Report on the Bright Line project

François Barnouin, Marwa Essalehi, Henri Gasc
Meriem Lamrani, Nestor Pelletier

Under the supervision of Maurizio Filippone

Contents

Chapter 1

Introduction

The UN estimates that almost 2.2 billion people worldwide[1] that have some sort of visual impairment, ranging from mild vision impairment to total blindness. For those people, it is necessary to use several kinds of tools in order to move around. However, it is hard to navigate without proper help or specialized infrastructures. This results in a marginalization of visually impaired people, even for something as simple as following a predefined path.

The main objective of this project is to develop a system capable of guiding a visually impaired person by helping them follow a line painted on the ground with a high contrast. The line detection have to as precise as possible, while the audio guidance must be clear for an unexperimented user. The application is expected to work for a long use, even though it will be evaluated only on tests lasting only a few minutes. The main requirements are that the system must include at least one video input device and one audio output device.
The material used (more information below) is the following : a Raspberry Pi, a camera module, a battery, and headphones. The Raspberry will use the camera to detect the line and use the headset to communicate with the user.

[1] `https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment`

Chapter 2

User manual

2.0.1 Specifications

First, some specifications about the materials used in this project:

- A Raspberry Pi 4 model B with 2 GB of RAM,
- A Raspberry Pi Camera Module 2,
- A PiJuice HAT Portable Power Platform for Raspberry Pi type Li-ion.

It should be noted that other similar material and any set of headphones should work as well (provided they have audio Jack).

2.0.2 Setting up

There are two possibilities for using this project. Either you use the binary release or you compile it yourself.
It is of interest for everyone to know there are many possibilities to run the program. You can use ssh to connect to the Raspberry and launch the program from there, you can make so that the code is executed at startup, or you can change the behaviour of the buttons on the battery to start it.

Binary release

The binary release of the project should have been given to you under the name *main.elf*. If you do not find it, it can be downloaded from our GitLab.

Compilation

In order to compile this project, you will first need MATLAB and SIMULINK installed on your laptop. When that is done, you will need to install the camera module and the PiJuice HAT battery firmwares on your Raspberry Pi. You can accomplish this part by looking at this document.
Then, you just have to compile and run the code associated with this report in order to use the project.

2.0.3 Indications

The camera should be attached at the user's waist so that the end of its feet are barely in the frame. Is that is not the case, the program should still run smoothly.
When a line is detected, you will hear a sound coming from your left if the line is on this side, and vice-versa for the right.
The further away you are from the line, the louder the sound will be. The higher the angle with the vertical is in the positive, the higher-pitched the sound is. The same principle applies in the other sign.

Chapter 3

Technical manual

We began by prototyping in python to have an idea of what we needed to do.

At first, we thought about using Canny Edge Detection to compute the edges of the line and use them as a guide. You can find the files of this time here or at the commit fba6d2e6.
However, we did not find this approach satisfying because of the time needed to compute everything and its low adaptability to real-life condition such as used paved roads.

This led us to our current approach which consists of interpreting each colour of the image recorded as points from a linear function. Thanks to it, we are able to detect the user's position and the path to follow using the coefficients of the function.
This information is interpreted to guide the user along the line.

3.1 Image

3.1.1 Compression

The images recorded by the camera are in colour which means there are three channels to compute each time. To be more efficient, we convert the images in greyscale, a format much more adapted to image processing.
Because of luminosity changes in the environment, an object of a specific colour can have different hues. To address this problem, we divide the value of each pixel to obtain an image with float ranging from 0 to the number of colour chosen. They are then floored to only have integer.

3.1.2 Filtering

To filter the image, we regroup all pixels from each colour we check that they respect the following criteria:

- The number of pixels has to be in a given range (i.e., 5% to 50% of the total number).
- The same is true if the number of unique pixels[1] along the axis of the line is less than a fixed amount of the resolution along this axis (i.e., 90%).

3.1.3 Fitting

Given a list of pixels, we use the Least Square Method to find the best linear function approximating the pixels. Finding those two coefficients allow us to easily work in the sound part interpreting the slope as the angle of the user to the line, and the intercept as the position of the user.
For better precision, we decided to slice horizontally the image and work on those slices rather than the whole picture. Indeed, a couple of segments are more efficient than a unique straight line to approximate a curve. However, too many segments would result in a very slow algorithm, without bringing much useful precision. After a few tests, we decided to work on five slices, which is enough to fit the shape of the followed line.

3.1.4 Sorting

Resulting from the fitting part, we have for each slice several pairs of coefficients. We use the one having the least error.[2]
Averaging the slope of each slice allows us to have a final line used in the sound part.

[1] If we have 3 pixels with coordinates [0,1], [1,2] and [0,15], the number of unique pixels along the first coordinate is 2 because the first and third pixels have 0 as their first coordinate

[2] The error is computed by summing the square of the difference between the real values and the predicted ones. We then multiply this by the number of unique pixels perpendicular to the line.

3.2 Sound

Next, was guiding the person in question on the detected line. We broke down this issue into two:

Firstly, if the person is on the line but not facing straight to it, we need to tell him to turn either left or right by a certain angle. To do it, we modify the frequency of the sound to reflect this angle.

The second part of the issue is when the person is parallel to the line, in the correct direction, just not the correct point. Again, we need to know where the person is (right or left) with respect to the line. If the person needs to move to the right, that's the earbud the sound will be emitting from, and so on with the left. If the person is facing the line, sound will be coming from both.

3.3 Difficulties

As said before, we used Python at first for the ease of use, to see what we could do. We faced serious problems when trying to use MATLAB and SIMULINK, so we decided to continue using Python.
However, we had to entirely convert our code to MATLAB after it was confirmed that the use of MATLAB and SIMULINK was mandatory for the project. Our lack of proper knowledge of MATLAB made us face many issues while doing the image processing and the line detection. Moreover, we faced some performance issues with it, especially when coding with SIMULINK. In fact, our SIMULINK prototypes were very slow, and the use of the module itself was not very instinctive for us.
Speaking of the material, we faced difficulties with the battery. It took us time to understand how it worked, and why it was always discharging even when the card was unplugged.

We solved most problems by researching them, and reading on the experience of others. It was a lot of work, and not very fulfilling, but the happiness of finally managing to work our project work as intended was worth it.

Chapter 4

Conclusion

We managed to create an affordable and easy-to-use solution for almost all visually impaired people with the use of such materials. Bright Line is far cheaper than most of already existing guidance solutions. It allows its users to move freely in an adapted environment, giving them some autonomy.
Unfortunately, the reason for it being an 'almost' and not 'all', is linked to the method we are using to communicate with the user. There are many reasons people may not want to use headphones: some may like to be able to focus their attention on other things, some may be deaf, etc. It can also take time to get perfectly used to the auditory indications, because we use a wide range of abstract audio instructions.
However, Bright Line is simple enough for the user so that they can start using the application right away, even if they don't know it perfectly.

Bright Line is currently limited by most urban environments, in which there is not any line to follow. Moreover, if there are numerous lines, the program can not lock the desired path, misleading the user.
On the other hand, a large part of these urban environments can easily be adapted to Bright Line by only painting a line contrasting with the floor. Our solution can also be applied indoors.
There are some other obvious limitations to our project, the first and most important one being the life and autonomy of the battery. Others include the relative slowness of the algorithm, how to fix and transport the Raspberry Pi, etc.

Even though a part of the algorithm may need adjustments and the device may require costly improvements, many of those limitations can be solved in order to make Bright Line better.