Assignment 2 Computer Vision course

Spadetto Matteo[214352]

Universita degli Studi di Trento, via Sommarive 9, IT
matteo.spadetto-1@studenti.unitn.it

Abstract. The goal of this report is to provide explain the implementation of the detecting and tracking algorithms used in the *assignment2* code, written in *C++*.

Keywords: Object detection · Object tracking · people.

1 Introduction to people detection and tracking

The goal of this software is to detect and track people in a video chosen from the *MOT challenge*. This video contains some critical problems:

- The video is recording a 3D world so people have different sizes, the nearest are bigger than the ones far away from the camera;
- In the scene there is a light pole with a road sign so it is difficult to look behind it;
- People are moving and so also their legs, generating a bigger bounding box due to the legs section;
- The background is not static everywhere. In the center part there is a security tape moved by the wind;
- In the scene there are windows that generate mirror effects;
- The are occlusions by the fact that people cross their selves;
- Sometimes people walk very near one to each other for a long period of time.

Due to all these problems the source code includes different algorithms in order to remove the problematic features and increasing the weight of the good ones.

2 Methods used

The video is provided as a set of frames so the first step is to create a vector of frames, a video. To do this a frame structure containing all the information needed is used:

- Img: the frame itself;
- Frame id: the id of the frame that is under processing;
- People: the vector of people that are in the current frame. This vector is composed by objects of the type described in the following structure.

For each person it is implemented another structure containing all the object information:

- Id: the id of the person;
- CenterX: the X coordinate of the person center of mass;
- CenterY: the Y coordinate of the person center of mass;
- Bb_left: the person rectangle bounding box left point;
- Bb_top: the person rectangle bounding box top point;
- Bb_width: the person rectangle bounding box width;
- Bb_height: the person rectangle bounding box height;
- Traj: the person trajectory.

The second step is the conversion of the frames from the *RGB* scale to another one, more suitable for processing. Different color scales was tested in order to find the better one for this specific application. The choice is relapsed on the *YUV* color scale because the *HSV* and *gray* ones was to noising for the following steps.

Then a very light *gaussian blur* is performed in order to smooth the noise. This process is not fundamental but provides a better result at the end.

After this initial phase the real detection part starts. As first thing a background subtraction model is required and the choice is relapsed on the *MOG2* algorithm.

Then some morphological erosion and dilation are applied in order to remove all the noise and marge the real objects. Until now the code goal was to create a good base for the main part of the detector.

The detection is based on contours so it is possible to see in the output video, not only the rectangle bounding boxes needed for the requirements of the assignment, but also the real contour of each person. In this part, is used a Canny filter in order to detect the edges in the scene and then the *"findContours"* algorithm to create the polygons of each contour. The contours are then modified in order to reach a better result: polygons approximation and a repair of the convex hulls are applied. Reached this point the polygons are almost good, so, if they have a reasonable perimeter, they are filled and morphological erosion and dilation are applied to repair the undesired holes. In this part are not yet generated the contours of each person but only a better representation of them.

Now, with the good represented object it is possible to draw their contours. The previous part is repeated in order to find the people outer polygon but, additionally, also the rectangle bounding boxes and their centers are generated and the data required for the assignment are stored in the previously defined structure. In this part also the occlusion problem finds a solution. The software looks on the 10 previous frames and if a person, in the previous frames, not only is near the current one, but also if their trajectories and bounding boxes dimensions are almost the same, then the id is maintained. For this reason in the initial structure there is a not required trajectory parameter. If these conditions are not respected there is a new person in the scene so a new id is generated.

The contours drawing part is performed in black and white range, so to generate the output video (in the *RGB* scale) a conversion of the contours to *RGB* is applied and then merged with the input video.

The final part is focused on drawing the trajectories and storing the results. The first part is performed locking at the video vector and drawing, for each person, the last 20 center points from the current one. The second task, instead, is to generate the output video in *.avi* format and store the video vector data in two files: *"detection.txt"* and *"tracking.txt"*.

3 Achieved results

People are almost well detected but some problems persists, however they have some possible solutions.

Due to the fact that some people are very near to each others the dilation part should have gradual parameters to not merge people near another one but still create a good object to detect.

People behind the light pole, mainly in the initial part, are not detect and tracked. A possible solution is to use a *Kalman filter* that predicts the following step and generate a fake bounding box around the center proposed by the *Kalman* filter.

Due to the fact that erosion is not gradual on the depth plane, some objects very far away are not detected until they reach a certain perimeter contour length.

The software was compiled and ran on a *ThinkPad T440p with an Intel® Core™ i5-4200M CPU @ 2.50GHz* 4 with 8Gb of *RAM*. In its first version it required to much *RAM* so a second optimized version was developed passing from 5.6Gb of required *RAM* to 3.2Gb.

This software works on a saved video so for this reason it was implemented as an *offline tracker* (each step explained before is performed for each frame of the video and then passing to the next task) but it is implemented in such a way that it permits to set it as an *online tracker* (all steps are performed for a frame and than it will compute the next frame). The possibility of making the software a *real-time* tracker finds its bottleneck on the *MOG2* computation (but it is still possible with a good hardware or less precision).

All references used in this assignment ([2], [1]) are listed below and the output video is at the following **link**.

References

1. Computer vision course slides and codes. https://didatticaonline.unitn.it/dol/course/view.php?id=16045, last accessed 22/12/19
2. Opencv c++. https://opencv.org/, last accessed 22/12/19