Real-time Social Distancing Detector

Lorenzo Agnolucci
lorenzo.agnolucci@stud.unifi.it

Federico Nocentini
federico.nocentini@stud.unifi.it

Abstract

Until the development of a vaccine social distancing is the most effective way to try to reduce the spread of COVID-19. Computer vision can help in verifying the respect of the rules established by the authorities.

The aim of this work is to present a tool to monitor people's compliance to social distancing in crowded places in real-time. Our approach is based on YOLOv3 neural network to detect pedestrian in a video stream and on homography to get the bird view of a user-selected ROI. From people's position in the bird view we measure the pairwise distance and then we show the results.

1. Introduction

The global spread of COVID-19 forced the scientific community to find ways to reduce new infections while waiting for the development of a vaccine. Social distancing is claimed as the best spread stopper in the present scenario as it aims at reducing the physical contact between possibly infected individuals and healthy people. As per the WHO norms [10] it is prescribed that people should maintain at least 1 meter of distance among each other in order to follow social distancing, however some countries recommends an even longer distance [7] (the more the better).

Pedestrian detection is a very actively researched task in the computer vision community, both in academia and industry. It has applications in many different domains, including robotics and autonomous vehicles, entertainment, and smart surveillance. Pedestrian detection also plays a critical role in various other computer vision research areas, such as multi-object tracking, human pose estimation and person identification. Pedestrian detection is a very challenging problem due to huge variations in pedestrian appearance arising from their scale, pose, illumination, surroundings, occlusion and the presence of confounders (*e.g.* advertisements, reflections). Occlusion and small-scale are the two most important challenges limiting the performance of current detectors, and they are responsible for most failures.

Therefore the monitoring of social distancing in crowded places can be seen as a pedestrian detection task in which the distance between the detected pedestrians must be measured. In this work we present a real-time social distancing detector based on deep learning to detect social distancing violations in several types of video streams. Our implementation requires an initial user input and then shows the results in real-time in an intuitive way. Our detector could be used to monitor people's respect of social distancing using the video stream of public or private CCTV footage processed remotely by a more powerful GPU-provided machine.

2. Proposed approach

Our first approach was to perform human pose estimation with OpenPose [3] to get the accurate position of people's feet. Unfortunately OpenPose proved to be too computationally expensive for our machine. Indeed we developed our project on a laptop MSI Prestige 15 with Ubuntu 18.04, 16GB RAM and a GPU NVIDIA GeForce GTX 1650 Max-Q with 4GB. The only way to get an acceptable number of FPS (*e.g.* 10) was to reduce the input resolution at the minimum, but this had the downside that the results of the pose estimation were unusable for our purposes. So we decided to use a faster solution with a YOLO neural network to perform a simple object detection and approximate people's feet from the bounding boxes.

The project was developed in Python and the neural network runs with the `OpenCV` deep learning module on the GPU.

2.1. Workflow

The following steps compose the workflow of our work:

1. Take in input a video stream among the available ones with the GUI

2. If the camera of the stream is available calibrate it and save its parameters

3. From the first frame of the stream take in input 6 mouse points: the first 4 for the Region Of Interest (ROI) and homography, and the last 2 for the distance threshold

4. Compute the homography matrix and transform the ROI into the bird view

5. For each frame detect people with the neural network, transform their position with the homography matrix and compute distances

6. Show in real-time the social distancing violations in the input video stream and the corresponding bird view

2.2. Stream sources

Our social distancing detector is designed to work with several types of streams:

1. Laptop webcam

2. IP webcam (or alternatively a smartphone camera using `IP Webcam` app [5])

3. Local video

4. Link to a video stream (the supported formats are those of *Streamlink* Python library [1])

In our case we performed the camera calibration (as described in section 2.3) only for the first two types of streams because the source (*i.e.* the camera) of the video stream was available, while for the last two types we skipped the calibration step because we did not have access to the camera. Depending on the type of the camera of the stream skipping the calibration step can still lead to acceptable results.

To make the choice of the input video stream more user-friendly we developed a basic GUI that allows to select the type of the stream and to insert the source and the optional camera calibration matrices.

When using the last two types of stream sources we noticed a considerable decrease in performance caused by the video compression and frame decoding. Indeed the `OpenCV` function to read and decode the next frame in a video stream is a blocking operation, so if it is used in the same thread that processes the frames it unavoidably slows the execution. This problem is less evident with the first two types of streams due to the more direct nature of those streams compared to for example a local compressed video, so the decoding of the frames is more straightforward. The solution we adopted was to implement a class that runs on a different thread with respect to the main one reading and decoding the frames, and that later puts them in a FIFO queue. At the same time the main thread takes each frame from the queue and process it. In this way we achieved a noticeable increase in terms of frames processing rate and an overall decrease of latency.

2.3. Camera calibration

Camera calibration is a necessary step in 3D computer vision to use a camera effectively as a quantitative sensor. However the distortion phenomena associated to an uncalibrated camera are more evident in a stereo camera, while in a modern regular camera they are almost negligible. In the computer vision context, camera calibration consists of finding the mapping between the 3D space and the camera plane. This mapping can be separated in two different transformation: first, the displacement between the origin of 3D space and the camera coordinate system, which forms the external calibration parameters (3D rotation and translation), and second the mapping between 3D points in space and 2D points on the camera plane in the camera coordinate system, which forms the internal camera calibration parameters. So the goal of camera calibration is to find the camera internal and external parameters.

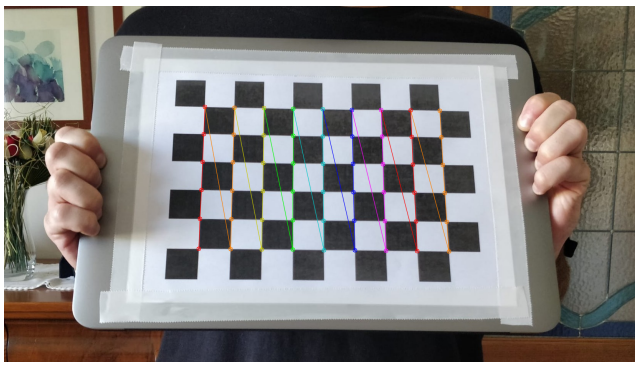

Figure 1. Example of image used for the camera calibration

For the calibration are needed at least 20 photos of a 2D chessboard from various angles and distances, taken by the same camera that will constitute the stream source of the social distancing detector camera. `OpenCV` provides a built-in function to find a chessboard in an image and obtain the coordinates of the corners. These 2D image points can be mapped to 3D object points corresponding to the 3D locations of points on the chessboard. This can be done because the chessboard dimensions and pattern in the real-world are known. This process allows to compute the camera internal and external parameters that can be saved as a YAML

file for later use.

Figure 1 on the preceding page shows an example of an image used for the camera calibration. The chessboard corners used for the mapping between 3D and 2D points are highlighted.

2.4. Bird view

The transformation from the ROI ground plane to the corresponding bird view relies on homography. An homography is a perspective transformation of a plane, that is, a reprojection of a plane from one camera into a different camera view, subject to change in the translation (position) and rotation (orientation) of the camera. In other words, given the 3×3 homography matrix and 4 (or more) points belonging to the same plane taken from a certain camera frame it is possible to see those same points from a different camera view.

The first 4 mouse points taken in input represent the ROI in which the social distancing is monitored. These points need to mark a rectangle in the 3D world to avoid distortions. First we compute the aspect ratio as described in [11], estimating the camera focal length and then evaluating the aspect ratio of the rectangle in the 3D world from its camera projection. Then we find the homography matrix that maps the input ROI points into a rectangle with the corresponding aspect ratio. This rectangle represent the bird view that allows to measure the distance between each person.

The last 2 input points represent 1 meter in the real world. We project these points on the bird view with the homography matrix and with the euclidean distance between these points we get the measure unit of the bird view. Multiplying this value for 2 we obtain the threshold distance of 2 meters under which there is a social distancing violation.

In figure 2 can be seen how the ROI is transformed into the bird view. A warped rectangle in the camera projection becomes a rectified rectangle as seen from a top view.

(a)

(b)

Figure 2. (a): 4 ROI points in the input video stream, (b): Bird view of the ROI with also the last 2 mouse points

2.5. People detection with YOLOv3

To perform people detection for each frame we chose the neural network YOLOv3 due to its speed and the consequent real-time detection suitability.

As the name suggests the "You Only Look Once," or YOLO, family of models can predict the type and location of an object by looking only once at the image. YOLO was first introduced in [9] and was designed for fast object detection at the expense of a generally lower accuracy with respect to Region-Based Convolutional Neural Networks (R-CNNs).

YOLO considers the object detection problem as a regression task instead of a classification one, taking a given input image and simultaneously learning bounding box coordinates and corresponding class label probabilities. The approach involves a single deep Convolutional Neural Network that splits the input into a grid of cells and each cell directly predicts a bounding box and object classification. The result is a large number of candidate bounding boxes that are consolidated into a final prediction by applying non-maximal suppression, that removes the duplicate detections for the same object that have lower confidence.

YOLO has undergone several variations in the years following its release: the first version proposed the general architecture, whereas the second one refined the design and made use of predefined anchor boxes to improve bounding box proposal, and version three further refined the model architecture and training process. [8] [4]

In this work we used an open-source YOLOv3 model pre-trained on COCO dataset [6]. Despite the fact that the network was able to detect all the 80 COCO classes, for our purposes the "person" class was the only useful

Figure 3. Usage example on a local video

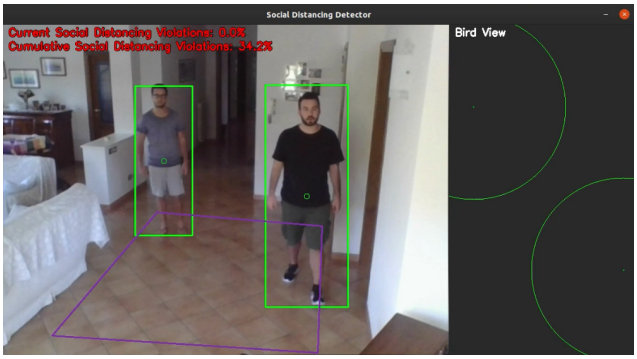

Figure 4. Usage example on a webcam stream

one. The model takes in input a video frame and for each result (*i.e.* detected person) it outputs the bounding box, the centroid coordinates and the associated confidence. To clean up the output we apply a simple heuristic based on a real-life assumption, that is humans being taller rather than they are wide, so we do not consider bounding boxes with the width greater than the height. Given the network output we decided to identify the feet of each person with the bottom-center point of the corresponding bounding box. This is obviously an approximation but without the human pose estimation there is not a considerably more accurate way.

We map the feet position of each person to the bird view using the homography matrix. We assume that at least every person in the ROI is standing on the same flat ground plane. We chose to consider only the ROI to allow the possibility of people outside of it (which are consequently mapped outside the bird view) being on a different plane. A possible alternative would have been to measure the social distancing in the whole image, thus increasing the considered area at the cost of possibly include not entirely accurate results in presence of different ground planes. For each detected person we compute the distance to all the others in the bird view, and we draw its bounding box with the corresponding color:

- Red: there is at least one person closer than the distance threshold
- Green: there are no social distancing violations
- Yellow: the person is outside the ROI so we do not consider it

Besides the input video stream with the colored bounding boxes we show the bird view, in which we draw a point with the corresponding color for each person in the position given by the homography matrix. Since a yellow bounding box means that the respective person is outside the ROI, the bird view does not contain yellow points because it represents only the ROI given in input. For each person represented in the bird view we draw a circle centered in its point with a radius equal to half the distance threshold. In this way when two circles intersect in the bird view it means that there is a social distancing violation.

Both in the input video stream and in the bird view we also draw a red line that connects respectively the centroids or the points of the people who are closer than the distance threshold.

To provide a quantitative measure of people's compliance to social distancing we compute some statistics from the video stream. In particular we measure the percentage of people violating social distancing with

regard to the total of people in the ROI, both for the current frame and for all the frames since the beginning of the stream put together. These statistics are displayed in the detector window.

Figure 3 on the preceding page and figure 4 on the previous page show usage examples of the detector: in particular the first one represents its application to a local video of a crowded public road (Oxford Town Center Dataset [2]) without camera calibration, the second one corresponds to a calibrated laptop webcam stream.

3. Conclusion and future work

In this work we presented a tool to monitor people compliance to social distancing in crowded places in real-time. Our social distancing detector needs an minimal initial user input and then relies on homography to get the bird view associated to the ROI and on a neural network for people detection. The results are shown in an intuitive way both on the input video stream and on the bird view.

As a future work it would be interesting to use a machine powerful enough to perform pose estimation and not only people detection to achieve more accurate results. Also, a calibrated IP webcam could be placed in a crowded street or square to fully test our social distancing detector in a real-world scenario.

3.1. Resources

Code and further details available at `https://github.com/LorenzoAgnolucci/SocialDistancingDetector`

References

[1] Streamlink, Github. `https://streamlink.github.io/plugin_matrix.html`.

[2] B. Benfold and I. Reid. Stable multi-target tracking in real-time surveillance video. In *CVPR 2011*, pages 3457–3464. IEEE, 2011.

[3] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[4] J. Hui. Real-time object detection with yolo, yolov2 and now yolov3. `https://medium.com/@jonathan_hui`.

[5] P. Khlebovich. IP Webcam. `https://play.google.com/store/apps/details?id=com.pas.webcam&hl=it`.

[6] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[7] National Health Service (NHS). Social distancing: what you need to do. `https://www.nhs.uk/conditions/coronavirus-covid-19/social-distancing/what-you-need-to-do/`.

[8] N. S. Punn, S. K. Sonbhadra, and S. Agarwal. Monitoring covid-19 social distancing with person detection and tracking via fine-tuned yolo v3 and deepsort techniques, 2020.

[9] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[10] World Health Organization (WHO). Covid-19: physical distancing. `https://www.who.int/westernpacific/emergencies/covid-19/information/physical-distancing`.

[11] Z. Zhang and L.-W. He. Whiteboard scanning and image enhancement. *Digital Signal Processing*, 17(2):414 – 432, 2007.