Image Processing Technique and Hidden Markov Model for an Elderly Care Monitoring System

Martínez Cuesta, Abraham; Ladero Gálvez, Júlia

***Abstract*—The main goal of this program is to be able to identify a fall from elderly people in different situations. The program can analyse both images and videos, and it can also analyse live media from the computer camera.**
Once the media is analysed via HOG from OpenCV, the program displays the original image or video with the people it has found framed, and then the console displays message about how the program is running (the steps it's making at the time), and also some information about the analysed media.

***Index Terms*—normal and abnormal states; fall detection; Mixture of Gaussians; graph cut; virtual grounding point; Hidden Markov Model [1].**

I. Introduction

THE original program is intended to serve as an elderly care monitoring system, but since this was clearly out of our reach, we have only worked on human detection. Regarding the detection, we first have created functions to previously modify the received media before the processing is made, in order to try and improve the quality of the detection. Unfortunately, those previous modifications did not work as expected, and even if they sometimes help discard false detection, they do not helps with the people that went undetected.
After those previously-applied modifications, the program has functions to treat with different kinds of media, whether with images, videos or live-taken media from a camera device (in this case the computer camera). Once we know which type of media we are dealing with and how we do have to process it, the corresponding detection functions are applied and, after that, the results (both in images and in data) are shown.
Furthermore, the program has some parameters that can be tweaked to adjust the detection to different environments, and the user is able to change how to process the media they have chosen.

II. Analysed data

First, the program has been tested on images and videos from people running through a street in order to verify if the detection works properly. *[Credits to Raghav Agarwal[2], for the video data and to J. Maus/BikePortland [3] for the image data]*
Later, it has been tested with more domestic situations, such as kneeling to pick something up or falling in the living room, to see if the program confuses both situations and shows kneeling as a fall. *[Self-recorded videos]*

This paper was produced recreating the job of Swe Nwe Nwe Htun, Thi Thi Zin and Pyke Tin.

III. Class: Parameters

This class is used as a parameter box. It receives all the info needed at once and creates a box where the info can be retrieved from.

IV. Functions

A. *prewitt_filter*

The function applies the Prewitt edge filter to the input image, convolving its two kernels with the image to obtain its edges and make the detection easier.
Used kernels:

$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

As observed in the obtained image, the implementation of this function does not really improve its analysis; in fact, it worsens it, but it was left in order to be able to see how it changes the detection.

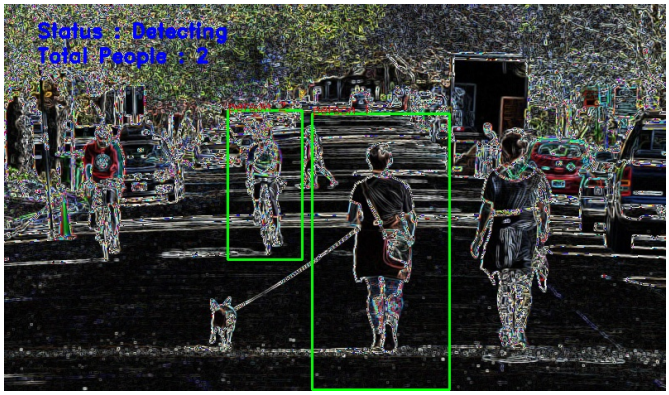

Fig. 1. Processed image using the Prewitt filter previous to the detection of people.

B. *detect*

Received the image, the colormap, the detector (we use HOG, from OpenCV [4], but you can use any other detector) and the changeable parameters, the function scans, shows and returns info about the people scanned. The input parameters are *show* and *confidence*, which is a limit value used to decide whether if the function has detected a person or not (if it is above the limit, the detection is a person). The confidence parameter should be calibrated for every data set so that the program is more precise.

The show option can be turned off just for analysis purposes (shows the image for true, and doesn't for false (it gives you the results of the calculations and the detection, but the image is not shown)) and also the confidence filter can be set at will. If a colormap is sent via the mode parameter, it is applied before scan. After the colormap is applied, the function detects the people and it returns the coordinates of the boxes containing them [5], how many people have been detected, and a weight, that is how sure the program is that the given box coordinates contain a person.

C. humanDetector

The function receives information about what the user wants to do and decides how the detector is applied.
It decides whether to analyse the information from the computer camera (for if you want to study live content), from a pre-recorded video or from a pre-taken image (if you want to study previous-taken footage). Once it knows what wants to be analysed, it uses one of this three functions: *detectByCamera, detectByPathVideo, detectByPathImage*.
To shut down the program (exit the camera/video/image), the "q" key must be used (otherwise, the program will re-open itself again).

D. detectByCamera

The function receives some info relevant for the detection and uses the computer camera as the source images. The function does not receive any input path, but it can receive an output path to store the analysed video *(unfortunately, and for unknown reason, the generated processed video can not be opened once saved)*. The media taken from the camera is anlysed frame-by-frame and the detection is made.
The function may not work for every computer, since it depends on the camera ID assigned by the operating system or the computer may simply not have a camera.

E. detectByPathVideo

The function receives a path to a video and opens it frame by frame to apply the detector. It first takes the information about the video (length in frames, size...), it checks that the video has been opened successfully, and then it starts analysing it.

F. detectByPathImage

The function receives a path to an image and opens it to apply the detector. It is noticeable that a resize is done to the image at 800 pixels, in order to optimize the program (if the received image is a 4K file, if your monitor is not 4K you will not see it, and the program will take very long to process the image).

G. get_colormap

The function receives a colormap name and returns its OpenCV code to be applied to the analysed images. Default case is RGB (same as no colormap).

Fig. 2. Show obtained from a processed video.

H. colormaps

The function is a tool to get all OpenCV colormaps available easily. It returns the name and code together for each colormap for easy future use.

I. compare

The function was used to do some tests for parallel detection. It creates threads of processing and queues them into the processor cores (CPU limited function). In each thread an image with a colormap applied is scanned and lastly all the info from the threads are put together.
For a 4 core CPU it saves around 25%-30% of time for a 22 colormaps run. It also shows performance between threads.
Not intended for showing off
[Potential usage] This function could be used to thread a video scanning and enhance the frames analysed per second by a 25% approx.

J. show_people

The function receives some rectangle coordinates and an image and presents them together.

K. filetype

The function receives the parameters box and retrieves the used file name for info purposes. It establishes whether the received media is a video, an image, or of it is taken with the camera.

L. test

The function prepares the parameter box for the compare threaded test.
Not intended for showing off

M. default_image and default_video

The functions are designed to ease the normal use of the human detector. The user sets up the parameters box with the variables and the program runs an ordinary scan.
They receive a colormap, an input and output path and all the other variables are fixed.

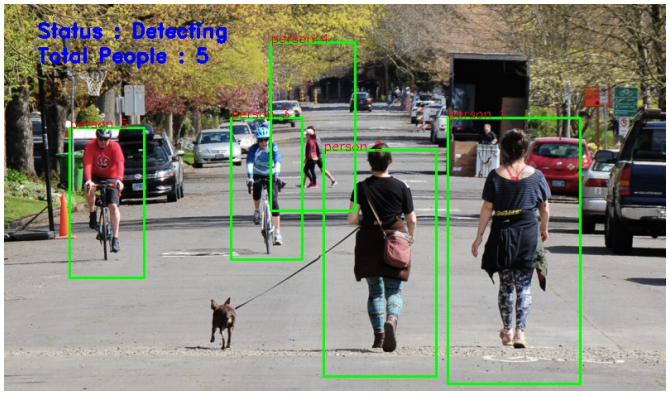

Fig. 3. Processed image using the default mode for the detection of people.

N. box_coordinates_analysis

The function receives the coordinates obtained from the detector and it calculates the center of mass, the virtual ground point, the person's area and the height-width ratio. Once all the information is extracted, the function calculates a number that is used to decide whether if the person has fallen or not.

O. showcase

The function shows the normal processed image, an image with the Prewitt filter applied, and an image with the HSV colormap.
It returns a table with all the data extracted from the analysis of the images.
Also it show a video scan and (if the user wants to give it a try) a camera scan can be shown (we recall the potential malfunction of the camera part due to uncontrollable variables between devs and users).

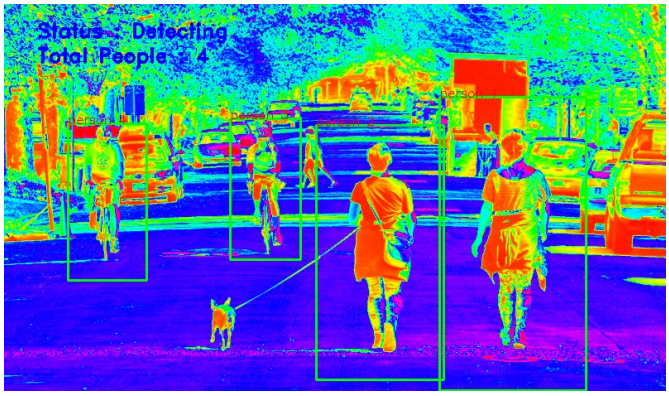

Fig. 4. Processed image using the HSV colormap previous to the detection of people.

V. Main

It is where the functions are called. First, there are some global images, and a timer (used for images showing) which can be set to 0 for unlimited showing (until the user closes it), or can be set to a specific time in milliseconds.
After the timer, HOG from openCV is initialised and it receives the properties we need for this program. Then, the working directory from the program is obtained, in order to open correctly the images (since openCV works with absolute paths).
If the user wants to change the colormap for the analysis, a list of the available colormaps is shown.
Lastly, the program shows the time elapsed for the detection.

VI. Conclusion

The final aim of this paper was to crate a tool to detect whether a person has fallen or not and, as simple as the question seems, the problems and struggles through the creation of the project were numerous. So, in this section we would like to briefly discuss some of these brick walls we encountered:

A. Human Detection

The first problem we faced was to be able to detect a person. Luckily this kind of 'miniproject' has been done by many before. Thus, after some research, we found a human detection project [4] that suited our necessities and introduced ourselves to HOG from OpenCV.

B. Slowness

Once people could be detected, we realised that detection people from a video was non-viable. A 10 second video would take over 30 seconds to be processed (as shown in the showcase) so we started exploring optimizations and after reviewing the time taken by each part of the program we concluded that no direct meaningful optimizations could be applied to the code. The detector takes from 0.2 up to 0.5 seconds per detection (numbers may vary from different computers) and the rest of the code only 0.1 (including displaying the image/video). So, in a perfect case of 0 seconds per code the fps cap sits at 3 to 5 detections per second which would not be marketable at all. Finally a big idea came in: Parallel detections. With parallel detection the program went from 0.5 to 0.1 seconds per detection (with our 8 threaded CPU which means 8 detections at the same time), which caps at 10 fps, and further improvement could only be done with a faster detector or migrating the code to C++ or another low-level programming language. But as good as a solution it was, this created a whole bunch of harder problems to solve thus by now we stuck with the linear detections.

C. Information and Time

Multi-threading solved the speed problem but created an information one. How come the program to know which person is which? Or in a more programming words, how the computer can trespass information between frames? Once the computer has done a detection and goes to another frame, how can we tell the computer that the detected people are the same? HOG detects people in an image but if we wanted to translate it into a video we had to find a way to assign people an ID. We patch this issue with a sorting method which is in itself very bad because if 2 people meet at the same spot the program would probably mix their IDs. This problem has not been yet resolved and one possible solution would be multi-dimensional sorting (although some specific directions would still mix the 2 people IDs).

D. Using different colormaps

Using an RGB representation sometimes made the detector make mistakes. The first way we approached to solve half of this issue was to apply a specific colormap before the detection. We say half of the issue because wrongly detected people were corrected with the application of a colormap but non-detected people were still undetected. Finally we preferred to use a more flexible method and opted to use the weights of the detections. HOG gives the user a confidence value as explained before and tweaking the threshold of the weight we could achieve more precise and better results. Using the showcase images one could see that the default RGB detection has done a better job than the HSV one. Another 21 colormaps were tested in the aforementioned compare function and none gave a better result.

E. Domestic use

Once the program was ready to use we tested it with some domestic footage to see its efficiency. The results were really bad and we had to sit down and see what failed. Finally we reached a verdict and probably a definitive solution. The problem we found was the detector could not detect the person correctly in its domestic situation and we deducted that the problem was not a programming one but a physical one. The test subject was not illuminated enough so the detector was having a hard time detecting it. Thus the solution we found was to extend the program to a multi-camera use. Instead of analysing one angle we could use various cameras to analyse other angles which could have better physical conditions.

F. Conclusion summary

After all the aforementioned struggles, there are still some fields of improvement for the program such as trespassing information between frames or the multi-camera solution for domestic use. Thus more upgrades and optimizations are expected in the future, but right now that progress does not fall to us.

ACKNOWLEDGMENTS

The authors would like to express sincere thanks to Artur Carnicer, a Professor at Universitat de Barcelona, for his advice during this research work. The authors also would like to thank reviewers for all valuable suggestions and comments for improving this project, and both GeeksforGeeks [6] and Stackoverflow [7] for its work in the programming community, which has been very helpful for this project.

REFERENCES

[1] Reducible. “Pagerank: A trillion dollar algorithm.” (), [Online]. Available: https://www.youtube.com/watch?v=JGQe4kiPnrU.

[2] R. Agarwal. “Towncentrexvid.” (), [Online]. Available: https://www.youtube.com/watch?v=fSo7Lk83uJU&t=193s.

[3] J. Maus/BikePortland. (), [Online]. Available: https://bikeportland.org/wp-content/uploads/2020/04/Screen-Shot-2020-04-23-at-1.16.32-PM.jpg.

[4] D. Flair. “Python project – real-time human detection counting.” (), [Online]. Available: https://data-flair.training/blogs/python-project-real-time-human-detection-counting/.

[5] R. Agarwal. “Pedestrian detection and count for certain number of frames in a video.” (), [Online]. Available: https://medium.com/analytics-vidhya/pedestrian-detection-and-count-for-certain-number-of-frames-in-a-video-f6563309ef78.

[6] “Geeksforgeeks.” (), [Online]. Available: https://www.geeksforgeeks.org/.

[7] “Stackoverflow.” (), [Online]. Available: https://stackoverflow.com/.