People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdès

Institute of Electrical and Electronic Engineering

Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of
the Requirements for the Degree of

'Master'

In Electronics

Option: Computer Engineering

Title:

Hardware/Software Codesign of Real Time Obstacle Detection System using Stereovision

Presented by:

BENDAHMANE Meryem

Supervisor:

Pr. KHOUAS Abdelhakim

Registration Number:…..……/2022

Abstract

Obstacle detection and localization are crucial abilities for a mobile robot to navigate in an environment. Therefore, a highly developed environment perception system is necessary. Systems based on stereo vision are strongly recommended due to their high resolution, relatively wide range of measurement, and low cost. Such systems generate a disparity map from two stereo images (left and right), the generation process and disparity map quality depend on the stereo matching technique. In this project, we built a software system for a real-time obstacle detection using stereo vision. For the stereo vision module, we used the RANK non-parametric transformation and the Sum of Absolute Differences (SAD) correlation matric to estimate the disparity between the two images, the obstacle detection phase was carried forward using the U-V disparity approach. This approach makes use of the disparity map to build the V-disparity and U-disparity maps, from which obstacles can be easily localized after extracting vertical and horizontal lines from the maps respectively. The software design processes 240×320 pixels images at a rate of 1/3 fps, with a single thread Central Processing Unit (CPU) rate of 667 MHz and a Double Data Rate (DDR) memory running at a frequency of 533 MHz, this result reveals the inefficiency of our system for real-time applications. To overcome this limitation, we involved the hardware/software codesign methodology, where the system is split into hardware and software partitions. The choice of the best partitioning depends on the characteristics required to be present at the final system implementation. For our project, the speed was the main factor to meet the real-time application requirements. In order to get a better partitioning, we used profiling tools to determine the most time-consuming functions in our software system. The profiling shows that the stereo vision module consumes 90% of the total execution time, as a result, we replaced the stereo vision module in our application by a hardware stereo vision module. The software/hardware codesign solution built with the same software characteristics and a hardware stereo vision module running at 7.8 MHz led to a huge improvement in the system's speed, for 240×320 pixels images the processing rate reaches 78 fps achieving a 234 times acceleration from the original software design.

Dedication

To my dearest parents

For their endless love, support, and encouragement

To my grandmother and my aunts

Malika, Fazia, Nassira, and Fatma

To my dear cousins

Imene, Lina, and Anais

To my second family

Khawla BOUSSAAD and Farah TAHRI

To special people I had the pleasure of meeting during my years in college

Sara ALILAT, Meriem CHAIB, hadjar HAMDI, and Mohamed Amine TAIB

Thanks for being a part of my journey and making it so special and beautiful.

Acknowledgement

First and foremost, I would like to praise and thank God, the Almighty, who has granted countless blessing, knowledge, and opportunity to the writer, so that I have been finally able to accomplish the thesis.

I would like to express my deepest appreciation to my primary supervisor Pr. Abdelhakim KHOUAS for his support, guidance, and for allowing me to duly embrace this project under his auspices despite his busy agenda. I am thankful for his confidence and the freedom he gave me throughout the course of this work. In addition, I would like to acknowledge Mr. Taki Eddine SAIDI who has been an ideal mentor. No acknowledgment would be complete without expressing my gratitude to my colleague Karim Yahia for the help and support he has provided.

I am grateful to the members of the jury for taking the time to read and analyze this project report.

My warmest thanks go to the teachers of IGEE for the devotion they have shown and the knowledge they have passed to me.

Table of content

List of Figures

List of Tables

List of Abbreviations

2-D	Two Dimensions
3-D	Three Dimensions
ACK	Acknowledgement
AMBA	Advanced Microcontroller Bus Architecture
AP SoC	All Programmable System On Chip
APU	Application processor unit
AXI	Advanced eXtensible Interface
AXIS	Advanced eXtensible Interface Stream
BMP	Bad Matched Pixels
BOM	Bills Of Material
BSP	Board Support Package
CDT	C/C++ Development Toolkit
CLB	Configurable Logic Blocks
COE	Coefficient
CPU	Central Processing Unit
DDR	Double Data Rate
DM	Disparity Map
DMA	Direct Memory Access
DSP	Digital Signal Processor
FF	Flip-Flop
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FPU	floating Point Unit
GDB	Gnu DeBugger
GIC	General Interrupt Controller
GUI	Graphical User Interface
HD	Hamming Distance
HDL	Hardware Description Language
HDMI	High-Definition Multimedia Interface
HLS	High-Level Synthesis
HP	High-Performance
I2S	Inter Integrated Circuit Sound
IDE	Integrated Development Environment
IIC	Inter-Integrated Circuit
ILA	Integrated Logic Analyzer
IOP	Input/ Output peripherals
IP	Intellectual property
JTAG	Joint Test Action Group
LPDDR	Low Power Double Data Rate
LRF	Laser Range Finder
LUT	Look Up Table
Matlab	Matrix Laboratory
MM2S	Memory Mapped to stream
MMU	Memory Management Unit
OCM	On Chip Memory
OLED	Organic Light Emitting Diodes
Opencv	Open-source Computer Vision
PCB	Printed Circuit Board

PL	Programmable Logic
PS	Processing System
RAM	Random Access Memory
RGB	Red Green Blue
ROM	Read Only Memory
RTS	Real Time System
S2MM	Stream to Memory Mapped
SAD	Sum of Absolute Differences
SCL	Serial Clock
SD card	Secure Digital card
SDA	Serial Data
SDK	Software Development Kit
SPDIF	Sony/Philips Digital Interface
SRL16	16-bit Shift Registers
SRL32	32-bit Shift Registers
SSD	Sum of Squared Differences
SWDT	System Watchdog Timer
TCF	Target Communication Framework
TCL	Tool Command Language
TTC	Triple Timer Controller
UDM	U Disparity Map
VDM	V Disparity Map
VDMA	Video Direct Memory Access
VTC	Video Timing Controller
WTA	Winner Take All
ZEDBoard	Zynq Evaluation and Development board

General Introduction

Out of the five senses, vision is undoubtedly the one that humans have come to depend upon above all others, and indeed the one that provides most of the data received. The main feature of the human visual system is the ease and immediate with which interpretation of a scene is carried out, within a fraction of a second. In the present day and age, scientists are trying to get machines to do much of his work. For simple mechanistic application, this is not particularly difficult, but for more complex tasks such as real-time applications and autonomous systems, the machine must be given a sense of vision.

Computer vision is the automated extraction of information from images. Information can mean anything from three-dimensional (3D) models, camera position, object detection and recognition to grouping and searching image content. Sometimes, computer vision aims to mimic the human visual system through stereo vision using one or more cameras set up, where the 3D information is extracted from two or more images taken from different viewpoints. This problem of extracting useful information from images or videos finds application in a variety of fields such as autonomous navigation, obstacle detection and avoidance, remote sensing, etc.

A stereo vision system estimates disparity or the difference in the position of the pixel of a corresponding location in the camera view by finding similarities in the left and right image, which is known as the stereo correspondence. There have been various costs governing the extent of the similarity. Some of them are Sum of Absolute Differences (SAD), Sum of Squared Differences (SSD), which are window-based local approaches where the cost value of a particular window in the left image is compared to the right image window by spanning it along a horizontal axis for a disparity range. Based on the metric measures' disparity selection is mostly performed in the Winner-Takes-All (WTA) strategy, i.e., the disparity with the lowest measurement is chosen. By performing the same process for all pixels, a Disparity Map (DM) is generated, from which the depth can be easily calculated.

A key part of the construction of a world model is the incorporation of obstacles, to help machines to interact with their environment efficiently, in the stereo vision case, this can be achieved by building the U-V Disparity Maps (UDM and VDM) upon a given DM. the UDM can be considered as a top view of the perceived scene, whereas the VDM is a lateral view of the scene such that the road plane is represented by an oblique line, the obstacles as vertical alignments in VDM, and horizontal alignments in UDM. The VDM is obtained by computing a histogram of each row of the DM, while the UDM is generated by computing a histogram of each column of the DM.

An obstacle detection system based on stereo vision requires a great processing performance in order to accept more data, process all that data, and make an appropriate response, and what is mainly challenging in such systems is that all the processing must happen in real-time. To attain the needed performance, an engineer would take different paths to design the system such as software design, hardware design, or software/hardware codesign. High processing performance in software design can be achieved by a high clock speed of the processor or the usage of a multicore processor. However, this would mean an increase in power consumption, heat and cost.

On the other hand, hardware design using FPGA or GPU is more powerful due to its capability to operate in a parallel way, but comes with Bills of Material (BOM) cost, design complexity and potentially increasing the product development time. A better design solution is proposed with the software/hardware codesign, where performance hungry tasks are implemented in hardware while keeping small tasks for software. With the increasing improvements in system on chip (SOC) technology, the codesign development is much easier. Xilinx SoCs are processor-centric platforms that offer software, hardware and I/O programmability in a single chip. Xilinx introduced Zynq products that integrate ARM microprocessors in FPGAs to address the hardware complexity and BOM cost concerns, thus, there is no need to devote engineering resources to optimize a new printed circuit board (PCB) and its chip-to-chip interactions.

The main goal of our work is to see the impact of the design methodology on the performance of a system. In the first stage, we implemented a software design of a stereo-vision module, we evaluated the module based on the window size and disparity range. In addition to an obstacle detection module using the U-V disparity approach. In the second stage, we divided the system into hardware and software partitions, the stereo vision module was the most time-consuming part, therefore we integrated a hardware stereo vision module, preserving the same software obstacle detection module.

The report is organized as follows. In the first chapter, we tackle the theoretical background of stereo vision, including the geometric basis needed to understand the system modeling, and the different techniques for depth estimation, then we discuss the U-V disparity approach for obstacle detection. In the second chapter, we present a global design of the stereo vision-based obstacle detection system with all algorithms implemented. The third chapter is about the software implementation of the system as well as the experimental results and system evaluation. In the fourth chapter, we addressed the implementation of the obstacle detection system using the hardware/software codesign methodology, in addition to the processing performance.

Chapter I: Obstacle detection using stereo-vision

I.1 Introduction

Autonomous navigation, obstacle avoidance, and off-world exploration, are all applications that have been suggested for intelligent robots; in which an excessive degree of autonomy is required. For the robots to function autonomously in unstructured or unfamiliar environments, a highly developed environment perception system is necessary. The environment perception system would help to deal with the obstacle detection problem; since the robot must be capable to detect the sudden appearance of obstacles in its path. To perform such task, the mobile robot system must be able to self-build a real three-dimensional (3D) of the environment. Stereo vision is one of the perception techniques that simulate the human vision system using two cameras to capture images from different viewpoints. Recent researches show that the stereo-based approach has distinct advantages in that it can sense the environment with a high resolution, precision, and a relatively wide range of measurement and low cost. In this chapter, we will present the theoretical background of stereo vision, including a geometric basis needed to understand the system modeling, different algorithm for depth extraction, then we will discuss the obstacle detection from depth information.

I.2 Stereo vision geometry

Stereo-vision is the process of extracting depth information from images recorded from multiple viewpoints. There are different methods to acquire these images:

- Single moving camera: A single camera is moved in space, recording images, thereby providing images from different viewpoints. This implies a more or less static scene, as moving objects would lead to errors in the depth map.

- Multiple camera setup: fixed camera array with the same focal length is used to acquire images from different viewpoints, the exact relative positions of all cameras are known. In this method, all images should be captured at the same time to ensure that no errors result from moving objects or movement of the camera array [1].

To understand the basic operation of stereo vision, one must have a basic understanding of cameras and how they perceive the world. There exist two camera models, pinhole model and camera model with lens.

I.2.1 Pinhole camera model

The pinhole camera is the simplest and the ideal model of camera function. This kind of camera can be imagined as a box with a pinhole, through which light enters and forms a two-dimensional image on the opposite site [2]. A point P = (X, Y, Z) in the three-dimensional (3-D) XYZ-space is projected to an image-point p = (x, y) in the two-dimensional (2-D) xy-space, as illustrated in Figure I.1. The projection coordinates are given by:

$$y = f\frac{X}{Z} \tag{I.1}$$

$$y = f\frac{X}{Z} \tag{I.2}$$

Where *f* is the focal length that represent the distance between pinhole and image plane.

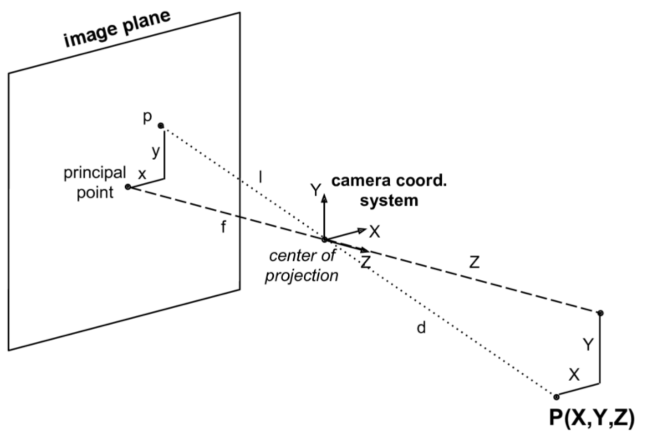


Figure I.1: Pinhole camera model *[3]*.

Equations (I.1) and (I.2) describe the mapping in Euclidean space into which a camera plane is embedded. Unfortunately, these two equations are both nonlinear with respect to Z coordinate because of the factor 1/Z. To represent both equations in a linear way, we transform the point (x, y) in the Euclidean plane to a point (x, y, 1) in the projective plane; these coordinates are called homogeneous coordinate [4].

The two equations are combined to generate a simple projection from world to image coordinates called perspective projection which can be written as:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \frac{1}{Z}\begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \frac{1}{Z}\underbrace{\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\tilde{P}}\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \qquad (I.3)$$

$\widetilde{\mathrm{P}}$ is the projection matrix, it is the result of two relationships as illustrated in Figure I.2 :

1. Relationship between the world and the camera coordinate system, it is expressed by the extrinsic parameters, that depend on the position of the camera in world coordinate system, it includes rotation matrix R which relates the rotation of the world coordinate system with the camera coordinate system, and a translation vector T containing the translations along the X, Y, and Z axes from the origin of the world coordinate system to the origin of the camera system.

2. Relationship between the camera coordinate system and the image coordinates. This relationship is expressed by the intrinsic parameters K; it describes the camera geometry such as the optical center and focal length of the camera [5, 2].

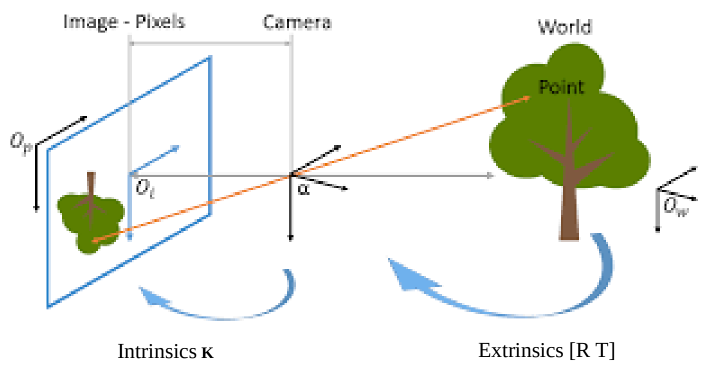


Figure I.2: Representation of the extrinsic and intrinsic parameters *[6]*.

Thus $\tilde{P}$ can be expressed as:

$$\tilde{P} = K[R \quad T] \qquad (I.4)$$

In pinhole model, we assumed that the optical center is at (0,0), the camera is centered at the origin of the 3-D coordinate, i.e., no rotation or translation. Equation (I.4) can be simplified as follow:

$$\tilde{P} = K[I \quad 0] = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad (I.5)$$

In general, we may want to have multiple cameras with different poses at the same time, where satisfying those assumptions is impossible. Thus, a more realistic projection representation considering the optical center at (u_0, v_0), camera is at (t_x, t_y, t_z) with rotation matrix R. Then the projection matrix can be written as:

$$\tilde{P} = K[R \quad T] = \begin{pmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \qquad (I.6)$$

where r_{ij} represents rotation parameters.

I.2.2 Camera model with lens

Unfortunately, the camera's pinhole through which the light enters is almost infinitely small in the previous model, the smaller the hole, the less light enters the camera and the longer exposure time is needed. In a dynamic environment, this is not possible because the objects in the scene are moving. Thus, in reality, lenses are used to project more light onto the pinhole [4].

Figure I.3 shows how the light rays reflected by an object are projected through a lens. *P* is a point in the real world, *d* is the distance between *P* and the lens, *f* is the focal distance of the lens, f_k is the effective focal length (distance between the lens and the projected object *P'*).

The relation between these parameters can be written in a mathematical form as follow:

$$\frac{1}{d} + \frac{1}{f_k} = \frac{1}{f} \qquad (I.7)$$

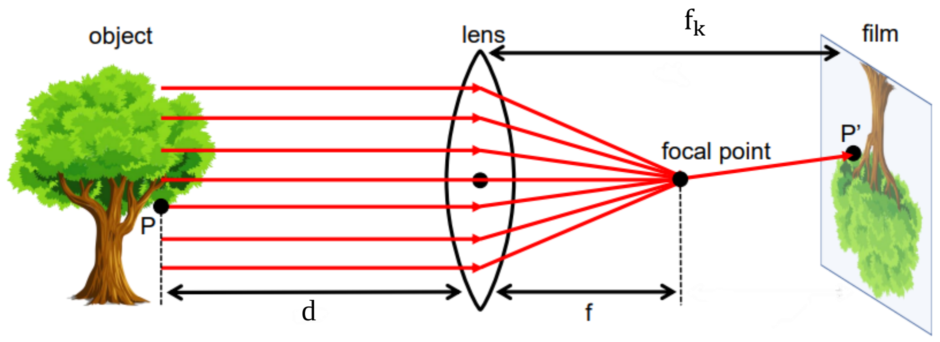


Figure I.3: light-rays projection through a lens *[7]*.

A negative effect of using lenses is distortion, which can be described as outwards (barrel distortion) or inwards (pincushion distortion) deviation from the ideal projection considered in pinhole camera model in reference to the center of the lens. Lens distortions are either due to the shape of the lens itself, or to an inaccurate alignment due to improper lens and camera assembly, this has no influence on the quality of the image, but it has a significant influence on the image geometry as shown Figure I.4.

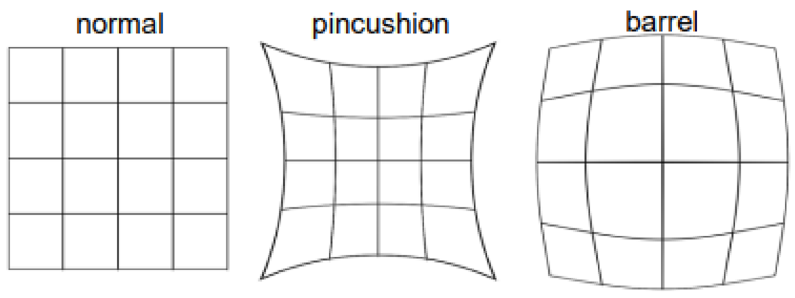


Figure I.4: Lens distortion effect on image geometry *[7]*.

I.2.3 Camera calibration

Camera calibration is an offline process of quantifying the internal camera geometric, optical characteristics (intrinsic parameters) and the 3D position and orientation of the camera frame relative to a certain world coordinate system (extrinsic parameters) [8]. These parameters can be used for various computer vision application including removal of lens distortion from an image. The Camera Calibration Toolbox for MATLAB can be used for both single and stereo camera calibrations. Another alternative to this toolbox is to use OpenCV library which is no more than the C implementation of the MATLAB camera calibration toolbox.

I.2.4 Epipolar geometry

The stereovision system utilizes a non-canonical stereo camera configuration, sometimes referred to as convergent system, where the optical axis of the two cameras is not parallel with one another. Such configuration is modeled by epipolar geometry.

A general epipolar geometry with two pinhole cameras arranged arbitrarily is shown in Figure I.5, where:

- I_l and I_r are the image planes of left and right camera respectively.
- C_l and C_r are projection centers for left and right cameras respectively.
- Baseline: the straight line that connects both cameras optical centers.
- Epipolar plane π: the plane containing the baseline and the corresponding points P_l and P_r of the scene point P.
- Epipoles: the points at which the line through the centers of projection intersects the image planes. Let e_l and e_r be the left and right epipoles respectively.
- Epipolar line: is the intersection of an epipolar plane with one image plane. All epipolar lines intersect at the epipole. An epipolar plane intersects the left and right image planes in epipolar lines.

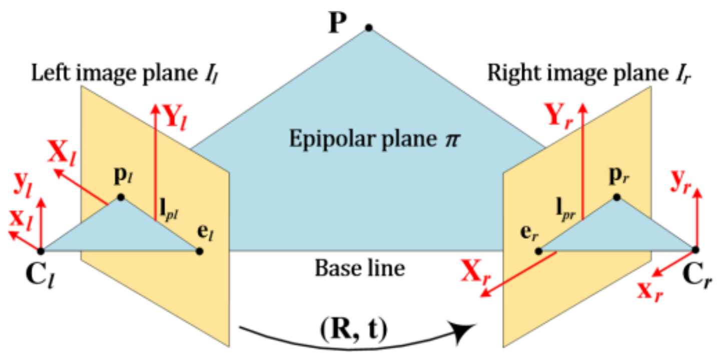


Figure I.5: Epipolar geometry *[9]*.

The mathematical interpretation of the epipolar geometry is that, for a corresponding point pair (pl, pr) in homogeneous coordinates, we have [10]:

$$P_l^{\mathsf{T}} F\, P_r = 0 \qquad (I.8)$$

where F is called fundamental matrix and $F\,P_r$ describes a line (an epipolar line) on which the corresponding point P_l must set.

I.2.5 Image un-distortion and stereo rectification

The image un-distortion is to compensate the effects of lens distortions, it consists of a transformation from physically distorted image to an ideal image under the pinhole camera model; according to the parameters calculated in the camera calibration step. For every pixel in undistorted image, we have to compute its distorted location [11].

The image rectification is the process of remapping images using the fundamental matrix such that the epipolar lines are horizontal. Thus, the resulting images are row aligned and the points in the left picture of one row to be in the same row of the right picture as shown in Figure I.6.

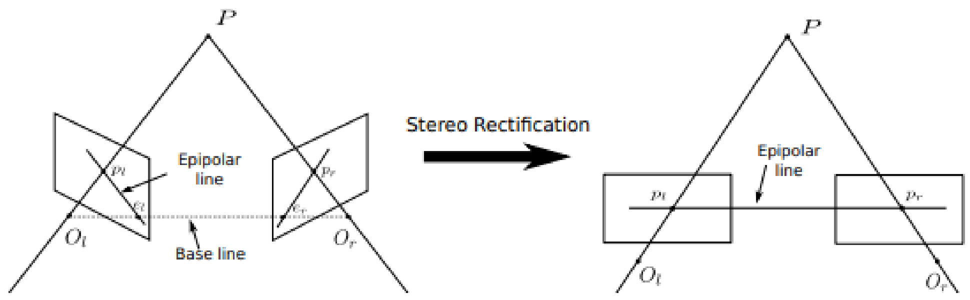


Figure I.6: Stereo rectification effect *[11]*.

I.3 Depth estimation

In ideal situation, where the two stereo images are perfectly rectified and un-distorted as illustrated in Figure I.*7*, depth *Z* is easily estimated.

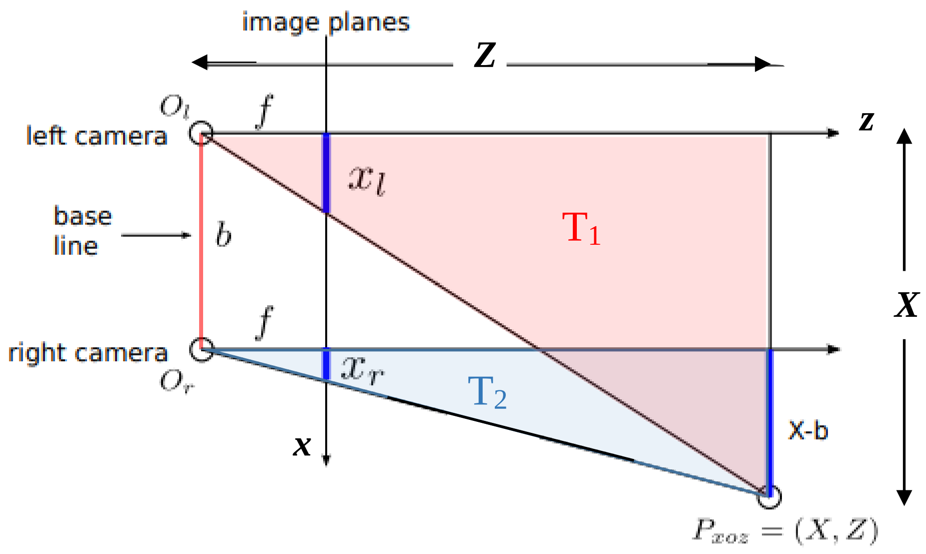


Figure I.7: 3D triangulation between two parallel, identical cameras *[11]*.

In Figure I.*7* the world point P= (*X*, *Z*) is projected into the left image as $P_l = (x_l, y_l)$ and into the right image as $P_r = (x_r, y_r)$, *f* is the focal length, *b* is the base line. From triangles T_1 and T_2, we can derive equations (I.9) and (I.10) respectively.

$$\frac{Z}{f} = \frac{X}{x_l} \tag{I.9}$$

$$\frac{Z}{f} = \frac{X-b}{x_r} \tag{I.10}$$

From the above equations, we can write:

$$\frac{X}{x_l} = \frac{X-b}{x_r} \tag{I.11}$$

After distributing terms and dividing by X we get:

$$b\,\frac{x_l}{X} = x_l - x_r \qquad (I.12)$$

Replacing ($\frac{x_l}{X}$) by ($\frac{f}{Z}$) in equation (I.12) we get:

$$Z = f\,\frac{b}{x_l - x_r} \qquad (I.13)$$

The quantity $(x_l - x_r)$ is referred to as disparity "d", so we can rewrite equation (I.13) as follow:

$$Z = f\,\frac{b}{d} \qquad (I.14)$$

From equation (I.14) we can deduce that:

- Far away objects (large distance from the observer) will move very little between the left and right image thus a small disparity.
- Very close by objects (small distance from the observer) will move quite a bit more means greater disparity.

I.4 Stereo correspondence

The previous section showed how to determine depth from disparity, which rely on solving what's known as stereo correspondence or stereo matching problem, where the aim is to identify corresponding points or objects in left and right image. This require a measure of similarity in order to find the point correspondences between the left and right images, for each pixel in the left image, there are a lot of possible candidates in the right image to be evaluated in order to determine the best corresponding pixel [12].

The search of corresponding points becomes even easier using rectified images. Then, corresponding points would have to be on the same horizontal line in both images. Solving the correspondence problem for all pixels, a disparity map (DM) can be built by gathering all the disparity measures in an image. In Figure I.8, two example pairs of corresponding points are indicated, with their position on the disparity map, which is a simple depth map, brighter areas are closer to the foreground and darker areas closer to the background [1].

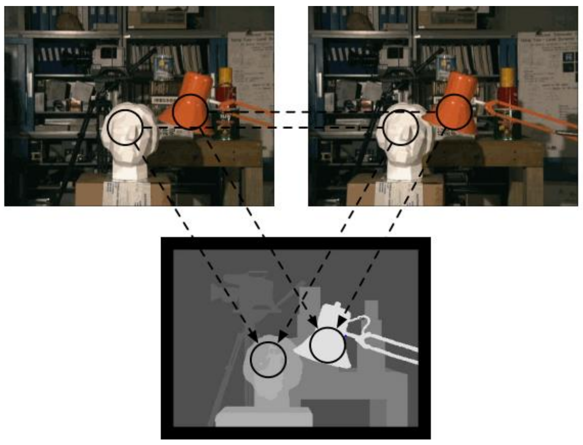


Figure I.8: Correspondence problem and DM estimation *[1]*.

Note that the disparity map in Figure I.8 is actually the ground truth which is a precise depth map generated using laser range-finder (LRF) and not a disparity map computed by an algorithm. It is important to keep in mind that is not possible to find perfect correspondences. Instead, it’s just a matter of finding the most probable correspondences [1]. The stereo correspondence algorithms can roughly be divided into feature-based and correlation-based, also known as region, area and intensity-based.

I.4.1 Feature based algorithms

Feature based algorithms extract features (e.g., edges, angles, curves, etc.) from images and try to match them in two or more views [13]. This technique has a lower computational complexity because only features have to be compared, lesser ambiguities due to the smaller number of possible matches and less sensitive comparisons between features are less sensitive to illumination changes [4, 14]. Despite these advantages, features-based technique’s main drawback, is that it yields a sparse DM, thus depth values are only known for corresponding features found and not for every pixel. Interpolation can be used to get a denser DM [4].

I.4.2 Correlation based algorithms

Correlation based also known as area-based algorithm produces a dense DM, which means that for each pixel of an image the algorithm tries to find its mutual pixel on the other view [13]. In order to find the best match, a pixel to be matched from a reference image (left image); essentially becomes the center of a small window of pixels, is compared to a similarly sized windows in the other image (right image) by spanning it along a horizontal axis for disparity ranges. Correlation metrics are used to provide a numerical measure of the similarity between the two windows [15, 16]. Based on these measures’ disparity selection is mostly performed in the Winner-Takes-All (WTA) strategy, i.e., the disparity with the lowest measurement is chosen [17, 18] as illustrated in Figure I.9.

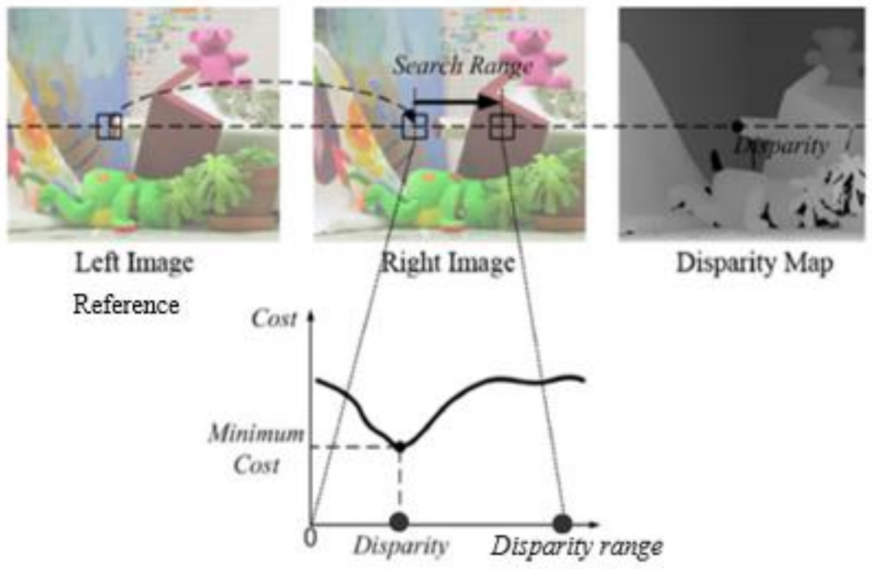


Figure I.9: Stereo matching procedures *[19]*.

I.4.2.1 Correlation metrics

Over the years, a large collection of intensity-based correlation metrics has been proposed. The simplest and least computationally intensive methods are Sum of Absolute Differences (SAD) and Sum of Squared Differences (SSD), which are based on the difference in pixel intensity and the assumption that corresponding areas in the two images will be similar in appearance [15]. The SAD and SSD are performed using equation (I.15) and (I.16) respectively:

$$SAD(x,y,d) = \sum_{i=x-n}^{x+n} \sum_{j=y-n}^{y+n} \left| f_{i,j} - g_{i+d,j} \right| \qquad (I.15)$$

$$SSD(x,y,d) = \sum_{i=x-n}^{x+n} \sum_{j=y-n}^{y+n} (f_{i,j} - g_{i+d,j})^2 \qquad (I.16)$$

Where *(x, y)* are the central pixel coordinates, $f_{i,j}$ and $g_{i,j}$ are the intensity of the pixel at coordinates *(i, j)* in the reference image (left image) and right image respectively, *n* defines the extend of the block in either direction around the center pixel, the window has therefore width and height 2n + 1. The variable *d* is the disparity range, that represents the distance across which the search for correspondence is conducted.

I.4.2.2 Non-parametric transformation

The problem with correlation metrics is that they are sensitive to radiometric distortion [20], and measures are not very distinctive when intensities of different pixels are the same such as in texture-less region. A commonly used approach to solve this problem, is to use larger windows, but this will lead to more false matches and increases the computation time, another approach is based on non-parametric transforms. Non-parametric transforms are applied to images prior to matching, which are based on the relative ordering of pixel intensities within a window, rather than the intensity values themselves. Two non-parametric transforms which are suited to fast implementation are the rank transform and the census transform.

1. **Rank transform**

The Rank transform of a central pixel *p* is defined as the number of pixels in the local neighborhood whose intensity is less than the intensity of *p*, by replacing each pixel in the window as a bit stream, where each pixel with intensity less than the center pixel's intensity is replaced by 1, otherwise it is set to 0, this principal is illustrated in Figure I.10. A pair of rank transformed images are matched using regular correlation metrics like SAD to compute the difference between image regions.

$$Rank\begin{pmatrix} 13 & 89 & 34 \\ 12 & 34 & 23 \\ 11 & 99 & 75 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & p & 1 \\ 1 & 0 & 0 \end{pmatrix} = 4$$

Figure I.10: Rank transformation using 3*3 window.

2. Census transform

The Census transform maps the window surrounding a central pixel p to a bit vector representing the local information about the pixel p and its neighboring pixels. If the intensity value of a neighboring pixel is less than the intensity value of the pixel p, then the corresponding bit is set to 1, otherwise it is set to 0. This principal is illustrated in Figure I.11.

$$Census\begin{pmatrix} 13 & 89 & 34 \\ 12 & 34 & 23 \\ 11 & 99 & 75 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & p & 1 \\ 1 & 0 & 0 \end{pmatrix} = 10011100$$

Figure I.11: Census transformation using 3*3 window

Two Census transformed images (left and right) are compared using Hamming distance (HD), which is a metric for comparing two binary data strings of same length, quantified by the number of bits that differ between the two strings. In stereo correspondence, HD is number the bits that differ between the two Census transformed images, as shown Figure I.12.

$$Census\ of\ left\ pixel = 1\underset{\updownarrow}{0}01110\underset{\updownarrow}{0}0$$

$$Census\ of\ right\ pixel = 1101111\,0$$

$$\Downarrow$$

$$HD = 2$$

Figure I.12: Hamming Distance principle.

I.5 Domain of U-V disparity

We assume that the stereo setup has coplanar pinhole cameras with the same intrinsic parameters and their horizontal co-axis are parallel to the road surface as depicted in Figure I.13, where the pitch angle to the ground plane is θ. For simplicity, the origin of World Coordinate System WCS (X_w, Y_w, Z_w) is put in the centre of the origins of the Left and Right Camera Coordinate Systems “LCCS” (X_l , Y_l , Z_l) and “RCCS” (X_r, Y_r, Z_r), with the optical axis of the WCS is parallel to the road surface [21].

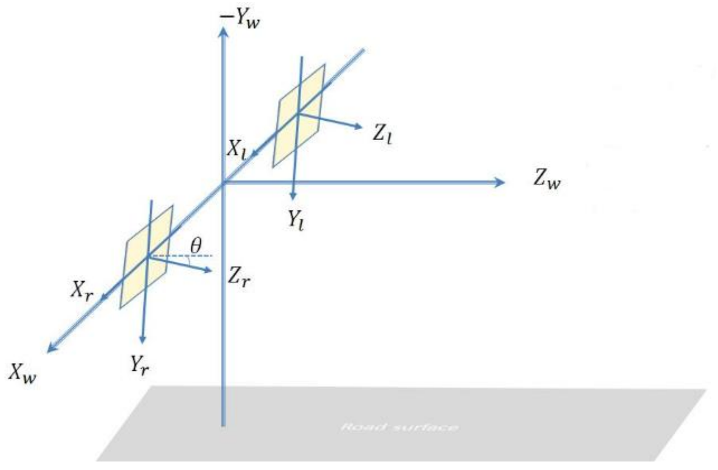


Figure I.13: Stereo vision setup in the world coordinate system *[22]*.

In the camera coordinate system, the position of a point in the image plane is given by its coordinates (u, v). The image coordinates of the projection of the optical center will be denoted by (u_0, v_0), assumed to be at the center of the image [21].

Based on Figure I.13, the transformation from WCS to the cameras coordinate systems is achieved by the combination of a vector translation of ± b/2, and a rotation around the X_w axis by an angle of –θ. Hence the projection matrix $\widetilde{P}$ is given by [21]:

$$\tilde{P} = K[R \quad T] = \begin{pmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & \pm\frac{b}{2} \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \end{pmatrix} \tag{I.17}$$

The projection matrix can be written as:

$$\tilde{P} = \begin{pmatrix} f & u_0 \sin\theta & u_0 \cos\theta & \pm f\frac{b}{2} \\ 0 & f\cos\theta + u_0 \sin\theta & -f\sin\theta + v_0 \cos\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \end{pmatrix} \tag{I.18}$$

Therefore, we obtain the transformation between the WCS homogenous coordinates $(X_w, Y_w, Z_w, 1)$ and the image coordinate $(u, v, 1)$:

$$\begin{pmatrix} u_{lr} \\ v \\ 1 \end{pmatrix} = \tilde{P} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{I.19}$$

From equations (I.19) and (I.18) we can simply calculate the camera's image coordinates (u_{lr}, v):

$$\begin{cases} u_{lr} = u_0 + f\dfrac{X_w \mp {}^{b}/_{2}}{Y_w \sin\theta + Z_w \cos\theta} \\ v = v_0 + f\dfrac{Y_w \cos\theta - Z_w \sin\theta}{Y_w \sin\theta + Z_w \cos\theta} \end{cases} \tag{I.20}$$

To simplify equation (I.20), we denote a new set of image coordinates (UV) with respect to the camera optical center [21]:

$$\begin{cases} U_{lr} = u_{lr} - u_0 + f\dfrac{X_w \mp {}^{b}/_{2}}{Y_w \sin\theta + Z_w \cos\theta} \\ V = v - v_0 + f\dfrac{Y_w \cos\theta - Z_w \sin\theta}{Y_w \sin\theta + Z_w \cos\theta} \end{cases} \tag{I.21}$$

Disparity Δ can also be deducted from equation (I.21):

$$\Delta = u_l - u_r = f\frac{b}{Y_w \sin\theta + Z_w \cos\theta} \tag{I.22}$$

Equations (I.21) and (I.22) describe the basic relationship between image coordinates (U, V) and disparity Δ for a coplanar stereo setup.

Combining equation (I.21) with we can find [21]:

$$\begin{cases} X_w = U\frac{b}{\Delta} - \frac{b}{2} \\ Y_w = \frac{b}{\Delta}(f\,sin\,\theta + V\,cos\,\theta) \\ Z_w = \frac{b}{\Delta}(f\,cos\,\theta - V\,sin\,\theta) \end{cases} \qquad (I.23)$$

I.6 3D planes projection in U-V-disparity

In a typical road scene, we can define different types of plans in the 3D coordinate system, horizontal, vertical, oblique and others as illustrated in Figure I.14. The projection characteristics of these plans on (U, V) coordinates are shown in Figure I.*14*.

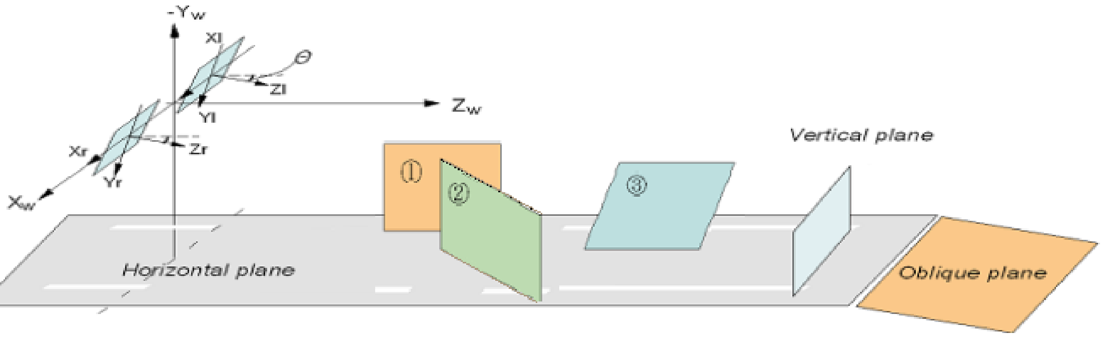


Figure I.14: The different plans in WCS *[22]*.

Table I.1: Projection characteristics of different planes in the U-V disparity map.

Plans	**The world coordinates system**	**U-V disparity domain**
Horizontal plan	$Y_w = h$	$\Delta = \frac{b}{h}(f \sin\theta + V\cos\theta)$
Vertical plan	$Z_w = p$	$\Delta = \frac{b}{p}(f\,cos\,\theta - V\,sin\,\theta)$
Oblique plan	$Z_w = \alpha Y_w + \beta$	$\Delta = f\frac{b}{\beta}(cos\,\theta - \alpha\,sin\,\theta) - V\frac{b}{\beta}(\sin\theta + \alpha\cos\theta)$
1	$X_w = r$	$\Delta = \frac{2\,b}{2r+b}U$
2	$Z_w = \rho X_w + \tau$	$\Delta = \frac{2\,b}{\rho b - 2\tau}(f\,cos\,\theta - \rho U - V\,sin\,\theta)$
3	$Y_w = \gamma X_w + \delta$	$\Delta = \frac{2\,b}{2\delta - \gamma b}(f\,sin\,\theta - \gamma U - V\,cos\,\theta)$

where $h, p, \alpha, \beta, r, \rho, \tau, \gamma$ and δ are constants for modeling the different planes, b is baseline and f is focal length.

From Table I.1 we can conclude that [21]:

- The horizontal planes in the WCS will be projected as straight slanted lines in the V-disparity domain.
- The vertical planes in the WCS will be projected as nearly vertical straight lines in the V-disparity domain.
- The oblique planes in the WCS will be projected as straight slanted lines in the V-disparity domain.
- Plane labeled as 1 in Figure I.14 represent side surfaces parallel to the Y-Z plane in the WCS, such planes will be projected as straight vertical lines in the U-disparity domain.
- Plane labeled as 2, 3 in Figure I.14 represent oblique side surfaces and special case for side surfaces respectively, these planes can be approximately projected as straight slanted lines in the U-disparity domain when the angle θ is small.

I.6.1 U and V disparity maps generation

U-V Disparity is built upon a given DM. To compute the V-disparity map (VDM), a histogram of each row of the DM is computed by accumulating pixels with the same disparity *d* in each row, these histograms are then placed onto a VDM at the corresponding row. Following a similar idea, the U-disparity map (UDM) can also be generated by computing a histogram of each column of the DM and placing these histograms onto a UDM at the corresponding column. The principle is shown in Figure I.15.

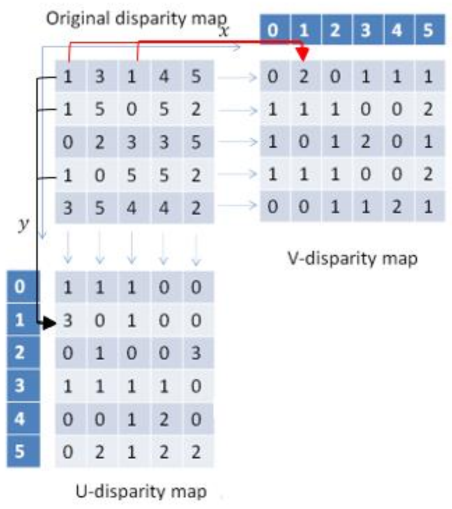


Figure I.15: U-V disparity maps generation *[21]*.

I.6.2 Obstacles detection using UV disparity maps

As described in the previous section, most kind of 3D surfaces in WCS are mapped into lines in V-disparity and the U-disparity domains. Thus, extracting those lines leads to obstacles detection. Before that, it is necessary to remove noise from the UDM caused by road surface or by small (neglected) objects, this can be achieved with applying a threshold, the threshold here indicates the height of potential obstacles measured in pixels. The characteristics of obstacles in UDM and VDM are as follows [23]:

- The projection of horizontal ground in the V disparity map is a diagonal line. The pixel of each line increases gradually from left to right in the V disparity map, therefore the disparity of the diagonal line decreases gradually. So, we can ensure the horizontal ground through detecting the diagonal line in the disparity map.

- The projection of an obstacle in the VDM is a vertical line because the distances between the obstacle and the two cameras are nearly the same. The height of the obstacle can be calculated through the length of the vertical line.
- The location of the vertical line represents the distance between obstacle and the cameras, the more left, the nearer. If a vertical line intersects a diagonal line, it means that the obstacle is on the ground.
- The projection of the obstacle in the UDM is a horizontal line and the width of the obstacle can be calculated through the length of the horizontal line.
- The location of the horizontal line represents the distance between obstacle and the cameras, the lower, the nearer.
- By combining information provided by UDM and VDM, we can obtain the size and the location of the obstacle.

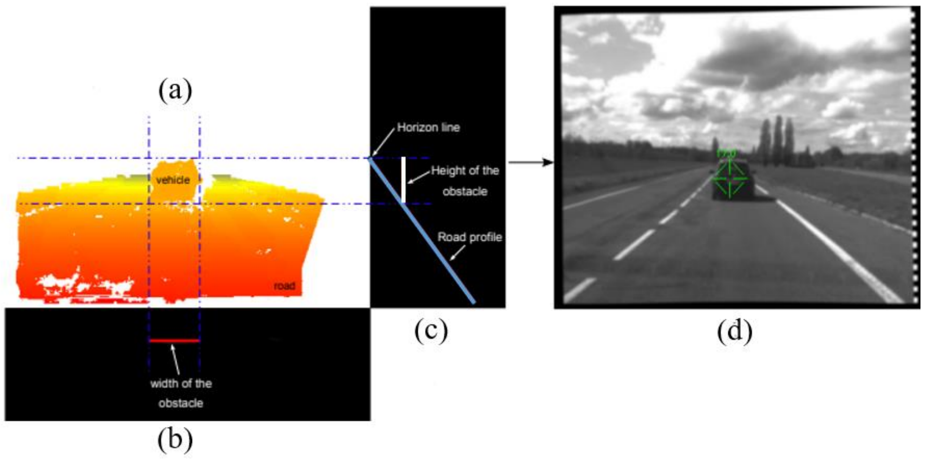


Figure I.16: Obstacle detection using UV disparity. (a) DM. (b) UDM. (c) VDM. (d) sample stereo image *[24]*.

Figure I.16 shows the VDM and UDM with different extracted lines, the blue line corresponds to decreasing disparity where it represents the ground, the vertical white line represents the car (obstacle) as the disparity pixels for the car are of the same value over the v-domain and the red line in the u-domain represents the width of the obstacle.

I.7 Summary

Stereo vision is an environment perception technique that imitates the sense of vision, it uses stereo cameras to estimate depth information in a scene. The process starts with; image acquisition where two images (left and right) of the same scene are taken from different angles, camera calibration is an offline process but central for image rectification and distortion removal, at that point, images are set for stereo matching and depth estimation. Stereo correspondence is the main task in stereo vision; where each pixel in one image is matched to a pixel in the other image. This task can be attained with correlation matrices such as SSD and SAD. Which are the fastest, simplest and most suitable when real-time performance is a concern, the final results are presented as a DM. The DM can be used to detect any possible obstacles by extracting vertical and horizontal lines in VDM and UDM respectively.

Chapter II: Obstacle detection system design

II.1 Introduction

In the previous chapter, we introduced the theory behind stereovision-based obstacle detection. This chapter will present a global software design for the obstacle detection system; translating theoretical concepts to algorithms that will be used in our implementation. Including the Rank transform, SAD correlation, U-V DM generation, vertical and horizontal lines extraction, and the obstacle detection algorithms.

II.2 Overview of the obstacle detection system

The global system design illustrated in Figure II.1, consists mainly of the obstacle detection system, that will read left and right images from an SD card. The reading process is performed pixel by pixel in a row. The data are transferred to a memory for the system to start performing the obstacle detection algorithm. The obstacle detection system generates different images and maps, starting with the Rank transform images, the DM, UDM, VDM and finally obstacles detection. For debugging purposes, the maps are saved in a frame buffer in the memory, that will be displayed with the HDMI controller.

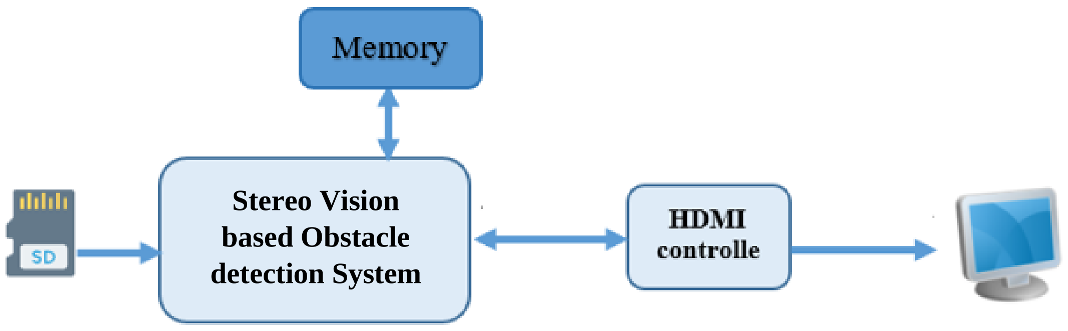


Figure II.1: Global design of obstacle detection system.

The obstacle detection system is the central component in our design, the system emulates the camera effect by reading the stereo images form the SD card pixel by pixel and transfer the data to a memory. Once the data are placed in memory, the Rank transform is applied on left and right images to generates the corresponding Rank transform images. The images are further processed by the SAD correlation to produce the DM, that is used to produce the UDM and VDM. Next, the horizontal and vertical lines are extracted from the UDM and VDM respectively. The extracted lines are arranged based on their locations, each pair of horizontal and vertical line placed at same location defines an obstacle boundary. The obstacle detection system design is shown in Figure II.2.

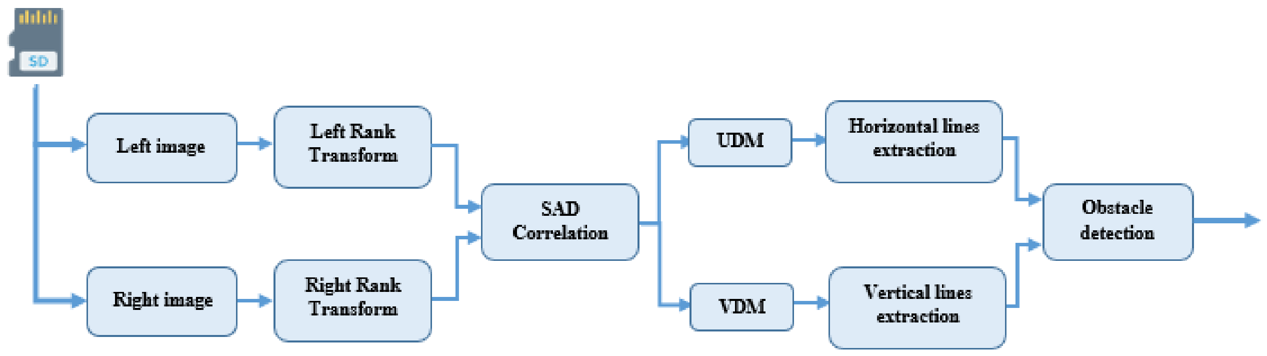


Figure II.2: Stereovision-based obstacle detection system design.

II.3 Obstacle detection system design

For the obstacle detection system, we used several algorithms, in this section we will present the main algorithms used in our implementation.

II.3.1 RANK transform

The Rank transform of a central pixel is defined as the number of pixels in the local neighborhood whose intensity is less than the intensity of the central pixel. Algorithm 1 shows the pseudocode of the Rank transform computation. The input *'I'* is the grayscale image of width *columns* and height *rows*. The index *'i'* of the loop at line 1 is used to cover the rows, while the *'j'* of the loop at line 2 covers the pixels in a row, and the *'w'* represents the size of the window. The algorithm starts with the first central pixel at *((w-1)/2, (w-1)/2)* and ends with the last central pixel at *(row-1-(w-1)/2, columns-1-(w-1))*, since we cannot calculate the Rank of the outer pixels as shown in the example in Figure II.3.

Algorithm 1 : RANK transform

Input : *Grayscale image I of size (rows * columns), Window width w (odd number);*

Output : *Image R of size (rows-(w-1))*(columns-(w-1));*

```
1.   for i =(w-1)/2 to row-1-(w-1)/2 do
2.     for j=(w-1)/2 to columns-1-(w-1)/2 do
3.           rank =0;
4.           Central_pixel= I (i, j);
5.           for m= -(w-1)/2 to (w-1/2) do
6.             for n= -(w-1)/2 to (w-1)/2 do
7.               if I(i+m, j+n) < central_pixel then
8.                   rank=rank+1;
9.               end if
10.
11.          R(i,j)=rank;
```

The Rank value '*rank*' is first initialized to 0 before starting comparison, and the central pixel intensity is saved in variable '*Central_pixel*'. The indexes '*m*' and '*n*' of loops (at line 5 and 6) are used to cover rows and columns of the window respectively. a comparison is performed between the central pixel and pixels of the window, each time a pixel is smaller than the central pixel, the rank value is incremented. Finally, the obtained Rank values are stored in 2D array '*R*' of size *(rows-(w-1))*(columns-(w-1))* as shown in Figure II.3.

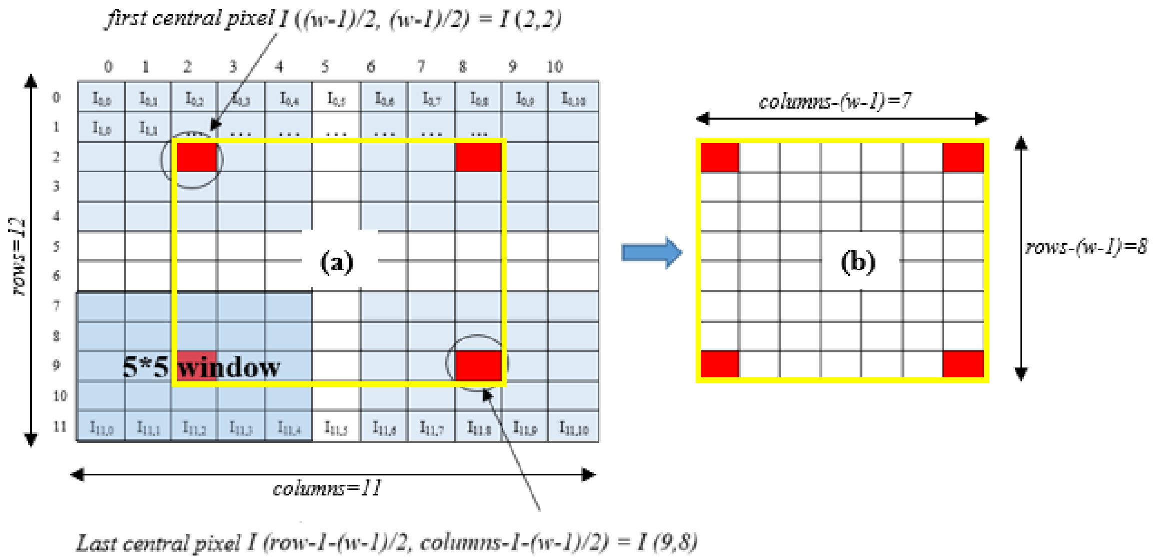

first central pixel I ((w-1)/2, (w-1)/2) = I (2,2)
rows=12
columns=11
5*5 window
(a)
columns-(w-1)=7
rows-(w-1)=8
(b)
Last central pixel I (row-1-(w-1)/2, columns-1-(w-1)/2) = I (9,8)

Figure II.3: Size change in the Rank transform image compared to original image, (a) the original image *I* of size 12×11 pixels, (b) image of rank transform *R* of size 8×7 pixels using 5*5 window.

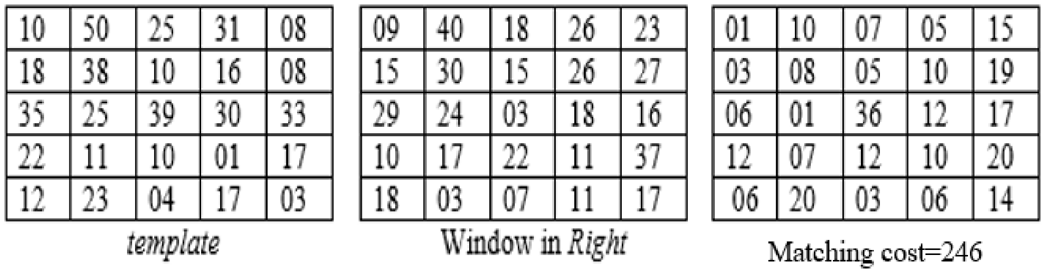
II.3.2 SAD correlation

As explained in sectionI.4.2.1, The SAD correlation is a measure of the similarity between image windows. It is calculated by taking the absolute difference between each pixel in a template window belonging to the reference (left) image, and the corresponding pixel in the window belonging to the right image. This measurement is performed over a disparity range, and the WTA method is used to select the best disparity.

Algorithm 2 shows the pseudocode of the SAD correlation metric. The inputs '*R_left*' and '*R_right*' represent the Rank transform of the left and right images, the width and height of the images is '*rows*' and '*columns*' respectively. The index '*i*' of the loop at line 1 is used to cover the rows, while the '*j*' of the loop at line 2 covers pixels in a row. The algorithm starts with the first central pixel at (*(w-1)/2, (w-1)/2)* and ends with last central pixel at (*row-1-(w-1)/2, columns-1-(w-1))* as shown in Figure II.5. '*w*' represents the size of the window, '*Dmax*' is the disparity range. The algorithm starts by selecting '*R_left*' as a reference image and then saving the window to be matched in a 2D array '*template*', the indexes '*c*' and '*r*' are used to store pixels intensity in the '*template*' while the '*m*' and '*n*' indexes are used to go through the window pixels in '*R_left*', the index '*d"* is used to cover the disparity range '*Dmax*'. The matching cost '*M_C*' is calculated between '*template*' and a window in '*R_right*' pixel by pixel as shown in Figure II.4. The minimum matching cost '*Minimum*' and disparity map '*DM(i,j)*' of a pixel *(i,j)* are initialized with the first '*M_C*' calculated and disparity 0 (lines 20,21), these variables are updated such that the '*Minimum*' holds the minimum *M_C* and the *DM(i,j)* holds the index of the minimum *M_C*.

Algorithm 2 : SAD algorithm

```
Input    :  RANK transform R_left, R_right (rows*columns),
            Window width w (odd number),
            Disparity range Dmax;

Output   :  DM of size (rows-(w-1))*(columns-(w-1));

1.   Create array template of size (w*w);
2.   M_C=0;
3.   Minimum;
4.   for i=(w-1)/2 to row-1-(w-1)/2 do
5.     for j=(w-1)/2 to columns-1-(w-1)/2 do

6.           for m= -(w-1)/2 to (w-1)/2 do
7.             r=0, c=0;
8.             for n= -(w-1)/2 to (w-1)/2 do
9.               template(r,c)=R_left(i+m,j+n);
10.              c=c+1;
11.            r=r+1;

12.          for d=0 to Dmax do
13.            for m= -(w-1)/2 to (w-1)/2 do
14.              r=0, c=0;
15.              for n= -(w-1)/2 to (w-1)/2 do
16.                M_C= M_C+ abs(template(r,c)-R_right(i+m,j+n));
17.                c=c+1;
18.              r=r+1;

19.            if d=0 then
20.              Minimum = M_C;
21.              DM(i,j)=0;
22.            elseif  M_C < Minimum
23.              Minimum = M_C;
24.              DM(i,j)=d;
25.            end if
```


template
Window in Right
Matching cost=246

10	50	25	31	08
18	38	10	16	08
35	25	39	30	33
22	11	10	01	17
12	23	04	17	03

09	40	18	26	23
15	30	15	26	27
29	24	03	18	16
10	17	22	11	37
18	03	07	11	17

01	10	07	05	15
03	08	05	10	19
06	01	36	12	17
12	07	12	10	20
06	20	03	06	14

Figure II.4: Matching cost using SAD of 5*5 window.

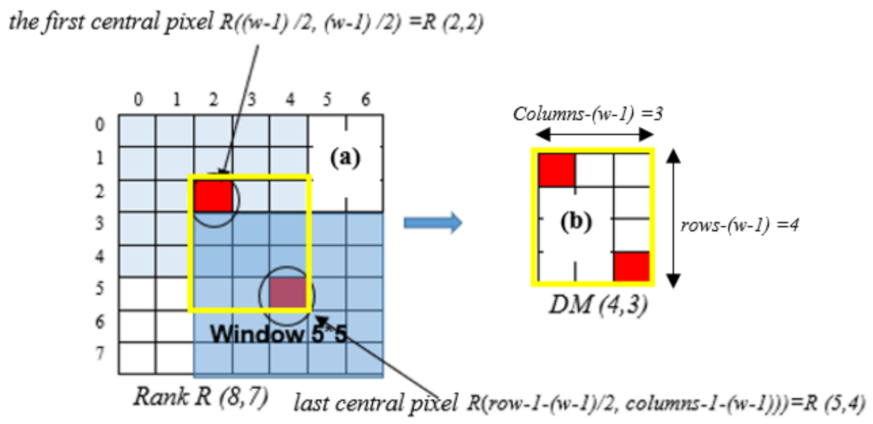

the first central pixel R((w-1) /2, (w-1) /2) =R (2,2)
0 1 2 3 4 5 6
0 1 2 3 4 5 6 7
(a)
Window 5*5
Rank R (8,7)
last central pixel R(row-1-(w-1)/2, columns-1-(w-1)))=R (5,4)
Columns-(w-1) =3
(b)
rows-(w-1) =4
DM (4,3)

Figure II.5: Size change in the DM compared to Rank transform image using 5*5 window.

II.3.3 UDM and VDM generation

To compute the VDM, a histogram of each row of the DM is computed by accumulating pixels with the same disparity in each row, these histograms are then placed onto a VDM at the corresponding row. Following a similar idea, the UDM can also be generated by computing a histogram of each column of the DM and placing these histograms onto a UDM at the corresponding column.

Algorithm 3 : UV disparity map algorithm

```
Input   :  Disparity map DM of size (rows-2(w-1))* (columns-2(w-1));
           Search range Dmax;

Output  :  array VDM of size (rows-2(w-1)*Dmax);
           array UDM of size (Dmax* (columns-2(w-1)));

1.   initialize elements of VDM and UDM array to zero.
2.   for i=0 to rows-2(w-1) do
3.       for j=0 to columns-2(w-1) do
4.           disparity=DM(i,j);

5.           UDM (disparity,j) = UDM(disparity,j)+1;
6.           VDM (i, disparity) = VDM(i,disparity)+1;
```

Algorithm 3 shows a pseudocode to generate the '*UDM*' and '*VDM*'. The input is the '*DM*' of size *(rows-2(w-1))* (columns-2(w-1)), 'rows'* and *'columns'* are the height and width of the stereo images. We start reading the disparity values in array '*DM*' pixel by pixel in a row, and store the value in variable '*disparity*'. For the '*UDM*', the '*disparity*' represents the row index at column '*j*' to be incremented, while for the '*VDM*', it represents the column index on row '*i*' to be incremented.

II.3.4 Vertical and horizontal lines extraction

Algorithm 4 shows a pseudocode of the vertical lines extraction, vertical lines are extracted from the '*VDM'* of size *((rows-2(w-1)) *Dmax),* the '*V_ threshold'* input represents the minimum intensity of the pixel, this parameter is used as filtering technique to remove noise. A line is defined by its starting point and the length, a vertical line in an array is a set of pixels that would lay on the same column along succussive rows as shown in Figure II.6, thus, a '*start'* of a line is the first-row index with intensity is greater than '*V_ threshold'*, while the '*length'* is the number of succussive points starting from the '*start'* point. The extracted lines are saved in a 2D array *V_lines* of size *((rows-2(w-1)) *Dmax);* to store line information at each point of '*VDM'*, the array is initialized to zero before starting. The indexed *'i'* and *'j'* are used to cover columns and rows respectively.

Algorithm 4 : Vertical line extraction algorithm

```
Input    :  VDM of size ((rows-2(w-1)) *Dmax);
            V_ threshold;

Output   :  2D array V_lines of size ((rows-2(w-1)) *Dmax);

1.   Start, length;
2.   Initialize V_line elements to zero;
3.   for i=0 to Dmax do
4.       Start=0;
5.       for j=0 to rows-2(w-1) do

6.           if VDM(j,i) > V_threshold then
7.               if  j=0 then
8.                       Define a line V_line[j,i ]
9.                       Start of V_line[j,i ]=j;
10.                      length of V_line[j,i ]=1;
11.                      Start=j;
12.              else
13.                  if length of V_line [j-1, i]!=0  then
14.                      length of V_line[start,i]= length of V_line[start,i]+1
15.                      length of V_line[j,i]= V_line[j,i]+1;
16.                  else
17.                      Define a line V_line[j,i ]
18.                      Start of V_line[j,i ]=j;
19.                      length of V_line[j,i ]=1;
20.                      Start=j;
21.                  end if
22.              end if
23.          end if
```

The algorithm starts by fixing a column then traversing the rows in *VDM*, we search for pixels with intensity greater than a predefined '*V_threshold'*, once a pixel is found, the row index is considered as a '*start'* of a line with length 1. For next pixel in the column, if its intensity is greater '*V_threshold'* we can say that it is a continuity of the line, thus we increment the length of the line started at previous pixel, we increment the length at that point as a sign for the coming pixel that the current pixel belongs to a line. The length of a line represents the height of potential obstacles. Lines are stored in a 2D array *'V_lines'* at the exact location of the line.

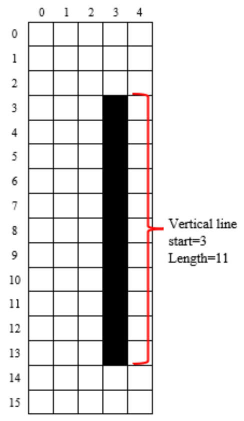


Figure II.6: Vertical line in VDM array example.

Algorithm 5: Horizontal line extraction algorithm

```
Input   :  UDM of size (Dmax*(columns-2(w-1)));
           H_threshold;
Output  :  create 2D array H_lines of size (Dmax*(columns-2(w-1)));
1.  Start, length.;
2.  Initialize H_line elements to zero;
3.      for i=0 to Dmax do
4.          start=0;
5.          for j=0 to columns-2(w-1) do

6.              if UDM(i,j) > H_ threshold then
7.                  if j=0 then
8.                      Define a line H_line[i,j ]
9.                      Start of H_line[i,j ]=j;
10.                     length of H_line[i,j ]=1;
11.                     Start=j;
12.                 else
13.                     if U_line[i,j-1].length !=0 then
14.                         length of H_line[i,start]= length of H_line[i,start]+1
15.                         length of H_line[i,j]= H_line[i,j]+1;
16.                     else
17.                         Define a line H_line[i,j ]
18.                         Start of H_line[i,j ]=j;
19.                         length of H_line[i,j ]=1;
20.                         Start=j;
21.                     end if
22.                 end if
23.             end if
```

Algorithm 5 shows a pseudocode of the horizontal line extraction, horizontal lines are extracted from the '*UDM'* of size *(Dmax*(columns-2(w-1)))*, the '*H_ threshold'* input is used to remove noise from the '*UDM'* caused by road surface or by neglected objects. A line is defined by its starting point and the length, a horizontal line in an array is a set of points that would lay on same row along succussive columns as shown in Figure II.7, thus, a '*start'* of a line is the first column index with intensity greater than '*H_ threshold'*, while the '*length'* of the line is the number of succussive points starting from the '*start'* point. The extracted lines are saved in a 2D array *'H_lines'* of size *(Dmax*(columns-2(w-1)))*, to store line information at each point of *'UDM'*, the array is initialized to zero. The indexes *'i'* and *'j'* are used to cover rows and columns respectively. The search of vertical line is done by fixing a row and then traversing the columns in *'UDM'*, we search for pixels with intensity greater than a predefined *'H_threshold'*, once a pixel is found, the column index is considered as a *'start'* of a line with length 1. For next pixel, if its intensity is greater *'H_threshold'* we can say that it is a continuity of the line, thus we increment the length of the line started at the previous pixel, we increment the length at that point as a sign for the coming pixel that the current pixel belongs to a line. The length of a line represents the width of potential obstacles. Lines are stored in a 2D array *'H_lines'* at the exact location of the line.

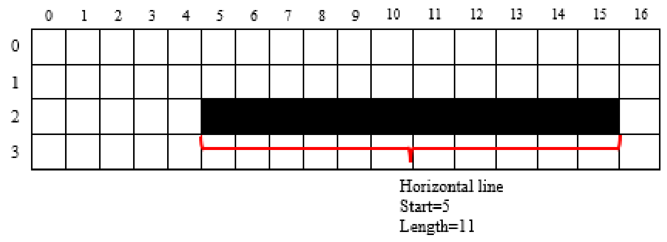


Figure II.7: Horizontal line in UDM array example.

II.3.5 Obstacle detection

Algorithm 6 shows a pseudocode of the obstacle detection, the inputs *'V_lines'* and *'H_lines'* contain the vertical and horizontal line information. An obstacle is modeled by a box that covers the object, the box is represented by 4 values: *'H_start'*, *'H_end'*, *'V_start'*, and *'V_end'* as shown in Figure II.8. That information is saved in a 1D array *'obstacle'*, the *'n'* at the end of execution will contain the number of obstacles detected. After extracting all the horizontal and vertical lines, lines are arranged based on row and column index, such that each horizontal and vertical line that have the same index for row and column respectively, define an obstacle. In the *'H_lines'* array, we search for lines that have a length greater than 1, we save location of the line as variable *'distance'*, then in the '*V_lines'* array we search for lines located at column index *'distance'*, that have a length greater than 1. If a line is found then it is considered as an obstacle which has horizontal and vertical boundaries '*H_star't, 'H_end'*, *'V_start'*, and *'V_end'.* The *'H_start'* and *'H_end'* represent the horizontal boundaries where '*H_end = H_start + length'* of the horizontal line. The *'V_start'* and *'V_end'* represent the vertical boundaries where *V_end = V_start + length of the vertical line.* The time complexity of the algorithm is $O(R*C^{*}Dmax)$, where *R=rows-2(w-1)* and *C= columns-2(w-1).*

Algorithm 6 : Obstacle detection algorithm

Input : *H_lines of size (Dmax*(columns-2(w-1))),*
*V_lines of size ((rows-2(w-1)) *Dmax);*

Output : *1D obstacle array*

```
1.  n=0;
2.  H_start, H_end, V_start, V_end;
3.  for i=0 to Dmax do
4.      for j=0 to columns-2(w-1) do
5.          if H_lines(i,j).length > 1 then
6.              distance=i;
7.              for v=0 to rows-2(w-1) do
8.                  if V_lines (v, distance). length > 1 then
9.                      n=n+1;
10.                     Define an obstacle[n];
11.                     H_start of obstacle[n]=j;
12.                     H_end of obstacle[n]=j+length of H_lines(distance,j);
13.                     V_start of obstacle[n]=v;
14.                     V_end of obstacle[n]=v+length of V_lines(v,distance);
15.                 end if
16.             v= V_end of obstacle[n];
17.         end if
```

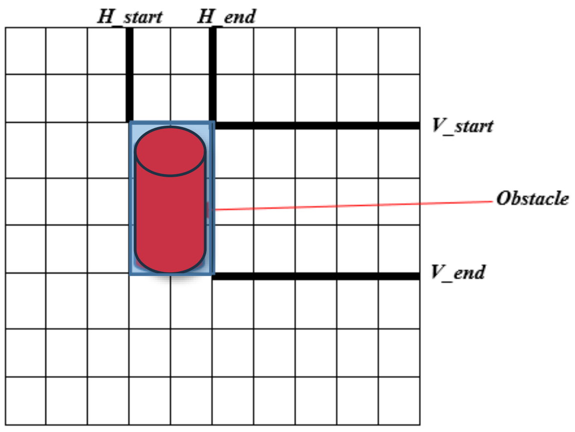


Figure II.8: Example of obstacle boundaries.

Chapter III: Software implementation and experimental results

III.1 Introduction

An obstacle detection system must be a Real-Time System (RTS), in which computing the right output is not enough and the timing of the result is an integral part of the correctness. Often the exact timed behavior is not required, but rather a guarantee of producing the output before the deadline. In this chapter, we will introduce the implementation of the obstacle detection system, along with the used tools, as well as the experimental results followed by system profiling to check if our system meets the RTS requirements.

III.2 Development tools

In this section we will present the tools that we used in the implementation, including Xilinx software, hardware development environments and embedded platforms that offer a comprehensive set of familiar and powerful tools, libraries, and methodologies. These environments reduce the development time while allowing the creation of custom hardware easily. Along with communication protocols and some basic concepts.

III.2.1 Software and hardware environments

The Xilinx vivado design suit is a hardware design tool, that provides an intuitive graphical user interface (GUI) with powerful features. All of the features are written in native tool command language (Tcl) format. The Xilinx Software Development Kit (SDK) provides a complete environment for creating software applications targeted for Xilinx embedded processors. It includes a GNU-based compiler toolchain, drivers for Xilinx IPs and bare-metal board support packages (BSP), middleware libraries for application-specific functions, and an IDE for C/C++ bare-metal and Linux application development and debugging. Based upon the open-source Eclipse platform, SDK incorporates the C/C++ Development Toolkit (CDT) [25]. For our implementation we used vivado and SDK version 2018.2.

III.2.2 Zynq Evaluation and Development board (Zedboard)

ZedBoard is a development kit based on the Xilinx Zynq® 7000 All Programmable System On Chip (AP SoC) (XC7Z020 -1clg484), which contains many interfaces to implement a wide range of applications. The interfaces available on the board, as well as other modules such as the ADV7511 transmitter, DDR3 memory, the SD card and the Zynq SOC are shown in Figure III.1.

- **Zynq SOC:** it is the central element of the Zedboard that allows managing all the resources that are included. It consists of dual ARM cortex-A9 referred to as the Processing System (PS) and a Field Programmable Gate Array (FPGA), its architecture will be described in detail in the coming section.
- **DDR3:** ZedBoard includes two Micron DDR3 128 Megabit x 16 memory components creating a 32- bit interface, totaling 512 MB. The DDR3 is connected to the hard memory controller in the PS. This memory is used to load applications where it is executed by the processor, or to run an operating system.
- **SD card:** is a 4GB card that can be used for non-volatile external memory storage as well as booting the Zynq SOC.
- **ADV7511 transmitter:** is an Analog Devices provides a digital video interface to the High-Definition Multimedia Interface (HDMI) in ZedBoard. Supporting 1080p60 with 16-bit, YCbCr, 4:2:2 mode color. The ADV7511 supports different audio input formats such as Inter

IC Sound (I2S) and Sony/Philips Digital Interface (SPDIF). The I2S interface is not connected on ZedBoard [26].

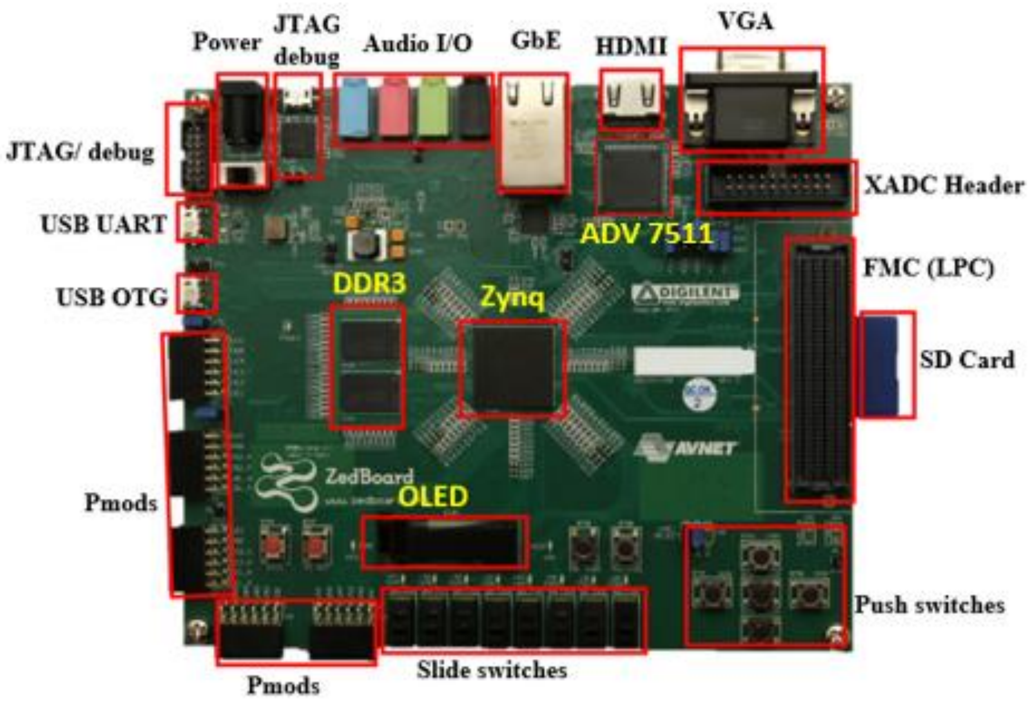


Figure III.1: Zedboard development board interfaces.

III.2.3 Inter-Integrated Circuit (IIC) protocol

For configuration, ADV7511’s registers must be programmed using the (IIC) protocol, its programming address is 0x39 [26]. IIC is a synchronous serial communication protocol, a half-duplex mode, multi-master, and multi-slave. It uses two lines, Serial Data (SDA) and Serial Clock (SCL) as shown in Figure III.2. The exchange procedure begins with the master forming the start state: it generates the transition of the SDA line signal from a high state to a low state when the level on the SCL line is high. This transition is perceived by all devices connected to the bus as a sign of the beginning of the exchange procedure. After that, the first byte sequence is sent to indicate the slave address to which the data is sent. Followed by an internal address register of the slave, then the data. Each byte of data (including the address byte) is followed by one ACKnowledgement (ACK) bit from the slave. The exchange procedure ends with the master forming a stop state, the transition of the SDA line state from a low state to a high state when the SCL line state is high. The start and stop states are always generated by the master. The process is illustrated in Figure III.3.

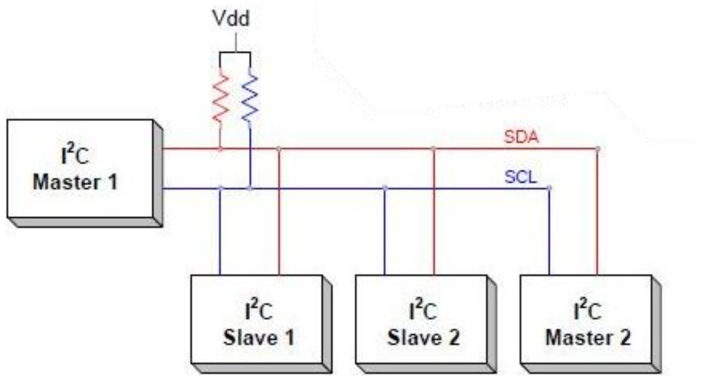


Figure III.2: Typical I2C protocol bus *[27]*.

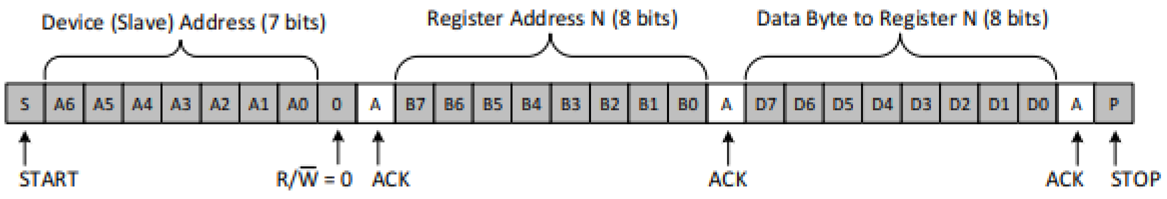

Device (Slave) Address (7 bits)
Register Address N (8 bits)
Data Byte to Register N (8 bits)
S A6 A5 A4 A3 A2 A1 A0 0 A B7 B6 B5 B4 B3 B2 B1 B0 A D7 D6 D5 D4 D3 D2 D1 D0 A P
START
R/W̄ = 0
ACK
ACK
ACK
STOP

Figure III.3: I2C write to slave device process *[28]*.

III.2.4 Video signals and timing

All video systems require management of video timing signals, which are used to synchronize processes of transmitting data to the display interface. Video format is characterized by its timing parameters: active video, front porch, sync pulse, back porch for horizontal and vertical line, in addition to pixel clock which represent the speed at which pixels are transmitted. Each active video line is enclosed by horizontal blanking periods during which horizontal sync signals will happen. The time between the beginning of the blanking period and the start of the sync is call front porch and the time between the end of the sync signals and the end of the blanking period is call back porch. The horizontal active time is the time between two horizontal blanking periods and it is the time where pixels are transmitted. The process of transmitting a set of pixels from a horizontal line is illustrated in Figure III.4.

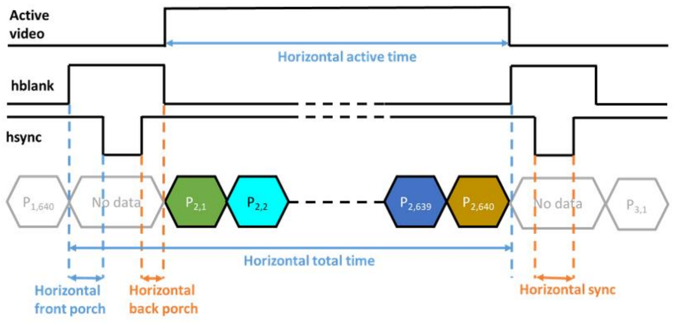


Figure III.4: Horizontal video timing of second line *[29]*.

Each frame is enclosed in vertical blanking periods during which vertical sync signals will happen. The vertical active time is the time between two vertical blanking periods and represents the end and start of one frame transmission. As shown in Figure III.5.

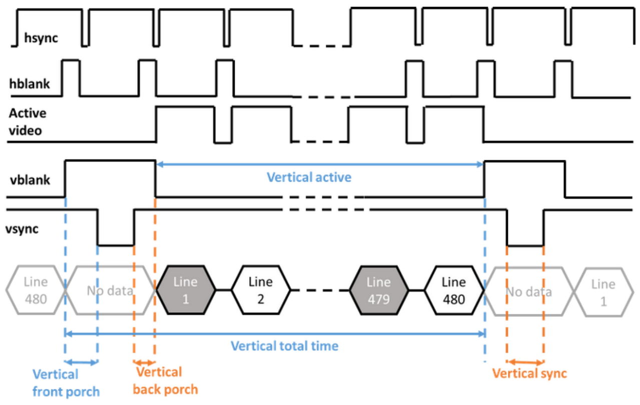


Figure III.5: Vertical video timing *[29]*.

III.3 Zynq architecture

The Xilinx Zynq AP SOC combines a dual-core ARM Cortex-A9 referred to as Processing System (PS), with a traditional FPGA based on Artix®-7 logic, known as the Programmable Logic (PL). The interface between the different elements within the Zynq architecture is based on the Advanced eXtensible Interface (AXI) standard as shown in Figure III.6, which provides for high bandwidth and low latency connections. The inclusion of a hard processor in Zynq delivers significant performance improvements. Also, by simplifying the system to a single chip, the overall cost and physical size of the device are reduced.

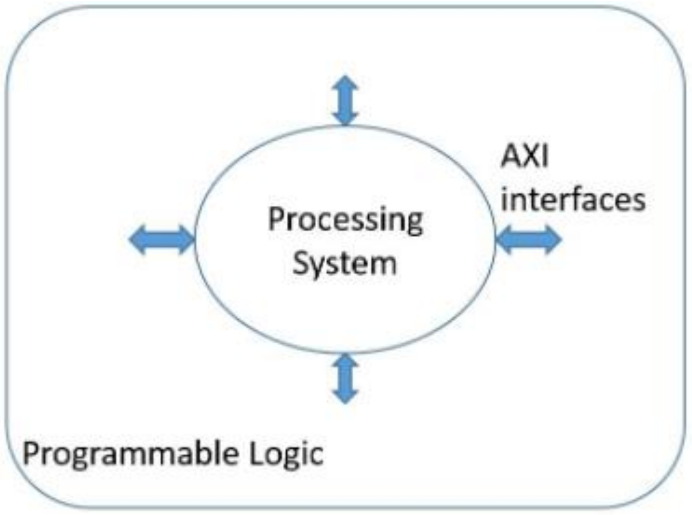


Figure III.6: Zynq overall view *[26]*.

III.3.1 Zynq Processing system (PS)

The PS comprises four major blocks: Application processor unit (APU), Memory interfaces, I/O peripherals (IOP) and Interconnect as illustrated in Figure III.7.

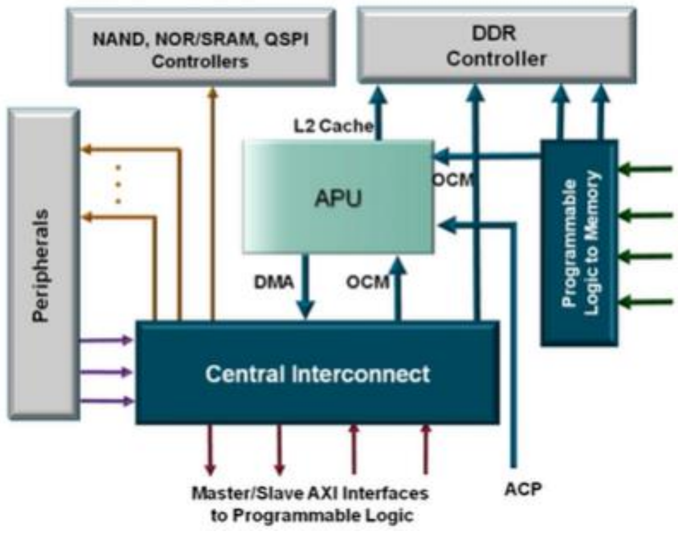

NAND, NOR/SRAM, QSPI Controllers
DDR Controller
L2 Cache
APU
OCM
Programmable Logic to Memory
Peripherals
DMA
OCM
Central Interconnect
Master/Slave AXI Interfaces to Programmable Logic
ACP

Figure III.7: Zynq Processing System architecture *[30]*.

1. The APU includes the dual ARM Cortex-A9 core processor that runs at up to 667MHz, each one is associated with NEON unit, floating point unit (FPU), memory management unit (MMU) and L1 caches. In addition, the APU also consists of Snoop Control Unit (SCU), on-chip memory (OCM), 8-channel Direct Memory Access (DMA), 32-bit system watchdog timer (SWDT), General Interrupt Controller (GIC) and 64-bit triple-timer controller (TTC) blocks.

2. Memory interfaces support several memory technologies, including a dynamic memory controller that supports DDR3, DDR3L, DDR2, and LPDDR2 memories. The static memory controllers support a NAND flash interface, a Quad-SPI flash interface, a parallel data bus, and a parallel NOR flash interface.

3. IOP corresponds to different interface standards, which are designed for communication with modules external to PS.

4. The APU, memory interface unit, and the IOP are all connected to each other and to the PL through a multilayered ARM AMBA AXI interconnect. The interconnect is non-blocking and supports multiple simultaneous master-slave transactions

III.3.2 Zynq Programmable logic (PL)

The PL consists of configurable logic blocks (CLBs) that contain two slices, these slices have no connection between them. Each slice contains four look-up tables (LUTs) used to implement a logic function of up to six inputs with, eight storage elements (Flip-flops FFs). The LUTs have memory capability within, register and shift register functionality. As a result, between 25–50% of all slices can use their LUTs as distributed 64-bit RAM, as 32-bit shift register (SRL32), or as two 16-bit shift registers (SRL16). Moreover, there are switch matrix to provide the connections among the different parts within and between the CLBs, as well as other parts of the PL, Block RAMs, and digital signal processing (DSP) slices That allow performing different arithmetic operations [31], as shown in Figure III.8.

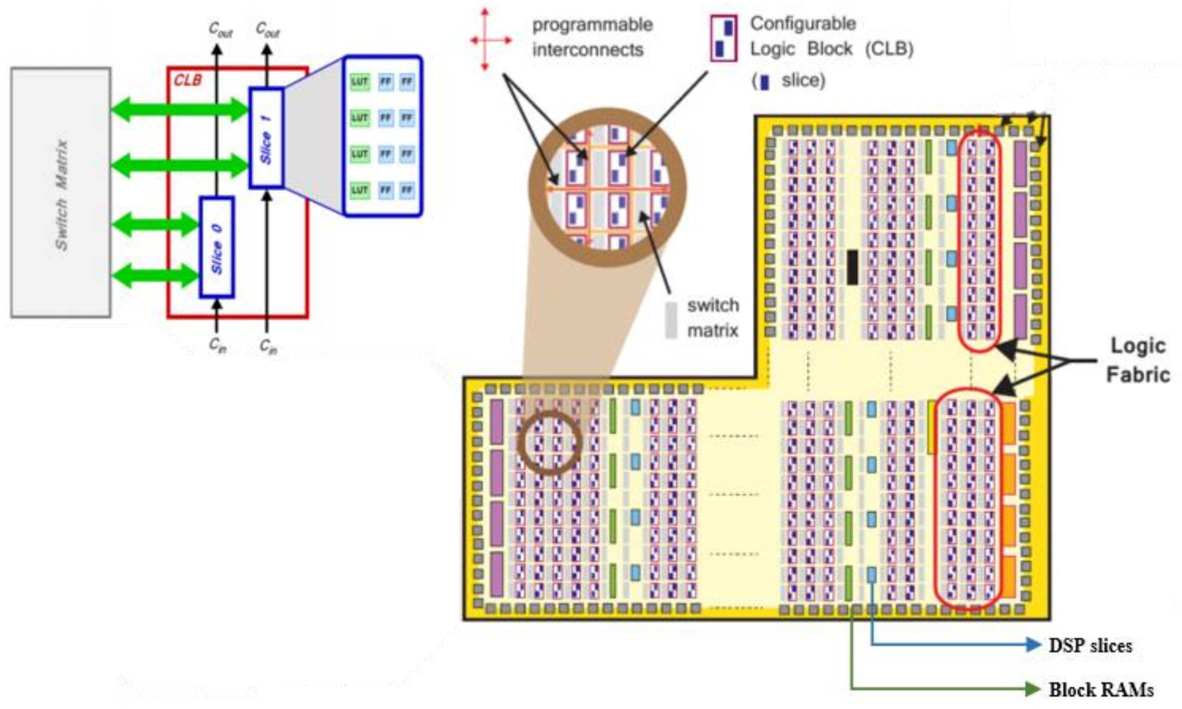


Figure III.8: Arrangement of Zynq PL basic resources *[32]*.

The Zynq-7000 devices contain the same PS components while the PL resources vary between the devices as shown in Table III.1. The Zedboard contains 7z020 Zynq device.

Table III.1: The PL resources per Zynq device type [33].

PL Resource	**Zynq device type**				
	7z007s	**7z014s**	**7z020**	**7z035**	**7z100**
Logic Slices	3,600	10,150	13,300	42,975	69,350
LUTs	14,400	40,600	53,200	171,900	277,400
Memory Resources BRAMs (kB)	50	107	560	500	755
Clocking	2	4	4	8	8
DSPs	66	170	220	900	2,020

III.3.3 Advanced eXtensible Interface (AXI) protocol

AXI is a standardized Intellectual property (IP) interface protocol based on the Advanced Microcontroller Bus Architecture (AMBA) specification. AXI is suited for designs that requires high bandwidth and low latency. Zedboard include AXI version 4. There are three types of AXI4 interfaces [33]:

1. **AXI4** for memory-mapped interfaces and allows high throughput bursts of up to 256 data transfer cycles with just a single address phase. AXI4 interfaces consist of five different channels, read address channel, write address channel, read data channel, write data

channel and write response channel. Data can move in both directions between the master and slave simultaneously, and data transfer sizes can vary.

2. **AXI4-Lite** is a light-weight, single transaction memory-mapped interface. It has a small logic footprint and is a simple interface with no burst, basically for control signals and configuration.

3. **AXI4-Stream** for high-speed streaming data. Removes the requirement for an address phase altogether and allows unlimited data burst size. AXI4-Stream interface transfers do not have address phases and are therefore not considered to be memory-mapped.

A common approach is to build systems that combine AXI4-Stream and AXI memory-mapped IP together. Often a DMA engine can be used to move streams in and out of memory.

III.4 Implementation of obstacle detection system

In this section, the implementation of the obstacle detection system presented in previous chapter is detailed. As explained before, for debugging purposes, it is necessary to display our results, thus we start our implementation by building HDMI display controller using Vivado 2018.2 IDE. Then we implemented the obstacle detection system using SDK.

III.4.1 HDMI controller implementation

To display the results, we implemented an HDMI display controller using several IP cores provided by Xilinx. The AXI 'Video Direct Memory Access' (VDMA) IP was used to fetch frame buffer from the Zynq's external memory DDR via one of the High-Performance Ports (HP0), and produce video content in the AXI4-Stream protocol. The data streaming was controlled by 'AXIS to video out' and 'video timing controller' (VTC) IP based on needed resolution. The HDMI interface is connected to PL via ADV7511 transmitter, which is configured using IIC protocol. Before displaying, data must be adjusted for the ADV7511 transmitter input. The global block diagram of the HDMI controller is shown in Figure III.9. The implementation is divided into hardware and software configuration.

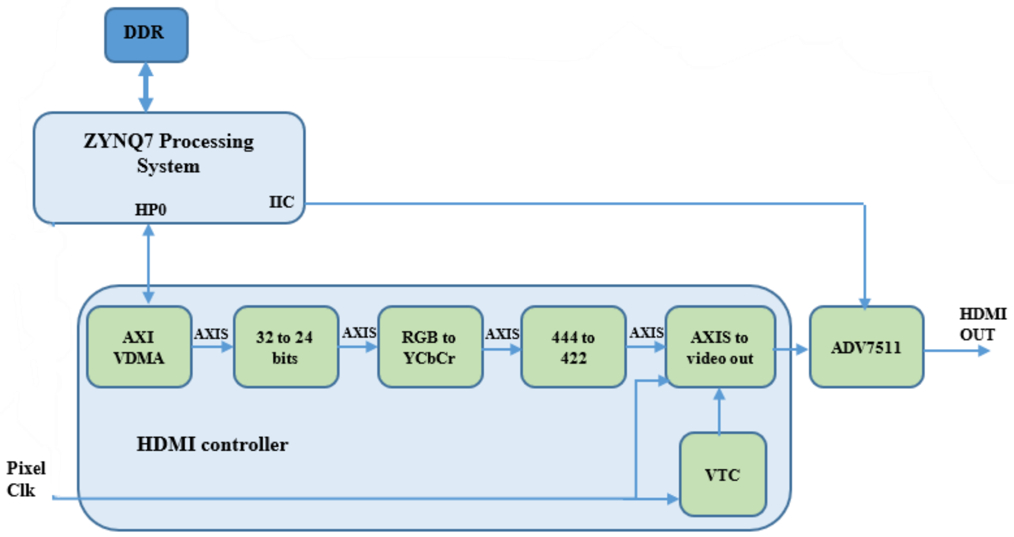


Figure III.9: Global block diagram of the HDMI Controller.

III.4.1.1 Hardware configuration

For hardware implementation and configuration, we used several LogiCORE Intellectual Property (IPs) provided by Xilinx, the first IP that we added to our block design is the ‘ZYNQ7 Processing System’ IP shown in Figure III.10, this IP is the software interface around the Zynq PS, which acts as a logic connection between the PS and the PL. Moreover, this IP allows us to setup the PS interfaces for external data communication such as the DDR and IIC controllers, GPIO, the AXI high performance (HP) and the 4 PL clock available.

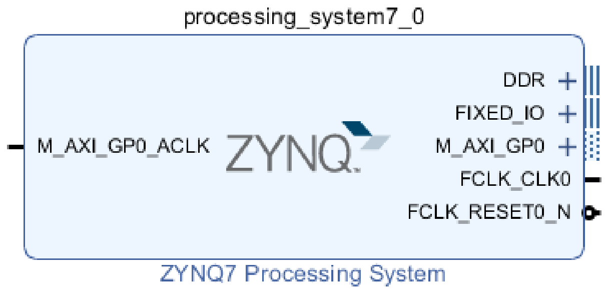


Figure III.10: Default Zynq SoC PS IP Block.

We notice that by default, the I2C and HP controllers are not connected to the PS, hence, we re-customized the IP as shown in Figure III.11 to add the HP slave AXI interface and the IIC controller. The PS IP is customized as illustrated in Figure III.12.

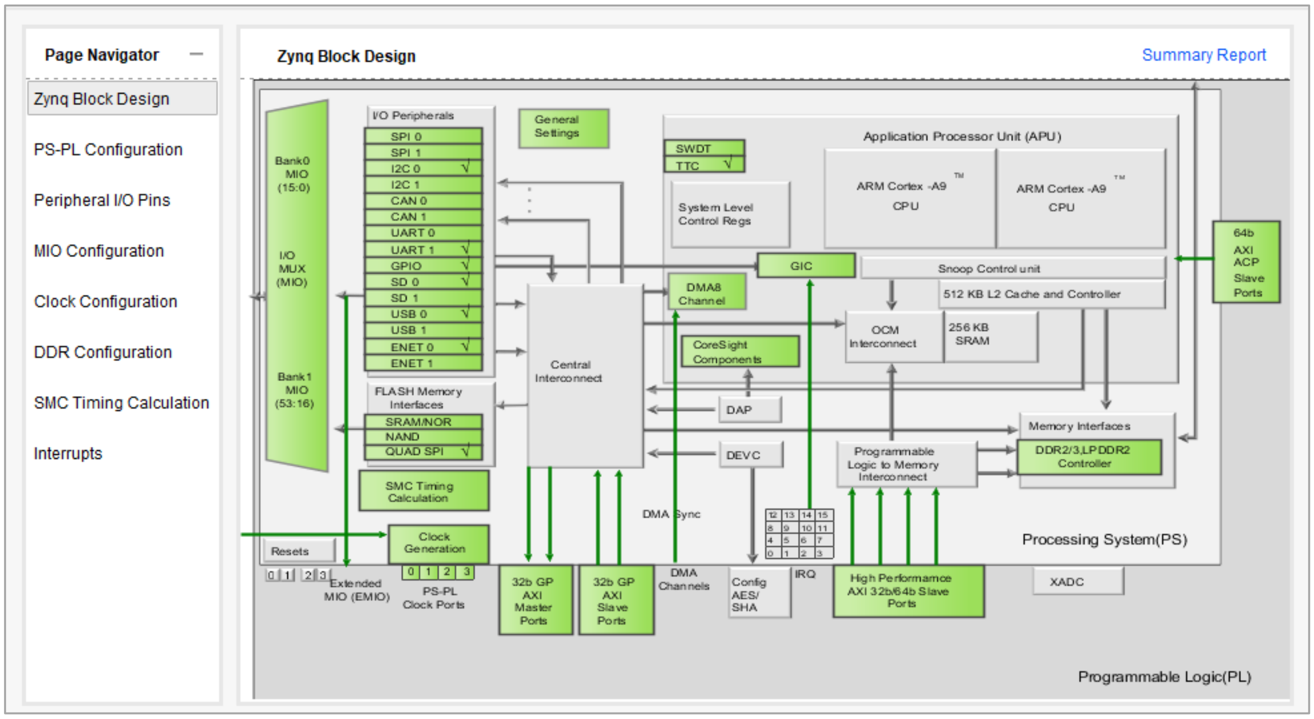


Figure III.11: Re-customize IP Dialog Box.

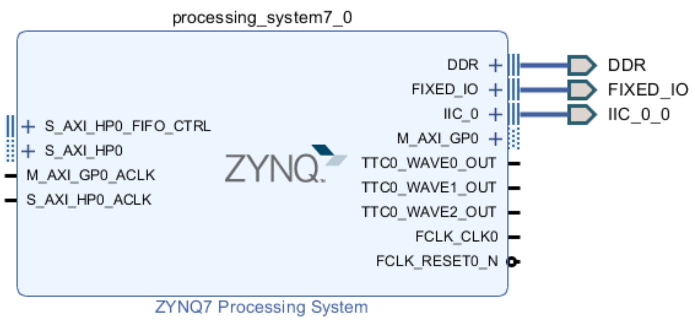
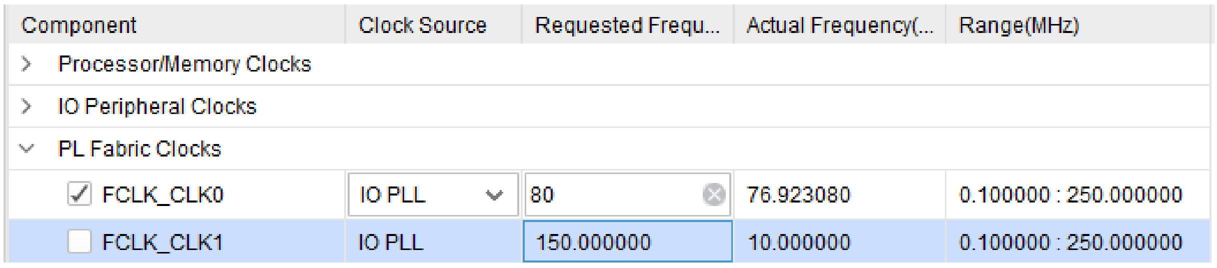


Figure III.12: Customized Zynq SoC PS IP Block.

- The HP interface provides high-bandwidth PL slave interface to the DDR3 memory, it contains control and data FIFOs to provide buffering of transactions for larger sets of bursts, resulting in higher minimum latency than other interfaces, which is ideal for workloads such as video frame buffering in DDR.
- The PS IIC_0 interface we used is directly connected to all I2C interfaces in PL, including the ADV7511 transmitter, that way no need for pin assignment.

We set FCLK_clk0 to generate 80 MHz as shown in Figure III.13.



Component	Clock Source	Requested Frequ...	Actual Frequency(...	Range(MHz)
› Processor/Memory Clocks				
› IO Peripheral Clocks				
⌄ PL Fabric Clocks				
✓ FCLK_CLK0	IO PLL	80	76.923080	0.100000 : 250.000000
☐ FCLK_CLK1	IO PLL	150.000000	10.000000	0.100000 : 250.000000

Figure III.13: Configuration of FCLK_clk0.

To read the frame buffers from DDR, we used the 'AXI VDMA IP' shown in Figure III.14, this core is designed to allow an efficient high-bandwidth access between memory and AXI4-Stream interfaces. It provides independent asynchronous read and write channel operation: In the Write path, the AXI VDMA accepts frames on the AXI4-Stream Slave interface (S_AXIS_S2MM) and writes it to memory using the AXI4 Master interface (M_AXI_S2MM). In the Read path, the AXI VDMA uses the AXI4 Master interface (M_AXI_MM2S) to read frame's pixels from memory and output the data on the AXI4-Stream Master interface (M_AXIS_MM2S). It supports the primary AXI4 data bus width of 32, 64, 128, 256, 512, and 1,024 bits. addressing up to 32 frame buffers.

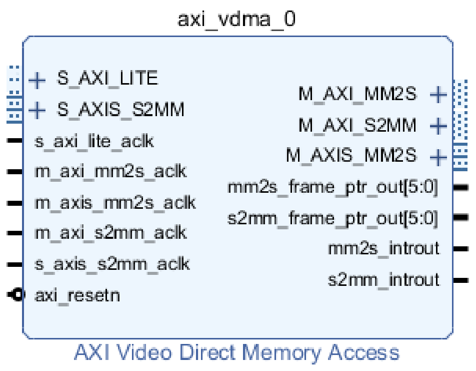
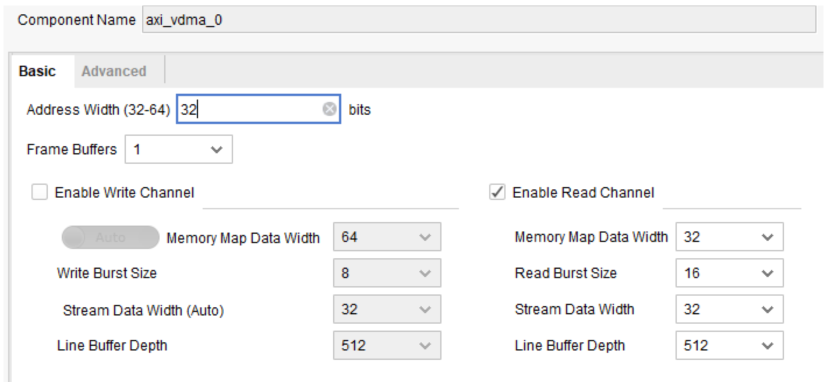


Figure III.14: AXI VDMA IP core.

To allow the AXI VDMA to transfer data from DDR, we enabled the read channel, set the number of frames to one, and kept other parameters in default, as shown in Figure III.15.



Component Name: axi_vdma_0

Basic | Advanced

Address Width (32-64): 32 bits

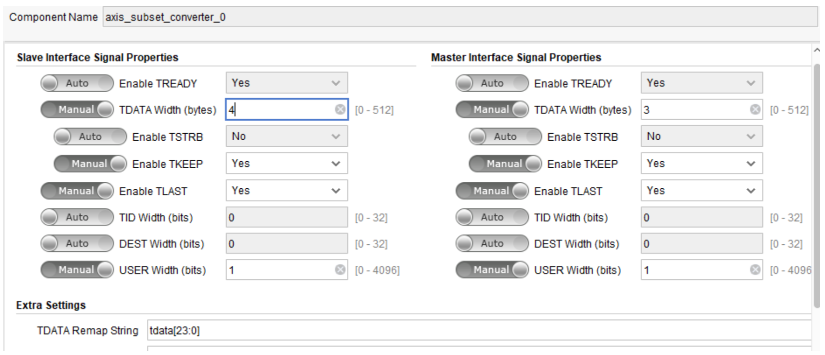
Frame Buffers: 1

☐ Enable Write Channel		✓ Enable Read Channel	
Auto Memory Map Data Width	64	Memory Map Data Width	32
Write Burst Size	8	Read Burst Size	16
Stream Data Width (Auto)	32	Stream Data Width	32
Line Buffer Depth	512	Line Buffer Depth	512

Figure III.15: AXI VDMA configuration.

The VDMA minimum data width is 32 bits, hence, the data in DDR memory must be saved as 32 bits. Pixels are saved as 32 bits grayscale format, while the ADV7511 input is 16 bits YCbCr 4:2:2, to solve the mismatching problem we used 3 IPs:

1. **AXI4 Stream subset converter** provides a solution for connecting slightly incompatible AXI4-Stream signal sets together. The IP has configurable AXI4-Stream signals for each interface that allows one to convert one signal set to another in consistent manner. To configure this IP, we set the input to 4 bytes (32 bits of the VDMA output) and output to 3 bytes (24 bits RGB format) this is done by eliminating the upper byte of the input as shown in Figure III.16



Component Name: axis_subset_converter_0

Slave Interface Signal Properties				Master Interface Signal Properties			
Auto	Enable TREADY	Yes		Auto	Enable TREADY	Yes	
Manual	TDATA Width (bytes)	4	[0 - 512]	Manual	TDATA Width (bytes)	3	[0 - 512]
Auto	Enable TSTRB	No		Auto	Enable TSTRB	No	
Manual	Enable TKEEP	Yes		Manual	Enable TKEEP	Yes	
Manual	Enable TLAST	Yes		Manual	Enable TLAST	Yes	
Auto	TID Width (bits)	0	[0 - 32]	Auto	TID Width (bits)	0	[0 - 32]
Auto	DEST Width (bits)	0	[0 - 32]	Auto	DEST Width (bits)	0	[0 - 32]
Manual	USER Width (bits)	1	[0 - 4096]	Manual	USER Width (bits)	1	[0 - 4096

Extra Settings

TDATA Remap String: tdata[23:0]

Figure III.16: AXI4 Stream subset converter configuration.

2. **RGB to YCrCb color-space converter** converts the RGB format to YCbCr 4:4:4 format. To configure this IP, we set the video component width to 8 bits and the input frame dimensions of the 720p which is 720×1280.
3. **Chroma Resampler** converts the YCbCr 4:4:4 to YCbCr 4:2:2 mode color. For configuration, we set the video component width to 8 bits and the input frame dimensions of the 720p to 720×1280.

We connected the three IPs in cascade, as shown in Figure III.17.

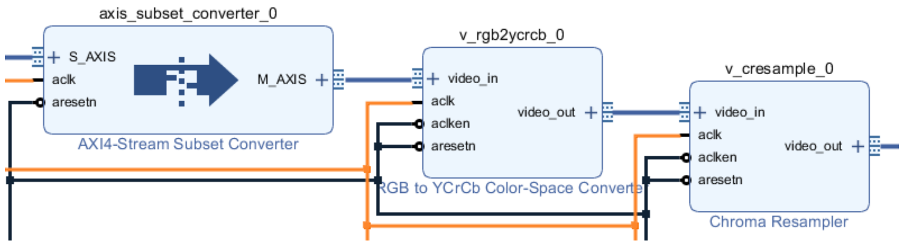


Figure III.17: 32 bits RGB to 16 bits YCbCr 4:2:2 converter design.

To generate a parallel video out of a stream of pixels, we used the ‘AXI4 Stream to video out’ IP, the core includes a FIFO allowing the AXIS for Video interface to run on a different clock. To configure this IP, we set the video format to YCbCr 4:2:2, video input and output component width to 8 bits and we enabled the independent clock mode, so that the input and display run at different rate, the input video data run at the same frequency as the IPs in Figure III.17, while the output runs at the ‘pixel_clk’ frequency to get the wanted resolution.

To control the stream of the pixels at the output, we use the ’Video Timing Controller’ (VTC) IP to provide the timing signals, based on the needed output resolution. We configured the VTC to generate 720p screen resolution, which transmits 720×1280 pixels per frame, with a 74.25 MHz pixel clock. The pixel clock was generated using the ‘clocking wizard’ IP which produces more specific frequencies compared to the available PL clocks. The chroma Resampler, VTC, and AXI4 Stream to video out IPs are connected as shown Figure III.18. We connected the 74.25MHz clock to VTC clk and to the video out clk.

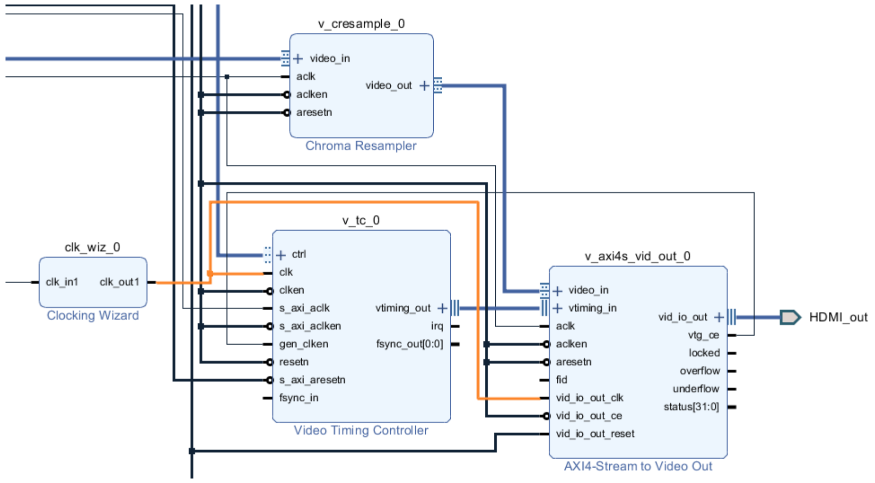
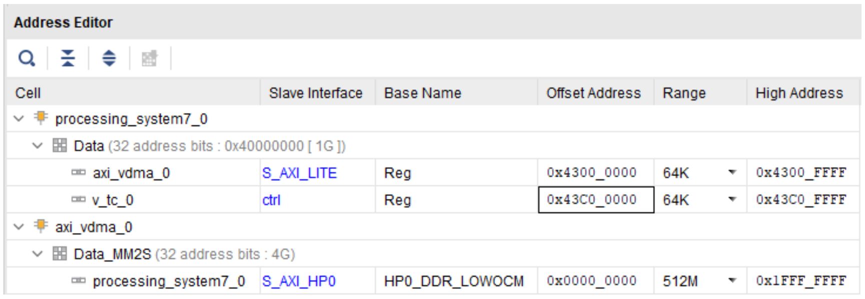


Figure III.18: Video output connection.

In our implementation, the PS and the VDMA IP are masters, they initiate data transaction with other slaves IPs. The PS controls the VDMA and VTC slaves, while the VDMA controls data transaction with the HP slave. Each master is connected to its slaves through an AXI interconnect. For the master to be able to exchange data with specific slave, slaves are memory mapped as shown in Figure III.19.



Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_vdma_0	S_AXI_LITE	Reg	0x4300_0000	64K	0x4300_FFFF
v_tc_0	ctrl	Reg	0x43C0_0000	64K	0x43C0_FFFF
axi_vdma_0					
Data_MM2S (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF

Figure III.19: Memory mapping of AXI slaves.

III.4.1.2 Software configuration

For the software obstacle detection system, we used the SDK 2018.2 tool. We started the implementation by creating a new application project with the following specifications:

- C programming language.
- Processor ps7_cortexa9_0, one of the dual cores to create a single thread application.
- Single thread application does not require any OS functionality such as memory management. Thus, we chose the standalone as OS platform, which is a simple low-level software layer. It provides access to basic processor feature.
- Create a new Board Support Package (BSP) that contains a collection of libraries, and drivers that allow software applications to access features on the hardware.

The HDMI controller configuration was encapsulated in a new library called 'hdmi_controller'. Our library consists of three functions, `hdmi_controller_init(HDMI_controller* HDMI)` that contains the I2C, VTC, VDMA and ADV7511 transmitter configuration, with HDMI is a '`HDMI_controller`' instance. The `hdmi_display_clear(HDMI_controller* HDMI)` function is used to clear the frame buffers to create a background, before saving our results. The `hdmi_display (HDMI_controller*HDMI, u16 V_location, u16 H_location, image_dimension image, u8 state)` function is used to save the left and right at (V_location, H_location) in frame coordinates. We defined 2 needed structures:

```
typedef struct
{
Xuint32 uDeviceId_VTC_HdmioGenerator;// VTC DeviceID
Xuint32 uDeviceId_VDMA_HdmiDisplay;// VDMA DeviceID
Xuint32 uBaseAddr_MEM_HdmiDisplay;//starting address of the frame buffer in DDR
Xuint32 uNumFrames_HdmiDisplay;// number of frames used by VDMA
XIicPs hdmi_out_iic;// i2c instance driver
XVtc vtc_hdmio_generator;// vtc instance driver
XAxiVdma vdma_hdmi; // vdma instance driver
XAxiVdma_DmaSetup vdmacfg_hdmi_read;// structure contains all the necessary
                                       information to start a frame read
Xuint32 hdmio_width;// width of the output frame
Xuint32 hdmio_height;// height of the output frame
} HDMI_controller;

typedef struct
{
      Xuint16 width;// width of the image
      Xuint16 hight;// height of the image
}image_dimension;
```

The DevicesIDs information are found in the ‘xparameters.h’ file which is automatically generated for the design by Xilinx. Instance drivers are used to configure the IPs, all Xilinx drivers have a structure which holds the various setup needed by the peripheral.

In the main function, we declared a global variable HDMI of type ‘`HDMI_controller`’, then we initialized the VTC and VDMA DeviceIDs, starting address of the frame buffer, the number of frames to 1 and we set width and height of the frame to 1280 and 720 respectively. To start the configuration of the hardware, we passe by reference the variable HDMI to ‘hdmi_controller_init’ function. The configuration was performed in the following order:

1. **IIC configuration** was performed using a set of functions defined in ‘XIicPs’ library:

```
XIicPs_LookupConfig(u16DeviceId);

XIicPs_CfgInitialize(XIicPs *InstancePtr, XIicPs_Config *ConfigPtr,u32EffectiveAdr);

XIicPs_SetSClk(XIicPs*InstancePtr, u32 FsclHz);
```

2. **ADV7511 transmitter configuration** was performed with the following function:

```
s32 XIicPs_MasterSendPolled(XIicPs *InstancePtr,u8 *MsgPtr, s32 ByteCount,u16 SlaveAddr);
```

   This function sends data to the FIFO and waits for the slave to pick them up, if the slave fails to remove data from FIFO, the send fails with time out. MsgPtr is the pointer to the send buffer, ByteCount is the number of bytes in the send buffer. SlaveAddr is the address of the slave we are sending to. MsgPtr is a pointer to a ‘`u8 Data[2]`’, Data[0] contains the slave register address and Data[1] contains the data (control word). We set ByteCount to send 2 bytes and SlaveAddr of the ADV7511 is (0x39),

3. **VTC configuration** to configure the VTC, first, we declared three configurations struct XVtc_Signal Signal, XVtc_Polarity Polarity and XVtc_SourceSelect SourceSelect. Signal contains the timing signal parameters; Polarity contains the polarity of the signal; SourceSelec contains the different functionalities of the VTC. These struct are defined in ‘xvtc.h’ file, we set all parameters to generate a 720p video format, then we invoked the following functions:

```
XVtc_LookupConfig(u16 DeviceId);

XVtc_CfgInitialize(XVtc *InstancePtr, XVtc_Config *CfgPtr, UINTPTR EffectiveAddr);

XVtc_SetPolarity(XVtc *InstancePtr, XVtc_Polarity *PolarityPtr);

XVtc_SetGenerator(XVtc *InstancePtr, XVtc_Signal *SignalCfgPtr);

XVtc_SetSource(XVtc *InstancePtr, XVtc_SourceSelect *SourcePtr);
```

4. **VDMA configuration** to configure the VDMA, we use the following function defined in 'xaxivdma' library:

```
XAxiVdma_LookupConfig(u16 DeviceId);

XAxiVdma_CfgInitialize(XAxiVdma*InstancePtr, XAxiVdma_Config*CfgPtr,UINTPTR EffectiveAddr);

XAxiVdma_DmaConfig(XAxiVdma *InstancePtr, u16 Direction, XAxiVdma_DmaSetup *DmaConfigPtr);

XAxiVdma_DmaSetBufferAddr(XAxiVdma *InstancePtr, u16 Direction, UINTPTR *BufferAddrSet);

XAxiVdma_DmaStart(XAxiVdma *InstancePtr, u16 Direction);

XAxiVdma_DmaStop(XAxiVdma *InstancePtr, u16 Direction);
```

III.4.1.3 HDMI results

To test the HDMI controller, we displayed colored square blocks in different regions, this is done by writing the hex color code into specific addresses in the DDR frame buffer. The addresses can be calculated since the pixels are stored in the frame buffers pixel by pixel in a row starting from the base address of the frame buffer `(HDMI->uBaseAddr_MEM_HdmiDisplay)`. For a frame with a width of `(HDMI->hdmio_width)`, we can conclude that the n_{th} row of the frame will be stored at address (destination_storage) such that:

```
Distination_storage= (HDMI->uBaseAddr_MEM_HdmiDisplay)+(HDMI->hdmio_width)*n+1
```

And each pixel *P* is stored at address:

```
Distination_storage= (HDMI->uBaseAddr_MEM_HdmiDisplay)+(HDMI->hdmio_width)*n+P+1
```

Writing the RGB hex code to the correct address using the **`hdmi_display()`** function, but first we called the **`hdmi_display_clear()`** to create a black background, here we called the function 5 times to display blocks at different region, and we got the result in Figure III.20.

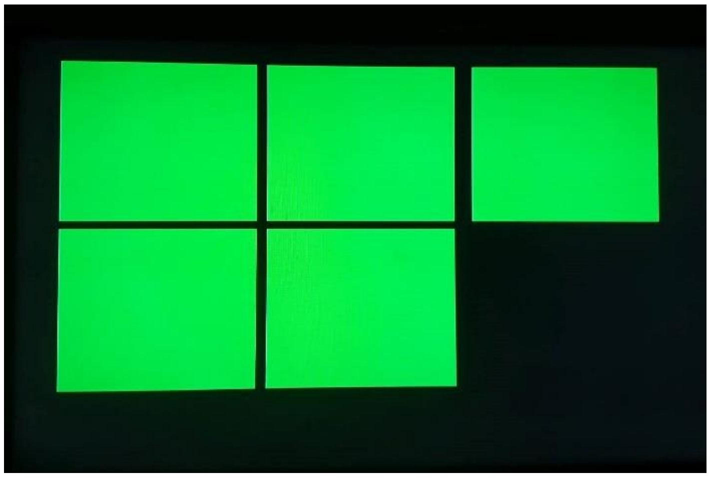


Figure III.20: HDMI controller test result.

III.4.2 Software implemenation of obstacle detection system

Based on the global obstacle detection system proposed in the previous chapter, we emulated the camera using an SD card to read the stereo images pixel by pixel. To read files from the SD card, we used a generic Fat Fill system library provided by Xilinx, this library consists of a set of functions to read from and write to files in the SD card and many others. For our implementation, we used a stereo image (Tsukuba) of size 288×384 pixels, shown in Figure III.21, from the Middlebury Stereo Vision data base [34], we generated binary file (.BIN) for each grayscale image using MATLAB, image pixels were saved as 32 bits, then we loaded the generated files to the SD card. To read the content of the SD card, we invoked the `f_read()` function defined as:

```
FRESULT f_read (FIL* fp, void* buff, UINT btr, UINT* br);
```

This function takes as parameters: a file instance `fp`, the base address for transferring the data `buff`, the number of bytes to read `btr` and the number of bytes has been read `br` at the end of the transfer. Once the data is placed in the DDR memory, several algorithms are applied as explained in the previous chapter. The implementation of each algorithm is detailed in the following subsections.

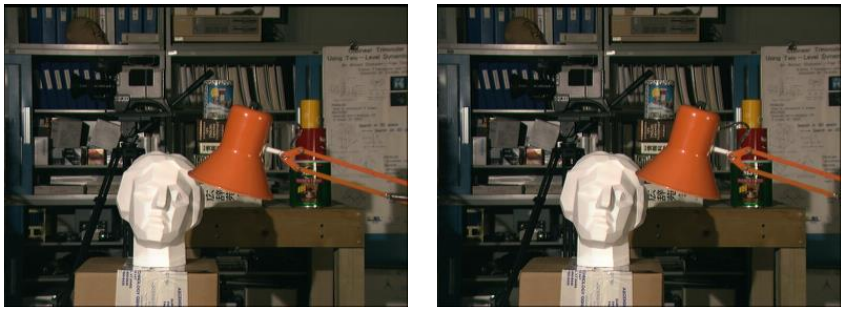

(a) (b)

Figure III.21: Tsukuba stereo images from the Middlebury StereoVision data base,(a) right image, (b) left image *[34]*.

III.4.2.1 Rank transform

For Rank transform implementation, we created a **`RANK_transform()`** function based on the algorithm introduced in section II.3.1. we defined the function as:

```
void RANK_transform (HDMI_controller* HDMI, u8 window, u16 V_location, u16 H_location);
```

This function takes as parameters: the `HDMI` instance to get the starting address of the frame buffer, the window width `window`, and the `V_location` and `H_location` that represent the vertical and horizontal location of the image in the frame coordinates. Any pixel of the image is saved at a specific location `Source_storage` such that:

```
Source_storage= (HDMI->uBaseAddr_MEM_HdmiDisplay)+(HDMI->hdmio_width)*row+column+1
```

The variable ‘row’ is used to cover all rows of the image starting from `V_location`, while the variable ‘column’ is used to cover all columns of the image starting from `H_location`. This variable is declared as a pointer to access pixels in the frame buffer instead of saving the images in a 2D array, we saved the Rank transform images in their selected regions shown in Figure III.22.

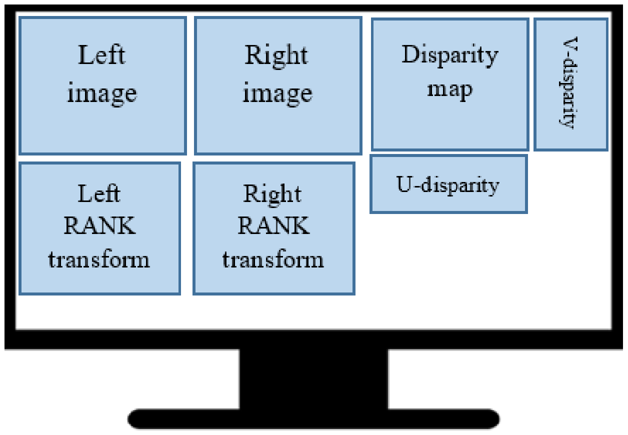


Figure III.22: Selected regions in frame buffer to display results.

III.4.2.2 SAD correlation

For SAD correlation implementation, we created a **`Sum_Absolute_Diff()`** function based on the on the algorithm introduced in section II.3.2, we defined the function as follow:

```
void Sum_Absolute_Diff (HDMI_controller* HDMI, u8 window, u16 V_locationL, u16 H_locationL,
                                                 u16 V_locationR, u16 H_locationR, int dmax);
```

This function takes as parameters: the `HDMI` instance to get the starting address of the frame buffer, the window width `window,` and the `V_locationL, H_locationL, V_locationL` and `H_locationL` that represent the vertical and horizontal location of the left and the right rank transform images in the frame coordinates and disparity range `dmax.` We declared two pointers, `StorageMem_Right` and `StorageMem_Left` to access the right and left rank transform images generated using the previous function, we selected the right as a reference image then we saved the window to be matched in a template array. We saved the disparity values into the selected region as shown in Figure III.22. The Rank transform and the SAD correlation functions represent the stereo vision module, we encapsulated the two functions in one library named `Stereo_Vision.`

III.4.2.3 U-V DM generation

To generate the U-V DM, we created a function for each map, **`V_disparity()`** function to generate the VDM and **`U_disparity()`** function to generate UDM, the functions are:

```
 void V_disparity(HDMI_controller *HDMI,u8 window,u16 V_location,u16 H_location);
 void U_disparity(HDMI_controller *HDMI,u8 window,u16 V_location,u16 H_location);
```

The two functions take as parameters the `HDMI` instance, the window size used to calculate the size the DM, the `V_location` and `H_location` represent the location of the DM in the frame buffer. We applied scaling on the generated maps, cause for small disparity range dmax, the UDM and VDM will have a narrow height and width respectively which will be hard to analyze the horizontal and vertical lines. For scaling, each column and row in the original VDM and UDM respectively is displayed as 4 columns or 4 rows, the difference is shown in Figure III.23. In the **`U_disparity()`**, we also performed the first filtering stage to remove the noise caused by the ground or any neglected or small objects. For this, for each row of the UDM we saved the maximum intensity in a row into a global 1D array, which corresponds to biggest object at that distance. Based on these values, we eliminated the intensities less than 20% of the maximum value at a given row. The obtained results are shown in Figure III.24. For the VDM, we saved the maximum intensity in the column into a global 1D array to be used in vertical line extraction function.

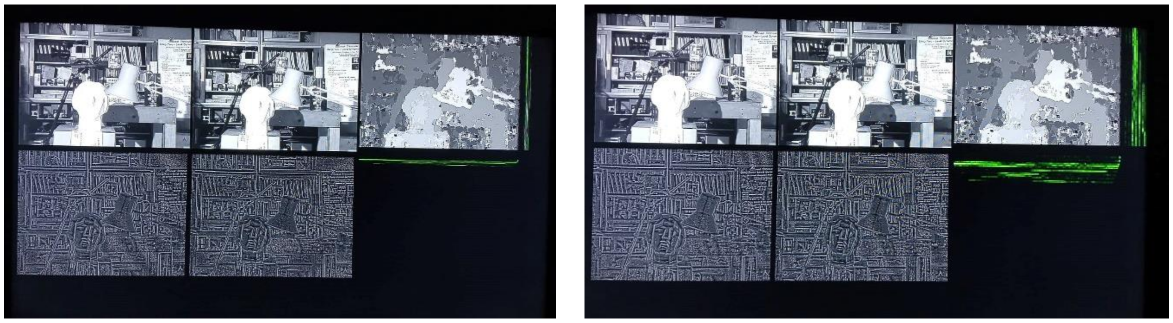

(a) (b)

Figure III.23: HDMI display (a) U-V DM without scaling, (b) U-V DM with scaling.

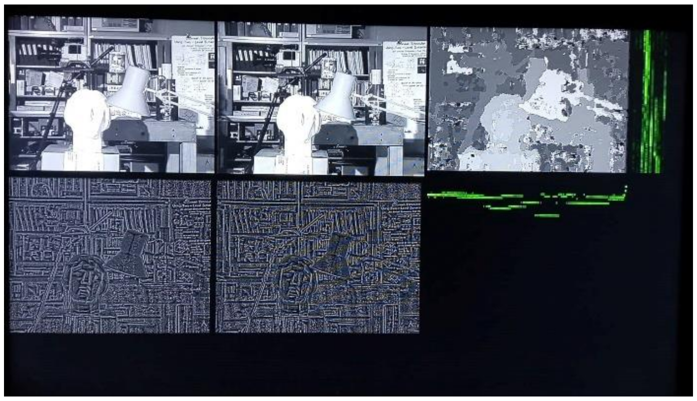


Figure III.24: HDMI display using noise removal for the UDM.

III.4.2.4 Vertical lines extraction

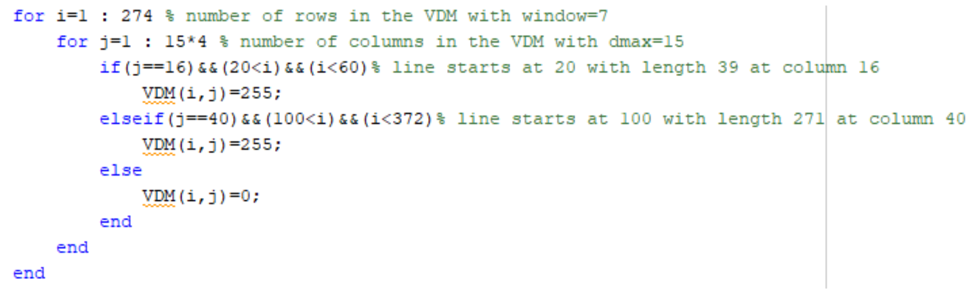
For the vertical lines extraction, we created **vertical_ROI()** function based on the algorithm introduced in section II.3.4, we defined the function as follow:

```
void vertical_ROI(HDMI_controller* HDMI, int dmax,u16 V_location,u16 H_location);
```

The parameters `V_location` and `H_location` represent the location of the VDM. As explained in section II.3.4, a line is defined by its starting point and the length, thus, we created a struct named `line`, the struct is defined as follows:

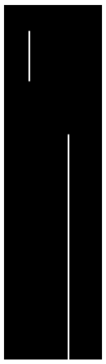
```
typedef struct line {
        int start;
        int length;
  };
```

We saved the extracted lines in a global 2D array that has the same size as the VDM to save all possible lines. The function includes two filtering stages, one is based on the pixel intensity to remove noise and another based on the length of the extracted line to remove the small objects. To mark the start of a line, the intensity of the VDM pixel must be greater than a certain threshold. The threshold depends on the maximum intensity in the column. The second filtering is based on the length of the line, lines with length less than V_threshold are removed and considered as invalid lines, we set V_threshold to 10% of the height of the VDM. To validate our algorithm and ensure that it extracted the right lines, with the correct length and starting point we used Matlab to construct a test VDM sample that consist of two lines with a predefined starting point, length and location as shown Figure III.25. We uploaded the BIN file of the sample into the SD card and we saved it at the appropriate region in the frame buffer, the extracted lines were saved in an array Vline as shown in Figure III.26, we can see that the extracted lines are exactly the generated ones in Matlab.



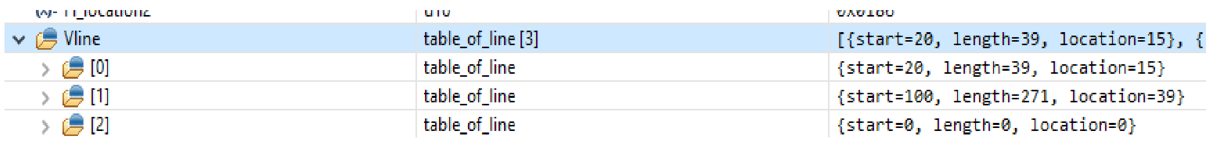
```
for i=1 : 274 % number of rows in the VDM with window=7
    for j=1 : 15*4 % number of columns in the VDM with dmax=15
        if(j==16)&&(20<i)&&(i<60)% line starts at 20 with length 39 at column 16
            VDM(i,j)=255;
        elseif(j==40)&&(100<i)&&(i<372)% line starts at 100 with length 271 at column 40
            VDM(i,j)=255;
        else
            VDM(i,j)=0;
        end
    end
end
```

(a)



(b)

Figure III.25: (a) Matlab code to reconstruct test VDM sample with 2 lines, (b) the VDM sample.



Vline	table_of_line [3]	[{start=20, length=39, location=15}, {
[0]	table_of_line	{start=20, length=39, location=15}
[1]	table_of_line	{start=100, length=271, location=39}
[2]	table_of_line	{start=0, length=0, location=0}

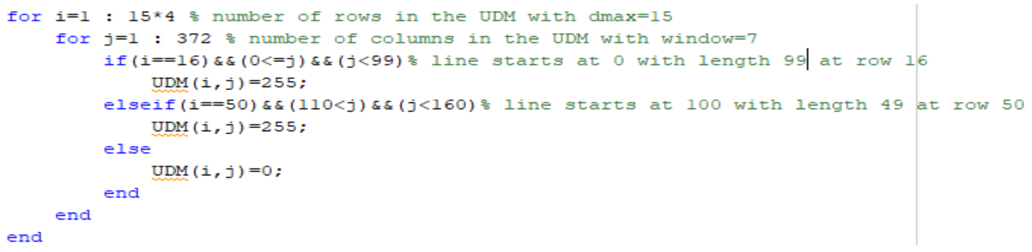
Figure III.26: Vertical lines extracted from the VDM sample.

III.4.2.5 Horizontal lines extraction

For the horizontal lines extraction, we created **`Horizontal_ROI()`** function based on the algorithm introduced in section II.3.4, we defined the function as follow:

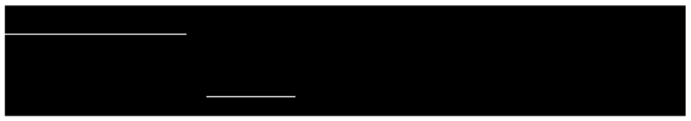
```
void horizontal_ROI(HDMI_controller* HDMI, int dmax, u16 V_location, u16 H_location);
```

The parameters `V_location` and `H_location` represent the location of the UDM. Same as the vertical lines, we used the created struct line to define the horizontal lines, we saved lines extracted in a global 2D array that have size as the UDM. To validate our algorithm, we created a UDM sample using Matlab as shown in Figure III.27. The extracted lines are shown in Figure III.28, we can see that the algorithm extracts the lines correctly.



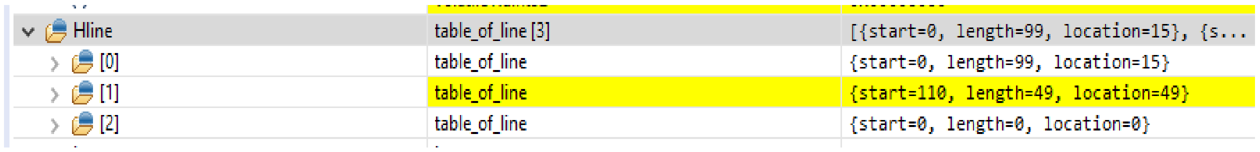
```
for i=1 : 15*4 % number of rows in the UDM with dmax=15
    for j=1 : 372 % number of columns in the UDM with window=7
        if(i==16)&&(0<=j)&&(j<99)% line starts at 0 with length 99 at row 16
            UDM(i,j)=255;
        elseif(i==50)&&(110<j)&&(j<160)% line starts at 100 with length 49 at row 50
            UDM(i,j)=255;
        else
            UDM(i,j)=0;
        end
    end
end
```

(a)



(b)

Figure III.27:(a) Matlab code to reconstruct test UDM sample with 2 lines, (b) the UDM sample.

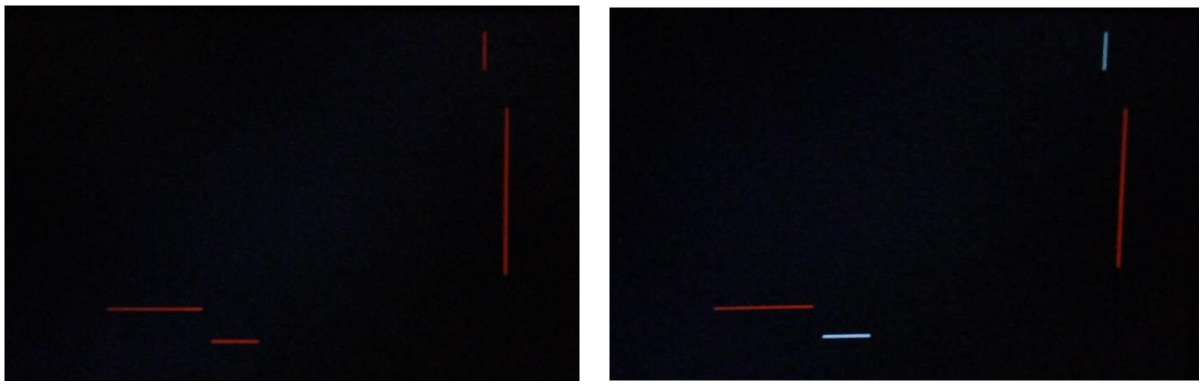


```
Hline      table_of_line [3]   [{start=0, length=99, location=15}, {s...
  [0]      table_of_line       {start=0, length=99, location=15}
  [1]      table_of_line       {start=110, length=49, location=49}
  [2]      table_of_line       {start=0, length=0, location=0}
```

Figure III.28: Horizontal lines extracted from the UDM sample.

After extracting all lines, we filtered the lines based on the lengths, to remove very small lines of length less then H_threshold, which represent objects with small width. Lines with length greater than H_threshold are considered as valid lines.

After extracting the horizontal and vertical lines, valid lines are colored in red as shown in Figure III.29, as illustrated in (a) all lines in red color are valid, since the length of horizontal and vertical lines is greater than H_threshold and V_threshold respectively. For case (b) one horizontal and vertical line is invalid (white lines), since their length is less than H_threshold and V_threshold respectively.



(a) (b)

Figure III.29: Horizontal and Vertical valid lines extracted using the algorithm, (a) H_threshold=20 V_threshold=30, (b) H_threshold=50 V_threshold=40.

III.4.2.6 Obstacle detection

As explained in section II.3.5, an obstacle is modeled by a box that covers the object, the box is represented by 4 values: H_start, H_end, V_start and V_end, thus, we created a struct `obstacle`, defined as follow:

```
typedef struct obstacle {
        int start_H;
        int end_H;
        int start_V;
        int end_V; };
```

The search and detection of the obstacle is performed by a function we named **ROI** (Region of Interest), defined as follows:

```
void ROI (HDMI_controller *HDMI, int dmax, u16 V_location2, u16 H_location2)
```

This function detects the obstacle and saves them in a table of type `obstacle`. After detecting all obstacles, the function draws a rectangular around each obstacle with different color.

III.5 Experimental results

Our system is divided into two modules: the stereo vision module and obstacle detection module. Thus, for a better analysis and evaluation, the results of each module are discussed separately.

III.5.1 Experimental results of the stereo vision module

In this section, we will analyze the stereo vision module based on the quality of the DM for different parameters (window size and disparity range). To evaluate the generated DM, we compared the estimated DM to the ground truth map, the comparison is based on an error measure known as the percentage of Bad Matched Pixels (BMP) [35]. The BMP measure is formulated as:

$$BMP = \frac{1}{N}\sum_{(x,y)} \varepsilon(x,y) \qquad (III.1)$$

Where N is the number of pixels in the DM and $\varepsilon(x, y)$ is a binary function at pixel(x, y), defined as:

$$\varepsilon(x,y) = \begin{cases} 1, & |D_t(x,y) - D_e(x,y)| > \delta \\ 0, & |D_t(x,y) - D_e(x,y)| \leq \delta \end{cases} \qquad (III.2)$$

Where D_t and D_e are the ground truth and the estimated DM respectively, δ is the error threshold, which was fixed to 1 (commonly used threshold).

Middlebury's stereo benchmark dataset is used in this section, this dataset was selected because it is widely used and known by the stereo vision research community. It is composed of four indoor stereo image pairs (the Tsukuba, the Venus, the Teddy, and the Cones) captured under controlled conditions. In our project we selected the Tsukuba and the Venus to examine our stereo vision module, which are illustrated with their associated disparity ground truth map in Figure III.30.

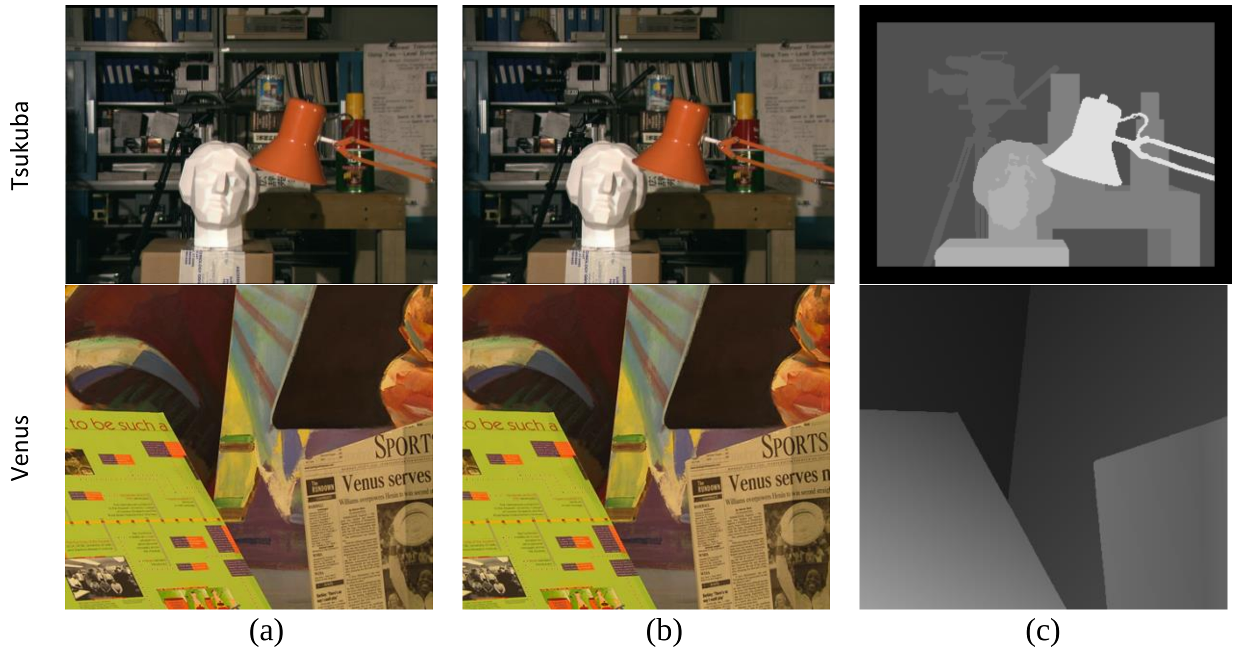


Figure III.30: Middlebury stereo vision dataset: (a) left images, (b) right images, (c) ground-truth.

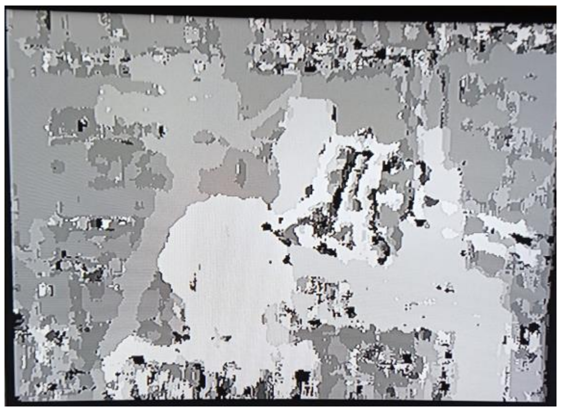
Note that the disparity ground-truth data of the Tsukuba stereo image pair excludes a border of 18 pixels, where no disparity value is provided. The size of Tsukuba images is 288×384 pixels, removing the border, the size of the DM is N=252×348 pixels, while the size of the Venus images is N=383×434 pixels. The error measurements BMP of the estimated DMs of Tsukuba and Venus images using the RANK transform and SAD correlation metric are shown in Table III.2.

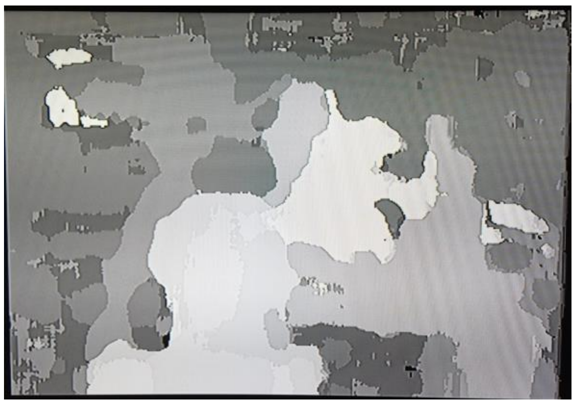
Table III.2: Evaluation results using the percentage BMP measure.

	Tsukuba				**Venus**			
Disparity range / **Window size**	**10**	**15**	**20**	**25**	**10**	**15**	**20**	**25**
7*7	22.51	20.69	21.94	22.13	36.97	23.33	23.96	24.21
9*9	19.01	16.42	17.29	17.40	35.55	20.99	21.24	21.28
11*11	17.12	14.35	14.94	15.01	35.19	20.10	20.16	20.18
13*13	15.68	12.79	13.11	13.18	34.92	19.41	19.42	19.42
15*15	14.83	11.95	12.91	12.25	34.93	19.18	19.19	19.17
17*17	15.71	12.36	12.74	12.43	34.81	19.01	19.03	19.02
19*19	16.50	13.29	14.12	14.17	35.24	19.66	19.84	19.80

From Table III.2, we can see that increasing the window size and the disparity range does not always lead to a drop in the BMP percentage and hence increase the DM quality. The Tsukuba DM reaches the minimum BMP (best DM in green) at window size 15 and disparity range 15, while the Venus reaches the best DM quality at the same disparity range and window size 17, the Venus pair requires a larger window size since it contains more texture-less surfaces. We can observe that increasing the disparity range beyond 15 does not affect the DM quality. The best and worst DM corresponding to the smallest (green) and biggest (red) BMP percentages are shown in Figure III.31. There is a non-linear relation between the DM accuracy, the window size, and the disparity range, so as the stereo vision parameters increase, the DM quality enhances to reach a maximum before it declines, therefore, selecting larger parameters would add unneeded execution cycles to the system and may have a negative impact on the DM.

Tsukuba





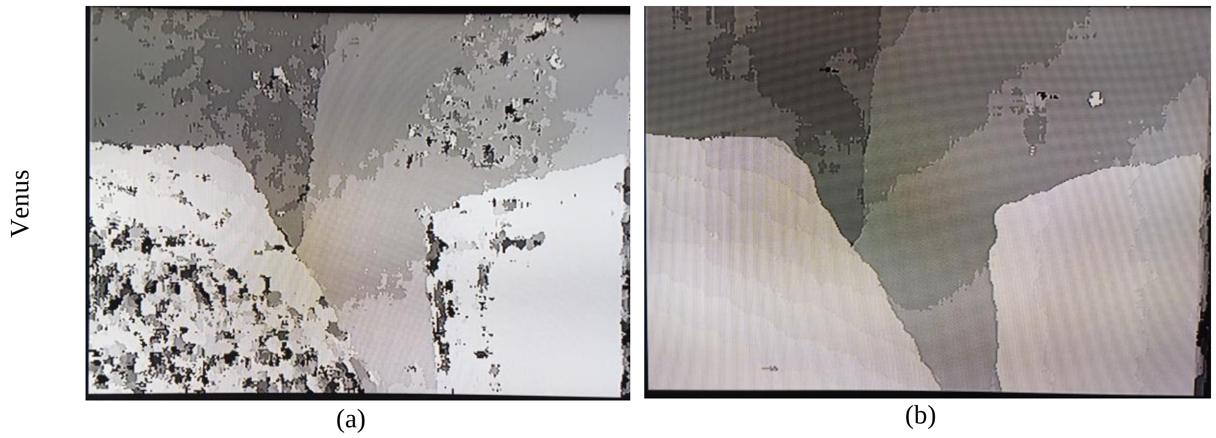


Figure III.31: Estimated DM, (a) worst DM with higher BMP percentage, (b) best DM with smaller BMP percentage.

III.5.2 Experimental results of the obstacle detection system

To analyze the obstacle detection system, we used the DrivingStereo dataset for stereo matching in autonomous driving scenarios, this dataset provides 2000 frames with different weathers (sunny, cloudy, foggy, and rainy). We selected four stereo pair images (one pair from each weather class) shown in Figure III.32.

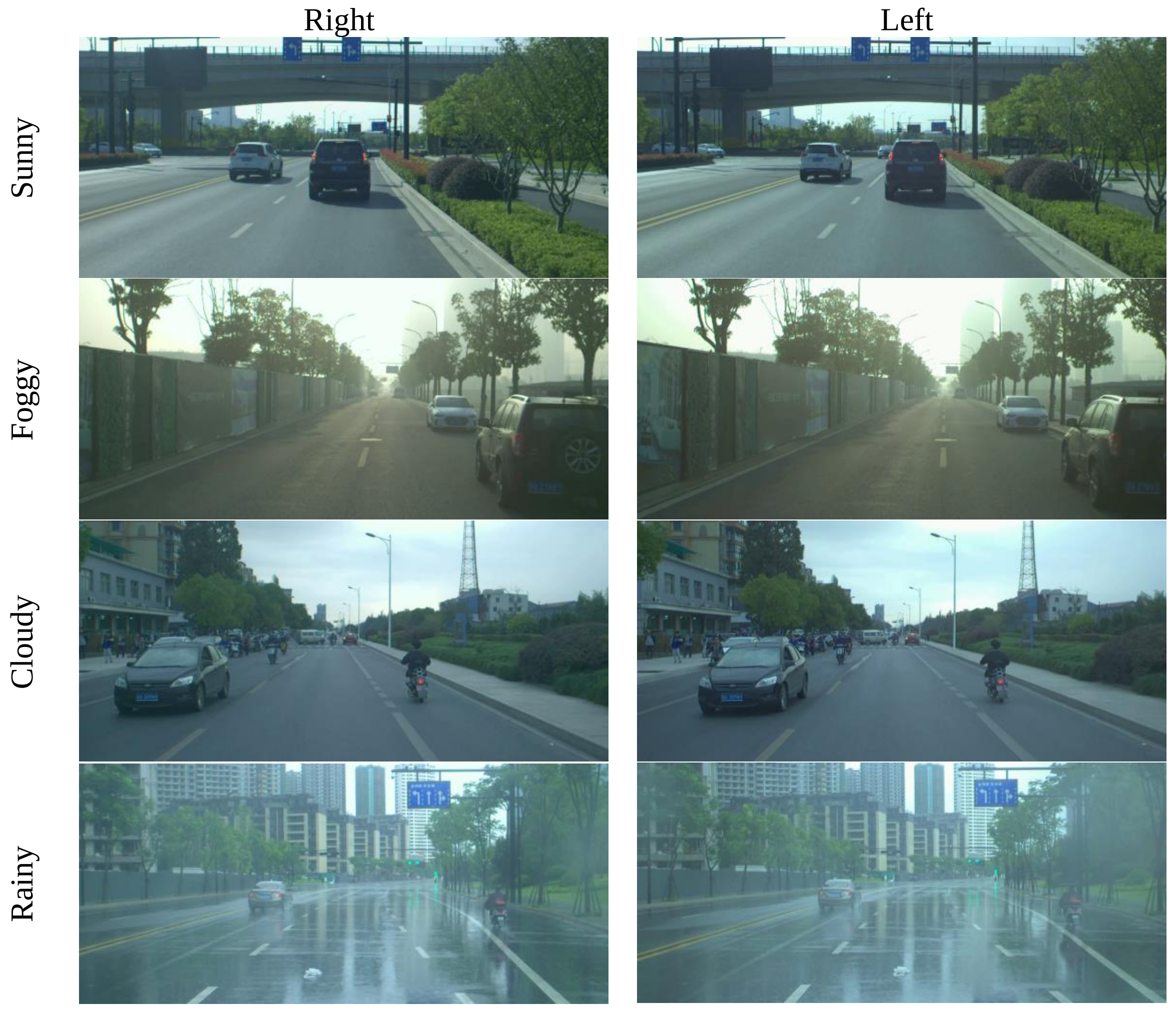


Figure III.32: Selected stereo pairs from DrivingStereo dataset.

The size of the provided images is 800×1758 pixels, to reduce the computation complexity of the obstacle detection system, we resize the images to 240×320 pixels.

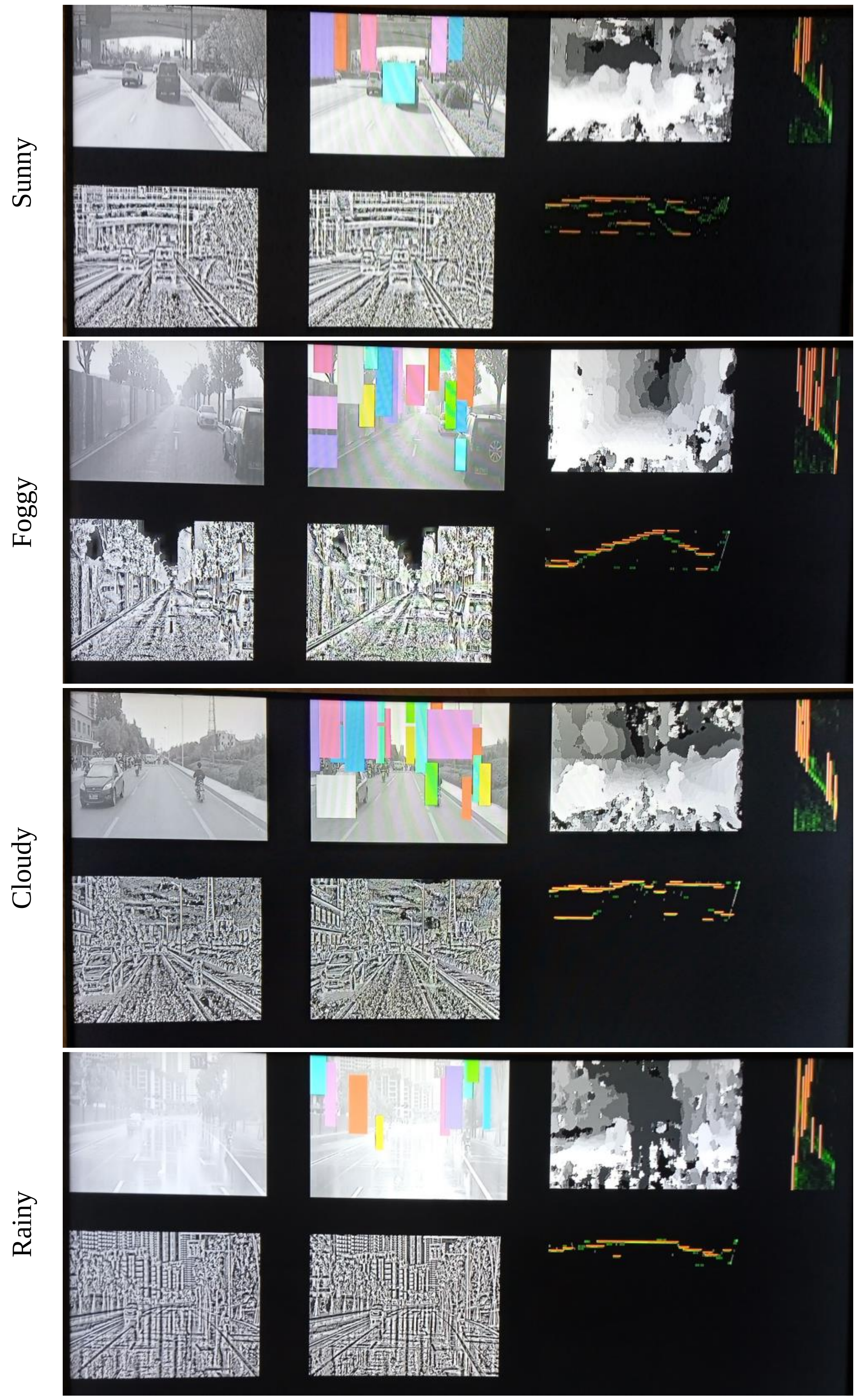


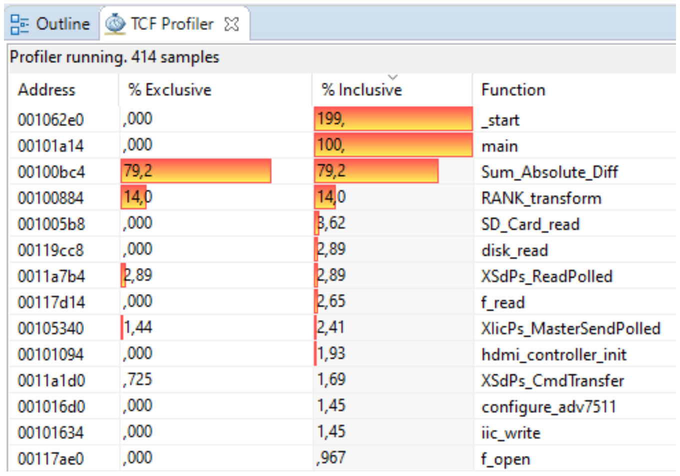
Figure III.33: Obstacle detection system results using stereo pairs from DrivingStereo dataset.

The obstacle detection system results using the U-V disparity approach in different weathers are shown in Figure III.33. We can see that this approach allows us to localize most of the obstacles on the scene despite the weather conditions that would affect the DM quality. For more evaluation, we measured the execution time of the system for different parameters. The measurements were taken after removing the unnecessary parts in our system, such as the scaling in the UDM and VDM functions, the piece of code used to color the extracted lines and obstacles. We measured the time needed to process images of 240×320 pixels with a CPU frequency of 667MHz. The execution time measurements in seconds are illustrated in Table III.3. The minimum execution time is around 3 seconds which clearly does not fulfill the real-time requirements. In our implementation, those requirements were defined by the frame rate of the stereo cameras available on the market, since our system must operate together in unison with any stereo acquisition system. Based on our research, we found that the frame rate of stereo cameras varies from 30 fps for OV7620 camera of resolution 320x240 pixels, up to 60 fps for Omnivision OV9750 of resolution 640x240 pixels.

Table III.3: Execution time of the obstacle detection system using images of size 240×320 pixels.

Disparity range / **Window size**	**10**	**15**	**20**
7*7	3.14	4.11	5.09
9*9	4.67	6.19	7.80
11*11	6.72	8.91	11.09

To determine the most time-consuming module in our system, we used the Xilinx Target Communication Framework (TCF) profiler in SDK, the results were taken for a window size 7*7 and disparity range of 10 pixels, the TCF results are shown in Figure III.34. The TCF results shows that the SAD and RANK transform functions representing the stereo vision module are the computationally intensive functions in our system, consuming 93% of the total execution time, which is around 2.925 seconds.



Profiler running. 414 samples

Address	% Exclusive	% Inclusive	Function
001062e0	,000	199,	_start
00101a14	,000	100,	main
00100bc4	79,2	79,2	Sum_Absolute_Diff
00100884	14,0	14,0	RANK_transform
001005b8	,000	3,62	SD_Card_read
00119cc8	,000	2,89	disk_read
0011a7b4	2,89	2,89	XSdPs_ReadPolled
00117d14	,000	2,65	f_read
00105340	1,44	2,41	XIicPs_MasterSendPolled
00101094	,000	1,93	hdmi_controller_init
0011a1d0	,725	1,69	XSdPs_CmdTransfer
001016d0	,000	1,45	configure_adv7511
00101634	,000	1,45	iic_write
00117ae0	,000	,967	f_open

Figure III.34: Obstacle detection system profiling.

III.6 Conclusion

In this chapter, we presented the software implementation of the obstacle detection systems on ZedBoard. We built a software system that is capable of detecting and displaying obstacles from stereo images using the U-V disparity approach. The obstacle detection system was tested using images from the DrivingStereo dataset, we tested our system in four weather conditions (sunny, cloudy, foggy, and rainy). The obstacle detection results were acceptable despite the effect of the weather on the DM. The performance of the obstacle detection system was investigated for real-time application using the TCF profiler, the processing speed of 240×320 pixels images is 1/3 fps, which reveal the inefficiency of our system for such application due to the low processing speed, caused by the sequential processing of software systems and the high computational complexity of the stereo vision module.

Chapter IV: Hardware/Software implementation of obstacle detection system

IV.1 Introduction

In recent years, the interest in hardware/software codesign has been revived. Although research in the co-development of hardware and software is not new, its revival occurred mainly due to continuous advancements and development of new technologies such as SOCs and the High-Level Synthesis (HLS) techniques. This advancement is driven by the market pressures to produce more adaptable, changeable, and more performance efficient systems. Hardware/software codesign is the concurrent design of both hardware and software of the system by taking into consideration the cost, energy, performance, speed, and other parameters of the system. During the design, trade-offs are made between the implementation of functionality in hardware and/or software depending upon both cost considerations and technical feasibility.

IV.2 Hardware/Software codesign methodology

Hardware/software codesign can help a designer to make trade-offs between the flexibility, cost, and the performance of a digital system. To achieve this, a designer needs to combine two radically different ways of design: the sequential way of decomposition in time, using software, with the parallel way of decomposition in space, using hardware. Hence, the codesign can be defined as the activity of partitioning a system into a flexible part (software) and a fixed part (hardware). Codesign involves several sub-areas as shown in Figure IV.1. The system specification sub-area is related to the design representation, modeling, and requirements. Generally, the design is initially specified at the abstract level. Successive refinements, interspersed with partitioning, bring the design down to the synthesis and compilation levels. This sub-area is also concerned with the best internal representation for the design [36].

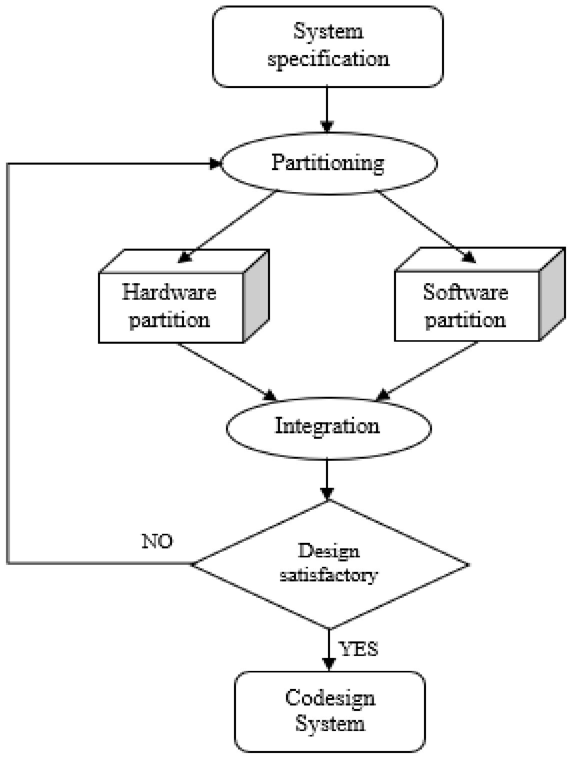


Figure IV.1: Basic flow of codesign methodology.

In hardware/software codesign, partitioning is the stage where implementation paradigm decisions, between hardware and software, are made. Partitioning has a first-level impact on system's performance because a wrong choice can result in an increased development time. A good partitioning that fits all performance requirements optimizes operation and minimizes costs usually resulting in an efficient project. To attain a good partitioning, several methods have been proposed such as evolutionary computing, artificial neural networks, and the profiling-based which is the most widely used nowadays [37]. The increasing complexity of codes raised the need for profiling tools. Profiling tools make code analysis and indicate several system features like functions execution time and memory usage. Each tool gives a specific group of information and most of them are based on clock cycle information. After obtaining profiling information, system can be modified to achieve the expected performance. Usually, the critical parts of the system (intensive processing and time-consuming functions) start being optimized by modifications and improvements, then followed by hardware development and acceleration in extreme cases. Once the hardware and software partitions have been decided, the next step is the integration of both designs, the hardware/software interfacing and scheduling are good examples of problems related to this area.

IV.3 Overview of the obstacle detection codesign system

For a real-time obstacle detection system, the speed is the main aspect, therefore, it was considered as primary parameter in the hardware/software codesign partitioning. The partitioning decision in our codesign implementation was taken based on the profiling results shown in the previous chapter, as a result, the software stereo vision module which is a computationally intensive task is replaced by a hardware stereo vision module implemented on FPGA by Yahia [38]. The module can treat up to 1000 fps of 240×320 pixels images using the same stereo matching technique (RANK transform and SAD correlation metric). The interfacing of the hardware stereo vision module and the software obstacle detection module requires transferring the disparity values generated by the stereo vision module to the DDR memory, the interfacing was a straightforward process using the stream to memory mapped (S2MM) channel available on the VDMA IP. For scheduling and synchronization of the two designs, we used an interrupt signal to trigger the processing of the obstacle detection module while keeping the stereo vision module in free-run mode (i.e., in free-run mode, disparity data are transferred as quickly as possible without waiting for any external trigger). For the camera emulation, we used a memory mapped to stream (MM2S) channel to deliver left and right pixels of the stereo pair images stored in the memory to the hardware stereo vision module. The global codesign obstacle detection system is shown in Figure IV.2.

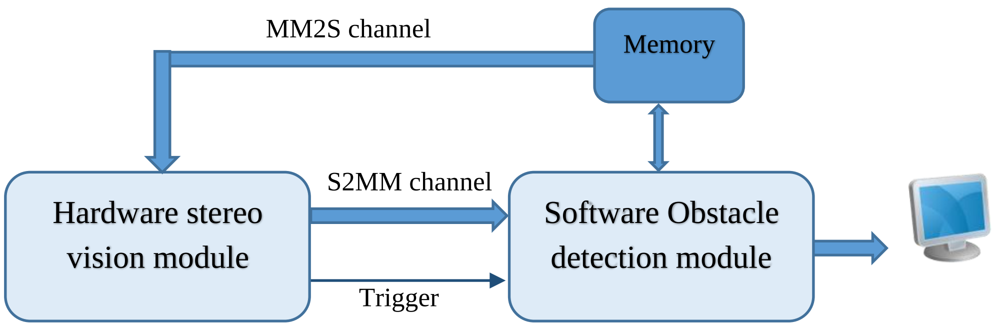

MM2S channel
Memory
S2MM channel
Hardware stereo vision module
Software Obstacle detection module
Trigger

Figure IV.2:Global codesign of obstacle detection system.

IV.4 Obstacle detection codesign implementation

IV.4.1 Hardware configuration

In order to implement the proposed codesign system, we made some modifications in the software design of the previous chapter. First, we imported the stereo vision module to our block design in Vivado, the stereo vision module block is shown in Figure IV.3, where RST input signal is used to reset the module at the end of each frame generation and the write signal represent the valid signal.

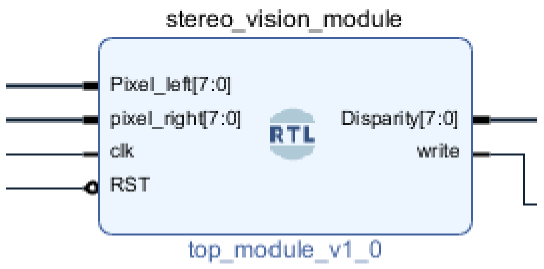
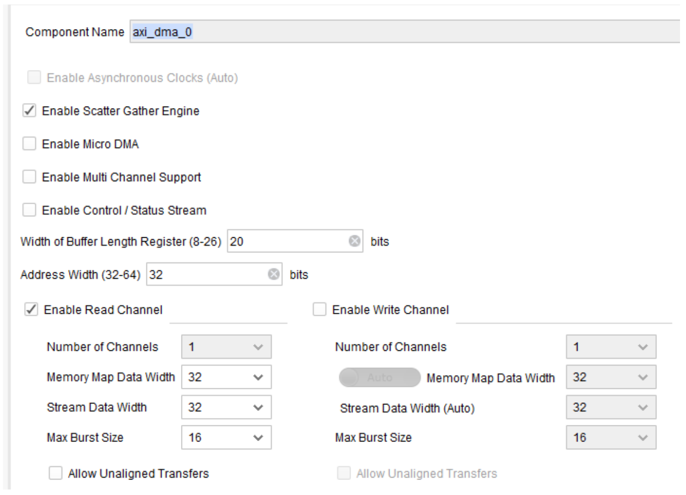


Figure IV.3: Hardware stereo vision module block.

Next, we adjusted the AXI VDMA used in the HDMI controller by enabling the S2MM channel (write channel), this channel allows us to transfer and write a stream of disparity values generated by the stereo vision module to DDR memory, we also set the number of frame buffers to 3. In the advanced setups, we set both channels to run in free-mode (no Fsync options) and we configured the write channel as a Genlock master so that the writing to the frame buffers cycles through frames 0,1,2,0,1,2, etc. The Read channel (MM2S) responsible of reading and displaying the frames buffers from DDR memory was configured as a Genlock slave. When a channel is configured as a slave, the channel works on the last completed frame by Master. The S2MM channel is composed of two sides, a slave stream side S_AXIS_S2MM connected the data stream source peripheral and a master side M_AXI_S2MM connected to a system memory. The stream side of the VDMA uses handshake signal ‘Tvalide’ that we connected to the write signal of the stereo vision module. To ensure that the HDMI display the results consecutively, we reduced the display resolution, so that the frame size of the display is as close as possible to the frame size of the stereo vision module output, we set the HDMI controller to the smallest resolution in the VTC IP which is the 352×288p that requires a clock pixel of 7.644MHz. To trigger the software obstacle detection module, we used S2MM interrupt signal, that will interrupt the Zynq PS when one frame is transferred to the DDR memory. To use interrupts, we had to enable the PL-PS interrupt ports in Zynq PS IP.

Finally, we added the AXI Direct Memory Access (AXI DMA) IP that provides high bandwidth direct memory access between memory and AXI4-Stream target peripherals. It supports AXI4-Stream data width of 8, 16, 32, 64, 128, 256 and 1024 bits [39]. Primary high-speed DMA data movement between system memory and stream target is through the MM2S master channel and S2MM slave channel. AXI DMA also enables up to 16 multiple channels of data movement on both channels in Scatter/Gather (SG) mode. It also provides the ability to queue multiple transfer requests. To emulate the camera, we used the MM2S channel to transfer the right and left pixels of the stereo images, for this, we configured the AXI DMA as shown in Figure IV.4.



Component Name axi_dma_0

☐ Enable Asynchronous Clocks (Auto)

☑ Enable Scatter Gather Engine

☐ Enable Micro DMA

☐ Enable Multi Channel Support

☐ Enable Control / Status Stream

Width of Buffer Length Register (8-26) 20 bits

Address Width (32-64) 32 bits

☑ Enable Read Channel		☐ Enable Write Channel	
Number of Channels	1	Number of Channels	1
Memory Map Data Width	32	Auto Memory Map Data Width	32
Stream Data Width	32	Stream Data Width (Auto)	32
Max Burst Size	16	Max Burst Size	16
☐ Allow Unaligned Transfers		☐ Allow Unaligned Transfers	

Figure IV.4: AXI DMA configuration.

We enabled the SG mode in order to use the cyclic mode and we set the stream data width to 32 bits. The buffer length register in the AXI DMA IP is used to hold the number of bytes to transfer, since the stereo vision module processes 240×320 pixels images, the number of bytes to transfer is 307200 bytes which can be written on 20 bits. The master MM2S channel uses ‘Tvalid’ and ‘Tready’ as handshake signals with stream target peripheral (stereo vision module), since the stereo vision module is running in free-mode, we connected the ‘Tready’ signal to a constant value 1, and the ‘Tvalide’ was connected to RST pin. To save the stereo images in DDR, both images were stored in the same binary file, we saved the pixels as a 32 bits data such that a left pixel saved in lower byte and a right pixel saved in upper byte. The file is then loaded to the SD card to be transferred by the PS to DDR memory. To re-extract the pixels values at the streaming side of the AXI DMA, we used two slice IPs to separate the lower (left) and upper (right) bytes as shown in Figure IV.5.

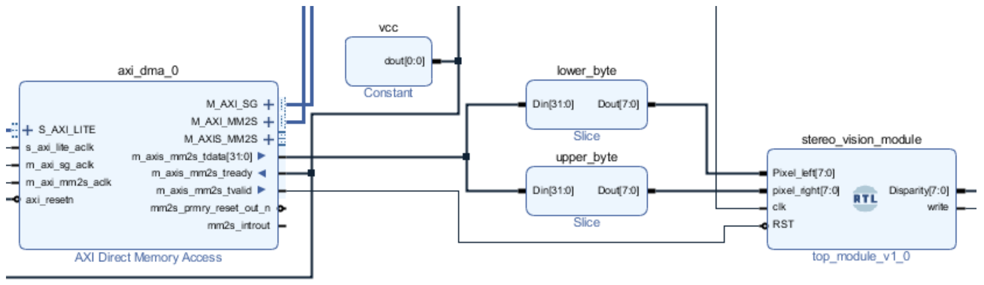


Figure IV.5:AXI DMA and stereo vision module connection.

For this implementation, we used three clock domains:

1. Clk1=80 MHz (used for AXI lite).
2. Clk2=7.644 MHz (used for HDMI controller).
3. Clk3=7.8 MHz (with this frequency, the stereo vision module will generate 101 fps).

IV.4.2 Software configuration

Our system is launched from the software partitions, first it starts by reading and transferring stereo pair file from the SD card to the DDR file using the `f_read()` function. Next, we configured the S2MM and the MM2S channels of the AXI VDMA. Then, we initialized the interrupt and defined the interrupt handler. Finally, we configured the MM2S channel of the AXI DMA. The configuration of different IPs was performed by writing the appropriate data to the internal registers of each IP.

1. Configuration of the S2MM VDMA channel:
   - We set the starting addresses of the 3 frame buffers to S2MM_Start_Address registers.
   - We wrote the control word to S2MM_VDMACR register to start the channel, enable circular_park, and interrupt at the end of each frame.
   - We set the frame horizontal and vertical size in S2MM_HSIZE and S2MM_VSIZE registers based on DM dimension.

2. Configuration of the MM2S VDMA channel:
   - We set the same starting addresses of the 3 frame buffers to MM2S_Start_Address registers.
   - We wrote the control word to MM2S_VDMACR register to start the channel and enable circular_park.
   - We set the frame horizontal and vertical size in S2MM_HSIZE and S2MM_VSIZE registers to the horizontal and vertical size of the 352×288p frame size.

3. Configuration of the MM2S DMA channel:
   - We define the buffer descriptor BD1, set the buffer address and size of one packet transaction.
   - We Set the next descriptor pointer to the current descriptor to continuously stream the same buffer descriptor BD1.
   - We enable the cyclic mode in the MM2S DMA control register.
   - We halt the MM2S channel in MM2S status register to be able to configure the DMA IP.
   - We set the MM2S DMA current descriptor pointer register to BD1.
   - We start the MM2S DMA channel in the MM2S DMA control register.
   - We set the MM2S DMA tail descriptor pointer register to random address.

4. Interrupt handler:
   - We reset the VDMA interrupt by writing 1 to FrmCnt_Irq bit in the S2MM VDMA status register.
   - We disable the Zynq interrupt.
   - We read the current working frame of the S2MM VDMA channel from the S2MM current frame pointer status register. Since the S2MM channel is in Genlock master mode, then we can deduce the transferred frame and its address.
   - We pass the address of frame buffer and disparity range dmax to the obstacle detection module, to execute the `V_disparity()`, `U_disparity()`, `horizontal_ROI()`, `vertical_ROI()`, and `ROI()` functions,
   - We enable again the Zynq interrupt.

IV.5 Experimental results

For the experimental part, we used the Tsukuba stereo pair of size 240×320 pixels and we fixed the window size to 5*5 and disparity range to 20 pixels in the hardware stereo vision module. To make sure that DM output is well transferred to the DDR memory, we disabled the interrupt and displayed the generated DM directly without processing the data, the DM display is shown in Figure IV.6.

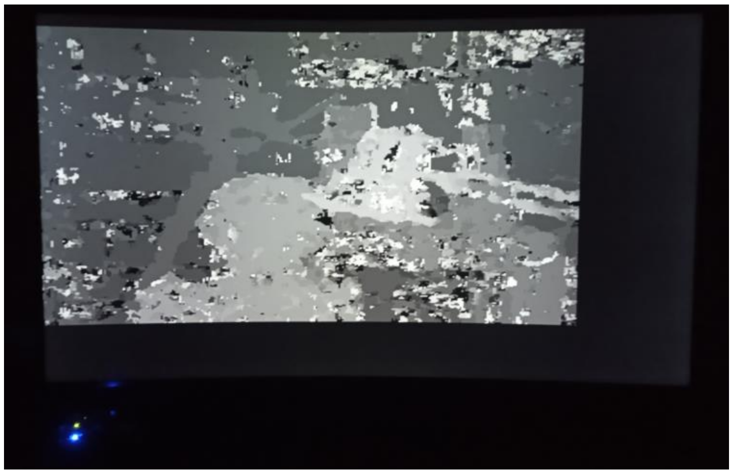


Figure IV.6: DM generated by the hardware stereo vision module.

Once we insured that DM is well received, we enabled the interrupt. During the display, we noticed the display was not clear, where some DMs are displayed, but no obstacles are detected. To investigate the reason, we measured the execution time of the interrupt handler, which was found to be around 0.017s, that results a processing rate of 58 fps. Hence, we concluded that some frames are skipped and not processed because our obstacle detection module is slower than the stereo vision module. This incident was clearly spotted using the Integrated Logic Analyzer (ILA) debugging tool to prob the VDMA S2MM interrupt signal, the ILA results are shown in Figure IV.7. The red cursor points to the first interrupt generated after transferring the first DM frame to DDR memory, the interrupt is quickly deasserted by the interrupt handler which signify that the frame is being processed. For the yellow cursor points to the second interrupt generated after transferring the second DM frame, the interrupt is not deasserted rapidly, since processor is still processing the previous DM frame.

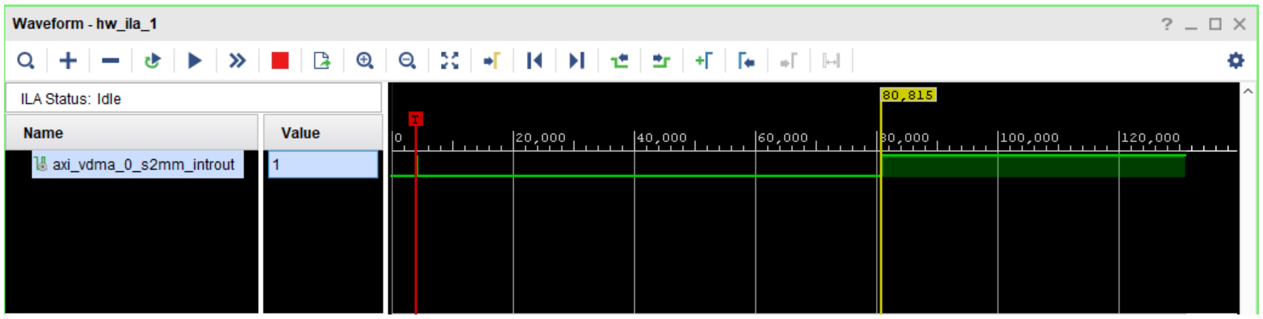


Figure IV.7: ILA results before optimization.

To reduce the flickering effect, we tried to optimize the obstacle detection module. For this purpose, we combined `V_disparity()` and `U_disparity()` functions in one function `UV_disparity()`. Using ILA, we extracted the results shown in Figure IV.8. We can see that the second interrupt is deasserted faster compared to the previous case. This optimization reduced the execution time of the obstacle detection to around 0.012s, hence increasing the processing rate to 78 fps.

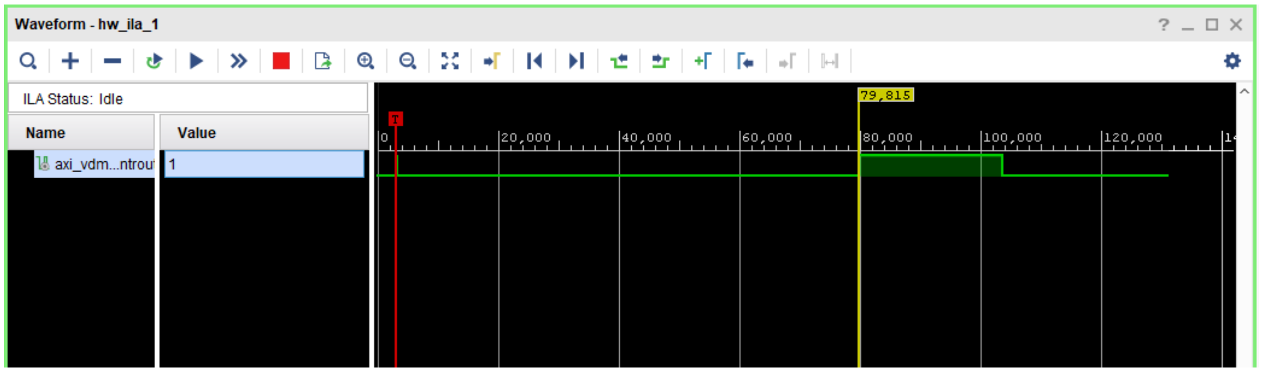


Figure IV.8: ILA results after optimization.

The codesign obstacle detection system results are shown in Figure IV.9(a). Comparing to the results generated by the software design using the same parameters shown in Figure IV.9(b), we can see some differences due to the difference in DM size. The stereo image used is of size 240×320 pixels; for a 5*5 window and a search range of 20 pixels, the DM size generated by software stereo vision module is (240-2*(w-1))×(320-2*(w-1))=232×312 pixels as explained in sections II.3.1 and II.3.2, while the hardware module generates a DM of size 232×292 pixels where additional dmax (search range) pixels are removed from DM width. This size difference affects the UDM, the VDM, the lines extraction, and hence the obstacle detection.

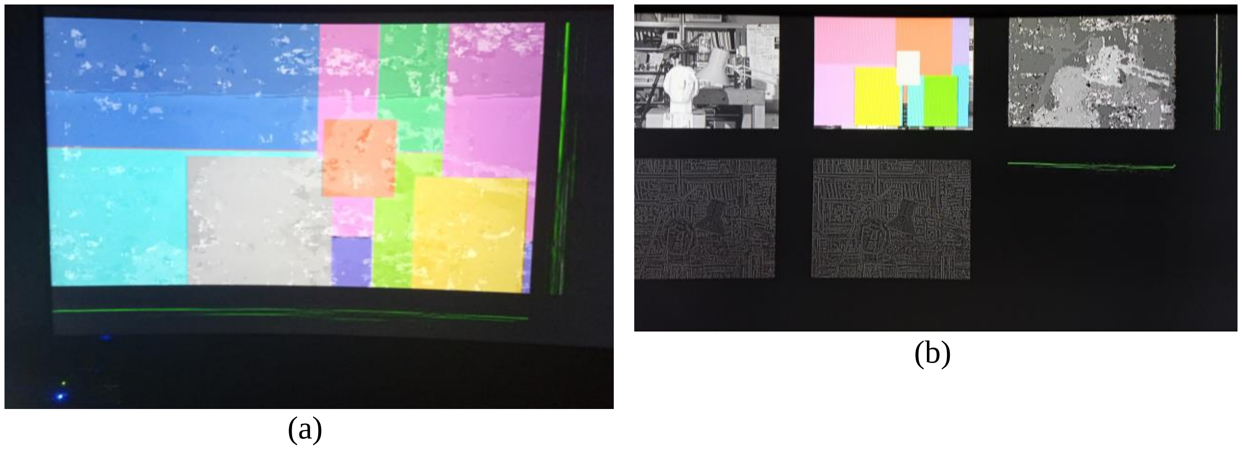


Figure IV.9: Obstacle detection results of (a) codesign, (b) software design.

As explained previously, in codesign methodology trade-offs are made. For our implementation a cost-performance trade-off is made to accelerate the processing speed by allocating more PL resources. Table IV.1 illustrates the PL resource percentage usage for the two used methodology. The resource usage of the codesign system is measured for search range 20 pixels, window size 5*5, and image size of 480*640 pixels, this percentage would increase when increasing any of the parameters.

Table IV.1: PL resource percentage usage of software design and codesign.

	Software design	**Codesign**
LUT	6.95%	37.93%
FF	6.01%	23.30%
BRAM	1.43%	4.29%

IV.6 Conclusion

In this chapter, we implemented the stereo vision-based obstacle detection system using the hardware/software codesign on a Zynq device. The partitioning of the system was based on the profiling information obtained from the previous software design; therefore, we used a hardware stereo vision module merged with a software obstacle detection module. The integration of the two modules was undemanding task due to the Xilinx IPs and protocols that facilitate the hardware/software matching. The interfacing of the two sub-systems was controlled using the VDMA S2MM and MM2S channels that combines AXI memory mapped and AXI stream protocols. Through the codesign system we obtained a performance rate of 78 fps for images size of 240×320 pixels, this enhancement is associated with an increase in PL resources usage.

General conclusion

In this project, we used two different designs methodologies for the implementation of a stereo vision-based obstacle detection. The first methodology is the software design, where we built a single thread application. The application was divided into two sub-modules: the stereo vision modules that generates a DM from which we can estimate the depth. The DM generation was achieved using the RANK transform, and SAD correlation metric techniques. The second module is the obstacle detection module that extract the obstacles using the UV DM approach. For the stereo vision module evaluation, we used the BMP percentage measurement, and we found that the relation between the DM accuracy and the two stereo vision module parameters: window size and search range is non-linear, thus, a precise selection of the stereo vision module parameters can result a good DM accuracy and reduce the unnecessary computations. The complete system was tested using images form the DrivingStereo dataset, that provides stereo images taken in different weather conditions. The obstacle detection results were very acceptable, since our system was capable of detecting most of the obstacles on the scenes despite the bad weather conditions.

The processing rate of the software stereo vision-based obstacle detection system was 1/3 fps for images size of 240×320 pixels, this result shows that the software system does not attain the real-time speed requirements. The low processing rate was mainly caused by the stereo vision module, which is a computationally intensive task that consumes 93% of the total execution time. To overcome this limitation in the software design, we implemented an obstacle detection system based on the hardware/software codesign methodology. In this implementation, we preserved the same software obstacle detection module and we integrated a hardware stereo vision module that was implemented on FPGA. The integration of the two modules was carried out using the S2MM channel in the VDMA IP and an interrupt signal. The S2MM channel was responsible of transferring the stream of disparity values generated by the hardware stereo vision module to the DDR memory, while the interrupt signal was used to trigger the execution of the software obstacle detection module at the end of each frame transaction. The hardware acceleration of the stereo vision module, enhanced the performance of the stereo vision-based obstacle detection system. As a result, the processing rate was accelerated by 234 times for images size of 240×320 pixels. This enhancement was associated with an increase in PL resources usage, which is a clear illustration of the cost-performance trade-off.

Although we successfully implemented the hardware/software codesign of the real time stereo vision-based obstacle detection system, still the software obstacle detection module slower compared to hardware stereo vision module. For a further work and to overcome this limitation, we suggest a software acceleration by implementing the obstacle detection module as a multi-thread application, and using the road plane extraction algorithm instead of the UV DM approach. For a further improvement, an acquisition system can be added to our system so that we can remove the camera emulation part in our design.

References

[1] P. Dufour, "Real-Time Stereo-Matching on Embedded Hardware for MAVs," Swiss Federal Institute of Technology Zurich, 15. April 2010.

[2] D. E. B. Orozco, "Using Epipolar Geometry in stereo image analysis," international school for advanced studies, Italy, 2005.

[3] N. Van Oosterwyck, "Real Time Human-Robot Interactions and Speed Control of a Robotic Arm for Collaborative Operations," *PhD diss., Ph. D. dissertation, 05 2018,* 2018.

[4] H. Entner, "Real-time 3D Reconstruction for Autonomous Football Playing Robots Using a Feature Based Stereo Approach," Technical report, PRIP, TU Wien, 2005.

[5] "What Is Camera Calibration?," [Online]. Available: https://www.mathworks.com/help/vision/ug/camera-calibration.html. [Accessed 2 05 2022].

[6] T. H. M. Siddique, Y. Rehman, T. Rafiq, M. Z. Nisar, M. S. Ibrahim and M. Usman, "3D Object Localization Using 2D Estimates for Computer Vision Applications," In 2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC), pp. 1-6. IEEE, 2021..

[7] K. Hata and S. Savarese, Course Notes 1: Camera Models.

[8] R. Tsai, "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," IEEE Journal on Robotics and Automation 3, no. 4 (1987): 323-344..

[9] X. Chai, F. Zhou and X. Chen, "Epipolar constraint of single-camera mirror binocular stereo vision systems," 19 August 2017.

[10] X. A. Quintana, "Modelling Stereoscopic Vision Systems for Robotic Applications," Girona university , spain, July 2003.

[11] Y. Li, "Stereo vision and LIDAR based Dynamic Occupancy Grid mapping : Application to scenes analysis for Intelligent Vehicles," Computers and Society [cs.CY], Université de Technologie de BelfortMontbeliard, 2014.

[12] I. Benacer, "Conception et implémentation sur FPGA d’une architecture pour l’extraction temps-réel du plan de sol," ecole militaire polytechnique, algeria, 8 / 09 / 2014.

[13] A. KILIÇ, "navigation of a mobile robot using stereo vision," university of gaziantep graduate school of natural & applied sciences, Turkey, 2010.

[14] m. Dominguez-Morales, A. Jiménez-Fernandez, R. Paz-Vicente and A. Linares-Barranco, "Stereo Matching: From the Basis to Neuromorphic Engineering," *In Current Advancements in Stereo Vision. IntechOpen,* July 2012.

[15] J. Banks, M. Bennamoun and P. Corke, "Non-parametric techniques for fast and robust stereo matching," *In TENCON'97 Brisbane-Australia. Proceedings of IEEE TENCON'97.*

IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications (Cat. No. 97CH36162), vol. 1, pp. 365-368. IEEE, 1997.

[16] S. Gautama, S. lacroix and M. Devy, "Evaluation of stereo matching algorithms for occupant detection," *In Proceedings International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems. In Conjunction with ICCV'99 (Cat. No. PR00378), pp. 177-184. IEEE,,* 1999.

[17] T. E. Saidi , A. Khouas and A. Abbes, "Accelerating stereo matching on mutlicore ARM platform," *In 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5. IEEE,* 2020.

[18] C. Stentoumis, L. Grammatikopoulos, I. Kalisperakis, G. Karras and E. Petsa, "Stereo matching based on census transformation of image gradients," *In Videometrics, Range Imaging, and Applications XIII, vol. 9528, p. 95280Q. International Society for Optics and Photonics,* 2015.

[19] S. Young-Ho, Y. Ji-Sang and K. Dong-Wook, "A new parallel hardware architecture for high-performance stereo matching calculation," *Integration, the VLSI Journal 51,* (2015): 81-91.

[20] G. Dubbelman, "Vision based Obstacle Detection for both day and night conditions," Intelligent Systems Laboratory Amsterdam, 2007.

[21] H. Zhencheng and K. Uchimura, "U-V-Disparity: An efficient algorithm for Stereovision Based Scene Analysis," *In IEEE Proceedings. Intelligent Vehicles Symposium, 2005., pp. 48-54. IEEE,,* 2005.

[22] "Parallax: disparity and UV-disparity: UV-disparity," [Online]. Available: https://programmersought.com/article/1006388192/. [Accessed 5 2022].

[23] M. Zhang, P. Liu, X. Zhao, X. Zhao and Y. Zhang, "An obstacle detection algorithm based on UV disparity map analysis," *n 2010 IEEE International Conference on Information Theory and Information Security, pp. 763-766. IEEE,* 2010.

[24] N. Fakhfakh, D. Gruyer and D. Aubert, "Weighted v-disparity approach for obstacles localization in highway environments," *In 2013 IEEE Intelligent Vehicles Symposium (IV), pp. 1271-1278. IEEE,* 2013.

[25] "Zynq-7000 All Programmable SoC: Software Developers Guide," 6 june 2018. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug821-zynq-7000-swdev. [Accessed august 2022].

[26] "ZedBoard_RevD.2_Schematic," 2013. [Online]. Available: https://community.element14.com/products/devtools/avnetboardscommunity/m/files/813. [Accessed august 2022].

[27] LTP, "STM32F0 I2C - Tutorial 7 with STM32CubeMX," 13 May 2017. [Online]. Available: https://letanphuc.net/2017/05/stm32f0-i2c-tutorial-7/. [Accessed june 2022].

[28] J. B. Jonathan Valdez, "Understanding the I2C bus," june 2015. [Online]. Available: https://www.ti.com/lit/an/slva704/slva704.pdf. [Accessed 2022].

[29] florentw, "Video Beginner Series 16: Understanding Video Timing with the VTC IP," 15 12 2021. [Online]. Available: https://support.xilinx.com/s/article/899769?language=en_US. [Accessed june 2022].

[30] "Zynq_Architecture," 2012. [Online]. Available: http://www.ioe.nchu.edu.tw/Pic/CourseItem/4468_20_Zynq_Architecture.pdf. [Accessed september 2022].

[31] "Zynq-7000 SoC Technical Reference Manual," 2 April 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug585-Zynq-7000-TRM. [Accessed september 2022].

[32] S. A. Area, "User interface development in embedded dual-core system for power converter controller application," 2018.

[33] "Vivado Design Suite Vivado AXI Reference," 15 July 2017. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug1037-vivado-axi-reference-guide. [Accessed september 2022].

[34] D. Scharstein, "2001 Stereo datasets with ground truth," [Online]. Available: https://vision.middlebury.edu/stereo/data/scenes2001/. [Accessed september 2022].

[35] S. Merrouche, M. Andrić and B. Bondžulić, "Objective Image Quality Measures for Disparity Maps Evaluation," *Electronics,* 9, no. 10 (2020): 1625..

[36] S. V. Cavalcante, "A Hardware-Software Co-Design System for Embedded Real-Time Applications," University of Newcastle upon Tyne, June, 1997.

[37] M. A. Dias and F. S. Osorio , "Hardware/Software Co-design for Image Cross-Correlation," University of Sao Paulo - USP, June 2011.

[38] Y. Karim, "FPGA Implementation of Stereo Matching Algorithm for Depth Estimation," IGEE, University of M'Hamed BOUGARA, Boumerdes, 2022.

[39] Xilinx, "AXI DMA LogiCORE IP Product Guide (PG021)," 14 June 2019. [Online]. Available: https://docs.xilinx.com/r/en-US/pg021_axi_dma/AXI-DMA-v7.1-LogiCORE-IP-Product-Guide. [Accessed 2022].