Adaptive Vision System

Rosen Spirov[1] and Neli Grancharova[2]

Abstract – **This paper presents the object detection algorithm implemented in FPGA was based on feature detection and image filtering. Object detection and tracking is the process of determining presention it in an image. A software-based algorithm was independently developed and examined in MATLAB to evaluate its performance and verify its effectiveness.**

Keywords – **Image Processing, Filters, FPGA, Video.**

I. INTRODUCTION

The goal of this project was to create an FPGA system to detect and track an object in real time. The overall setup included the Verilog program, an Altera DE2 board, a camera, and a VGA monitor. The object detection algorithm implemented here was based on feature detection and image filtering. After the object region was detected, its location was determined by calculating the centroid of neighboring feature pixels [1].

A software-based algorithm was independently developed and examined in MATLAB to evaluate its performance and verify its effectiveness. However, it was infeasible to implement the same algorithm in Verilog due to the limitations of the language. Hence, several stages of the algorithm were modified.

Experimental results proved the accuracy and effectiveness of the hardware realtime implementation as the algorithm was able to handle varying types of input video frame. All calculation was performed in real time.

Although the system can be furthered improved to obtain better results, overall the project was a success as it enabled any inputted face to be accurately detected and tracked.

II. DESIGN AND THE SOFTWARE ALGORITHM

Different approaches to detect and track dynamic objects, including feature-based, appearance-based, and color-based have been actively researched and published in literature. The feature-based approach detects a dynamic's objct based on dynamic object features, such as human eyes and nose. Because of its complexity, this method requires lots of computing and memory resources. Although compared to other methods this one gives higher accuracy rate, it is not suitable for power-limited devices.

[1]Rosen Spirov is with the Faculty of Electronics at Technical University of Varna, 1 Studentska Str, Varna 9010, Bulgaria, E-mail: rosexel@abv.bg.

[2]Neli Grancharova is with the Faculty of Telecommunications at Technical University of Varna, 1 Studentska Str, Varna 9010, Bulgaria

Hence, a color-based algorithm is more reasonable for applications that require low computational effort. In general, each method has its own advantages and disadvantages. More complex algorithm typically gives very high accuracy rate but also requires lots of computing resources. General design stages are illustrated in next steps.

- First, the original image was converted to a different color space, namely modified YUV. Then the skin pixels were segmented based on the appropriate U range.
- Morphological filtering was applied to reduce false positives. Then each connected region of detected pixels in the image was labeled.
- The area of each labeled region was computed and an area-based filtering was applied.
- Only regions with large area were considered face regions.
- The centroid of each face region was also computed to show its location.

Converting the object pixel information to the modified YUV color space. The conversion equations are shown as follows:

$$Y= (R+2G+B)/4;\ U=R-G;\ V=B-G \quad (1)$$

These equations allowed *thresholding* to work independently of object color intensity.

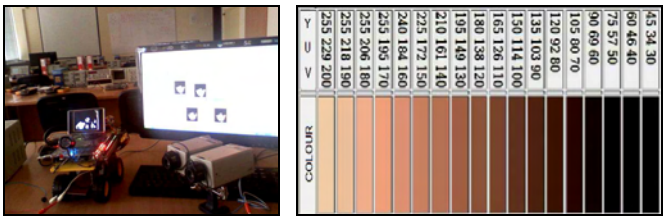

Figure 1 Experimental kit and different tone samples

After object pixels were converted to the modified YUV space, the pixels can be segmented based on the following experimented threshold $10 < U < 74$ and $-40 < V < 11$.

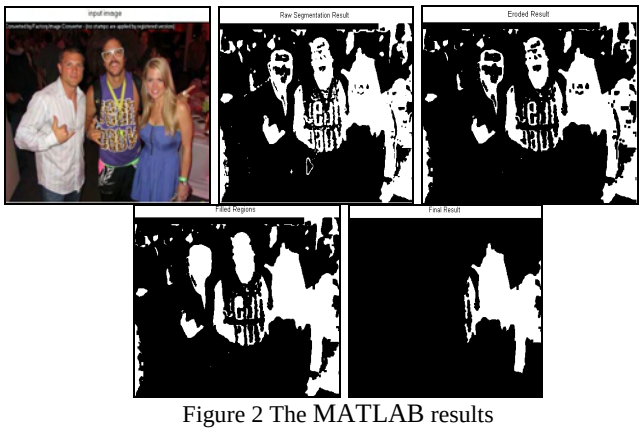

Figure 2 The MATLAB results

As seen in figure2, the blue channel had the least contribution to human skin color. Additionally, leaving out the blue channel would have little impact on thresholding and

skin filtering. This also implies the insignificance of the V component in the YUV format. Therefore, the object detection algorithm using here was based on the U component only. Applying the suggested threshold for the U component would produce a binary image with raw segmentation result.

Applying morphological filtering including erosion and hole filling would, firstly, reduce the background noise and, secondly, fill in missing pixels of the detected face regions [2]. The MATLAB provided built-in functions—*imerode* and *imfill* for these two operations as:

- outp = imerode(inp, strel(*'square'*, 5));

The command *imerode* erodes the input image *inp* using a square of size 5 as a structuring element and returns the eroded image *outp*. This operation removed any group of pixels that had size smaller than the structuring element's.

- outp = imfill(inp, *'holes'*);

The command *imfill* fills holes in the binary input image *inp* and produces the output image *outp*. Applying this operation allowed the missing pixels of the detected face regions to be filled in. Thus, it made each face region appear as one connected region.

After each group of detected pixels became one connected region, connected component labeling algorithm was applied. This process labeled each connected region with a number, allowing us to distinguish between different detected regions. The built-in function *bwlabel* for this operation was available in MATLAB. In general, there are two main methods to label connected regions in a binary image, known as recursive and sequential algorithms.The command *regionprops* can be used to extract different properties, including area and centroid, of each labeled region in the label matrix obtained from *bwlabel*.

Filtering detected regions based on their areas would successfully remove all background noise and any skin region that was not likely to be a object. To be considered a object region, a connected group of skin pixels need to have an area of at least 26% of the largest area. This number was obtained from experiments on training images. Therefore, many regions of false positives could be removed in this stage, as depicted in:

- *object_idx = find(object_area > (.26)*max(object_area));*
- *object_shown = ismember(L, object_idx);*

These two commands performed the following tasks:

- look for the connected regions whose areas were of 26% of the largest area and store their corresponding indices in *face_idx;*
- output the image *face_shown* that contained the connected regions found.

The final stage was to determine object location.

The centroid of each connected labeled object region can be calculated by averaging the sum of X coordinates and Y coordinates separately. The centroid of each object region is denoted by the blue asterisk. Here the centroid of each connected region was extracted using regionprops

III. Design and Implementation

Each current video frame was captured by the camera and sent to the FPGA's decoder chip via a composite video cable [3]. After the video signal was processed in different modules in Verilog, the final output passed through the VGA driver to be displayed on the VGA monitor.The hardware algorithm is shown in Figure 3.

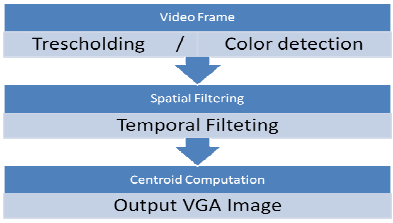

Figure 3 - Hardware Algorithm

❖ Thresholding is in this step, each input video frame was converted to a "binary image" showing the segmented raw result. Since 10-bit color was used in Verilog, adjusting the aforementioned U range yields 40 < U < 296.

❖ Spatial Filtering is in 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.

To examine the neighbors 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 4, each register was 640-bit long to hold the binary values of 640 pixels in a row.

Figure 4 - Ten shift registers for ten consecutive rows

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 neighborhood.

There was a trade-off between the number of shift registers being used (i.e. the size of the neighborhood) and the performance of the spatial filter. A larger neighborhood required more registers to be used but, at the same time, allowed more noise to be removed.

❖ Applying temporal filtering allowed flickering to be reduced significantly. The idea of designing such a filter was borrowed from the project "Real-Time Cartoonifier" [4]. Even small changes in lighting could cause flickering and made the result displayed on the VGA screen less stable. The temporal filter was based on the following Verilog fragment:

```
// *** TEMPORAL FILTERING *********************************//
IF ((VGA_X1 < AVGX_LPF + 10'D5) && (VGA_X1 > AVGX_LPF - 10'D5) &&
```

```
(VGA_Y1 < AVGY_LPF - 10'D5) && (VGA_Y1 > 10'D5))
BEGIN       FLTR_REG <= FLTR_REG + FLTR3[0];
END
ELSE IF ((VGA_X1 == 10'D600) && (VGA_Y1 == 10'D400))
BEGIN       FLTR_REG <= 16'D0;
END         IF (FLTR_REG > 16'D50) BEGIN
IF (AVG2 > 10'B1110111111)
BEGIN // CAN ALSO TRY B1110110111
FLTR3 <= 10'H3FF;
FLTR3_R <= 10'H3FF;
FLTR3_G <= 10'H3FF;
FLTR3_B <= 10'H3FF
// DRAW CENTROID
IF (CNTR > 19'D500) BEGIN // THRESHOLD  WHEN #PIXELS IS TOO SMALL, NOTHING WILL BE
DETECTED
IF ( ((VGA_X1 < AVGX_R2 + 10'D10) && (VGA_X1 > AVGX_R2 - 10'D10) &&
   (VGA_Y1 < AVGY_R2 + 10'D10) && (VGA_Y1 > AVGY_R2 - 10'D10)) ||
   ((VGA_X1 < AVGX_L2 + 10'D10) && (VGA_X1 > AVGX_L2 - 10'D10) &&
 (VGA_Y1 < AVGY_L2 + 10'D10) && (VGA_Y1 > AVGY_L2 - 10'D10)) ) BEGIN
FLTR3_R <= 10'H0;
FLTR3_G <= 10'H0;
FLTR3_B <= 10'H3FF;
……..
END
ELSE BEGIN
FLTR3 <= 10'H0;
FLTR3_R <= 10'H0;
FLTR3_G <= 10'H0;
FLTR3_B <= 10'H0;
IF (CNTR > 19'D500) BEGIN
IF ( ((VGA_X1 < AVGX_R2 + 10'D10) && (VGA_X1 > AVGX_R2 - 10'D10) &&
   (VGA_Y1 < AVGY_R2 + 10'D10) && (VGA_Y1 > AVGY_R2 - 10'D10)) ||
   ((VGA_X1 < AVGX_L2 + 10'D10) && (VGA_X1 > AVGX_L2 - 10'D10) &&
   (VGA_Y1 < AVGY_L2 + 10'D10) && (VGA_Y1 > AVGY_L2 - 10'D10)) ) BEGIN
FLTR3_R <= 10'H0;
FLTR3_G <= 10'H0;
FLTR3_B <= 10'H3FF;
……..
END
```

The filtered result of a pixel in this stage was determined based on its average value *avg_out*. If its average value was greater than 0.06 , becouse the number obtained from experiments, the pixel was considered skin. Otherwise, the pixel was non-skin. Experments of temporal filtering for a two pixels is shown in Figure 5 and Figure 6, / blue is 1,orange is 0/.

кадър	i	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15	+16	+17
преди филтрация																		
изчислена стойн.		.26	.42	.56	.4	.56	.66	.74	.84	.6	.42	.57	.68	.76	.55	.66	.52	.41
след филтрация																		

Figure 5 Example of temporal filtering for a pixel p1

кадър		i+1	I+2	i+3	i+4	i+5	i+6	i+7	i+8	i+9	i+10	i+11	i+12	i+13	i+14	i+15	i+16	i+17
преди филтрац.	1	0	1	0	1	1	0	1	0	1	0	1	0	1	1	1		0
изчислена стойност	0	0.01	0.27	0.21	0.4	0.55	0.38	0.59	0.39	0.6	0.44	0.61	0.46	0.59	0.71	0.79	0.61	0.4
след филтрац.	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 5 Example of temporal filtering for a pixel p2

❖ Centroid Computation, was computed to locate the face region. Because connected component labeling 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 figure7. 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 labeled as originally planned, simply calculating the centroid of all detected pixels still gave a good estimate for the face location. 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.

Figure 7 The FPGA results, when there was a moved man

First the neighboring 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 neighboring pixels of the centroid were not skin pixels, it meant the centroid was somewhere in the background located between two detected face regions. The Verilog fragment is:

```
//*** COMPUTING CENTROID FOR ALL DETECTED PIXELS **** //
IF ((VGA_X1 > 10'D20) && (VGA_X1 < 10'D620) &&
 (VGA_Y1 > 10'D20) && (VGA_Y1 < 10'D460)) BEGIN
IF (FLTR3 == 10'H3FF) BEGIN
SUMX <= SUMX + VGA_X1;
SUMY <= SUMY + VGA_Y1;
    CNTR <= CNTR + 19'B1;
END
END
IF ((VGA_X1 == 10'D2) && (VGA_Y1 == 10'D478)) BEGIN
AVGX <= SUMX / CNTR;
AVGY <= SUMY / CNTR;
AVGX_LPF <= AVGX_LPF - (AVGX_LPF >> 'D2) + (AVGX >> 'D2);
AVGY_LPF <= AVGY_LPF - (AVGY_LPF >> 'D2) + (AVGY >> 'D2);
SUMX <= 30'B0;
SUMY <= 30'B0;
CNTR <= 19'B0;
END
//***COMPUTING CENTROID FOR LEFT HALVED FRAME ******//
IF ((VGA_X1 > 10'D20) && (VGA_X1 < AVGX_LPF - 10'D10) &&
 (VGA_Y1 > 10'D20) && (VGA_Y1 < 10'D460)) BEGIN
IF (FLTR3 == 10'H3FF) BEGIN
SUMX_L <= SUMX_L + VGA_X1;
SUMY_L <= SUMY_L + VGA_Y1;
CNTR_L <= CNTR_L + 19'B1;
END
END
IF ((VGA_X1 == 10'D20) && (VGA_Y1 == 10'D478)) BEGIN
AVGX_L <= SUMX_L / CNTR_L;
AVGY_L <= SUMY_L / CNTR_L;
AVGX_L2 <= AVGX_L2 - (AVGX_L2 >> 'D2) + (AVGX_L >> 'D2);
AVGY_L2 <= AVGY_L2 - (AVGY_L2 >> 'D2) + (AVGY_L >> 'D2);
SUMX_L <= 30'B0;
SUMY_L <= 30'B0;
CNTR_L <= 19'B0;
END
//*** COMPUTING CENTROID FOR RIGHT HALVED FRAME ****//
IF ((VGA_X1 > AVGX_LPF + 10'D10) && (VGA_X1 < 10'D620) &&
(VGA_Y1 > 10'D20) && (VGA_Y1 < 10'D460)) BEGIN
IF (FLTR3 == 10'H3FF) BEGIN
SUMX_R <= SUMX_R + VGA_X1;
SUMY_R <= SUMY_R + VGA_Y1;
CNTR_R <= CNTR_R + 19'B1;
END
END
IF ((VGA_X1 == 10'D621) && (VGA_Y1 == 10'D478)) BEGIN
AVGX_R <= SUMX_R / CNTR_R;
AVGY_R <= SUMY_R / CNTR_R;
AVGX_R2 <= AVGX_R2 - (AVGX_R2 >> 'D2) + (AVGX_R >> 'D2);
AVGY_R2 <= AVGY_R2 - (AVGY_R2 >> 'D2) + (AVGY_R >> 'D2);
SUMX_R <= 30'B0;
SUMY_R <= 30'B0;
CNTR_R <= 19'B0;
END
```

Since area-based filtering was also not applied, 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 objects present, calculating the centroid of all detected pixels would only track the location between two objects, rather than track each object separately. To separately track each object in a two-object frame, additional steps were required. First the neighboring pixels around the centroid were checked to see if they were colour object pixels. If they were, it meant the centroid accurately located the object region. However, if the neighboring pixels of the centroid were not object pixels, it meant the centroid was somewhere in the background located between two detected object regions. To solve this problem, the video frame was split into two according to where the centroid.

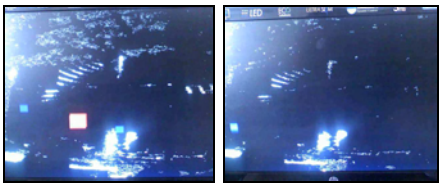

Figure 8 The FPGA results, when there was light effects

To show how an object was tracked, a small box was drawn around the centroid. The box moved according to the movement of the object. However, if the object moved too fast, the movement of the box might become less stable. Applying temporal filtering here allowed the box to move smoothly. The implementation of the temporal filter here was slightly different from the one shown previously.

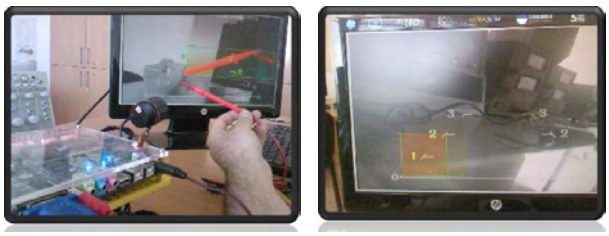

Figure.9 The object tracking

The input Xn here was the location of the centroid before filtering. What this equation meant was, with α being close to 1, current output Y_n would be more dependent on previous output Y_{n-1} than on current input. This prevented the centroid box from moving too fast when there was an abrupt change in the movement of an object, as: $Y_n = (1-\alpha)X_n + \alpha Y_{n-1}$

A clock of 27 MHz was used for the face detection and tracking algorithm. Since the timing was synchronized with the VGA clock, the VGA display was able to update within the time gap between drawing two consecutive frames [5]. The camera was able to detect and track objects in real time. Error seemed to occur only when there was a transition from one person to two people or vice versa in the video frame. The figure10 shown bloks and the working hardware system.

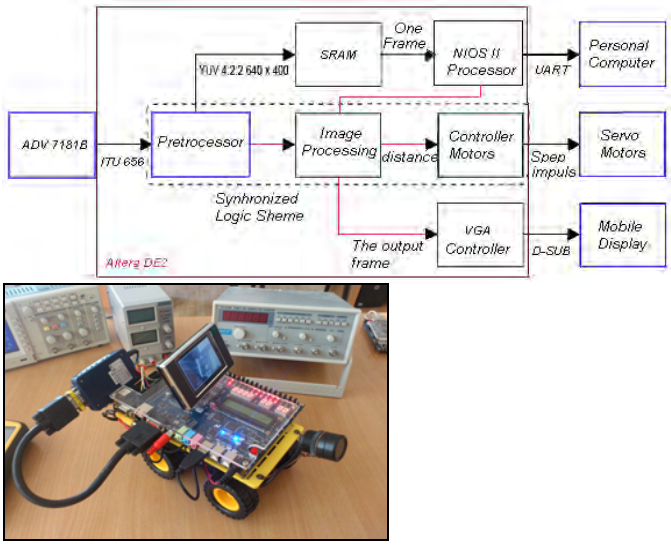

Figure10 The bloks and the hardware system

Within the lab setting, noise was very minimal and did not alter the results. As long as a person was in the camera's view, his face would be accurately detected and tracked. His distance relative to the camera did not affect the result. In the presence of three or more people, the system could only detect the faces but failed at tracking them.

III. Conclusion

The Image Processing Toolbox provided in MATLAB allowed the process of developing and testing the algorithm to be more efficient. Furthermore, verifying the accuracy of the detection algorithm on still pictures provided fair results. Object detection and tracking is the process of determining whether or not present it in an image. Unlike face recognition, which distinguishes different human faces, face detection only indicates whether or not an object is present in an image. Object detection and tracking has been an active research area for a long time because it is the initial important step in many different applications, such as video surveillance, face recognition, image enhancement, video coding, and energy conservation.

References

[1] Furi A., Hang H.M., An efficient block-matching algorithm for motion compensated coding, Proc. JSASSP, pp.1063-1066, 2007.

[2] Хуанга М. Обработка изображении и цифровая фильтрация,Под ред. Т:Мир 2009.

[3] Diaz J., E. Ros, F. Pelayo, "Fpga-based real-time optical-flow system," Circuits and Systems for Video Technology, IEEE Transactions on, vol. 16, Feb. 2006

[4] Advanced Microcontroller Final Projects, or online at: http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects

[5] Jentz B, J. Rotem, Leveraging. FPGA coprocessors to optimize high-performance digital video surveillance systems. www.dsp-fpga.com/articles/jentz_and_rotem