MCGILL UNIVERSITY

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

ECSE 456 - FINAL REPORT

A Real-Time Object Tracking System

Authors:

Benjamin BROWN
benjamin.brown2@mail.mcgill.ca
260450182

Taylor DOTSIKAS
taylor.dotsikas@mail.mcgill.ca
260457719

Supervisors:

Warren GROSS, PROF.

Arash ARDAKANI

Abstract

Video processing represents an extremely relevant challenge as both the demand for intelligent, aware systems and the quality of modern video technology increases. This increase in quality comes at a cost of large amounts of data being handled under strict time constraints [3]. This project aimed to study how a common video processing algorithm such as object motion tracking can be accelerated using custom hardware. This report concludes on the background, design, and findings of Phase 1 of the project; where platform research, algorithm research, and software implementation was performed. It was clear that an FPGA would be the best solution for fast, low power hardware implementation and a Kalman filter based algorithm was the best solution for a predictive algorithm not prone to noise. Finally, the large amount of time the software needed to process relatively short videos proved that direct hardware implementation of the algorithm is essential for applications. Software implementation was performed in MATLAB, and hardware implementation will be done on an Altera Cyclone II FPGA during Phase 2 of the project.

Acknowledgments

The authors would like to thank Professor Warren Gross, for undertaking the responsibility of supervising this project. We would also like to thank Arash Ardakani for all of his advice, help, and time spent meeting with us throughout the semester. Finally, we would like to thank anyone who contributed to the wealth of online resources about object tracking that we hope to contribute to.

Contents

1 Abbreviations & Notation

FPGA - Field Programmable Gate Array

ASIC - Application Specific Integrated Circuit

CPU - Central Processing Unit

DFG - Delta Frame Generation

2 Introduction

2.1 Motivation

Scene recreation and analysis is imperative in digital systems that must understand and react to events in their environment. Some typical examples of this include surveillance, robotics, and human-computer interaction. A variety of sensors can be employed for such a task including ultrasonic, radar, and passive infrared, but all of these sensors do not come close to modeling an environment as completely as a video camera. With the increase in image quality and device accessibility, the video camera seems like the obvious solution.

However, due to the vast amount of data and system imposed processing time constraints, video processing is a challenge. For instance, the transition to a high-definition video platform produces six times more data than the previous standard-definition one [3]. This project aims to study how a complex video processing algorithm such as real-time object tracking can be greatly accelerated when implemented directly in hardware. Object tracking represents an excellent example of the preceding challenges because it requires capturing an image of a scene, processing the image to locate the object in motion, and reconstructing said scene with emphasis placed on the motion, all in real-time.

2.2 Applications

Taylor

3 Background

3.1 Video Processing

The most fundamental way of understanding video data is to consider it as a collection (static) or stream (real-time) of discrete images called frames. A frame is an MxN matrix of pixels. Each pixel is the smallest, discrete element of the image, and store intensity data about the image. There are many different ways of representing intensity in a pixel.

In the case of a color image, the pixel contains multiple values that describe the color space. Common representations of the color space are RGB or YCbCr [1]. In both of these models, each pixel has 3 values. Since MATLAB's `VideoReader` class uses RGB, for convenience, this color space is used for the remainder of the project. A single color RGB frame is mathematically described as an $MxNx3$ matrix with the form

$$F_i = \begin{bmatrix} (R_{11}, G_{11}, B_{11}) & ... & (R_{1N}, G_{1N}, B_{1N}) \\ ... & ... & ... \\ (R_{M1}, G_{M1}, B_{M1}) & ... & (R_{MN}, G_{MN}, B_{MN}) \end{bmatrix}. \quad (1)$$

Where

$$i = 1, 2, ..., n.$$
$$0 \leq R_{ij}, G_{i,j}, B_{i,j} \leq 256$$

The first frame, F_1, is defined as the *base frame*, and the remaining frames are defined as the *current frame* for processing. If a video is t seconds long, the *frame rate* is defined as

$$f = \frac{t}{n}. \quad (2)$$

In the case of a black and white image, the pixel contains a single value that represents the grayscale intensity. Similar to F_i, the grayscale frame also contains n frames, and its elements, Y_{ij}, are limited between 0 and 256. However, it is an MXN matrix only. It will become apparent later that conversion between RGB and grayscale is imperative for many video processing algorithms. Unsurprisingly, there are multiple ways to do this. The colorimetric conversion principle converts RGB to grayscale using the following weighted sum:

$$Y_{ij} = .2126 \cdot R_{ij} + .7152 \cdot G_{i,j} + .0722 \cdot B_{i,j}. \quad (3)$$

Finally it should be mentioned that video data can be streamed in either progressive or interlaced format. Progressive format is the standard one frame at a time while interlaced divides the frames in half into fields. Each field contains either odd or even rows of it's corresponding frame. While this does requiring processing more frames, it gives a clearer, smoother picture as there are less scene changes between frames [1]. This design choice plays a larger role in Phase 2 of the project, since software implementation uses static (pre-recorded) videos.

3.2 Optical Flow

-Definitions of optical flow from literature, how it can be used to find motion.
-Delta Frame Generation
-Measuring the center of the object by intersecting two lines from left/right top/bottom of that object

3.3 Kalman Filter

A Kalman filter is used to combine continuous predictions from a theoretical system model with continuous measurements from a real implementation of the same system. This is done to help reduce the effects of noise on system measurements, and can provide a prediction of the system's next state if for some reason the measurements fail or contain lots of noise. Implementation of a Kalman filter is highly intuitive for a sensor that has been characterized with testing. But for object tracking, it is much more difficult to understand as all of the Kalman filter jargon is focused around "sensors" and "measurements" that sound odd when referring to an object tracking system.

The Kalman filter equations can be divided into two key sets: the predication equations and the update equations [4], [12].

Prediction Equations:

$$\vec{x_{k+1}} = F \cdot \vec{x_k} + B \cdot \vec{u_k} \tag{4}$$

$$P_{k+1} = F \cdot P_k \cdot F^T + Q \tag{5}$$

Intermediate Calculations:

$$\vec{y_{k+1}} = \vec{z_k} - H \cdot \vec{x_{k+1}} \tag{6}$$

$$S_{k+1} = H \cdot P_{k+1} \cdot H^T + R \tag{7}$$

$$K_{k+1} = P_{k+1} \cdot H^T \cdot S_{k+1}^{-1} \tag{8}$$

Update Equations:

$$\hat{\vec{x_{k+1}}} = \vec{x_{k+1}} + K_{k+1} \cdot \vec{y_{k+1}} \tag{9}$$

$$\hat{P_{k+1}} = (I - K_{k+1} \cdot H) \cdot P_{k+1} \tag{10}$$

There are a few key observations that can be immediately made about this set of equations. Note that the distinction between intermediate calculations and predication equations is one that is not usually made in descriptions of Kalman filtering like the one by Trucco & Verri [4]. However, from an implementation perspective this distinction makes sense, as the system in which the Kalman filter is placed in (or main algorithm) is blind to these equations. Also notice that matrices and vectors without subscripts are ones that do not change each iteration, and remain constant from initialization to completion.

While all elements in this set of equation have physical meanings and distinct characteristics, for the sake of simplicity the only once that need to be discussed in detail are the inputs, outputs, and constant variables.

3.4 Fixed vs. Floating-Point

Ben

4 Requirements

Taylor

5 Design

5.1 Platform

In this section we look to formally justify the aforementioned choice of an FPGA, and specifically an Altera FPGA, for hardware implementation in comparison to other embedded hardware choices.

5.2 Algorithm

Discuss the merits of the algorithms we researched and why we chose DFG with a Kalman filter.

5.3 Software

Discuss how the software works.

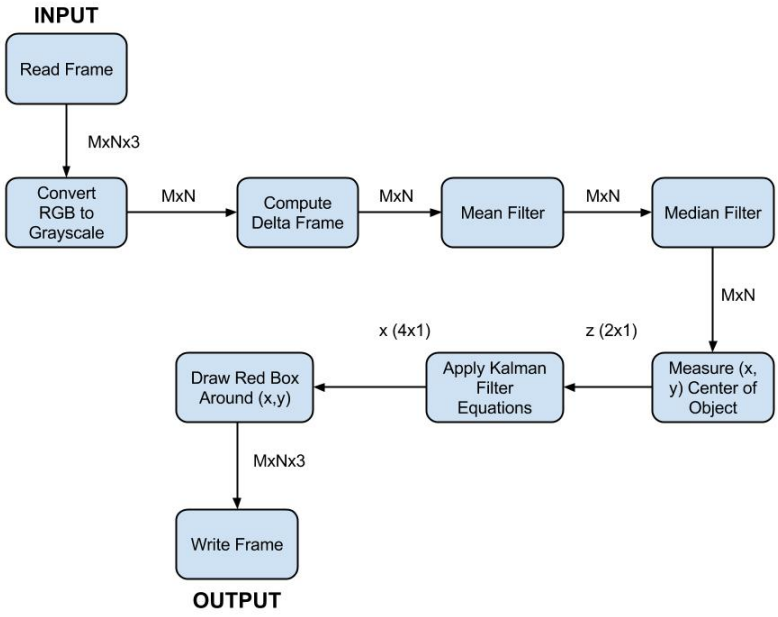

Figure 1: Software Flowchart

6 Future Work

Ben & Taylor

7 Impact on Society

Taylor

8 Allocation of Work

Ben & Taylor

9 Conclusion

Ben & Taylor

References

[1] F. Roth, "Using low cost FPGAs for realtime video processing", M.S. thesis, Faculty of Informatics, Masaryk University, 2011.

[2] A. Saeed et al., "FPGA based Real-time Target Tracking on a Mobile Platform," in 2010 International Conference on Computational Intelligence and Communication Networks, 2010, pp. 560-564.

[3] "Video and Image Processing Design Using FPGAs." Altera. 2007. January 2014. http://www.altera.com/literature/wp/wp-video0306.pdf

[4] E. Trucco and A. Verri, "Chapter 8 - Motion," in Introductory Techniques for 3D Computer, pp. 177- 219.

[5] S.A. El-Azim et al., "An Efficient Object Tracking Technique Using Block-Matching Algorithm", in Nineteenth National Radio Science Conference, Alexandria, 2002, pp. 427 - 433.

[6] Caner et al., "An Adaptive Filtering Framework For Image Registration", IEEE Trans. Acoustics, Speech, and Signal Processing, vol. 2, no. 2, 885-888. March, 2005.

[7] Yin et al. *Performance Evaluation of Object Tracking Algorithm* [Online]. Available: http://dircweb.kingston.ac.uk/

[8] G. Shrikanth, K. Subramanian, "Implementation of FPGA-Based object tracking algorithm," Electronics and Communication Engineering Sri Venkateswara College of Engineering, 2008.

[9] E. Pizzini, D. Thomas, "FPGA Based Kalman Filter," Worcester Polytechnic Institute, 2012.

[10] M. Shabany. (2011, December 27). *Floating-point to Fixed-point Conversion* [Online]. Available: http://ee.sharif.edu/ digitalvlsi/Docs/Fixed-Point.pdf

[11] N. Devillard. (1998, July). *Fast median search: an ANSI C implementation* [Online]. Available: http://ndevilla.free.fr/median/median.pdf

[12] D. Kohanbash. (2014, January 30). *Kalman Filtering - A Practical Implementation Guide (with code!)* [Online]. Available: http://robotsforroboticists.com/kalman-filtering/

Appendices

A Additional Figures