كلية الهندسة بحلوان

جامعة حلوان

[2020/2021]

Optimizing Computer Vision by FPGA Hardware Accelerator

Faculty of Engineering Electronics and Communication Engineering Department

Graduation Project

Academic Year 2020/2021

Supervisor: Dr. Mohamed Elbably

Dr. Aya Abdallah

Team members:

• Eslam Ahmed Abd Al-azeem	Communication
• Alhassan Mohsen Abd Al-ghaffar	Communication
• Fayrouz Hammad Hamed	Communication
• Amr Ragab Mabrouk	Communication
• Eslam Salah Hassanin	Communication
• Refaat Hassan Mahmoud	Mechatronics

Contents

ABSTRACT:

AI algorithms, especially deep learning (DL) and computer vision algorithms, require significant storage to use large databases and powerful computational and fast processing. To produce an autonomous adaptive smart system for highly accurate and human-like behavior. Hardware acceleration makes faster and more efficient processing, reduces cost, and makes the system have high accuracy. Our project aims to use a computer vision (CV) algorithm in a specific application and implement this algorithm using FPGA (hardware accelerator). FPGA Based Computer Vision will be suitable for applications that are depending on battery-operated systems where power consumption and speed are critical factors. Therefore, we chose to our application to be a task from "Advanced Driver Assistant Systems (ADAS)" that depends on computer vision, which is, Lane Departure Warning System.

ADAS is considered the most developed field that depends on the battery-operated operations; the lane departure warning system (LDWS) is a driver aid that utilizes visual sensors to detect lane markers ahead of the vehicle. The LDWS alerts the driver when the vehicle is laterally approaching a lane boundary marker (indicated by a solid line, a dashed line, or raised reflective indicators such as Botts dots). The LDWS sounds an audible tone or beeps and is often associated with a visual dash lamp or display icon to indicate which side of the vehicle is departing the lane.

Lane Departure Warning system (LDW) is used to provide safer driving and reduces the risk of accidents where, LDWS have been in commercial development for both passenger cars and heavy vehicles for over twenty years. Moreover, this is a visual task deployed on a battery-operated platform, which suit the characteristics of the project.

Keywords

Computer vision, LDW, ADAS, algorithms, FPGA, ECU, ASIC, DL, implementation, code, control, power, processing, lane, design, model, optimization, hardware acceleration

1. Chapter 1: introduction:

Nowadays computer vision algorithms are very important for the society, it is used in many applications and fields, and it gives us many features and advantages. It looks like you make the machine (computer) act as a human, can see the event in the real life by the camera, and take the action based on every application. For example, some application you need an alert only for attention, other applications may need the action is to do something like controlling a car, operating the fire extinguishing system, etc. based on every application you make the action needed. We choose the advanced driver assistance system (ADAS) to be our application, and the Lane Departure Warning System (LDWS) the task that we will work on.

The connected, electric, and autonomous vehicles will alter how cars are evaluated and used in particular; the value of electric and autonomous vehicles is increasingly found in the electronics, and not the mechanical aspects of the vehicle. In the new age of mobility, companies that are able to own and optimize the design of critical electronics will capture more of the profit available. This fact is bringing traditional automotive manufacturers onto the electronics business, and simultaneously attracting tech companies like Google and Facebook into the automotive industry. There are many levels of autonomous vehicles, the first level is Fully autonomous vehicles and the other level is Advanced driver assistance system (ADAS).

Our aim is to use hardware acceleration to optimize lane keeping assist system (LKAS) as one of (ADAS) tasks which depend on computer vision. In ADAS, the speed of the reaction is critical and the power consumption due to the fact that cars are battery operated.

1.1. Computer-Vision:

Computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Using digital feedback from cameras and videos machines can accurately identify and classify objects, and then react to what they "see. (6)"

Computer vision tasks include methods for acquiring, processing, analyzing, and understanding digital images, so we can say that computer vision work in three basic steps they are:

- **Acquiring an image:**

Images, even large sets, can be acquired in real-time through video, photos, or 3D technology for analysis.

- **Processing the image:**

Through Filters, or Deep learning models that automate much of this process, but the models are often trained by first being fed thousands of labeled or pre-identified images.

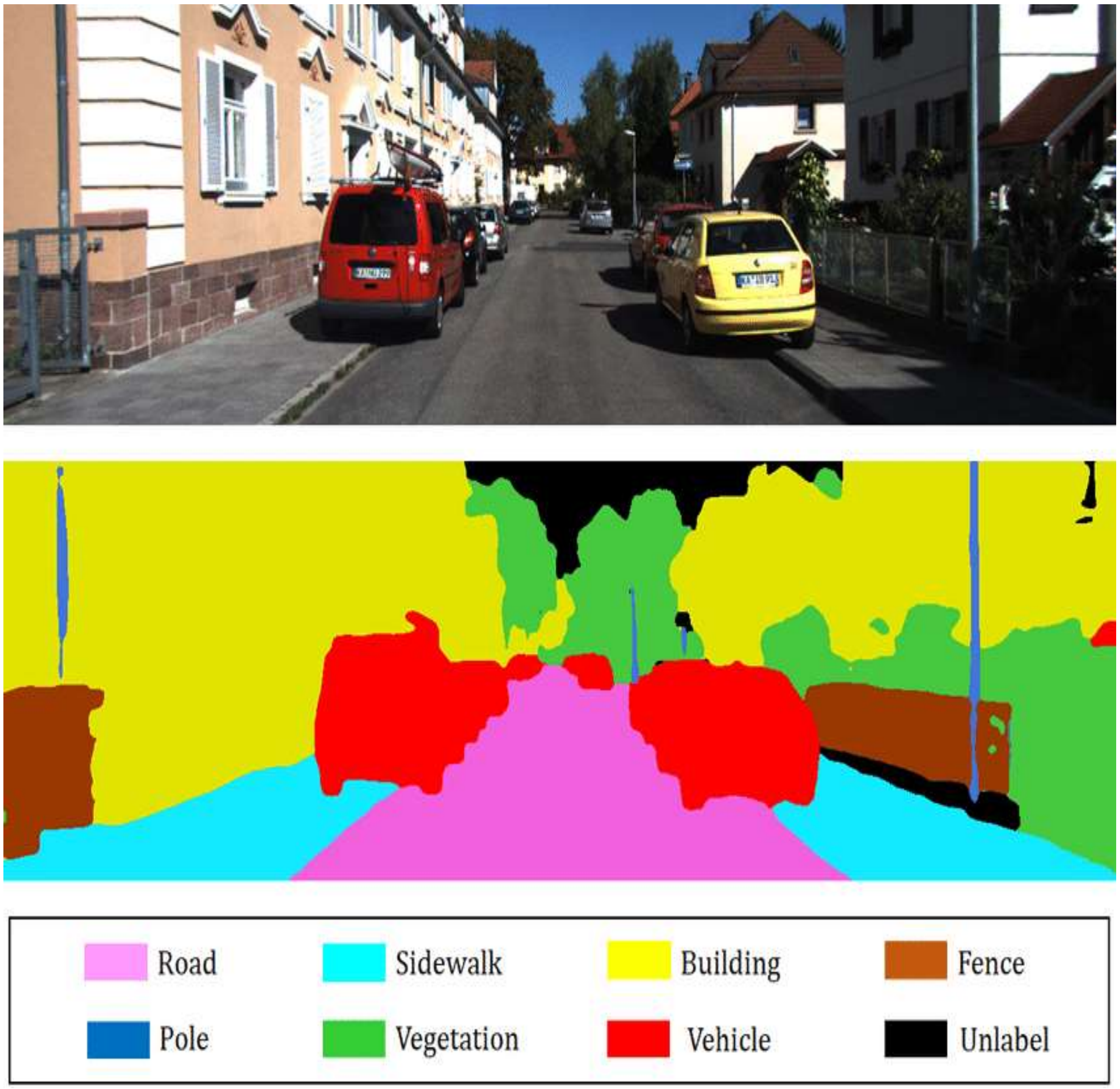
- **Understanding the image:**

The final step is the interpretative step, where an object is identified or classified.

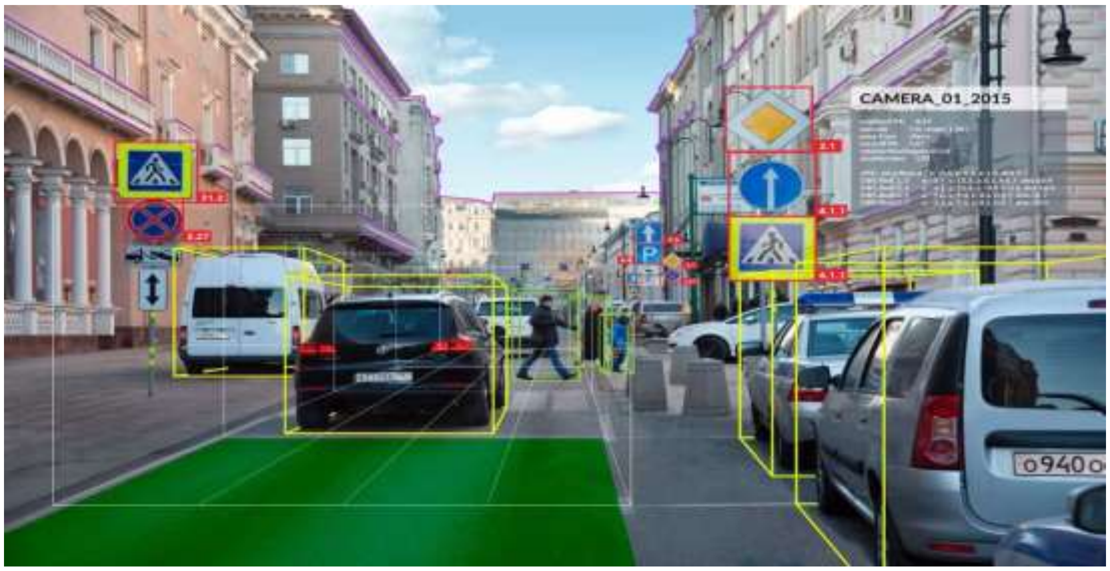
1.1.1. Computer vision Techniques:

- **Image segmentation:**

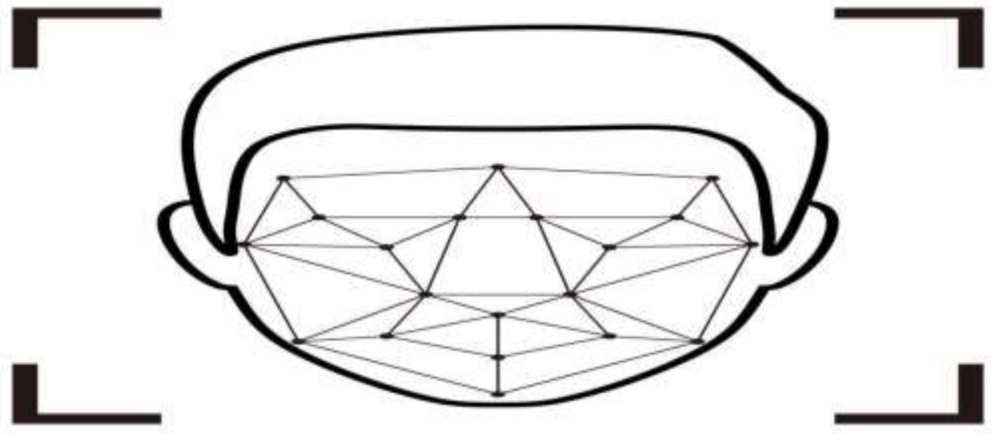
  partitions an image into multiple regions or pieces to be examined separately, for example shown in figure below segmentation between vehicle, road, building, sidewalk, etc.



- **Object detection:**
  identifies a specific object in an image. The advanced object detection can recognize many objects in a single image for example in the football field it can detect an offensive player, a defensive player, a ball and so on. These models use an X,Y coordinate to create a bounding box and identify everything inside the box.



- **Facial recognition:**
  is an advanced type of object detection that not only recognizes a human face in an image, also identifies a specific individual.



- **Edge detection:**
  is a technique used to identify the outside edge of an object or landscape to better identify what is in the image.

- **Pattern detection:** is a process of recognizing repeated shapes, colors and other visual indicators in images like fingerprint.

.

- **Image classification:** groups images into different categories, like classify cats or dogs and etc.
- **Feature matching:** is a type of pattern detection that matches similarities in images to help classify them.

Simple applications of computer vision may only use one of these techniques, but more advanced uses, like computer vision for self-driving cars, rely on multiple techniques to accomplish their goal(6).

1.1.2. Computer vison Applications:

Now we have a very large number of applications that we use computer vision in, and here some examples of fields and applications:

- Medicine
- Military
- Autonomous vehicles (like self-driving cars)
- Industry
- Sports

Each track and field of this you can use computer vision in many things for a lot of applications.

1.1.3. Benefits and economic impacts of CV:

- It makes the computer (machine) act like human, can see and take an action control or monitor.
- Classify and identify the objects.
- In any application, it makes our life easier.
- Increasing human safety in dangerous situations.

1.2. ADAS:

The (ADAS) is abbreviation for advanced driver assistance system, which is an electronic system that assist drivers in driving and parking functions.

Through a safe human-machine interface, (ADAS) increase car and road safety reduce the road accidents and may solve traffic problems. (ADAS) use automated technology, such as sensors and cameras, to detect nearby obstacles or driver errors, and respond accordingly.

Currently, ADAS is considered the most developed field, which depends on battery-operated operation (1).

1.2.1. ADAS applications:

- **Blind spot detection (BSD):**

  Majorly based on radar sensors, however it can be built on lidar, camera and ultrasonic sensor.

- **Autonomous emergency braking (AEB):**

  Majorly based on radar and LIDAR sensors.

- **Adaptive cruise control (ACC):**

  Majorly based on radar and LIDAR sensors.

- **Forward / rear collision warning system (FCW&RSW):**

  Majorly based on radar and LIDAR sensors.

- **Intelligent parking Assistance (IPA):**

Is based on camera and Ultrasonic sensors.

- **Cross traffic Alert (CTA):**

  Majorly based on radar sensors, however it can be built on LIDAR, camera.

- **Lane change assist (LCA):**

  Majorly based on camera and LIDAR sensors.

- **Lane keeping assist (LKA):**

  Majorly based on camera.

- **Lane departure warning (LDW):**

  Majorly based on camera.

- **Automatic parking.**

  Majorly based on camera and ultrasonic.

- **Blind spot vision.**

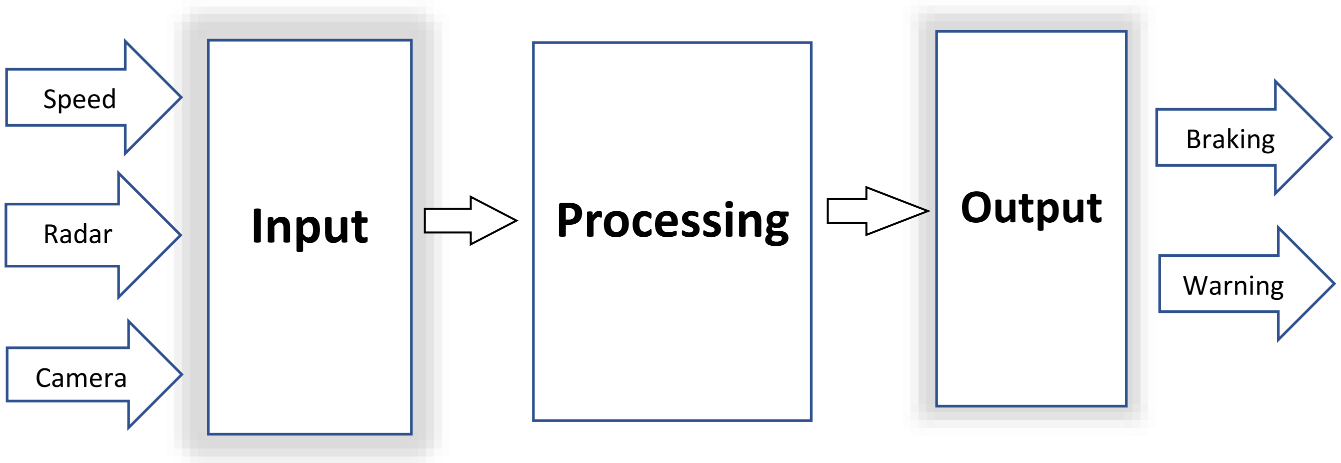
  Majorly based on camera and LIDAR sensors.

- **Night vision and navigation system.**

  Majorly based on RADAR and LIDAR sensors.

These are some examples for the (ADAS) applications and features, but there are many other features that can be added to the system to help the driver (1).

1.2.2. Typical system-level design for ADAS in automotive:



1.3. Lane Departure Warning System

Lane Departure Warning System (LDWS) is a driver aid that utilizes visual sensors to detect lane markers ahead of the vehicle. The LDWS alerts the driver when the vehicle is laterally approaching a lane boundary marker (indicated by a solid line, a dashed line, or raised reflective indicators such as Botts dots). The LDWS sounds an audible tone or beeps and is often associated with a visual dash lamp or display icon to indicate which side of the vehicle is departing the lane.

It works to keep you in your lane and easily cancelled by nudging the wheel, the system will not work when you light the right or left sign;

First you may receive an alert (via a sound, flashing light or vibration) if you drift out of your lane. This feature relies on painted lane markings to operate. These include the markings between lanes and along the edges of the road. Some versions of this feature may also help prevent you from driving off the road.

Lane departure warning systems have been in development by industry for over 20 years. LDWS are generally visual devices that look at the lane line markers to compute a predicted moment of lane departure and alert the driver when unintended lane departures are about to occur without causing undue false warnings due to subtle lateral lane position changes. Beginning with simple line scan video, LDW has developed into sophisticated lane marker identification and lane boundary projection systems that provide the driver with a warning if the vehicle has a trajectory that will take it out of lane. While most LDWS apply video techniques, other areas of research include infrared, Lidar, magnetic, and electronic mapping technologies.

According to American Association of State Highway and Transportation Officials (AASHTO), almost 60% of the fatal accidents are caused by an unintentional lane drifting of a vehicle on major roads. Similarly, in a Minnesota crash study, it was reported that 25 to 50 % of the severe road departure crashes in Minnesota occur on curves, even though curves account for only 10 % of the total system mileage. Systems that predict the driver's attentive state and intent of lane change and provide map-based route guidance and/or warning about unintentional lane departure are all useful to reduce major road crashes. The Majority of these crashes involve crossing of an edge line, centerline, or otherwise leaving the intended lane or trajectory. According to a recent study, which compared crashes with and without an LDWS, it was found that an in-vehicle LDWS was helpful in reducing crashes of all severities by 18%, with injuries by 24%, and with fatalities by 86% without considering for driver demographics (4).

2. Chapter 2: mathematical foundations:

2.1. Distortion correcting and Camera calibration:

Our ultimate goal of this algorithm is to find a correct information about the lane of the road and the position of the car with respect to these lanes to give the car the correct steering order. To reach our goal we need first to know some information about the environment, the most important information is to calculate the curvature of the lanes and the offset of the car from the ego line. However, the input of the system, which is an image of the road, is distorted in all possible ways and we need to equalize these effects. In this section, we are going to describe the image distortion and how to make the image in a suitable form for extracting the information we need.

Image distortion occurs when a camera looks at 3D objects in the real world and transforms them into a 2D image; this transformation is not perfect. Distortion actually changes what the shape and size of these 3D objects appear to be so in order to measure the curvature of the lanes we cannot use a distorted image. Therefore, the first step in analyzing camera images is to undo this distortion so that you can get correct and useful information out of them.

2.1.1. Types of Distortion:

Real cameras use curved lenses to form an image, and light rays often bend a little too much or too little at the edges of these lenses. This creates an effect that distorts the edges of images, so that lines or objects appear more or less curved than they actually are. This is called radial distortion, and it is the most common type of distortion.

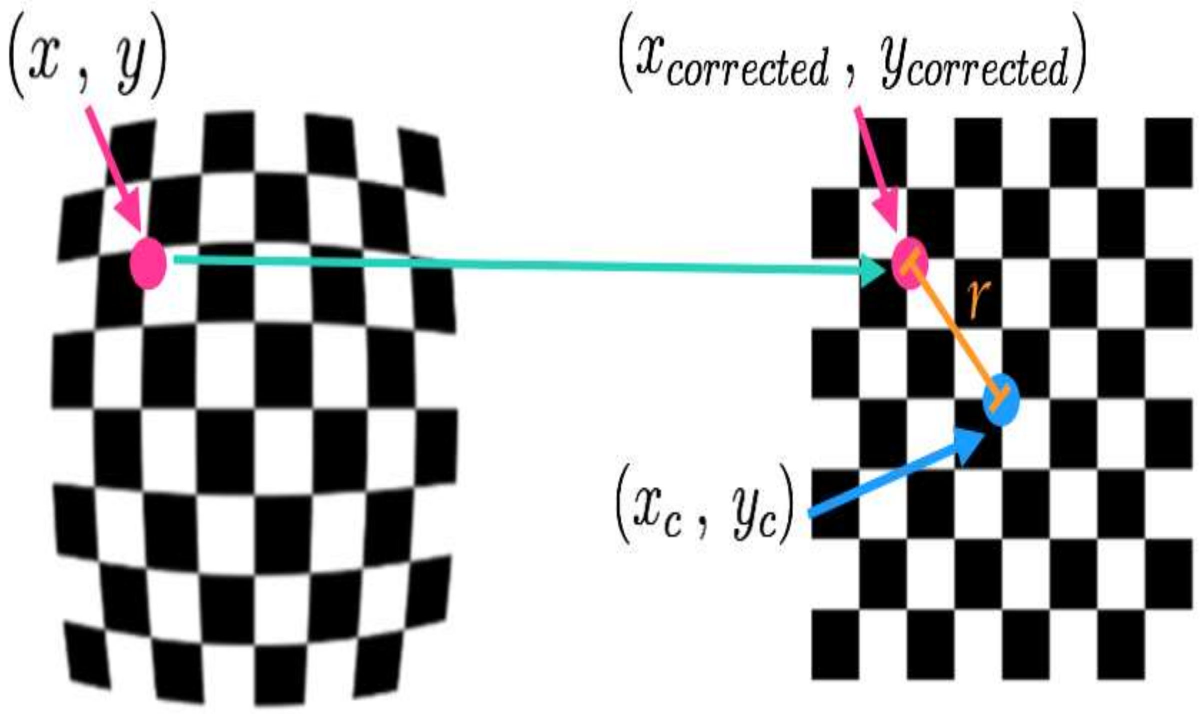
Another type of distortion is tangential distortion, which occurs when a camera's lens is not aligned parallel to the imaging plane, where the camera film or sensor is. This makes an image look tilted so that some objects appear farther away or closer than they actually are.

2.1.2. Distortion Coefficients and Correction:

For calibration, we use a chessboard picture because its regular high contrast pattern makes it easy to detect automatically. Therefore, we use our camera to take multiple pictures of the chessboard pattern against a flat surface and using these pictures, we can be able to detect any distortion as follows.

This distortion can generally be captured using five numbers (k1, k2, p1, p2, k3) which called distortion coefficient which its values reflect the amount of radial and tangential distortion, using these coefficients we can calibrate our camera to restore the undistorted version of the image. Three of these coefficients are needed to correct for radial distortion: k1, k2, and k3. To correct the appearance of radially distorted points in an image, one can use a correction formula.

In the following equations (*x*,*y*) is a point in a distorted image. To undistort these points, we use the distance r between a point in an undistorted (corrected) image ($X_{corrected}$, $Y_{corrected}$) and the center of the image distortion, which is often the cente of that image (X_c, Y_c). This center point is sometimes referred to as the *distortion center*. These points are pictured below.



There are two more coefficients that account for tangential distortion: p1 and p2, and this distortion can be corrected using a different correction formula(4).

2.2. Perspective transform

A perspective transform maps the points in a given image to different, desired, image points with a new perspective. The perspective transform you will be most interested in is a bird’s-eye view transform that let us view a lane from above; this will be useful for calculating the lane curvature later on. Aside from creating a bird’s eye view representation of an image, a perspective transform can also be used for all kinds of different viewpoints.

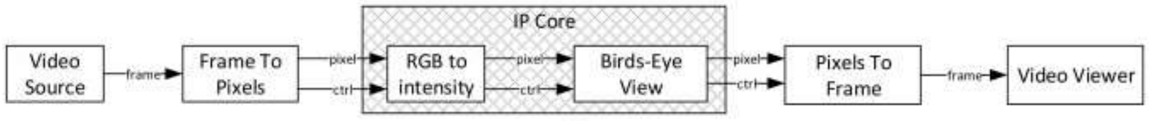
This geometric transformation, generically known as perspective transformation, is mathematically governed by the homograph matrix,

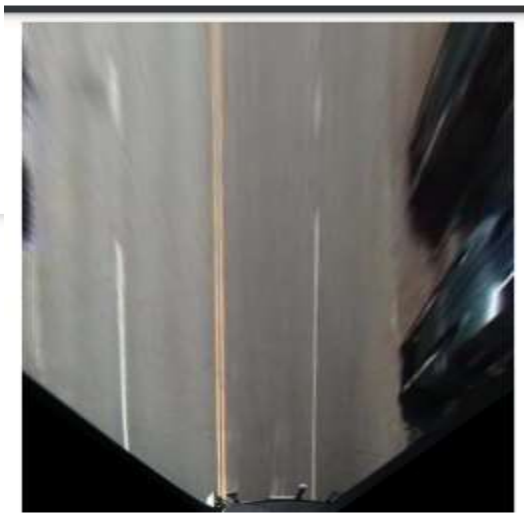
$$h = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \qquad (1)$$

The input-output relation is given as

$$(\hat{x}, \hat{y}) = \left(\frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \right)$$

Where (x ^, y ^) and (x, y) denote the co-ordinates pair of the pixels in the transformed and input frames, respectively (2).





2.3. Sobel Operator

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high frequency that correspond to edges. Typically, it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.

Here we also need a window to do the scanning work and here we still choose 3 by 3 as the size of the window, then we apply two kernels on the sliding window separately and independently. The x directional kernel is shown as the following equation:

$$G_x = \begin{bmatrix} 0.125 & 0 & -0.125 \\ 0.25 & 0 & -0.25 \\ 0.125 & 0 & -0.125 \end{bmatrix}$$

In addition, y directional kernel is similar. By summing up the x and y directional kernel response, we get the final Sobel filter response.

The operator of Sobel filter is:

$$G = G_x^2 + G_y^2$$

Where the Gx is Sobel kernel in x - direction and similarly Gy in y-direction(4).

2.4. HLS Color Space

A color space is a specific organization of colors; color spaces provide a way to categorize colors and represent them in digital images.

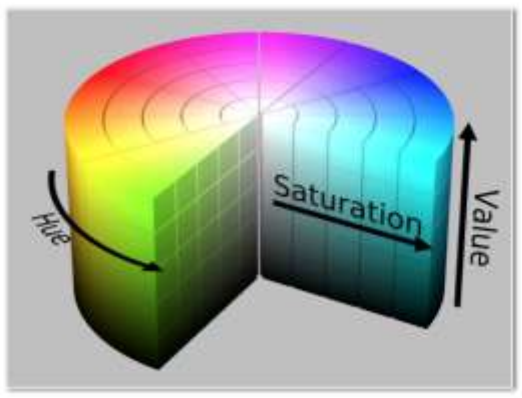
RGB is red-green-blue color space. You can think of this as a 3D space, in this case a cube, where any color can be represented by a 3D coordinate of R, G, and B values. For example, white has the coordinate (255, 255, 255), which has the maximum value for red, green, and blue.

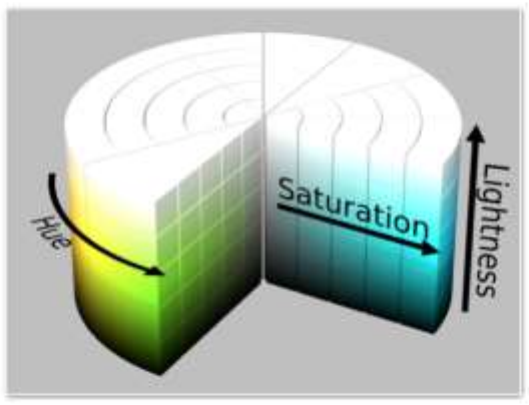
The RGB is not the only available way or space to represent the color information in the image; there are many other ways for that.

For example, there is also HSV (Hue, saturation and value) color space, also HLS (hue, lightness, and saturation) space and many other spaces that can be used for different purposes.

For our application, we will work with HLS space; to get more intuition about what HLS means you can generally think of Hue as the value that represents color independent of any change in brightness. So if you imagine a basic red paint color, then add some white to it or some black to make that color lighter or darker -- the underlying color remains the same and the hue for all of these colors will be the same.

On the other hand, Lightness and Value represent different ways to measure the relative lightness or darkness of a color. For example, a dark red will have a similar hue but much lower value for lightness than a light red. Saturation also plays a part in this; saturation is a measurement of colorfulness. So, as colors get lighter and closer to white, they have a lower saturation value, whereas colors that are the most intense, like a bright primary color (imagine a bright red, blue, or yellow) (4), have a high saturation value. You can get a better idea of these values by looking at the 3D color spaces.





To convert an image from the RGB space to HLS space you must first get Vmax and Vmin

Where:

$$V_{max} \leftarrow max(R, G, B)$$

$$V_{min} \leftarrow min(R, G, B)$$

Next the H channel is calculated using one of the three following equations depending on the value of Vmax

$$H \leftarrow \frac{30(G - B)}{V_{max} - V_{min}}, if\ V_{max} = R$$

$$H \leftarrow 60 + \frac{30(B - R)}{V_{max} - V_{min}}, if\ V_{max} = G$$

$$H \leftarrow 120 + \frac{30(R - G)}{V_{max} - V_{min}}, if\ V_{max} = B$$

The L channel conversion is

$$L \leftarrow \frac{V_{max} + V_{min}}{2}$$

And the S channel using one of the two equations

$$S \leftarrow \frac{V_{max} - V_{min}}{V_{max} + V_{min}}\ , if\ L < 0.5$$

$$L \leftarrow \frac{V_{max} - V_{min}}{2 - (V_{max} + V_{min})}\ , if\ L \geq 0.5$$

3. Chapter 3: Project idea:

Computer Vision is one of the key technologies used in our day-to-day life applications. However, some of these applications require real-time processing for videos and images to perform the intended task, which develop the need for more computing power.

One of the most important computer vision application that requires a real-time processing is the advanced driver assistance systems (ADAS) where the time is a critical factor that needs to be consider. However, the high computing power required for real-time processing is not always practical in such system where the power is a limited and valuable resource.

Therefore, the development of the need for alternative solution to implement ADAS computer vision tasks rather than the traditional implementation on general purpose CPU.

In this chapter after explaining the parameters that we want to optimize in our design we are going to see how the LWDS is implemented and what is the main characteristics of the proposed solution.

3.1. Design Parameters

ADAS is like any system restricted by the environment and operating condition and needs to be optimized taking these conditions into consideration for the ADAS system the operating condition is characterized by two important features which are:

- **Critical Execution time**: Where the ADAS task's main goal is to reduce the number of accidents and to provide more save and easy driving. Therefore, the timing in detecting error or taking action during the drive is very critical and any delay may cause accidents
- **Limited Power Resource**: where the vehicle is battery-operated system, the power is valuable resource that needs to be used wisely so the system must consider such restriction.

Therefore, in our implementation to the task our choices will always be based on the goal to produce a stable system that maintain a practical execution time with minimum power consumption.

3.2. Previous Work

Recently, Customers accord more importance to safety features rather than the cost of the vehicle. Recently, manufacturers have introduced driver assistance systems such as LDWS for the enhanced safety of the passenger. Moreover, increasing focus on the integration of passive safety systems with active safety systems will aid in the growth of this market. Additionally, driver-safety system manufacturers are collaborating with OEMs to develop low-cost driver assistance system for compact vehicles. This is likely to bring about a significant reduction in the price of safety systems, thereby resulting in their widespread implementation of driver assistance systems in low-cost vehicles during the forecast period. Therefore, a consistent increase in demand for compact and mid-sized vehicles equipped with advanced safety features in markets will spur the growth prospects of this market in the coming years. Technavio’s market research analysts have predicted the global LDWS market to grow at an impressive CAGR of more than 28% until the end of 2020.

In this industry research report, the analysts have estimated factors integration of LDWS with other new-age automotive technologies to aid in this market’s growth over the forecast period. To make the most out of LDWS technology, auto manufacturers are focusing on collaborating with automotive safety experts and automotive camera manufacturers to integrate radar and cameras in vehicles leading to the integration of video sensing, radar sensing, and data fusion technologies into a single module. Moreover, as driver assistance and LDWS are becoming increasing common and a standard for automobiles, a fusion of image and radar sensors will help reduce costs in the coming years. For instance, Delphi combines radar sensing, vision sensing, and data into its single module called the Radar and Camera System (RACam). Therefore, the integration of various sensors for enhancing passenger and vehicles safety will impel the growth prospects of this market until the end of 2020.

The common LWDS uses a camera system to look ahead of the vehicle while it is being driven down the highway. Several companies have developed smartphone applications, which use the camera on the devices, including iOnRoad and MinTron. Additionally, there are numerous LDWS installed in production passenger cars, but they have not been implemented into heavy vehicles. PSA and

Bosch are co-developing an infrared LDWS for passenger cars, but currently are not expected to extend licensing to heavy vehicles or other OEMs.

In 2013, many manufacturers offered LDWS, ranging from integrated systems to aftermarket kits and smart-phone applications. Below is a list of 10 systems, followed by an overview of some of their capabilities.

- Bendix – AutoVue - formerly known as Iteris by Audiovox
- Meritor WABCO – SafeTraK – Takata (now called OnLane)
- Mobileye – ADAS C2-270 (also 560)
- Continental LDW
- Delphi Forewarn LDW
- Bosch LDWS
- PSA Peugeot - Infrared LDWS
- iOnRoad – iPhone Application
- MinTron – iPhone Application
- Surveillance Video LDWS - BLS-3000

Auto Vue is a vision-based LDWS that applies a wide-angle camera to view and track visible lane lines. Its on-board computer calculates the vehicle position in the lane and alerts the driver if the vehicle begins to drift out of the lane without applying the turn signal in that direction. The audible warning sounds like a tire passing over a highway rumble strip. This is a driver’s aid for maintaining the vehicle in the lane, but does not intervene in the driving function. AutoVue is in use at Daimler and is being tested for potential applications at another large-vehicle OEM.

The AutoVue LDWS is designed to work in most weather conditions, whether daytime or nighttime, rain or fog, as long as lane markings are visible to the camera. The LDWS tracks both solid and dashed lane lines. The AutoVue goes into a disabled mode if the lines are covered over or not visible and then it warns the driver with an orange reduced-function light.

The AutoVue is optimized to reduce false alarms by disabling alarms when the turn signal is applied and when the vehicle speed is less than 37 mph (59.5 km/h). The warning sound is maintained if vehicle drift occurs in the direction opposite to the intended turn.

Another solution is the Continental LDWS uses a camera mounted on the front of the vehicle to monitor the road up to 40 m (131 ft) ahead of the vehicle. If the vehicle unintentionally drifts out of the lane, the Continental LDW will warn the driver through acoustical sounds and haptic vibrations. The passive LDW can be made active for steering intervention in lane keeping assistant (LKA) mode, which activates above 37.3 mph (60 km/h). Continental began supplying LDWS to European passenger car and heavy-vehicle markets in 200. Continental LDW was introduced into the American market in 2013(7).

3.3. Proposed Solution

The proposed Solution is an image-processing visual lane tracker that alerts the driver if the vehicle unintentionally departs from the lane. It detects visible lane lines in front of the vehicle and provides audible, tactile, or visual alerts.

The proposed solution tries to optimize the execution time and power consumption of the image processing tasks that the system is carrying out.

In this chapter through the following sub-sections, we will try to explain how we are going to achieve the previously mentioned goal.

3.3.1. The need for alternatives

As previously mentioned, the ADAS require real-time capabilities with low power consumptions. Therefore, the task of choosing a platform to deploy our algorithm on was an interesting task and a very critical one to reach our goals.

The previously mentioned solution and in most of computer vision application nowadays the general-purpose CPU is an obvious and available platform. However, this solution is not practical in the ADAS because of the constraints of the performance and power.

Another alternative to the CPU is graphical processing units, which will guarantee a high real-time performance. However, the GPU are also known for its high-power consumption, which does not suit our goals or constraints.

Therefore, we chose to use Field Programmable Gate Array (FPGA) which can optimize our performance with minimum power consumption(7).

3.3.2. FPGA

Field Programmable Gate Array (FPGA) is a reconfigurable integrated circuit. Its parallel computational architecture and convenient access to local memories make it the most appropriate platform for driver assistance system. An FPGA is able to perform real-time video processing such that it could issue corresponding warnings to the drivers timely. Besides, the cost and power consumption of modern FPGAs are relatively low, compared to CPU and GPU(7).

- **What is FPGA?**

FPGAs are off-the-shelf programmable devices that provide a flexible platform for implementing custom hardware functionality at a low development cost. They consist mainly of a set of programmable logic cells, called configurable logic blocks (CLBs), a programmable interconnection network, and a set of programmable input and output cells around the device. In addition, they have a rich set of embedded components such as digital signal processing (DSP) blocks, which are used to perform arithmetic-intensive operations such as multiply-and-accumulate, block RAMs (BRAMs), look-up tables (LUTs), flip-flops (FFs), clock management unit, high speed I/O links, and others(7).

- **Why using FPGA?**

There have been different approaches in the literature to implement vision algorithms in embedded systems. For many years, only microcontrollers and microprocessors were used, due to their programmable functionalities. There are many examples in the literature of vision systems with a microcontroller or microprocessor implementation.

A similar approach is the use of digital signal processors (DSPs). DSPs can do single cycle multiply and accumulation operations and have some parallel processing capabilities, which enhance processing speed. Traditionally, DSPs have been used in image and audio signal processing when the use of microcontrollers was not enough.

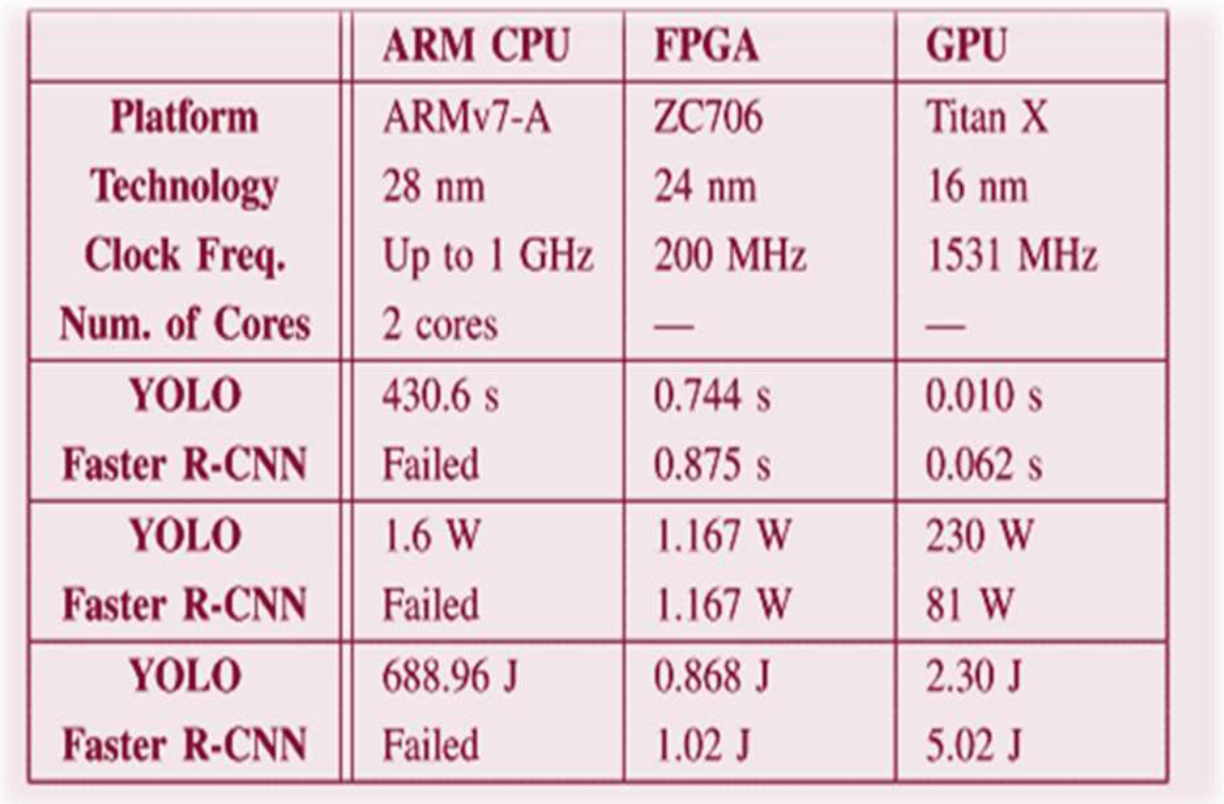
However, the increase in resolution and frame rate in recent cameras makes it difficult to achieve real-time performance using only microprocessors or DSPs. Hardware implementation represents an alternative solution, since it can achieve a much better computational performance. During last years, there have appeared a big number of implementations in Application-Specific Integrated Circuits (ASIC) or field-programmable gate arrays (FPGAs). However, we chose to use FPGA for the following reasons:

- FPGAs have an important advantage over ASICs: they are reconfigurable, which gives them some of the flexibility of software.
- FPGAs have the advantage of maximizing performance per Watt of power consumption, reducing costs for large-scale operations. This makes them an excellent choice as accelerators for battery-operated devices, which suits our application.
- The adoption of software-level programming models such as the open computing language (OpenCL) standard or MATLAB, in FPGA tools made them more attractive to use for fast developing in short timeline applications.
- FPGAs offer a clear advantage as they can create customized hardware circuits that are deeply pipelined and inherently multithreaded which give is a real-time processing capability.

For a case study in the following table, we can see how the FPGA implementation of YOLO algorithm has improved the efficiency of the algorithm by minimizing the time of execution maintaining a low power consumption rate where we can see the following:

- The Fastest implementation of the algorithm is the GPU implementation but it consumes the most power.
- The CPU (traditional) implementation uses less power than GPUs but with Higher Execution time.

- The FPGA implementation has a very good execution time with less power Consumption than the GPU and the CPU(7).



	ARM CPU	FPGA	GPU
Platform	ARMv7-A	ZC706	Titan X
Technology	28 nm	24 nm	16 nm
Clock Freq.	Up to 1 GHz	200 MHz	1531 MHz
Num. of Cores	2 cores	—	—
YOLO	430.6 s	0.744 s	0.010 s
Faster R-CNN	Failed	0.875 s	0.062 s
YOLO	1.6 W	1.167 W	230 W
Faster R-CNN	Failed	1.167 W	81 W
YOLO	688.96 J	0.868 J	2.30 J
Faster R-CNN	Failed	1.02 J	5.02 J

4. Chapter 4: project implementation

4.1. Control System

We have five inputs:

- **S** refer start.
- **L** is out of lane detection algorithm; it is referring to car in left deviation case.
- **R** is out of lane detection algorithm; it is referring that the car in right deviation case.
- **R-sig** is input from right signal of the car.
- **L-sin** is input from left signal of the car.

Start initiating the system by pressing the S button, which is a physical user control button this initiate the system by starting the camera module and capture the road scene then passing it to the lane detection sub-system that return a signal which indicate the state of the vehicle.

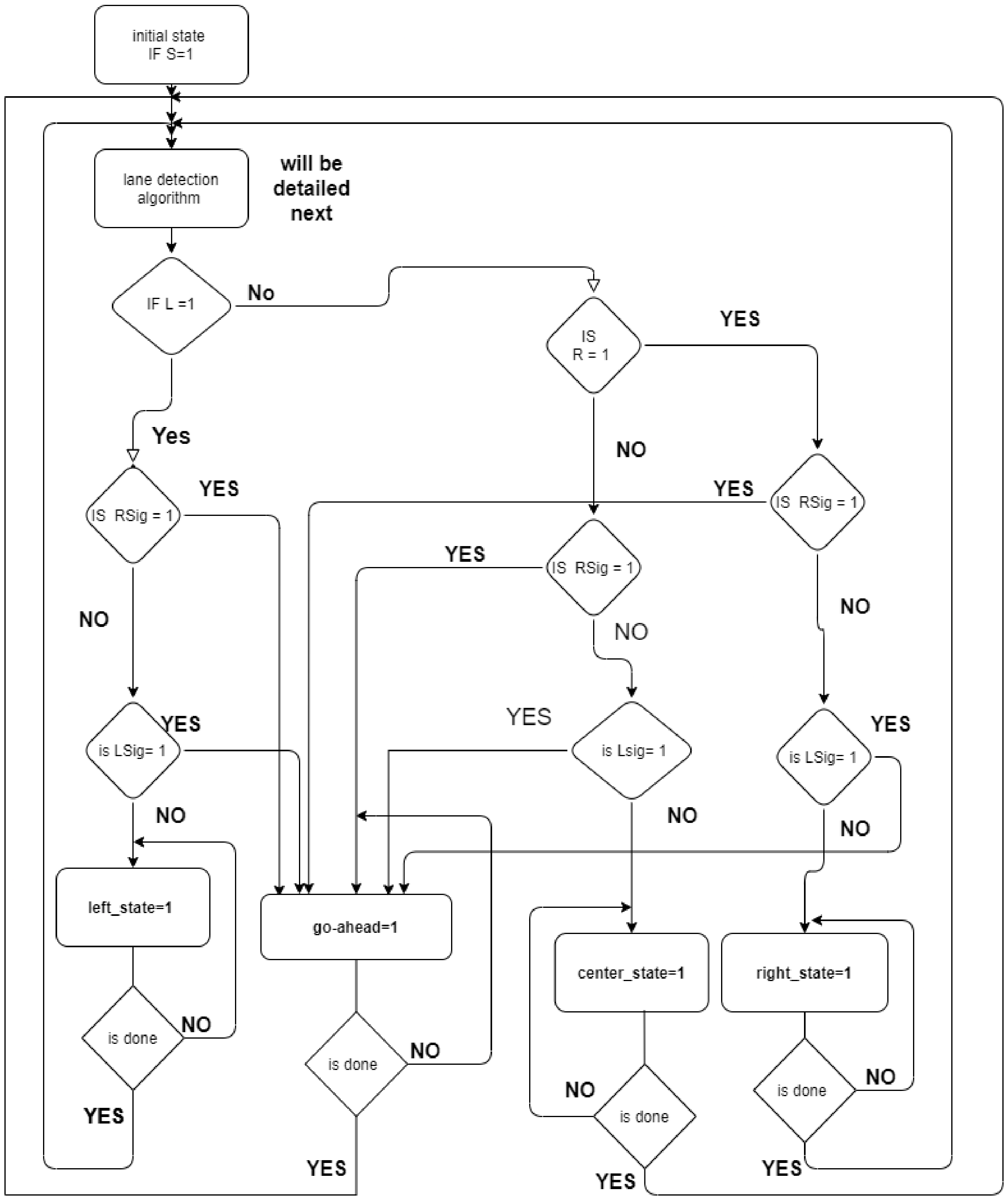
The vehicle has three states:

- Left deviation state (L): which indicate that the care has deviated from the lane to the left side.
- Right deviation state (R): which indicate that the care has deviated from the lane to the right side.
- Center state: indicates that the car is centered in the lane.

Then we take the output of the Lane detection sub-system combined with the inputs from the driver that may indicate the he is going to take a right turn (R-sig) or a left turn (L-sig) to enter a decision tree where action is taken based on the state of the vehicle.

The system is designed with alerting system that is equipped with LCD to display warning message, warning LEDs and vibrating elements to warn the driver in case the vehicle is in deviation state.

4.1.1. Flow chart of the control system:

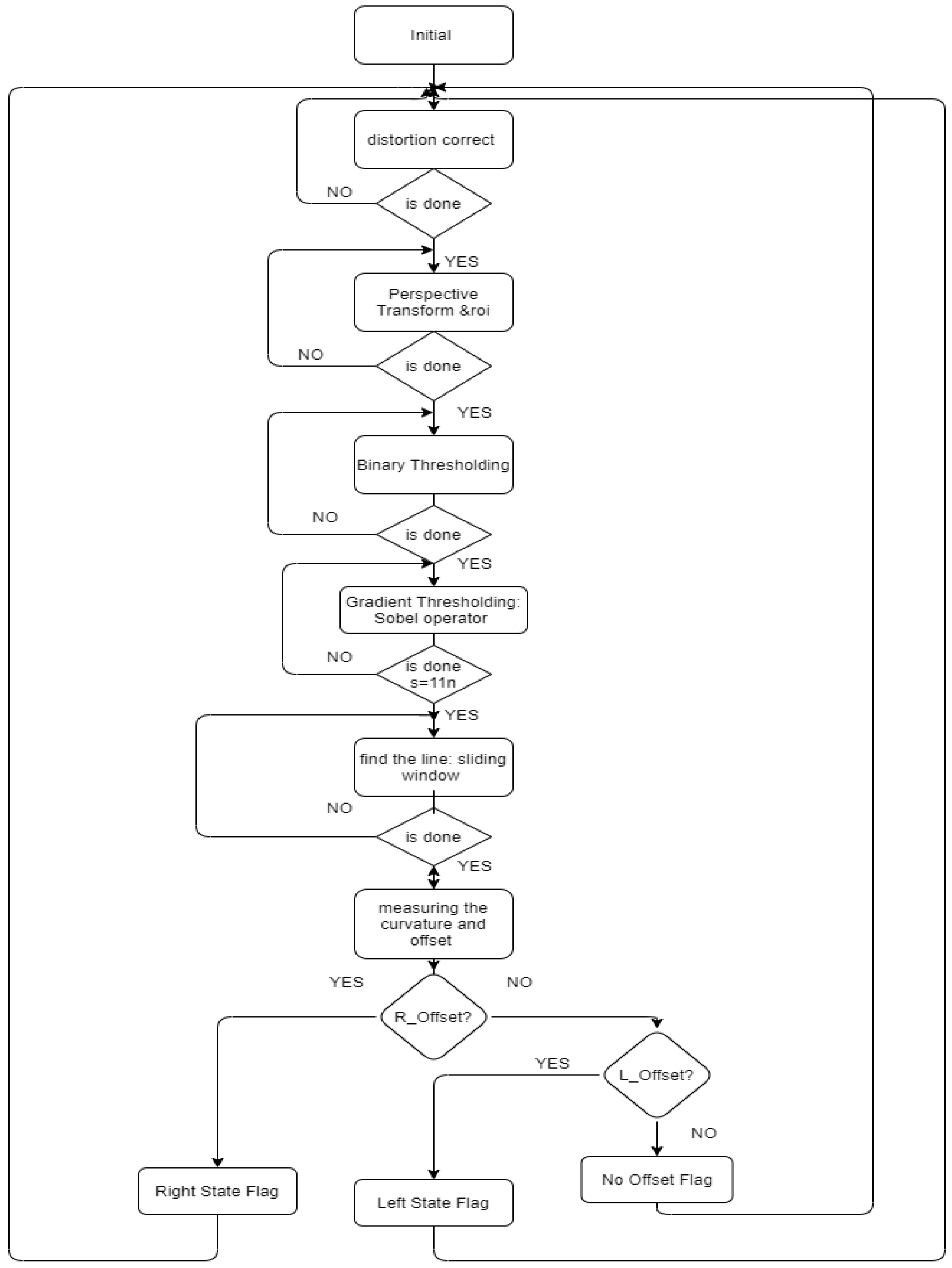


4.1.2. LDW algorithm:

In LDW algorithm, our goal is to take every frame of the road input video and process it to find the lane position in the fame and calculate the car position to the lane.

The algorithm is divided to the following steps:

- **Camera Calibration**: Distortion takes place because of camera, which captures wrong dimensions. Therefore, the aim of this stage is to find the calibration matrix of the camera and use it to remove the distortion from the image. (to find more on the calculations and how this happens you can review the fundamental chapter)
- **Perspective transform**: the camera module is mounted on the front shield of the car so the taken picture need to be transformed to the eye-bird view first before the next stages
- **Edge detection**: to find the edges of the possible lanes in the frame.
- **Binary thresholding**: to convert the frame to a binary frame with pixels either 0 or 1.
- **Find the lanes**: this stage identifies the pixels, which belongs to the lane lines, and the pixels, which are not.
- **Measuring the road curvature and the offset of the car**: this is the final stage of the algorithm where we decide if the vehicle is keeping its lane or not.


Initial
distortion correct
is done
NO
YES
Perspective Transform &roi
is done
NO
YES
Binary Thresholding
is done
NO
YES
Gradient Thresholding: Sobel operator
is done s=11n
NO
YES
find the line: sliding window
is done
NO
YES
measuring the curvature and offset
R_Offset?
YES
NO
L_Offset?
YES
NO
Right State Flag
Left State Flag
No Offset Flag

4.2. Implementation on Python:

In this stage we are implementing our previously described algorithm on python to easily experiment and modify the algorithm before go for implement it on MATLAB and FPGA.

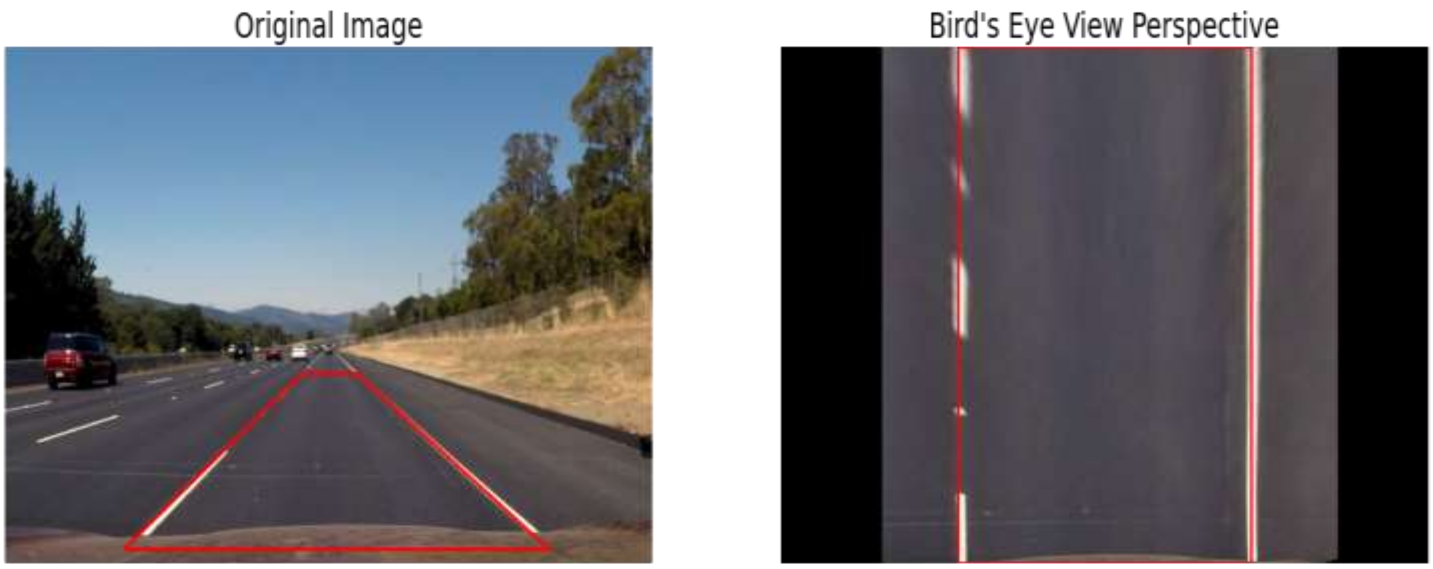
In this section, we will show the results of the python implementation of every stage in the LDW algorithm described in the previous section.

4.2.1. Camera Calibration & Distortion correction:

Apply a distortion correction to make image undistorted see chapter 2 for the mathematical foundation of how this is done.

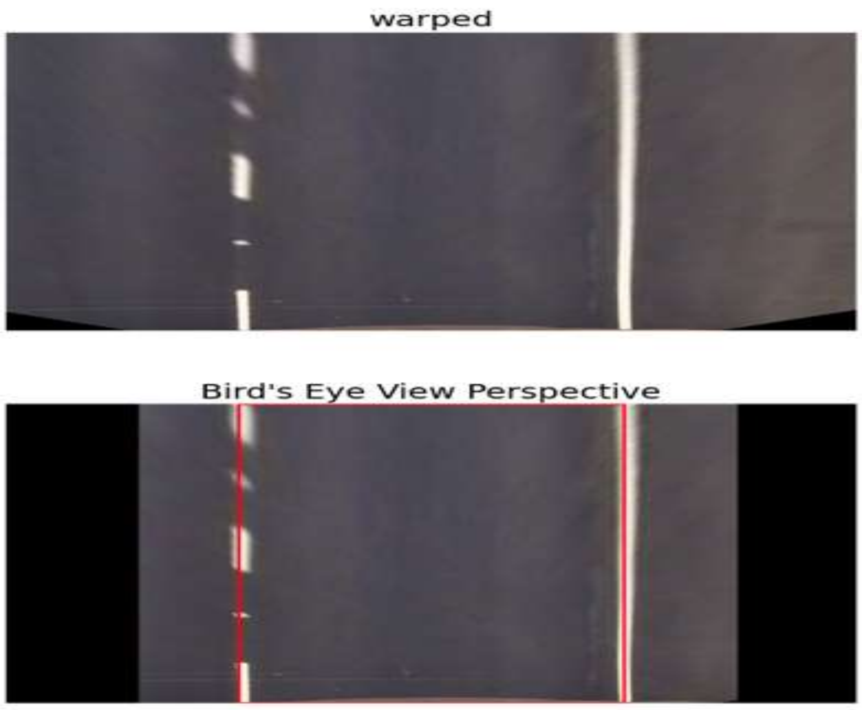
4.2.2. Perspective Transformation:

A perspective transform maps the points in a given image to different, desired, image points with a new perspective. The perspective transform you will be most interested in is a bird’s-eye view transform that let us view a lane from above; this will be useful for calculating the lane curvature later on. Aside from creating a bird’s eye view representation of an image, a perspective transform can also be used for all kinds of different viewpoints.



4.2.3. Region of interest (ROI)

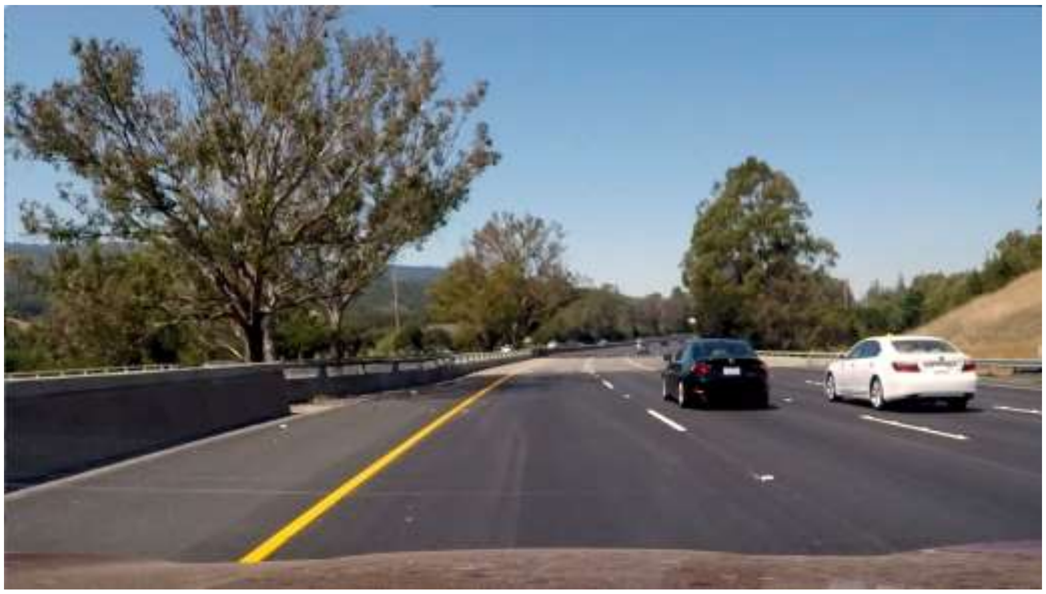
Transforms an image by preserving only the ROI represented by the the 'vertices' and removes the remainder of the image by setting the pixel intensity to 0.



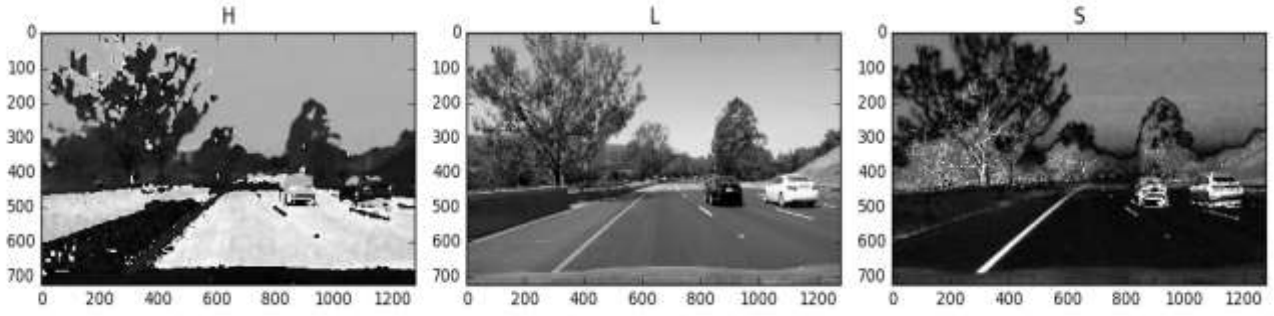
4.2.4. Generate Threshold Binary image:

In this stage our output is as described in the algorithm description section is the input image converted to the HLS space that gives us a robust output from thethresholding stage because it mentain information about colors in the pictures other color spaces loses. Therefore, it can help us more with photos taken in the different lighting conditions.

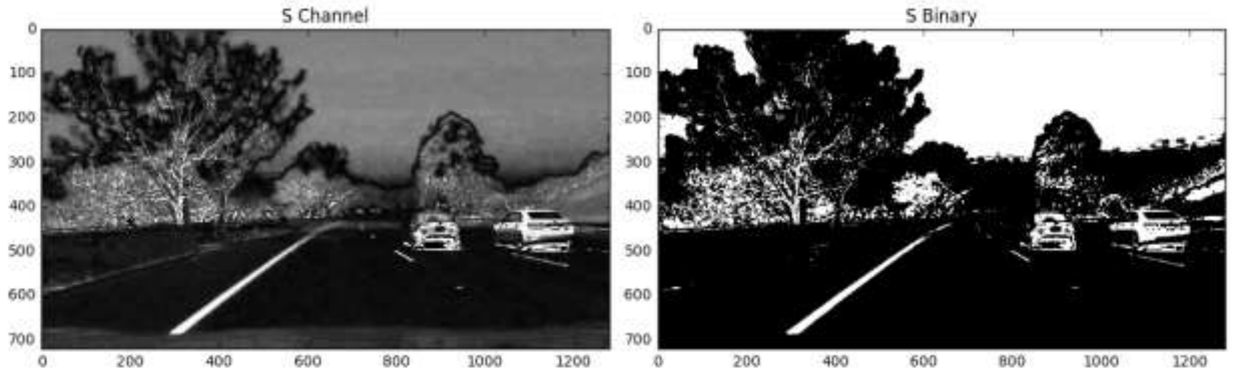
Next is some expermints to find out what is the best channel we can use for thresholding with the following input image.



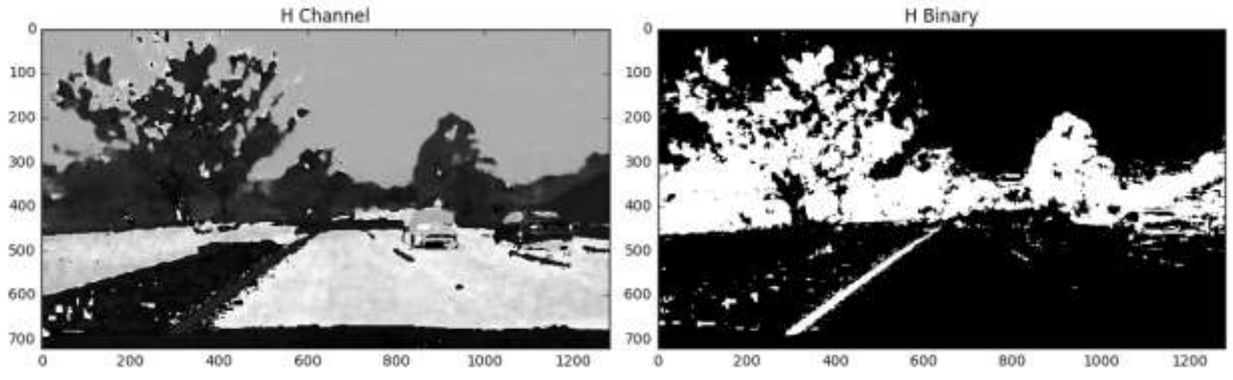
Converting it to the HLS space



Noticing that the S channel is picking up the line pretty well so we apply the thresholding on it to get the following output



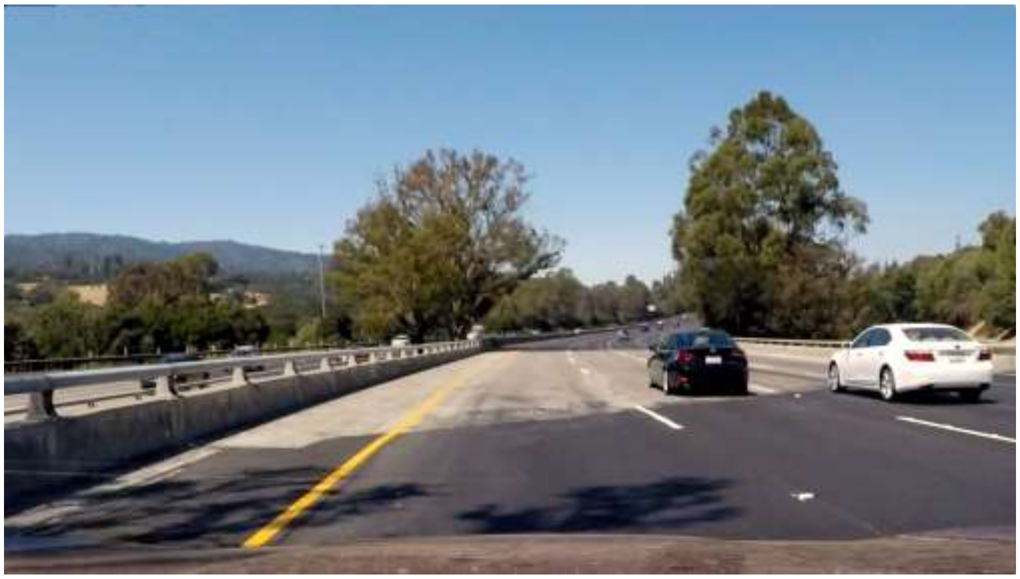
You can also see that in the H channel, the lane lines appear dark, so we could try a low threshold there and obtain the following result:

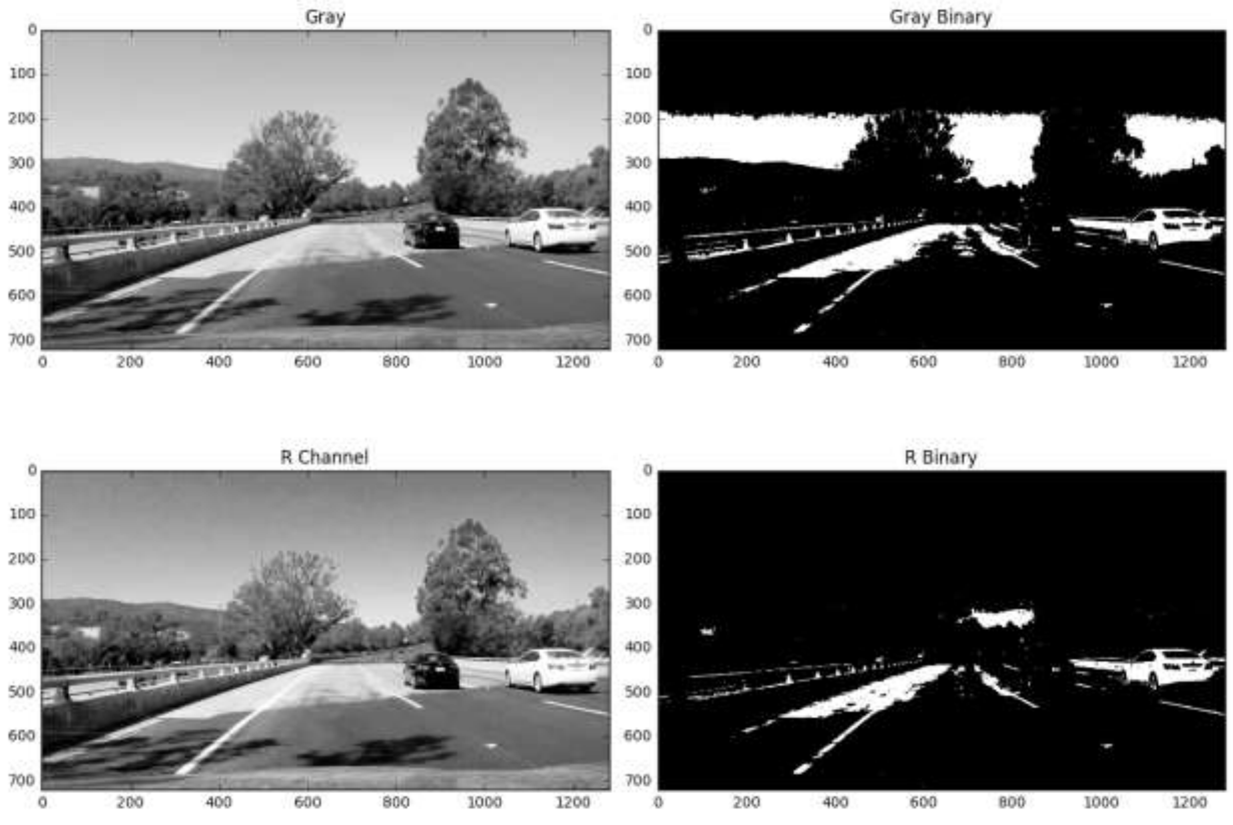


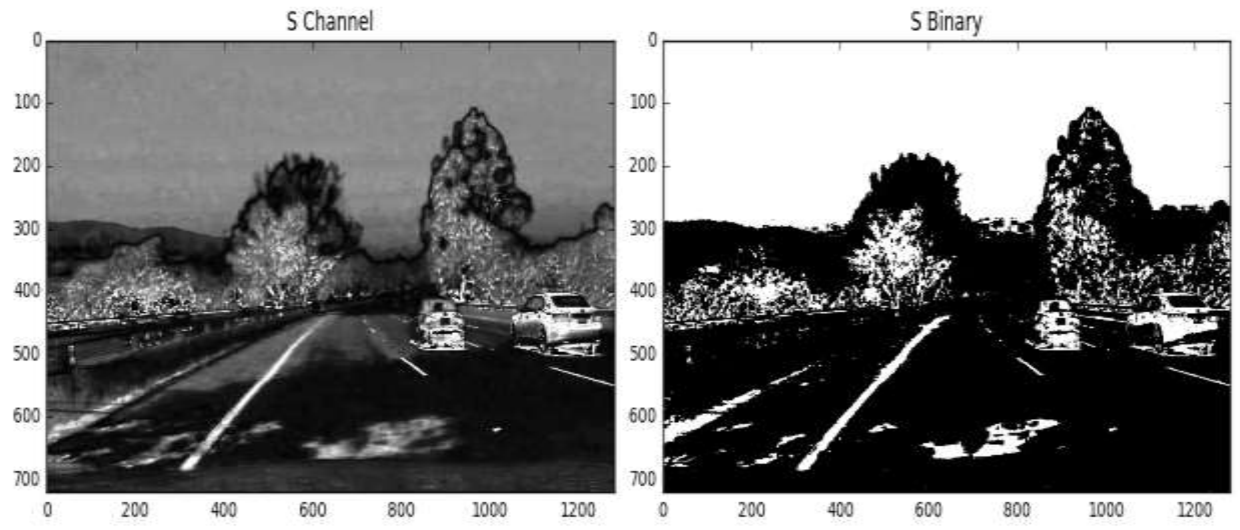
From these examples, you can see that the S channel is probably the best bet. However, it is not clear that one method is far superior to the others.

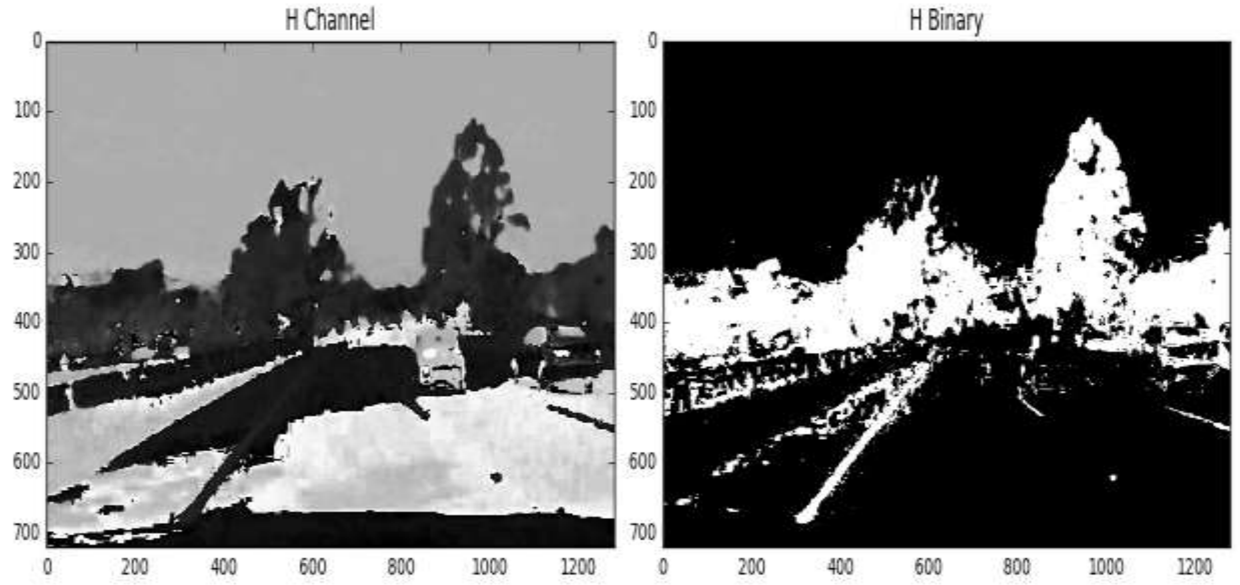
In each case, we have tweaked the threshold parameters to do as good a job as possible of picking out the lines. Where we can really see a difference in results, however, is when we step to a new frame, where there are shadows and different colors in the pavement.

Look at the same thresholds applied to each of these channels for this image:









Now you can see that, the S channel is still doing a robust job of picking up the lines under very different color and contrast conditions, while the other selections look messy. You could tweak the thresholds and get closer in the other channels, but the S channel is preferable because it is more robust to changing conditions.

4.2.5. Applying Sobel

This stage of the program is the responsible for the detection of edges in the image and we use this to find the edges of the lane lines (you can know more about how this is done by review the fundamentals chapter).

The following is the output of this stage in the program.

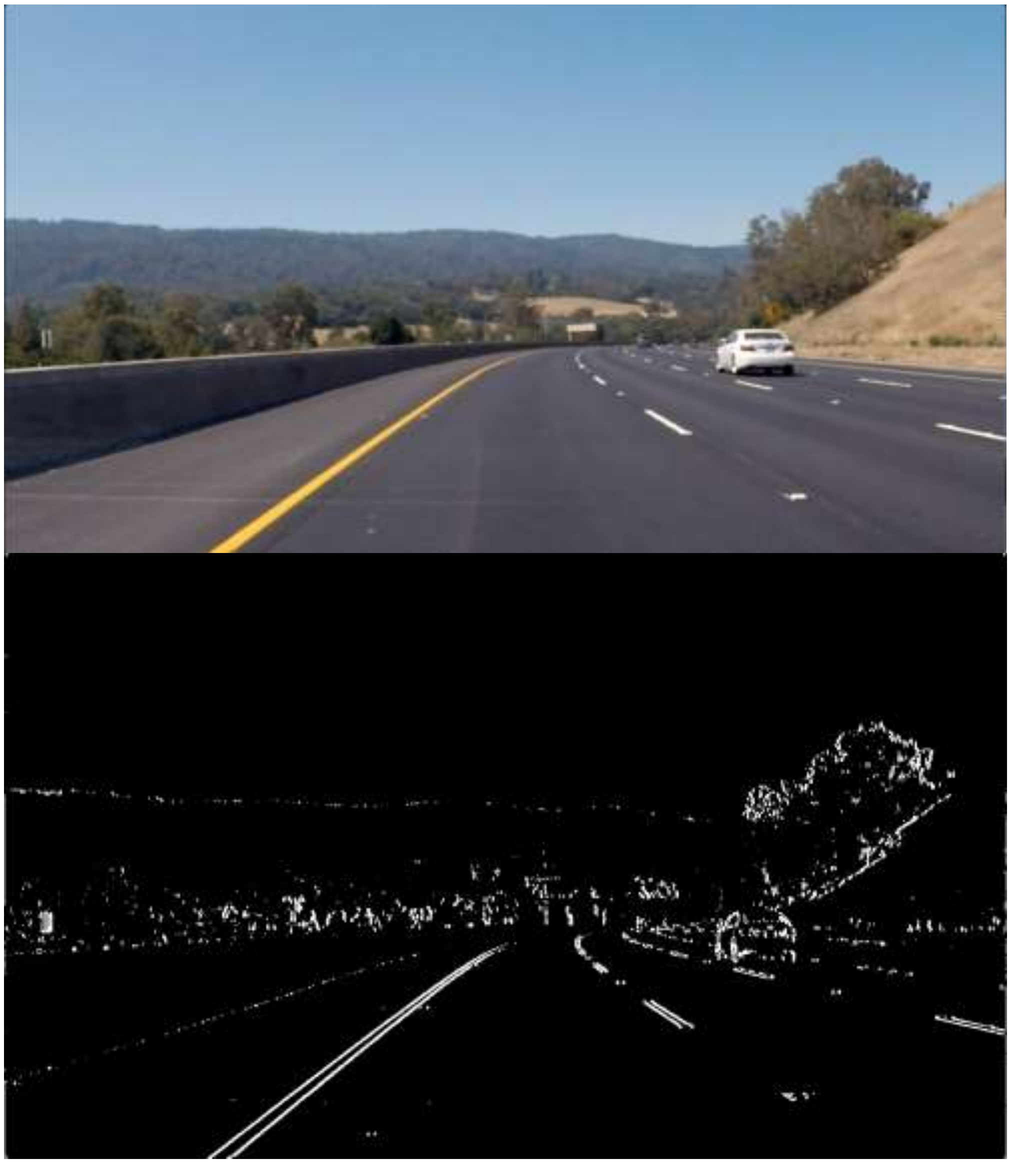
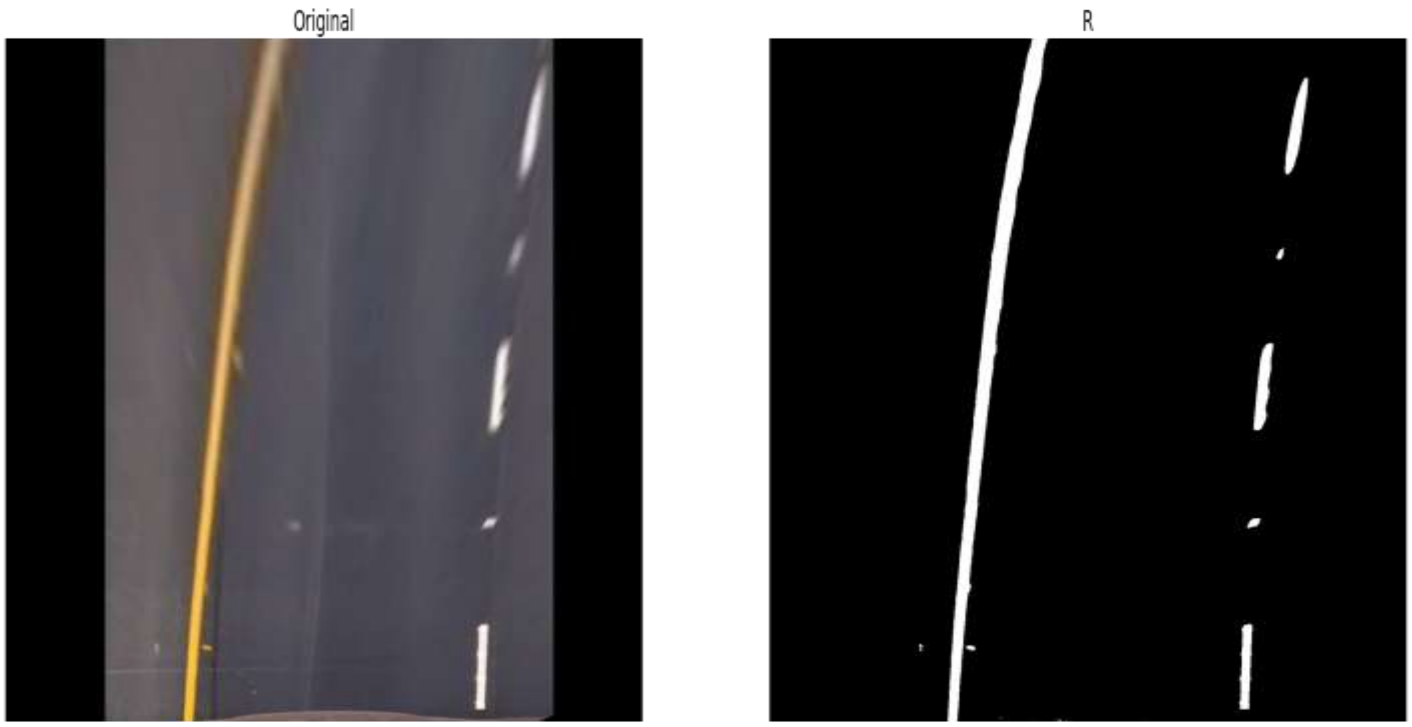


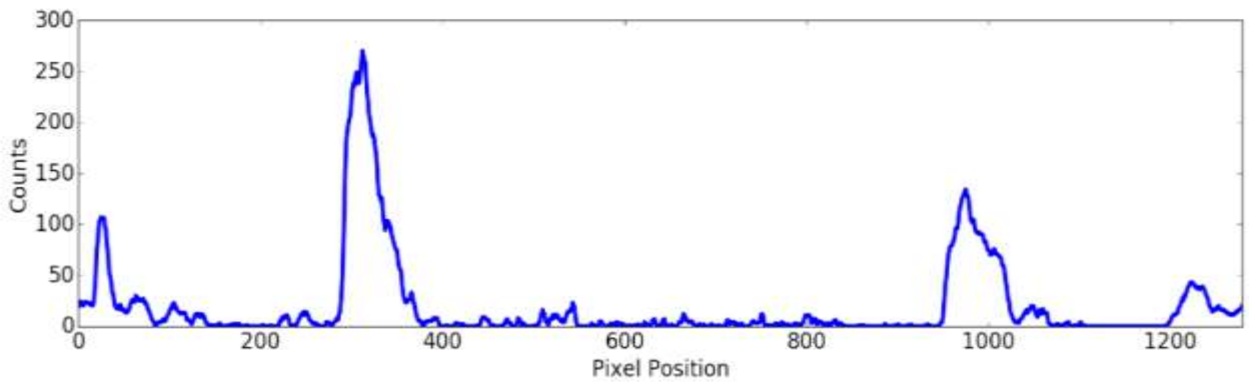
Figure below shown that applying Sobel edge detection after applying bird's eye and ROI:



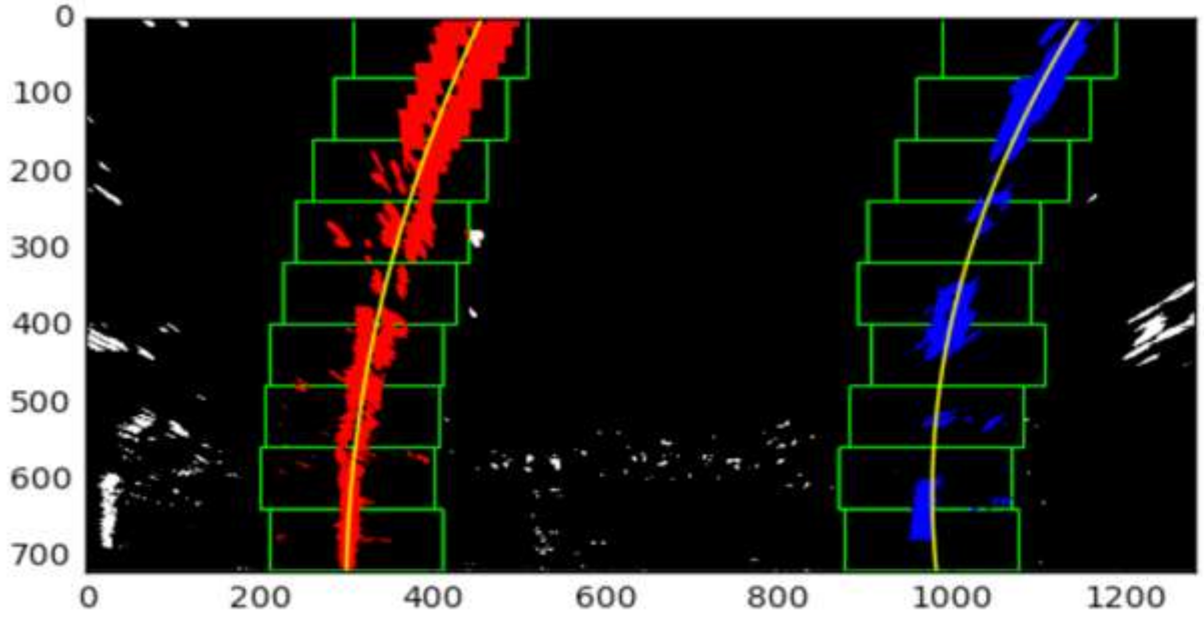
4.2.6. Detect Lane Lines: Peaks in Histogram & Sliding Window Technique:

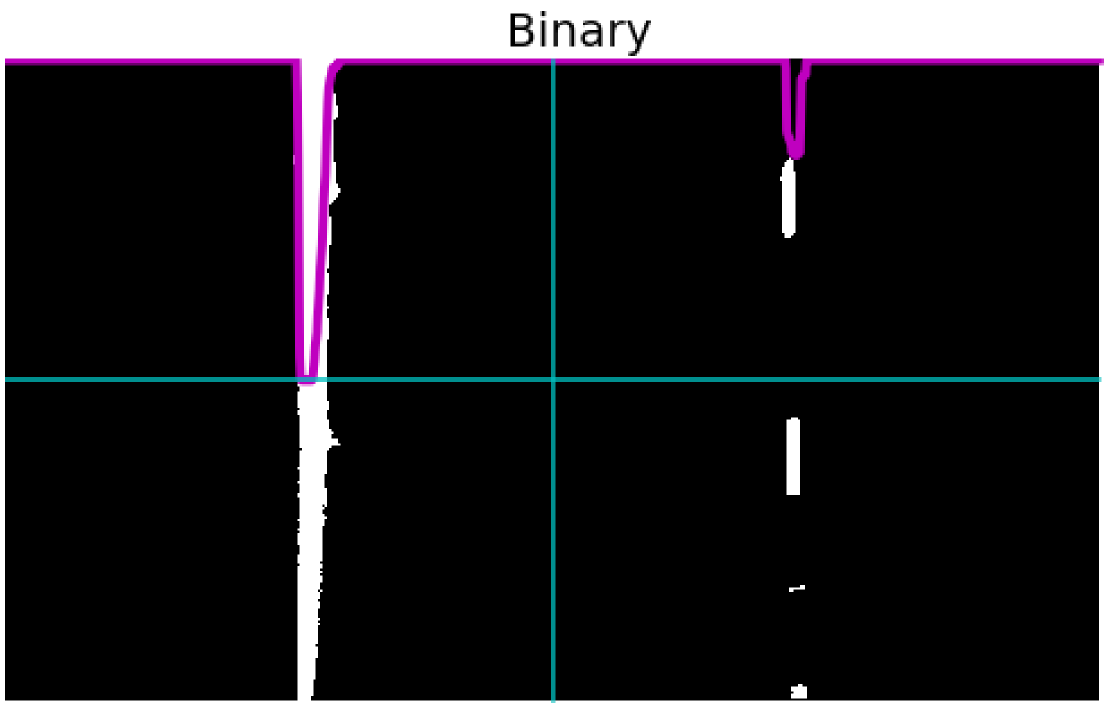
At this point we have a binary image with the lane lines are clear. Therefore, the next step is to decide which pixels are belong to the lane lines

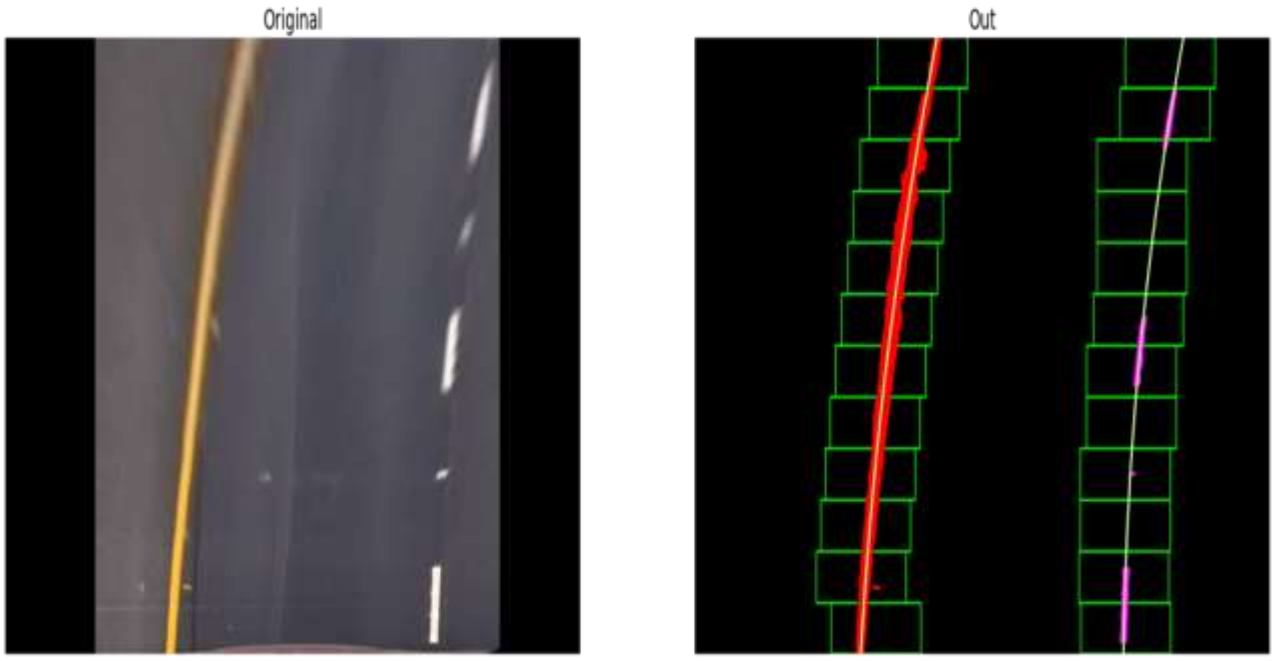
Histogram and sliding window technique is firstly plotting a histogram of where the binary activations occur across the image as following:



Next, we use the two highest places as starting points of our lane lines to produce an image with the lane lines pixels are detected as follows:

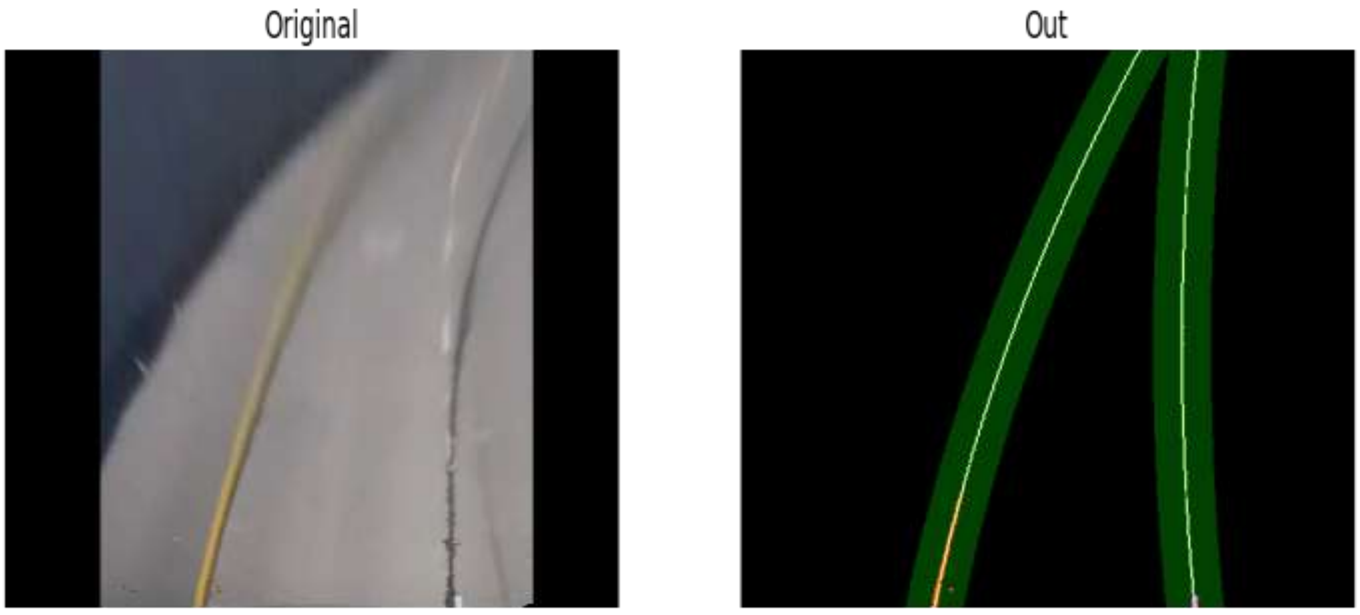




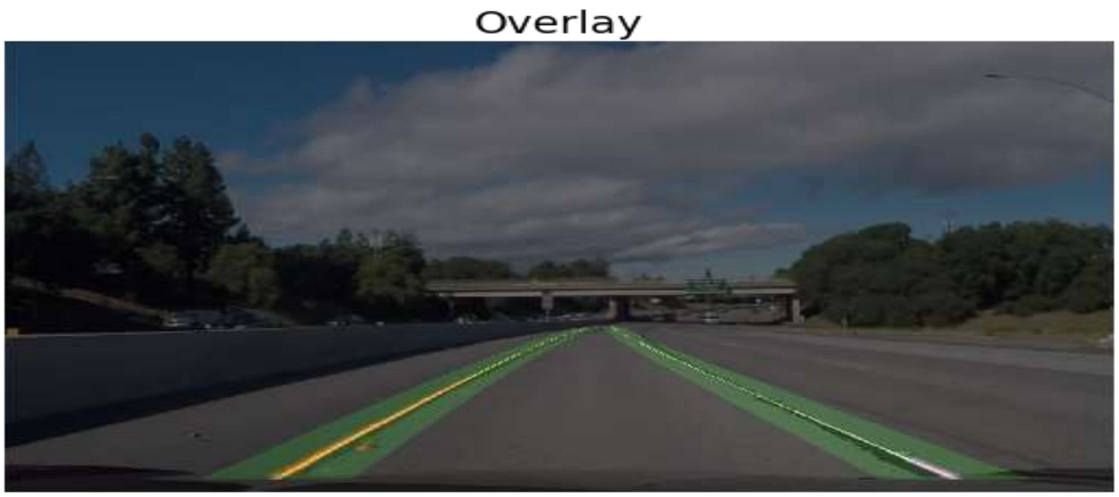


1. Detect Lane Lines: Adaptive Search:

Uses the sliding window technique to perform incremental localized adaptive thresholding over the previously detected lane line trajectory to develop a thresholder binary image.

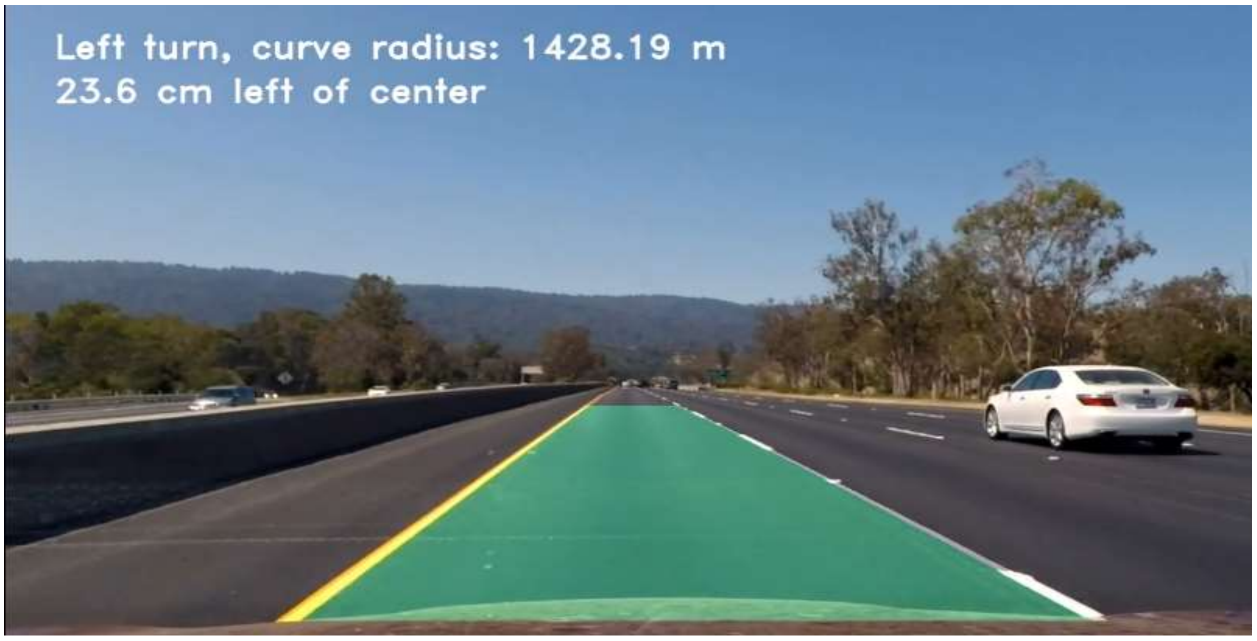


Then uses this generated binary image to detect and fit lane lines in a margin around the previous fit rather than performing a blind search.



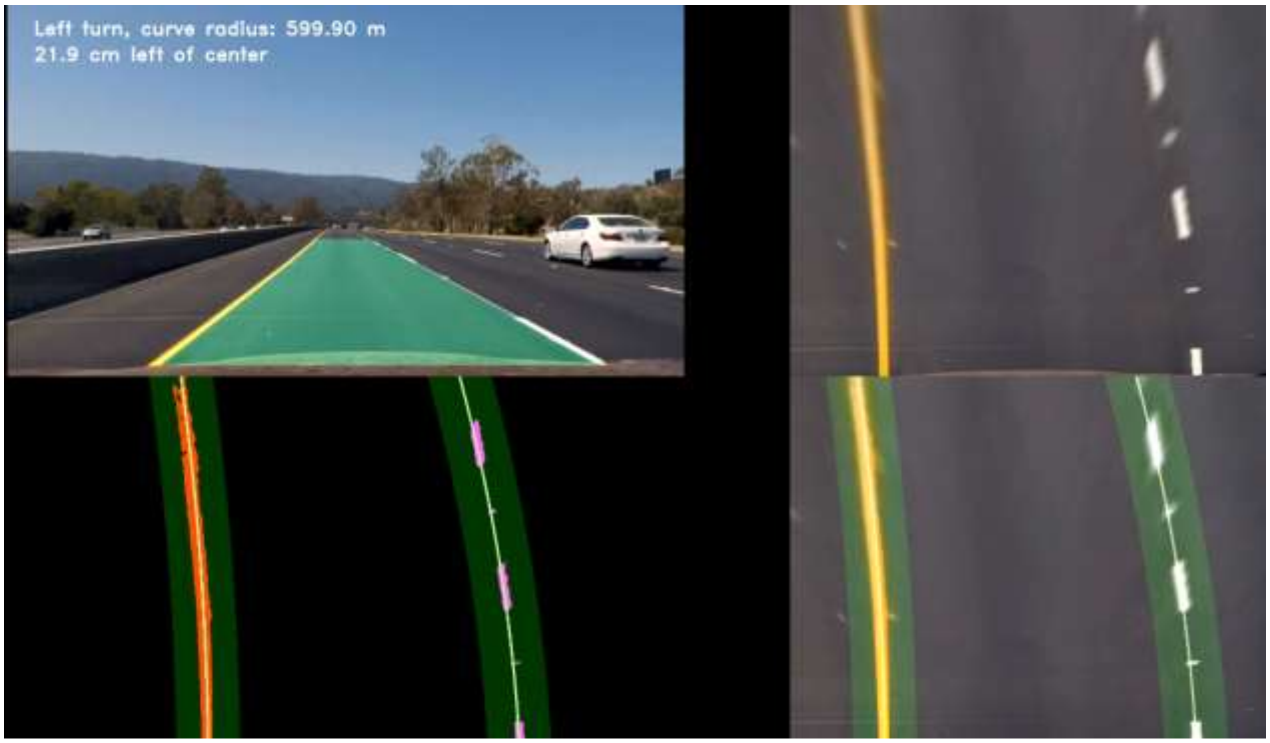
4.2.6. Compute Lane Line Curvature

Determine the curvature of the lane and vehicle position with respect to center, Warp the detected lane boundaries back onto the original image. (For more information on how this is done review the fundamentals chapter)



Visualize output

Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position



4.3. MATLAB simulation:

We chose the Mode Based Design (MBD) approach for the implementation of our project:

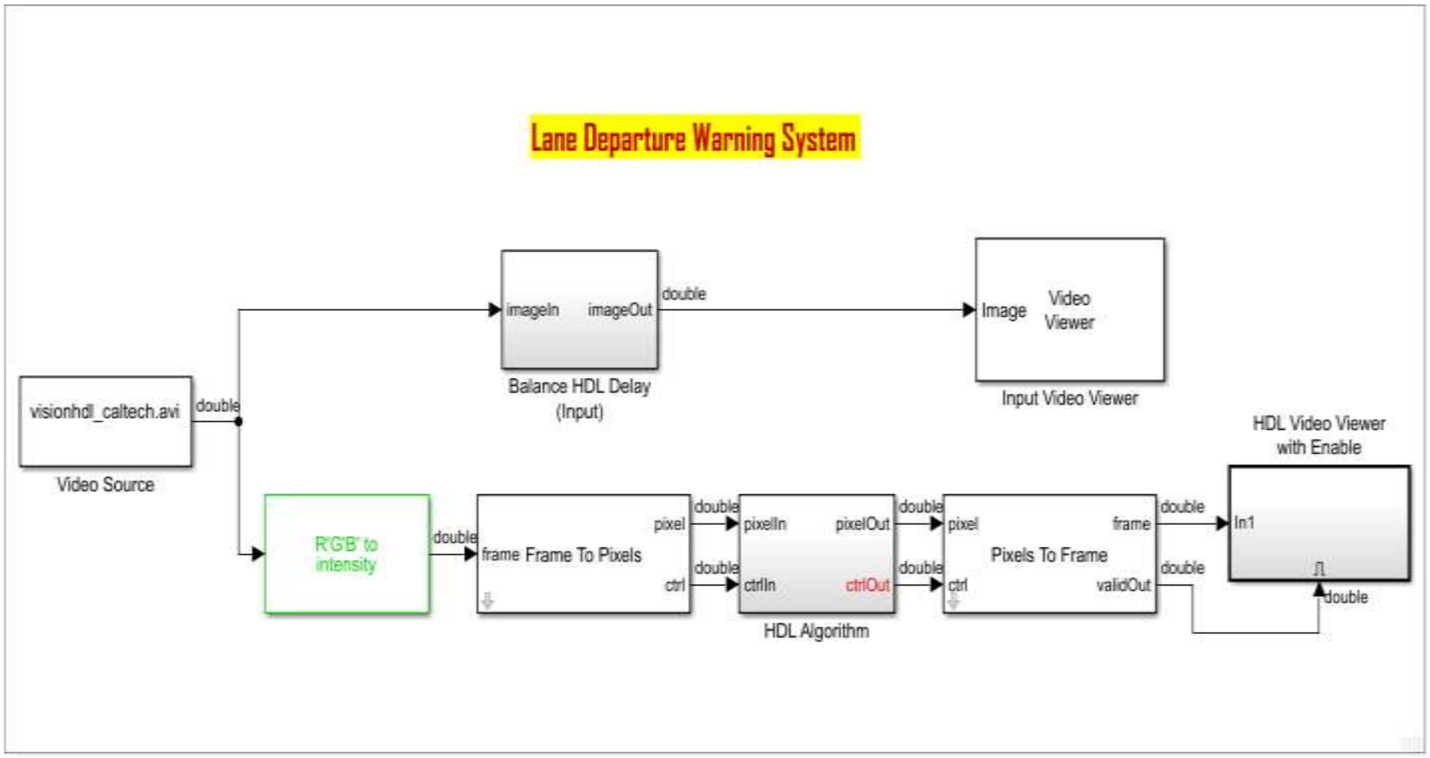
Model-based design (MBD) performs verification and validation through testing in the simulation environment. It covers various disciplines, functional behavior, and cost/performance optimization to deploy a product from early concept of design to final validation and verification testing (5).

Therefore, MATLAB was the tool we choose to model and simulate our designed system; at this section, we are going to explain our model and the result of the simulation.

The building blocks we used are blocks from the Vision HDL tool in Simulink, which are designed to be converted easily to HDL code.

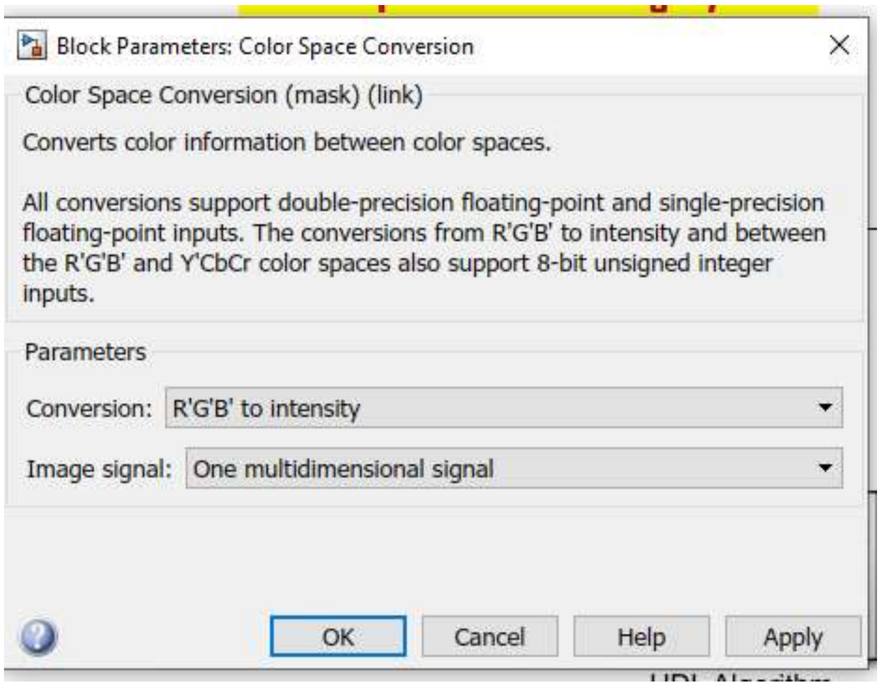
4.3.1. System Model

The input video is converted to a streaming pixel format using the Frame to Pixels blocks, passed through the HDL Algorithm subsystem and then converted back to a frame using the Pixels to Frame block. The model is configured for HDL code generation using the hdlsetup function (5).



The Color Space Conversion block converts color information between color spaces. Use the Conversion parameter to specify the color spaces you are converting between

- R'G'B' to Y'CbCr
- Y'CbCr to R'G'B'
- R'G'B' to intensity
- R'G'B' to HSV
- HSV to R'G'B'
- sR'G'B' to XYZ
- XYZ to sR'G'B'
- sR'G'B' to L*a*b*
- L*a*b* to sR'G'B'


Block Parameters: Color Space Conversion

Color Space Conversion (mask) (link)

Converts color information between color spaces.

All conversions support double-precision floating-point and single-precision floating-point inputs. The conversions from R'G'B' to intensity and between the R'G'B' and Y'CbCr color spaces also support 8-bit unsigned integer inputs.

Parameters

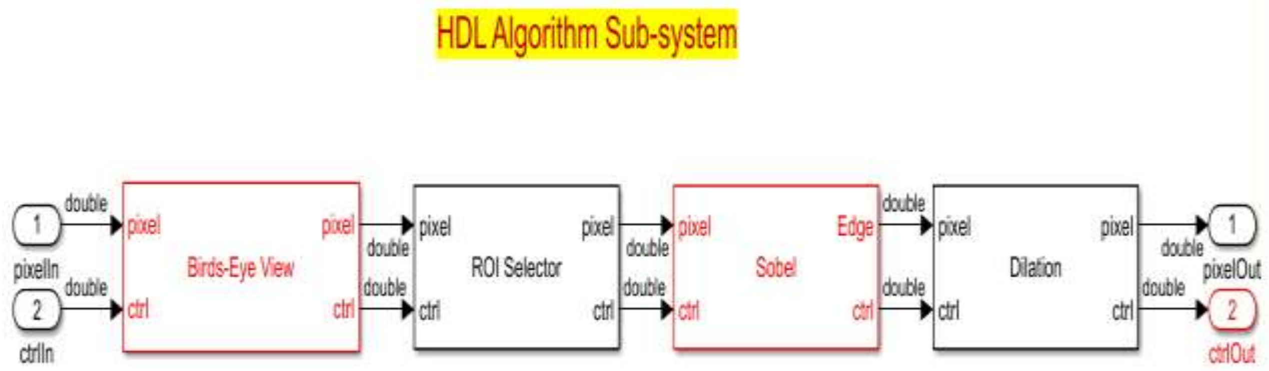
Conversion: R'G'B' to intensity

Image signal: One multidimensional signal

For our algorithm, it was configured to convert from RGB to intensity, which gave us the best results.

After the color space conversion, the frame is passed through a frame to pixels block where video frames is converted to a pixel stream and control signals. The control signals indicate the validity of each pixel and its location in the frame. The pixel stream format can include padding pixels around the active frame. You can configure the frame and padding dimensions by selecting a common video format or by specifying custom dimensions. The pixel stream can support scalar streaming or multi-pixel streaming. Multi-pixel streaming provides 4 or 8 pixels per clock cycle to support high-rate or high-resolution formats.

Then the frame is ready to enter pixel by pixel (or multi-pixel) to the HDL Algorithm sub-system where the calculation are done as shown in the next figure.



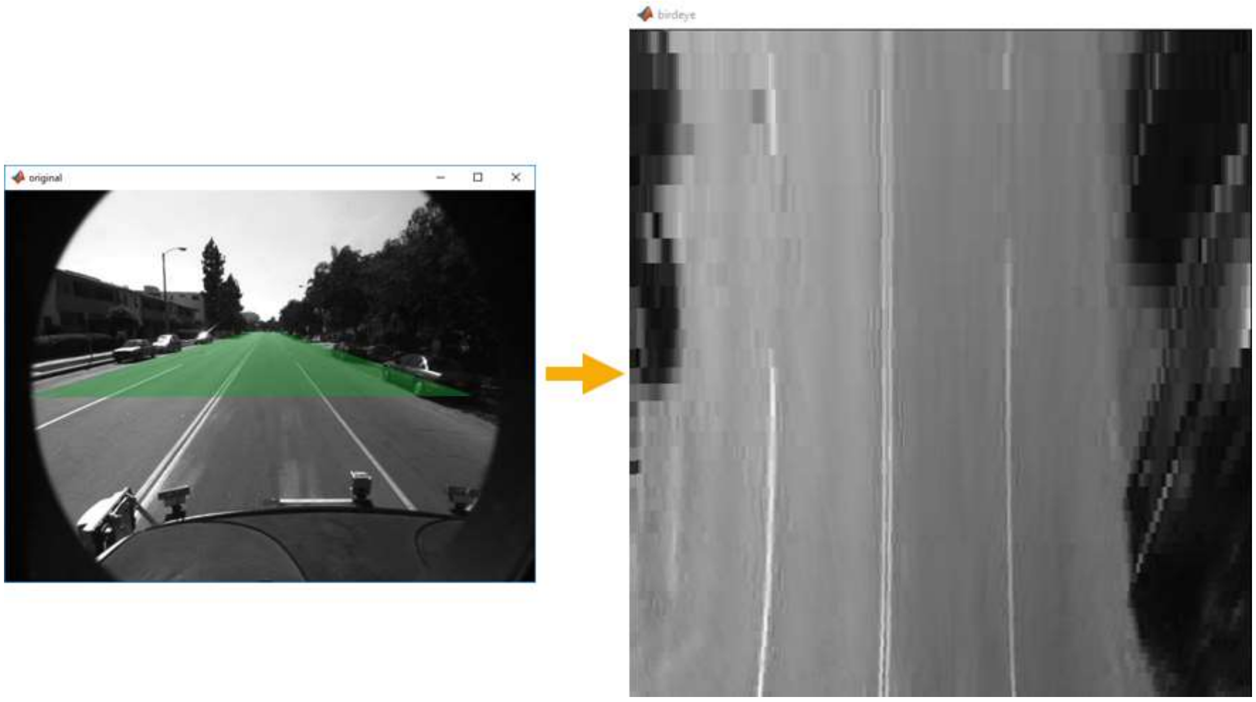
For the remaining of this section we will describe the stages of the HDL Algorithm sub-system and its configurations.

- <u>**Bird-Eye View**</u>

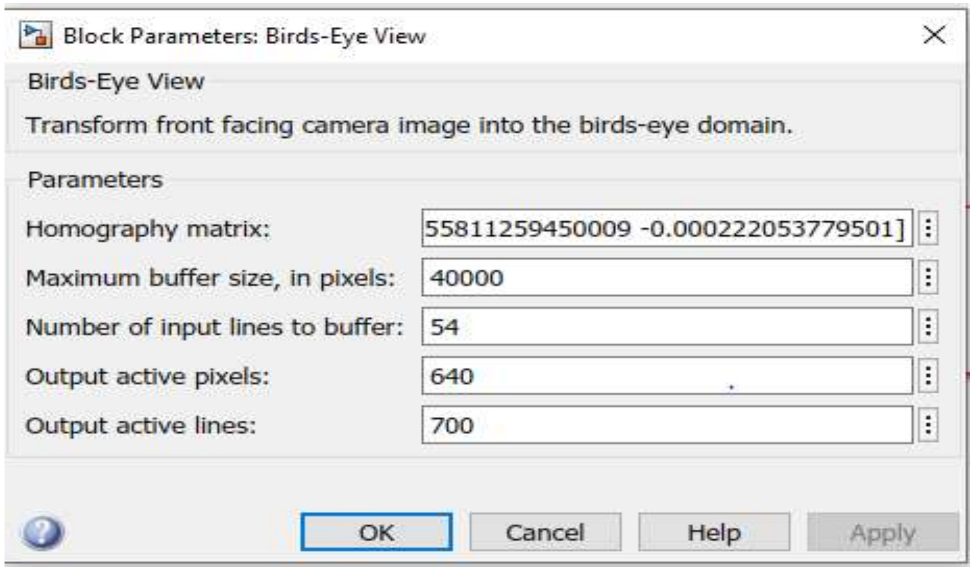
Firstly, we map the view of the front facing video to a Bird-Eye view (to review the mathematical details of this review the fundamentals chapter).

The Birds-Eye View block warps a front-facing camera image into a top-down view. It uses a hardware-efficient architecture that supports HDL code generation. To configure this block we must first calculate the Homography matrix, which is a camera dependent matrix that can be calculated from physical camera properties, or empirically derived by analyzing an image of a grid pattern taken by the camera. The block uses the matrix to compute the transformed coordinates of each pixel. The transform does not interpolate between pixel locations. Instead, it rounds the result to the nearest coordinate. The block operates on a trapezoidal region of the input image below the vanishing point.

These images show the input region selected for transformation and the resulting top-down view (3).



The following image shows our configuration to the sample video we used in the testing.



Block Parameters: Birds-Eye View

Birds-Eye View

Transform front facing camera image into the birds-eye domain.

Parameters

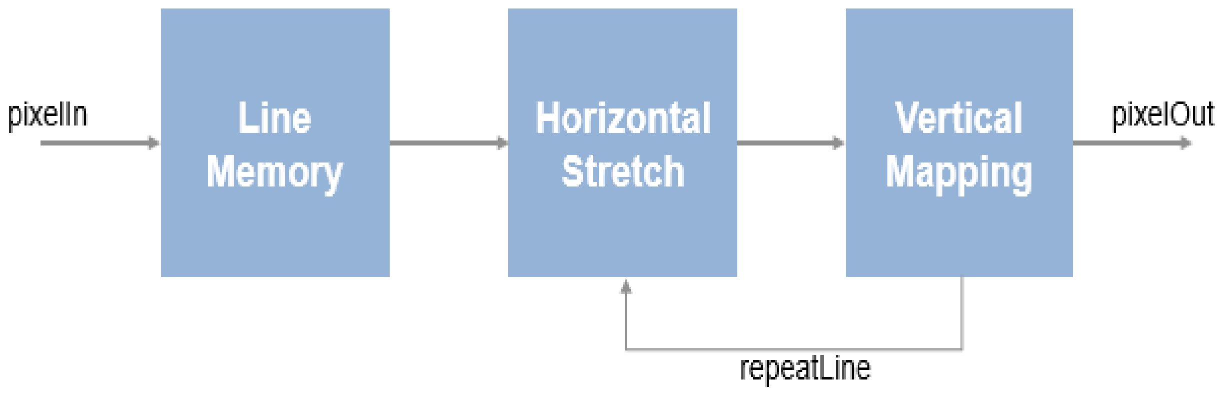
Homography matrix:	55811259450009 -0.000222053779501]
Maximum buffer size, in pixels:	40000
Number of input lines to buffer:	54
Output active pixels:	640
Output active lines:	700

Number of lines to transform, specified as an integer. The block stores and transforms this number of lines into the output bird's-eye view image, starting at the vanishing point as determined by the Homography matrix. Storing the full input frame uses too much memory to implement the algorithm without off-chip storage. Therefore, for a hardware implementation, choose a smaller region to store and transform, one that generates an acceptable output frame size

The implementation of the bird's-eye transform in hardware does not directly perform the calculation mentioned in the fundamentals chapter, which is

$$(\hat{x}, \hat{y}) = round\left(\frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}\right)$$

Instead, the block precomputes lookup tables for the horizontal and vertical aspects of the transform.



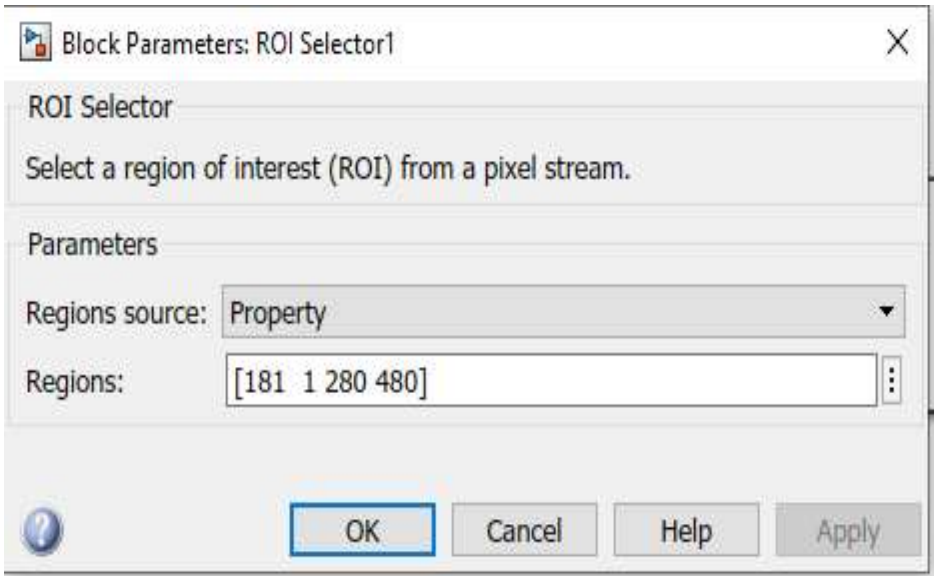
First, the block stores the input lines starting from the precomputed vanishing point. The stored pixels form a trapezoid, with short lines near the vanishing point and wider lines near the camera. This storage uses Maximum buffer size, in pixels memory locations. The horizontal lookup table contains interpolation parameters that describe the stretch of each line of the trapezoidal input region to the requested width of the output frame. Lines that fall closer to the vanishing point are stretched more than lines nearer to the camera. The vertical lookup table contains the y-coordinate mapping, and how many times each line is repeated to fill the requested height of the output frame. Near the vanishing point, one input line maps to many output lines, while each line nearer the camera maps to a diminishing number of output lines (3).

The lookup tables use 3*Number of input lines to buffer memory locations

- **<u>ROI Selector</u>**

The ROI selector is used to choose a portion from the frame to deal with aside from the rest of the frame.

The following figure shows our parameter configuration, where our goal was to select the area at the middle of the frame which contain the information of the lane.


Block Parameters: ROI Selector1

ROI Selector

Select a region of interest (ROI) from a pixel stream.

Parameters

Regions source: Property

Regions: [181 1 280 480]

The four elements that define each region are the top-left starting coordinates and the dimensions of the region and must be of the form [hPos vPos hSize vSize]. The coordinates count from the upper-left corner of the active frame, defined as [1, 1]. hSize must be greater than one (4).

The generated HDL code for the ROI Selector block uses two 32-bit counters. The block does not use additional counters for additional regions.

- **Sobel Edge Detector**

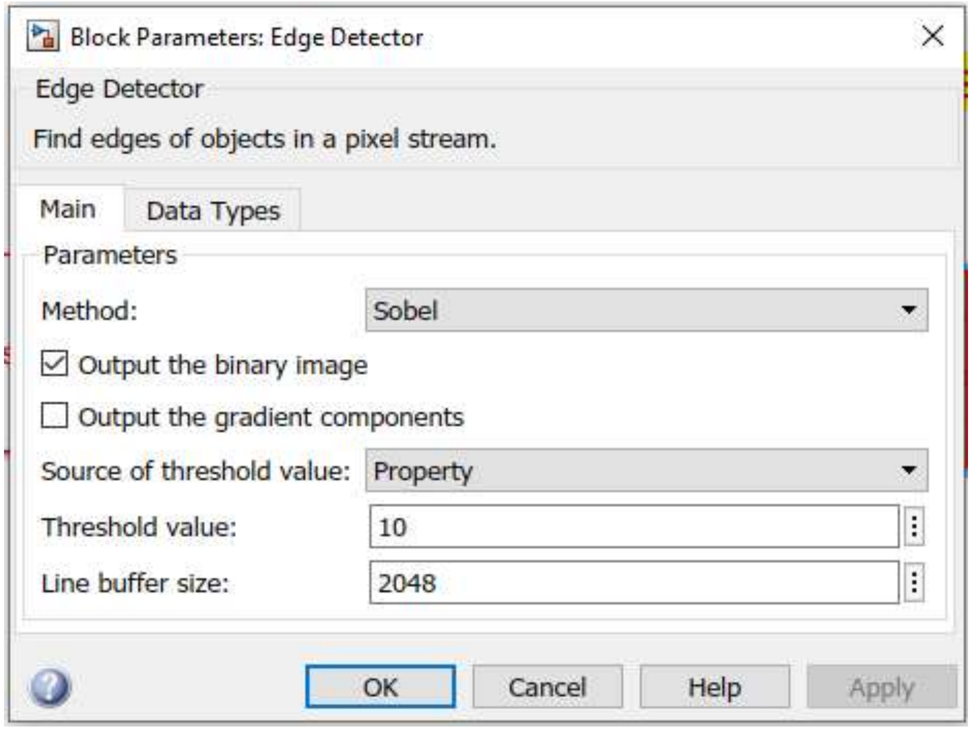
In this stage, we have a region of the original frame contain the original frame converted to HLS color space and transformed to the bird-eye view. Therefore, the frame is now ready to find the possible edges within it and this is the purpose of this block.

The Edge Detector block finds the edges in a grayscale pixel stream by using the Sobel, Prewitt, or Roberts method. The block convolves the input pixels with derivative approximation matrices to find the gradient of pixel magnitude along two orthogonal directions. It then compares the sum of the squares of the gradients to the square of a configurable threshold to determine if the gradients represent an edge.

By default, the block returns a binary image as a stream of pixel values. A pixel value of 1 indicates that the pixel is an edge. You can disable the edge output. You can also enable output of the gradient values in the two orthogonal directions at each pixel.

This block uses a streaming pixel interface with a bus for frame control signals. This interface enables the block to operate independently of image size and format. The pixel, Edge, and gradient ports on this block support single pixel streaming or multipixel streaming. Single pixel streaming accepts and returns a single pixel value each clock cycle. Multipixel streaming accepts and returns a vector of 4 or 8 pixels per clock cycle to support high-frame-rate or high-resolution formats. Along with the pixel, the block accepts and returns a pixel-control bus containing five control signals. The control signals indicate the validity of each pixel and their location in the frame. For multipixel streaming, one set of control signals applies to all four or eight pixels in the vector.

The following image shows our configuration of the block parameters.



Block Parameters: Edge Detector

Edge Detector

Find edges of objects in a pixel stream.

Parameters

Method: Sobel

☑ Output the binary image

☐ Output the gradient components

Source of threshold value: Property

Threshold value: 10

Line buffer size: 2048

Threshold value that defines an edge, specified as a scalar. The block compares the square of this value to the sum of the squares of the gradients.

When you use multipixel streaming, the block uses a single line memory and implements Number Of Pixels filters in parallel. This increase in hardware resources is a tradeoff for increasing throughput compared to single-pixel streaming.

- **<u>Dilation</u>**

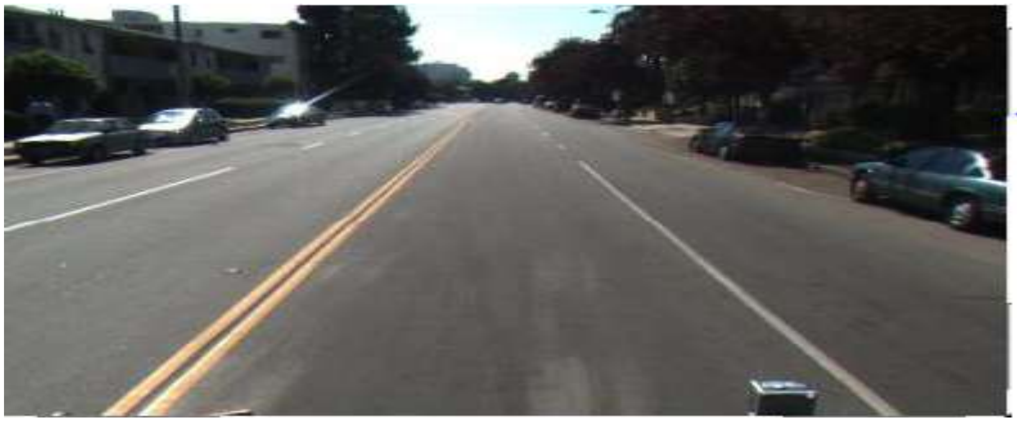
The Dilation block replaces each pixel with the local maximum of the neighborhood around the pixel. The block operates on a stream of binary intensity values. You can specify a neighborhood or structuring element of up to 32-by-32 pixels.

This block uses a streaming pixel interface with a bus for frame control signals. This interface enables the block to operate independently of image size and format. The pixel ports on this block support single pixel streaming or multipixel streaming. Single pixel streaming accepts and returns a single pixel value each clock cycle. Multipixel streaming accepts and returns 4 or 8 pixels per clock cycle to support high-frame-rate or high-resolution formats. Along with the pixel, the block accepts and returns a pixel control bus that contains five control signals. The control signals indicate the validity of each pixel and their location in the frame. For multipixel streaming, one set of control signals applies to all four or eight pixels in the vector.

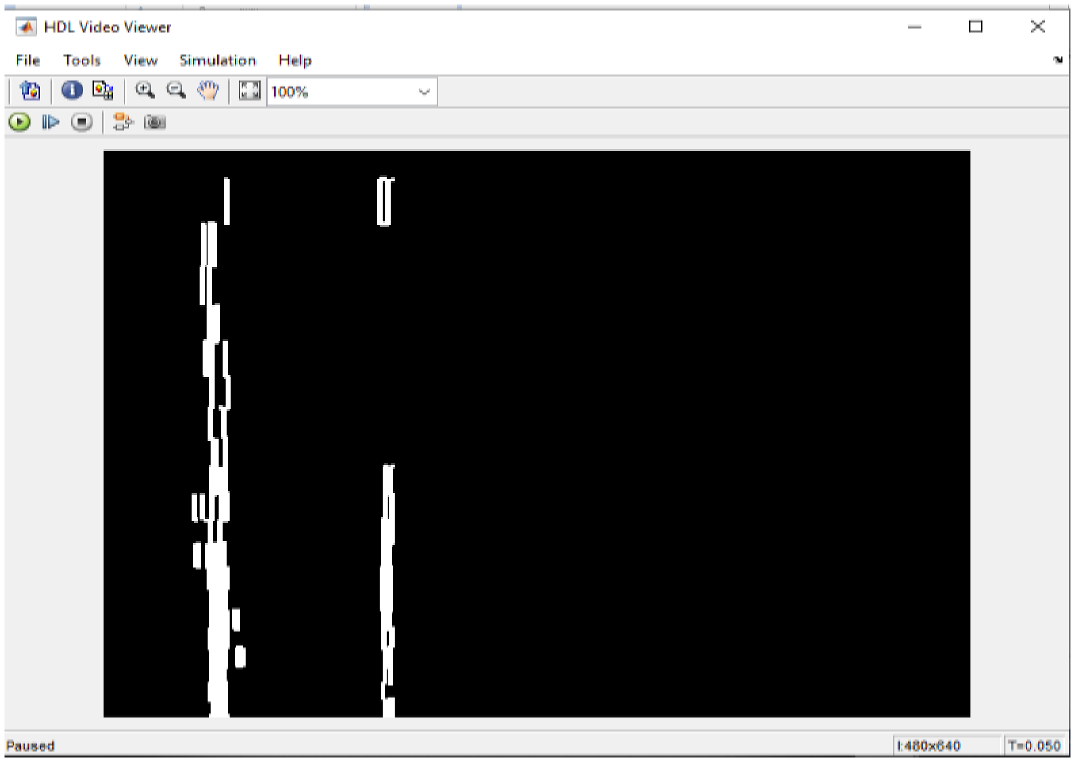
4.3.2. Simulation results

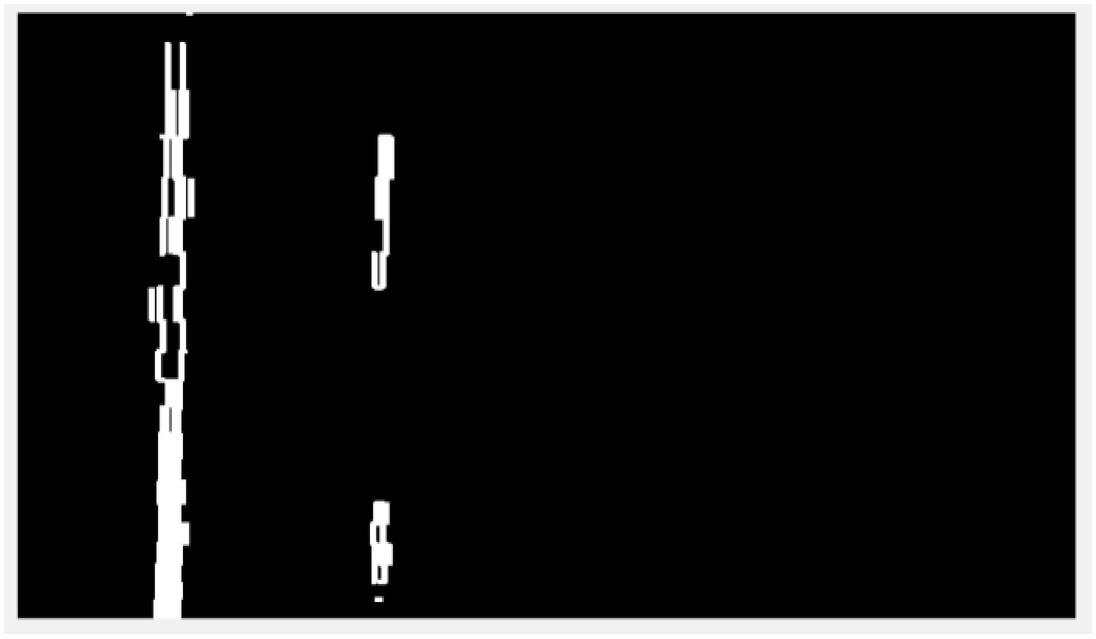
This section concludes the results from the system for an input video with a front camera view of a road.

For the following input frame:

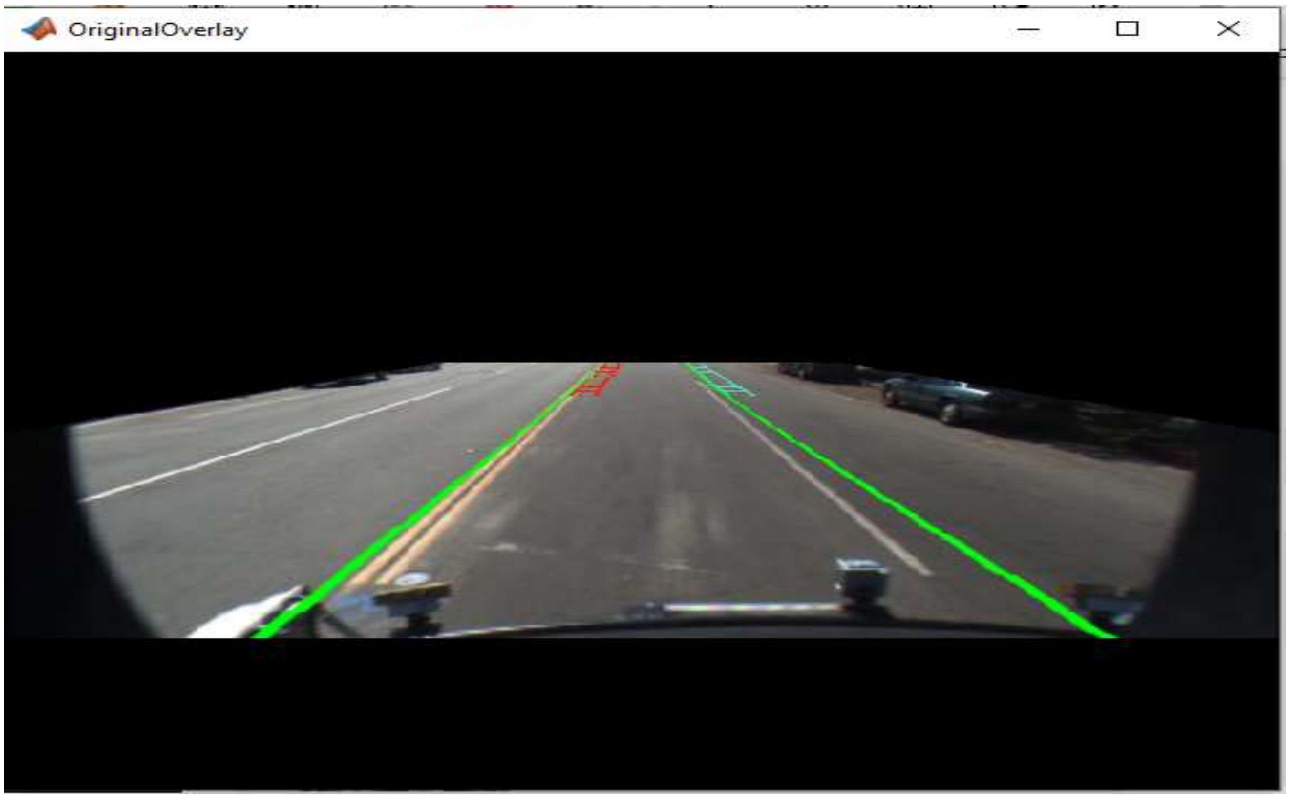


The output of the accelerated portion of the algorithm is a frame contain only the lane lines as follows:





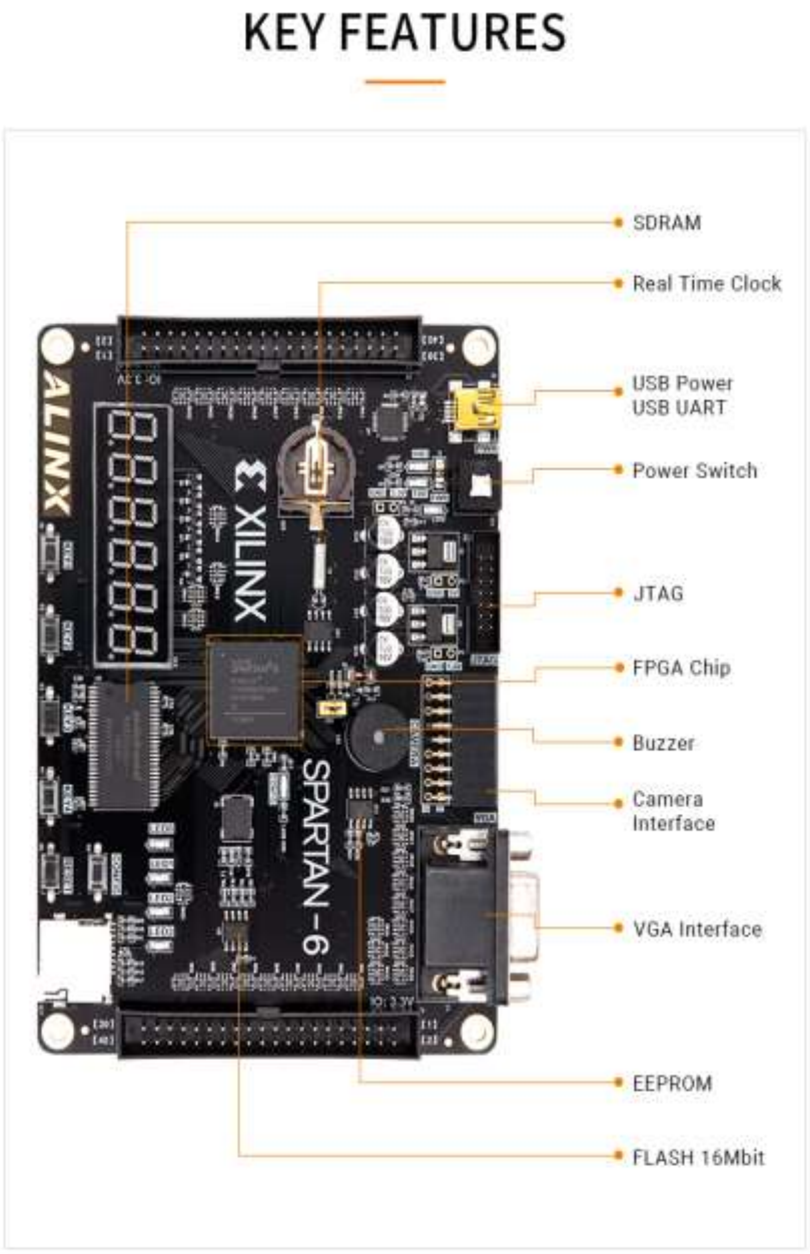
The output of the overall system with the software portion included is a frame contains the two lane lines spotted and marked:

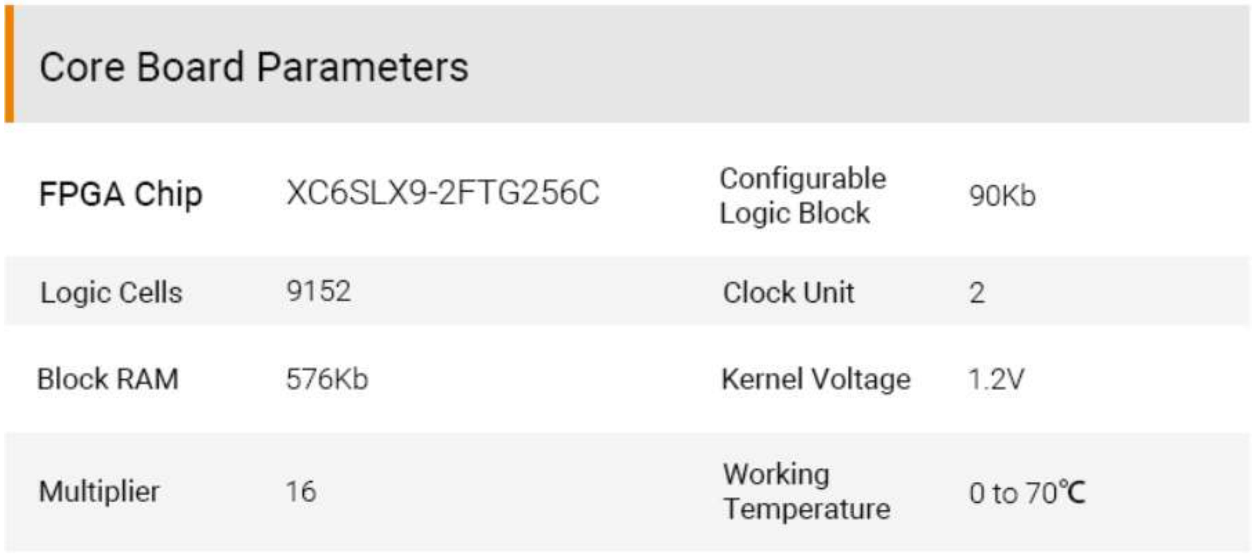


4.4. FPGA implementation:

4.4.1. Development Board

The Board used in implementation is XILINX Spartan-6 XC6SLX9 FPGA Development Board; in this section, we are going to list the specification of the board (3).



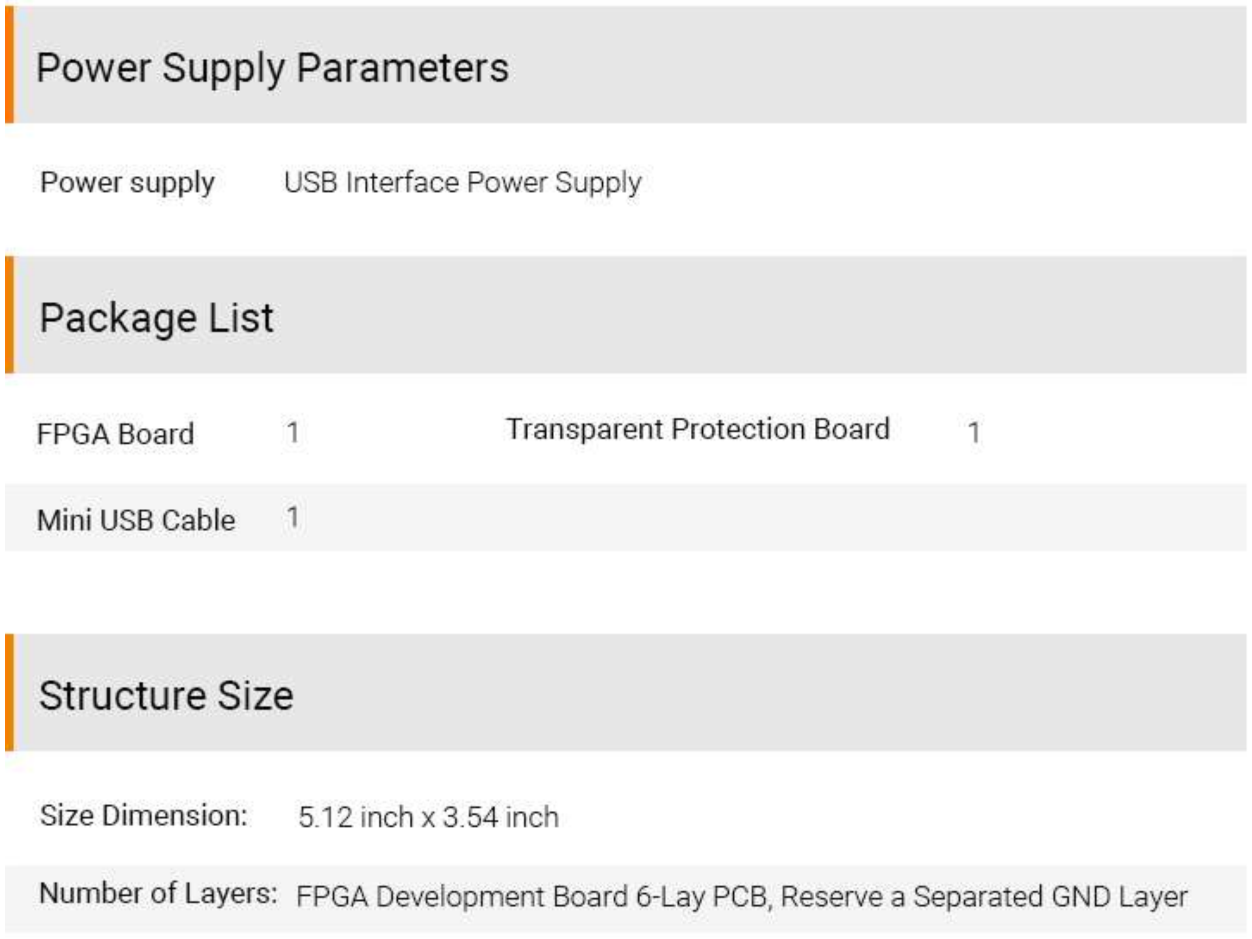


Core Board Parameters

FPGA Chip	XC6SLX9-2FTG256C	Configurable Logic Block	90Kb
Logic Cells	9152	Clock Unit	2
Block RAM	576Kb	Kernel Voltage	1.2V
Multiplier	16	Working Temperature	0 to 70℃

Interface and Function

JTAG	10-pin 0.1-inch Standard JTAG Port for Programs Debug and Download
USB Interface	USB Power Supply Interface, and Can Realize USB to Serial Port Function
SDRAM	256Mbit SDRAM, Used as FPGA Data Cache
SPI FLASH	16Mbit, Used as Storage for FPGA Configuration Files and User Data
Camera Interface	Connected with 5 million OV5640 Monocular Camera Module AN5640
VGA	VGA Interface, 16bit, Display 65536 Colors
Real Time Clock	RTC with a Battery Holder, The Battery Model is CR1220
EEPROM	EEPROM 24LC04 with IIC Interface On-Board
40-Pin Expansion Ports	Two 40-Pin Expansion Ports (0.1 inch Pitch), Can be Connect with Various ALINX Modules (Binocular Camera Module, TFT LCD Screen, Camera, AD/DA and Other Modules).
Crystal Oscillator	50MHz Provide Stable Clock Source for the System
LEDs	4 User LEDs
KEYs	5 Keys, 1 Reset KEY, 4 User KEYs
SD Card Slot	1 Micro SD Card Slot, Support SPI Mode
Digital Tube	6-bit Digital Tube to Realize 6-Digit Dynamic Display



Power Supply Parameters

Power supply	USB Interface Power Supply

Package List

FPGA Board	1	Transparent Protection Board	1
Mini USB Cable	1		

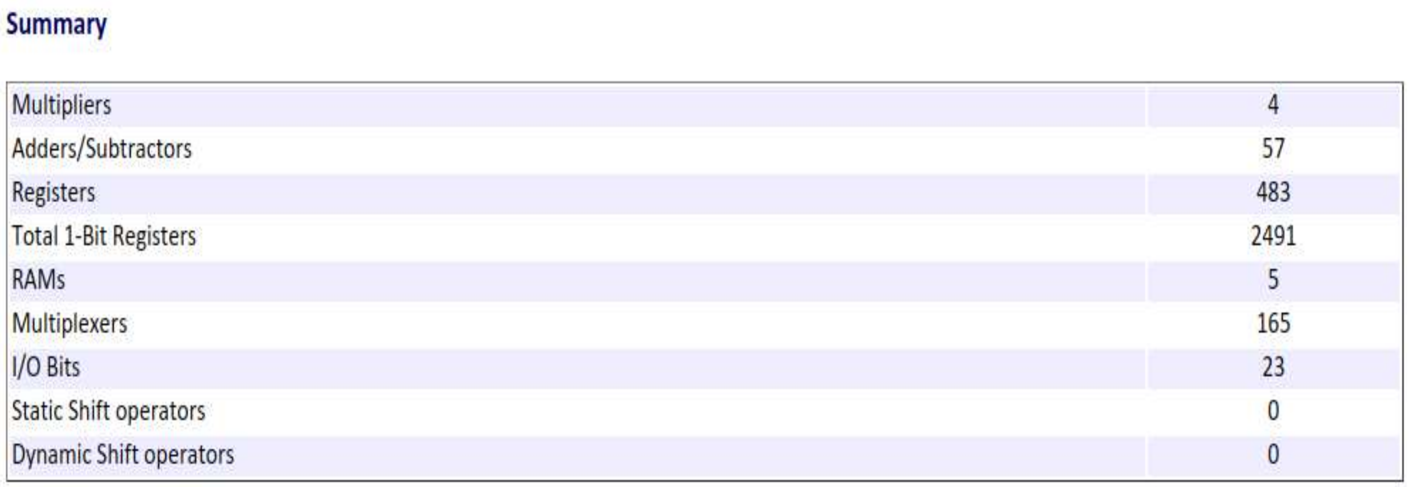
Structure Size

Size Dimension:	5.12 inch x 3.54 inch
Number of Layers:	FPGA Development Board 6-Lay PCB, Reserve a Separated GND Layer

4.4.2. The implementation Report

In this section, we are going to detail the FPGA implementation report of the HDL coder in the MATLAB.

For the high-level resource report, the resources used are as follows:



Summary

Multipliers	4
Adders/Subtractors	57
Registers	483
Total 1-Bit Registers	2491
RAMs	5
Multiplexers	165
I/O Bits	23
Static Shift operators	0
Dynamic Shift operators	0

Detailed as follows:

- **Multipliers (4)**

16x28-bit Multiply: 1

16x16-bit Multiply: 1

11x11-bit Multiply: 2

- **Adders/Subtractors (57)**

16x16-bit Adder: 18

16x16-bit Adder: 3

28x28-bit Adder: 1

17x17-bit Adder: 4

18x18-bit Adder: 2

11x11-bit Adder: 16

9x9-bit Adder: 4

10x10-bit Adder: 4

23x23-bit Adder: 1

17x17-bit Subtractor: 2

11x11-bit Subtractor: 2

- **Registers (483)**

16-bit Register: 53

1-bit Register: 300

2-bit Register: 3

8-bit Register: 58

28-bit Register: 2

17-bit Register: 4

18-bit Register: 4

11-bit Register: 44

3-bit Register: 2

9-bit Register: 4

10-bit Register: 4

22-bit Register: 4

23-bit Register: 1

- **RAMs (5)**

65536x8-bit RAM: 1

2048x8-bit RAM: 2

2048x1-bit RAM: 2

- **Multiplexers (165)**

16-bit 2-to-1 Multiplexer: 9

1-bit 2-to-1 Multiplexer: 71

1-bit 3-to-1 Multiplexer: 3

2-bit 7-to-1 Multiplexer: 1

1-bit 5-to-1 Multiplexer: 6

16-bit 5-to-1 Multiplexer: 1

16-bit 3-to-1 Multiplexer: 4

1-bit 4-to-1 Multiplexer: 8

28-bit 55-to-1 Multiplexer: 3

8-bit 2-to-1 Multiplexer: 7

2-bit 2-to-1 Multiplexer: 2

11-bit 2-to-1 Multiplexer: 12

1-bit 14-to-1 Multiplexer: 2

1-bit 11-to-1 Multiplexer: 10

1-bit 13-to-1 Multiplexer: 2

2-bit 14-to-1 Multiplexer: 2

1-bit 6-to-1 Multiplexer: 10

3-bit 11-to-1 Multiplexer: 2

8-bit 15-to-1 Multiplexer: 3

8-bit 3-to-1 Multiplexer: 2

22-bit 2-to-1 Multiplexer: 2

1-bit 9-to-1 Multiplexer: 3

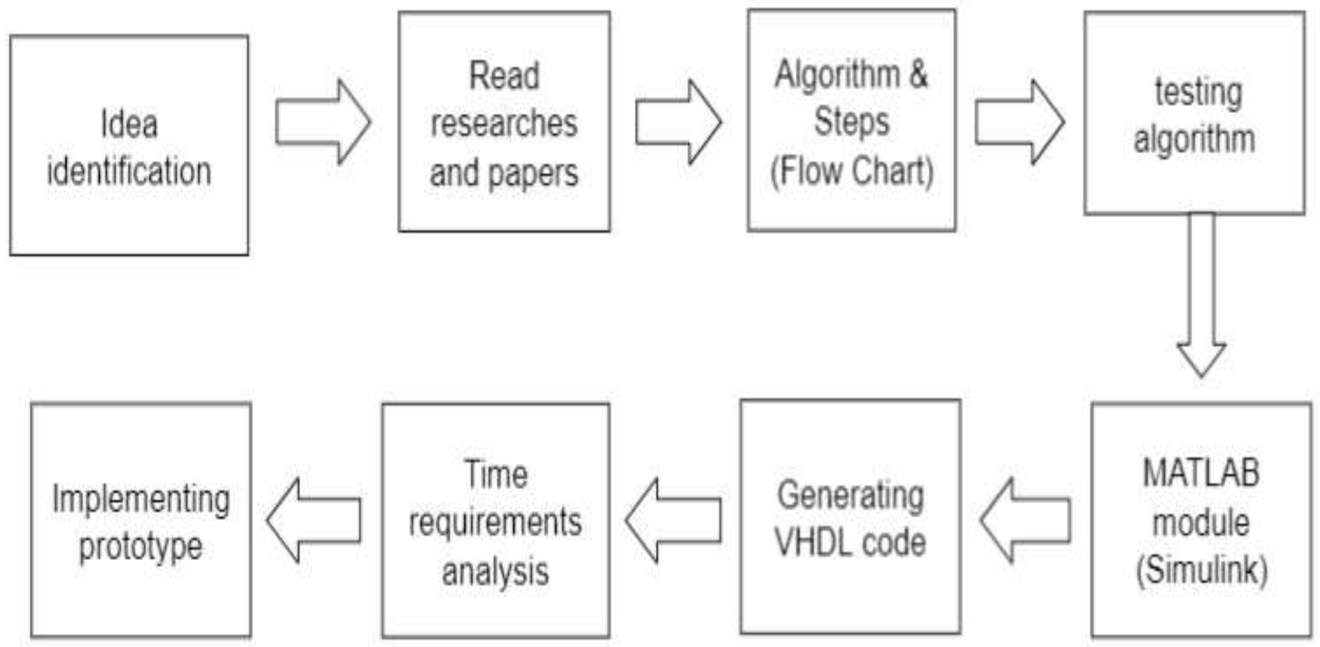
- **Number of I/O Bits (23)**

[+] Number of Input Bits: 16

[+] Number of Output Bits: 7

5. Conclusion and future work:

5.1. Project Stages:



Idea identification:

In this stage, we getting ready for the project by searching for the best idea which solves a critical problem in life, and finally our project idea in the autonomous vehicles field in which power consumption and speed are critical factors. Therefore, our project worked to solve this problem; by use hardware acceleration to optimizing computer vision algorithms in LDW embedded in vehicles to achieve good performance with less power consumption.

Read researches and papers:

After finding the idea, we started searching to study the basic knowledge we need to learn and searching in Previous papers and scientific articles which studied computer vision and hardware acceleration, etc., to learn about the latest developments and problems that occurred with them

Algorithms &steps (flow chart):

In this stage we made a control system to describe a control of our system, and made an algorithm for Lane departure warning with its steps.

Testing algorithm in python:

We use python IDE, OpenCV library to test our algorithm and expected results have been achieved

MATLAB and SIMULINK:

We chose the Mode Based Design (MBD) approach for the implementation of our project, MATLAB was the tool we choose to model and simulate our designed system; at this section, and we used SIMULINK in MATLAB to designed our system to generate HDL code for our system after that.

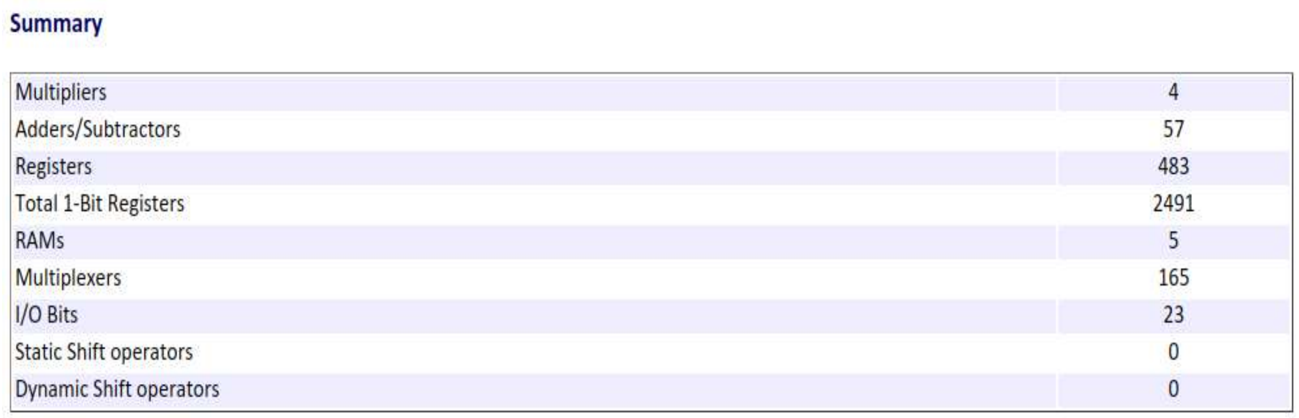
Generating HDL code:

In this section we generate HDL code by MATLAB HDL generation

And test this code using Xilinx ISE

FPGA implementation:

In this stage we make a report to show FPGA synthesis

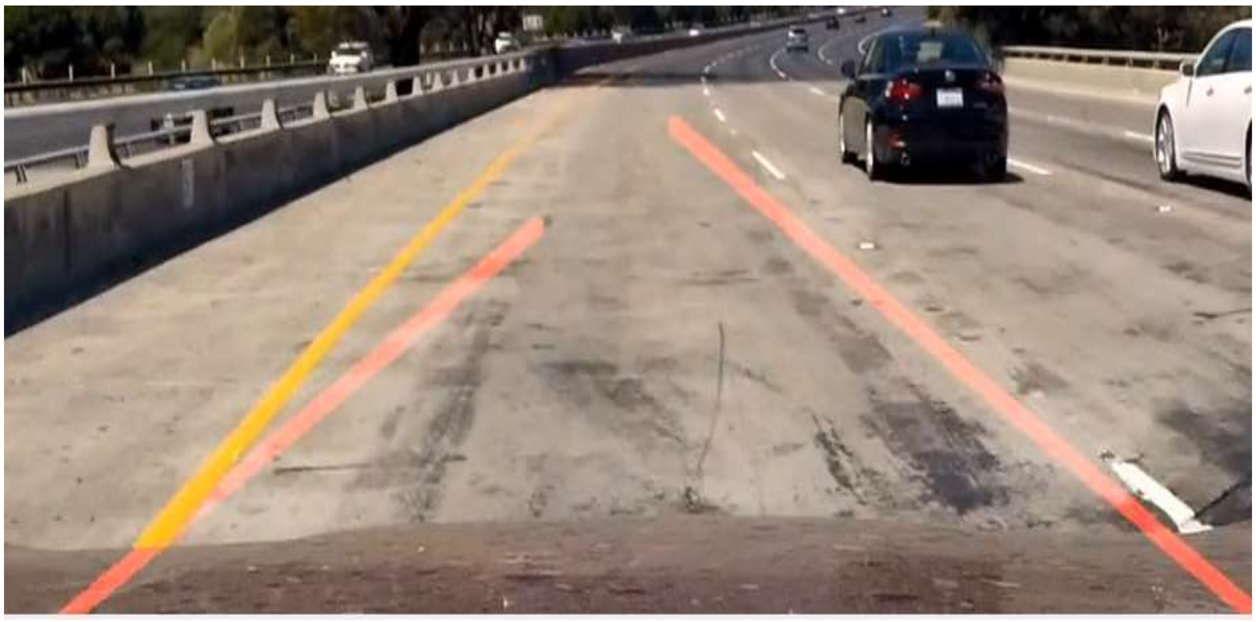


Summary

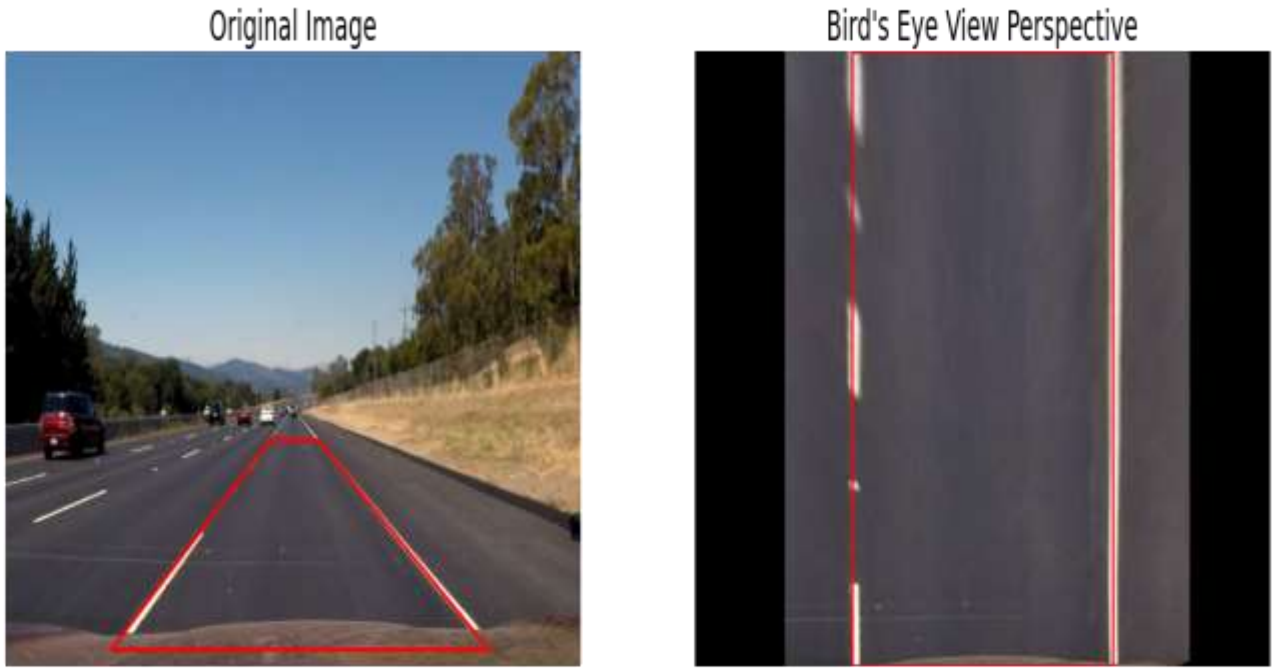
Multipliers	4
Adders/Subtractors	57
Registers	483
Total 1-Bit Registers	2491
RAMs	5
Multiplexers	165
I/O Bits	23
Static Shift operators	0
Dynamic Shift operators	0

5.2. problems and solutions

dimensions taken from the frame don't fit the real ones because the camera perspective differentiate from human's perspective of the road

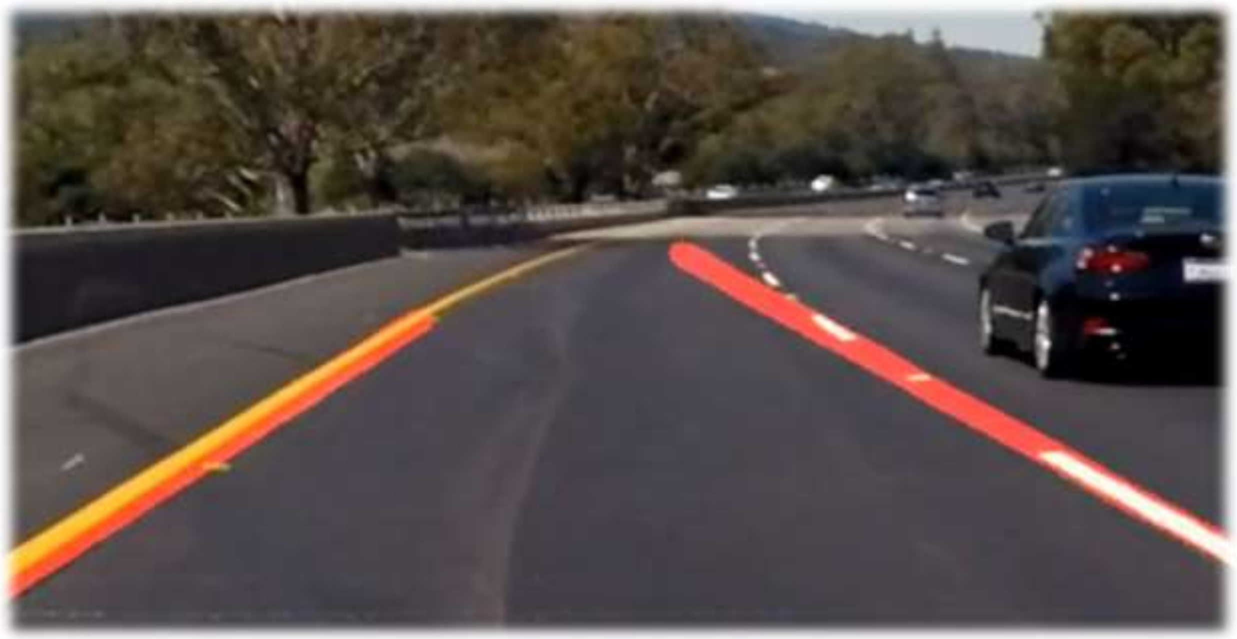


so, we use bird's eye transform to solve this problem and the result is shown in figure below

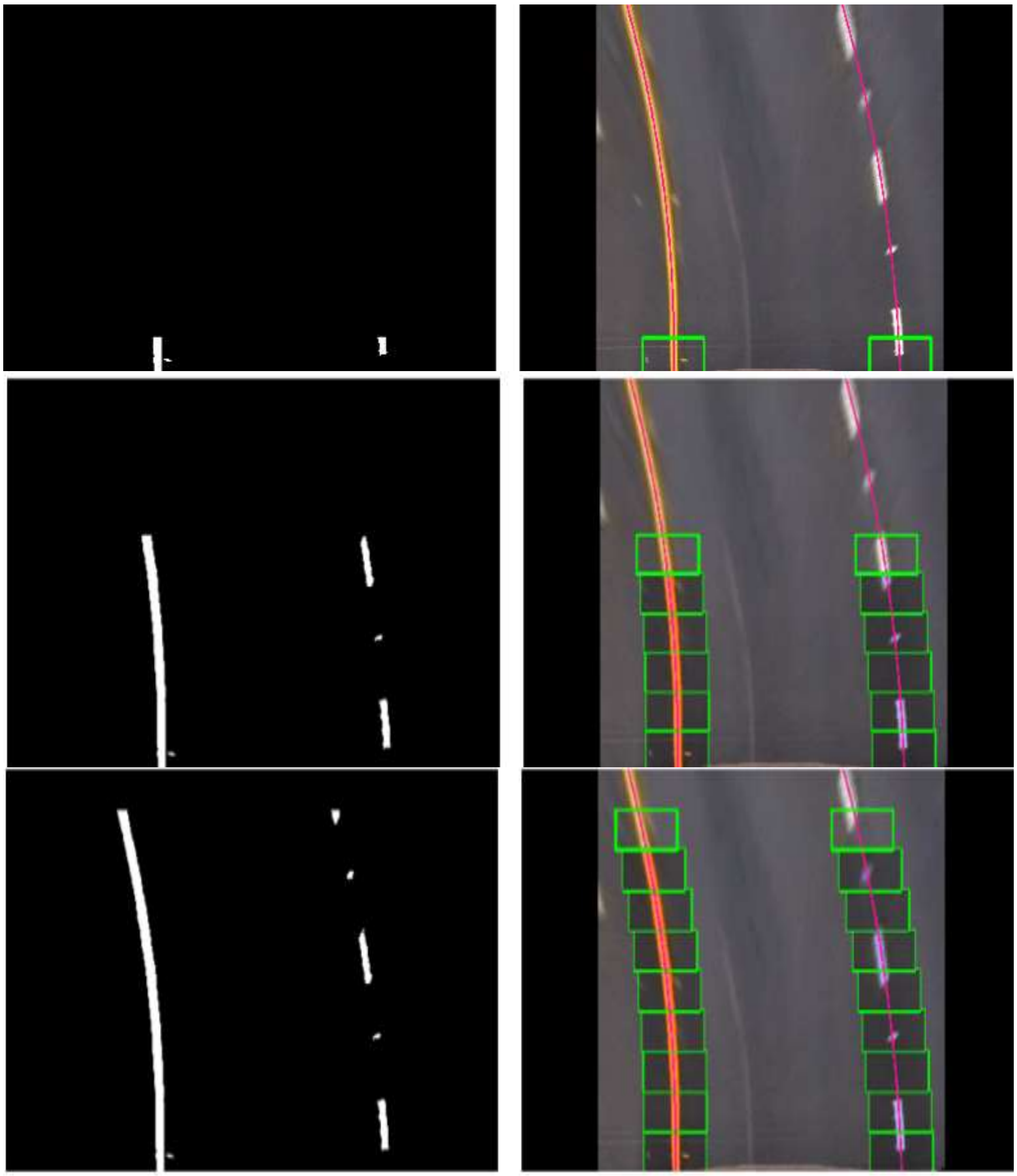


Also, the system won't track the lane correctly if the lane deviates

As shown in the figure below



so, we use sliding window technique with adaptive search to solve this problem and also bird’s eye help to apply this technique correctly



5.3. Future work:

5.3.1. Real time implementation

Having a legal access to a camera which can applicate our project and satisfy its requirement such as (ov5640) will make the system able to be used in real time; and then it can be applied to a vehicle traveling on a public road with a lane lines and system will detect lane and make warning to driver of vehicle deviates from lane Instead of the system was applied to a video recorded in a memory.

Figure 1 0v 5640 camera module

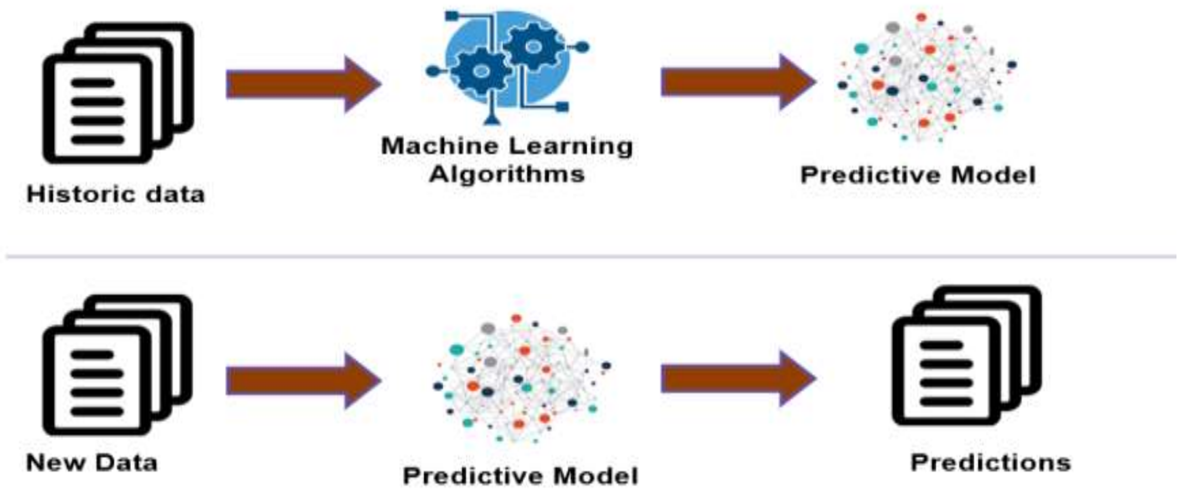
Figure 2 LDW in real time

5.3.2. Deep learning algorithm

Deep learning, which is a subset of machine learning has shown a significant performance and accuracy gain in the field of computer vision, so it will take more interest in the future rather than the traditional computer vision.

Data sets for lane line road are not available so if the data sets are available could make implementing using a deep learning algorithm, because machine learning and deep learning models depend on data. Without a foundation of high-quality training data, even the most performant algorithms can be rendered useless. Indeed, robust machine learning models can be crippled when they are trained on inadequate, inaccurate, or irrelevant data in the early stages.

Deep learning depends on data because it takes a part of the data to be trained on it and design a special model, and then after the training is finished, it tests on the test data.



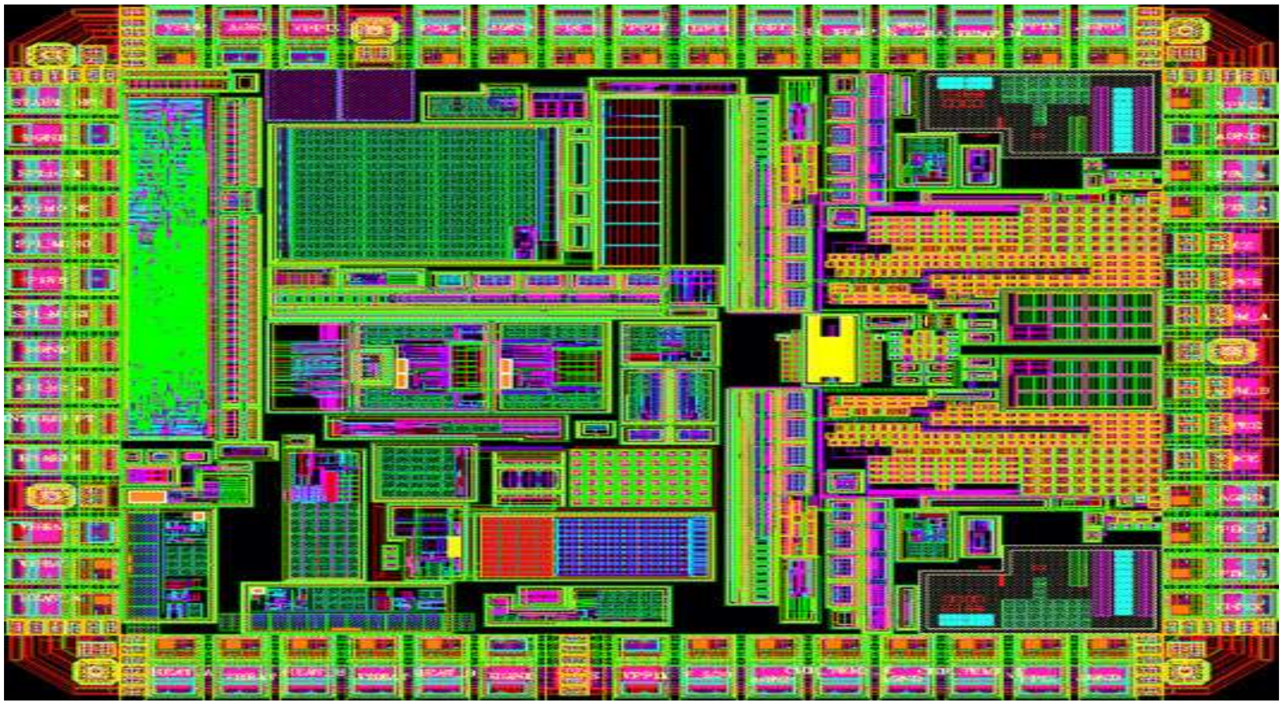
When the system is implemented in real time, then enough data will be available to build a deep learning model that active the same goal with high performance and accuracy.

5.3.3. Manufacturing

one of the future works on the system can be its implementation on ASIC to enter the manufacturing stage.

ASIC is an (application-specific integrated circuit) is an integrated circuit (IC) chip customized for a particular use, rather than intended for general-purpose use.

it used in the manufacturing stage if you want to add your system in industry and manufacturing, it will be a good solution due to its small size and low energy consumption compared to other options



References

1. **1. Velez, G.,.** A reconfigurable embedded vision system for advanced driver assistance. 2014.

2. **Bilal, Muhammad.** Resource-efficient FPGA implementation of perspective transformation for bird's eye view generation . 2019.

3. *ALINX AX309] XILINX Spartan-6 XC6SLX9 FPGA Development Board LX9 - Spartan6/7 - ALINX.*

4. Singh, R. and Ranasinghe,. Accelerating Computer Vision on mobile embedded platforms. 2016.

5. Kortli, Y., Marzougui, M. and Atri, M. . Kortli, Y., Marzougui, M. and Atri, M., 2016. Efficient implementation of a real-time lane departure warning system. 2016 International Image Processing,.

6. Zhao, J.,. Video/Image Processing on FPGA. 2015.

7. ***MATLAB and Simulink. MathWorks. (n.d.). https://www.mathworks.com/.***

Sponsors:

1- ASRT-Graduation project:

We had the financial aid in Automotive projects category with fund 15,740 Egyptian pounds according to the mentioned list of components.

2- Egypt IOT & AI Challenges:

It is co-organized by the Academy of Scientific Research & Technology and The Institute of Electrical and Electronics Engineers (IEEE TEMS) in partnership with NTRA & ITIDA.

Egypt IoT & AI Challenge is a capacity building and pre-incubation program for Senior University Students and Startups that have innovative ideas in the area of Internet of Things (IoT), Artificial Intelligence (AI) and related fields.