• U C •

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA
Electrical and Computer Engineering Department

Object Tracking and Detection on FPGA Board Cyclone II

Digital Systems Project 2014/2015

Group 11

Bruna Nogueira

Ricardo Mendes

Abstract

Object detection and tracking are widely used in diverse fields, with increasing popularity in surveillance systems, military and healthcare applications, for example. This project addresses the problem of object detection in real time using an Altera DE2 FPGA for image storing and processing. The selected object detection algorithm was Background Subtraction.

Background Subtraction is one of the simplest algorithms for object detection in an image. There are numerous of extensions of higher complexity that produce better results. The primary objective of this project is to build a basic working implementation which can be used as a framework for enhancement in future projects.

In order to accomplish the proposed goals, an incremental approach was followed. The implementation of each feature was always preceded by research on each subject or component.

The implemented system shows satisfactory results in controlled environments - static background and little variation in lighting conditions – with these restrictions being directly related to limitations of the Background Subtraction algorithm.

Table of Contents

Introduction

The aim of this project is to develop a real time object tracking system. For that purpose, the TRDB-D5M Terasic CMOS sensor will be used for image capturing, the DE2 Altera FPGA board will be used for image data storing and processing, and a VGA monitor will be used for displaying the results in real time.

Object detection and tracking has numerous and diverse applications in current days – it is widely use in healthcare context, surveillance systems, in traffic control and also in military applications. Thus, it seems to us of great interest to deepen our knowledge on the subject. One of the key elements for a good object tracking is the object tracking algorithm used. Usually, there is some metric of comparison between a given frame and a temporally previous frame (which could be the immediate previous frame, a unique reference frame, or somewhere in between) that is used for the estimation of movement. There are numerous algorithms for this purpose ranging different levels of complexity, yet we focused on the simplest ones so that a real time operation would be more easily achieved.

The problem of object tracking was addressed by the implementation of Background Subtraction algorithm (BS). This algorithm uses the difference of the current video frame and a reference background frame for the detection of new object(s) in the scene. This algorithm was chosen due to its simplicity of implementation as opposed to others, which can bring some timing advantages to the project, as it proposes to work in real time. Evidently, such simplicity presents considerable disadvantages such as the poor robustness to noise, variations in the background and illumination conditions. Some measures, such as image filtering and thresholding, can be taken to minimize the effects of some of the disadvantages. Due to the lack of robustness of the algorithm, the project will be best suited for controlled environments in terms of illumination and movement.

The BS algorithm will be applied to images captured from the TRDB-D5M camera and stored in the SDRAM chip of the DE2 FPGA board. The resulting data will be further processed for more accurate results and later displayed on the VGA screen.

This document addresses the operation of the overall system as well as the interaction of its main components.

The first section begins by introducing the system's functionalities through the user's perspective, and proceeds with a high level description of the system's architecture and behavior, exploring its operation through the interactions of its main integrating components.

The following section provides implementation details. This includes new created components and adaptations to the camera's configuration registers and modules from the installation demo.

The third main section presents the obtained results and their discussion.

Finally, some observations regarding future work on this project are made, as well as some concluding remarks on the overall process.

Throughout the development of this project a user manual with information about the TRDB-D5M camera, the SDRAM chip on DE2 and the way the two interact in the demo was developed [1].

System Overview

This section begins with an explanation of the operation of the system from the user's perspective, exploring its functionalities and user interaction features, followed by the description of the system's architecture and behavior. This description emphasizes the main intervening modules in the system's operation and explains the data flows among them.

System Functions

For the correct operation of the project, one needs:

- DE2 FPGA board
- TRDB-D5M CMOS sensor
- VGA monitor
- Connection cables

The FPGA's switches and keys are the means of interaction user-system. The following table lists the switches/keys associated with the different system functions.

Function	Key
Color Mode	SW[17]
Threshold Adjustment	SW[7..0]
Start/Resume Capture	Key[3]
Pause Capture	Key[2]
Reset System	Key[0]

Under normal operation, the images displayed on the VGA monitor show a red contour around the new objects in the scene, by comparison with a previously saved background reference frame. There are two modes of image display: original color object or binarized subtraction image display.

In the binarized subtraction image display mode, the displayed data is the result of a binarization of the image that results from the pixel-by-pixel intensity subtraction of the background image to current frames. The binarization is a function of a threshold value which can be adjusted by the user so as to adapt to different environment conditions (noise, lighting, colors, etc.). This way, with a good threshold, the object will appear in white and the remaining scene will be black.

In the original color object mode, the object appears in its original RGB color and the remaining scene will be black.

It is possible to restart the whole process **by resetting the system and starting a new capture**.

Extra information can be taken from the FPGA's LEDs and 7-segment displays:

- The FPGA's green LEDs blink at the frame rate frequency.
- The 7-segment displays HEX1 and HEX0 show the value of the currently selected threshold in hexadecimal format, with the most significant 4 bits on HEX1 and the least significant 4 bits on HEX0.
- The remaining 7-segment displays, HEX7 – HEX2, show the frame counter since the last system reset.
- The red LED LEDR[17] is set when the background frame is completely saved in the SDRAM.

System Architecture

In this section, the architecture design of the system will be explored, with focus on the main modules composing the system and their communication.

The architecture design can be described, at a high level, by figure 1. The modules in blue are those already implemented in the installation demo of the camera. The orange modules and arrows correspond to parts entirely developed in this project, and the purple ones were adapted from their demo versions. For further details on function and operation of the blue and purple represented modules, the user manual developed along with this project can be consulted.

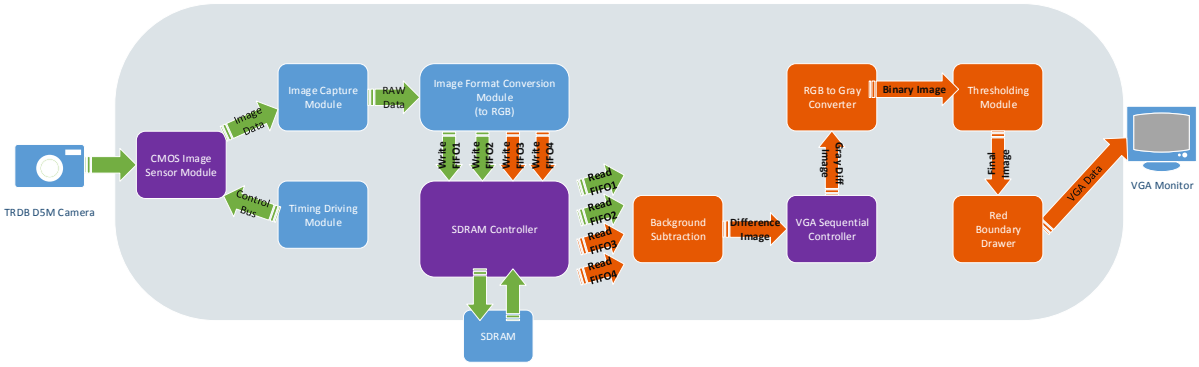

Figure 1.The system's architecture design and the flow of data among its main component modules.

The system's global operation can be divided in four main operations: image capture (which contains some of image data pre-processing), image storage, image post-processing and image display. Herein, the operations involved in each of these stages are briefly described to enable a better understanding of each module's role on the data flow until the data is ready for display.

Image Capture

The image data is first captured by the camera's CMOS sensor and is forwarded to the Image Capture Module, where valid pixel information is extracted according to validity control signals. The resulting RAW data is then converted to RGB format.

Image Storage

The image's RGB data is subsequently written to the SDRAM with the help of four buffering FIFOs and the SDRAM controller. The use of the FIFOs for reading and writing is necessary to overcome clock mismatches between the camera's pixel clock, the SDRAM and the VGA clock.

For storing purposes, each pixel's data is segmented into two halves – one containing all red and half of the green pixel information, and the other containing all blue and the remaining half of the green pixel information. Each half is buffered to an individual FIFO, and each FIFO writes to a designated SDRAM bank. A pair of writing FIFOs holds pixel data regarding the current image and the other pair holds image data of the background.

The same goes for the four reading FIFOs. The reading FIFOs store the RGB data concerning the current and background frames read from the SDRAM.

Image Post-Processing

After reading the image's RGB data from the SDRAM, the absolute difference value between the reference background and the current frame for each pixel is calculated in the VGA controller to be then output to a thresholding operation which binarizes the pixel value. As already mentioned, the value of this threshold can be adjusted through the configuration of switches SW[7..0].

After binarization, a final component draws a red contour around white structures in the binarized image through a simple edge detection implementation.

Image Display

The displayed data depends on the selected threshold value for binarization and on the selected color mode. If the selected color mode is the binarized differences image, the binarized image and red contours obtained in the described image post-processing phase will be displayed. Otherwise, the original RGB object data and the same red contours will be displayed.

High Level Finite State Machine

This project deals with some controllers: the SDRAM controller, the VGA controller and I2C controller (the latter is used to assign new values of camera configuration registers, sending these values from the FPGA board to the CMOS sensor). Although a controller module for the whole project was not implemented, the control was made by adapting the code of these controllers. At a very high level of abstraction, the project follows the sequential behavior depicted in figure 2.

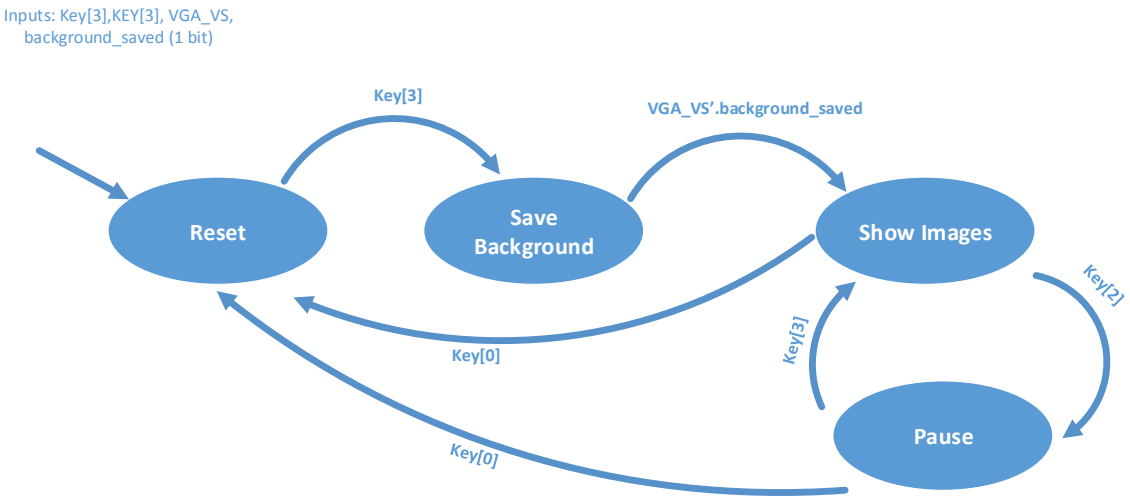

Figure 2.The system's high level FSM

The outputs are not listed in the Finite State Machine (FSM) of figure 2, as they were too extensive to be included. Below, we list the most relevant actions associated with each state:

Reset

The reset state is entered when the System Reset key is pressed or when the system initializes. On this state, camera and SDRAM controller state information is reset to the initial state, with writing and reading addresses to and from the SDRAM also being reset.

Save Background

This state involves the buffering of the background frame information, pixel by pixel, in the pair of writing FIFOs designated for the background frame. The remaining 6 FIFOs are not operating in this state. The necessary signals for the generation of writing commands and address information are also provided to the SDRAM controller. From this information, the SDRAM controller generates the control signals and computes addresses – bank, bank line, and bank row - for accessing the SDRAM and writing pixel contents.

When all the pixels of the background frame have been stored, and the VGA controller generates a vertical SYNC signal denoting the end of a VGA frame, the system can enter the display state, **Show Images**.

Show Images

This state involves the writing of the current captured frame to the SDRAM and the reading of information regarding the current image and the background reference image for later display. There are 6 FIFOs operating in this state:

One pair of writing FIFOs holds the data of the successively captured frames, for writing in the SDRAM. The four reading FIFOs buffer the data read from the SDRAM – each pair of FIFOs holds the data of one of the two stored images. For reading and writing from the SDRAM, the necessary address information and control signal information must be provided to the SDRAM controller.

The read images go through the already mentioned operations of background subtraction, RGB to grayscale conversion, thresholding and boundary drawing. The image display format depends on the selected color mode.

Pause

The pause state can be used to hold the current image on the screen. In this state one can change the color mode and analyze the static image.

Implementation Details

This section will provide technical details on the system's implementation. We won't cover all the Terasic's demo details; instead, we will focus on what was changed and what was created. For further details on the camera's demo configurations we recommend the consultation of the User Manual produced alongside with this project and the camera's Hardware Specification Manual[2].

Camera Configurations

In order to obtain a real time image processing we had the need to configure the camera to get a clock signal of higher frequency. The table below contains the camera's internal registers values used in order to achieve the pixel clock frequency:

Factors and respective registers	Decimal value
PLL_m_Factor – R0x011[15:8]	16
PLL_n_Divider – R0x011[7:0]	2
PLL_p1_Divider – R0x012[4:0]	1

Which produces pixel clock frequency $f_{pixclk} = 86.4\ MHz$ with an external clock frequency of 27 MHz.

We used the default exposure time:

Sensor_exposure – {R0x08[15:0], R0x09[15:0]}	1984

As for the image resolution we weren't able to successfully reduce it through the alteration of the appropriate register fields according to the camera's hardware specification manual[2]. It should be noted that demo values don't follow the camera's hardware specifications rigorously. Therefore, we are using the demo's default resolution, through the following configurations:

Factors and respective registers	Decimal value
Row_size – R0x03[15:0]	2047
Column_size – R0x04[7:0]	2559
Row_Bin and Row_Skip – R0x022[5:0]	1 and 1
Column_Bin and Column_Skip – R0x023[5:0]	1 and 1

And thus having a resolution of 1280x1024 pxl*pxl.

As we weren't able to reduce the resolution and there was a need of storing at least one frame in memory, we opted to use the SDRAM memory.

SDRAM Controller Configurations

The Cyclone II FPGA from Altera has an 8 MB SDRAM chip with 4 banks composed of rows and columns. The Terasic's demo uses four FIFOs (4-port SDRAM controller) to store and read the captured frames from the SDRAM. As we needed to store and read the background concurrently to this process we had to use more FIFOs. Our first attempt was to use the already existing 4-port SDRAM and add 2 additional FIFOs and the respective needed signals. With this two FIFOs we were supposably able to store the background gray levels which was the intended for the first version. Unfortunately that approach didn't work. We suspect that we have missed the implementation of some conditions for the SDRAM control. We then found out that there's an 8-port SDRAM controller from Terasic that we could use in our project. Thus we currently have 8 working FIFOs – 4 for reading purposes and 4 for writing purposes. The camera uses 4 (2 for read and 2 for write) to use the SDRAM as frame buffer while we concurrently use the other 4 to store and read the background.

The tables below contain the main configurations for all the 8 FIFOs. For simplicity we will call them FIFOs 1 to 4 and have a write and a read version of each. We will then proceed to the explanation of some of the used values.

	FIFOs1		**FIFOs2**	
	Write	**Read**	**Write**	**Read**
Data	Frame: {0,G[11:7], B[11:2]}		Frame: {0,G[6:2], R[11:2]}	
Enabler	sCCD_DVAL & state1	Read & state1	sCCD_DVAL & state1	Read & state1
Starting Address	0		22'h100000	
Max Address	640*480		640*480+22'h100000	
Clock	~CCD_PIXCLK	~VGA_CTRL_CLK	~CCD_PIXCLK	~VGA_CTRL_CLK

	FIFOs3		**FIFOs4**	
	Write	**Read**	**Write**	**Read**
Data	Background: {0,G[11:7], B[11:2]}		Background: {0,G[6:2], R[11:2]}	
Enabler	sCCD_DVAL &(!bkg_saved)	Read & state1	sCCD_DVAL &(!bkg_saved)	Read & state1
Starting Address	22'h200000		22'h300000	
Max Address	22'h200000+640*480		22'h300000 + 640*480	
Clock	~CCD_PIXCLK	~VGA_CTRL_CLK	~CCD_PIXCLK	~VGA_CTRL_CLK

FIFOs1 and FIFOs2 are responsible for storing and retrieving the current frame, using the SDRAM as frame buffer, and FIFOs3 and FIFOs4 are responsible to save and retrieve the background. Each FIFOs store half of the information of a pixel and thus the reading must be concurrent with the corresponding pair (example Read FIFO 1 and Read FIFO 2).

The clock used to read from the SDRAM is the negation (not) of the VGA_CTRL_CLK signal, which is the clock used to control the VGA pixel clock. For writing, the CCD capture pixel clock was used.

The starting addresses correspond to the first address of each bank, to which we added 640*480 - the *Max Address* - to store the amount of pixels needed for the VGA display.

For the enablers (write and read enable) we had to add some control on the 8-port controller. We had some synchronization problems using all the 8 FIFOs concurrently. In fact, we couldn't get an image on the VGA with all the FIFOs and the image processing operations we had. To overcome that problem, we added two flags (1 bit output registers) – *bkg_saved* and *state1*. The flag *bkg_saved* indicates whether the background is completely saved in the SDRAM. While this condition isn't met, the pixel data buffered in FIFOs 3 and 4 must be written to the SDRAM. When this flag is set to HIGH, this two FIFOs stop writing and they are no longer needed until a new reset signal is received.

On the other hand we will want the other 6 FIFOs to start operating when the background is completely saved. So, as a first try we used only this flag (equivalently to have *state1 = bkg_saved* on the FIFO tables) to control all FIFOs. When the background writing ended we started the readings. However, the VGA has a different clock from the one of the CCD capture and when the background was completely saved the VGA was still in a middle of a frame sweep. Thus, the image representation wouldn't start at the upper left corner of the screen but rather at a random position. To solve that issue we added the flag state1 which is define as follows:

$$\text{state1} <= ((\text{bkg\_saved} \ \& \sim(\text{VGA\_VS})) \mid \text{state1})$$

Basically the bit is set when the background is completely saved and there's a vertical sync from the VGA indicating that we are on the first pixel of the sweeping. Also, after that happened once, we want this bit to remain active, and thus the *or* operation with itself.

In summary, this two flags control the operation of the FIFOs enabling and disabling them in order to produce the right results. First the two writing FIFOs write the background and then we synchronize the reads and writes with the VGA using the other six FIFOs.

It is important to refer that the SDRAM was configured to perform operations with the following parameters:

Parameter	Value
INIT_PER	32000
REF_PER	1536
SC_CL	3
SC_RCD	3
SC_RRD	7
SC_PM	1
SC_BL	1

And thus obtaining 133MHz in read/write operations.

VGA Controller

The VGA controller is the component responsible for the VGA control signals and for sending pixel data to the screen. We have reused Terasic's VGA Controller but we added some important changes.

As we wanted the binary image and the original image we thought it would be faster to do the subtraction directly in this module. The figure below illustrates the changed module:

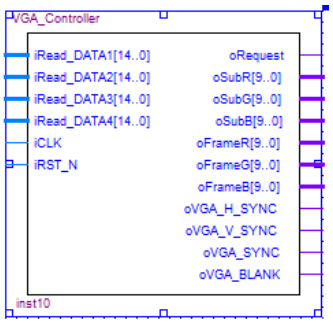

Figure 3. The updated VGA_Controller.

The read data inputs are the pixel data where the DATA1 and DATA2 hold the current frame values and DATA3 and DATA4 hold the background data of the corresponding pixel. The oRequest is the Read signal used to read from the SDRAM FIFOs; the oSubR, oSubG and oSubB are the RGB components resulting from the absolute value difference between the current frame pixel and the corresponding background pixel; the oFrame outputs are the current frame colors in RGB. The other outputs are the signals needed for the VGA control.

To implement the subtraction we had to use unsigned inputs and outputs, and to make sure that we didn't get negative results, so we applied a simple ternary operator to obtain the absolute value of the subtraction.

In summary, from the output of the VGA controller we obtained the current frame and the result of the subtraction of the current frame with the background in RGB.

Ideally, this simple subtraction would provide the needed object identification but this is not the case. This process is highly sensible to image variations in illumination conditions, movement, light reflection, noise, and is color dependent up to some level. So, in order to better standout the objects and to eliminate some of the noise, a threshold was applied to the absolute difference value, which we will explain in the sub-section below.

Image Processing

Having the subtracted frame was not nearly enough to a good object detection. The resulting image needs some processing in order to reduce the noise and to highlight the object. Thus, we started out by converting the image of differences into a **gray scale,** over which a simple **threshold** was applied to reduce false positives and noise. The conversion implemented is a simple average of the three RGB components:

$$Gray\ level = \frac{(R+G+B)}{3},$$

producing satisfactory results.

We have also tried using the luminance equation, but as we are working with integers and unsigned values, and it would be rather complex to implement floating point numbers to the equation. Therefore we made the following approximation:

$$Y = \frac{2*R}{10} + \frac{7*G}{10} + \frac{1*B}{10}.$$

This approximation yield worse overall results than with the RGB component average and so we kept using the average instead.

To apply a **threshold** to the image we simply applied a value comparer to check whether the pixel's gray value is higher than the value of the switches SW[7..0] or not, granting a dynamic user defined threshold.

To **identify** the objects boundaries in the image we have implemented a very simple red boundary module that surrounds the object with a red line in the presence of an edge. The module does the following: Every four upcoming pixels, a binarized pixel is stored and compared to the current pixel information to check for an edge. If the current pixel is different from the stored pixel (xor is 1) then it will draw the remaining pixels (until the next saving) in red.

In the worst case scenario this will draw the red border three pixels after object ends which is an acceptable error. We tried using only 1 pixel to mark the borders but it was hardly noticeable with the current VGA resolution.

Below we present the connections between the modules above mentioned:

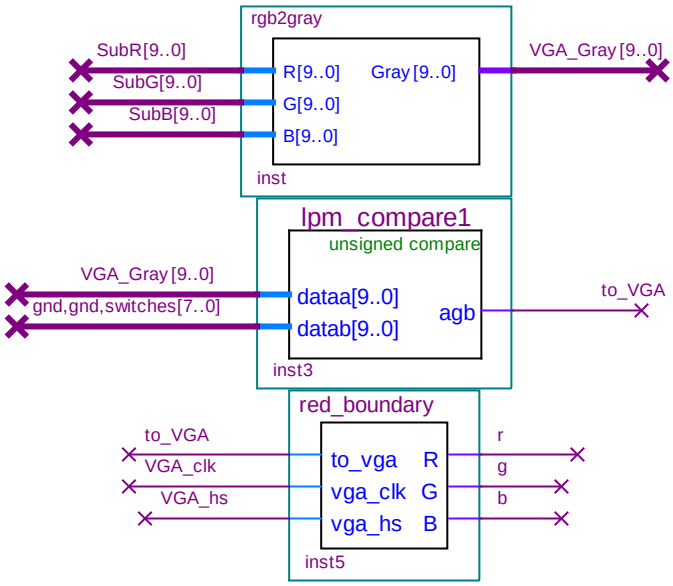

Figure 4. The pixel by pixel image processor modules

The outputs of the *red_boundary* module are the following: r, g and b are the bits indicating if the red, green and blue components are present in a given pixel. If *to_VGA* is 1 (white binarized pixel) and it is not part of the boundary, r, g and b will output 1; If it belongs to a boundary, then r will be equal to 1 and g and b will be 0 (red boundary). If *to_VGA* is 0 then so are r, g and b.

After this small process we have defined each pixel, by the r, g and b outputs, either as black – non object pixel –, white – object pixel – or red – boundary pixel. In order to give the option of showing the object either binarized or in RGB (both with the red identifying border), we've created the *colorChooser* module. This last module is responsible for sending the pixel color data to the VGA by interpretation of the r, g and b values from the *red_boundary* outputs and of the SW[17] (*color_mode*) input. If the *color_mode* switch is off, then the displayed image will show the white object with the red boundary. Otherwise, it will show the object in RGB colors also with the red boundary. In both cases, scene elements other than the object will appear in black.

Results and Result Analysis

As previously stated, this algorithm presents some limitations, leading to undesirable results under certain conditions. Nonetheless, using the gray scale image and the dynamic threshold, we were able to prove that this implementation does work if some conditions are met.

The figures below depict the obtained results using an almost ideal background:

Figure R1. The Background used for the images R2, R3 and R4.

Figure R2. Binaryzed image of a hand and a rectangular object

As you can see from figure R2, the objects – the hand and a box – were successfully detected. There's still some noise in the image but quite reduced. It is important to note the following: The box has the word "mouse" printed in black and the background is a dark green and so the subtracted gray values from that area are eliminated with almost any threshold. The color dependency is one of the limitations of the BS on our implementation, as previously stated.

Figure R3. This image is a representation of the image R2 using the color mode on.

Figure R4. Detection using a less geometrical object.

Figure R3s and R4 illustrate the results obtained using color mode on. Our results are similar to the ones used on some applications in television and movies, where they use a certain background color and then subtract that color from the image.

Using a not so ideal background, but a well illuminated one, the obtained results are quite satisfactory. The following figures show the results:

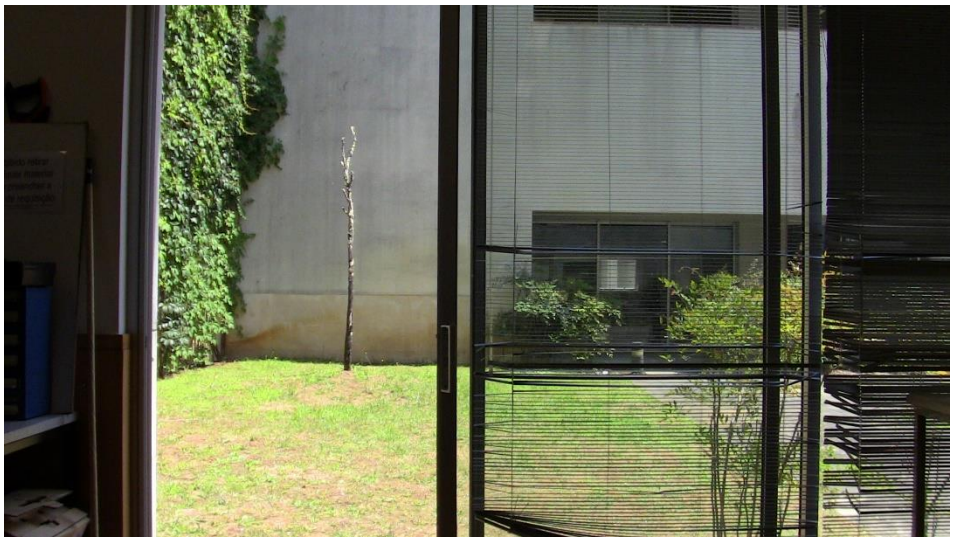

Figure R5. Background used for the images R6. and R7.

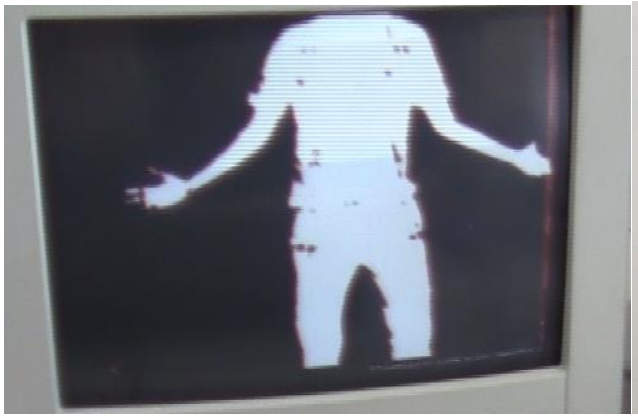

Figure R6. Binarized human "object"

Figure R7. Color mode human object

Analyzing the results, we conclude that the algorithm works well under certain circumstances. Illumination, background colors and clock speed are the strongest limiting factors. The results could have been further enhanced using image processing filters and more appropriate edge detectors.

Future Work

Although good results can be achieved when in a controlled environment, further enhancements could be performed so as to make the system more robust to light and noise variance. However, these improvements can only lead to a certain level of performance, due to the limitations of the BS algorithm. To increase performance over this level, other object tracking algorithms should be considered. Herein, we list some measures that could be taken in order to improve the system's performance.

For noise attenuation purposes, low pass and median filtering could be performed over the background and current images. This attenuation would be important for decreasing false positives – by removing noise and reduce lighting dependency - and for more accurate boundary detection, and at the same time enabling lower thresholds (high thresholds for noise compensation can lead to only partial detection of objects).

For better boundary detection, an adequate edge detection algorithm should be used, with appropriate filters for the purpose – gradient filters or laplacian/LOG operators.

To obtain a better performance, as already suggested, other object tracking algorithms should be considered – like sum of absolute differences (SAD) or sum of squared differences (SSD). These are still relatively simple implementations, but demand far more processing operations and thus more processing time, which could possibly interfere with the real time requirements without a full restructuration of the system.

If possible, the system breakdown in its composing modules and their individual adaptation is suggested as a way of handling only the strictly needed operations of each module and thus taking only the necessary bandwidth. This restructuration would, in most cases, lead to a more efficient system when in need of adaptations.

To make the project more complete and appealing to the user, some additional information could be provided. For example, a bounding box surrounding various detected objects; the option of detecting objects of different scales in the screen, given their size in image coordinates; Showing the centroid of the object. In a more complex approach, the motion field could be computed and the corresponding vectors displayed along with the object.

Conclusions

We have successfully implemented the background subtraction algorithm and, with simple image processing, produced satisfactory results in real time.

The final product of our work now combines the working background subtraction algorithm using FPGA cyclone II from Altera and Terasic's TRDB-D5M CMOS camera and a User Manual that summarizes the needed information to work with the different components used in this project.

This project was built under the objective of reusability for future works. Therefore, we created a simple programming architecture so that extensions can be added to further enhance the object detection results.

References

[1] CMOS TRDB-D5M Camera and 8MB SDRAM A2V4S40CTP User Manual.

[2] Terasic Technologies, TRDB-D5M, Terasic TRDB-D5M Hardware specification, Document Version 0.2, June 10, 2009.

Other Sources of Information

Some other useful sources of information for the development of this project are listed below:

Terasic Technologies, TRDB-D5M 5 Mega Pixel Digital Camera DevelopMent Kit, User Manual

Ruiwen Zhen, Enhanced Raw Image Capture And Deblurring, Graduate Program in Electrical Engineering, Notre Dame, Indiana, April 2013.

Cato Marwell Jonassen, Embedded Demonstrator for Video Presentation and Manipulation, Master of Science in Electronics, Norwegian University of Science and Technology, June 2010.

Alexander Bochem, Rainer Herpers and Kenneth B. Kent, Acceleration of Blob Detection in a Video Stream using Hardware, Faculty of Computer Science University of New Brunswick Fredericton, Canada, May 26, 2010.

Daniel Bengtsson, Richard Fång , Developing a LEON3 template design for the Altera Cyclone-II DE2 board , *Master of Science Thesis in Integrated Electronic System Design,* Chalmers University of Technology University of Gothenburg Department of Computer Science and Engineering Göteborg, Sweden, August 2011.

V. B. Jagdale, R. J. Vaidya, High Definition Surveillance System Using Motion Detection Method based on FPGA DE-II 70 Board, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-2, Issue-2, December-2012.

Muhammad Shahzad , Object Tracking Using FPGA, (An application to a mobile robot), Master Thesis, School of Information Science, Computer and Electrical Engineering, Halmstad University, May 2012

M. Surumbar Khuzhali, *Back Ground Subtraction Algorithm For Moving Object Detection In FPGA,* Department of ETC, Bharath University, India, Middle-East Journal of Scientific Research 20 (2): 198-204, 2014.

Bing Han, William Roberts, Dapeng Wu, Jian Li*, Robust Feature-based Object Tracking*, Department of Electrical and Computer Engineering University of Florida Gainesville

G. Shrikanth, Kaushik Subramanian, Implementation of FPGA-based Object Tracking Algorithm, Electronics and Communication Engineering Sri Venkateswara College of engineering, Sriperumbudur, April 2008.