Poulain Matthieu
Telecom Saint-Étienne
Technological University Dublin

Internship Report

On this 21st of August ends my 2 months internship at the Technological Institute of Dublin under Damon Berry's supervision. The main purpose of this internship was to take an existing project previously led by Shane Ormonde and improve it in order to make it capable of performing more functions.

The true goal behind this project is to warn the general public of the dangers of Amazon Alexa's data gathering, but it could easily be adapted to work with other assistants. This project can also be considered to go hand in hand with my comrade Ophelie's one, working on the benefits that these assistants can bring. [A link to this project will be inserted here when available.]

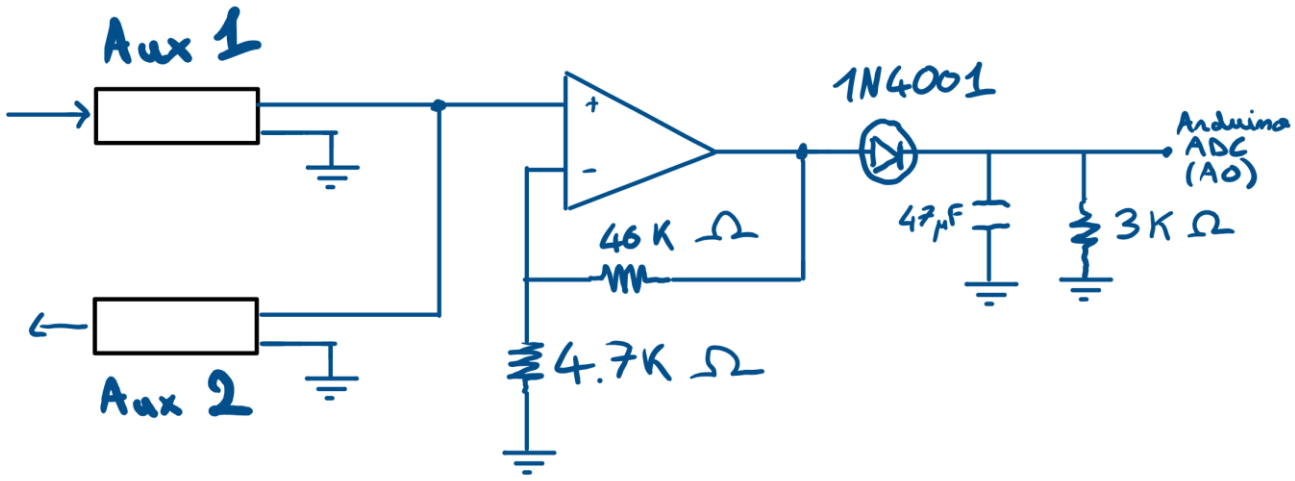
[This document is made in order to give my thoughts and share my learnings with anyone who would like to continue this project. It may be incomplete on some part I though easy to understand, but if some parts aren't, feel free to contact me at the mail address given at the end of this report.]

Envelope detector:

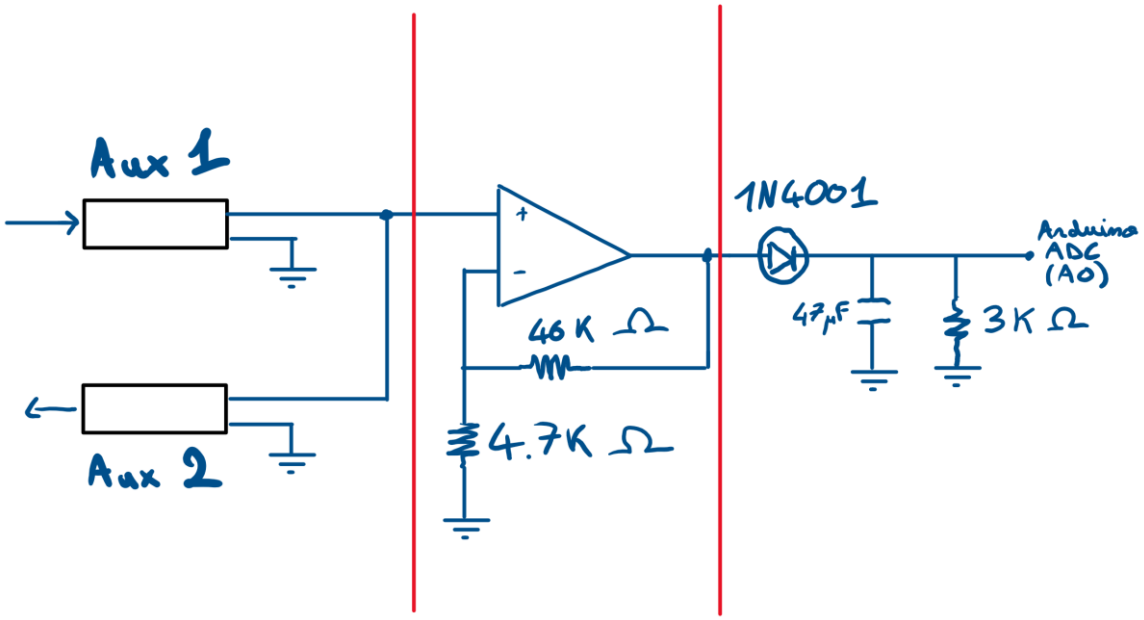
A working prototype has previously been made by Shane Ormonde and can be found here: http://www.wattnotions.com/spooky-alexa-mask-documentation/

The main parts of this project were inspired by his work.

Being able do understand what a circuit is made for is important when working in electronic fields. The easiest way to do so is to split the circuit made by Shane in order to identify primary functions.

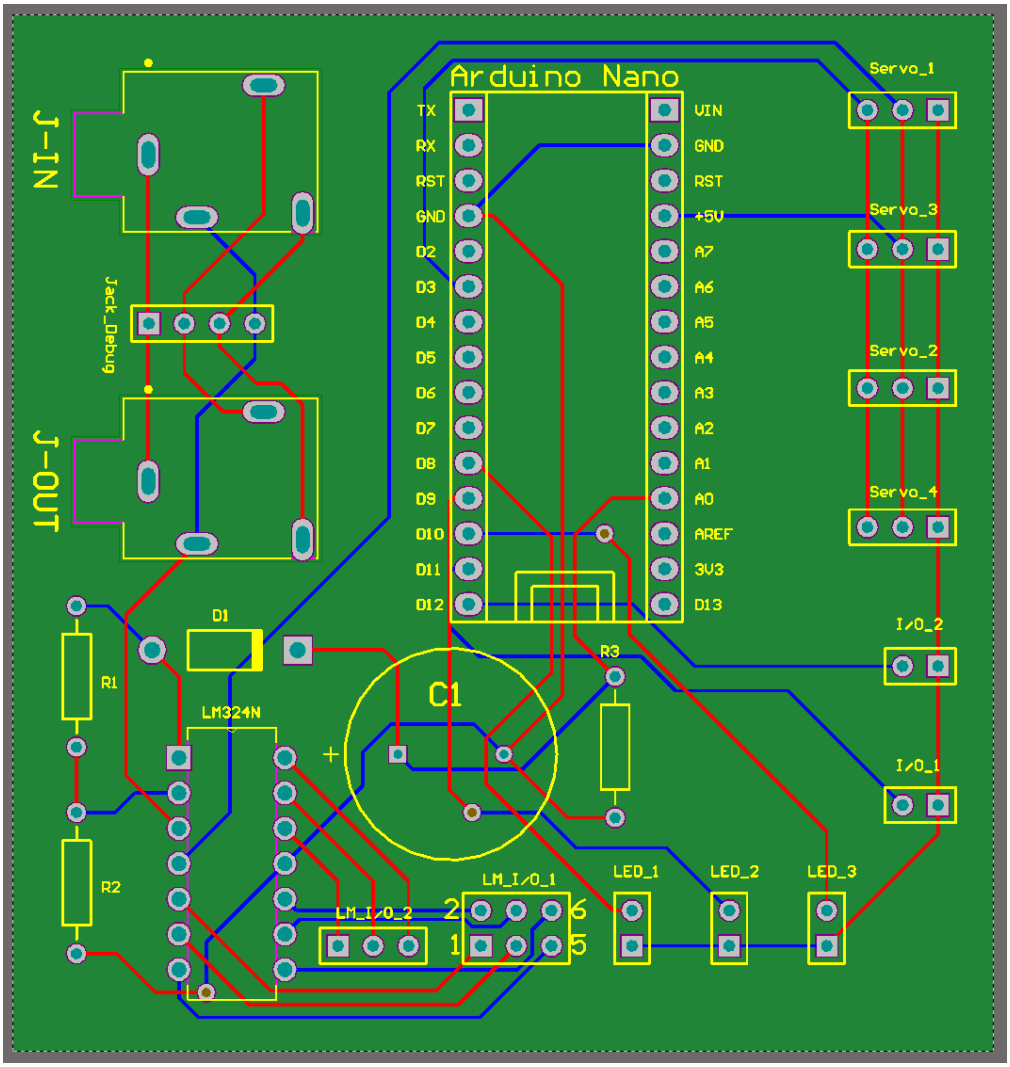


Here, our circuit can be split into 3 parts, the first one sending the signal to the envelope detector and the speakers, the second one doing an amplification of the signal, getting as an output a signal going from 0 to 5v and the last one filtering the frequencies. This way, we are able to get only the voice from the input signal. Take Note that our amplification is set to work with the output of an Amazon Echo device, and may need to be adjusted depending on the device you will use.



With the coronavirus situation, we were confined while working on this project. As we were missing the components required to create this project and because Mr Berry made it clear the purpose of this project was to improve the current mask, and if possible make it a bit more professional, some ideas started to pop up in my head.

In a science fair, when the general public discovers a circuit made of breadboards and tens of wires on the verge of getting disconnected with the weakest impact, it can be a bit confusing. I then had the idea of making with Altium Designer a board containing the different parts of our circuit.



All the files are contained in a directory inside the GitHub repository containing this report.

This board still needs to be modified to include the future changes, and then printed by a manufacturer.

At the start of my internship, we were planning to work with the ultrasonic sensors, as Shane started to use these. After several tests, it appeared they weren't the ideal solution, as their results weren't constant and had trouble detecting clothes, and the more people were far from the sensor, the bigger they had to be in order to get detected.

Hence, the different ultrasonic sensor connectors that were implanted on the board are no longer required.

At this point we have a working prototype of the envelope, connected to an Arduino controlling the mouth servomotor. However, without having the full mask, it isn't possible yet to set the values that will be required to move the mouth at its full range. These values will require to be set at the end of the project when the mask is available.

Those are the improvement that can still be added to the project:

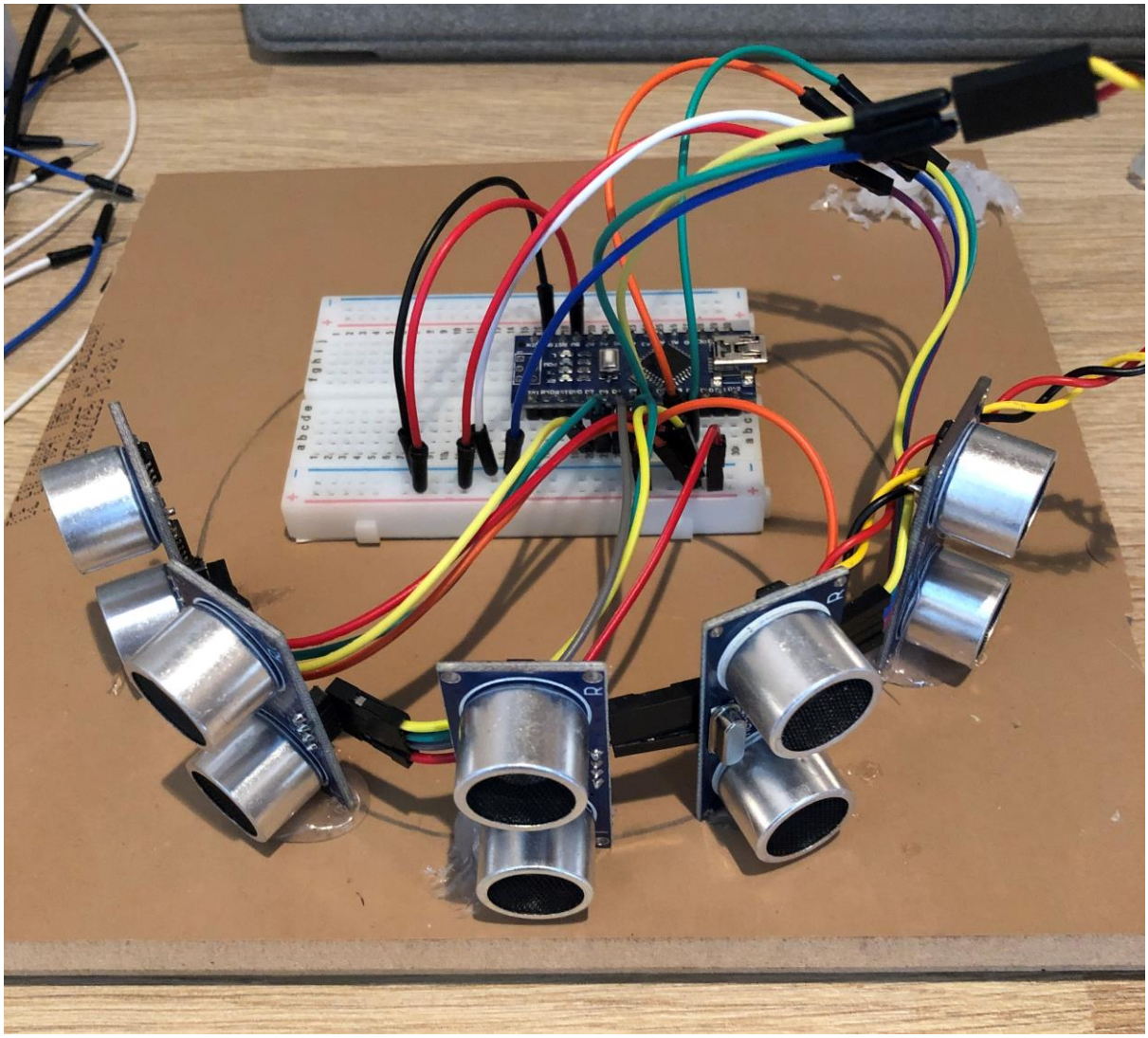
- Increase the connectors size on the PCB files, in case they aren't large enough. This depends on the manufacturer precision. If they are precise enough, this isn't required.

- Mouth opening range variables need to be set properly depending on the output volume from the Echo Dot.

- Print the PCB Board and solder it. (+buy some additional components which were given in the pcb project folder)

Regarding the Ultrasonic sensors part:

A prototyping breadboard using 5 Ultrasonic sensors (1 hasn't been used as I didn't have enough female wires) to detect movements in the room has been assembled.



To this day, I made a Software initialising itself in an empty room measuring the maximum distance it should be able to detect by doing a mean on several measures (can be set manually to increase precision).

The idea is to compare the current different measures (of the 5 sensors) to the initial measures by normalising them and then compares which ratio is the lowest. If one measure fulfils all the requirements (lower than the others and normalised value lower than a certain percentage of the initial value), then the eyes' servo motor is set to make them look in the corresponding direction.

The main issue here is that ultrasonic sensors aren't great in detecting clothes, and for a long distance (>1m), they easily can miss the target if it isn’t large enough. In order to get better results, I started working on an image processing software, first on a computer, and then compiled it for a Raspberry Pi (Model 3).

In order to do so, I learned how to compile OpenCV both on windows and Linux (Raspbian). This part is important and takes a long time, but it is required to be able to compile the code. This part also made me learn how to link required libraries in a project on raspberry pi with the Code::Blocks’ IDE,

to fix the compilation errors. Reminder: all the steps required to compile OpenCV on the Pi (and BBAI) are given in the SetupTheBoards file on the GitHub repository.

When a simple project reading the webcam output successfully compiled, I started working on a facial detecting software, using Haar cascade descriptors. Here, the main idea is to detect a face in front of the camera and track its position on the image. Using the middle point of the square surrounding the face, we can extract the X position of the face (and later we could do the same with the Y position in order to prioritize the following of children faces) on the frame.

Once we have the X position, we need to send it to the arduino. We can do that using serial communication. The first way would be by using the uart ports from the raspberry pi, but they are extremely useful when the communication with the Pi over wifi gets cut, to get access to the console. We can disable the console on those pins and enable it again when needed but that would require a reboot and to change arguments from the cmdline.txt. During the development phase (Mostly when compiling a library) I advise to let the uart console available.

The second way is to communicate through the usb ports, from the raspberry pi usb port to the data transfer port on the arduino. Be aware this method can create some troubles because of the way the Raspberry Pi manages the usb ports. Mine were being disconnected and their numbers were changed randomly (ttyUSB0 changed to ttyUSB1 without warnings, the camera started producing errors when trying to get data from it, and soon, the board died on me, refusing to boot. Reflashing the SD card fixed it, for some time, and then it did it again. [edit 1: After some experiments, it seems the error didn't come from the Pi but from the arduino. Port D12 seems to cause troubles to our Arduino when used with a servo motor. Changing the port to the port 3 seems to have decreased the issue's frequency | edit 2 : The port change indeed improved things, but did not completely fix the issue.]) An advice from Damon Berry would be : once the development stage is over, disable the Uart console and communicate through the GPIO TX and RX ports.

I found in the process that pin D12 is a MISO (Master In Slave Out) pin, which may have caused the previous troubles. However, after setting the servo PWM on pin 3, I still got errors. Using 2 capacitors (47uF,25V) on the command pin(3) seems to have fixed the issue, for now... It may be a bit odd, as the servo gets its power from 5V & GND pins... If the issue remains, try to put the capacitor on the 5V pin, or to use a capacitor with an increased value.

I learned several things while working on this part:

Communication through the GPIO would require the use of one of the following libraries: WiringPi or BCM2835. The first one is much easier to use, while the second one is harder, being a low level library, but it is working much faster according to the website framboise314.fr, which is a very important French raspberry community's website .

The BCM2835 library needs to be compilated manually, but it takes less than 5 minutes (>>1h for openCV)

Although I did not implement communication through the GPIO, I thought it was good mentioning it here.

In addition, based on the ID of a peripheral (Vendor and product ID) , it's possible to create an alias redirecting to the current ttyUSBX associated: for example: ttyUSBMyPeripheral will redirect to ttyUSB0 if MyPeripheral is connected to the port 0, ttyUSB1 if it's connected to 1, in a dynamic way. It means that if by chance our peripheral gets disconnected and reconnects to another ttyUSBX file, we don't have to reload the software, the change will be done automatically.

IMAGE BUFFER Issue :

A common issue with OpencCV is the fact it uses an image buffer, which can cause a delay in processing the current information on the least powerful devices. The issue won't appear on a powerful device like a PC, but it does on raspberry pi, as the Haar Cascade processing takes some time on it.

As I said previously, OpenCV camera objects are using a Buffer containing several images, which are preventing us from getting the last frame acquired (the one we need!). In order to get the last frame, we must set this buffer's size to the least possible (=1) and then we need to empty it to get the current image the webcam is facing. My first though was to use threads to empty the buffer, and at first it appeared as a good idea, as many people were talking about that method over internet, by reading continuously the buffer.

But the thing is combining it with our code made the software much slower (about 50% slower), as it requires a lot of memory accesses. The key part here is combining the 2 previous methods, by setting the buffer size to 1. Then, it only requires reading the buffer a first time to empty it, and a second time to get the current frame. As a result, we read the frame only when required, and we can free resources and speed up the process while doing exactly the same, but in a more efficient manner. If there was a method allowing us to not fill the buffer until required, that would improve things a lot more, as it would avoid any useless memory access, but there is currently no such function available.

About THREADS :

If you get a compilation error, you may have to add -lpthread in the build option, linker settings - other linker options on Code::Blocks.

See the Makefile for BBAI, which may also be used on the Pi.

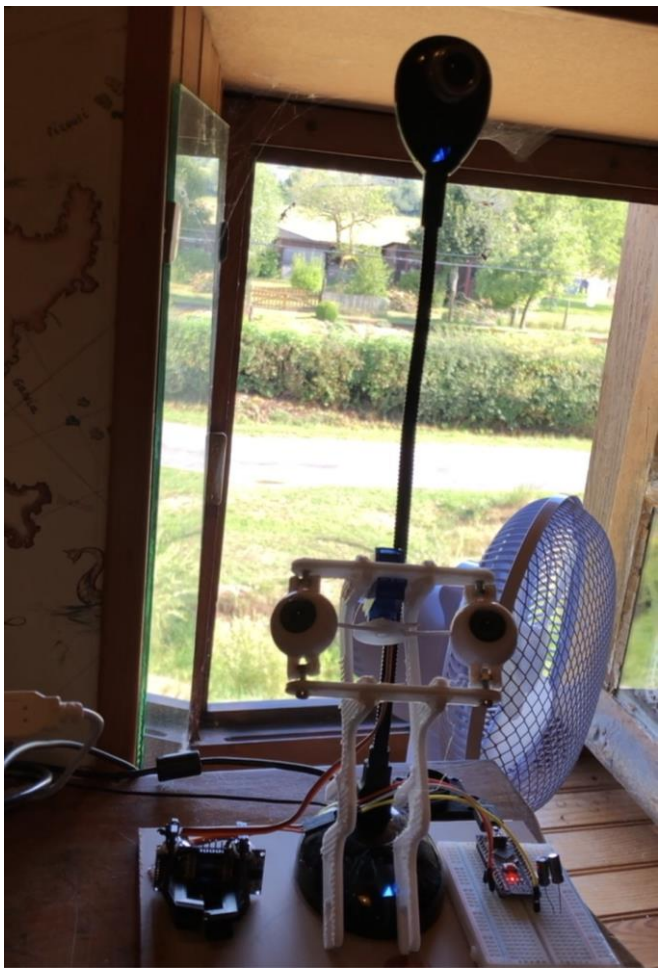
3D printing

You can print the files as send, but be careful: If the 3D printer is a cheap one, sold for less than 300€, you need to have a very good profile, which takes things slowly, and furthermore, do only print your parts one by one. It will take longer, but that way, you can avoid losing all your parts if the print fails.

It took me a long time getting the parts correctly done. The eye plate needs to be shortened, as they are a bit too long to fit without touching the higher and lower parts. If those parts are already printed, you can do that with a soldering iron, set to 210-230 C.

An important part that will be affecting a lot the result is the metal bars used to move the eyes plates. They must be shaped with great care and need to be tightened precisely in order to avoid any clearance between the 2 eyes. (Unless you want the mask to have squinting eyes, which would be pretty funny (Scary?) too.

Once those parts have been assembled, the result is pretty good, but having it work faster would be a great improvement.



Beaglebone AI

It tried to get the software work on a BeagleBone AI, which also has enhanced capabilities with AI, thanks to its hardware made to accelerate these kinds of software.

At start, compiling OpenCV seemed to be a bit troublesome, and I was trying to find a way to do it easily.

In the end, I found that the memory of that board wasn't enough, and the compilation required to use a swap file in order to be successful.

With the given CMake instructions in my file, there is currently an error appearing when acquiring the frame. It comes from the instructions given, but I haven't been able to fix it yet. It doesn't prevent the software to run, but it is bothersome. A fix given on Internet is to add the option WITH_JPEG=OFF in the CMake instructions, as it is said to come from incompatibilities between the current OpenCV version and JPEG libraries (though it worked with the Pi). Anyway, adding this option didn't solve the issue, but it also removed the possibility of setting the buffer size.

There are many things left to do in this project, such as rewriting the Pi/BBAI code in python in order to use the Ai libraries and add some kind of awesome features to the project. I'll also try later to use the code with a Pi4 (vs Pi 3 currently) to see if there are any improvements worth the invest. Another improvement possible would be to use a custom OS, made with Buildroot (Or Yocto). I also have one already made, so I'll try compile the code for it to see how it runs.

[This document may be changed as I add some useful improvements.]

For any further question, if I didn't explain completely something or if I made some mistake, feel free to contact me on this mail: matthieu.poulain@telecom-st-etienne.fr