Implementation Of The Viola Jones Algorithm For Face Detection And Tracking On Robots With Limited Computation Power

Dimitar Bezhanovski
Ss. Cyril and Methodius University
Faculty of Computer Science and Engineering
Skopje, North Macedonia
dimitar.bezhanovski@students.finki.ukim.mk

Aleksandar Stankovski
Ss. Cyril and Methodius University
Faculty of Computer Science and Engineering
Skopje, North Macedonia
aleksandar.stankovski@students.finki.ukim.mk

***Abstract*—This paper aims at presenting how to implement face detection and tracking on a low powered robot that uses hardware such as the Raspberry Pi using the Viola Jones algorithm. With face detection and tracking implemented, the robot can follow the movement of a person, and move accordingly.**

I. Introduction

This paper aims to explain how the challenges of face detection and tracking with robots that have low processing power can be overcome. Face detection in image processing is a computer technology that is able to identify the presence of people's faces within digital images or videos, and it is not to be confused with face recognition, which refers to the ability to distinguish and recognize the person in the image. The main challenge that we face is the face detection part. For a mobile robot (robot with wheels or caterpillars) to be able to move towards a person and also to orient towards them, first we need a method by which we will detect the person's face. Even though the whole human figure can be detected, in this research paper, we focus on face detection because the face of a person always has the same order of characteristics. In contrast, the whole human figure can change depending of the movement or pose that the person is in, such as stretched arms or bent knees.

When it comes to face detection, we do not have a huge choice to solve this problem. A widely used approach to overcome this challenge is by applying deep learning technique, but in our case, there is a problem with that. Deep learning is too demanding in terms of computing power for devices like the Raspberry Pi. Deep learning would be an adequate approach if the embedded computer used has built-in CUDA cores. Such embedded computers can cost more than 100 USD, but this paper focuses on devices that cost less than 50 USD, as per example Raspberry Pi devices and similar variants. The purpose of this is to demonstrate a way for students to be able to implement tracking functionality in robots that are built with low cost. This research explains an implementation that is a good compromise between performance and functionality using Viola-Jones face detection algorithm. Once the face is detected we apply an algorithm to calculate the distance and location of a person in relation to the robot. The code for this paper running on the Ricardo robot can be found on the following repository https://github.com/dimitarbez/Ricardo

II. Related Work

Many existing works use neural networks to implement the feature of user following [4]. Other works use a Bayesian based approach for facial recognition and following using SIFT descriptors [5]. This is effective however it is a problem when implementing such functionality on a device that doesn't have a dedicated graphics processor such as the Raspberry Pi.

III. Viola Jones Algorithm

The Viola-Jones object detection algorithm [1] is an object detection algorithm that was proposed by Paul Viola and Michael Jones in 2001. Although it can be trained on various classes of objects, the main motivation of the algorithm was detection of faces.

The Viola-Jones algorithm uses a machine learning approach to detect objects in pictures. The classifier works with very simple image features. One part of the image is subtracted from another one. That alone is not very good. But if there are thousand such features, then we are well on our way to detection.

When we have a black and white image, the eye area is darker than the forehead area. When we subtract the lighter part from the darker part, we get classification. The algorithm works on this simple principle. Viola-Jones requires a full front-facing view. So in order for the face to be detected it will have to be aimed at the camera and it should not be tilted to one side. While this sounds like something that would reduce the usefulness of the algorithm, in practice these limitations are quite acceptable.

The features found by the algorithm involve a sum of pixels within the boundaries of rectangular regions. As such, they have similarities to the Haar basis functions, which have been used to detect objects in an image. But because the features used in Viola-Jones depend on more than one rectangular region, they are generally more complex.

This algorithm consists of fours stages:

1) Haar features.
2) Create an integral image
3) AdaBoost training
4) Cascading

A. Haar features

The detection procedure classifies based on values from simple features. There are several motivations for using features instead of working with pixels directly. The most common reason is because with features knowledge can be encoded in ad hoc domain. There is another critical reason and that is that a feature-based system is much faster than a pixel-based system. The simple features used are reminiscent

of Haar basis functions. The algorithm uses three types of features. The value of a two-rectangular feature is the difference between the sum of the pixels in the two regions. The regions have the same size and form and are vertically or horizontally adjacent. Feature with three rectangles calculates the sum of the two outer rectangles subtracted from the middle rectangle. Finally, a feature with four rectangles calculates the difference between diagonal pairs of rectangles.

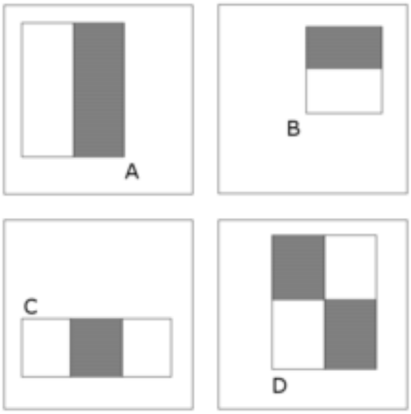

Fig. 1. Haar features [1].

B. Integral image

Rectangular features can be computed very quickly using an image that is called an integral image. The integral image at location x,y contains all the pixels above and left, inclusively.

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y')$$

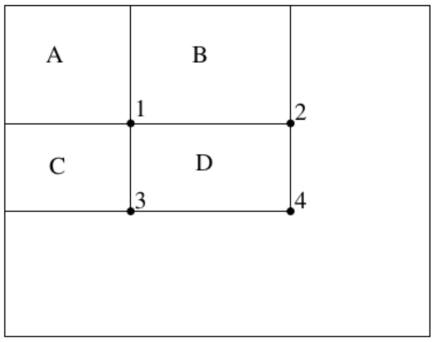

Fig. 2. Calculating the area with the integral image [1].

ii(x,y) is the integral image and i(x,y) is the original image. Using the following formulas:

$$s(x,y) = s(x,y-1) + i(x,y)$$
$$ii(x,y) = ii(x-1,y) + s(x,y)$$

(where s(x,y) is the cumulative sum of the row, s(x,-1) = 0) the integral image can be calculated with just one iteration across the pixels of the original image. Using the integral image, any rectangular sum can be calculated using four references to an array. The difference between two rectangular sums can be calculated in eight references. Because two rectangular features above involve neighboring rectangles, they can be calculated in six references to an array, eight in case of three rectangle features and nine in four rectangle features. With this, the calculation complexity of the sum of the rectangular regions becomes of O(1). Calculation of the integral image has to be done only once per new frame and its complexity is O(n). This is way better than having complexity of O(n^2) for the entire image.

C. Cascades with attention

This section explains the algorithm for constructing a cascade of classifiers that manages to increase detection performance but drastically reduce the computation time. A smaller and more effective classifier can be constructed that will reject many of the negative sub-windows while at the same time it will detect almost all positive instances. Simpler classifiers are used to reject most of the sub-windows, before more complex classifiers are used in order to obtain a low negative rate. The general form of the detection process is that of the degenerative decision tree, which we call cascade. A positive result from the first classifier provokes the evaluation of the second classifier which is also set to manage to reach very high rates of detection. A positive result from the second classifier causes the third classifier and so on. A negative result in any step is followed with the rejection of sub-window.

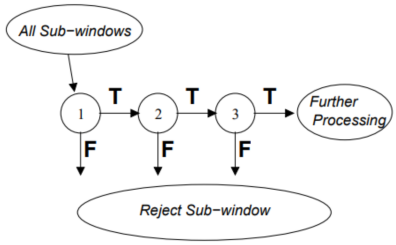

Fig. 3. Processing of sub-windows [1].

The cascade tracks are constructed by training a classifier using AdaBoost and then changing the threshold values to minimize false negatives. The default threshold values is designed to have a low error rate for data training. In general, a lower threshold leads to higher installments of false positives. For example, an excellent first-stage classifier can be constructed from two functional strong classifiers with threshold reduction to minimize false negatives. Measured against validation data, the threshold can be adjusted to detect 100% of persons with a false positive rate of 40%. The calculation of the classifier of the two features is about 60 instructions for a microprocessor. It seems hard to imagine that any simpler filter can achieve higher bounce rates. For comparison, to scan a simple template per image or a single layer of perceptron, 20 times more sub-window operations are required.

D. Training of a cascade of classifiers

The cascade training process involves two types of exchanges. In most cases, multi-feature classifiers will achieve higher detection rates and lower false positive rates. At the same time, the classifiers with more features require more time to calculate. Optimization framework can be defined in which the number of classification stages, the number of features in each phase and the threshold of each phase is traded in order to minimize the expected number of rated features. Unfortunately, finding this optimum is an incredibly difficult task.

In practice, a very simple framework is used to produce an effective classifier which is highly efficient. Each phase of the cascade reduces the false positive rate. Target is selected to minimize false positives and to maximize detection. Each stage is trained by adding characteristics until the targets are met, the stages are tested with testing the detector on the validation set.

Fig. 4. Example of frontal upright face images used for training [1].

IV. Practical implementation on the robot

The goal of the implementation is to give the robot the ability to follow users it sees. It does this by detecting their faces [2]. This can be implemented on any kind of robot that uses wheels or tank tracks to allow it to move. For this implementation the Ricardo robot is used. A 3D printable robot that uses tank tracks powered by DC motors. The robot has four DC motor that drive the two tank tracks.

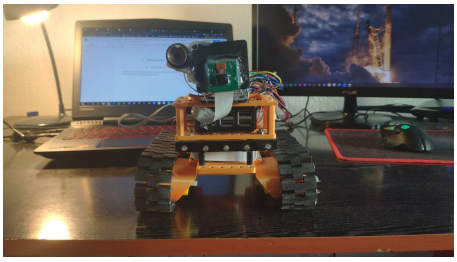

Fig. 5. Frontal view of Ricardo.

The movement of the robot to follow the user is being calculated based on the size (width) of the user's face bounding box on the screen and its position accordingly [2]. The robot's movement can be divided into two categories: translational and rotational. The translational movement is being calculated based on the user's face size, meaning whether the user is close or far away from the robot, the robot should move forward or backward. There are two thresholds for calculating at which distance the robot needs to be from the user. The value we are working with is the width (size) of the users face. The higher threshold is for detecting when the user is too close. It means that the size of the user's face on the screen has passed the upper threshold. In that case the robot has to move backwards to get to a far enough distance. The lower threshold means that the size of the face has become too small on the screen. This is when the user gets too far from the robot. In this case the robot has to move towards the user to get close enough. The robots stops moving when the face size reaches a value in between the two thresholds [3].

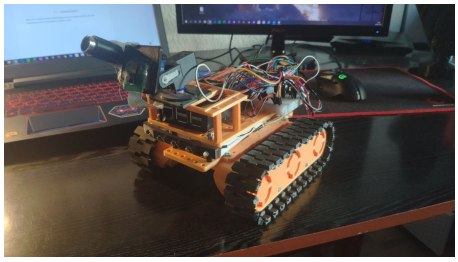

Fig. 6. Angle view of Ricardo.

These thresholds have to be set accordingly to the resolution of the image. For the current implementation which runs on a resolution of 400x300, the upper threshold works best at a value off 150 (pixels wide) and the lower threshold should be a value of 120 (pixels wide) for a robot placed on a table in front of you. These thresholds can be increased or decreased depending on what distance we like to keep the robot at and also keeping in mind the resolution of the image. The rotational movement is calculated based on the position of the face on the image. There are two borders that trigger the rotational movement. If the face bounding box enters the left border then the robot starts to rotate towards the left. If the face enters the right border then the robot proceeds to turn right. If the robot is in neither of the borders then just the translational movement is being calculated. If the face has entered both borders then the translational movement is being calculated only.

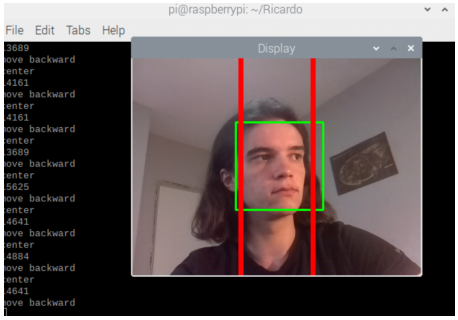

Fig. 7. Dimitar Bezhanovski.

If the face has entered both borders then the user is too

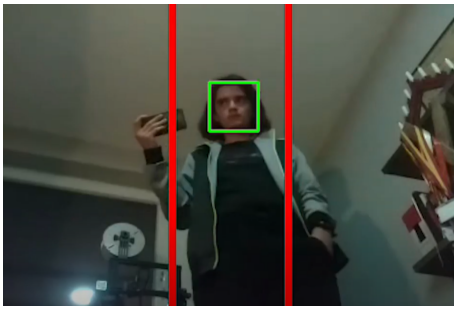

Fig. 8. Dimitar Bezhanovski.

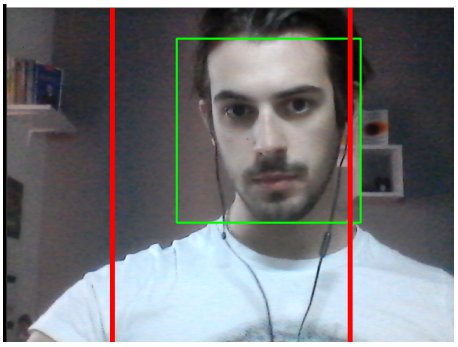

Fig. 9. Aleksandar Stankovski.

close and the robot has to back off. The rotation speed increases linearly as the user's face enters deeper into the border that triggers rotation.

V. Conclusion

The optimal way to implement face detection and tracking on a robot with low powered embedded computer is:

- Detect faces using the Viola-Jones algorithm
- Implement translational movement based on the person's face size in the image
- Implement rotational movement based on the person's face position in the image

This implementation is for devices that do not have a dedicated graphics processor. An argument could be made that the Nvidia Jetson Nano which has a dedicated graphics processor falls into the low powered embedded computer category and that we can run a neural network on it. But devices such as the Jetson Nano are not as available and abundant compared to devices like the Raspberry Pi. Especially due to the ongoing chip shortage, devices with graphics processors are hard to find to and are very expensive. With this Viola Jones implementation we can effectively implement a basic robot following system od readily available and cost effective hardware.

References

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 2001, pp. I-I, doi: 10.1109/CVPR.2001.990517.

[2] M. D. Putro and K. Jo, "Real-time Face Tracking for Human-Robot Interaction," 2018 International Conference on Information and Communication Technology Robotics (ICT-ROBOT), 2018, pp. 1-4, doi: 10.1109/ICT-ROBOT.2018.8549902.

[3] K. Goyal, K. Agarwal and R. Kumar, "Face detection and tracking: Using OpenCV," 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), 2017, pp. 474-478, doi: 10.1109/ICECA.2017.8203730. Technology Robotics (ICT-ROBOT), 2018, pp. 1-4, doi: 10.1109/ICT-ROBOT.2018.8549902.

[4] Hanjing Ye, Jieting Zhao, Yaling Pan, Weinan Chen, Hong Zhang, "Following Closely: A Robust Monocular Person Following System for Mobile Robot" Under review in 2022 IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS 2022), arXiv:2204.10540

[5] Cruz, Claudia Sucar, Luis Morales, Eduardo. (2008). Real-Time face recognition for human-robot interaction. Proceedings of the 8th IEEE International Conference on Automatic Face and Gesture Recognition. 1 - 6. 10.1109/AFGR.2008.4813386.