Navigational aid for the blind and visaully impaired

Pieter Beuselinck, Kevin Buyck, Jasper Vaneessen, Gylian Verstraete

University Of Ghent

Abstract. The goal of this project is to create an application that will help visually disabled people. Certain techniques like GPS (Global positioning system) don not always work well in urban areas. The idea of this project is to support a navigational program by methods of computer vision. We have created and tested this program using videos that were provided by the faculty. The emphasis of this project lies in extracting features from the videos and detecting where the user is positioned by using trained support vector machines.

This project is not meant to work on all possible tracks; it is a proof of concept on the route between the Ghent-Sint-Pieters station and the University of Ghent. In this proof of concept we divided the route into zones and were able to detect our position through these.

1 Introduction

This project is part of a large study that investigates the usability of computer vision as a navigation aid for the blind and the visually impaired. Navigation is an important part of everyday life. Most blind people memorize a small number of routes: they remember the number of steps to the next corner, to the pedestrian crossing, etc. The number of routes that they can visit on their own is limited because this method requires a lot of detail. Using a navigation system based on computer vision could significantly increase the number of routes and improve their quality of life.

The technological support that is currently available for the visually disabled is rather limited. Most members of the blind community still use a white cane as their only navigation aid. This is remarkable since the white cane has been around for more than eighty years. Today we live in a digital age where computer vision could play a very important role in supporting the visually disabled. A system that analyzes and recognizes the environment in real-time could have a great impact on the everyday life of the blind community.

Vision might seem a strange way to implement navigation. However in many situations it is the only reliable option: satellite navigation is inaccurate in most cities and using radio beacons for large distances requires too much external

infrastructure. Most of the time people navigate by using vision and memory; this study wants to emulate that in a computer vision algorithm.

This project focuses on one route. This route starts at Ghent-Sint-Pieters station and ends at the P-building of HoGent, campus Schoonmeersen.

There are three main contributions of our project to the earlier mentioned study. We will introduce each of these briefly below and then describe them in detail in subsequent subsections.

The first contribution of this project is a detector for detecting the amount of grass in an image. This is an interesting feature because our route can be easily divided in zones that contain grass (the Hogent campus) and zones that do not contain grass.

The second contribution of this project is a sequence of detectors that detect different tile structures. We focused on analyzing the line structures of the tiles.

Our third contribution is a program that analyzes a given video of the route and determines the current location of the user.

1.1 Overview

The two following sections of this paper will describe the different feature extractions. Section two will discuss the Color Detection, while section three will tell more about the Tile Structure Detection. In these sections we will describe the most significant aspects of the Feature Extraction and the Feature Selection. The next two sections explain how the segmentation of the route was performed and how the different zones are identified. Finally, our results are summarized in section six.

2 Color Detection

First we begin by explaining the color detection. The first thing we thought to use as a feature was the color. In certain parts of the video we noticed short and long zones with grass, be it on the left, right or both sides. Naturally we concluded that we would try to detect green.

Another color that pops out is yellow. There are some yellow poles along the way and there are also yellow road markings at Hogeschool Gent. We did not use yellow in the final version, because the sun influences this color a lot. It could not be easily distincted from the lightgray of the pavement or concrete paths.

2.1 Feature Extraction

To extract colors from the current frame, we convert it to the HSV range. HSV describes colors (Hue) by their shade (Saturation) and brightness (Value). So now we need to find the HSV range for the colors we are looking for.

We used this range to detect green:

Table 1. Green HSV Range

color	H1	S	V	H	S	V
green	40	20	20	90	255	255

Now we only need to check if a given pixel is inside the HSV range. If so the pixel value is set to 255 in a mask, otherwise the pixel value is set to 0.

We are not interested in the full frame, because the gras is only seen on the left and right side. Because of this we use two regions-of-interest (ROI) to detect the green color: one on the left and one on the right.

2.2 Feature Selection

The features are described by the amount of green pixels. So we count the amount of 'green' pixels inside both regions. This is easy because the pixels are set to 255 if they are green. The featurevector that is fed to the SVM is the number of green pixels in the left and right ROI.

Fig. 1. Frame after using CLAHE.

A big problem with this approach was the image quality. Sometimes the green color was very daft and so the color didn

t get detected. To fix this we used a *Contrast Limited Adaptive Histogram Equalization* or CLAHE (see figure 1 for an example). This made the green much more green and made the detection resistant to the image quality.

To improve the results of the SVM we applied a threshold to the pixel counts. If the amount of green pixels was higher than 20 000 the result was set to 1 otherwise it was 0 (see table 2).

Table 2. Example featurevectors

original featurevectors	adjusted featurevectors
-1 1:12890 2:342	-1 1:0 2:0
1 1:23591 2:342	1 1:1 2:0
1 1:291 2:83922	1 1:0 2:1
1 1:49802 2:123322	1 1:1 2:1

3 Tile Structure Detection

In this section we will explain how the structure of the tiles was detected. The tile structure is a good feature for determining the current zone because:

- the tile structure is very typical for a zone;
- the tile structure remains the same when the weather changes.

By observing the different tile structures we noticed that the tiles could be easily divided into groups by looking at the horizontal and vertical lines that extended the edges of the tiles.

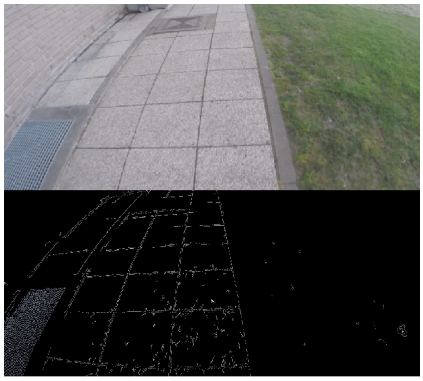

Fig. 2. Effect of Canny edge detector

First we tried to reduce the level of noise by applying a Gaussian blur. This also reduces the influence of textures we are not interested in, like the grainy texture of the stone tiles in Figure 2. Next we applied a Canny edge detector to extract the edges of an image. Figure 2 shows this effect of the tile structure we mentioned earlier. By changing the threshold values of the Canny-function, it is possible to extract the structure of the tiles. One problem still remains: lines appearing next to the tiles are also detected, as can be seen at the bottom right of figure 2. To solve this the region of interest was limited to the tile zone itself.

3.1 Feature Extraction

To extract the lines we used the Hough Transform. This transformation detects all possible lines. These lines are defined by polar coordinates as shown in figure 3. Since we are only interested in horizontal and vertical lines, we count the lines with ϑ values close to:

- vertical: 0 rad, π rad, 2π rad
- horizontal: $\pi/2$ rad, $3\pi/2$ rad

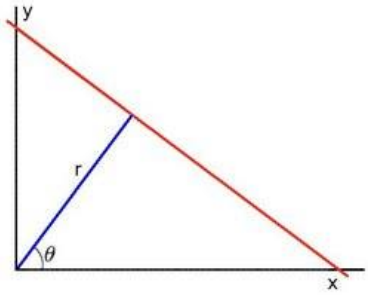

Fig. 3. Polar coordinates of lines in Hough Transformation

We improved the tile detectors by using a perspective transformation. Because of this transformation we now have a top-down view, which makes the tiles more rectangular (the camera angle warps the image into a certain perspective). This makes it possible to detect the distance between adjacent lines more accurately. To estimate the perspective transformation, we used several frames and compared the distance of a tile at the bottom of the frame and at the top of the frame.

3.2 Feature Selection

We focus on three different kinds of features to describe the tile structure:

- number of horizontal and vertical lines;
- maximal distance between lines in horizontal and vertical direction;
- minimal distance between lines in horizontal and vertical direction.

To separate the good and the bad features we divided the route in different tile zones. Figure 4 shows the most siginificant tile types. We used a Support Vector Machine (SVM perf) to estimate the precision of each feature.

Fig. 4. Most siginificant tile types on route between Gent-Sint-Pieters and P-building of HoGent

Counting the number of horizontal and vertical lines appeared to be a bad classifier in most cases, because there are too much similarities between the different tile types. The Support Vector Machine consistently returned precisions below 50 percent.

The maximum distance between the lines result in two features: one maximum distance in horizontal direction and one maximum distance in vertical direction. This method is more accurate than the previous technique for zones with large tiles, but the general precision remains below 50 percent.

The last method also results in two features: one minimum distance in horizontal direction and one minimum distance in vertical direction. This technique is accurate for most of the tile structures and was eventually used for all tile structure detectors.

Tabel 3 shows the different detectors. The exact precisions can be found in section 6.

Note that the narrow tile detector also detects narrow tiles in the zone with big square tiles. The detector responds to the narrow border to the right of the tiles. This information could also be used to better detect the square tile zone in the segmentation part though we did not implement this.

Table 3. Tile detectors

detector	tiles
concrete detector	1
small square detector	4
narrow tile detector	2,5,6
big square detector	5
default pavement	3

4 Segmentation

In this section we will describe how the route is divided into different zones. The size of the zones has to be chosen carefully. Each zone cannot be too short, because if the zone is too small, than it might be possible that it is not detected. However the length of the zones cannot be too long, because they have to include the obstructions for the visually challenged (eg. crossing a road).

All these parameters have been taken into account led to the following zones. For each zone we have created a number of detectors. The first zone starts at the railway station and ends at the first pedestrian crossing. The following zone is situated between the pedestrian crossing and the HoGent campus. When the visually disabled person walks onto the Hogent campus, he enters the third zone. This zone mainly consists of square tiles and ends when the user reaches the large white tiles next to the sports hall. The fourth zone starts alongside the sports hall and terminates when the user leaves the grass field. The user than enters the fifth and last zone, which consist of the tarmac of the university.

Table 4 shows the different zones and the detector(s) they use.

Table 4. Different zone for segmentation

detector	zone 1	zone 2	zone 3	zone 4	zone 5
grass detector			x	x	
concrete detector	x				
small square detector			x		
narrow tile detector		x			
big square detector				x	
pavement		x	x		

Taken into account the above parameters (detectability, obstructions) we have chosen our zones. Tests by the matching algorithm have proven that the zones have been chosen well.

5 Matching and positioning

In this section we will describe the computer vision techniques that we used to detect in which zone we were. This will be done by using a C++ program that will call the feature-selectionprograms.

At first, we will discuss what the possibilities were of detecting the user's position. The first option was to train some support vector machines that could detect each zone. The problem with this option is that there are a lot of blurry images that will give bad results for the support vector machines. The second option was a bit more advanced. It involved detecting the user's start position at first and checking if the user is still in the same zone, or if he has moved to the next zone. This increases the performance of the application, as not every support vector machine has been checked. The bad results given by blurred images were removed thanks to an intelligent voting system with a buffer of the last ten frames. In this application there was chosen to use the second option due to the advantages described earlier.

The second problem was to come up with a good voting system, that was both powerful and quick to react, but was also able to filter out blurry (and possibly falsely interpreted) frames. The first voting system that was used was utilizing the certainness[1] from the support vector machine. It picked the biggest number of both the current zone and the next zone (if there was a positive factor). The problem with this voting system is that some support vector machines were a little to certain (they put up big numbers), giving some sections a lot of false positives.

The second voting system involved a more intelligent one. Each support vector machine of the current and possibly next zone can increase a number if his certainness is positive. This number will be divided by the total number of support vector machines in this zone and will be compared, giving the following formulas:

$$h = numberOfCurrentVotes/totalCurrent \tag{1}$$

$$v = numberOfNext/totalNext \tag{2}$$

Afterwards formula (1) and (2) are compared and if one is bigger than the other, then a vote will be given to that section. If they are equal no vote will be made. If there are no support vector machines the number will be a small number. This will give the section a vote if none of the other sections their support vector machines give a positive. This voting system gave good results.

The final problem was how to implement the buffer. The buffer was implemented by a queue and two integers. The integers represented the amount of votes for

[1] The support vector machine will return a number of how certain he is of the result. This can be ranging from very positive (big positive number) to neutral (almost 0) to very negative (big negative number).

respectively the current and the next zone. The queue is a simple queue of booleans. Each vote will increment either the first or the second integer and the result will be pushed onto the queue (with current being true and next being false). When the maximum queue size has been reached, then the first element will be popped off the queue and the corresponding integer will be decreased. In this project we opted to take a buffer size of nine votes while picking a frame each 23 frames (which roughly means one frame per 2 to 2.5 seconds). This will give the application a reaction time of:

$$5 * 2seconds = 10seconds \tag{3}$$

Implementing this program two parameters can be tuned to fit the user's needs. If the user wants speed above accuracy, than the buffer can be made smaller and the amount of frames per second can be increased. If the user prefers accuracy than the buffer should be bigger and the amount of frames per second smaller.

The matching and positioning system of our application offers a stable and well performing framework that can be adjusted to the user's needs (accuracy versus speed).

6 Results

This section contains the results of our project. We have created six detectors as shown in table 5. The usefulness of each detector can be evaluated by looking at the three values returned by the Support Vector Machine:

- *Accuracy* is the proportion of the total number of predictions that were correct.
- *Recall* is the proportion of positive cases that were correctly identified.
- *Precision* is the proportion of the predicted positive cases that were correct.

Table 5. Detectors

detector	accuracy	precision	recall
grass detector	97,83	97.05	98.67
concrete detector	85.22	55.56	95.24
small square detector	92.42	73.33	91.67
narrow tile detector	97.83	97.05	98.67
big square detector	97.90	96.58	99.12
footpath	96.00	97.15	96.81

Most detectors have a high precision except for the concrete detector. This is caused by the fact that there are some blurry images on the route which are

falsely identified as concrete. Since those blurry images do not occur in sequence, the false positives can easily be eliminated by keeping track of subsequent images.

Eventually these detectors were used during segmentation and positioning. The results where satisfactory: we were able to detect the different zone transitions in both directions.

7 Conclusions

We have presented a way to detect the current location of a person based on images of the surroundings. First we created a number of feature extractors that filter specific elements of the images, like color and tile structure characteristics. To select the best features each extractor was used to train a separate supported vector machine. The extractors were modified until the SVM could make accurate predictions about most images. This resulted in six different classifiers.

Afterwards we have divided the track between the railway station and the university in five logically divided parts. These could not be too short otherwise we will have detection problems. If they were too long than the visually disabled can not be warned about certain obstructions

Finally we have created a framework that can be used on all possible tracks. In order to use this framework on a different track, a programmer only needs to change the detectors and train their SVM. It can handle both blurry and falsely detected images. It can also be adapted to the users needs.