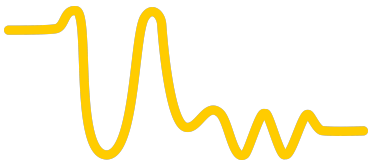
EURECOM
Sophia Antipolis

S5 PROJECT REPORT :



VISUWALK

Group 2 (A) :
Alexandre Avy, Hichem Khettab, Lou Marze, Hamza Parnica,
Guillaume Ung

Supervisors : Renaud Pacalet, Chiara Gialdi, Massimiliano Todisco

Tutor : Navid Nikaein

Contents

1 Abstract

For this semester project, to aid the 2.2 billion visually impaired persons in the world, we were asked to develop a system capable of guiding the latter following a line.
Despite the access to multiple technologies capable of helping them navigate, these devices are for the most part not fit for sports. The most popular solution being the white cane : it is very efficient but not so practical, as it needs a lot of practice to use it and prevents you from using both of your arms/hands if you want to do something.
For this purpose, we developed Visuwalk for this semester project, a device able to guide you in following a line through audio feedback.
The tools we were given for project are a Raspberry Pi and its camera. The main goals of this project are the following :
- Detecting the line from the Raspberry Pi's camera
- Process the information given by the line
- Generate sound cues to guide the user

2 User manual

In order to use our software on your own device, just flash the Visuwalk flavoured Raspbian image from our Gitlab page, onto your SD Card. However, if you have got some issues, or an existing installation that you don't want to mess up, make sure you have the `python3-numpy`, `python3-pyaudio`, `python3-opencv` and `python3-picamera` installed on your device, copy the Python scripts that are in the `rasp_prod/` folder of our repository, and execute `main.py`.

3 Technical manual

3.1 Introduction to the Raspberry Pi and Matlab/Simulink

The first steps towards this project were to connect the Raspberry Pi to our computers and to discover Matlab, which were not easy tasks. The whole installation was quite lengthy, and did not work for most of us. And if it did work, we then had to connect the camera to the Raspberry Pi, which was also quite hard. So hard that we even thought our camera was broken, until we tested it on other computers, which took a lot of time. In the end, we were left with only 2 Raspberry Pi connected and 2 cameras (Alexandre bought one himself).
Concerning Matlab and Simulink, some of us had already used these softwares, but only on surface-level, so we pretty much had to learn from scratch how to use them and more specifically, how to use them with the Raspberry Pi. And this last part was complicated, leading us to this next paragraph.

3.2 Choosing between Python and Matlab/Simulink

The option of using Python showed up because other groups had issues just like us and switched to Python. Thus, we talked about the switch, and decided not to switch completely as we had already advanced on the Matlab code. Instead, we let Guillaume,in

parallel, try to do the video processing part (that was done on Matlab) on Python, to see how easier (or harder) it would be.
It was much easier and produced better results.And so we were left with one last question : should we also do the sound generation part on Python ? We were quite reluctant at first, given that almost the whole code was already done in Simulink and that switching and translating it all would take an important amount of time. Besides, we searched and found multiple ways to make a bridge between Python and Simulink. But time was ticking and we still had not found a way to make any of these bridges work correctly, so we made the switch. And it did not take as long as we expected, despite the issues we encountered.
Still, we found some interesting features like the C code generation in Matlab/Simulink, which would be used to embed the program on a portable system. Knowing that shared memory in C is possible and probably easier than in Matlab, we searched for a way to use shared memory in Python, and to manually modify the C generated code to implement the share of memory between the sound generation part and the Python video processing, so that the second could indicate the direction to the first.

3.3 Image processing

First discarded idea

The first idea we had was to horizontally cut the frame in 2 and process each of them.
On the upper frame, we would calculate the angle between the middle of the frame and the detected line. On the bottom frame, we would calculate the distance between the detected line and the middle of the frame and this distance would tell us whether the user was standing on the right, left or directly on the line.

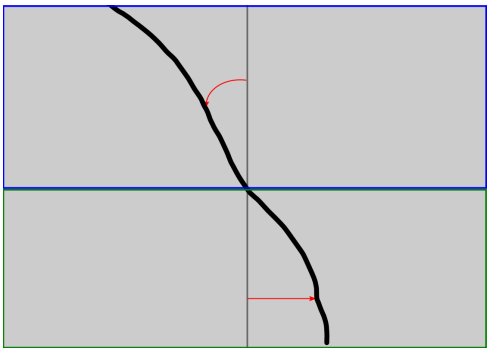


Figure 1: Representation of the first video processing idea

However, we discarded this idea as the calculation of both the angle and distance were not accurate and neither was the instruction. Indeed, rather than taking the fastest path, it would force the user to always stay on the line by making him take sharp turns, leading to a strange pathing.For example,here, it would tell the user to go right because his feet are not on the line and then left when the fastest path would be to go straight and left.
So we changed our approach to the one explained in the following paragraph.

Final approach for the image processing part

The first step toward computing the image and giving an angle is the image acquisition. We use the python library `picamera` in order to get the camera's images, at a quite low quality, because it has proven to be more time-efficient.

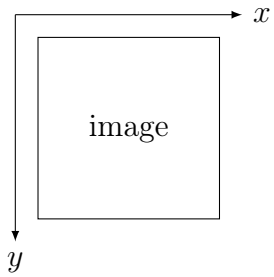


Figure 2: OpenCV x and y axis representation

The frames are then read by openCV and converted into grey. OpenCV represents images with matrices whose indexes may also be seen as coordinate of the following base. An image pixel is then characterized by its coordinate in the following order: `image[y,x]` (matrix line, matrix column).
We then apply a bilateral filter to the image, as it is a filter that preserve the sharp edges and smooth the rest of the image: this is exactly what we need, especially as there is a pattern with lines on the carpet of the Marconi amphiteater that needs to be removed and finally, we apply a canny edge detector, giving us the following image:

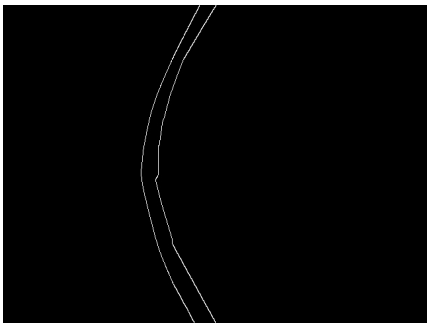


Figure 3: Fully filtered image of the line

After having a clear image of the line, we apply the Hough transform which will give us a set of straight lines detected on the image. But before calculating any angle, we need to reorient the lines accordingly to our purpose. Indeed, OpenCV will return us vectors that are oriented towards increasing x, but we need to have them all in the same direction as y, because the user moves on the y axis.

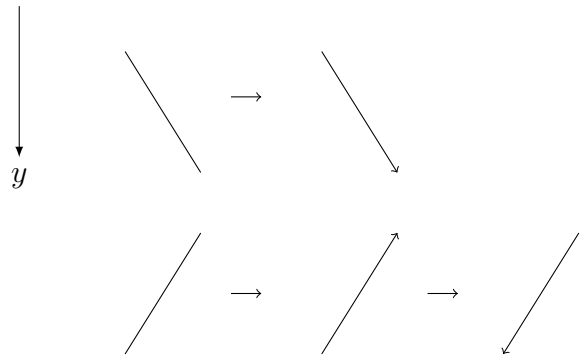


Figure 4: Reorienting the vectors

Afterwards, we consider only the bottom of the frame and we compute the *oriented* angle of each vector v_i with the y axis (defined as the α angle), using:

$$\alpha_i = \text{sign}(x)\arccos\left(\frac{y_i}{\|v_i\|}\right)$$

This angle is effectively the one that has our interest, since it is the same if we flip the vectors in the other direction along the y axis. Then we calculate the average of the α_i angles, which gives us our final α.
In the case the user is not on the line, we had the following strategy to get him back to it: we calculate the angle between the user and a target zone of the line where we want the person to go. As the evaluation is a race, and as a failure could not worsen anything by itself (we count the number of failures and not their durations), we put the target zone a little bit ahead of the line, so that the user gets back to it quickly. This angle has been called the β angle.

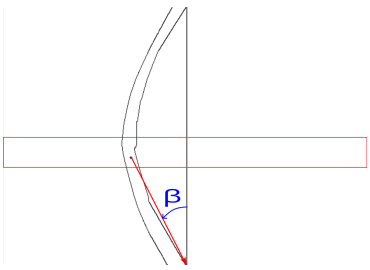


Figure 5: Representation of the β angle

In order to compute it, we take a portion of the image vertically, and look at the barycentre of the pixels that were detected as an edge. Then we have a vector between the barycentre and the user (bottom middle horizontally), which angle with the y axis is computed the same way as for the α_i.
We have to know when to take the β angle into account. For this, we compute the distance between the user and the bottom of the line, parameter we call d. The method is the same as for the target zone: we compute the barycentre of a portion of the pixels that were detected as edges, and then we just count the distance in pixels between the middle of the image and this barycentre (only along the x axis).

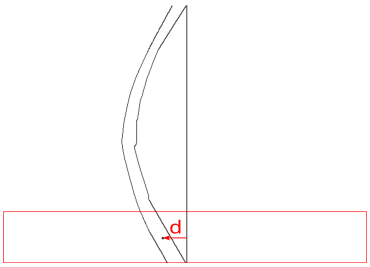


Figure 6: Representation of the distance d

If there is no line in the portion where we look for it, we then only take the *beta* angle into account, because that means that the user is very far from the line and should first get back to it. Otherwise, we mix the α and the β angle relatively to the d parameter, so that the further the user is from the line, the more we take the second angle into account.

We use a d-dependant weight function, which is

$$f(d) = (1 - e^{\frac{-d}{\tau}})$$

We choose τ so that when we are at the image extremity, the value of f is 0.95. Thus we have

$$3\tau = \frac{\text{width}}{2}, \quad \tau = \frac{\text{width}}{6}$$

It follows,

$$f(d) = 1 - e^{\frac{-6d}{w}}$$

Then, for the angle communicated to the user, called γ, we choose this expression to have the behaviour we explained sooner.

$$\gamma = (1 - f(d))\alpha + f(d)\beta$$

Hence we have,

$$\gamma = \alpha + (1 - e^{\frac{-6d}{w}})(\beta - \alpha)$$

Before sending γ to the sound generation module, we bound the value of the angle in $[-70, 70]$. We also reserve $\gamma = 100$ when no line is detected on the image. Then we send the angle.
Note: would we have a cover to fix the raspberry on the body, could we better tune the position of the zones we look at, because we know the Raspberry's camera aperture, and giving the average height of a body, we know which distance we cover with camera and what pixel corresponds to what position in real life.

3.4 Sound generation

Once the angle γ between the user and the line is established, we have to give an audio feedback to the user telling him where to turn. To do so, we first used Matlab and Simulink.
The goal was to use stereo audio to guide the user : if he/she hears sound in his left ear, he/she has to turn left, same for the right.

We simulated the γ angle with a slider ranging from -70 to +70 and linked it to 2 Audio Oscillator of the Audio Toolbox on Simulink (cf the scheme).To start, we set both oscillators at a constant frequency of 440Hz.To determine the amplitude of the oscillators, we did the following:
if γ is positive (the user should go left), the amplitude of the right oscillator (Ar) is zero and the amplitude of the left oscillator (Al) equals 1. Otherwise, it is the opposite (Al = 0 and Ar = 1)
Then, the audio oscillator output is a matrix of double that we converted to integers for the Raspberry Pi. To produce the final stereo output, the two matrices (containing the values of the sinus wave) were concatenated and sent to the Raspberry Pi. The final matrix contains two columns (one for each audio channel) : one is equal to 0 while the other one has the oscillator values (so the sound) and inversely.

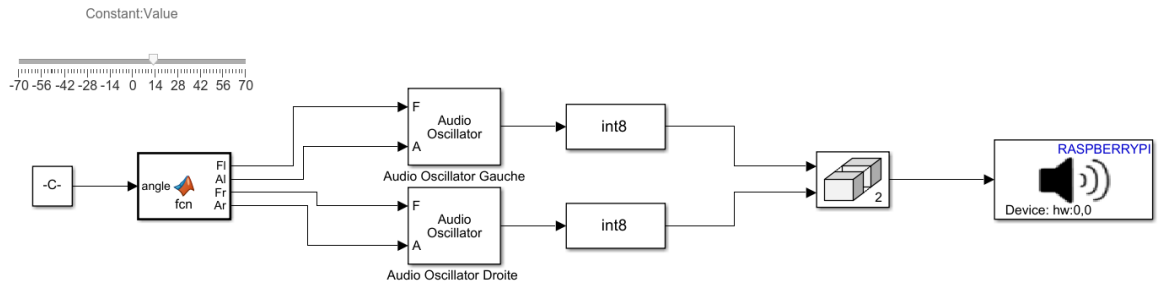


Figure 7: Simulink audio part

To improve the accuracy of the instruction,we decided to implement a varying amplitude to change the loudness of the sound depending on how much the user has to turn.At first, we thought about 2 type of functions for the amplitude A:

$$A = k * |\gamma|$$

$$A = k * \gamma^2$$

With the proportional function, the variation of amplitude is the same for big and small angles, but we found it more relevant to make the amplitude increase a lot for big angles (i.e the user has to make a sharp turn) in order to raise the user's attention in a louder way, leading us to choose the square function. To determine k, we tested different values until hearing a saturation of the sound for $A > 0.7$, So we chose to have $A \in [-0.7, 0.7]$), giving us

$$k = \frac{0.7}{\sqrt{70}} = 0.08$$

However, we also wanted to give information also through the frequency, but not with too high frequencies given that old persons tend to hear them less.We first tested the range of low frequencies that were comfortable to hear: [230 Hz - 500 Hz]. We decided to use a proportional function for the frequency f, and calculated the coefficient the following way :
Given that the biggest angle for γ is 70, we have :

$$\frac{500 - 230}{70 - 0} = 3.8$$

Thus :

$$f = 3,8|\gamma| + 230$$

Finally, after some tests, we concluded that the frequency f was enough to guide the user for big angles (high frequencies) but not for smaller ones. So the loudness variation had to be more important for small angles, leading us to change the function for A to the following square root function :

$$A = 0.09\sqrt{|\gamma|}$$

As previously mentioned, we were not able to link the image processing part made on python to Simulink, so all of the sound generation code had to be converted in python, using the `pyaudio` and `numpy` modules. The methods and functions are the same. After testing the final program, we decided to change the audio for small angles. In fact, when the user is close to the line and going straight, the sound is passing from one ear to another quickly. We decided, with Hamza, to put sound in both ears for very small angles (i.e $\gamma \in [-4, 4]$) to tell the user to go straight.

3.5 Description of source files directory

In the folder of the source code archive, you will find the `rasp_prod` folder, containing the production source code for the raspberry device, the `improc` and `soundproc` folder containing our test codes for development, and the case folder containing some aborted concept protection and fixation covers for the raspberry.

3.6 Packaging

We used pi-gen to generated a Visuwalk flavoured Raspbian image, installing the needed packages and the source files, and running the software at startup. See by clicking on this.

4 Conclusion

We achieved the main goal of this project : guiding a visually impaired person to follow a line. However, with more time on our hands, we could drastically improve this project. Firstly, the handling of the device. We wanted to 3D print a PLA casing to put the device in, so that the user could wear it on his belt and move freely, without having to hold the Raspberry Pi. Not only would it be more practical, it would also improve the performance of the system by fixing the camera orientation and angle relative to the ground, allowing us to fine tune parameters in the code.

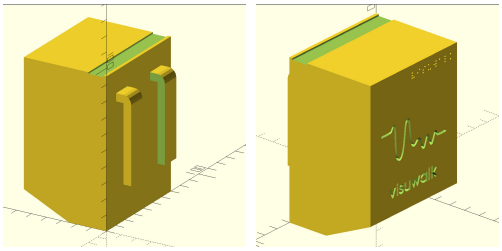


Figure 8: 3D models for possible casing of the device

Another major improvement is the sound of the instruction. As of now, it is functional, but not really nice to the ears. The objective would be to have a sound that is not too invasive and loud, but also gives accurate instructions. 3D audio could also be considered, as it could take into account changes of the ground's height.
And to give even more accurate instructions, we would also have to reduce the latency between the moment the frame is captured and the moment the sound is in the user's ears. To do so, we intended on launching the image processing part and the sound generation part asynchronously. This way, the two processes would communicate without having to wait for one another.
Regarding the business strategy, we thought about putting our source code in open source, allowing everyone to copy it, tweak it and upgrade it to their needs. The benefits we would make from Visuwalk (if we were to be a company) would come from us providing technical support and repair services for the device, lenghtening its lifetime.Thus Visuwalk would be a project fitting in a circular economy dynamic.
Overall, this project was tough but rewarding. Learning how to use a Raspberry Pi and all that comes with it while working in a group of 5 people was challenging, but we still managed to complete the goal in the end, and are very proud of having presented this work to you.