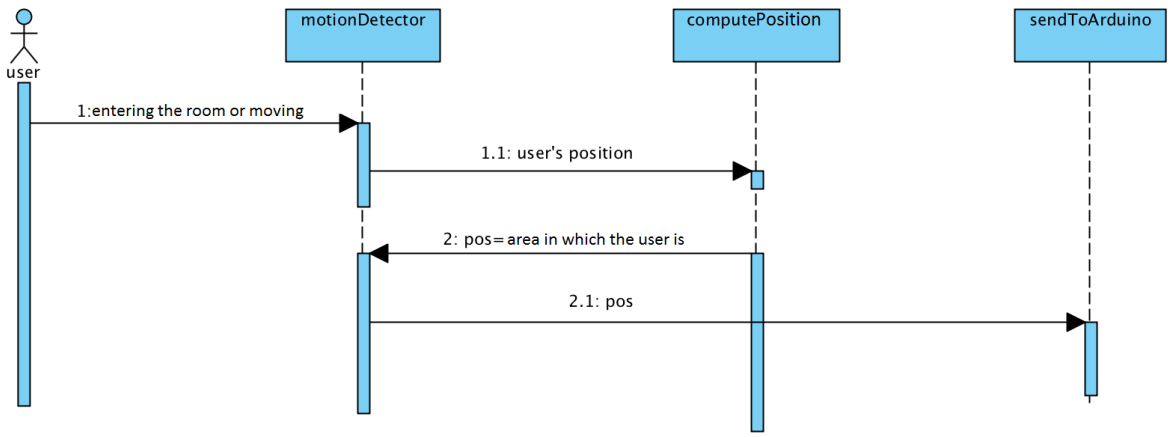


Figure #5 – Sequence diagram for video module

Problems encountered during video development

A big slice of the problems we encountered during the video module development is driven by the difficulty of handling all the changes that may happen in the environment. This module, in fact, can potentially recognize any kind of change: either human's or animal's movements, either lights' variations, and so on. A partial fix was the implementation of the threshold with a value that is approximated to the size of a human being: it is anyway impossible to distinguish what is passing through the camera range during the video elaboration. If an animal passes sufficiently close to the webcam, his movement will be catched and analyzed.

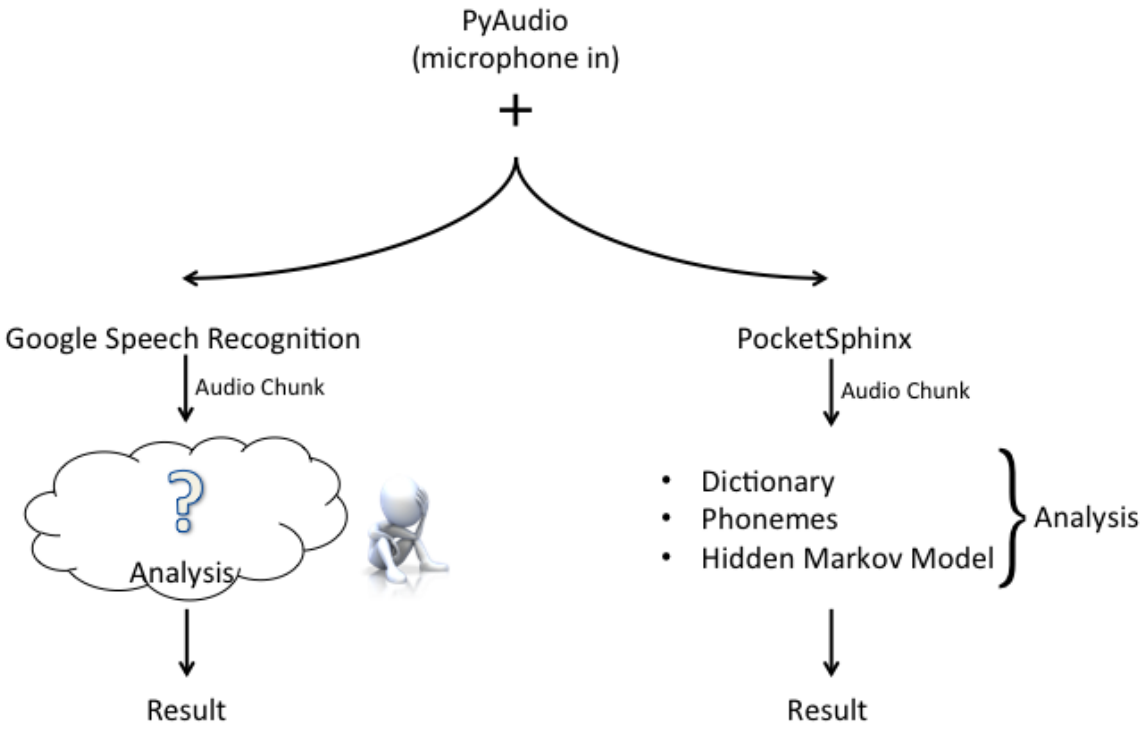
This is mainly the reason why we decided to juxtapose an audio recognition module to the video analysis: in this way the user (a person) is the only one who can consciously enable and disable the lights' system.

Another possible issue is given by the possible conflits during video elaboration, i.e. two or more subjects moving simultaneously in the room: this problem have been fixed forcing the system to follow the movement of the first subject framed.

Audio Module

The audio module, as we already briefly mentioned before, is responsible for the implementation of the user's voice recognition: it's necessary for the system to be started and stopped as the user wishes, and in order to grant the energy saving we're focusing on.

The whole audio module is based, in this first software's version, on the library Pyaudio: this open-source library, which is implemented in Python language, lets the user either play or record audio on a variety of platforms, such as GNU/Linux, Microsoft Windows and MacOSX. In our design application, we're using Pyaudio to collect all the audio portions simply enabling the use of the microphone integrated in the laptop.



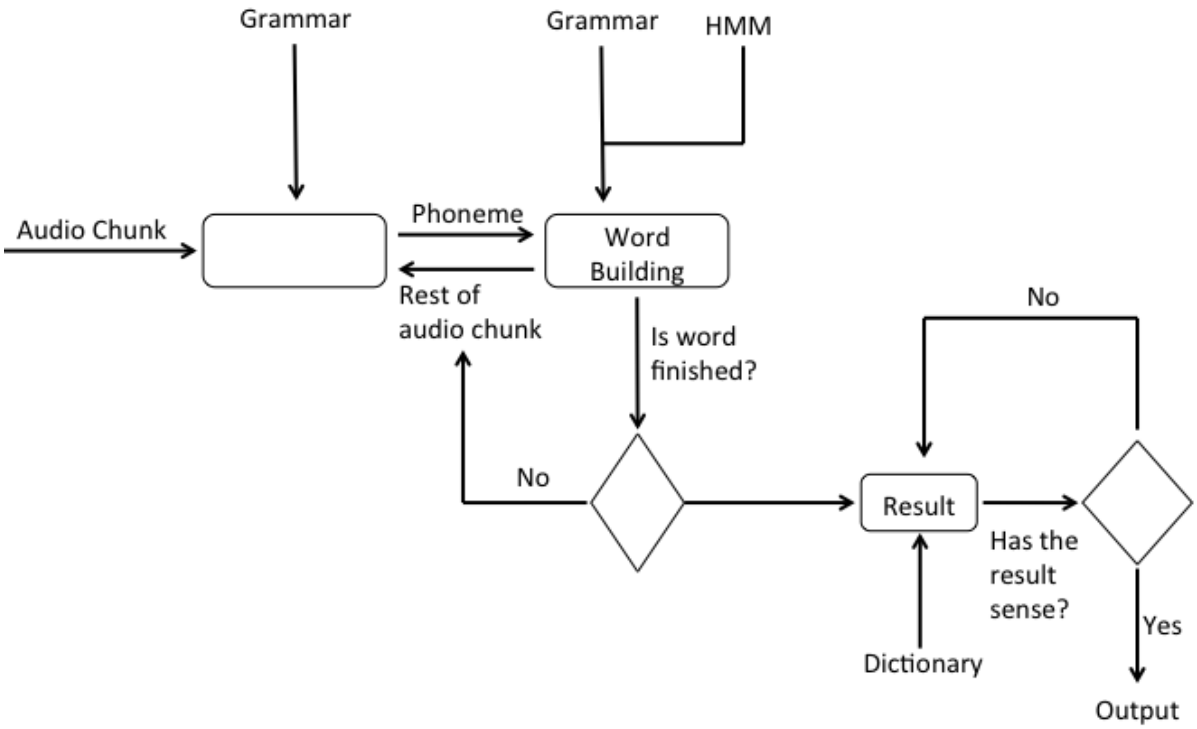
Once we had the possibility to listen to chunks of external audio, we focused on how to analyze those and how to catch some usefull keywords for the application: during the development we encountered two possible and alternative solutions:

1. Google Speech Recognition: it's a simple-designed application that forwards all the listened audio portions to Google, in a way that is completely invisible to the user. This means that the user has no possibility to analyze what happens in between the audio chunk and its recognition, neither he knows how that is performed. The Google Speech Recognition module is really accurate, as soon as the user respects a correct english pronuntiation, but it is, on the other hand, really slow: sending audio portions to Google requires, at first, to be

connected to the internet, and, consequently, some elaboration time due to the upload/download of the chunk and its recognition;

2. Pocketsphinx: it is a lightweight speech recognition engine, specifically tuned for handheld and mobile devices, though it works equally well on computers: it depends on another library, called SphinxBase, and implements a very complete set of modules to ensure the speech recognition. The whole analysis is based on a stochastic approach, and uses:

    a. A dictionary, which contains a large set of words (in our application, it is in english language) and is used to determine the final result of the speech analysis;
    b. A grammar, that describes the way words are built through the usage of phonemes: more specifically it contains the list of all possible phonemes in the english language;
    c. The hidden Markov model, which is used to statistically determine, given a certain phoneme in the grammar and a listened audio chunk, what is coming next in the recognition: more formally, each phoneme rapresents the current state of the analysis and it is associated to a set of transition probabilities to determine what is going to be appended to that particular phoneme during the audio recognition.



The pocketsphinx library is, in some way, the opposite of Google Speech Recognition: this means that it is really fast, since all computations are done on the local machine and without the need of any internet connection but, on the other hand, the recognition is inaccurate:

the fact that pocketsphinx's english dictionary and grammar are very large, requires the system to be trained to understand user's most spoken words.

Since the goal of the application is to create a energy-saving oriented software, we decided to implement the audio module with the Pocketsphinx library, that doesn't require any internet connection and grants a faster reaction time, necessary to reach liveness of the system.

Functionalities

Once the system is initialized and program is launched, the audio module starts running and performs the following operations:

- It creates a decoder meant to analyze the audio portions given some configurations: it is necessary to specify all the dictionary, grammar and hmm models that have to be taken into account. In this first software version, we decided to use the basic modules provided by Pocketsphinx, so the ones proper for the english language, avaliable in the source folder. The practical code example is given beyond, where MODELDIR represents the default directory where pocketsphinx is located:

```
# Create a decoder with certain model
config = Decoder.default_config()
config.set_string('-hmm', os.path.join(MODELDIR, 'en-us/en-us'))
config.set_string('-lm', os.path.join(MODELDIR, 'en-us/en-us.lm.bin'))
config.set_string('-dict', os.path.join(MODELDIR, 'en-us/cmudict-en-us.dict'))
decoder = Decoder(config)
```

- It initializes the microphone, which is the one integrated in the laptop, and opens an input stream with the possibility to set up the audio format, the number of channels, and the audio framerate. This stream is meant to keep continuously listening everything that comes from an external source, reducing the ambient noise at its beginning.

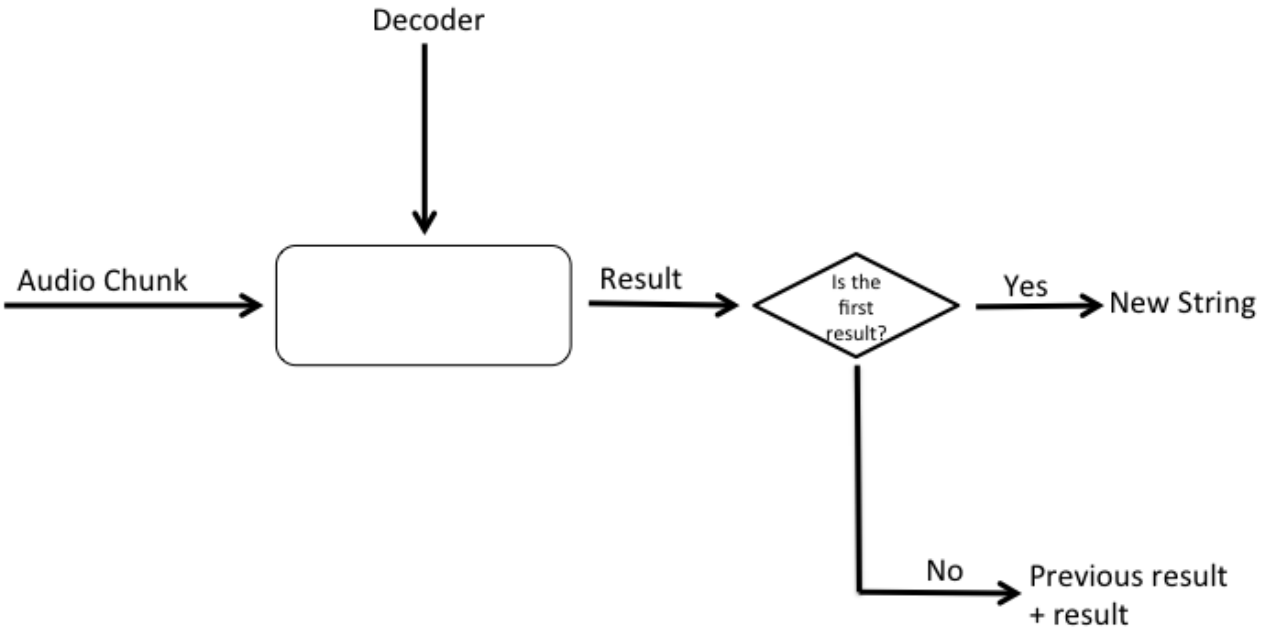
```
# Decode streaming data.
decoder.start_utt()
p = pyaudio.PyAudio()
stream = p.open(format=pyaudio.paInt16, channels=1, rate=16000, input=True,
                frames_per_buffer=1024)
stream.start_stream()
```

From now on, each listened audio chunk is going to be analyzed and the output will be visible to the user: until there's some audio recognition from an external source, the stream and the decoder are respectively going to read and analyze that.

```
while True:
    buf = stream.read(1024)
    decoder.process_raw(buf, False, False)
    if decoder.hyp() != None:
        rec = decoder.hyp().hypstr
        print rec
```

Continuous audio recognition

The in-microphone and the decoder have both been inizialized and the input stream is open. The system has recognized some audio input. The decoder begins processing the chunk and outputs a first result. While there's some external sounds, those will be analyzed by the decoder and the results will be appended to the previous ones.



Keyword detection

The decoder has outputted some results. Software saves decoder's output on a local string, and begins working on it.

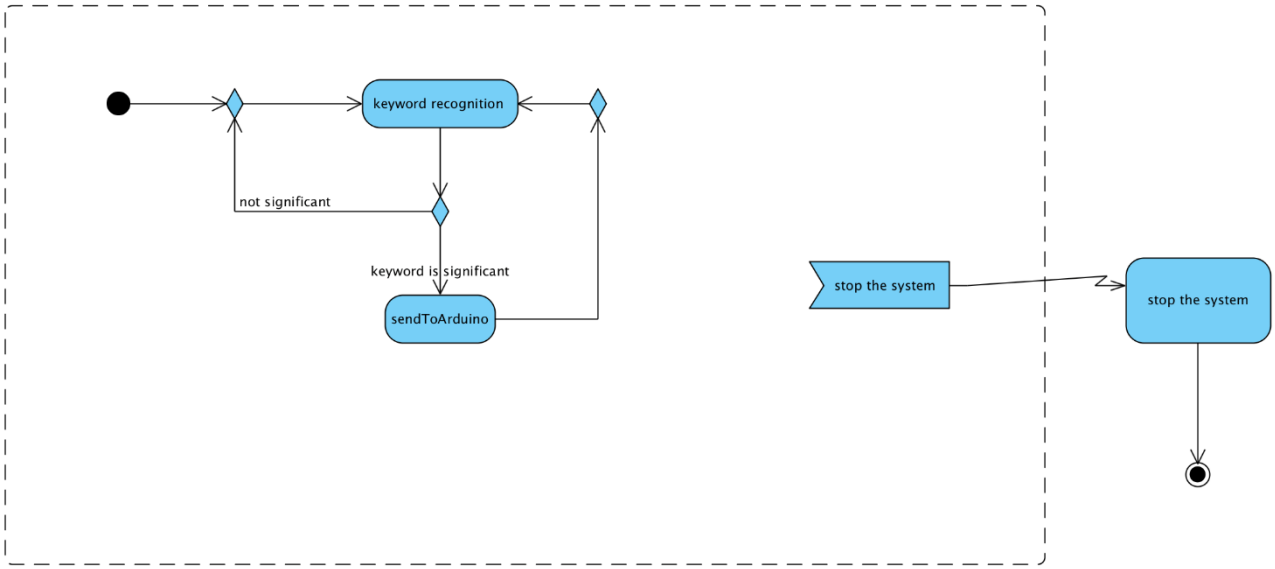


Figure #7 – Activity diagram for audio module's keyword recognition

When the keyword is recognized, the decoder is restarted, in order to avoid the data overload in the software.

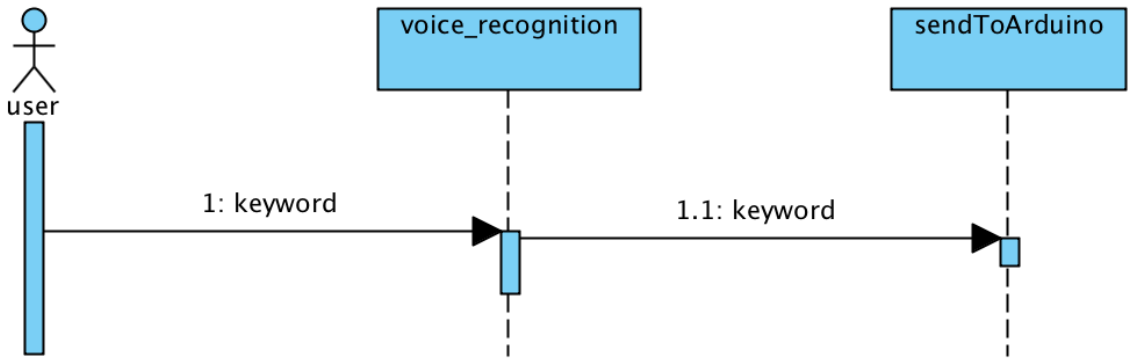


Figure #8 – Sequence diagram for audio module's keyword recognition

As already mentioned before, Pocketsphinx is a system that needs to be trained to perform as well as possible: due to the lack of time and the variety of english pronunciation, we adjusted the list of possible keywords to be recognized on a limited number, relying on some empirical experiments to determine which were the unambiguous ones.

Here's a list of [voice command] → [action] → [character] we used:

- " on", "switch on", "turn on", "hello", "start" → Turn on the lights → "0"
- "raise", "ray", "raze", "race", "plus", "**last**" → Raise the intensity of the lights → "R"
- "minus", "down", "decrease", "**degree**", "lower" → Lower the intensity of the lights → "L"

- “quit”, “**great**”, “stop” → Switch off the system → “Q”

It’s noticeable how the meaning of some words is completely different from the command that we want to be executed: this is because some phonemes can be misunderstood during the audio analysis.

Depending on the chunk recognized, software automatically converts the right command in a character and then forwards it via serial port to Arduino, responsible to perform the desired action on the lights.

Arduino module

Arduino is an open-source computer hardware and software company, project and user community that designs and manufactures microcontroller-based kits for building digital devices and interactive objects that can sense and control objects in the physical world. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. It is possible to tell the board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

We decided to choose Arduino because of its simple and accessible user experience. The basic pros are:

1. Unexpensiveness: you can buy the ArduinoOne board with some starter pieces with less than 30$;
2. Cross-Platform: different versions of Arduino IDE are avaliable for all the most used OSs;
3. Simple and clear programming environment;
4. Open source and extensible software & hardware.

An Arduino board historically consists of an Atmel 8-, 16- or 32-bit AVR microcontroller (although since 2015 other makers microcontrollers have been used) with complementary components that facilitate programming and incorporation into other circuits. An important aspect of the Arduino are its standard connectors that enable users to use a variety of add-on modules known as shields. In this way is incredibly easy to build products of all types.

All the data are managed and then elaborated by the program and a corresponding letter is sent to serial port of Genuino Uno for every event happened. There are several reasons that led us to this choice: first of all, there is a limited buffer size in Arduino, and because we are using serial port, comunication is sent one byte at time.

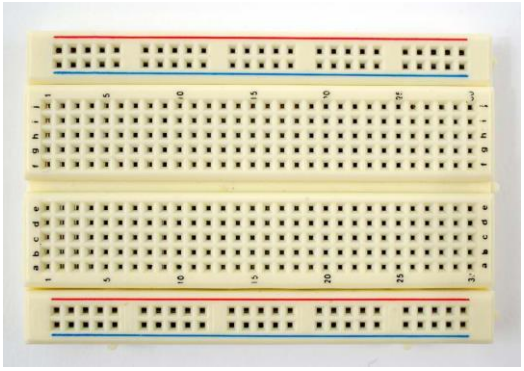
Using single characters extremely quickens the comunication operation; further more, it's a very simple implementation.

Hardware specifications

Hardware system is composed by 3 elements:

1. Arduino GenuinoUno board: the Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button.

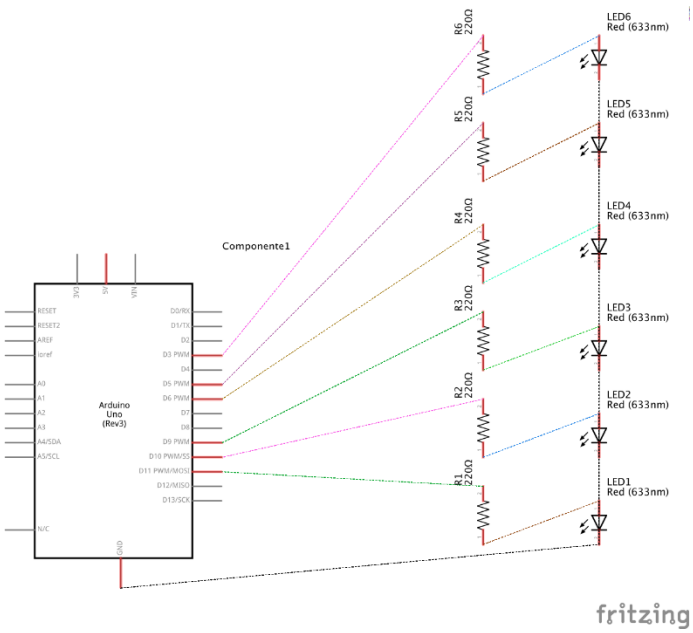
2. BreadBord: a breadboard is a construction base for prototyping of electronics.Because the solderless breadboard does not require soldering, it is reusable: it is easy to use for creating temporary prototypes and the experimenting with circuit design. For this reason, solderless breadboards are also extremely popular with students and in technological education.

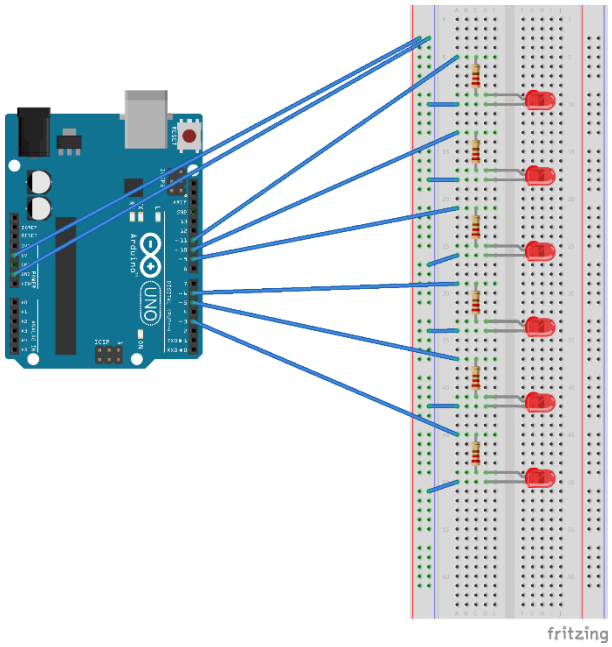


A modern solderless breadboard consists of a perforated block of plastic with numerous tin-plated phosphor bronze or nickel silver alloy spring clips under the perforations. The clips are often called tie points or contact points. The number of tie points is often given in the specification of the breadboard.

3. Six Red Leds: a light-emitting diode (LED) is a two-lead semiconductor light source. It is a p–n junction diode, which emits light when activated. When a suitable voltage is applied to the leads, electrons are able to recombine with electron holes within the device, releasing energy in the form of photons. This effect is called electroluminescence, and the color of the light (corresponding to the energy of the photon) is determined by the energy band gap of the semiconductor.

Circuitation and Arduino schema





Video commands management

Only the coordinate of the object tracked is sent to Arduino. To avoid an overflow of Arduino Serial Buffer, that will cause slowing down of responsiveness of the system, we decided to send coordinates only when camera notice a movement. In our first experiment we observed that this kind of solution results in amazing improvement of performance about real time feedback. All the data are sent using serial port with standard speed of 9600 bits per second (baud). To guarantee the correct user detection, camera is always on, even if the system is temporarly turned off.

In this way, people are always tracked by the system to let it mantain control over their behaviour, ensuring that, when turned on, the system reaches the correct status of lights.

Command received can be:

A. 1st light from left side;
B. 2nd light from left side;
C. 3rd light from left side;
D. 4th light from left side;
E. 5th light from left side;
F. 6th light from left side.

In our prototype, lights consist of six red 5mm basic led, with 2.0v and a max current of 20mA power. Every light simulates the center of a particular range in which the user stands.

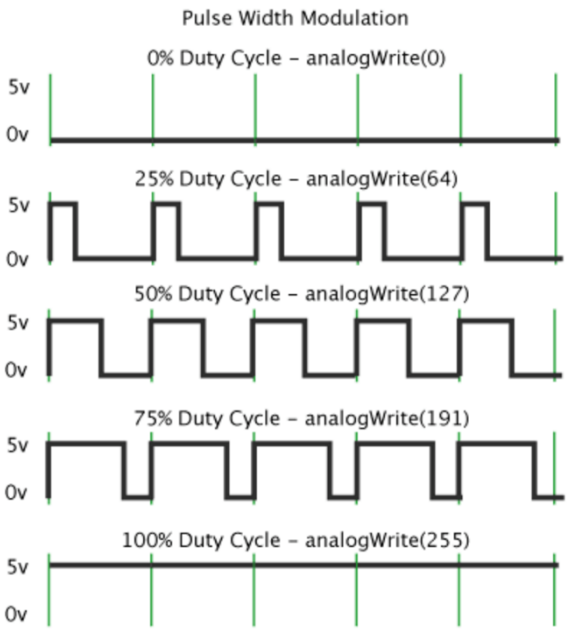
In case of 6 lights, we have 6 different position ranges. Every coordinate the Arduino board receives, falls in one of this range and according to that, the corrisponding lights are turned on. The default setting is to light up the closest space to the user with a standard intensity, and to also light the sorrounding space by 2 sideslights by the center one. The standard system intensity for the lights is:

- Center: 255
- Sides: 63

To modulate the intensity we used Pulse-width modulation**.**

Pulse-width modulation (PWM)

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.



In the graphic, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() function is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time).

Audio commands analysis – Modify lights' intensity

As it does with the video module, Arduino will receive audio inputs through the serial port as well. Basically users can increase or decrease the lights' intensity. Starting from the standard setting, increasing the lights implicates an increment of brightness of the lights in a determinate step: the decrementation operation works in the same way. This table shows the different states of the system reguarding the intensity of lights:

State	Center	Sides
State 1	63	0
State 2	127	25
State 3 (default)	255	63
State 4	255	160
State 5	255	255

The key-character Arduino board receives are:

1. "R" indicates that user need more light → increase lights' intensity;
   This is equal to moving down in the light's intensity state[2].
2. "L" indicates that user need less light → decrease lights' intensity:
   This is equal to move up in the light's intensity state[2].

The state of the system is always saved to ensure that, when the system is restarted, it will be restored with the same settings present at the shut down time.

[2] See table above

Fade-In/Fade-Out

At the beginning of our project, the lights changed brigthness in a smooth way, using fade-in and fade-out approach. There wasn't a standard function to do this, so we tried writing some proper code. After some tries, we thought we found a good way, shown below:

```
for (int i=currentValueOfIntensity, i<=nextValueOfIntensity, i+=5)
    analogWrite(centerLights) = i
    wait(0.1)
```

The problem with this solution that looked very cool, was the impact on performance of the system. Even if its cost is O(c), it decreased noticeably the speed of the system and caused a loss of the perception of real time feedback.

According to this, we decided not to implement it, and to directly modify the brightness of the lights, reaching a great experience of real-time for the user.

Conclusions and future development

Video/Audio concurrency limitations

In our system, both video and audio modules run in a parallel way, such that the user can manage the lights while moving in the domestic environment: Arduino can then receive both audio and video input simultaneously. Due to hardware limitations, and because of the serial port is capable of handling only one input at a time, data is processed sequentially.

System Management

All the system can be considered as an accidental interaction manager. User isn't directly conscious that a camera is permanently following his movements: this is why we decided to include the possibility to start/stop light management by simply asking the user to pronounce the corresponding command.

Future development

For some possible future software versions we could implement the following features:

- Background training and refresh: since it may happen that the background significantly changes during daytime, it would be usefull to refresh it and to reboot the system at some established intervals to avoid it to be out-dated;
- Pocketsphinx training: would be interesting to personalize the audio module's behaviour according to user's pronuntiation and native language;
- Face detection and recognition: in order to avoid all the video missdetection we can implement a face recognition to be sure that the subject interacting with the system is a human being.