Real-Time Skin Detection and Tracking based on FPGA

Saranya.S, Keerthikumar.D.N

1Student, M.E, Applied electronics, Kingston Engineering College, Vellore
2Assistant Professor, Kingston Engineering College, Vellore

Abstract - This paper presents the Skin cognizance system is an application for identifying presence of someone from image or videos. Skin detection system offers many applications, especially in video surveillance, biometrics or video coding. The face detection algorithm involved color-based skin filtration. In order to track the image, the location is derived by calculating the centroid of the detected area. Firstly, the project thought is implemented in MATLAB, and then it is converted into Verilog code. The system has been implemented on an DE2_115 FPGA kit and it is demonstrated that 640x480 pixel image are executed in 90ms and use 976332 memory bit totally. Skin detection and tracking is the process of determining whether a skin is present or not. In addition, it will also show the centroid of the image.

Keywords — DE2_115, Real time image, skin detection, image processing, Centroid computation.

I. Introduction:

Face detection in image stream has been an active research area and has been used for various kinds of products in past decade years. Face detection and tracking has been the topics of an extensive research. This system will achieve the robust and accurate solution by adapting a lot of heuristic and pattern recognition. Among feature-based face detection methods, the ones using skin color as a detection cue have gained strong popularity because it has a characteristic color, which is easily recognized by humans. So trying to employ skin color modeling for face detection was an idea suggested both by task properties and common sense.

When building a system, that uses skin color as a feature for face detection, the researcher usually faces three main problems. First, what color space to choose, second, how exactly the skin color distribution should be modeled, and finally, what will be the way of processing of color segmentation results for face detection. This paper covers the first two questions, leaving the third (an equally important one) for another discussion.

In this paper we discuss pixel-based skin detection methods that classify each pixel as skin or non-skin individually, independently from its neighbors. In contrast, region-based methods [1], [2], [3] try to take the spatial arrangement of skin pixels into account during the detection stage to enhance the performance. Skin color detection is also used as a preliminary step in a broad range of computer vision tasks, including gesture analysis, tracking, or content-based image retrieval systems. We evaluate a fast and straightforward adaptive skin detection method for videos. We adapt our decision rules upon a first step of face detection using the well-known approach from [5]. We propose a method which takes high advantage of the temporal relationship between frames in an image sequence and deals well with time dependent illumination changes.

The main drawback of using static color decisions is the high number of false positive detections [4]. The multiple model approach can reduce this number dramatically which is shown in an extensive evaluation.

A real time human body tracking system based on skin detection using Altera's DE2 board, a VGA monitor and a camera. Video streams were obtained from a camera, filtered, averaged and stored in a down sampled memory block. The down sampled frames were used to compute the location of the head and arms of the user. The user can change the current VGA view through a set of switches. The resulting system is able to mimic the user's real time body movements.

II. RELATED WORK

The main goal of skin color detection or classification for skin contents filtering is to build a
decision rule that will discriminate between skin and non-skin pixels. There are different ways to classify skin by color in frames of videos. They can be grouped into three types of skin modelling, parametric, and nonparametric. The parametric models use a Gaussian color distribution since they assume that skin can be modeled by a Gaussian probability density function [7].

FLOW CHART:

Software:

```
Steps involved in designing Software architecture:
1. Input is given as raw image.
2. To detect the skin present in each image thresholding is done.
3. Skin is filtered out so that only R and G pixel can build the skin pixels.
4. With the detected skin in the image the centroid is calculated.
5. Output is shown as the VGA.
```

Hardware:

```
Steps involved in designing Hardware architecture:
1. Video input frames per second.
2. Thresholding skin.
3. Spatial filtering.
4. Temporal filtration.
5. Output VGA
```

III. SKIN FILTER:
A human skin pixel is only dictated by its R and G values. In the Face Tracking + Perspective Projection project, logarithmic values of R and G are used compute if a pixel can be skin. The drawback in this case is that the ranges of logarithmic R and G values need to be tuned for each skin color. On the other hand, the Real-Time Face Detection and Tracking project uses YUV color or the difference between R, G and B values
to compute skin pixels. The advantages of this method are that all of the skin color detection operations are combinational and the intensity of the pixel is removed (component Y). Since the intensity is what dictates the shade of the skin, removing the intensity component allows all skin colors to be detected given a single detection algorithm. Specifically, RGB values can be transformed into YUV components via the following equations:

\[
\begin{align*}
Y &= \frac{R+2G+B}{4} \\
U &= R - G \\
V &= B - G
\end{align*}
\]

To detect skin in YUV color space, the following relationship must be satisfied for 10bit RGB color:

\[10 < U < 74\]

The paper that generated the range values above only performed analysis on humans in pictures for 8bit color. Since the color used in the VGA controller for DE2 board is 10bit RGB, the above values are modified to allow more precision.

\[100 < U < 500\]

The paper also suggested placing a simple color filter to rule out non-skin pixels by ensuring \(R > G\) and \(R > B\). Since the U value must be positive to fall within the valid skin color range, only the \(R > B\) check is performed in the filter.

1. **YCrCb:**

YCrCb is an encoded nonlinear RGB signal, commonly used by European television studios and for image compression work. Color is represented by luma (which is luminance, computed from nonlinear RGB, constructed as a weighted sum of the RGB values, and two color difference values Cr and Cb that are formed by subtracting luma from RGB red and blue components).

\[
\begin{align*}
Y &= 0.299R + 0.587G + 0.114B \\
Cr &= R - Y \\
Cb &= B - Y
\end{align*}
\]

The transformation simplicity and explicit separation of luminance and chrominance components makes this color-space attractive for skin color modelling.

2. **TEMPORAL FILTERING:**

Temporal filtering allowed flickering to be reduced significantly.

\[\text{avg\_out} = (3/4) \text{avg\_in} + (1/4) \text{data}\]

Data: filtered result obtained from the previous stage of a pixel, namely \(p\), in current frame \(\text{avg\_in}\): average value of \(p\) from previous frame \(\text{avg\_out}\): average value of \(p\) in current frame. This is approximately equal to averaging four consecutive frames over time. To ease the computational effort, the equation above can be re-written as:

\[
\text{avg\_out} = (\text{avg\_in}(1/4) + (1/4) \text{data})
\]

\[
\text{avg\_out} = \text{avg\_in} - \text{avg\_in} >> 2 + \text{data} >> 2
\]

3. **THRESHOLDING:**

Since 10-bit color was used in Verilog, adjusting the aforementioned U range yields \[3\]

\[40 < U < 296\]

In this step, each input video frame was converted to a “binary image” showing the segmented raw result.

4. **SPATIAL FILTERING:**

This step was similar to the erosion operation used in the software algorithm. However, the structuring element used here did not have any particular shape. Instead, for every pixel \(p\), its neighboring pixels in a 9x9 neighborhood were checked. If more than 75% of its neighbors were skin pixels, \(p\) was also a skin pixel. Otherwise \(p\) was a non-skin pixel. This allowed most background noise to be removed because usually noise scattered randomly through space, as shown in Figure 1. In Figure 2, because \(p\) only had 4 neighboring pixels categorized as skin, \(p\) was concluded to be a non-skin pixel and, thus, converted to a background pixel.

Fig. 1 Spatial filtering for a pixel \(p\) before filtering.

Fig. 2 Spatial filtering for a pixel \(p\) after filtering.
To examine the neighbours around a pixel, their values needed to be stored. Therefore, ten shift registers were created to buffer the values of ten consecutive rows in each frame. As seen in Figure 3, each register was 640-bit long to hold the binary values of 640 pixels in a row. Each bit in data_reg1 was updated according to the X coordinate. For instance, when the X coordinate was 2, data_reg1[2] was updated according to the result of thresholding from the previous stage. Thus, data_reg1 was updated every clock cycle. After all the bits of data_reg1 were updated, its entire value was shifted to data_reg2. Thus, other registers (from data_reg2 to data_reg10) were only updated when the X coordinate was 0. Values of data_reg2 to data_reg10 were used to examine a pixel’s neighbourhood.

Centroid was computed to locate the skin region. Because connected component labelling was not implemented as initially planned, it was infeasible to calculate the centroid for each face region separately. This limited the number of faces to be detected to two as maximum. First assume that only one face was present. Therefore, its centroid would just be the centroid of all detected pixels, as shown in Figure 4. Note that this calculation would only be correct if one face was present. Although the pixels of one face region might not be connected (and labelled) as originally planned, simply calculating the centroid of all detected pixels still gave a good estimate for the face location. Since area-based filtering was also not applied (due to the lack of connected component labelling), other skin regions—mostly the hands were not entirely removed. However, even if the hands were present, calculating the centroid of all detected pixels still allowed us to locate the face region. This was a reasonable estimate because, compared to the face area, the area of the hand/hands was much smaller.

However, when there were two faces present, calculating the centroid of all detected pixels would only track the location between two faces, rather than track each face separately. To separately track each face in a two-person frame, additional steps were required. First the neighbouring pixels around the centroid were checked to see if they were skin pixels. If they were, it meant the centroid accurately located the face region. However, if the neighbouring pixels of the centroid were not skin pixels, it meant the centroid was somewhere in the background located between two detected face regions, as described in Figure 5.
Obtaining the centroid of each face region allowed us to locate the face of each person present in a two-person video frame. To show how a face was tracked, a small box was drawn around the centroid. The box moved according to the movement of the face. However, if the face moved too fast, the movement of the box might become less stable. Applying To solve this problem, the video frame was split into two according to where the centroid was, as represented in Figure 6. Figure 7 shows the separate calculation for the centroid of each detected region. This technique was done based on the assumption that people typically sit side by side. Temporal filtering here allowed the box to move smoothly. The implementation of the temporal filter here was slightly different from the one shown previously.

\[ Y_n = (1 - \alpha)X_n + \alpha Y_{n-1} \]

- \( X_n \): current input
- \( Y_n \): current output
- \( Y_{n-1} \): previous output

The input here was the location of the centroid before filtering. What this equation meant was, with \( \alpha \) being close to 1, current output would be more dependent on previous output than on current input. This prevented the centroid box from moving too fast when there was an abrupt change in the movement of a person’s face.

V. RESULTS AND OUTPUT

CODE COVERAGE:

![Fig. 8 code coverage of design](image-url)
Simulation Result:

Fig. 9 Simulated output of skin detection.

MATLAB output:

Fig. 13 result on MATLAB

DE2 Board connections:

Fig. 14 DE2_115 FPGA Board connection
Fig. 15 Raw input to VGA

Fig. 16 Skin is detected and the red coloured point is marked at centre of each skin detected portion

Fig. 17 summary of utilized FPGA resources
Table I:

<table>
<thead>
<tr>
<th>Execution Time and Hardware Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time</td>
</tr>
<tr>
<td>Logic element</td>
</tr>
<tr>
<td>Memory block</td>
</tr>
</tbody>
</table>

VII. CONCLUSION:

In this project, the goal of implementing a hardware system to detect and track human faces in real time was achieved. A software implementation of the algorithm was examined in MATLAB to verify its accuracy [Fig. 13]. Also the transition from software to hardware required some modification to the original algorithm, the initial goal was still accomplished [Fig. 14, 15, 16]. The face detection algorithm was derived from a skin detection method. Face tracking was achieved by computing the centroid which is used to determine the location of each detected region. Different types of filter were applied to avoid flickering and stabilize the output displayed on the VGA screen. The system was proved to work in real time without any lagging and under varying conditions of facial expressions, skin tones, and lighting.

VIII. REFERENCES:


