People detection and tracking on video

Author:

Mateusz Szumilas

Supervisor: Ing. Vitěslav Beran, Ph.D.

Brno, May 2019

Contents

Abstract

This work will show and describe the algorithms used for detection and tracking objects on films without the use of neural networks. The work presents algorithms such as AdaBoost, an algorithm that creates one strong and many haul-cascades, to detect through predefined patterns. At work, we do not experience algorithms in which neural networks exist. In this work I would like to show algorithms that are less popular, however, it is worth looking at them for the sake of history. Currently, other, much faster and more accurate algorithms are often used, but it is worth looking at it from the perspective of time and seeing how earlier such have been dealt with.

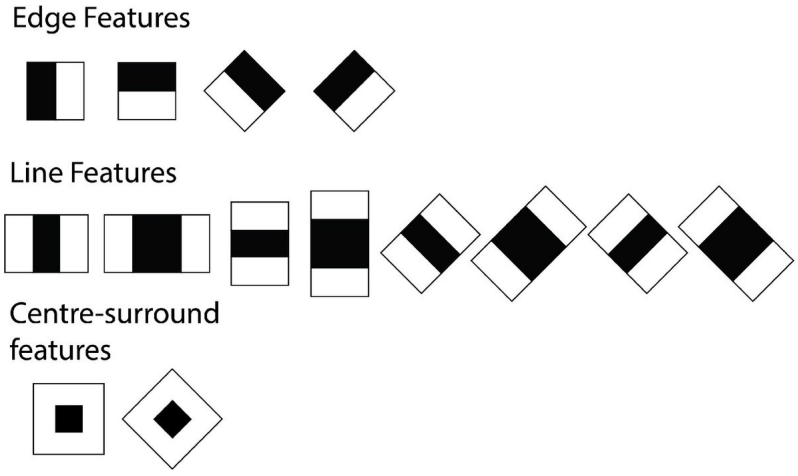
Chapter 1

Haar-Feature

1.1 Detecting by Haar-Feature

1.1.1 Description

To detect by this method we used a set of classifiers. Which are they looking for so-called Haar's features. These features contain information about changing the contrast value between rectangular groups. These features can be easily scaled and rotatable. To speed up the calculation, a matrix is created from pixels, where each field is the sum of all values from the left and over the pixel. As a result, we always do only 4 operations while checking the classifier. What is the complexity of O(1).



1.1.2 Description of the action

This classifier distributes pixels to black and white rectangles. It then checks if the area is an edge, a line or a central point. Next, the substraction of the sums of bright and dark pixels is calculated and determines whether the given field matches the pattern.

$$\Delta = dark - white = \frac{1}{n}\sum_{dark}^{n} I(x) - \frac{1}{n}\sum_{white}^{n} I(x)$$

Where:

- Dark means all pixels in black part of classifier.
- White means all pixels in white part of classifier.
- n means number of pixels
- I(x) means value of this pixel

In perfect case Δ will be equal 1

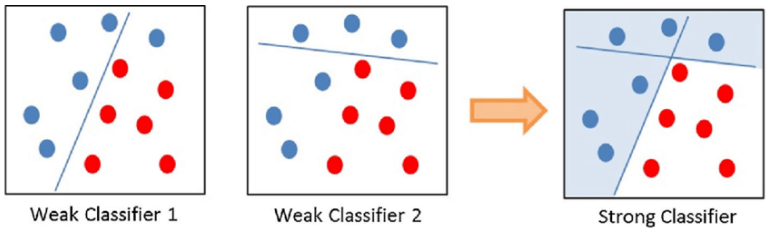
Comment 1.1.1. In Haar-cascade we are using a hundreds and even thousands of classifiers. Most often combined into one using AdaBoost.

Chapter 2

AdaBoost

2.1 AdaBoost

AdaBoost is an algorithm that has to do from many weak classifiers, create one arbitrarily strong classifier. The algorithm in k-th iteration checks k weak classifiers on a set of examples D with variable weights



Definition 2.1.1. " The "weaker" classifier is a relatively simple classifier, with a very low expression power, able to classify test data with an efficiency greater than 50% "[4]

2.1.1 Algorithm

1. The input algorithm receives the training set (x_1, y_1),(x_2, y_2),...,(x_n, y_n), where each x_i should perform the appropriate X problem (e.g. detecting a car), and each label (decision) y_i should be made a certain set of Y. Let us assume that every y_n from Y $\in \pm 1$. Let us mark our set of classifiers K $(f_1, \ldots, f_K)$. to our weaker classifications.

2. At the beginning of the algorithm's operation, all weights in this algorithm are equal. Let $(d_1, \ldots, d_n)$ will be the set of weights for $(x_1, \ldots, x_n)$, respectively. At the beginning $d_i = \frac{1}{n}$, where i = 1, . . . ,n. We also remember the condition that:

$$\sum_{j=1}^{n} d_j^{(m)} = 1$$

3. AdaBoost calls the selected "weak" learning algorithm. We assume that the error of the classifiers obtained on the training set is less than $\frac{1}{2}$. So we teach the given algorithm f_m on X.

4. Then we calculate the received weighted error of the learning algorithm. We use the following formula for this:

$$err_m = \sum_{j=1}^{n} d_j^{(m)}[f_m(x_j) == y_j]$$

The task of the "weak" learning algorithm is to build a hypothesis f_m: X→Y appropriate for the current distribution D_t.

5. Then we calculate the weight of the classifier. It is interesting that if the classifier wrongly classifies an example, the badly classified by its predecessors then it will get a worse rating

$$w_m = \frac{1}{2} \cdot \log(\frac{1 - err_m}{err_m})$$

6. When AdaBoost gets the classifier f_m, the w_k parameter is selected. Intuitively, w_k is responsible for the importance we attach to the classifier f_m. Note that $w_k \geq 0$ when $err_k \leq \frac{1}{2}$. In addition, w_k grows when err_k decreases.

$$d_j^{(k)} = \begin{cases} d_j^{(k)} e^{-w_k}, if f_k(x_j) = y_j \\ d_j^{(k)} e^{-w_k}, if f_k(x_j) \neq y_j \end{cases}$$

7. The distribution of D_t is then changed to increase (decrease) the weight of the training set elements that are bad (well) classified by f_m. Hence, weights tend

to focus on "difficult" examples. We must also remember about the first condition. So we have to normalize the result:

$$s_k = \sum_{j=1}^{n} d_j^{(m)}$$

$$d_j^{(k+1)} = \frac{d_j^{(k)}}{s_k} \quad j = (1, \ldots, n)$$

2.2 Lucas-Kanade method

2.2.1 Optical flow

Optical flow is a set of vectors describing the movement of an object on subsequent images. We distinguish three main methods of finding optical flow: - gradient method - frequency method - correlation method The biggest problem with optical flow is lighting, which very much affects the results. One of the most popular algorithms is the Lucas-Kanade algorithm.

2.2.2 Method LK or optical flow

It is a method of optical flow, which means that it looks for a pattern of apparent motion. This method involves examining the gradient between consecutive images. He also assumes that the difference between the images is small and that the intensity of the pixels does not change. Unfortunately, but the pattern has more unknowns than equations. What makes it impossible for us to solve it, fortunately is the way to do it.

$$I(x, y, t) = I(x + \delta_x, y + \delta_y, t + \delta_t)$$

After transforming this formula from the right with the use of Taylor's theorem and after its over-conversion, we get the following pattern:

$$\frac{\Delta I}{\Delta x} v_x + \frac{\Delta I}{\Delta y} v_y + \frac{\Delta I}{\Delta t} = 0$$

Where I'_x, I'_y, I'_t are the gradient of rank in each direct, and v_x and v_y are values of moving in a given direction. Unfortunately, we still have two unknowns, which makes it impossible for us to calculate an unambiguous value. However, if we add the additional, that the neighborhood of each pixel does not change, it allows us to store subsequent points, then using the least-squares method, we are able to obtain the following formula:

$$Av = b \quad where:$$

$$A = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \dots & \dots \\ I_x(p_n) & I_y(p_n) \end{bmatrix}, \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad b = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ \dots \\ -I_x(p_1) \end{bmatrix}$$

Using the method of least squares we obtain the following formula:

$$V = (A^T A)^{-1} A^T b$$

Chapter 3

Implemetation

3.1 Implementation

To solve the problem I used the Python programming language, and the OpenCV library which greatly facilitated the work on the project.

3.1.1 Functions

The program has two implemated functions:
- check_collision (A, B)
This function is responsible for checking if the rectangle B is in the rectangle A. To make the function works correctly, give the following four: (X, Y, W, H) where X and Y are the x and y coordinates of the upper left upper edge. W is the width of the rectangle and H is the height of this rectangle.
- multi_rects (BArr)
This function is responsible for checking whether in a given set of rectangles there are not some rectangles inside them, it uses the function check_collision (A, B)

3.1.2 Variables

- font

  This variable include information about font from cv2 used to write text over

rectangles
Used: font = cv2.FONT_HERSHEY_SIMPLEX

- person_cascade
  Cascade classifier class for object detection load from file.
  Cascade classifier based on Haar Feature.
  Used: cv2.CascadeClassifier('.//data//haarcascade_fullbody.xml')

- cap
  Cap is a object of slass for video capturing from video files, image sequences or cameras. Parametr is a path to video file or image sequence which we will actually use.
  Used: cv2.VideoCapture('.//test.mp4')

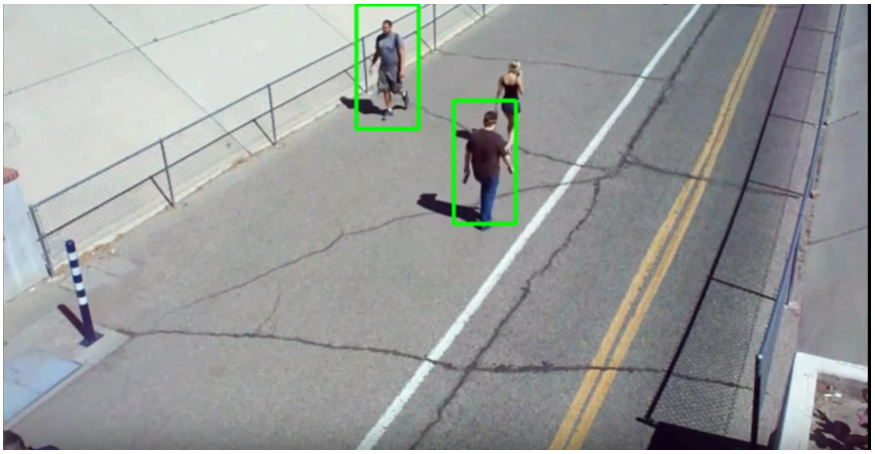
- arrays
  - bbox_arr[]
    It is array which contains all rectangles currently tracking and detected people.
  - tracker_arr[]
    It is array containing all trackers.
  - color[]
    This array includes color respectively to trackers.
  - rects[]
    Something like buffor, includes all detected object on currently frame.

3.1.3 Main loop and detection

We are beginning from clear array with rects, and select counter of frames on 0. Then we read next frame by cap.read() to variable frame, and errors to r. Next step is resize frame to lower resolution and change colors in frame in the gray scale.
When we did it we are using method detectMultiScale() which detects objects of

different sizes in the input image. The detected objects are returned as a list of rectangles. This function use Haar-cascade on frame.

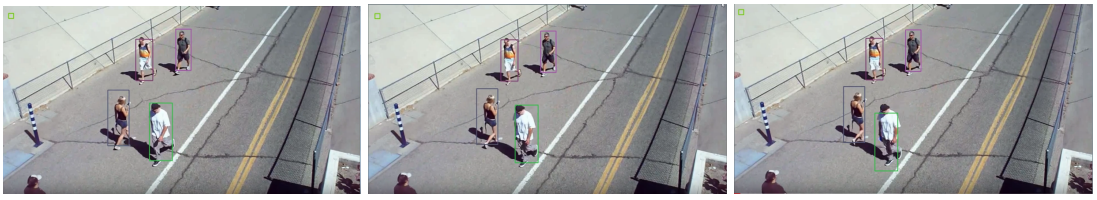
If something is detected we start tracking it.



3.1.4 Tracking

First what we have to do is check if some object is not detected multiply. If not then we are adding it to tracking, then we check nothing in our set of tracking object is not tracking twice or more. if is then erase it. When we over it, we create trackers for not tracking objects.
After this we are updating the tracker, find the new most likely bounding box for the target. If some object came after frame border then we erase this tracker and border.

Last step is show the frame with rectangles. And every 22th frames we are detecting all object again.



Summary

The solutions used are not very effective and slow, but I think that it is worth to know them because of the influence of stories and interesting solutions.

Bibliography

[1] Adam Dąbrowski Damian Jackowski, Tomasz Marciniak. *IMPLEMENTACJA I ANALIZA MODELI DETEKCJI I ŚLEDZENIA OBIEKTÓW W SEKWENCJACH WIDEO Z ZASTOSOWANIEM MODUŁU Z PROCESOREM SYGNAŁOWYM*. Foundation of Computer Science, January 2010.

[2] dr inż. Michał Grochowski. Rozpoznawanie wzorców i twarzy.

[3] Łukasz Kucharczyk. Wykrywanie i śledzenie twarzy na obrazie z kamery, 2011.

[4] Damian Łoziński. Zespoły klasyfikatorów, 2008.

[5] Miron Moderau. Sterowanie animacją przy użyciu kamery internetowej, 2011/2012.

[6] Dhara Patel. *Optical Flow Measurement using Lucas kanade Method*. Foundation of Computer Science, January 2013.

[7] Krzysztof Sopyla. Przetwarzanie obrazu z wykorzystaniem splotu funkcji.

[8] Piotr Wilkowski. Wykorzystanie algorytmu detekcji i lokalizacji w zadaniu chwytania, 2009.