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

VGA - Video Graphics Array

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

The device generates object localization data that can be used by other systems. Systems that require responsive and accurate data for real time use can are the main applications. These secondary systems can now react to moving objects in the selected environment.

Specifically our algorithm tracks a single object from a camera that watches a stationary environment. This makes our design ideal for tracking a disturbance in the field of view. For example, the addition of our device to a camera surveillance system would provide data about where and when disturbances take place. This information could be used to quickly react to disturbances such as a trespassers on private property.

Dedicated hardware allows for simpler connectivity to secondary systems. It presents a modular design which is ideal for testing purposes and debugging. If object tracking needs to be performed on higher definition video, dedicated hardware has its own memory which can be specifically allocated for handling larger image frames. Performance depends on the technical specifications of the hardware being used. Our current hardware is limited due to our budget, however it would be possible to upgrade the hardware to match the needs of the desired application.

The Kalman filter in our design tracks the object more smoothly and can also guess the location of the object if it disappears from the view temporarily. This further enhances the performance of secondary systems by providing them with more accurate and useful object tracking data, which is the main goal of the device.

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

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

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

$$f = \frac{t}{n}. \tag{2}$$

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:

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

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

Optical flow addresses the idea of determining apparent motion based on changes in image intensity (i.e. brightness) over space and time [4]. All of the proposed differential and matching (feature-based) techniques proposed in Trucco & Verri [4] go beyond the scope of this project. The simplest of these algorithms involve derivatives of the brightness constancy equation and a least squares solution at each pixel. Not only would they be difficult to implement in both software and hardware, but they produce what is known as the motion field of the image. This is more information than what is needed for the Kalman filter algorithm that is discussed next, which requires simply knowing the (x, y) coordinate of the object in each frame.

A far simpler, less robust approach is used of locating the object in motion for each frame known as Delta Frame Generation (DFG) [8]. This method makes the following assumptions about the image:

1. The object of interest is in motion.
2. The object of interest is the only part of the scene in motion.
3. The first frame of the sequence does not contain the object.
4. The lighting and background of the scene does not change between frames.

The delta frame is then calculated as

$$\Delta_i = |F_i - F_1|. \tag{4}$$

Where F_i is the current grayscale frame and F_1 is the base grayscale frame discussed in the previous section. Assuming the above assumptions hold, the delta frame, Δ, will be non-zero for only the object. Knowing that Δ contains only the object, drawing a line from the top/bottom most points, and finding its intersection with another line from left/right most points produces the center point of the object. The mathematical background for this is as follows. By means of simple search algorithms, the top, bottom, left, and right most points are located as

$$p_t = (x_1, y_1),$$
$$p_b = (x_2, y_2),$$
$$p_l = (x_3, y_3),$$
$$p_r = (x_4, y_4).$$

Then the two lines are given as

$$l_1(t, b) : y = m_1 x + b_1,$$

$$l_2(l, r) : y = m_2 x + b_2.$$

The intersection of these two lines, denoted (x^*, y^*) is given as

$$x^* = \frac{b_1 - b_2}{m_2 - m_1} = \frac{((x_1 y_2) - (y_1 x_2))(x_3 - x_4) - (x_1 - x_2)((x_3 y_4) - (y_3 x_4))}{D},$$

$$y^* = \frac{m_2 b_1 - m_1 b_2}{m_2 - m_1} = \frac{((x_1 y_2) - (y_1 x_2))(y_3 - y_4) - (y_1 - y_2)((x_3 y_4) - (y_3 x_4)}{D}.$$

The common denominator in both results is denoted D, and can be expanded as

$$D = (x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_1 - x_2).$$

When $D = 0$, the case of parallel lines occurs. In this case, the middle of the object is simply taken to be the average of the top and bottom points.

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:

$$\vec{x_{k+1}} = F \cdot \vec{x_k} + B \cdot \vec{u_k} \tag{5}$$

$$P_{k+1} = F \cdot P_k \cdot F^T + Q \tag{6}$$

Intermediate Calculations:

$$\vec{y_{k+1}} = \vec{z_k} - H \cdot \vec{x_{k+1}} \tag{7}$$

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

$$K_{k+1} = P_{k+1} \cdot H^T \cdot S_{k+1}^{-1} \tag{9}$$

Update Equations:

$$\hat{\vec{x_{k+1}}} = \vec{x_{k+1}} + K_{k+1} \cdot \vec{y_{k+1}} \tag{10}$$

$$\hat{P_{k+1}} = (I - K_{k+1} \cdot H) \cdot P_{k+1} \tag{11}$$

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. The two quantities already discussed are the measurement vector $\vec{z}$ of length m and the state vector $\vec{x}$ of length n. For this application, $m = 2$ since by methods of optical flow discussed previously we are able to get a 2D Cartesian coordinate for the object.

$$\vec{z} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{12}$$

The length of the state vector depends on the model that has been chosen for the system. Two common models for object tracking applications are the constant velocity model and the constant acceleration model. The assumptions made in these models is self explanatory; $n = 4$ for constant velocity and $n = 6$ for constant acceleration. In this system, the constant velocity model is used.

$$\vec{x} = \begin{bmatrix} x_p \\ y_p \\ v_x \\ v_y \end{bmatrix} \tag{13}$$

The next system dependent variable is the state transition matrix F which is nxn. As equation 5 indicates, it describes how the theoretical, predicted behavior of the system changes with each iteration [12]. For a 2D constant velocity model, this matrix is just implementing kinematic equations:

$$x_{new} = x_{old} + t * v_x \tag{14}$$

With a similar equation existing for y. In matrix form this is

$$F = \begin{bmatrix} 1 & 0 & f & 0 \\ 0 & 1 & 0 & f \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{15}$$

Where f is the frame rate (i.e. the time step) described in equation 2. Also stemming from equation 5, the vector $\vec{u}$ describes any external inputs applied to the system between iterations. For a simple, constant velocity object tracking system, there are none. This makes the vector irrelevant as well the associated matrix B. Getting back to the measurements, the only constant matrix that incorporates m is the measurement model H [4]. This is an mxn matrix that relates the predicated state to the measured value. For this application, it should take the vector x of length 4 and place it in the same vector space as z of length 2. Thus,

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{16}$$

The final two constant matrices that must be initialized prior to starting are Q (nxn) and R (mxm). These matrices are related to the error in the theoretical model (i.e. air resistance, friction) and the error in the measurement model (i.e. your "sensors", in this case optical flow) respectively [12]. In theory, if there are vectors $\vec{w}$ and $\vec{v}$ that characterize the error in the model and measurement respectively, then Q and R should be diagonal matrices containing the variance of these vectors [4].

The final, and perhaps most important, value that should be touched on is the co-variance matrix P (nxn) which changes with each iteration. In simple terms, this matrix is a measure of how well the measurements follow the model, and thus how accurate the filter is [12]. Unlike the previously discussed values, P will eventually converge to certain values do to the iterative nature of the algorithm.

3.4 Fixed vs. Floating-Point

Fractional portions of numbers can be represented in either fixed or floating-point in hardware. The essential difference between these two data representations is that in fixed-point the decimal point (i.e. dividing the integer and fractional portions) does not really exist from a hardware perspective. It is the job of the programmer to "know" where the decimal

point is for each binary string, and then use the data under that interpretation. Most modern computing solutions use floating-point numbers, as it allows the user to implement real numbers without constraint. For example, by default all numbers in MATLAB are 64-bit double precision [10]. A 64-bit architecture (i.e. all numbers are 64-bits long) as well as a floating-point unit take up considerable space in hardware [10], and embedded hardware solutions have limited resources and power. It is clear that floating-point data representation will not be possible for an application such as this, and fixed-point conversion of the software is necessary.

Fixed-point conversion works by scaling all real (non-integer) numbers by a large number and then rounding to make it an integer. The idea is that if the number is scaled large enough, rounding will only remove a very small portion of the number, if any. The scale factor is generally used as a power of 2 or power of 10. Powers of 10 are much more intuitive from a programming sense, as humans use base-10, but powers of 2 are much more useful for hardware implementation, as multiplication by a power of 2 represents a shift along a binary string. The object tracking algorithm discussed in the previous sections only uses addition, subtraction, multiplication, and division operations. There is one matrix inverse operation, but since it is a $2x2$ matrix it is treated as swapping entries and dividing. When treating division operations (there are only two) as multiplication by an inverse and subtraction as addition by a negative number, the algorithm can be converted from floating-point to fixed-point by implementing the algorithms for conversion, addition, and multiplication discussed in [10].

MATLAB implementation of the three algorithms just mentioned are shown below. Note that every fixed-point number has an associated word length W and a fractional portion F. W denotes how long the binary string used to represent the number is, and F denotes the location of the decimal point from the least-significant (rightmost) bit.

Fixed-Point Conversion:

```
function [fixed] = floatToFix(float, F)

    d = float .* 2^(F);
    fixed = round(d);

    %Verify that enough fractional bits were used to represent this number
    if (float(1,1) ~= 0 && fixed(1,1) == 0)
        disp('ERROR: The chosen F is too small to represent this value.');
    end

end
```

Fixed-Point Addition:

```
function [result, F] = fixedAdd(fix1, F1, fix2, F2)

    %Normalize
    if(F1 > F2)
        fix2 = fix2.* 2^(F1-F2);
        F = F1;
    else
        fix1 = fix1.* 2^(F2-F1);
        F = F2;
    end

    result = fix1 + fix2;

end
```

Fixed-Point Multiplication:

```
function [result, F] = fixedMult(fix1, F1, fix2, F2)

    result = fix1*fix2;
    F = F1 + F2;

end
```

Going through the floating-point software and replacing all non-integer numbers with their converted value as well as replacing all operations with these algorithms converted the software to fixed-point. Finally, it is important to note that using these algorithms recursively or repeatedly causes the fractional portion F to grow significantly. Due to architecture constraints described above, it is necessary to normalize the fixed-point numbers by "chopping" extra bits when there are too many. This is done in MATLAB by calling `floatToFix.m` with a negative fractional component, and would be done in VHDL or other hardware description languages by simply removing bits since they are accessible binary strings.

4 Requirements

Several requirements had to be met so that the device could be viable for use. We placed a strong emphasis on code efficiency in our software development. Computational time dedicated to video processing needs to be kept at a minimum so that the object tracking is done as close to real time as possible. The main goal is to have a live video feed from a camera displayed on a screen and a cursor tracking the object. This will be visual proof that the device works. If our code was inefficient or too intensive, there would be a delay between what was actually happening in front of the camera, and the output video feed that has the object located. The device would not have much use if there was a lot of lag. There will be some delay as it would be impossible to accomplish the necessary computations instantly, but it is possible to minimize the delay so that it is barely noticeable. This is why

it was crucial to make sure computations are not redone if they don't need to be and unnecessary computations are removed. Every computation needs to be done in the optimal way considering we don't have accesses to more advanced hardware. These optimization challenges will be more apparent when we begin hardware implementation with VHDL. The hardware limitations of the FPGA certainly pose some constraints to our design but we should be able to obtain a good result with it.

All other algorithm based requirements have been met at this point in development. The ability to filter out noise and locate a single object has been implemented in our MATLAB code.

5 Design

5.1 Platform

During the beginning of the project, rapid research was performed in order to determine what the best embedded hardware solution would be to implement the object tracking algorithm on. This design choice needed to incorporate findings from similar projects in order to avoid common failures, as well as factors specific to this project.

The main embedded hardware solutions seen today for applications that required dedicated computing are microprocessors, ASICs, and FPGAs. An FPGA surpasses these other two solutions in a number of ways for video processing solutions [1]:

- Configuration after release for bug fixes and hardware modifications.
- Internal logic of microprocessors is not custom, and thus not application specific.
- ASIC development cost is extremely high.
- Video signals have moved to digital formats (some exceptions such as VGA), making programmable logic solutions more relevant.

Starting with research of similar projects, Roth [1] chose to use an FPGA from the Altera Cyclcone family for a real-time video application over Lattice of Xilinx due to the fact that Lattice had poor online support an Altera was lower cost than Xilinx.

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.

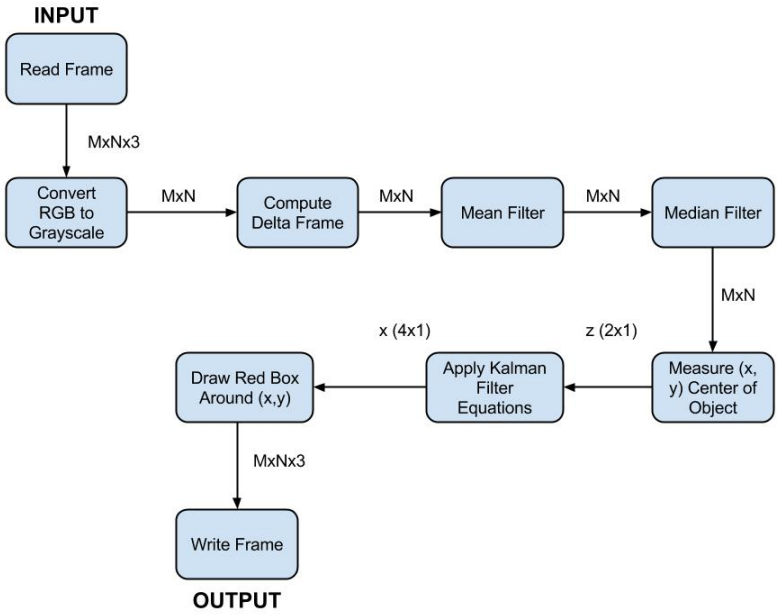

Figure 1: Software Flowchart

6 Future Work

Ben & Taylor

7 Impact on Society

Our dedicated hardware object tracking system will have some impacts in society, but no profound negative impacts. The most notable impact will be its automating effect in the applications it is integrated with. Its introduction to another system will fulfill a task that could have been previously accomplished by a human or even by software previously embedded within the system. The consumer would have to purchase our hardware to implement our system. Our choice of hardware, the Altera DE2 Board has an upfront cost of around $500. This can be a substantial price for the consumer but the performance benefits outweigh that of embedded software, and in the case of surveillance for example, the cost of running our system will be less than paying a worker to watch a video feed for objects. The introduction of new technology that makes a manned job obsolete is often viewed as a negative impact on society, but this sort of progress is inevitable.

In the design process so far, we have not made any sort of environmental impact as we have just been developing software. Our system is going to be a combination of previously manufactured parts which were presumably made with the environment in mind. There are so many electronics already in use in society today, the addition of our system's envi-

ronmental impact can not be quantified. Our product does not produce waste nor does it need constant physical additions. It requires a small constant supply of electricity. When it is no longer needed it can be properly recycled in the same way that computers and other electronics are. Our system does not pose any health or safety risks.

As mentioned earlier, the value of our device comes from the additional performance gained by a secondary system using it. It is possible that the secondary system has some sort of malicious intent but our product cannot be responsible for any negative consequences the secondary system produces.

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