AUTONOMOUS STEREO VISION SYSTEM FOR DEPTH COMPUTATION OF MOVING OBJECT

by

Alejandro Emerio Alfonso Oviedo

Bachelor of Automation Engineering, 2006, Havana University of Technologies

Master of Digital Systems and Computer Electronics, 2011, Havana University of Technologies

A Project

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Engineering

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2017

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this project. This is a true copy of the project, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this project to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this project by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my project may be made electronically available to the public.

ABSTRACT

Autonomous Stereo Vision System for Depth Computation of Moving Object

Master of Engineering

2017

Alejandro Emerio Alfonso Oviedo

Master of Engineering Electrical and Computer Engineering

Ryerson University

This work targets one real world application of stereo vision technology: the computation of the depth information of a moving object in a scene. It uses a stereo camera set that captures the stereoscopic view of the scene. Background subtraction algorithm is used to detect the moving object, supported by the recursive filter of first order as updating method. Mean filter is the pre-processing stage, combined with frame downscaling to reduce the background storage. After thresholding the background subtraction result, the binary image is sent to the software processing unit to compute the centroid of the moving area, and the measured disparity, estimate the disparity by Kalman algorithm, and finally calculate the depth from the estimated disparity. The implementation successfully achieves the objectives of resolution 720p, at 28.68 fps and maximum permissible depth error of ±4 cm (1.066 %) for a depth measuring range from 25 cm to 375 cm.

Table of Contents

List of Tables

List of Figures

Chapter I Introduction

From the beginning of the creation of digital images, one of the main goals of research has been to extract useful information from images, starting from enhancing the quality of the projection, to object detection, recognition, segmentation, and object tracking which was first done in two dimensions, and soon after in three dimensions as well.

Through decades, there has been a significant interest in depth information to be able to incorporate a third dimension to tracking systems, thus a complete vision of a scene is obtained. Observing the way humans and animals solve this challenge, the main general algorithm of using two different views of a scene in order to be able to extract the depth information was generated. Several methods and algorithms, generally known as computer vision, have been developed based on this concept. Some of these algorithms are more complex than others according to their specifications and objectives. For example, approaches to determine the depth map or depth information of an entire scene are highly complex since they give more information. On the other hand, algorithms based only on features of a specific object being tracked are simpler.

1.1 Motivation

Nowadays, with the increase of autonomy in the automobile industry, robotics, and security detecting, locating, and tracking objects in three dimensional scenes is becoming a common feature of most products. In the automotive industry, for instance, car development is heading towards autonomous emergency breaking systems, as well as the autonomous feature of following the vehicle in front, which, in both cases, involves recognizing shapes and keeping certain distance from them. In robotics, this technology is also used to avoid obstacles, as well as in the guidance of a caravan of several units. In this case, the first unit is the only one that has to

be guided. In security, for example, a stereo camera set can detect the distance of an approaching object, and set off alarms when a specified distance is reached.

The main motivation behind this work is that, given the significance of the application of this technology currently, it is extremely important to design systems for implementation and verification in order to explore and evaluate their variants, issues, drawbacks, and best solutions from all possible approaches depending on the target application, and its specifications and constraints.

1.2 Objectives

The general goal of this work is to implement a system able to perform the application of detecting, locating and tracking a moving object in a three dimensional scene, targeting its depth information. More specifically, the implemented system should be able to locate a moving object in the proximities in front of it. Another objective is to choose the platform and algorithms that allow the system to perform at the resolution 720p at 30 frames per second, and depth error no greater than ±4 cm. Finally, the last goal is to include the design of a verification platform as well.

1.3 Stages

Accomplishing the general goal of this work sets a set of specific tasks or stages to be followed. These tasks are related to the specific objectives of this work such as detecting the moving object in the scene, locating the centroid of the object which is the reference point from which the object is tracked, and computing the object depth information. The stages are these:

1. A research on algorithms for detecting movement in a scene has to be done, including the platforms needed for achieving this goal.

2. Once the moving area is detected, other algorithms are needed to create a reference point or centroid. Therefore, more research on algorithms has to be done for this purpose.
3. At this point, the object is located in the two dimensional space, the horizontal and vertical, which brings the challenge of studying the methods and approaches for computing the third dimension, depth or distance, from the system to the centroid.
4. After studying the range of possible algorithms and platforms, the next step is to choose the proper ones to achieve the objectives with more detailed specifications and constraints.
5. The following step is to design the system from a high synthesis level, partitioning it into hardware and firmware components. This step requires analyzing the design in terms of which portions are either computationally intensive or algorithmically intensive, so the first ones are implemented in hardware and the second ones as firmware.
6. Each component needs to be designed according to its functionality, data structure and interface.
7. Finally, the last step is to implement and verify the full design on the hardware and software target platforms.

1.4 Original contribution

This work consists of designing, implementing and verifying a solution to target a specific application. Therefore, its original contribution is described as follows:

1. The research of suitable algorithms and target platforms to solve the application problem.

2. Several solutions and approaches from the wide range of algorithms and methods in the literature are analyzed in order to determine the most appropriate ones to meet the specified objectives.
3. In addition, this work also presents an original design of the methods and algorithms selected, specifically oriented to the selected platform and application to meet its specifications and constraints.
4. Finally, this project also designs and implements original verification methods to analyze the results of the proposed solution in terms of performance and accuracy.

1.5 Organization

Chapter 2 provides a brief overview of some of the algorithms and technologies available and currently in research for developing a stereo vision system capable of locating and tracking a moving object in a 3D scene. It presents a review of algorithms for detecting and segmenting the moving object in static and dynamic backgrounds. From the wide range of existing methods for this purpose, the recursive filter of first order and PBAS are described. Furthermore, regarding object tracking, this chapter explains the effective Kalman Filter algorithm. In addition to this, for depth estimation, the approaches based on correlation and feature detection are reviewed. From the correlation methods, the SGM is comprehensively described as well.

Chapter 3 is the first stage of the system design, where the suitable algorithms are selected and the system is partitioned according to the complexity of the selected algorithms. The general architecture is proposed, including the general blocks to capture the frames, detect and track the moving object, and compute the depth. The hardware and software architecture is described after partitioning the system. In general, a detailed plan of the system design is explained in this chapter.

Chapter 4 describes the details of the actual design, including its implementation and verification. This chapter begins with the specifications and the selection of the target platform. According to the specifications, the camera module is also selected, and its interface and features are explained. The hardware blocks are presented in detail one by one through their interface, algorithms and functionalities, data structure and data flow, and resources utilization. The software portion of the design is also described in this chapter, including initialization, configuration, and constraint verification. The results of this work are all presented in this chapter.

Finally, chapter 5 provides a summary and some conclusions of the work, and highlights potential areas for future research.

Chapter II Background

2.1 Introduction

Several studies have been developed using different approaches to estimate the position of specific objects in a scene, and keep track of their trajectory. Recent studies have included stereo vision platforms and algorithms to estimate the depth of the object as well.

Applications such as visual surveillance, traffic monitoring, vehicle tracking, autonomous navigation, aerospace, and computer vision have the basic requirement of identifying objects and locate them in a scene in real time. For this purpose, there is a wide range of algorithms, from general ones with great accuracy but slow and very expensive in resources utilization, to others with high speed and frame rate, inexpensive and accurate enough for the purpose of this specific application.

This chapter overviews current algorithms and tools to carry out a complete positioning and tracking system for moving objects, such as stereo vision for the depth estimation, camera set calibration, moving object detection algorithms, as well as object tracking algorithms.

2.2 Related Work

Stereo Disparity: Disparity computation algorithms are usually classified in two groups. The first group comprises the most general approaches, which are based on the estimation of the disparity map by correlation-based methods. The second group includes algorithms that compute the disparity of the desired pixel only by mean of feature-based methods. The Feature-based method is commonly applied on object detection/tracking algorithms on both paired images, usually looking for pre-determined image characteristics such as object's centroids, contours, edges and corners.

Correlation-based algorithms essentially find the correspondence of the pixels from one image on the other image by correlating their similarities. Implementing them requires the application of correlation matching methods like SAD (Sum of Absolute Differences), SSD (Sum of Square Differences) and CC (Cross-Correlation). Using these methods is very expensive computationally since the matching has to be executed several times, depending on the maximum intrinsic disparity in the stereo camera platform, which implies high latency and longer cycle times. However, it can be fully pipelined and parallelized, but the use of resources highly increases.

The correlation-based algorithms may be classified as global and local [1]. Global methods exploit various techniques of global optimization of the whole disparity maps, while local methods provide the disparity maps using local optimization of the disparity map around a pixel only.

Global methods include graph cuts technique and belief propagation technique. These contemporary algorithms provide relatively smooth disparity maps. Unfortunately, these methods are highly complex and expensive in terms of resources. They need significant processing power as well as large memory volume. Consequently, the real-time implementations of these methods are subject to extensive research.

These computational demands produce significant problems in mobile implementations. For example, belief-propagation algorithm [2], used for depth estimation with VGA (640×480) resolution and 32 considered levels of disparities, might require as much as 80MB of space for message passing cache. Another more recent global approach is presented in [3], a sequential tree-reweighted message passing (TRW-S) that can be implemented in hardware and at the same time has reliable convergence achieving 22.8 fps for QVGA.

On the other hand, local methods use various types of block matching in order to find the disparity for each pixel independently. Large number of independent block matching allows for massive parallelization of the local disparity estimation algorithms. This enables the usage of even larger blocks (e.g. 35×35 pixels), which mostly provides disparity maps that are better compared to those obtained using small blocks. Unfortunately, large blocks are also very expensive computationally. Hence, the local methods with large blocks are implementable in real-time using the previously mentioned parallelization.

An interesting approach [1] proposes a depth estimation method that would have the advantages of both local and global methods. It uses small-block matching and analyzes estimated disparity values to enhance spatial consistency of the output disparity map. This work achieved very good results, but for resolution 320x240 with only 32 disparity levels. Others like [4,5,6] applied Semi-Global Matching (SGM) methods by Census Transform and Hamming Distance as main matching approaches.

In the work presented in [7], a Guided Image Filter (GIF) is proposed to reduce the complexity of the cost aggregation step, in local Adaptive Support Weight (ADSW) algorithms, for results in the HD video quality 1280x720 at 60 fps implemented on a Kintex-7 FPGA (Field Programmable Gate Array).

On the other hand, more specific approaches to determine the position of a given object in the scene, such as *feature-based algorithms*, can be used as well with very effective results; it all depends on the characteristics of the application. For example, in [8], OpenCV library is used to track up to 8 specific bicolor targets attached to the objects, and then determine their center coordinates per camera. The difference between the two coordinates is the disparity of the particular object. A more sophisticated approach is presented in [9], where Histograms of

Oriented Gradients (HOG) is applied to detect vehicles on the road. Again, this is based on the same idea of creating a blob of the detected object, and then computing the centroid on each camera. Another feature-based method was implemented in [10] by using background subtraction to segment a moving object, and then determine its center of mass as centroid.

The relationship between the disparity and the distance of the object in the scene depends on parameters defined by the geometry of the stereo set, which must be known and remain unchanged for the duration of the image pairs acquisition. Assuming that the two camera planes are perfectly parallel simplifies the algorithm of knowing disparity-depth relationship.

However, even when the camera set is precisely built, the two camera planes will not be perfectly parallel. Moreover, the lenses induce distortions to each camera individually. To overcome these issues, a calibration process is required. Some works such as [8,9] perform calibration to adjust the individual camera distortion, as well as the camera planes, as a pre-processing algorithm using OpenCV and Matlab, respectively. This calibration makes the post-disparity processing easier by increasing delay and computation expenses at the pre-processing stage.

Other works such as [10,11] propose taking several reference points to create a curve of disparity over distance. After obtaining the curve that best fits the experimental data, they are able to create a calibration post-processing function that has to be computed only once per frame. This last aspect confers a great advantage to this approach.

Another interesting method is presented in [12], where both pre-calibration as well as post-calibration stages are applied to increase the accuracy of the system.

Moving Object Detection and Tracking: Robust object detection and tracking are important components of many real world image processing and analysis systems. There are

countless applications in which these algorithms are needed, among them, autonomous guided vehicles, automated video surveillance systems and video traffic monitoring. For this section of the discussion, there are several related works as well. In [13], background subtraction was applied along with Canny edge detection to detect the boundaries of the object. In this case the background was known and static. A similar approach but using Sobel is presented in [14] and in [15] without applying any edge detection filter.

The challenge in this case is that in many real world applications the background elements may not be static. In such cases, naive approaches like subtracting current frame and background image followed by thresholding are not enough to obtain accurate outcomes. On the other hand, false so-called "ghosts" may occur when an object stops moving. In this case, the object becomes part of the background and when it starts moving again is not detected.

A recent approach, presented in [16], addresses this issue by offering an algorithm for dynamic background for object detection based on recursive filter of first order real-time updating background. To track the object, they applied Kalman filter so the trajectory is smoother.

Others such as [17] implemented a Pixel-Based adaptive Segmenter (PBAS) foreground object detection algorithm, classified as multi-variant. This method combines the advantages of both recursive and non-recursive approaches. PBAS consists of creating a background model of N samples. What makes these algorithms different to any typical buffer method is that the relationship between the samples is not relevant.

The challenge of handling small objects in complicated non-flat scenes is targeted in [18] with a more complex algorithm. In this paper, the authors present a robust method to effectively segment moving objects from a moving platform. They propose two-level registration to

estimate and compensate the camera motion. Then, to extract the potential foreground, they apply Gaussian mixture model. They also present the application of Hidden Markov Model (HMM) to classify the pixels. Finally, foreground objects are tracked by a particle filter to improve the detection accuracy.

2.3 Theory

This section reviews the most frequently used approaches for locating and tracking moving objects in a scene, with the particular inclusion of the depth estimation.

2.3.1 Stereo Vision System

In stereo vision, the distance from the object to the stereo camera set is computed by obtaining the disparity, which is the difference of the projected points on the two stereo images, and then applying trigonometry [6,10,19,20]. Using a general configuration of a camera set, where the cameras can be in any position with respect to each other, the equation system to solve the depth from the disparity is very complex. For this reason, it is preferable to establish some conditions in the design of the set of cameras, so the computation complexity decreases. Setting two identical cameras at the exact same high and with their image planes perfectly parallel allows us to measure the distance of the object by using the parallel equation related to the distance information. In such a case, the geometry parameters of the stereo system can be modeled as presented in Figure 2.1.

Each image plane (u, v) is perpendicular to the z axis and parallel to the x, y coordination system plane. Ol and Or are the centers of projection of each camera, and b is the distance between them. A is an object represented by only one point in the world at (X, Y, Z). The plane created by b, Ol, Or and A is called epipolar plane, which intersects the image planes

in lines called epipolar lines. In this particular case we are meeting the epipolar constraint as the two epipolar lines are parallel as well as the scan lines in each image plane. Meeting this constraint greatly benefits the algorithm, because it considerably reduces the search area to a single horizontal line. In Figure 2.2, the distance from the object A to the center of projection plane is denoted by Z.

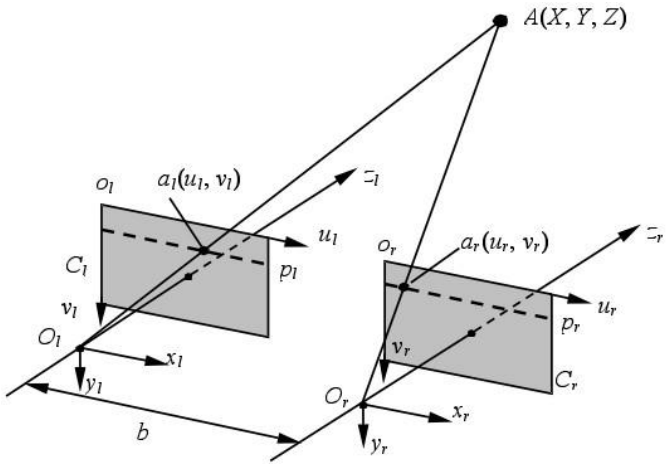


Figure 2.1 Stereo camera parallel model

Via similar triangles, where one triangle is formed by (A, pl, pr) and the other one by (A, Ol, Or), equation (2.1) below can be obtained, where xl and xr are the distances from the center of the image to the object projection. Let us say $(xl - xr)$ is the disparity d. Then after rearranging and simplifying we can obtain (2.2). The conversion model of xl and xr from the image coordinate system to pixels can be obtained from the parameters of each camera model after calibration.

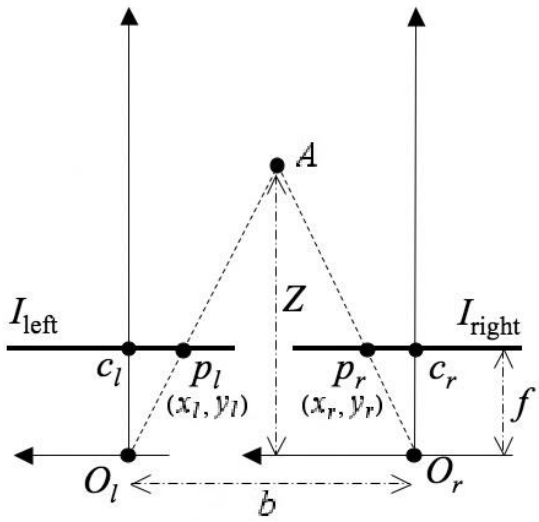


Figure 2.2 Triangulation model

$$\frac{b}{Z} = \frac{b + xl - xr}{Z - f} \tag{2.1}$$

$$Z = f\frac{b}{d} \tag{2.2}$$

2.3.2 Rectification and Calibration

Usually, due to lens distortion or camera misalignment the epipolar lines are not parallel to the baseline, failing to meet the epipolar constraint. Therefore, equation (2.2) cannot be applied to compute the depth from the disparity. There are two general problems related to stereo vision set calibration: the first one emerges from having the individual cameras not calibrated, and the other one from having the stereo set not calibrated to meet the epipolar constraint. If the individual cameras are not calibrated, the problem is that the ground truth disparity for a specific depth may not be constant through the entire image due to the individual distortion on each camera. When the camera set does not meet the epipolar constraint, the depth cannot be

calculated by using the equation (2.2); therefore, a more complex method has to be applied to compute the triangulation.

These issues can be solved through a rectification process, which uses the parameters of each camera to undistort each image, and then, the parameters from the stereo camera set to rectify the images planes [5,6,8,12]. These parameters are predetermined by a calibration process, in which a regression method is used to determine the distortion of a predefined pattern, usually a checkerboard. The process of rectification is a series of rotations, translations and scaling.

There are several tools, such as Matlab and OpenCV, to generate and apply the calibration parameters of individual cameras and stereo set. The process of applying this transformation to the incoming video stream is very complex and demands high computational power; hence, it is a factor that truly affects the performance of the system.

For systems based on correlation matching, this pre-processing rectification-calibration stage is critical. First, it simplifies the matching computation. Second, the better the calibration is the more accurate the disparity map will be.

On the other hand, for feature-based systems there are other alternatives because the depth computation is just over a few points on the image. One of the alternatives for overcoming the epipolar constraint is the offline generation of the relationship between disparity and depth proposed by [10,11], for instance. This mechanism generates a curve from several reference points, which relates the disparity to depth. For this approach to be effective, it is required as many reference points as possible across the entire z-axis. This approach allows to generate two possible models: one depending only on disparity (better for systems with at least undistorted images), and another one depending on disparity, x and y (for systems without any kind of

calibration). The second model is more accurate, because it addresses each camera distortion and the epipolar constraint issue, but for the same reason it is also more complex.

Generating these models requires a tool such as Matlab along with a curve fitting tool or algorithm such as least-square. The reference points may be automatically detected from the images using a similar checkerboard pattern and a feature detection algorithm. The application of this type of calibration is performed after the disparity calculation, which infers that there should be enough time to compute the distance until the next feature is detected and the next disparity is ready.

2.3.3 Correlation-Based Algorithm

Local approaches compute the matching cost just by aggregating neighbouring pixels for each disparity candidate. These neighbouring pixels are determined by a so-called support window, as shown in Figure 2.3. In the reference image, the support window is located centered on point x, while in the target image, there are $dmax$ support windows centered on points from $x - dmax$ to x respectively [20]. As previously mentioned, cost aggregation functions are several from SAD, SSD, and CC to more hardware optimized versions like Census Transform and Hamming Distance. Then, after computing the aggregated cost of all the possible disparities of each pixel, the best disparity of each pixel is determined by the disparity with the best aggregated cost.

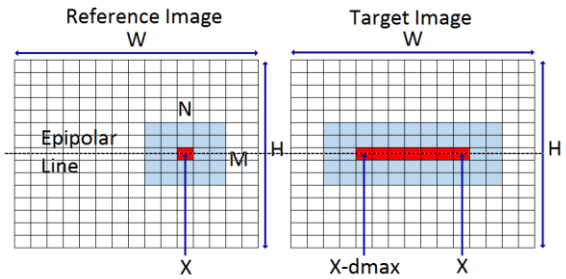
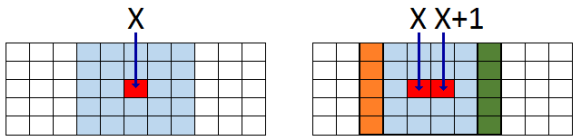


Figure 2.3 Support windows on reference and target image

The number of operations in the local method can be dramatically reduced by optimizing the cost aggregation applying incremental techniques. The full cost aggregation for one disparity candidate requires the accumulation of the cost of all the neighbouring pixels, as presented in Figure 2.4.a. However, in Figure 2.4.b, 1D incremental optimization technique [21] is applied, reducing the computation of the cost aggregation of pixel $x + 1$, just by updating the previous cost aggregation of x by adding the costs in green and subtracting the costs in orange, which means adding the new column of the support window and subtracting the last column that is not part of the supporting window anymore.



a-) *b-)*

Figure 2.4 a-) Full Cost Computation. b-) 1D incremental optimization technique

Adapting weight algorithms aggregate costs according to weights assigned by examining the image content [21]. The accumulation of costs of the support window depends on weights, that put more relevance in some points than in others. For instance, one common method is inspired by bilateral filtering, where points with similar magnitude to the central point have more influence in the overall cost. Another characteristic is that points closer to the central point have also more significance.

Correlation methods with support window implicitly assume that all pixels within the windows have the same distance from the camera. Abrupt changes in depth discontinuity result in wrong correspondence calculation, thus in random disparity outcomes. Matching wrong content near depth discontinuities brings severe correlation errors, particularly for NCC. Consequently, SSD is used more often. In addition, SSD decreases errors due to matching non-

corresponding pixels. Other non-parametric cost functions such as Rank and Census also reduce this problem. Adapting the size and shape of the window can also diminish this problem. Ultimately, a real solution is only possible by matching pixels individually instead of matching windows.

Global Matching approaches use the entire image content to generate the correspondences by applying individual pixel correlation and smoothness constraints that penalizes discontinuities. The general global approach is commonly formulated as in (2.3).

$$E(D) = \sum_{p} \left\{ C(p, D_p) + \sum_{q \in N_p} PT\left[\left|D_p - D_q\right| \geq 1\right] \right\} \tag{2.3}$$

The first term adds all pixels matching costs over the entire image, while the second term introduces a penalty for all pixels with neighbours that have a different disparity. Consequently, discontinuities are permitted only when the matching is stronger than the penalty. The disparity image D is solved by minimizing (2.1). As this is a nondeterministic polynomial (NP) problem, many approximate solutions have been developed such as Graph Cuts and Belief Propagation, but they still have the drawback of low speed and memory consumption.

Semi-Global Matching [5,6,22] effectively combines ideas of global and local algorithms for accurate pixel-wise matching at low runtime. The first SGM implementations used Mutual Information (MI), which is very suitable for unrectified images, because the transformations are irrelevant when only individual pixels are considered. The results of MI degrade with increasing local radiometric differences caused by shadows, for example. Furthermore, it does not scale well with increasing radiometric depth.

As an alternative, Census is identified as the most robust matching cost function for stereo vision. Census maps a support window surrounding the pixel in calculation, creating a vector that only scores if the compared pixel has a lower value than the center pixel. See Figure 2.5 for an example of a transformation [4,20].

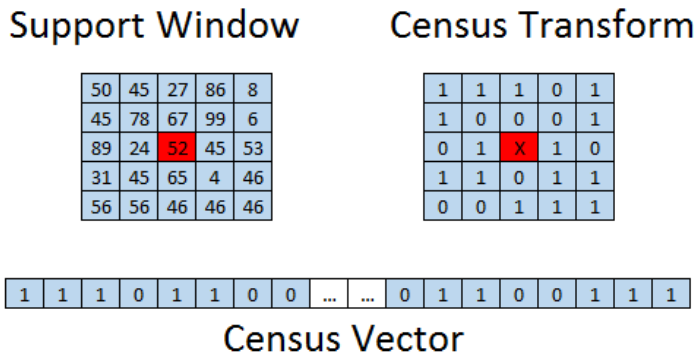


Figure 2.5 Census transformation

Then, the matching cost is computed by the Hamming Distance of the bit vectors of the corresponding pixels from each image. Hamming Distance represents the amount of bit positions that are respectively different from comparing the two vector of the same size. Figure 2.6 presents an example of this calculation, which is done by an XOR operation of the two vectors and then counting the amount of ones of the result.

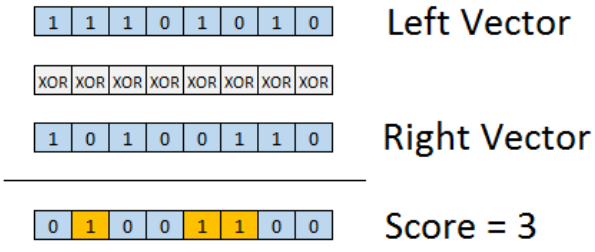


Figure 2.6 Hamming Distance

The principle of Census makes it suitable for global and local radiometric changes. However, it is slightly inferior to MI, if there are only global radiometric changes, and better than MI, for local radiometric changes. The local radiometric changes are present in many real-world applications. *Pathwise Aggregation for SGM* effectively penalizes minor disparity steps as expressed by (2.4) [6].

$$E(D) = \sum_{p} \left\{ C(p, D_p) + \sum_{q \in N_p} P_1 T\left[\left|D_p - D_q\right| = 1\right] + \sum_{q \in N_p} P_2 T\left[\left|D_p - D_q\right| > 1\right] \right\} \quad (2.4)$$

The second term adds a constant penalty $P1$ for all pixels q in the neighbourhood N_p of p, for which the disparity slightly changes. It can be just 1 pixel. The third term adds a large penalty $P2$, for all larger disparity changes, where $P2 \geq P1$.

In order to avoid NP computation, which would be minimizing (2.4) in 2D, SGM brings a novel idea of computing along several paths, symmetrically from all directions. Usually 8 optimization paths are used. Figure 2.7 shows the 8 paths approach. The number of paths should be at least 8. Using 16 paths provides a good coverage as well.

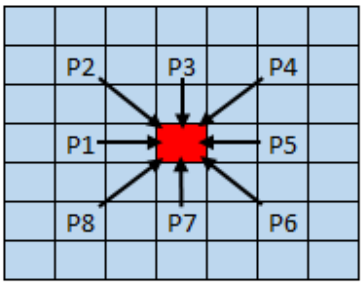


Figure 2.7 Eight paths optimization

Equation (2.5), below, shows the cost path function. $Lr(p,d)$ is the cost of the pixel p at disparity d, and C is the pixelwise matching cost that could be either Census Transformation or MI.

$$Lr(p,d) = C(p,d) \tag{2.5}$$
$$+ \min\Big(Lr(p-r,d), Lr(p-r,d-1) + P1, Lr(p-r,d+1)$$
$$+ P1, \min_i Lr(p-r,i) + P2\Big) - \min_k Lr(p-r,k)$$

The second term computes the minimum over four values. The first one is the path cost at the previous pixel, at the same disparity, and without any penalty. The second and third values are the path cost at the previous pixel, with the next lower and higher disparity, and with a small penalty $P1$ added to them. The last value is the minimum cost at the previous pixel over all disparities, with the additional higher penalty $P2$. The last term subtracts the minimum path cost of the previous pixel from the whole value. This is an approach to limit the constant increase of L, keeping its value as $L \leq Cmax + P2$. The information from all the paths is combined for all the pixels and disparities by (2.6), and the disparity for each pixel corresponds to the minimum cost (2.7).

$$S(p,d) = \sum_r Lr(p,d) \tag{2.6}$$

$$D_L(p) = \underset{d}{\operatorname{argim}} S(p,d) \tag{2.7}$$

The pathwise cost solution does not handle occlusions. However, occlusions can be identified by computing the disparities separately in both directions, from left to right and from right to left, and comparing the results as a consistency check. Further post-processing steps are

possible for cleaning up the disparity image. All the inconsistent disparities are set to invalid, also known as holes. Finally, a hole filling stage is required. This step can be performed by a weighted median filter, for example.

2.3.4 Feature-Based Algorithm

Depending on the platform where the system will be implemented, it is sometimes too expensive performing either the lens distortion or the stereo system rectification to meet the epipolar constraint. For some other applications in which only a single object will be tracked, or even several objects, but all of them known, the feature-based approach is well known for its simplicity and less resources utilization.

In all computer vision system whose main objective is tracking objects by movement, shape, color, or specific features, there will be always a blob. Therefore, from the blob we can compute either the centroid or center of mass. By executing this method from both cameras, left and right simultaneously, it is possible to establish a correspondence between the relative positions of the object from both points of view, which is the definition of disparity [8,9,10].

2.3.5 Moving Object Detection and Tracking

The proper determination of an object motion is significantly important in future stages of computer vision systems, such as object tracking, recognition, and path planning. The main algorithms can be classified in three groups based on the method that they apply. These three groups are background subtraction, temporal difference of successive frames, and optical flow.

Background subtraction is characterized by a simple pixel-by-pixel subtraction of a fixed reference frame or background, and the current frame, in order to determine the difference between them. The result difference is the area where the moving object is in the current frame.

On the other hand, the temporal difference of successive frames has the same principle as the background subtraction algorithm, but in this case the reference is not fixed, it is updated from previous frames. Finally, the optical flow approaches are more complex because they analyze the dynamic scene from a stream in time, and generate a complete motion field of every pixel.

Background subtraction: can be generalized as shown in Figure 2.8, below, where the pre-processing stage could be any specific filter to target brightness changes due to camera exposure (poison noise), or just because of light changes in the scene. Well-known filters for this stage are the mean filter or the median filter.

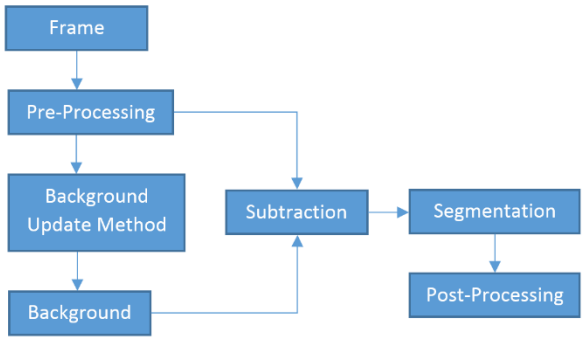


Figure 2.8 Generalized Background Subtraction method

Each new frame after being pre-processed is subtracted from background. Then, in the segmentation stage, the result from the subtraction is compared to a threshold to create a binary image that contains the segmented object extracted from the background. This explanation is mathematically described by (2.8) below, where $S(x, y, t)$ is the result segmented image, T_h is the predetermine threshold, $F(x, y, t)$ is the current frame, and $B(x, y)$ is the reference or background. In $S(x, y)$, all pixels with value of '1' are considered as moving objects.

$$S(x,y,t) = \begin{cases} 1, & if\ |F(x,y,t) - B(x,y)| > T_h \\ 0, & otherwise \end{cases} \qquad (2.8)$$

A stage of post-processing that targets the noise introduced in the segmented image comes after the segmentation. At that point, the object in the segmented image does not have well defined borders. There are also several small artefacts in the rest of the image S creating false detected objects. This noise is usually attacked with either erosion mode or median filter.

Dynamic Background: When scene changes can easily cause false positives results, such as tree branches moving by the wind, there are two effective methods to adequately update the background: the recursive filter of first order, and the PBAS algorithm.

The *recursive filter of first order* is described by (2.9), where $I_{bk}(x,y)$, $I_{bk-1}(x,y)$ and $I_k(x,y)$ are the current background, the previous frame and the current frame, respectively, [16]. Term α is the recursive coefficient, which establishes the background-updating rate and has a value between 0 and 1.

$$I_{bk}(x,y) = (1-\alpha)I_{bk-1}(x,y) + \alpha I_k(x,y) \qquad (2.9)$$

When α is large, the background adapts to changes rapidly. However, when α is small, the system is more suitable for detecting slow velocity objects. When $\alpha = 0$, the new background will be equal to the previous one, then it will not be updated with the new frame. On the other hand, when $\alpha = 1$, the new background will be exactly equal to the new frame, keeping no information from the previous background.

In contrast, *PBAS*, see original articles [17,23], is a more complex algorithm that includes segmentation and background model update. As mentioned before, PBAS is a pixel-based algorithm. In general, the background model is based on a buffer of N samples recently observed from the video sequence $B(x_i)$ as shown in (2.10). The update of this model is performed by random samples with the probability determined by $T(x_i)$, which will be defined later.

$$B(x_i) = \{B_1(x_i), \dots, B_k(x_i), \dots, B_N(x_i)\} \tag{2.10}$$

Pixel x_i is part of the background, when the distances between the value of the current pixel $I(x_i)$ and at least Z amount of samples from the background $B(x_i)$ are smaller than the threshold $R(x_i)$. Equation (2.11) describes the mathematical solution of this model. If F is 1, the pixel x_i is foreground, and background if 0. The distances are defined by the absolute values of the result of subtracting $I(x_i)$ and each background pixel from $B(x_i)$.

$$F(x_i) = \begin{cases} 1, & if \sum_{k=0}^{N} \{dist(I(x_i), B_k(x_i)) < R(x_i)\} < Z \\ 0, & otherwise \end{cases} \tag{2.11}$$

The *background model update* has to be performed to compensate the dynamic changes in the scene. There are two possible upgrading solutions: one liberal and the other one conservative. In the liberal one, all the pixels are updated, while in the conservative one only the ones marked as background are. The conservative method is more convenient because it avoids inclusion of foreground objects into the background model, which is the main disadvantage of the liberal method. However, the conservative method has its own particular problem, which is that once a pixel is updated as foreground, it will not be updated anymore, and therefore it will be stuck at that status forever. There is a solution that can be applied to prevent from getting this error, which is having a counter for the pixels marked as foreground, and then forcing an update once the counter reaches certain threshold. The update is done by, first, randomly selecting a sample from the background model $B(x_i)$, and then replacing it with the value of the pixel $I(x_i)$.

Updating the threshold $R(x_i)$ is done dynamically, so the procedure adapts to the dynamics of the background. Having the measurement of the dynamic of the background is

based on the following procedure. First, whenever an update of the background model is executed, the minimum distance between $I(x_i)$ and $B(x_i)$ is saved in another array $D(x_i) = \{D_1(x_i), \dots, D_N(x_i)\}$. Thus, a history of minimum distances is created for the pixel x_i. The mean of the values stored in $D(x_i)$ defines $\bar{d}_{min}(x_i)$ and the dynamic of the background. Therefore, the decision threshold is dynamically adapted according to (2.12). In [30], the parameters $R_{\frac{inc}{dec}} = 0.05$ and $R_{sc} = 5$ are set for a robust algorithm.

$$R(x_i) = \begin{cases} R(x_i)(1 - R_{\frac{inc}{dec}}), & if\ R(x_i) > \bar{d}_{min}(x_i)R_{sc} \\ R(x_i) * (1 + R_{inc/dec}), & otherwise \end{cases} \quad (2.12)$$

The probability of updating the pixel x_i in the background model is given by (2.13), where $T(x_i)$ is the parameter that adjusts this probability for each pixel.

$$p = 1/T(x_i) \quad (2.13)$$

Updating the learning rate $T(x_i)$ is described by (2.14), where $T_{inc} = 1$ and $T_{dec} = 0.05$. $T(x_i)$ is also limited between 2 and 200 to avoid erroneous results in highly dynamic and fully static backgrounds.

$$T(x_i) = \begin{cases} T(x_i) + \frac{T_{inc}}{\bar{d}_{min}(x_i)}, & if\ F(x_i) = 1 \\ T(x_i) - \frac{T_{dec}}{\bar{d}_{min}(x_i)}, & if\ F(x_i) = 0 \end{cases} \quad (2.14)$$

Kalman Filter is a very popular and effective mechanism for object tracking [24,25,26,27] that not only smoothens the tracking trajectory of the object, but also corrects

possible errors from the position measuring system or sensor. In addition, it adds the possibility of temporarily track the object in momentarily occlusions and sensor failures.

Its theory derives from the assumption that the state of a system at time *t* can be predicted from its previous state at time *t*-1 having the model of the entire system according to the equation (2.15).

$$X_t = F_t X_{t-1} + B_t u_t + w_t \qquad (2.15)$$

In the previous equation, X_t is the state vector, u_t is the control input vector, F_t is the state transition matrix that relates the states at time *t*-1, and *t*, B_t is the control input matrix, which applies the effect of the vector u_t on the state vector, and w_t is the process noise for the state vector derived from a multivariate normal distribution with covariance given by the matrix Q_t. Then, there is an observation or measurement model described by equation (2.16), where, Z_t is the vector of measurement, H_t is the transformation matrix that relates the state vector and the measurement, and $v_t$is the measurement noise derivative from Gaussian white noise with covariance R_t.

$$Z_t = H_t X_t + v_t \qquad (2.16)$$

In case of one-dimensional tracking problem, the model gets the form of the motion equation (2.17). Here, the definitions are as follows: $X_t = [X_t, V_t]$, where X_t is tracked position and V_t is the speed of change. $t-1$ represents the current state and t the predicted state, $F_t = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$; $B_t = \begin{bmatrix} \frac{(\Delta t)^2}{2} \\ \Delta t \end{bmatrix}$, and u_t a scalar number that represents the relationship between the forces that move and break the object.

$$X_t = F_t X_{t-1} + B_t u_t \tag{2.17}$$

The algorithm is based on two main procedures: prediction, and measurement update. The prediction equations are (2.17) and (2.18), where Q_t is the noise covariance related to the noise in the control inputs. For this specific problem Q_t can be defined as (2.19), where the noise magnitude will just amplify the specified noise model. P_{t-1} is initially equal to Q_t.

$$P_t = F_t P_{t-1} F_t^T + Q_t \tag{2.18}$$

$$Q_t = \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & \Delta t^2 \end{bmatrix} * noise\_magnitude \tag{2.19}$$

The next procedure is the measurement update. This process is given by the equations (2.20) and (2.21), where $H_t = [1 \quad 0]$ and K_t is the Kalman Gain. The Kalman Gain has to be updated before the measurement update process, which is before equations (2.20) and (2.21). It is represented by equation (2.22) below. In this example, R_t is a scalar number with the measurement noise amplitude.

$$X_t = X_{t-1} + K_t(Z_t - H_t X_{t-1}) \tag{2.20}$$

$$P_t = P_{t-1} + K_t H_t P_{t-1} \tag{2.21}$$

$$K_t = P_{t-1} H_t^T (H_t P_{t-1} H_t^T + R_t)^{-1} \tag{2.22}$$

At every measurement we wish to know the best possible estimate of the location of the object, the information for computing the estimation is provided by two sources. First, the estimation is made from the last known position and speed, and second from the real

measurement of the position. Therefore, one cycle of steps to compute Kalman algorithm are prediction applying equations (2.17) and (2.18). Then, there is an intermediate step to update the Kalman Gain using equation (2.22). Finally, the measurement update step is done with equations (2.20) and (2.21).

2.4 Summary

In this chapter, a general overview of algorithms and approaches for stereo vision, object detection, and object tracking applications was presented. In stereo vision applications, obtaining the depth information of the scene can be a highly computationally intensive process, demanding expensive resources. For this reason, choosing the right approach for the application is critical. In order to compute the depth information from a scene, the algorithms based on correlation are very accurate and effective, but at the same time highly computationally expensive. On the other hand, the feature-based algorithms are accurate and effective enough for some applications, with the advantage of being less complex, requiring less computational resources.

Another algorithm also reviewed in this chapter was the Background subtraction, which is a well known approach to detect moving objects. It can be combined with the recursive filter of first order as background update algorithm for targeting dynamic backgrounds with great results. Finally, the trajectory of the tracking object can be smoothened by applying the Kalman filter. Kalman algorithm also allows to decrease measurement errors created by noise.

Chapter III System Architecture Development

3.1 Introduction

In chapter I, the main objective of this work was defined as the implementation of a system able to perform the application of detecting, locating and tracking a moving object in a three dimensional scene. In order to achieve this general objective, the process was divided into different stages or tasks to be accomplished, such as detecting the moving object, computing its centroid, and then from the centroid calculating the depth. Still, all these steps are very general; therefore, in this chapter, they are described in detail. In addition, the selected algorithms to be used are described as well.

3.2 General Architecture

In every computer vision algorithm images are needed. Consequently, a camera is required in order to capture the scene and its changes. Figure 3.1 shows the block diagram of the general architecture to be designed for this project in order to build a system able to perform the required application.

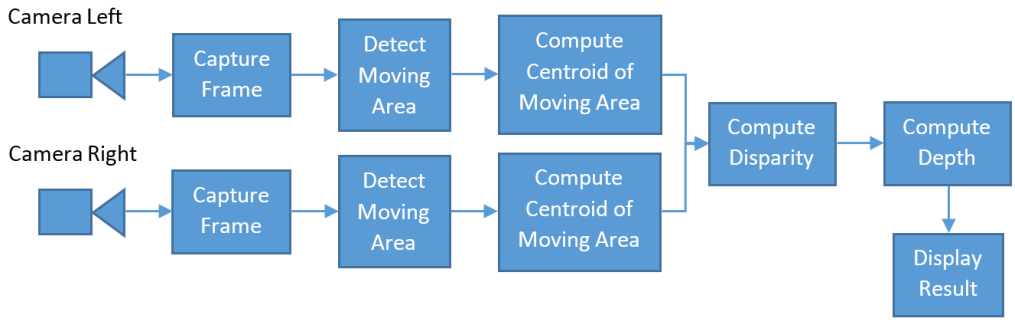


Figure 3.1 General Architecture of proposed design

The first component to include in the system architecture is a Camera Module (CM) with the capability of streaming a video sequence taken from the scene in front of it. The CM outputs frame by frame the scene, allowing next modules to further analyze each frame. The Capture Frame Block (CFB) is a helper that understands the CM interface and converts the streaming frame into a regular interface for next modules.

After capturing a frame, it is required to extract the area that is moving or changing in it. That is the reason why Detect Moving Area Block (DMAB) is placed after the CM. This module is in charge of segmenting each frame. Through this process a segment of the frame is marked as static (no moving object in this segment), and the rest is marked as moving (moving object in this segment). The Compute Centroid of Moving Area (CCMA) computes the reference point of the moving segment. This way, the system is able to track the moving object always according to the same reference. In this work, the center of the moving area is selected as centroid. Following the analysis in chapter II, to extract the depth information from a scene it is required to have two different points of view of the same scene at the same time –this is known as stereo vision. Hence, the portion of the system described so far is reproduced to acquire the second view of the scene. As a consequence, the system is made of two separated channels of CM, CFB, DMAB and CCMA, one for the left view, and one for the right view. These two channels are connected to the Compute Disparity Module. This module calculates the difference between the two points of view of the same object, which is known as disparity. Having this disparity as input, the next module, Compute Depth, is able to determine the distance from the object to the center of the stereo camera set. The last block, Display Result, is for debugging and verification purposes.

3.3 Algorithm Selection and System Partitioning

3.3.1 Hardware

Detecting moving area

In order to detect the moving object, this work proposes the algorithm background subtraction. Background subtraction is simple to design and it does not require many resources, as it works locally on each pair of pixels. This method subtracts each pixel of the current background from the corresponding one of the new frame. This characteristic makes it suitable for processing on a streaming data structure. Then, it compares the absolute value of the difference with a threshold to create a binary image that segments both portions of the frame, the static one and the moving one. Background subtraction is also very effective with some pre-processing and post-processing filters.

One of the challenges to overcome when using Background subtraction is to exclude lighting changes in the scene, and poison noise from being segmented as moving area. That is why the mean filter is selected in this case. This work combines mean filter with downscaling of the frame to solve this issue as pre-processing module before Background subtraction. Downscaling the frame allows the system to store the background directly in internal memory, which is faster. As a result, this part of the algorithm can be implemented in hardware. Each frame is captured and then downscaled to $1/x$ of the resolution by the mean of all the pixels in squared regions determined by x. Figure 3.2 represents an example of downscaling to $1/8$ by mean of 8x8 pixels blocks (64 pixels). In this case, each block of 8x8 pixels in the bigger image becomes 1 pixel in the small image. For an initial resolution of 640x480, the $1/8$ scaled resolution is 80x60 pixels.

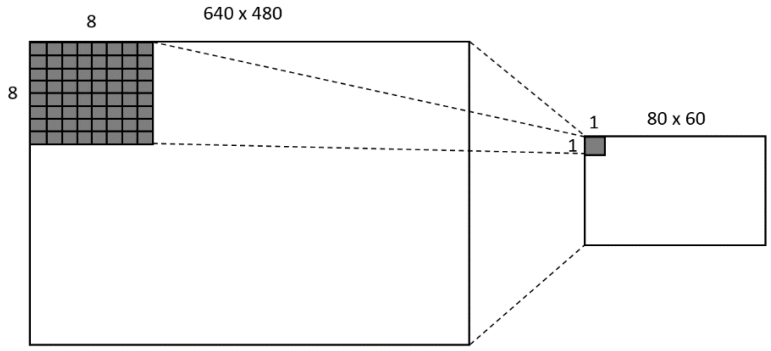


Figure 3.2 1/8 Down-Scale example by mean of 8x8 pixels blocks

Other small changes in the scene, known as Dynamic Background, are another important task to solve. A well-known recursive filter of first order background update is presented in this project to deal with this situation. As mentioned in chapter II, the recursive filter of first order background update is a pixel wise update algorithm. This feature makes it suitable for being implemented in hardware, as the involved operands per cycle are only the corresponding pixels of the new frame, and the current background frame already stored.

3.3.2 Software

Centroid

Up until now, the design has created a binary image segmenting the area where there is movement. Subsequently, the centroid of this area has to be determined in order to track the target object. This project also has the objective of detecting multiple moving objects in future work. For this reason, the detection of centroids is left to be performed in software due to its complexity for multiple object detection. For the specific work presented here, the system only detects a single object.

The extraction of the centroid of the object presented in this work consists in determining first the bounding box that delimits the object, and then the center of such box, which is the centroid. The bounding box is determined by four sides: top, bottom, left, and right, where top is the minimum Y pixel position segmented as moving object, bottom is the maximum Y, left is the minimum X and right is the maximum X. The centroid is then computed as expressed by equations (3.1) and (3.2).

$$Centroid\_X = (left + right)/2 \quad (3.1)$$

$$Centroid\_Y = (top + bottom)/2 \quad (3.2)$$

Disparity

Once the system has calculated the centroid of the moving area from the two cameras, it calculates the disparity by determining the difference between the two centroids X coordinates. Being more specific about this calculation, a valid disparity is determined by (3.3). It is important to note that a valid disparity is when this difference is positive.

$$Disparity = Centroid\_X\_Left - Centroid\_X\_Right \quad (3.3)$$

This disparity value is known as the measured disparity, because it is calculated by processing the sensors, which are both, left and right, cameras. This measurement is not always perfect because it is subject to noise, either from the environment components or from the algorithm itself. That is the reason why an estimation approach on top of the measurement is considered in this work to compensate the noise and smoothen the tracking trajectory. The estimation approach is based on Kalman algorithm. This algorithm generally involves floating-point computation, update and storage of almost all the variables implicated in the algorithm, and

update of the estimation model. According to these characteristics, Kalman algorithm is more appropriate for being implemented in software.

Depth

Depth computation is also a floating-point function based on the characteristics of the stereo vision camera set that is suitable for software as well. This function is a model that relates disparity to depth. This work determines the model of the stereo camera set offline. Taking pictures of several reference points where the distance is known, and then extracting the disparity for every reference point make possible to create the needed relationship function between disparity and depth. Afterwards, this information is entered in a math software tool to extract the function model.

3.4 Hardware Architecture

In this section, the hardware architecture is explained in more detail. Figure 3.3 represents a block diagram of the hardware architecture including all components proposed by this work to achieve its objectives. There are four main areas in this architecture: the algorithms implemented in hardware, a CPU (Central Processing Unit) platform to execute the software algorithms and configuration, the communication components between hardware and software, and finally verification components.

First, the two stereo channels, left and right, require to be configured. At the start-up of the system, the CM is initialized and configured by the CPU with the desired frame resolution, output format, and several other parameters. At any point of execution, the CPU is able to reset the system and reconfigure the CM with a different set of parameters. The CPU transmits the CM configuration commands to the hardware module SCCB (Serial Camera Control Bus) which

is able to properly communicate with the CM using the SCCB standard protocol [28]. The output of the camera is connected to CFB. This module is also initialized and dynamically configured by the CPU with the frame resolution that the camera has been programmed. This block also has an enable signal connected to the CPU to either enable or disable the capturing process. When CFB switches from disable to enable, it waits for a valid start of frame to start capturing, this way the first frame is captured from its beginning. On the other hand, when it switches from enable to disable, it finishes capturing the entire frame it is currently on before stopping, again ensuring the full frame is acquired.

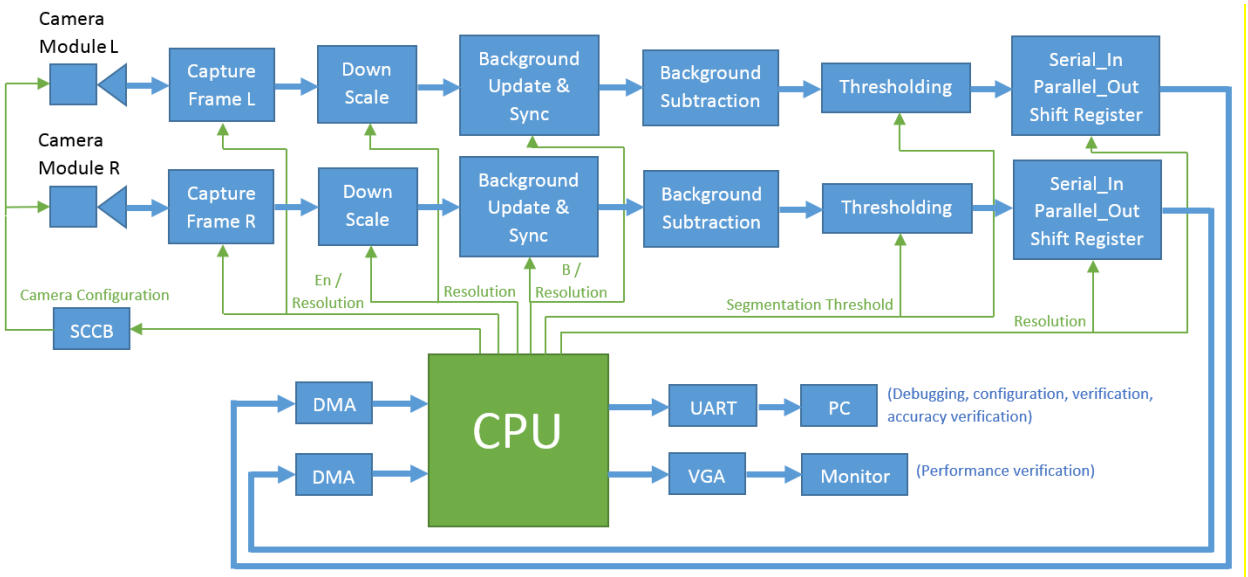


Figure 3.3 Hardware Architecture of proposed design

The CFB is connected to the Down-Scale Module (DSM), which is also configured by the CPU with the frame resolution the system is working with. According to the working resolution, this module is responsible for grabbing the frame coming from CFB and downscaling it $1/x$ of the resolution. First, the algorithm creates blocks of x by x pixels, and then computes the average of each block. Once it finishes with the full frame, it outputs the downscaled frame to the next module.

The downscaled frame is then passed as input to the Background Update and Synchronization Module (BUSM). This particular module is initialized and reconfigured by the CPU with the update rate parameter of the recursive filter of first order background update (α). In order to avoid a floating point computation because $0 < \alpha < 1$, this work modifies the background update equation (2.9) from chapter II by transforming $\alpha = \frac{1}{\beta}$, where $1 \leq \beta \leq max$. Thus, the BUSM parameter is β instead. The modified representation is presented by the expression (3.4). In this case, when β is small, the background adapts to changes rapidly. On the other hand, when it is large, the system is more appropriate for objects moving slowly.

$$I_{bk}(x,y) = \frac{(\beta - 1)I_{bk-1}(x,y) + I_k(x,y)}{\beta} \tag{3.4}$$

The background storage elements have some intrinsic latency. This feature is the reason why the synchronization part of the module is intended to synchronize the incoming current frame pixel with its corresponding background pixel. This action requires some additional registers depending on the storage element latency.

BUSM is connected to Background Subtraction Module (BSM). A pair of corresponding pixels, background and current frame, are the input to BSM, which subtracts them and computes the absolute value of the result. Then, the Thresholding Module (THM) compares the absolute value with a programmable threshold configured by the CPU. THM segments the image frame into two parts: moving area and static area. Its output is a binary image. Both binary images, from left and right cameras, are pushed to external memory through DMA (Direct Memory Access) for further processing by the CPU.

As the images are binary, every pixel is one bit and DMA is able to transfer words of a predetermined size. Therefore, a serial-in parallel-out shift register is used to decrease the

amount of DMA transactions. The shift register accumulates the binary pixels into words of size equal to the DMA data bus size, decreasing the amount of transactions by that size.

Debugging, verification, performance, and accuracy measurement are very important aspects to implement in every design. As part of the experimental setup, a UART communication channel connected to a PC, as well as VGA capability connected to a monitor are added to the system prototype. The UART channel introduces debugging capabilities by setting checkpoints so all the components are properly configured, dynamic reconfiguration of the segmentation threshold, background update rate, and resolution. In addition, it allows measuring performance by setting extra checkpoints to make sure that the frame rate specification is met. Also, in terms of accuracy, the system is able to display the exact depth value that it computes, this way the result can be compared with the ground truth. VGA capability connected to a monitor is another way of performance verification by visualizing how fast and precise the trajectory of the object is updated.

3.5 Software Architecture

The software architecture includes the algorithmically intensive methods of the design, and it is divided into three main parts: initialization frame acquisition, and depth computing. In the initialization and start stages, the system configures for the first time all the components and sets all the variables, starts the system, and then enters in a loop executing compute depth block. Segmented frame acquisition is the block called when the DMA receives a segmented frame. This block initiates the reception of another frame and computes the bounding box and centroid of the received frame. Finally, compute depth block waits for the two frames, left and right, to be ready, which means received and calculated centroid. Then, it computes the measured disparity, the estimated disparity by Kalman algorithm, and depth, and finally shows the results.

3.5.1 Initialization, start and operation block

Initialization is where the software algorithm starts. It starts off by configuring the general platform components as presented in Figure 3.4. The first component to be initialized is the UART communication with the PC to start debugging the system immediately. Right after, the DMA for left channel, and then the DMA for right channel are also configured. Then the system setups the interrupt controller, so the two DMA channels can interrupt when a full segmented frame is received. After the general components are initialized, the application specific components are configured as well. This part of the process begins by configuring the CM. The CPU writes the proper registers of the camera with the correct parameters. This way, it delivers the frames accordingly. The most important parameters that define the behaviour of the rest of the modules are the resolution and the output format. The CFB and DSM are also initialized with that same resolution. Then, the BUSM is also configured with the parameter background update rate according to the scene speed. Lastly, the software initializes the THM passing the segmentation threshold parameter to it.

After initializing and configuring the general platform and the application specific components, the system is ready to switch to operation mode. Starting the operation mode is done by enabling the CFB. Then the CFB starts capturing frames and passing them to the rest of the system. Once started, the application enters in a loop of depth computing and showing results. At this time, the block that computes the depth is called.

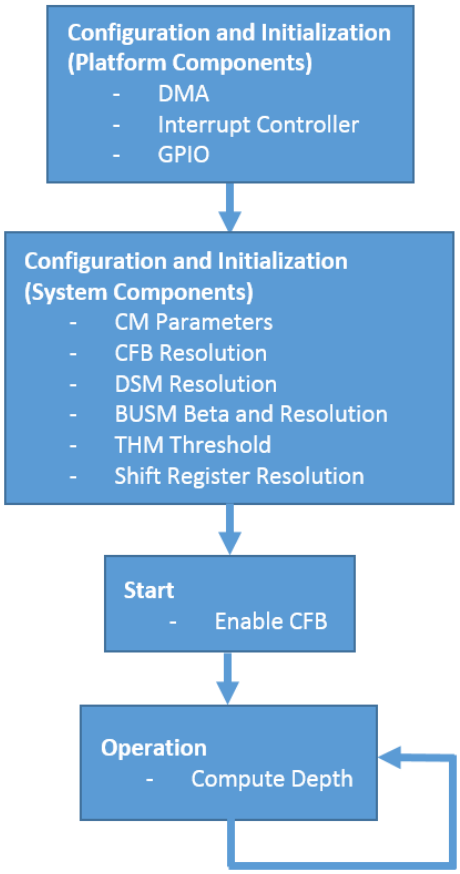


Figure 3.4 Software Architecture: Initialization, start and operation block

3.5.2 Segmented frame acquisition block

The segmented frame acquisition block has two main parts: the DMA interrupt handler, and, after it, a section to determine the bounding box and centroid. This block is repeated, as there are two acquisition channels, left and right. Figure 3.5 represents the segmented frame acquisition block. The DMA interrupt handler will be called when an entire segmented frame is received. Each channel invokes its own handler. Inside the interrupt handler, the application acknowledges receiving the frame, initializes the DMA channel to receive another frame, and calls the block for computing the bounding box and centroid. Once the bounding box and centroid are ready, it sets a flag for that channel to tell that it is ready for depth computing. The

block that computes depth is always waiting for these two flags to be ready to be able to go ahead and determine the depth from the left and right centroids.

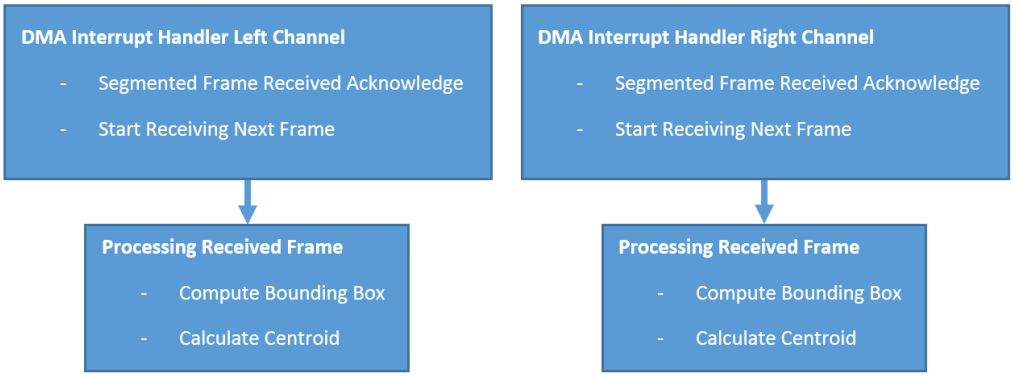


Figure 3.5 Software Architecture: Segmented frame acquisition block, Left and Right Channels

3.5.3 Compute depth block

The compute depth block is invoked after enabling CFB. Then, the application enters in a loop where it waits for both receiving channels' flags to be ready. Once they are both ready, the block computes the depth, shows the results, resets the flags, and returns back to waiting again. Figure 3.6 shows the block diagram of this portion of the application.

The first step is to calculate the measured disparity by subtracting the x axis of both channels' centroids as previously presented in equation (3.3). Then, this new measured disparity is used as the input to Kalman algorithm to compute the estimated disparity. Kalman parameters for the specific model of disparity will be determined experimentally. The model for disparity is a one variable estimation and its speed of change. After setting the initial values of the Kalman parameters, they can be dynamically modified according to changes in the behaviour of the model. In this work, only fixed parameters are used.

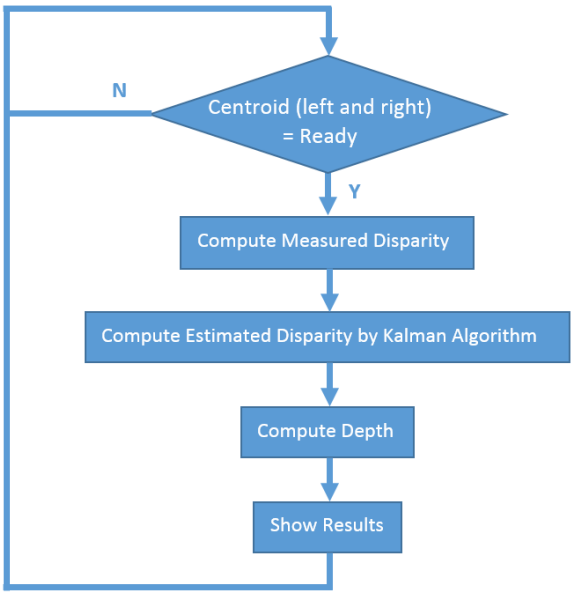


Figure 3.6 Software Architecture: Compute depth block

The disparity estimated by Kalman is converted into depth by using a model of the stereo camera set that relates disparity to depth. In this work, the stereo camera set model is a function previously determined offline that takes as input the estimated disparity and computes the depth. If the stereo camera set is changed, the model has to be recalculated and updated in the application.

Finally, the system shows the results in two different ways. The first way is by sending the depth through UART to the connected PC in order to show the exact measurement of the depth in a terminal window. The second way is more visual, as it represents a two-dimensional tracking map on the attached monitor. The depth is represented on the monitor by scaling it according to the monitor resolution and the depth range of measurement. The equation (3.5) describes this solution, where $Vres$ is the vertical resolution, $Depth_{max}$ is the maximum, $Depth_{min}$ is the minimum depth that the system wants to represent on the monitor, and $VerPos_{VGA}$ is the vertical position where the object has to be drawn on the monitor to represent its position.

$$VerPos_{VGA} = \frac{Depth * Vres}{Depth_{max} - Depth_{min}} \quad (3.5)$$

After showing the results, the application resets the ready flag of both segmented frame acquisition channels and returns back to main to wait for the next set of ready frames.

3.6 Summary

In this chapter, the general architecture of the system was presented with the main objective of locating and tracking a moving object targeting its depth information. As a general approach to achieve the main objective of this work, the system first detects the moving object in the scene from two different views, and then computes the centroid of the object as well from the two different perspectives. The system calculates the disparity using the difference from the two centroids, and finally the depth information from the disparity.

The general architecture includes two video acquisition channels, left and right, to capture the scene from two different views. Each channel is filtered by mean filter, then down-scaled to reduce the background storage. Background subtraction is the main algorithm implemented to detect the moving object. The background is updated applying the recursive filter of first order on the previous background and the new frame. The Background subtraction and segmentation modules create a segmented binary image with the information of the moving area from each channel. These segmented binary images are sent to a CPU via DMA to extract the bounding boxes and the centroids of the moving object per channel. Finally, the CPU is also in charge of computing the disparity from the centroids and smoothen its trajectory using Kalman filter. The CPU then calculates the depth of the object evaluating the disparity in the stereo camera system model.

Chapter IV Implementation, Verification and Analysis

4.1 Introduction

In the previous chapter, the algorithms to be use to accomplish the objective of this work were selected, analyzed and partitioned. In this chapter, the software and hardware implementation of those algorithms is explained accordingly. In addition to this, the specifications of the design are presented, and the platform selected for the system prototype is also described.

4.2 Specifications

In this section, the first set of specifications is presented. These specifications are based on the main objective of this work, which is to implement a system able to detect, locate and track a moving object in a three dimensional scene, at the resolution 720p at 30 frames per second. The 720p video standard is the resolution of 1280 horizontal lines and 720 vertical lines, for a total of 921600 pixels per frame. As it is at 30 frames per second, one full frame including visible and non-visible area happens in 33.33 ms. The design most be able to process a full set of stereo vision frames in less than 33.33 ms. The implementation requires a feature to measure the specified performance as well as the accuracy of the depth tracking.

4.3 Platform

The general platform required to implement the set of algorithms selected in this work includes the following components: a set of two cameras to build the stereo vision set, a development board to implement the hardware and software design, a PC, and a monitor. In this particular case, the camera selected is the Omnivision Complementary Metal–Oxide–

Semiconductor (CMOS) OV5642 [28, 29, 30], which is able to provide the resolution and the frame rate specified. This work proposes the Digilent ZYBO development board [31] as hardware/software platform because it includes a Xilinx Zynq-7000 family [32, 33], the Z-7010. The Z-7010 is based on Xilinx System-on-Chip (SoC) architecture, which integrates a dual-core ARM processor with Xilinx 7-series Field Programmable Gate Array (FPGA) logic. This development board is equipped with one PMOD (Peripheral Module) port connected to the CPU for configuring the camera and four more PMOD ports directly connected to logic useful for interfacing the cameras output and further processing.

4.3.1 Omnivision (CMOS) OV5642 Camera Module

The OV5642 camera module is a low voltage, high performance 5 megapixels CMOS image sensor. It provides all required image processing functions, including exposure control, gamma, white balance, color saturation, hue control, compression engine, etc. This module has an image array capable of operating at up to 7.5 frames per second (fps) in 5-megapixel (2592x1944) resolution and YUV output format. **Error! Reference source not found.** presents the camera pinout with description. The OV5642 can also operate at the following resolutions:

- 1080p (1920x1080) at 15 fps YUV
- **720p (1280x720) at 30 fps YUV**

The specifications dictate that 720p at 30 fps is the resolution and frame rate at which the system should operate; therefore, the camera module has to be programmed to run as the specifications. Figure 4.1 below shows the OV5642 interface used in this work with 8 bits of data. The camera module is powered from a single +3.3V power supply. An external clock provides the clock source for the camera module through *xclk* pin, which is generated inside FPGA logic. With proper configuration of the camera internal registers via SCCB interface

(SIOC, SIOD), the camera supplies pixel clock (*pclk*) and data (*data*[7:0]) back to the FPGA with synchronization signals like *href* and *vsync*. One full PMOD port is required to interface the data lines and power the camera. Four more lines are also needed for synchronization signals. In addition, two extra lines from the same PMOD are used for configuration. The camera is configured to have *href* active high and *vsync* active low. The data output is programmed for YUV422 (UYVY).

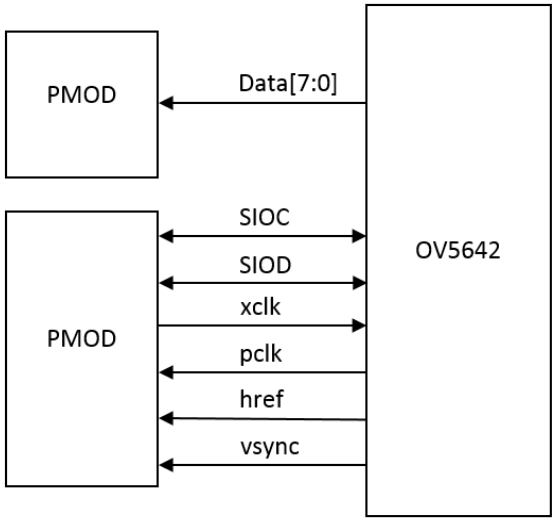


Figure 4.1 OV5642 8-bits data interface

As stated before, the CM is configured to output a resolution 1280x720 in the format UY_1VY_2. The following format transmits two pixels using 4 bytes. The two chrominance, U and V components are the same for the two corresponding pixels. On the other hand, luminance Y_1 and Y_2 are the brightness components of the first and second pixel respectively. In this work, only luminance is used as image information, which means that only the gray scale information from the scene is processed. In Figure 4.2, the output row timing diagram is presented. The luminance value of the first pixel (Y_1) is the second data byte transmitted by the camera. Then, the second pixel (Y_2) is the fourth byte. Note that *href* and Data have to be sampled at the rising edge of *pclk*. The signal *href* tells that the camera is transmitting a valid row and, as discussed

before, it is active high. Therefore, when *href* is high, for every rising edge of *pclk* there is a valid byte on data bus that must be sampled.

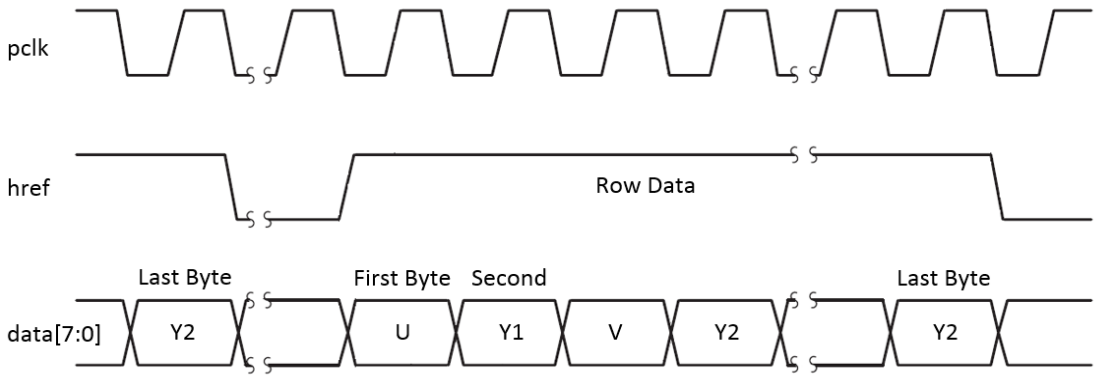


Figure 4.2 OV5642 Output row timing diagram

The frame timing format is shown in Figure 4.3 below. The signal *vsync* is active low and represents that a valid frame is being transmitted. Inside the active state of *vsync* (low), there are 720 *href* high states representing the 720 rows. The camera maximum *pclk* frequency is 96 MHz for a 30 fps YUV output data format at 720p resolution. Therefore, the proposed design has to be able to perform at a frequency of at least 100 MHz to properly process the scene at 720p and 30 fps.

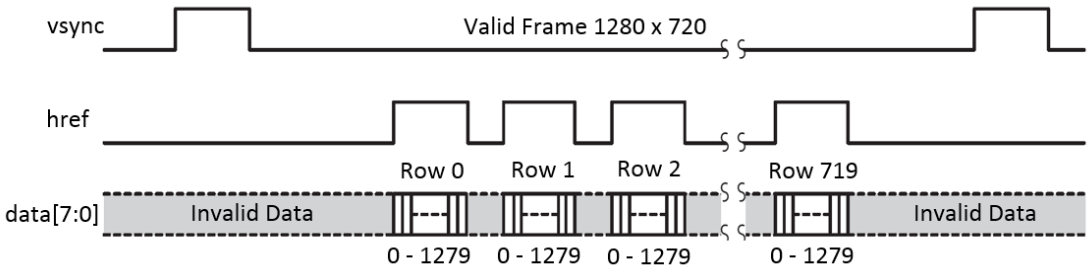


Figure 4.3 OV5642 Frame timing diagram

4.3.2 Digilent ZYBO development board

From all the features the Digilent Zybo development board includes, this description only focuses on the ones related to this work. As mentioned before, this board offers as main computational power a SoC from the Xilinx Zynq-7000 family, the Z-7010, which contains a dual-core ARM processor Cortex-A9 and FPGA logic equivalent to Artix-7. Zybo board is a ready-to-use platform great for prototype designs. It can handle a wide variety of embedded applications because of its computational power, interfaces, and availability of expansion. Its main features related to this work are as follows:

- 650 MHz dual-core ARM processor Cortex-A9
- DDR3 memory controller with 8 DMA channels and 512MB x32 DDR3 with 1050Mbps bandwidth
- Low-bandwidth peripheral controller: UART
- Reprogrammable logic equivalent to Artix-7 FPGA
- 16-bits per pixel VGA source port
- On-board JTAG (Joint Test Action Group) programmable and UART to USB converter
- Six PMOD connectors

4.4 Hardware Implementation

4.4.1 Capture Frame Block (CFB)

The Capture Frame Block is the module that interfaces the CM. In general terms, it is driven by the clock generated by the CM (*pclk*) and has three sets of interfaces: CM interface, standard *axi_stream* interface, and configuration interface. Figure 4.4 shows the three interfaces

presented in CFB. The signals *pclk*, *vsync*, *href* and *data* come from the CM. The *axi_stream* interface includes all the necessary signals to transfer the captured pixels to further processing blocks, as a standard communication interface. Its signals are identified with the prefix "*m_axis_*" followed by the name of the signal [34]. The last interface is connected to the CPU for configuration and control such as *frame resolution*, *enable*, *capture*, and *reset*.

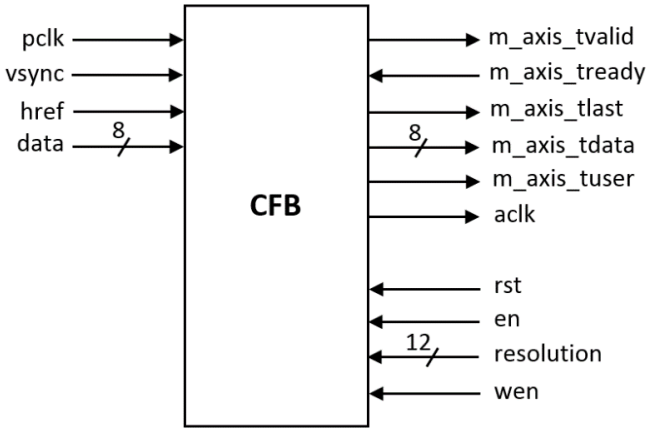


Figure 4.4 CFB block interface

The CM interface was explained earlier in this chapter. The *axi_stream* and the configuration interfaces are as follows:

- m_axis_tvalid: (output) when active, the value on *m_axis_tdata* is valid to be sampled.
- m_axis_tlast: (output) when active, indicates the last pixel of a row.
- m_axis_tdata: (output) data value of the pixel (luminance).
- m_axis_tuser: (output) when active, indicates the first pixel of a frame.
- m_axis_tready: (input) when active, the next module is ready to receive data.
- aclk: clock to synchronize the next module.
- rst: (input) resets the module to its initial state. After reset, the module maintains the last configured resolution.

- en: (input) enables the capturing mode.
- resolution: (input) signal used to set the horizontal resolution.
- wen: (input) *write enable* signal to write the resolution.

There are several parallel functionalities implicated in this module. Initially, the inputs coming from the camera are all registered, consequently all these processes work on the registered input signals. That is one more clock cycle to be added to the latency of the module. In this description, the signals with *"_registered"* at the end of the name are the input signals already registered in a previous cycle. The module has an initial default resolution, which can be reprogrammed at any time following the sequence of *disable capturing*, *write resolution*, *wait* for the equivalent time of one full frame and then *enable the capturing* back again. The horizontal resolution is used to determine the end of every row in the image frame. Figure 4.5 presents the flow of configuring the horizontal resolution, the *eol* (end of line) computation, and the *sof* (start of frame) detection. The register address is initialized with zero when *vsync_registered* is not active for blank interval between frames.

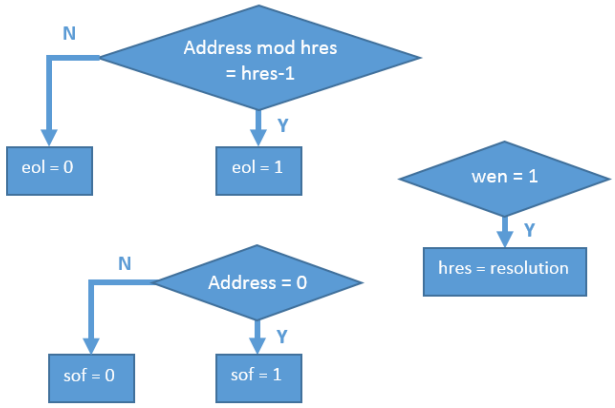


Figure 4.5 CFB eol, sof and resolution configuration

The challenge of detecting the active portion of a frame starts from detecting when *vsync_registered* is active. When it is not active, the address register should be reset as well as a flag for extracting only the luminance (*href_last*) from data input. At this point, *enable capturing* is also checked and set to internal registers (*reg_en*). The signals *href_last* and *reg_we* are used to extract only the Y from the data input in the format of UYVY, which is every other data input byte starting from excluding the first byte of every horizontal line. Notice that address is incremented when *reg_we* is active, meaning the new pixel value has been captured.

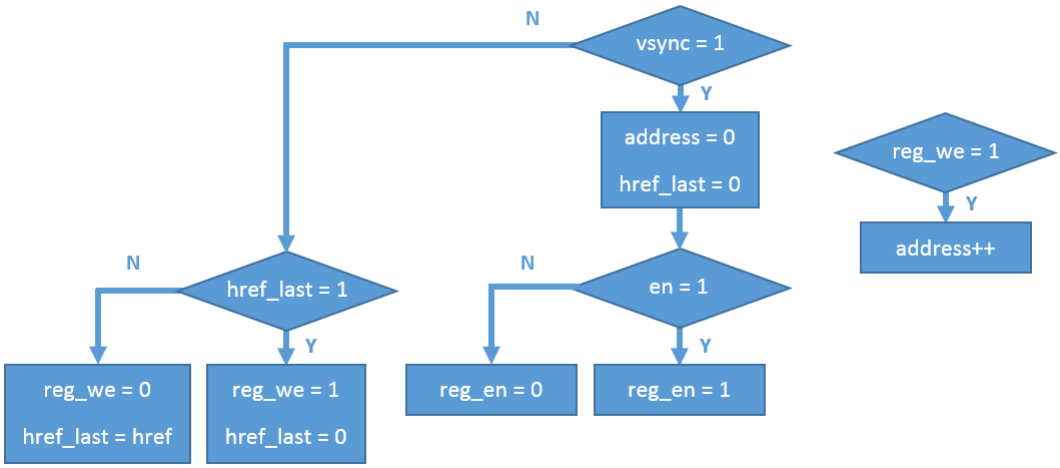


Figure 4.6 CFB Detecting active portion of frame, enabling capturing and filtering valid portion of data input

Finally, if the internal register for *enable capturing* (*reg_en*) is active, the module is in operational mode by assigning the internal registers to the output. A valid output is set by *reg_we* through *m_axis_tvalid*. Figure 4.7 shows the output assignment functionalities. Note that when capturing is disabled the outputs are set to zero. In addition, the output *clk* is the same as the input. Data input is registered one more time when *href* is active, and it is always connected to data output (*m_axis_tdata*). CFB works with the clock generated by the CM. Consequently, a FIFO (First-in First-out) with two clock inputs is needed to cross to the internal clock domain.

One side of the FIFO is connected to the CFB synchronized by *aclk*, and the other side is connected to DSM synchronized by an internal clock at higher frequency.

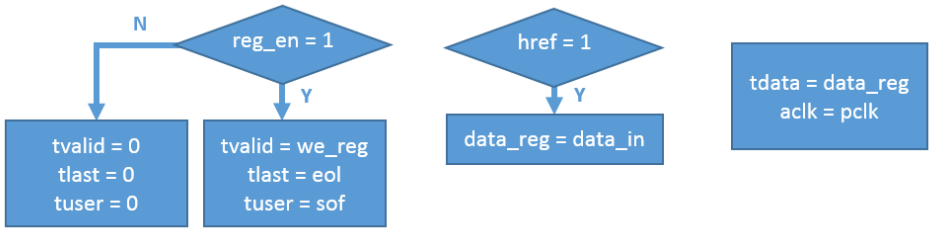


Figure 4.7 CFB output assignment

The CFB module has a latency of three clock cycles. The resources used by this module are summarized in Table 4.1 for a maximum resolution of 720p.

Resources	Utilization	Available	Utilization %
Slice LUTs (Luck-Up-Table)	130	17600	0.74
Slice Registers (Flip-Flops)	51	35200	0.14

Table 4.1 CFB resources utilization based on target platform Xilinx Zynq-7010 architecture

4.4.2 Down-Scale Module (DSM)

DSM downscales the image frame captured by the CFB, with the purpose of filtering small lighting changes in the background scene and camera noise, and decreasing the amount of memory needed to store the background. The algorithm is based on dividing the image into square blocks of a predetermined size, and then computing the mean of all the pixels in each block. The scaled image is composed by all the calculated mean values. As the image pixels come in streaming mode, starting from the top-left pixel, row by row, the approach followed by this module is as follows. At first, there is a register of size equal to the amount of blocks in a row, which holds the mean of every block. For every pixel, its horizontal position is calculated to

determine which block it belongs to, and then average its value with the average already stored in the corresponding register. For example, if the block size is 8x8 pixels, the three least significant bits of the horizontal position hold the pixel horizontal position inside the block. Then, the rest of the bits specifies the block itself. Also, using this example, when the three least significant bits of the horizontal and vertical positions are equal to block size minus one, it determines the end of every block, so a pixel of the scaled image has to be released as output.

DSM is connected to the clock domain crossing FIFO in turn connected to CFB. As the CM cannot be interrupted, the CFB cannot be interrupted either as a consequence. If any delay in further modules occurs, this FIFO is able to accumulate the values until the system is ready again. Figure 4.8 presents the module interface. It has three groups of interfaces: resolution configuration, slave *axi_stream* as input, and master *axi_stream* as output.

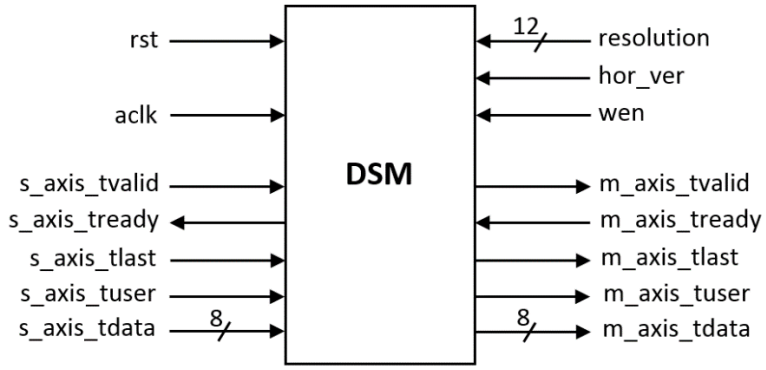


Figure 4.8 DSM block interface

The resolution configuration interface is connected to the CPU. The slave *axi_stream* interface is connected to the previous module, which is the FIFO connected to the CFB. The master *axi_stream* interface is connected to the next module BUSM. The purpose of each of the signal is as follows:

- s_axis_tvalid: (input) when active, the value on *s_axis_tdata* is valid to be sampled.
- s_axis_tready: (output) when active, tells the previous module that it is ready to receive data.
- s_axis_tlast: (input) when active, indicates the last pixel of a row.
- s_axis_tdata: (input) data value of the pixel (luminance).
- s_axis_tuser: (input) when active, indicates the first pixel of a frame.
- m_axis_tvalid: (output) when active, the value on *m_axis_tdata* is valid to be sampled.
- m_axis_tready: (input) when active, the next module is ready to receive data.
- m_axis_tlast: (output) when active, indicates the last pixel of a row.
- m_axis_tdata: (output) data value of the pixel.
- m_axis_tuser: (output) when active, indicates the first pixel of a frame.
- aclk: clock of the module.
- rst: (input) resets the module to its initial state. After reset, the module maintains the last configured resolution.
- resolution: (input) signal used to set the either horizontal or vertical resolution.
- hor_ver: (input) signal used to define whether the value in input resolution is either horizontal or vertical.
- wen: (input) *write enable* signal to write the selected resolution.

Like all the modules in this work, the input signals are registered in every clock cycle. There is a first functionality that computes the horizontal and vertical positions, and accumulates

the pixel value with the corresponding block average register. Figure 4.9 shows the block diagram of these functionalities. The signals with *"_registered"* at the end of the name are the input signal already registered in a previous cycle. This functionality uses the registered input signals *tvalid_registered*, *tuser_registered*, and *tlast_registered* to determine a valid input, start of frame and end of line. However, it delays the increment of the position by one cycle as well as temporal registers for *tvalid* (*reg_tvalid*) and *tlast* (*reg_tlast*), for the next pipelined functionality to work properly with the result of the pixel accumulation.

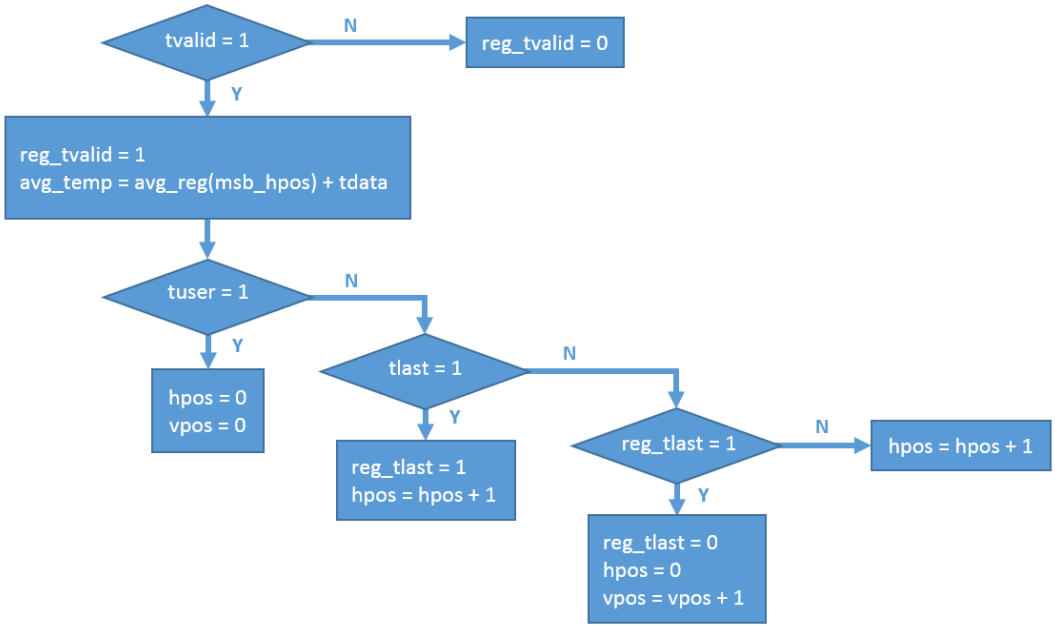


Figure 4.9 DSM pixel position and data input accumulation functionalities

The next pipelined functionality receives the result of the accumulation and either updates the block average register or finishes the mean computation and transmits the result. If the pixel position is inside the block, the design updates the block average register. On the other hand, if it is the last pixel of the block, the design divides the corresponding average register by the amount of pixels per block finishing the average computation. This division is substituted with a shift to the right. In this particular example, the block size is 8x8 pixels, the least

significant bits are 3, and as the division is by 64, the right shifting is by 6 bits. The amount of bits to be shifted is calculated by $\log_2(bsize * bsize)$. Notice that *sof* of the scaled image is not at position zero of the input image, because it is at the end of the first block.

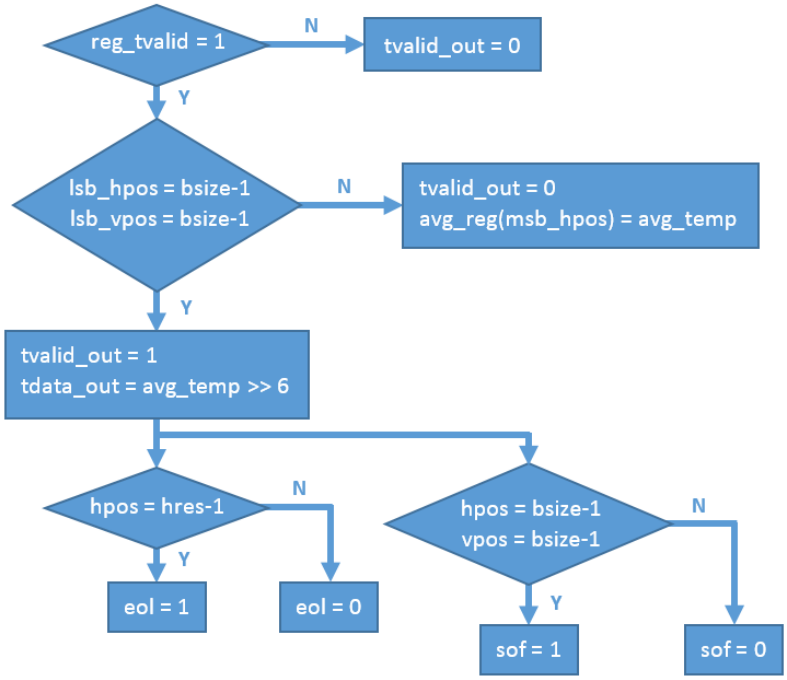


Figure 4.10 DSM eol, sof, average division and output assignment

The DSM module has a latency of three clock cycles. The resources utilized by this module are summarized in Table 4.2 for a maximum resolution of 720p.

Resources	Utilization	Available	Utilization %
Slice LUTs (Luck-Up-Table)	1089	17600	0
Slice Registers (Flip-Flops)	2300	35200	0

Table 4.2 DSM resources utilization based on target platform Xilinx Zynq-7010 architecture

4.4.3 Background Update and Synchronization Module (BUSM) plus Background Subtraction (BSM), and Thresholding Module (THM)

In this section, the Background Update and Synchronization Module (BUSM), the Background Subtraction Module (BSM), and the Thresholding Module (THM) are designed together as one single unit. BUSM uses two memory elements to store the background. It switches the memories in a way that one stores the current background and is read, and the other one is written with the new background. The new background is calculated using the value read from the current background memory, the pixel value from the current frame and equation (3.4) from chapter III. Background subtraction and thresholding are computed using the same two first operands and the equation (2.8) from chapter II. Figure 4.11 shows the interface of the combined design with the typical *axi_stream* slave input interface as well as the master output interface. Note that output *m_axis_tdata* is only a one-bit signal because the result of this module is a binary image. The background update rate (*beta*) and segmentation threshold (*seg_th*) are also inputs to the module as configuration interface.

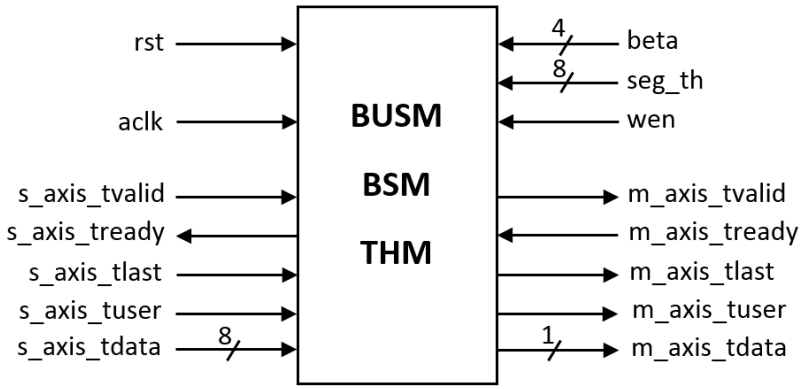


Figure 4.11 BUSM-BSM-THM block interface

In the first clock cycle, all the input signals are registered. In the second clock cycle, the input values registered in cycle one are then visible to the functionality described in Figure 4.12.

At this cycle, the registered values are registered again to pass them to the next functionality in a pipeline mode. In addition, this part of the module determines the valid start of frame by checking on *tvalid* and *tuser* signals, with the purpose of initializing the address to access the background memory, reading the new values of *threshold* and *beta*, and switching the background memories. If it is not a start of frame state, it only increments the background memory address.

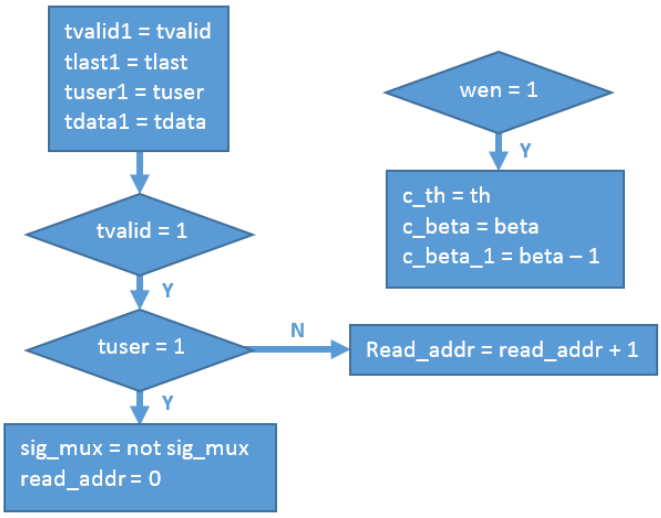


Figure 4.12 BUSM-BSM-THM sof initialization and valid input operation

There are two memory elements as storage component in this module. Part of the acceleration was to use the FPGA dedicated memory resources BRAM (Block Random Access Memory), which are very close related to the FPGA logic, providing clock cycle read and write operations for high speed functionalities. In Figure 4.13, the distribution of these memories is described. The calculated new background, denoted by the signal *reg_1_new_bg*, is the input to the memories, and the selected output is the current background (*curr_bg*). The signal *sig_mux* selects the memory to be read and the one to be written. The memory with the current background is read first, then, after computing the new background, it is written in the other memory.

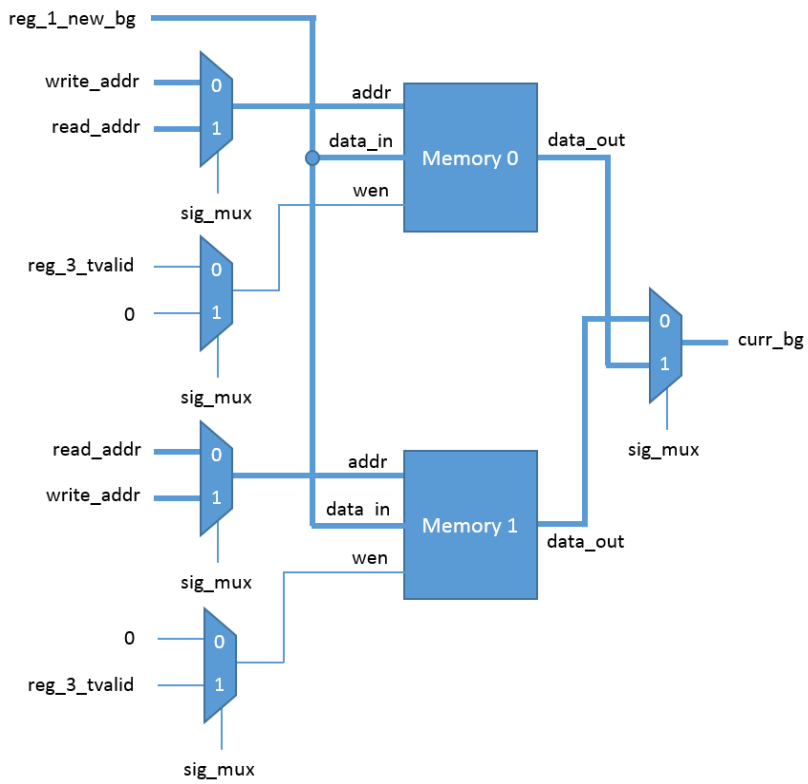


Figure 4.13 BUSM-BSM-THM Current background and new background memory distribution

The computation of background subtraction, thresholding and new background is shown in Figure 4.14. This block diagram represents a pipeline distribution of registers and operations implicated in this functionality. The signals r*eg_0* and r*eg_1* represent the added registers to compensate the memory latency of two cycles. The memories are configured to register the output values. This registration step adds one more cycle to their intrinsic latency of one clock cycle. The operations of multiplication and addition for new background are set in the same clock cycle, because it makes use of FPGA dedicated DSP (Digital Signal Processing) resources, which save on FPGA logic and accelerate the operation. The writing address is the same as the reading address with some delay to be synchronized with the new background result (*reg_1_new_bg*). The signal *reg_3_tvalid* goes through the same process because it represents the *write enable* signal (*wen*) of the memories.

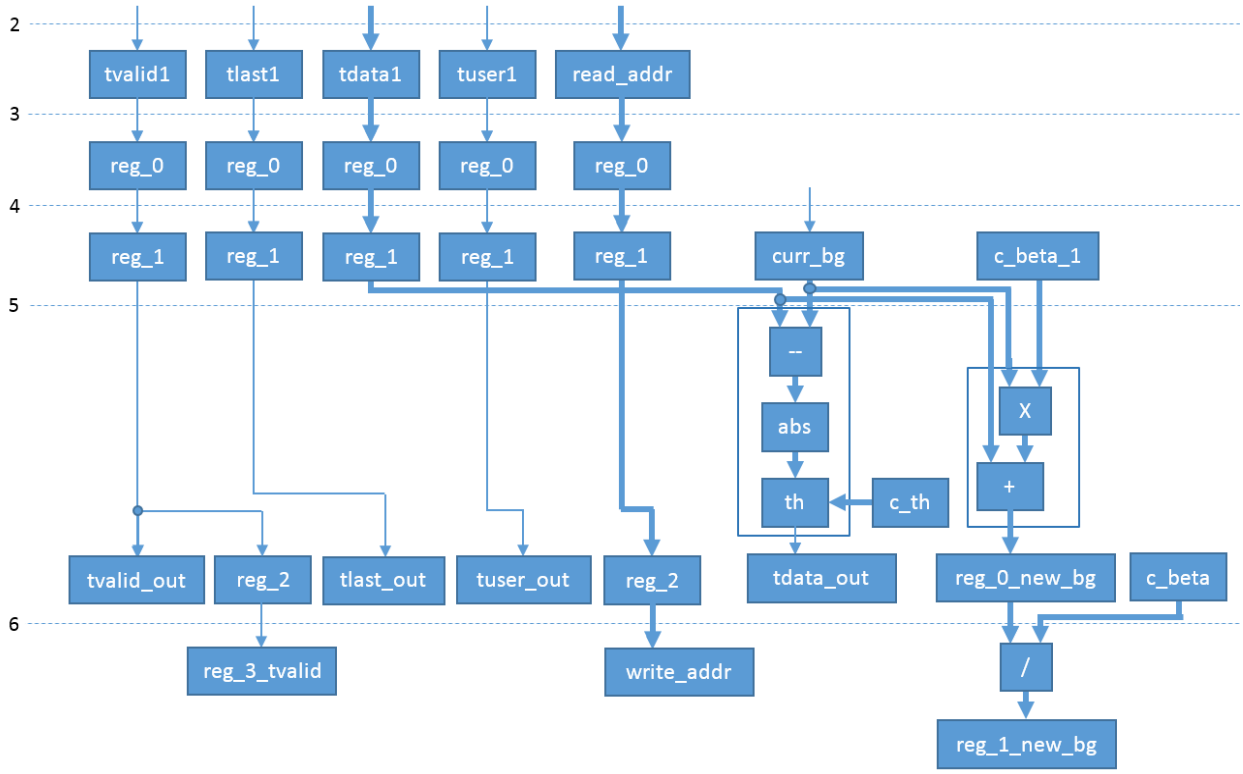


Figure 4.14 BUSM-BSM-THM Computation of background subtraction, thresholding and new background

BUSM-BSM-THM has a latency of 5 clock cycle to the output and 6 to write the new background to memory. After the latency it outputs one result per cycle time, which is one clock cycle. The resources used by this module are summarized in Table 4.3 for a maximum resolution of 720p.

Resources	Utilization	Available	Utilization %
Slice LUTs	190	17600	1.08
Slice Registers	104	35200	0.30
Memory (BRAM)	8	60	13.33
DSP	1	80	1.25

Table 4.3 BUSM-BSM-THM resources utilization based on target platform Xilinx Zynq-7010 architecture

4.4.4 Serial-In Parallel-Out Shift Register

At this point, the input image to the Shift Register module has the format of 1-bit per pixel, and the signal *tlast* is active at the end of every row. This module converts the previous format into an array of 32-bits words with only one *tlast* at the end of the whole frame. As the DMA system is able to transmit 32-bits word to DRAM, it is worthy to employ some resources to reduce the amount of DMA transactions. Reducing these transactions reduces the DMA interruptions along with the CPU interruptions as well, reflecting an improvement in performance. DMA will interrupt only once a full frame has been transmitted. Notice in Figure 4.15 that input *s_axis_tdata* is 1-bit wide, and *m_axis_tdata* is 32-bits wide.

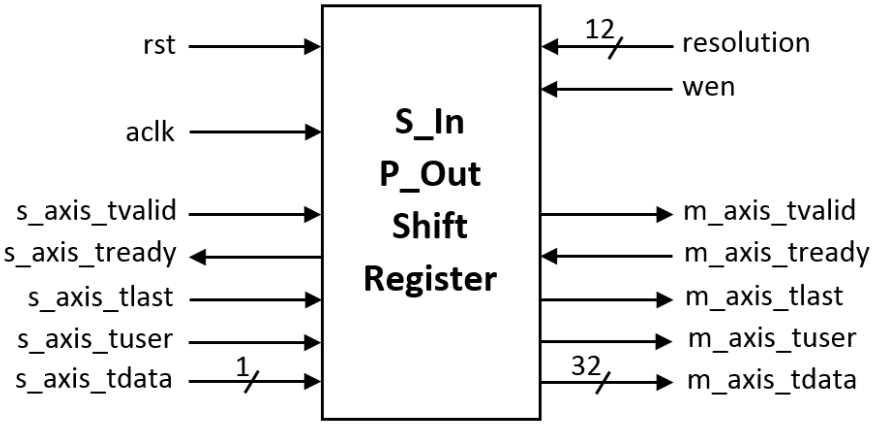


Figure 4.15 1-bit Serial-In to 32-bit Parallel-Out Shift Register block interface

There are two main functionalities in this module, one to count the amount of rows to determine the end of the frame, and another one to do the conversion to 32-bits, and transmit the result when 32 bits have arrived. In the first clock cycle, all the inputs are registered. In the second cycle, only the values of *tvalid*, *tlast*, and *tdata* are registered again to be passed to the next functionality in a pipeline mode. Figure 4.16 shows that the signal *last_counter* is incremented until it gets the value of *vres* (vertical resolution), then it is reset. The second

functionality checks for *tlast* to be active and *last_counter* to be zero, which means it is the last pixel of the frame, therefore *tvalid_out* and *tlast_out* have to be set to active. If it is not the end of the frame yet, the data out is shifted anyways and the new data is concatenated. Then, the condition of full register (*counter_32bit* = 31) is also checked to activate *tvalid_out* and transmit the register. If the register is not full yet, *counter_32bit* is incremented.

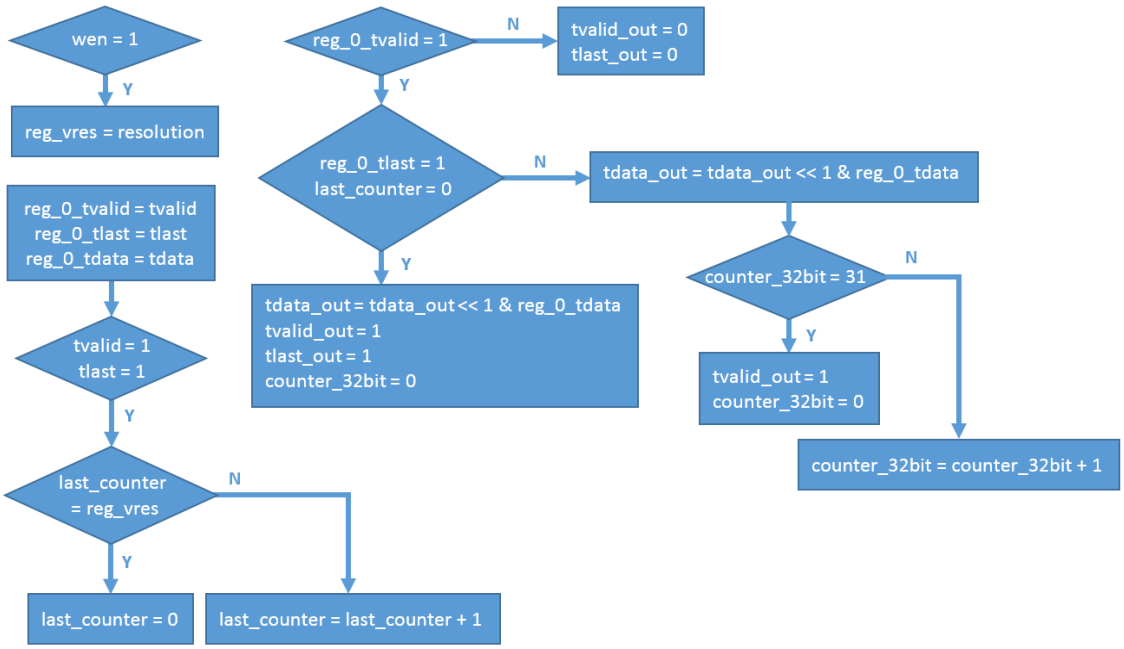


Figure 4.16 1-bit Serial-In to 32-bit Parallel-Out Shift Register functionalities block diagram

This module has a latency of three clock cycle to the output. The resources utilized by this module are summarized in Table 4.4 for a maximum resolution of 720p.

Resources	Utilization	Available	Utilization %
Slice LUTs	17	17600	0.1
Slice Registers	49	35200	0.14

Table 4.4 Serial-In Parallel-Out Shift Register resources utilization based on target platform Xilinx Zynq-7010 architecture

4.4.5 Serial Camera Control Bus (SCCB)

In order to configure the CM, the CPU has to send the configuration commands to the SCCB module, and then this one to the camera. The commands are sets of 24-bits words stored in DRAM, 16 bits for the register address and 8 bits for the value. The amount of registers to be written in one configuration could be about 600 and the SCCB communication is quite slow (400KHz clock maximum). That is why, it is very convenient to use DMA to send the commands, so the CPU can continue with other tasks. The SCCB block is connected to the CPU through another DMA channel configured as read only. Hence, the CPU executes one simple DMA transfer, with the initial DRAM address where the commands are stored, and the amount of commands to be transmitted. This approach adds one more DMA channel to the system along with one more FIFO and some other integration components. Table 4.5 below shows the resources used by this component without taking into account the storage component (FIFO) connected between SCCB and DMA.

Resources	Utilization	Available	Utilization %
Slice LUTs	59	17600	0.34
Slice Registers	93	35200	0.26

Table 4.5 SCCB resources utilization based on target platform Xilinx Zynq-7010 architecture

In order to change the resolution, it is required to first disable the frame capturing. Then, the system waits for the next DMA interruption, which means that the system is completely disabled. At this state, the system is ready to receive the new configuration. After reconfiguring the modules with the new resolution, the camera module can be reconfigured. Then, the system can be set back to enable again.

4.5 General Hardware Characteristics and Performance

It is important to mention that in addition to the FIFO included in between CFB and DSM, more FIFO modules were included as well to connect every module up to DMA in order to ensure proper connectivity. All these FIFOs are 32 words deep except for the one connected to DMA, which is 1024, to guarantee one row computation if the CPU is delayed for any reason. After every row, there is a blank spot that provides some extra time to advance with the computation. These FIFO components are original IP cores from Xilinx, FIFO Generator version 12.0 [35]. They are configured as *axi_stream* interface, and only the one connected to CFB has two independent clocks. The DMA components are also property of Xilinx, AXI Direct Memory Access version 7.1 [36], configured as *write channel only*.

In general, the hardware implementation, consists of two modules CFB, two DSM, two BUSM-BSM-THM, two Serial-In Parallel-Out Shift Registers, three DMA modules, two SCCB modules, one Zynq processing unit with all the necessary connectivity components, and the GPIO modules for enabling and configuration. When a pixel arrives to the CFB, the hardware implementation adds a latency of 22 clock cycles up to DMA. This means that the segmented binary frame result is ready in DRAM, 22 clock cycles plus DMA latency after the last pixel arrives to CFB. By adding this latency to the time that the CPU takes to process the two segmented frames, the total latency is obtained. The total latency is described in next sections. The hardware implementation is designed in pipeline mode, which makes it able to properly handle the camera clock streaming data at 96 MHz, by performing at a frequency slightly higher of 100 Mhz.

Table 4.6 presents the resources utilized by the entire system. There are still resources for either increasing the operation resolution or adding more functionalities and filters, which may improve the accuracy of the system.

Resources	Utilization	Available	Utilization %
Slice LUTs	7193	17600	40.87
Slice Registers	10245	35200	29.11
Memory (BRAM)	25	60	41.67
DSP	1	80	1.25
IO	48	102	47.06
Clocking	6	32	18.75
Processing Unit 7	1	1	100

Table 4.6 Entire Implementation resources utilization based on target platform Xilinx Zynq-7010 architecture

4.6 Software Implementation

This section describes the software algorithms designed to be implemented in the CPU core included in the target platform. The last hardware components in the processing pipeline are the DMA blocks to transmit the segmented binary image to DRAM, and interrupt the CPU for further processing. Once the CPU is interrupted by a channel, it reconfigures the channel to receive the next frame, and processes the bounding box and centroid of the corresponding binary image. The CPU proceeds to compute the measured disparity, then the estimated disparity by Kalman algorithm, and finally it calculates the depth and shows the results.

4.6.1 Initialization, Start and Operation Block

This is the first section of the software algorithm. In this part the components are initialized and configured, and the start command for operation mode is given. First, the platform components such as DMA, Interrupt controller, and GPIO (General-purpose input/output) are

initialized and configured. Then, the system components are also initialized such as CM parameters, CFB resolution, DSM resolution, BUSM-BSM-THM beta and threshold, and the Shift Register vertical resolution. Once the configuration is done, the start command for initiating capturing is released by enabling CFB.

Initially, the platform components have to be checked to retrieve their hardware configuration and ensure that they are present. If they are present, the retrieved hardware configuration is used to create an instance of both DMA controllers, which is the next step. The same procedure is used for the interrupt controller, and the GPIOs. There are several GPIO components, one for each of the system components that require configuration. Using these instances, the platform and system components are configured.

The next step is to connect the DMA interrupt handlers to the interrupt controller and enable them. By this step, the system gets ready to be interrupted by both DMA channels once they acquire one full frame each. The handler subroutines are functions corresponding to each DMA channel that will be called when the DMA interrupts. If the CPU is inside one of these functions, it means that a full frame is already in DRAM. Each handler is related to a channel; this way the channel can be identified. In the next section the DMA interrupt handler is fully described. Then, an initial simple DMA transfer is started. At this point, both DMA channels are waiting for data to transfer to DRAM. This simple transfer is started with two parameters, the DRAM initial location, where the DMA will start coping the data, and the length of the packet to receive.

The initial location is a 32-bits address in the dedicated DRAM space allocated for this purpose. Calculating the length needed is as follows. In case of 720p resolution (1280x720 pixels), the downscaled image is 160x90 pixels, which is equivalent to 14400 pixels. The result

of the segmentation is an image of the same amount of pixels but each pixel is only 1-bit wide. Then, after the Shift Register, the result packet is 14400 bits divided by 32 bits, which is equal to 450 words of 32 bits each. Initializing a DMA transfer requires this amount to be based on bytes, so the length is 1800 bytes.

After the CM is configured, the rest of the system components are initialized as well, such as CFB horizontal resolution, DSM horizontal and vertical resolution, BUSM-BSM-THM background update rate (beta) and segmentation threshold, and Shift Register vertical resolution.

The platform components configuration is done only once when the system starts up. However, the system components configuration can be done at any time when changes to system parameters are required. Beta and segmentation threshold can be changed without the need of resetting the system. However, for horizontal and vertical resolution, the system must be reset.

Once the platform and system components are configured and initialized, frame capturing can start. After configuring the CM, it starts sending frame information to CFB, but if this one is disabled, it does not receive any data. When enabling CFB, it starts receiving the data coming from CM, processing it, and transferring it for further processing by the next modules. At this point, the system is fully operating. Then, the software algorithm enters in the loop of computing depth. Computing depth block is described later in this chapter.

The CPU running at 650 MHz, takes 18 us to initiate and configure the platform components and the system components except CM. Then, to configure the CM, it takes 53.571 ms because the SCCB is a standard bus limited to 400 KHz communication frequency.

4.6.2 Segmented Frame Acquisition Block (DMA Interrupt Handler)

During the configuration steps, the DMA interrupt handler was connected to the interrupt controller in order to make the CPU to jump and execute this function once a segmented frame is

acquired. There are four main functionalities inside this handler: servicing the interrupt, checking for frame rate violation, setting up the DMA to receive the next frame, and computing the bounding box and centroid of the moving object, if there is one in the received frame. Figure 4.17 describes the main steps to accomplish the objectives of this function. Servicing the interrupt requires to read the status of the interrupt controller that contains the current asserted interrupt, and then use this information to acknowledge that the same interrupt is being processed.

The frame rate violation process is a step to verify that the performance of the software algorithm is meeting the required frame rate. The software algorithm has to process a full segmented frame before the next frame is completely acquired. The flag "*progress*" is set to *IN_PROGRESS* while a frame is being processed, and to *NO_PROGRESS* when finished. If a new frame is acquired and this flag is still at *IN_PROGRESS* state, an error condition is generated. Otherwise, the software algorithm is capable of meeting the frame rate specification.

This work uses two buffers that are switched per received frame. With this approach, while one frame is being processed, the next one is being acquired at the same time. To accomplish this idea, it is necessary to have a flag to track which buffer is being used for acquiring and for processing at every interrupt. After knowing which buffer is for acquiring and which one for processing, a next acquisition is started using the right buffer for this purpose at this time, and the other buffer is used for the next processing step, which is computing the object bounding box and centroid.

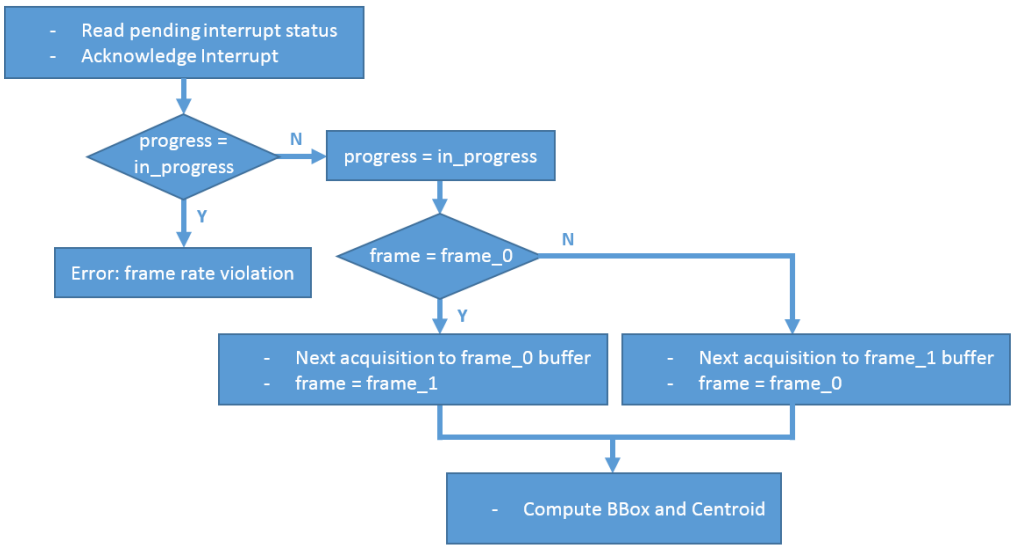


Figure 4.17 Frame acquisition block diagram (DMA Interrupt handler)

Computing the bounding box and the centroid of the segmented object is a method based on going through all the pixels that conform the image to find the top, bottom, left, and right positions of the boundaries of the segmented object. Once the entire image is processed, the center of that box (centroid) can be calculated. Figure 4.18 presents the algorithm used by this work to compute the bounding box and centroid. As the segmented image is stored in DRAM in a format where each bit represents a binary pixel, the approach is to process each bit of every 32-bits word in the buffer. If the bit is '1', it means that the pixel it represents is part of the moving object. As the processing goes from the top-left corner of the image to the bottom-right, the vertical position of the first pixel to be object is the top border of the bounding box. The bottom border is the vertical position of the last object pixel. The left and right borders are the most left and right horizontal positions respectively. If there is at least one object pixel, it means that the object was actually detected by that camera, so the algorithm defines that frame as "*VALID*". Only when both frames are valid, the depth can be computed. The bounding box is then up-scaled back to its original resolution. The centroid, X and Y coordinates are calculated by

computing the center of left and right borders, and top and bottom borders, respectively. At the end, the centroid progress flag is set to "*DONE*" to let the rest of the algorithm know that it can proceed with depth calculation.

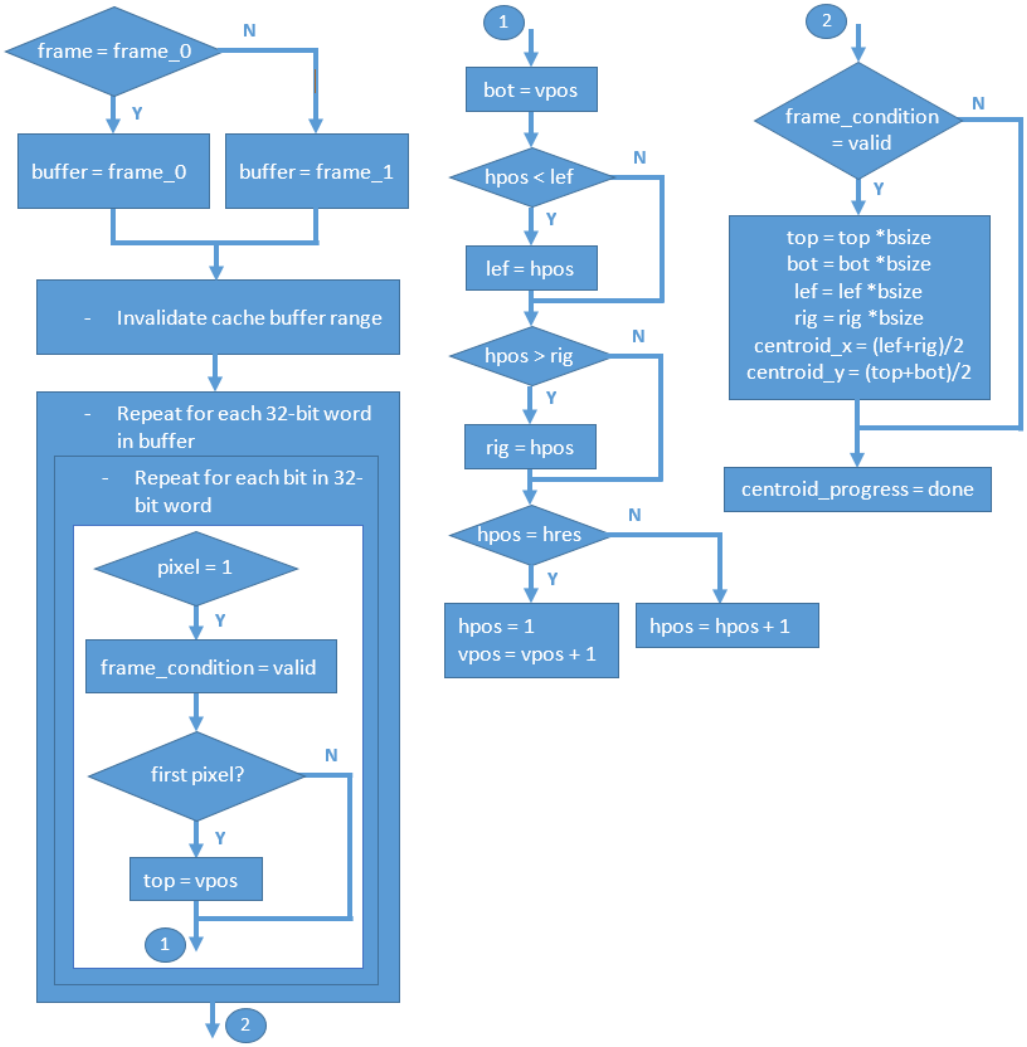


Figure 4.18 Bounding box and centroid computation

A performance measurement block was added right at the beginning of the interrupt handler. The system was able to measure its performance in terms of frame rate by using this block. The time it takes from one interrupt to the next one should be approximately 33.33 ms in order to tell that the objective of 30 fps is achieved. The performance measurement block

detected that the average interrupt interval was about 34.867 ms, equivalent to approximately 28.68 fps. In the next segment, the performance of the software implementation is described as the time it takes from DMA interrupt to depth computation.

4.6.3 Computing Depth Block

This block is executed in a loop where three functionalities are computed in sequential order. First, this block waits for both frames, left and right, to be pre-processed up to bounding box and centroid computation, which means that it waits for centroid progress flag to be "*DONE*" in both channels. Then, it processes the disparity, if the centroids are valid, applies Kalman to the measured disparity to calculated the estimated one, and finally converts estimated disparity to depth or distance from the stereo camera set origin to the object. Notice in Figure 4.19, where this process is described, how the measured disparity is checked to be positive for a valid measurement, then a flag for depth condition is set as either valid or invalid accordingly. At the end of this step, the centroid flag is set as "*NOT_DONE*" and the progress flag as "*NO_PROGRESS*" to wait again for the next frame to come.

The Kalman algorithm for one variable model is used by this work to smoothen the measured disparity. In order to simplify the Kalman disparity model, an offline transformation of the matrixes involved in the model and their operations was performed, and some of the parameters were experimentally adjusted. As the tracking of the disparity is a one-dimensional problem, the prediction model is taken from the example in chapter II. In this work, the software approach represents the matrixes as arrays starting from the top-left element row by row.

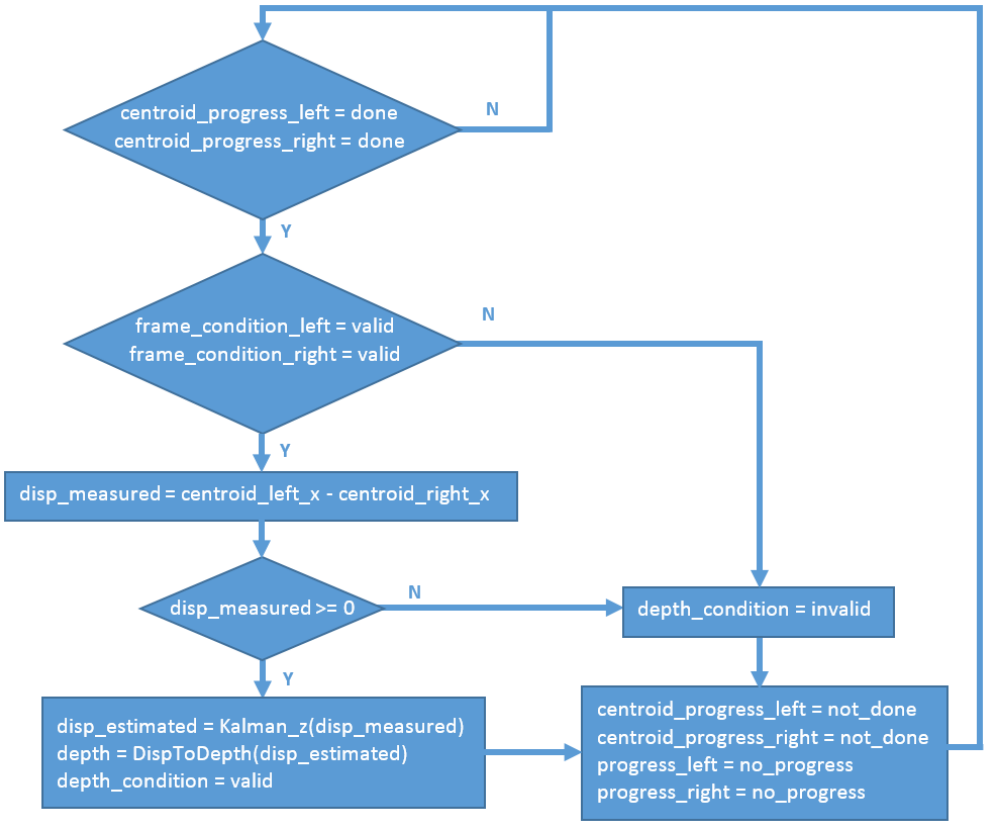


Figure 4.19 Depth computing block

The first step is to predict the next state of the object disparity by equation (2.17) in chapter II. After working out this matrix equation, the linear equations derived from it are as presented in equation (2.1), and (2.2), where $X[0]$ is the estimated disparity, $X[1]$ is the speed of change, $\Delta t = 1 \; and \; u = 0.005$.

$$X[0] = X[0] + X[1]\Delta t + B[0]u \qquad (4.1)$$

$$X[1] = X[1] + B[1]u \qquad (4.2)$$

The next step in the prediction state is to estimate the next covariance P using the equation (2.18) in chapter II. As P is a matrix of four elements, it generates four linear equations (4.3), (4.4), (4.5), and (4.6), where the noise magnitude that multiplies Q is 0.001.

$$P[0] = P[0] + P[2]\Delta t + Q[0] \quad (4.3)$$

$$P[1] = P[1] + P[3]\Delta t + Q[1] \quad (4.4)$$

$$P[2] = P[0]\Delta t + P[2]\Delta t^2 + P[2] + Q[2] \quad (4.5)$$

$$P[3] = P[1]\Delta t + P[3]\Delta t^2 + P[3] + Q[3] \quad (4.6)$$

After the prediction state, then comes the measurement updating state that starts with updating the Kalman Gain matrix K, which is only two elements generating just two linear equations shown by (3.1) and (3.2) bellow, where R^2 is 1.

$$K[0] = P[0]/(P[0] + R^2) \quad (4.7)$$

$$K[1] = P[2]/(P[0] + R^2) \quad (4.8)$$

The next step in the updating state is to update the state estimate X and the covariance estimate P, which is performed by using the equations (2.20) and (2.21), respectively, found in chapter II. As described before, the state estimate is a two element matrix that generates two linear equations shown below in equations (4.9) and (4.10), where $disparity_{measured}$ is the disparity previously calculated from the subtraction of the X coordinate of both centroids. The covariance estimation linear equations are presented in (4.11), (4.12), (4.13), and (4.14). The estimated disparity $X[0]$ from (4.9) is the value used to calculate the depth in the next procedure of converting disparity to depth.

$$\mathrm{X}[0] = \mathrm{X}[0] + \mathrm{K}[0](\mathrm{disparity}_{\mathrm{measured}} - \mathrm{X}[0]) \quad (4.9)$$

$$X[1] = X[1] + K[1](disparity_{measured} - X[0]) \quad (4.10)$$

$$P[0] = (1 - K[0])P[0] \quad (4.11)$$

$$P[1] = (1 - K[0])P[1] \quad (4.12)$$

$$P[2] = (1 - K[0])P[0] + P[2] \quad (4.13)$$

$$P[3] = (1 - K[0])P[1] + P[3] \quad (4.14)$$

Calculating the depth from the estimated disparity is the next and final step to be performed by the software system to achieve the objectives of this work. The model disparity-depth is created offline. Then, using the estimated disparity $X[0]$ as input in that model, the depth can be obtained.

The disparity-depth model is created by taking as many images as possible from both cameras of a known object at different depths. The object centroid is extracted from those images. Then, the disparity is calculated from the centroids. Disparity and depth build a ground truth curve used to determine the equation of their relationship. Figure 4.20 shows the curve disparity over depth for the stereo camera set built by this work at a resolution of 720p. The reference points were limited to the range from 25 cm to 375 cm. This lower limit is set because the fields of view of both cameras do not overlap each other at that distance; consequently, the center of the object is not projected on both cameras at the same time. At the upper boundary, the changes in disparity are very small, greatly affecting the accuracy of the result.

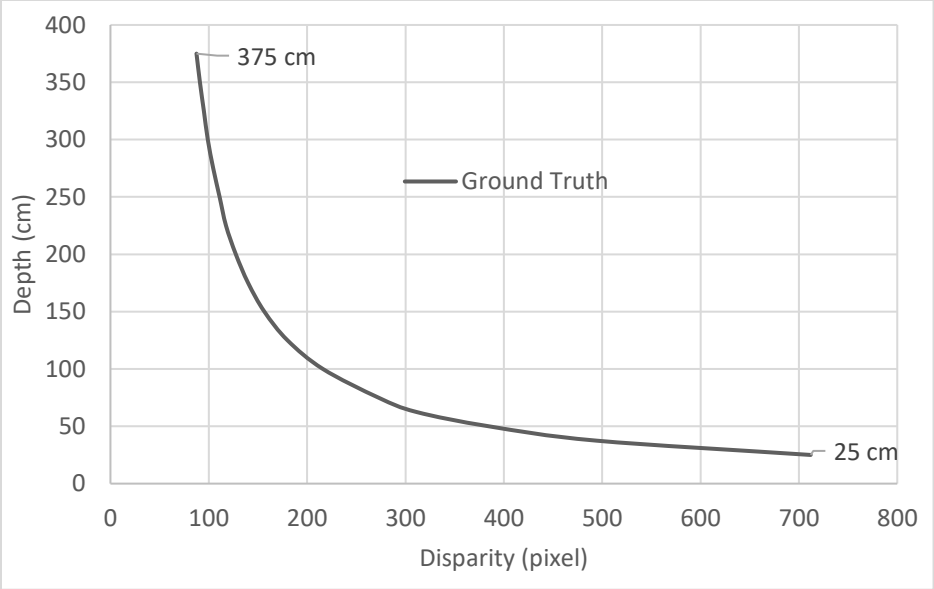


Figure 4.20 Disparity/Depth ground truth curve built from the extracted reference points

The created model that best fits the ground truth includes two different functions for two regions. The ground truth is divided into two regions: one for disparity less or equal to 272, and another one when greater than 272. Therefore, before evaluating the disparity, the right model is selected according to the disparity. For the section (≤ 272), the curve fits a 6^{th} degree polynomial model, and for the rest (> 272) fits a cubic model. In Table 4.7, the parameters of the models are shown. Figure 4.21 presents the model over the ground truth curve. Notice the greatest offset is at 272 when the switching of models occurs.

Region (disparity ≤ 272) $y = p_1x^6 + p_2x^5 + p_3x^4 + p_4x^3 + p_5x^2 + p_6x + p_7$	Region (disparity > 272) $y = p_1x^3 + p_2x^2 + p_3x + p_4$
$p_1 = 3.4716e - 11$	$p_1 = -9.2084e - 07$
$p_2 = -4.4035e - 08$	$p_2 = 0.0016047$
$p_3 = 2.3067e - 05$	$p_3 = -0.97846$
$p_4 = -0.0064193$	$p_4 = 240.5$
$p_5 = 1.0101$	-
$p_6 = -86.927$	-
$p_7 = 3406$	-

Table 4.7 Models that fit the ground truth curve in two regions

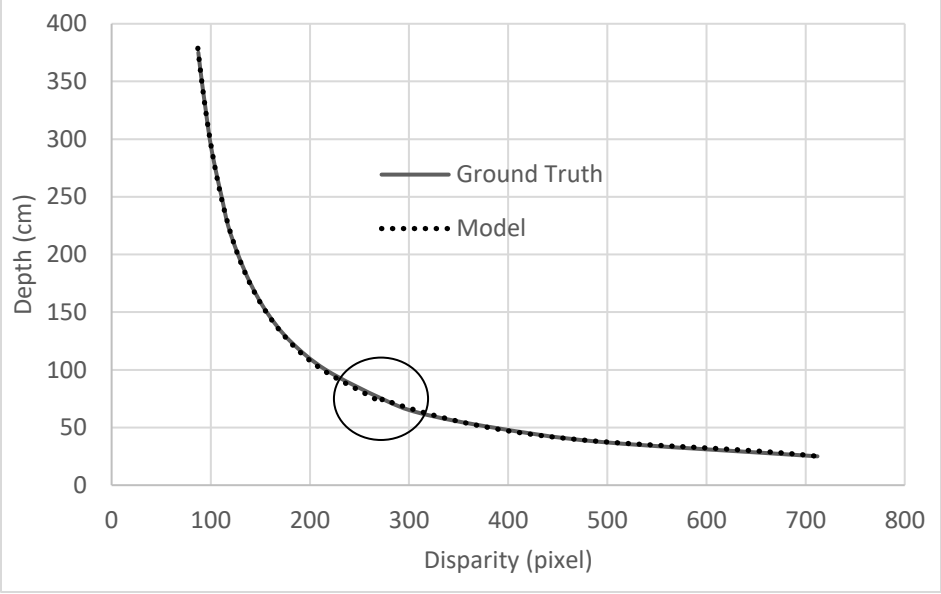


Figure 4.21 Disparity/Depth model over ground truth

As mentioned before, the depth measurement range is limited from 25 cm to 375 cm. The upper boundary is where the disparity is about 87 pixels and the accuracy is ±4 cm, meeting one the accuracy specifications of this work. The maximum percentage error of the measurement is 1.066 %. Figure 4.22 presents the error in depth according to the disparity. This error is determined by the stereo camera set design, the characteristics of the cameras, and their resolution. Changing the distance in between the cameras, moves the range closer or further, but

keeps the same range. On the other hand, increasing the resolution decreases the error, increasing the depth range.

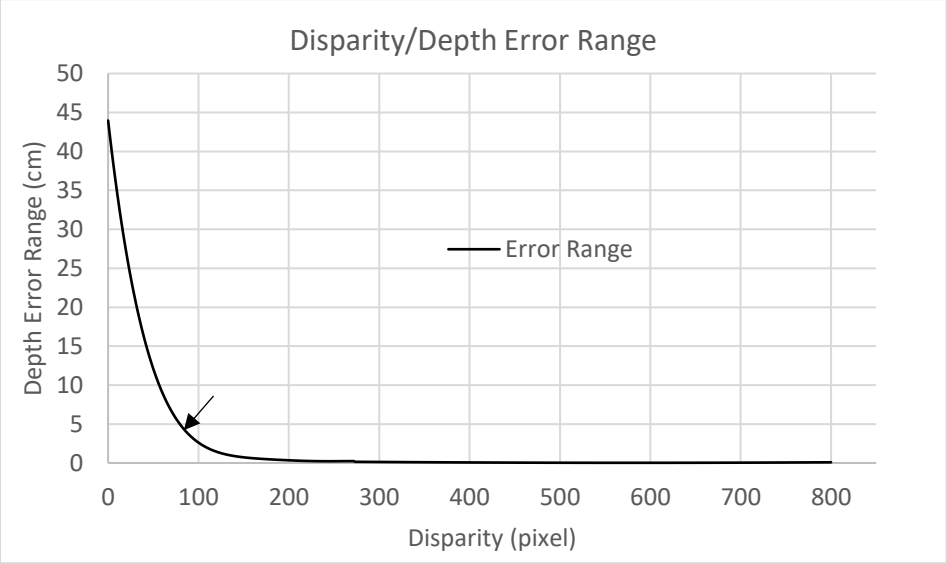


Figure 4.22 Relation Disparity/Depth Error Range

In terms of performance, this work analyses the time the software implementation takes to completely process the two segmented frames and finally compute the depth. The CPU takes from 445 us to 625 us depending on the size of the moving object. Since the hardware latency is insignificant compared to the software latency, the total latency of the system is in the range of the software latency. Another analysis from these values is that the software design is able to perform its functionalities in the period of time in between two frames, which is 33.33 ms, according to the frame rate specification for this work. There is a substantial difference in the performance of the software design and the specification. Therefore, the software implementation is able to easily handle higher resolutions. The software design resulted in the amount of data presented in Table 4.8.

Text	Data	BSS	Dec	Hex
48840	3176	35812	87828	15714

Table 4.8 Software Implementation building result

4.7 Showing Results

As a method of showing the computed results, the UART to USB communication channel is used to send the exact calculated depth from the processing unit to a PC. This channel is also allocated for debugging and reconfiguring the system. For the particular function of showing the depth result, an interface such the one presented in Figure 4.23 is used. This result is compared with the actual distance at which the object is, and then an evaluation is done. If the object is either not present in one of the cameras, or simply there is not object, the depth is declared Invalid as shown in Figure 4.24. This information can be sent to another module for further processing or actuation.

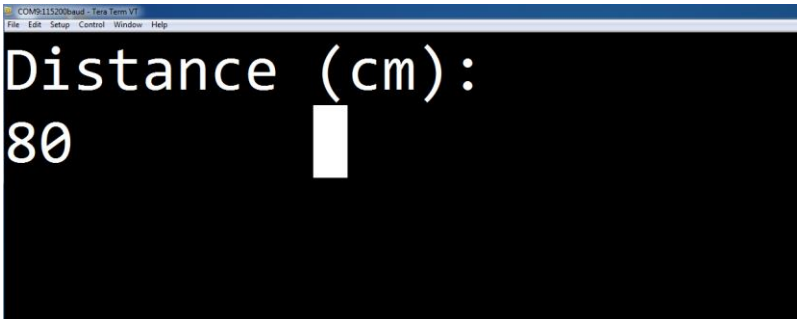


Figure 4.23 Depth result when moving object is 80 cm far away from the camera set

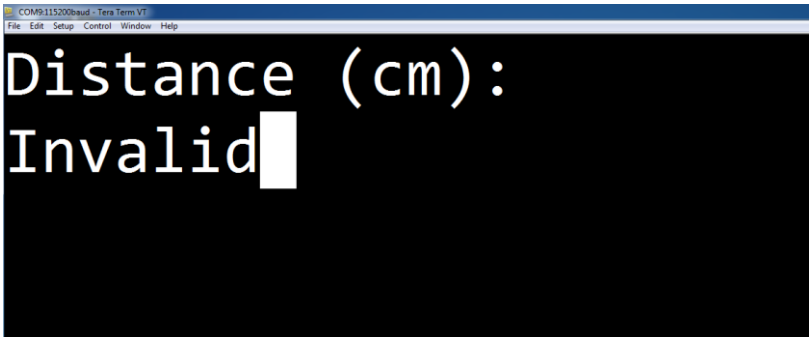


Figure 4.24 Invalid depth result

For visual performance measurement and better debugging, a display connection is also added to the design. Figure 4.25 shows on the display a two dimensional space with the two cameras as reference points. In addition, the horizontal lines are one meter apart. The object is represented by the block close to the one-meter horizontal line, for a depth of 80 cm.

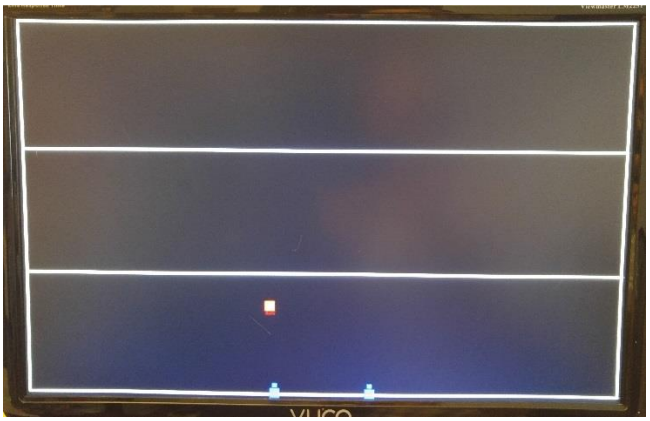


Figure 4.25 Display for visual performance measurement and debugging

4.8 Power Consumption Control

In terms of power consumption control, this work proposes the approach of adding clock power down capability when the system is not capturing. This approach utilizes the IP Core property of Xilinx, Clocking Wizard (version 5.1) [37]. This core generates several clocks from one single clock source, with a feature of reset and power down in case of energy saving modes are required. Figure 4.26, below, presents the diagram of the clock power down system. The CPU generates the signal *clk_power_down* that controls the clocks generated by Clocking Wizard Module. If *clk_power_down* is active, the four generated clocks are shut down, putting the dependent blocks in standby mode. Using this strategy reduces the power consumption by 23.08 % during standby mode. Table 4.9 compares the energy consumption characteristics of the implementation with first just the platform by itself without any configuration, then the implementation in standby mode, and finally the implementation in operating mode.

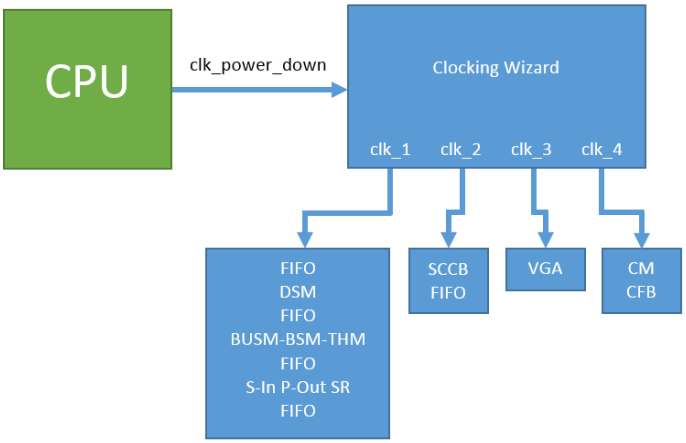


Figure 4.26 Clock Power Down Control Diagram

System	Current (mA)	Power (W)
Not configured Platform	250	1.250
Configured Platform, Standby (including cameras)	350	1.750
Configured Platform Operating (including cameras)	455	2.275

Table 4.9 Implementation DC characteristics

4.9 Summary

In this chapter, the set of specifications of the system was presented, which is to implement a system able to detect, locate and track a moving object targeting its depth information, at the resolution 720p at 30 frames per second. In order to achieve this goal, the hardware platform was also selected and described, which is the Omnivision OV5642 camera module and the Digilent ZYBO development board. In addition to this platform, a PC and a display were also added to the prototype setup for debugging and testing purposes. The specific hardware and software implementations were explained in detail including block diagrams, interfaces and resources utilization.

The hardware implementation represented less than 50% of the available resources of the SoC platform. The software processing unit is able to process the two segmented frames and

finally compute the depth in the range of 445 us to 625 us meeting the specifications of 30 fps. The measurement range of the system was limited to 375 cm maximum to meet the requirement of ±4 cm of maximum error.

Chapter V Summary and Future Work

Stereo vision is a difficult and computationally demanding problem with many useful applications. The inspiration driving this work was to research, design, implement, and verify the application of this technology in a real world problem, and to expose its variants, modifications, advantages, and issues according to the specifications and constraints of the specific problem. The real world problem targeted by this work is the computation of the depth information of a moving object in a scene. This is the reason why its main objective is to build a system able to perform the application of detecting and tracking a moving object in a three dimensional scene, at 720p of resolution, 30 frames per second and with depth error no greater than ±4 cm, for a maximum percentage error in the measurement of 1.066 %.

This work successfully accomplished its specific task of researching a wide range of algorithms related to possible solutions for this problem. Several approaches were introduced, such as stereo disparity computation based on correlation algorithms and feature algorithms. Correlation algorithms were also extended to global and local approaches because of their complexity. The significance of calibration and its variants were also presented. In addition to this, methods to detect moving objects were also described, such as background subtraction. In this case, this work specifically examined the issues and the solutions for different background dynamic. Finally, Kalman algorithm was proposed as very effective method to track objects and smoothen their trajectories.

From the previously researched algorithms, some were deeply analyzed and selected to be part of the solution proposed by this work. Two independent vision channels from a stereo camera set were used to capture the stereoscopic view of the scene. Background subtraction was the main algorithm for detecting the moving object. The background subtraction was supported

by the recursive filter of first order as background update method, and mean filter as pre-processing approach. The mean filter was combined with the downscaling of the frame, in order to reduce the background storage resources. The result of the background subtraction was segmented by thresholding to create a binary image. This image was later sent to DRAM for further processing by the CPU. The algorithms were partitioned in hardware and software portions according to their computationally or algorithmically intensive behaviour. DMA technology was used to transmit the segmented images to DRAM without interrupting the CPU. Once the segmented stereo frames were in DRAM, the CPU was given the tasks of computing the centroid of the moving area, the measured disparity, estimating the disparity by Kalman algorithm, and finally calculating the depth from the estimated disparity.

This work introduced the platform to be used for implementation of the design and verification of the specifications and constraints. The target platform as hardware/software computational power was the Digilent Zybo development board. The cameras used were OmniVision OV5642. A PC was also used to visualize the exact value of the depth, as well as for debugging and performance verification. In addition to this, a display was connected as a performance measurement.

The implementation successfully achieved the objectives of resolution 720p, and maximum permissible depth error of ±4 cm. To accomplish the goal of the maximum depth error, the depth measurement range was limited to a maximum distance of 375 cm, because further than that the error would be greater than ±4 cm. The depth limit could be improved by increasing the resolution of the cameras. The operation frame rate was about 28.68 fps, because that is the actual frame rate coming from the camera module. The system is capable of handling the exact 30 fps required to meet the objectives, but the cameras were about 1.32 fps slower.

Using higher performance and quality cameras guarantees an improvement in the operation frame rate.

The hardware portion has a latency of 22 clock cycles, and the cycle time is one clock cycle. The entire system has a latency that mostly depends on the software portion, which is in the range of 445 us to 625 us, depending on the size of the moving object. This is the time the CPU takes to process the two segmented frames up to have the depth information. The system configuration time depends directly on the camera module configuration, which takes about 53.571 ms, because the cameras use SCCB communication channel at 400 MHz maximum frequency. The hardware resources utilized by the design were below 50 % of the target platform, allowing room for improvement and future work.

In terms of power consumption, the implementation requires 2.275 W in operating mode, and 1.750 W in standby mode. This power consumption takes into account the entire system, including the cameras' consumption. The clock power down control is very useful to reduce power if standby mode is required. The reduction is about 23.08 %.

5.1 Future Work

There are several paths for future research that could be explored in relation to this work. As this work has focused on detecting and tracking just one moving object in the scene, a future improvement is to implement an algorithm to detect and track several objects individually. As noted in chapter 3, the centroid computation of one single object could be easily executed by hardware, but the algorithm for multiple objects is too complex, therefore it is better executed by software. This is precisely what was proposed in chapter 3 of this work: to execute the bounding box and centroid of the moving object by software in order to build the base architecture for future multiple objects detection.

Camera calibration and undistortion are improvements to be added in future work. Individual camera undistortion guarantees a uniform distribution of the disparity along the entire image, which results in a uniform depth wherever the object is located in the scene. Stereo calibration simplifies the model that relates depth and disparity, and also improves the calculation of parameters of the model by completely automating this process.

This work is focused on performing at least at 30 fps, which limits the resolution to 720p according to the camera performance. Therefore, by using higher performance and quality cameras, this resolution will be increased, amplifying the measurement range and its accuracy. Along with the resolution, a future upgrade to this work is the use of more storage for the background, avoiding losing quality when downscaling the image.

The clock power down control manages only the system components related to hardware processing. In future work, the goal will be controlling more components, such as platform components and the CPU itself, by stopping or decreasing the operation frequency of more clock domains. This will definitely improve the energy consumption of the system in standby mode.

REFERENCES

[1] M. Domański et al., "Fast depth estimation on mobile platforms and FPGA devices," 2015 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON), Lisbon, 2015, pp. 1-4.

[2] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, D. Nister, “Realtime global stereo matching using hierarchical belief propagation”, British Machine Vision Conference, pp. 989–998, Edinburgh, 2006.

[3] J. Choi and R. A. Rutenbar, "Video-Rate Stereo Matching Using Markov Random Field TRW-S Inference on a Hybrid CPU+FPGA Computing Platform," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 26, no. 2, pp. 385-398, Feb. 2016.

[4] K. Nakazato, Y. Touma, H. Hagiwara, K. Asami and M. Komori, "FPGA-based stereo vision system using census transform for autonomous mobile robot," Informatics, Electronics & Vision (ICIEV), 2015 International Conference on, Fukuoka, 2015, pp. 1-4.

[5] A. Qamar, F. B. Muslim and L. Lavagno, "Analysis and Implementation of the Semi-Global Matching 3D Vision Algorithm Using Code Transformations and High-Level Synthesis," 2015 IEEE 81st Vehicular Technology Conference (VTC Spring), Glasgow, 2015, pp. 1-5.

[6] Hirschmüller, H. Semi-Global Matching Motivation, Developments and Applications. Proceedings of the Invited Paper at the 54th Photogrammetric Week, Stuttgart, Germany, 5–11 September 2011; pp. 173–184.

[7] C. Ttofis; C. Kyrkou; T. Theocharides, "A Low-Cost Real-Time Embedded Stereo Vision System for Accurate Disparity Estimation Based on Guided Image Filtering," in IEEE Transactions on Computers, vol.PP, no.99, 2015, pp.1-1

[8] R. Neves and A. C. Matos, "Raspberry PI based stereo vision for small size ASVs," 2013 OCEANS - San Diego, San Diego, CA, 2013, pp. 1-6.

[9] Donguk Seo, Hansung Park, Kanghyun Jo, Kangik Eom, Sungmin Yang and Taeho Kim, "Omnidirectional stereo vision based vehicle detection and distance measurement for driver assistance system," Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE, Vienna, 2013, pp. 5507-5511.

[10] C. Sánchez-Ferreira, J. Y. Mori, C. H. Llanos and E. Fortaleza, "Development of a stereo vision measurement architecture for an underwater robot," Circuits and Systems (LASCAS), 2013 IEEE Fourth Latin American Symposium on, Cusco, 2013, pp. 1-4.

[11] Jones Y. Mori, Janier Arias-Garcia, Camilo Sánchez-Ferreira, Daniel M. Muñoz, Carlos H. Llanos, and J. M. S. T. Motta, “An FPGA-Based Omnidirectional Vision Sensor for Motion Detection on Mobile Robots,” International Journal of Reconfigurable Computing, vol. 2012, Article ID 148190, 16 pages, 2012.

[12] V. D. Nguyen, T. T. Nguyen, D. D. Nguyen and J. W. Jeon, "Toward Real-Time Vehicle Detection Using Stereo Vision and an Evolutionary Algorithm," Vehicular Technology Conference (VTC Spring), 2012 IEEE 75th, Yokohama, 2012, pp. 1-5.

[13] S. Jadhav, R. Narvekar, A. Mandawale and S. Elgandelwar, "FPGA Based Object Tracking System," Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on, Gwalior, 2015, pp. 826-829.

[14] P. Gujrathi, R. A. Priya and P. Malathi, "Detecting Moving Object Using Background Subtraction Algorithm in FPGA," Advances in Computing and Communications (ICACC), 2014 Fourth International Conference on, Cochin, 2014, pp. 117-120.

[15] C. Sánchez-Ferreira, J. Y. Mori and C. H. Llanos, "Background subtraction algorithm for moving object detection in FPGA," Programmable Logic (SPL), 2012 VIII Southern Conference on, Bento Goncalves, 2012, pp. 1-6.

[16] W. Liu, H. Chen and L. Ma, "Moving object detection and tracking based on ZYNQ FPGA and ARM SOC," IET International Radar Conference 2015, Hangzhou, 2015, pp. 1-4.

[17] T. Kryjak, M. Komorkiewicz and M. Gorgon, "Hardware implementation of the PBAS foreground detection method in FPGA," Mixed Design of Integrated Circuits and Systems (MIXDES), 2013 Proceedings of the 20th International Conference, Gdynia, 2013, pp. 479-484.

[18] M. Shakeri and H. Zhang, "Detection of small moving objects using a moving camera," 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, 2014, pp. 2777-2782.

[19] Min-jeong Kang, Choong-Ho Lee, Jin-Hwan Kim and Uk-Youl Huh, "Distance and velocity measurement of moving object using stereo vision system," Control, Automation and Systems, 2008. ICCAS 2008. International Conference on, Seoul, 2008, pp. 2181-2184.

[20] G.S.Joshi, A.P. Tadamalle, "A Stereo Correspondence Cost Function for FPGAs", IJETAE, Volume 3, Issue 2, Feb 2013, pp. 455-459

[21] S. Mattoccia, "Stereo Vision Algorithms for FPGAs," 2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops, Portland, OR, 2013, pp. 636-641.

[22] D. Honegger, H. Oleynikova and M. Pollefeys, "Real-time and low latency embedded computer vision hardware based on a combination of FPGA and mobile CPU," 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, 2014, pp. 4930-4935.

[23] M. Hofmann, P. Tiefenbacher and G. Rigoll, "Background segmentation with feedback: The Pixel-Based Adaptive Segmenter," 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, Providence, RI, 2012, pp. 38-43.

[24] B. Zheng, X. Xu, Y. Dai and Y. Lu, "Object Tracking Algorithm Based on Combination of Dynamic Template Matching and Kalman Filter," Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2012 4th International Conference on, Nanchang, Jiangxi, 2012, pp. 136-139.

[25] S. Huang and J. Hong, "Moving object tracking system based on camshift and Kalman filter," Consumer Electronics, Communications and Networks (CECNet), 2011 International Conference on, XianNing, 2011, pp. 1423-1426.

[26] R. P. Tripathi, S. Ghosh and J. O. Chandle, "Tracking of object using optimal adaptive Kalman filter," *2016 IEEE International Conference on Engineering and Technology (ICETECH)*, Coimbatore, 2016, pp. 1128-1131.

[27] Yohan Noh, Hongbin Liu, Sina Sareh, Damith Suresh Chathuranga, Helge Würdemann, Kawal Rhode, Kaspar Althoefer, "Image-Based Optical Miniaturized Three-Axis Force Sensor for Cardiac Catheterization", *Sensors Journal IEEE*, vol. 16, pp. 7924-7932, 2016, ISSN 1530-437X

[28] OmniVision Technologies, ver. 2.2, June 2007, OmniVision Serial Camera Control Bus (SCCB) Functional Specification, Application Note

[29] OmniVision Technologies, ver. 2.05, April 2011, OV5642 Color 5 megapxel image sensor with OmniBSI and embedded TrueFocus technology, Datasheet Product Specification, Datasheet CSP3

[30] OmniVision Technologies, Rev. 1.10, Mar. 2010, OV5642 Camera Module Software Application Notes, Application Notes

[31] Digilent, Rev. B, Feb 2014, Zybo Refence Manual, Reference Manual, DOC#: 502-279

[32] Xilinx, ver. 1.8, May. 2015, Zynq-7000 All Programmable SoC Overview, Product Specification, DS190

[33] Xilinx, ver. 5.3, Oct. 2013, LogicCORE IP Processing System 7, Product Guide for Vivado Design Suite, PG082

[34] Xilinx, ver. 13.1, March 2011, AXI Reference Guide, Reference Guide, UG761

[35] Xilinx, ver. 12.0, June 2015, FIFO Generator, LogiCORE IP Product Guide, PG057

[36] Xilinx, ver. 7.1, Oct. 2016, AXI DMA, LogiCORE IP Product Guide, PG021

[37] Xilinx, ver. 5.1, April 2015, Clocking Wizard, LogiCORE IP Product Guide, PG065

Glossary of Acronyms and Abbreviations

2D: Two dimensional.

3D: Three dimensional.

ADSW: Adaptive Support Weight.

ARM: Acorn RISC Machine

BRAM: Block Random Access Memory.

BSM: Background Subtraction Module.

BUSM: Background Update and Synchronization Module.

CC: Cross-Correlation.

CCMA: Compute Centroid of Moving Area.

CFB: Capture Frame Block.

CM: Camera Module.

CMOS: Complementary Metal–Oxide–Semiconductor.

CPU: Central Processing Unit.

DDR3: Double data rate type three SDRAM (DDR3 SDRAM) is a type of synchronous dynamic random-access memory (SDRAM).

DMA: Direct Memory Access.

DMAB: Detect Moving Area Block.

DRAM: Dynamic Ramdom-Access Memory

DSM: Down-Scale Module.

DSP: Digital Signal Processing.

FIFO: First-in First-out.

FPGA: Field Programmable Gate Array.

fps: Frames per Second.

GIF: Guided Image Filter.

GPIO: General-purpose input/output.

HMM: Hidden Markov Model

HOG: Histograms of Oriented Gradients

IP Core: Intellectual property core.

JTAG: Joint Test Action Group

KHz: Kilohertz

LUT: Luck-Up-Table.

mA: Milliampere

Matlab: Multi-paradigm numerical computing environment and programming language.

MB: Mega Bytes.

Mbps: Mega Bit per Second.

MI: Mutual Information.

NP: Nondeterministic Polynomial

OpenCV: Library of programming functions mainly aimed at real-time computer vision.

PBAS: Pixel-Based adaptive Segmenter.

PMOD: Peripheral Module.

QVGA: Quarter of area of VGA. Standard video resolution 320×240 pixels.

SAD: Sum of Absolute Differences.

SCCB: Serial Camera Control Bus.

SGM: Semi-Global Matching.

SoC: System-on-Chip.

SSD: Sum of Square Differences.

THM: Thresholding Module.

TRW-S: Sequential tree-reweighted message passing.

UART: Universal Asynchronous Receiver/Transmitter.

VGA: Video Graphics Array. Standard video resolution 640×480 pixels.

W: Watt.

YUV: Colorspace. The Y component determines the brightness of the color (referred to as luminance or luma), while the U and V components determine the color itself (the chroma).