Face-following robot arm with emotion detection

Vernon Stanley Albayeros Duarte

Resumen – Este proyecto se basa en la creación de un brazo robot con seguimiento de cara que responde a emociones detectadas utilizando una Raspberry Pi como controladora central. Debido a las limitades capacidades de cómputo de la Raspberry Pi, el proyecto explora la Google Cloud Platform como vía en la cual depender a la hora de necesitar recursos de cómputo para operaciones más intensivas como la detección de emoción en una cara. El proyecto también incluye la creación de un conjunto de librerías para el control simplificado de servomotores y que otorgan acceso a la API de Google Cloud Platform, destinados a ser aprovechados por proyectos posteriores. En el proyecto también se crea un conjunto de modelos listos para impresión 3D utilizables para reemplazar partes, o replicar el robot, junto a la lista de partes necesarias para montarlo.

Paraules clau – Raspberry Pi, Robótica, Robot, Servo, Python, Librería Python, LED Direccionable, Pi Camera, Detección de Rostro, Vision por computador, Google Cloud API

Abstract – This project aims to create a face following robot arm that responds to user emotions using a Raspberry Pi as the main controller. Given the computational limitations of the Raspberry Pi, we explore the possibility of using Google's Cloud Platform as a means to offload compute intensive tasks such as emotion detection. The project also aims to create a framework of python libraries that can be used to power similar projects, and grant access to simplified servomotor control and the Google Cloud API. The project also provides a set of models ready for 3D printing that can be used to replace parts or replicate the robot, along with the bill of materials needed.

Keywords – Raspberry Pi, Robotics, Robot, Servo, Servomotor, Python, Python Library, Addressable LED, Pi Camera, Face Detection, Computer Vision, Google Cloud API

1 Introduction

As 3D printing, robotics components like microcontrollers (Arduino), single board computers(SBC from now on) like the Raspberry pi or Beagelbone, and general components (sensors, motors, etc.) become more accessible to the average user, there is a growing community that takes these technologies and manages to build fully fledged projects out of them, some reaching industrial prototype-levels of quality.

This space is driven forward people from several backgrounds, as it requires knowledge from various disciplines. Mechanical engineers can design structural parts for 3D printing, electrical engineers can design smaller, more efficient circuits, and computer engineers can design robust control software handled by a microcontroller like Arduino or a small form factor Raspberry Pi [1].

This growing community also seeks bigger challenges as time goes on [2], and sometimes, an idea is just too big to fit in the small packages provided by current microcontrollers and SBCs. While these devices are very powerful on their own, sometimes an application requires not only results, but also *fast* results. Common SBCs are more than capable of running an algorithm required to compute the likelihood of a face detected in a picture being a sad face or a happy face, but it might not be fast enough to compute the answer and relay it to an application that needs real-time responsiveness.

The aim of this project is to build an interactive robot and use Google's Cloud Computing[3] solution as a means to offload some of these computing tasks to get a real time implementation of a function that would be otherwise too complex to compute within a reasonable time by an SBC.

This article will walk through the development cycle of a robot consisting of a three degrees-of-freedom arm, using a camera for computer vision and an addressable led array for user interaction.

• E-mail de contacte: stanley.albayeros@gmail.com
• Menció realitzada: Enginyeria de Computació
• Treball tutoritzat per: Fernando Vilariño Freire (departament)
• Curs 2018/19

2 OBJECTIVES

The project has two distinct parts with different objectives. First and foremost, we can state the ”big picture”, or ”meta-objectives” of the project:

1. Explore Raspberry Pi and the Raspbian Linux distribution as the main controller of a robot as an alternative to Arduino ICs.
2. Learn about Google's Cloud computing services, specifically their Cloud Vision offering and how it can be used to offload traditionally resource intensive computational tasks like recognizing the emotions of a person in a photo.
3. Provide python package libraries for servo motor control and Google Cloud Vision API access.

To achieve these objectives, I set out to build a robot inspired in one of the previous Robotics, Language and Learning subject's projects. This part came with it's own objectives:

1. Design around an existing physical frame to build a ”DIY-ready” robot.
2. The robot must detect a face in real-time using the attached Raspberry Pi Pi Camera.
3. The robot must be able to keep a detected face within a region in it's field of view, meaning it ”follows” the face around.
4. The robot must utilize the Cloud Vision API to recognize emotion features in a detected face.
5. The robot uses a 16 addressable LED ring to interact with the user and provide feedback on it's current status.
6. The robot must use the responses of the Cloud Vision API to react to the perceived emotional state of the user.
7. The robot must perform these functions fast enough for them to be perceived as real time (less than a second).

3 STATE OF THE ART

The market for DIY robots and electronics projects is on the rise. In the past decade, microcontrollers like Arduino have made electronics DIY projects very affordable to the average consumer. The ease of programming these boards, and the visual and physical impact these projects produce have also opened the field to people that might not otherwise be inclined to grab an IDE and learn to code. The current offering of projects of this type is mainly centered around 2, 3 and 4 wheeled vehicles that usually implement some sort of obstacle avoidance or remote control via Bluetooth or RF. As a laboratory assistant the past couple of years for the Robotics subject, I have also noticed that most people, even with the programming knowledge needed to implement robust computer vision or machine learning techniques do not take too many risks when designing a robot using small components.

I believe the main reason for the simplicity of these designs is the lack of raw computing power on these microprocessors and SBCs. Most Arduino boards have a very limited memory size, and SBCs like the Raspberry Pi are limited by RAM size and processor power, considering they have to run a full blown OS parallel to our control software. This climate is ripe for a change, and I believe tapping into the potential of the cloud to offload these tasks can lead to an increase in design complexity of these projects.

4 METHODOLOGY AND TOOLS

On this section, I will provide information about the methodology followed through the design and development cycle, and the tools needed to complete it.

4.1 Methodology

The project followed an agile methodology organized in sprints corresponding to the milestones planned.

4.2 Planning

Name	Start Date	End Date	Days Elapsed
Google Cloud API Research	10-01	10-04	3
Objectives definition	10-04	10-08	4
Testing Google API	10-08	10-10	2
Dependency checks on Raspberry Pi	10-10	10-12	2
Source/design robot frame	10-12	10-22	10
OpenCV face detection	10-22	11-04	13
Emotion detection	11-04	11-17	13
Print Frame	11-17	11-24	7
Servo parametrization	11-24	11-29	5
Validation / Testing	11-29	12-02	3
Gesture parametrization	12-02	12-06	4
Validation / Testing	12-06	12-10	4
Emotion reaction parametrization	12-10	12-21	11
Vision and motion engines integration	12-21	2019-01-06	16
Validation / Testing / Optimization	2019-01-06	2019-01-21	15
Documentation	2019-01-21	2019-02-11	21

TABLE 1: PLANNING

The project's planning was based in milestones to reach in a certain number of days. The original planning was

made without taking into consideration several development hurdles that appeared in the project's development cycle. Table 1 describes the final development planning.

The project's planning was kept track with Microsoft Project. The final Gantt chart can be found in the appendix.

4.3 Tools

I used several software packages to complete this project.

- Raspbian OS[4]: Linux distribution used in the Raspberry Pi.
- Fritzing[5]: Software package used to generate the connection graphics for the electronics components.
- FreeCAD[6]: CAD program used to generate or modify some of the printable parts needed for the robot.
- OnShape[7]: Online CAD program, used to create and modify most of the printable parts needed for the robot.
- Simplify3D[8]: STL Slicer, used to transform the 3D models generated by the previous CAD programs into printable GCode.
- 3D Printer (FDM): I used my own 3D printer to print the whole robot. It is an FDM-based 3D printer, model Tevo Tarantula.
- LucidChart[9]: Online tool used to create the state machine diagram in this paper.
- Office365[10]: Used to create the reports throughout the project's life cycle.
- Overleaf[11]: Online LaTeX editor and package manager used to generate this article.
- Visual Studio Code[12]: IDE used to edit all of the source code.
- Git: Source control.
- WinSCP[13]: used to transfer files to and from the Raspberry Pi.
- Putty: SSH client used to access the Raspberry Pi without the need to connect a monitor.
- RealVNC[14]: VNC Client used to access the Raspberry Pi's display out to verify camera feed.

5 Why Google Cloud?

When searching for cloud-based compute engines, Google's offering had an excellent documentation, a community supporting it, and had a few interesting offerings like the Cloud Vision API that we use in the robot. Even as a commercial compute engine, Google offers students a fair bit of documentation and free credits to utilize the engine for academic purposes.

5.1 Google Compute Engine

The Google Compute Engine is a suite of cloud computing services offered by Google that leverage their infrastructure to provide compute power to clients. The API offers a wide arrange of services, ranging from providing raw compute cores to run our own custom code on them, to data analysis tools, cloud storage and machine learning functionality.

5.2 Google Cloud Vision

I was especially interested in Google's Cloud Vision API. This API is capable of analyzing pictures to extract features. A few examples of things that this API can recognize are: face and landmark detection, OCR and logo recognition. For our use case, I found the face features detection to be the most interesting. When we send a photo with a face in it to the Cloud Vision API, part of the response includes the following:

```
"joyLikelihood": enum(Likelihood),
"sorrowLikelihood": enum(Likelihood),
"angerLikelihood": enum(Likelihood),
"surpriseLikelihood": enum(Likelihood),
```

The Cloud Vision API is capable of recognizing a given person's emotion from a photo, and returns the likelyhood of one out of four emotions. Before implementing this in our robot, we need to test how reliable is this information, and how fast we get it. The details of this testing and its results are discussed in the "Results" section of this article.

6 Robot Prototype

Initially, I was going to take advantage of an existing robot made by a group of students a few years ago called "Pixi-Lamp" [1]. Unfortunately, the hardware configuration of the robot made behaviour extremely unstable. The combination of high-torque servomotors used with the very heavy metal-based frame, made for very high inertia movements when I tried to simulate some emotion response routines, leading to overheating of the servomotors and general frame instability.

As the existing frame would not work for this project, I set out to look for an alternative. I found the base files of another student project with a 3D printed arm, originally created by Slant Concepts called LittleArm 2C [15]. I contacted the company and was granted permission to use their files for this project, and to modify them for the needs of the project as long as their original files were not made public.

6.1 Hardware

This section will discuss the bill of materials needed for the project. Table 2 reflects the final budget for the robot.

- SG-90 servomotors were chosen for their size and respectable 1.3 Kg-cm torque.

[1] https://rlpengineeringschooluab2018.wordpress.com/2018/05/29/pixi-lamp/

TABLE 2: BUDGET

Name	Units	Cost	Total
SG-90 Servo	3	8.95 €	26.85 €
Raspberry Pi 3B+	1	38.95 €	38.95 €
25W PSU	1	15.00 €	15.00 €
10W Power Adapter	1	12.00 €	12.00 €
PCA9685 PWM Controller	1	12.00 €	12.00 €
Neopixel 16 LED Ring	1	16.15 €	16.15 €
Total			**120**.85 €

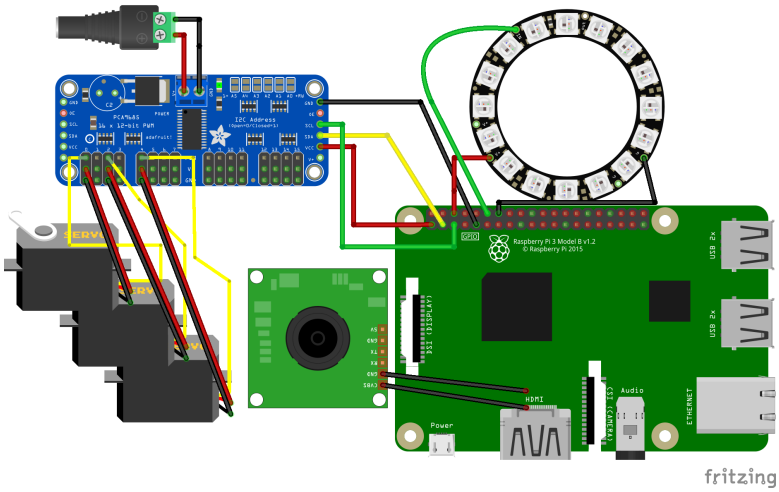

Fig. 1: Fritzing Diagram of all the hardware connections.

- A Raspberry Pi was chosen as the main controller for the robot.
- The Raspberry Pi only has a few hardware PWM enabled pins, with only two PWM channels available. To use more pins as PWM-enabled pins with more channels there are several software-based solutions, but I found these to negatively affect the performance of the robot. As a result, I used an I2C based PWM generator, the PCA9685 12-Channel PWM Driver. This component can be connected to the Raspberry Pi through the I2C interface, and does not rely on the Pi's CPU to accurately generate the pulses needed to drive our three motors. This completely frees up the PWM pins on the Raspberry Pi.
- I used a 25W (5V 5A) power supply to power the servo driver and the servomotors. At peak current, the servomotors use around 2A in total.
- 10W power Adapter: This is the power supply for the Raspberry Pi. At first, I was feeding power through a voltage regulator (3-12V To 5V) from the main PSU, but the voltage drops caused by the servomotors reaching their peak current and the voltage drop inherent in voltage regulators meant that the raspberry pi was getting undervolted consistently while face tracking was active, causing the CPU to underclock to prevent shutting down. This affected performance negatively almost by a 3x factor.
- The Neopixel ring is an array of 16 addressable LEDs that only require one PWM pin to be controlled.

The final connections are configured as pictured in Figure 1.

6.2 Frame

The LittleArm 2C was originally created for slightly different motors and meant to be used with an Arduino-based control board, but the main frame of the robot could be adapted to fit the needs of this project. I re-sized the motor sockets to fit our motor's dimensions, and strengthened the walls of the arm, as it would now need to support a heavier load. I created a control box to house the electronics and a base for the arm that connects to the control box. This

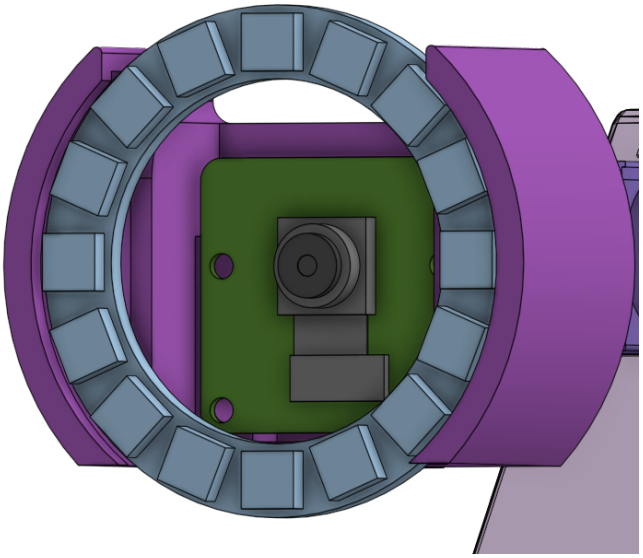

Fig. 2: Camera and LED ring holder.

makes the robot completely self contained, and the added weight of the control box with the PSU and Raspberry Pi inside provides stability to the arm.

I also created an adapter for the arm to house the camera and the LED ring (Fig 2), which goes in the place where the gripper would be in a traditional pick and place arm. Since the arm no longer needs a servomotor to power the "hand", I modified the forearm to be slightly shorter, and to allow the Pi Camera's ribbon cable and Neopixel Ring's wires to pass through. This added weight lead to the need for sturdier walls for the firearm, so I also increased their width slightly.

You can find a picture of the final robot arm in Figure3.

6.3 Dependencies

The robot's software uses the following:

- CircuitPython Library [16]: This library provides functions to control the PCA9685 servo driver.
- Neopixel Library [17]: This library provides functions

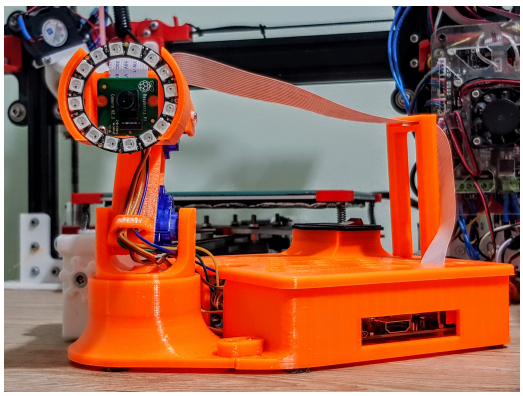

Fig. 3: Final Prototype.

to control the Neopixel Ring.

- OpenCV [18]: Provides functions to analyze images. In our case, it allows us to detect a face within a photo frame using the Raspberry Pi's hardware.
- Google Vision API [3] : Analyzes images and identifies features.

Figure 4 shows how the libraries are connected within the robot's control system.

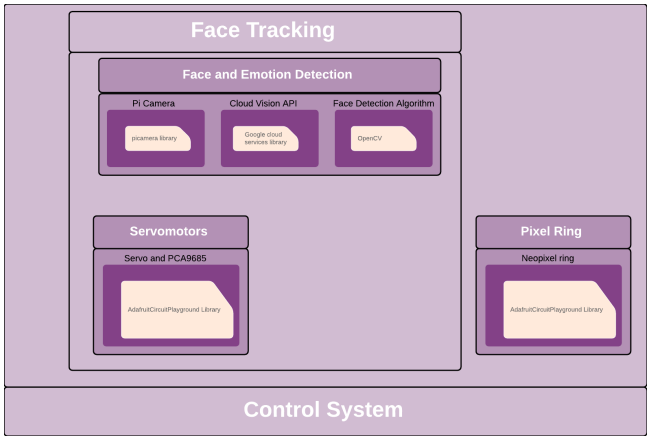

Fig. 4: Software dependencies interaction

6.4 Robot State Machine

In this section, I will describe the robot's behaviour. When the robot's PSU and Raspberry Pi are connected to a power source, the robot first initializes all motors to a known starting position. This position is the middle point of each servomotor's movement range. The reason for this, is that we can't know the position of the servomotors before starting the robot, and when we first initialize them through code, they will move as fast as possible to the first position we ask them to. This can negatively affect the servomotors if the robot was left in a configuration too far away from the initial conditions. I chose the middle point as the initial position because it is reasonably easy to replicate with the robot turned off, minimizing the distance needed for the motors to travel for the initialization.You can find the Robot's state diagram in figure 5.

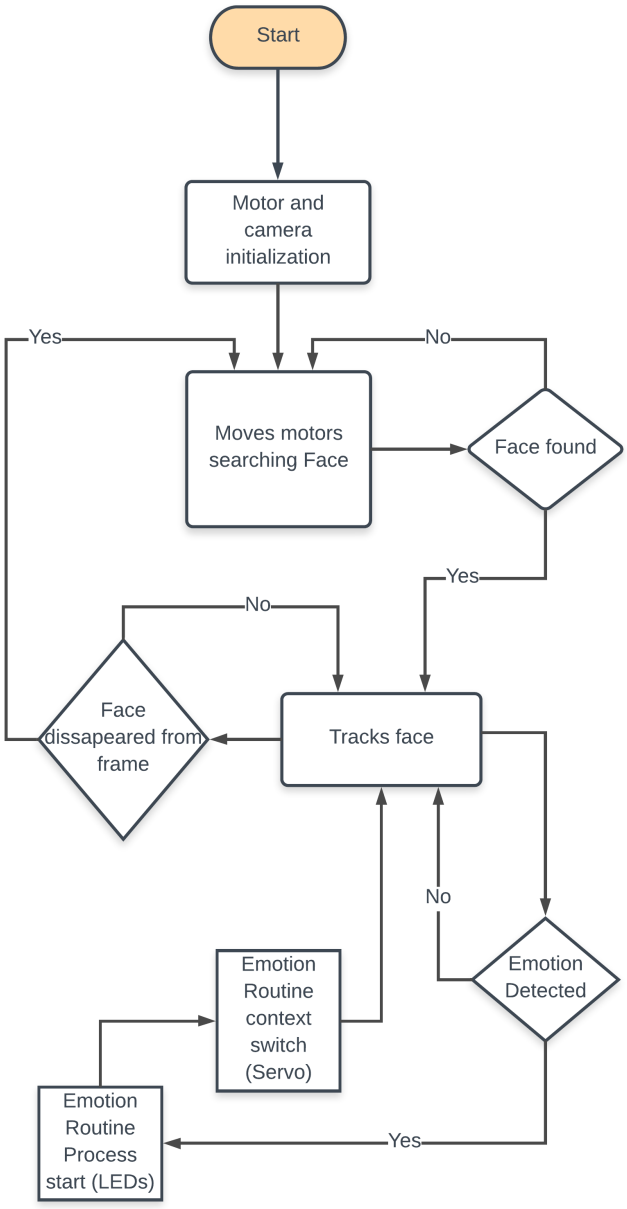

Fig. 5: State Machine

7 TESTING

Every robot component had software module has been tested. I utilized the following tests for this project:

- Unit Test: Tests a single component or software function. This was done to verify that every single hardware component was functioning properly, and that the methods implemented were working properly. The following unit tests were made:
- Integration Testing: Verifies whether the interaction between components or methods provoke unforeseen behaviour.I used this to verify that the hardware component's functionalities did not interfere with the execution of control software.
- System Testing: Verifies the correct functionality of the system as a whole.

SW / HW Unit Tested	Purpose	Status
Servo(x3)	Establish our servo's duty cycle limits	OK
Servo(x3)	Verify that all our servos can achieve their full range of movement	OK
LED Ring	Verify that all LED's can achieve a visually similiar color given the same values, and how power requirements vary given different colors and brightness values	OK
PCA9685	Verify that the PWM generated by the driver translates to the same range of movement on the servomotors	OK
Raspberry Pi	Setup environment and verify that all libraries work correctly	OK
Pi Camera	Verify capture timings and image quality	OK
Face Detection	Verify that it detects a face in a frame	OK
Emotion Detection	Verify the accuracy of detected emotions	OK

TABLE 3: UNIT TESTS

SW/HW Module Tested	Purpose	Status
Pi camera + Face Detection	Verify that we can take a frame from the camera directly to the face detection algorithm and get a correct detection	OK
Face Detection + Emotion Recognition	Verify that a frame that was detected as containing a face by the Face Detection algorithm can be successfully analized by the Emotion Recognition method	OK
Face Detection + Servo Movement	Verify that the servomotor's movement methods are precise and fast enough to keep a detected face within defined boundaries in the camera's field of vision	OK

TABLE 4: INTEGRATION TESTS

System Tested	Purpose	Status
Face Tracking + Emotion Reaction	Verify that integration between the face tracking module with the Emotion reaction module for the addressable LEDs is possible	(!) One Issue

TABLE 5: SYSTEM TESTS

After completing the system test, I ran into the only issue in the project. Integrating the code that runs with the Neopixel Library with the code that runs the servomotor movement can occasionally result in a small time frame where the robot does not move the servo motor due to a conflict with the Python Global Interpreter Lock, where the thread running the LED control module is holding up CPU focus while the thread in charge of moving the servomotors is waiting for release. This problem can be solved by migrating from using threads for the program to using the multiprocess python library, but this in turn would create issues with the methods in charge of getting the response from the Google Cloud Vision API call. As this problem only results in a small time frame (¡2 seconds) where the servomotors appear to stop tracking, and is a very rare occurrence, I decided to not migrate the code from Threads to Processes, as the "unknown" factor of potential problems arising from process-specific issues was too big of a risk factor.

8 RESULTS

8.1 Cloud Vision API testing results

To test the accuracy of the Vision API, I used the Cohn-Kanade dataset. The Cohn-Kanade dataset, CK dataset from now on, is a well-known dataset for facial expressions. To test the dataset, I first had to prepare it. The original dataset consists of 10.708 images, grouped in 593 sequences across 123 subjects. Not all these images are labeled for emotions, but they provide a helpful file system where every image that has emotion data has a corresponding FACS label file. Facial Action Coding System, or FACS, is a system to taxonomize human facial movements by their appearance on the face, based on a system originally developed by a Swedish anatomist named Carl-Herman Hjortsjö. The dataset was labeled based on what expression was requested rather than what may have been performed. The provided final emotion labels are the result of analyzing the FACS labels and transforming them into recognizable, "simple" emotions. The FACS labeling system and how to interpret FACS formulas are beyond the scope of this project, so we will be using the emotion labels provided by the dataset authors. To prepare de CK dataset, I wrote a python script to hunt down the image files that had emotion labels attached to them.

The CK_Images folder contains around 120 folders labeled SXXX (where XXX is a three-digit number) each with up to 6 other subfolders labeled YYY. Within these folders, we have the actual images of the dataset, labeled SXXX_YYY_ ZZZZZZZ.png, after the two folders above their path.

The Emotion folder follows a similar structure with text files named SXXX_YYY_ ZZZZZZZ_emotion.txt, after the file they are describing. Within these text files, we find a single float number in scientific notation. The number is in the 0 to 7 range, describing the following emotions: 0=neutral, 1=anger, 2=contempt, 3=disgust, 4=fear, 5=happy, 6=sadness, 7=surprise.

Now, in this we find two main problems:

1. Their classification system is more precise than what

TABLE 6: ORIGINAL DATA RESULTS

Row Labels	Hits	Misses	Total	Accuracy
R_Anger	29	64	93	0.31
R_Joy	69	22	91	0.76
R_Sorrow	26	32	58	0.45
R_Surprise	82	3	85	0.96
Total	206	121	327	0.63

Google can classify with their Vision API. We see that the CK dataset provides a couple more emotions than what the Google API returns. Cloud Vision has the following emotions: Anger, Joy, Sorrow and Surprise, while the CK dataset includes labels for Neutral, Anger, Contempt, Disgust, Fear, Happy, Sadness and Surprise. Now, we have a couple of choices on how to move forward on our test:

(a) We discard every CK file that has a label that doesn't match with Vision API's label system.

(b) We try to group CK labels into Cloud Vision API labels.

2. There are several instances where the maximum likelihood that the Vision API returns does not give a clear enough picture of what emotion the face is expressing. We might be able to solve or reduce this problem by solving the previous problem first.

For the first problem, I decided to explore option B. Taking all the data we have aggregated into a single spreadsheet ("Comparison" sheet in CK_Data.xlsx), we can play around with Excel's filters and sorting systems. A quick look around and we find that there are plenty of occasions where the emotion detected by the Vision API as most likely to be in the image does match the CK emotion label, but there are also many rows where the Vision API doesn't give a similar answer to what we expected judging from the CK labels, many of which don't exist in the Vision API label system.

TABLE 7: RESULTS COMPARISON

Labels	A	C	D	F	H	S	Su
R_Anger	29	10	50	1	0	2	1
R_Joy	1	6	3	12	69	0	0
R_Sorrow	15	2	6	9	0	26	0
R_Surprise	0	0	0	3	0	0	82
Grand Total	45	18	59	25	69	28	83

After considering this table and the PREVIOUS ONE, we can notice that some new tags could be "joined" into the Cloud Vision API to better interpret the results. Anger could very well be interpreted as disgust or contempt by CK's methods using the more nuanced FACS nomenclature, for example. Taking this into account, I merged the following emotions:

- Anger: Contains Anger, Contempt and Disgust.
- Joy: Contains Happy.
- Sorrow: Contains Fear and Sadness.
- Surprise: Contains Surprise.

With these changes in mind, I can re-make both previous tables and judge the results based on this.

TABLE 8: MODIFIED DATA RESULTS

Row Labels	Original Data Accuracy	Modified Data Accuracy
R_Anger	0.31	0.96
R_Joy	0.76	0.76
R_Sorrow	0.45	0.6
R_Surprise	0.96	0.96
Total	0.63	0.84

We can see an immediate improvement over the old accuracy after we made these changes. I believe that this accuracy is more than adequate for our Use Case, since the camera feed will be polled periodically for new expressions, and there is a good chance we never go more than a few seconds without getting an accurate facial recognition hit. I think it's interesting to note that the Vision API misses for Joy are mostly failing in favor of CK's Fear label.

After testing, I checked the logs of the duration of every query to the Vision API, and found that it averaged 0,45 seconds over one hundred thousand API calls. I consider this fast enough for our application, given that we don't need to update the user's emotion faster than once every few seconds. Figure 6 describes how the Vision API Control loop works.

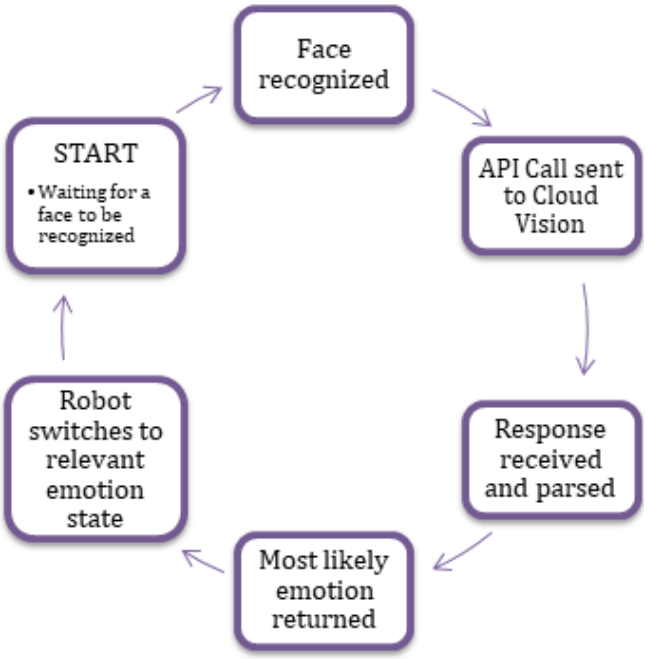

Fig. 6: Cloud Vision Implementation

8.2 Response time

There are three functions that the robot has to be able to perform fast enough for them to be perceived in "real time":

- Emotion Recognition: The results for the speed tests of the Emotion Recognition functionality were gathered

when testing the Cloud Vision API. We can reliably get a recognized emotion provided there is a face in the camera's frame every 0,45 seconds.

- Face Detection: When unit testing the face detection module over one million iterations, the average time needed to complete a frame capture to face detection was under 0,048 seconds. This translates to a real time tracking with over 20 updates to a face's relative position within a frame per second.
- Servomotor Movement: The servomotors move at a near instantaneous rate for small angle deltas like the ones we use for face tracking. Their movement is solely limited by how fast the Face Detection algorithm performs, and since it performs in real time, so do the servos.

9 Conclusions

At the start of the project, we had a series of objectives. This section will summarize the achievement status of these objectives.

First, we had three "meta-objectives". Regarding the first objective, we have explored Raspberry Pi and Raspbian OS as the main controller of a robot, and successfully implemented a control system for a robot with sensors and actuators within this OS. This objective is considered completed.

The objective to research and implement Google Cloud's API into a robot and use it as a means to offload a computational task that would otherwise take too long for a Raspberry Pi to complete has also been achieved.

I also set out to make the classes I used to control the servomotors and Google Cloud Vision access into public packages. Unfortunately, Google's terms and conditions are unclear on this matter, and given the doubt I will not publish the Google cloud access classes of my project. I did, however, make available the servomotor classes as libraries through the Python Package Index. This library can be installed using the command: "pip install –index-url https://test.pypi.org/simple/ tfg-servo-arm "

Regarding the objectives of the robot, I believe I achieved all of the points after the system testing phase. The issues regarding the Python Global Interpreter Lock are very rare, and could probably be fixed by implementing a lower-level library for controlling the LEDs or partially migrating Threads to Processes instead. Regardless, the issue has been extremely rare to encounter while executing the robot.

In all, I believe I succeeded in completing the main objectives of this project. I have learned how to use a Raspberry Pi as a robot controller, I investigated the use of Google's API to offload one of the robot's function to it, and designed the control system for the robot arm.

10 Future Work

The robot can be improved in a number of ways. Multiprocess could be implemented in the CPU-dependent modules to get around the Python GIL limitations of only being able to run one python thread at a time. The servomotors could be replaced with metal axis variants of the SG90 models. A higher end PWM driver could be implemented to increase the number of steps available to the servomotors, resulting in slightly less jerky motion. A more powerful PSU can also be placed in the box to allow everything to run from a single button press instead of needing to connect the Raspberry Pi separately. While the Pi Camera's image quality is very good, it doesn't perform too well under low light conditions, so a way to improve this could be to implement some computer vision trickery to improve low-light images or changing the camera module for one with better performance.

As I have used a Raspberry Pi for the robot, it could be extended with another service, like a Telegram chat-bot to gain more interaction, an Android/iOS application serving as a remote control, maybe even an IR communication system to have two robots "play" among them.

11 Acknowledgements

I would like to thank my tutor, Fernando Vilariño, for the continued support throughout this project's development and through the graduate as a professor and mentor. I thank my family for supporting me through my college years and would also like to thank my partner for putting up with me through crunch times.

References

[1] Andrew K Dennis. *Raspberry Pi Home Automation with Arduino*. Packt Publishing Ltd, 2013.

[2] Robert Faludi. *Building wireless sensor networks: with ZigBee, XBee, arduino, and processing*. " O'Reilly Media, Inc.", 2010.

[3] Google. *Google Cloud Vision API*. `https://cloud.google.com/vision/`. Last accessed 10 January 2019.

[4] Raspberry Pi Foundation. *Raspbian OS*. `https://www.raspbian.org/`. Last accessed 10 January 2019.

[5] Friends of Fritzing Foundatio. *Fritzing*. `http://fritzing.org/home/`. Last accessed 10 January 2019.

[6] FreeCAD Team. *FreeCAD*. `https://www.freecadweb.org/`. Last accessed 10 January 2019.

[7] Onshape Inc. *Onshape*. `https://www.onshape.com/`. Last accessed 10 January 2019.

[8] Simplify3D. *Simplify3D*. `https://www.simplify3d.com/`. Last accessed 10 January 2019.

[9] Lucid Software. *Lucidchart*. `https://www.lucidchart.com/`. Last accessed 10 January 2019.

[10] Microsoft. *Microsoft Office*. `https://cloud.google.com/vision/`. Last accessed 10 January 2019.

[11] Overleaf. *Overleaf*. https://www.overleaf.com/about. Last accessed 10 January 2019.

[12] Microsoft. *VS Code*. https://code.visualstudio.com/. Last accessed 10 January 2019.

[13] WinSCP.net. *WinSCP*. https://winscp.net/eng/download.php. Last accessed 10 January 2019.

[14] RealVNC. *RealVNC*. https://www.realvnc.com/en/. Last accessed 10 January 2019.

[15] Slant Concepts. *LittleBig Robot*. https://www.littlearmrobot.com/littlearm-2c-details.html. Last accessed 20 September 2018. 2016.

[16] Adafruit. *CircuitPython Library*. https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/installing-circuitpython-on-raspberry-pi. Last accessed 10 January 2019.

[17] Adafruit. *Neopixel Library*. https://learn.adafruit.com/neopixels-on-raspberry-pi/python-usage. Last accessed 10 January 2019.

[18] Itseez Intel Corporation Willow Garage. *OpenCV*. https://github.com/opencv/opencv. Last accessed 10 January 2019.

A Annex

A.1 Videos

Youtube playlist with videos URL: http://bit.ly/RobotArmPL

A.2 Frame Pictures

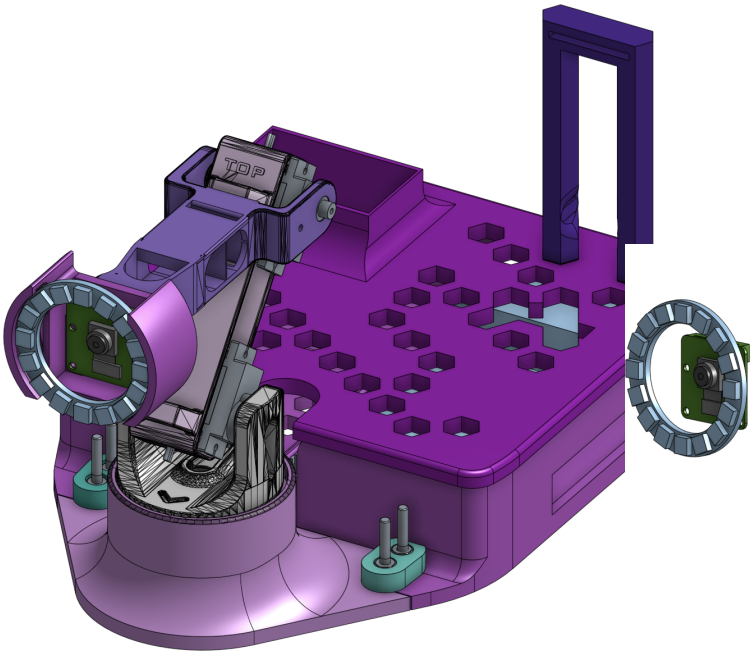

Fig. 7: Base Assembly

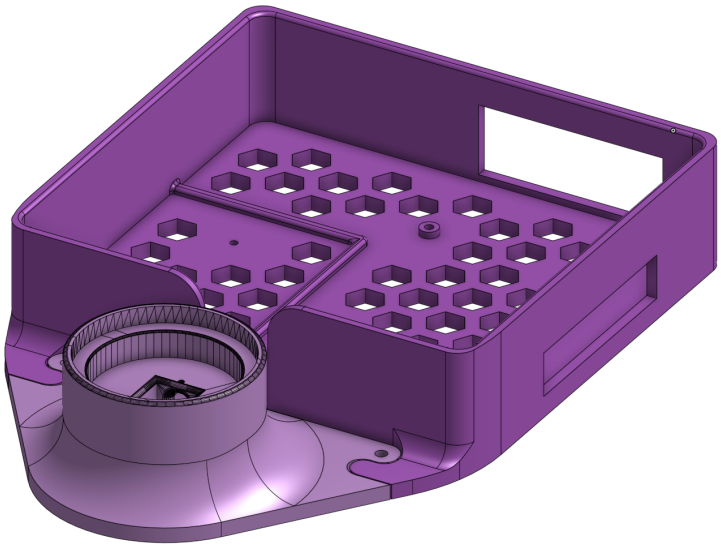

Fig. 8: Final Assembly

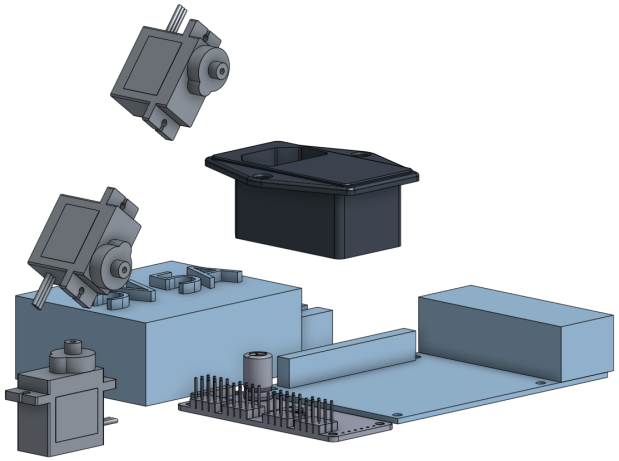

Fig. 9: Only Electronics

A.3 Gantt Diagram