Pedestrian detection and tracking for driver assistance

A Project Report

submitted by

YASARLA RAJEEV

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2016

THESIS CERTIFICATE

This is to certify that the thesis titled **Pedestrian detection and tracking for driver assistance** submitted by **YASARLA RAJEEV** to the Indian Institute of Technology, Madras, for the award of the degree of **Dual Degree**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. A.N. Rajagopalan
(Research Guide)
Professor
Dept. of Electrical Engg.
IIT Madras
Chennai - 600 036.

Place: Chennai
Date: 28th April, 2016

ACKNOWLEDGEMENTS

I take this opportunity to express my deepest gratitude to my project guide Dr. A.N. Rajagopalan for his valuable guidance and motivation throughout the project. I am very grateful to him for providing his valuable time to guide me during the project.

It is a privilege to be a student in IIT Madras. I express special thanks to all my teachers for all the academic insight obtained from them. I also acknowledge the excellent facilities provided by the institute to the students.

I am indebted to my parents and my brother for their unconditional love, support and guidance. I dedicate this thesis to them.

ABSTRACT

KEYWORDS: left-right stereo image pair, disparity, Depth first search traversal.

With increasing number of road accidents, the need for systems that assist the driver in emergencies have increased. Electronics has developed to a large scale and good algorithms on faster processors allow orders of reduction in response time compared to human reaction time.

This project is an attempt to reduce car accidents by identifying the drivable areas on the road and alert when unexpected obstacles come up. These alerts can be used to develop semiautomatic or automatic cars. We develop a real time algorithm , which quickly detects objects like pedestrians and cars with help of the disparity map, Histogram of Oriented Gradients (HOG) and Haar cascade classifier. It also tracks them to estimate the future motion of the object with respect to the car by using optical flow and feature points.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

3**D**	Three-dimensional
DOF	Depth of field
MAP	Maximum aposterior probability
WCS	World coordinate system
CCS	Camera coordinate system
LCCS	Left camera coordinate system
RCCS	Right camera coordinate system
OpenCV	Open computer vision
StereoSGBM	Stereo semi global block matching
StereoBM	Stereo block matching
HOG	Histogram of oriented gradients
RAM	Random access memory
CPU	Central processing unit
GPU	Graphics processing unit

CHAPTER 1

INTRODUCTION

The aim of this project is to detect pedestrians and cars on the road and alert the driver to save him from unexpected accidents. In this project, we also estimate area of the road where he can safely move. In order to do all these, we should estimate the road in the image captured while the car is moving. We should also detect the objects relative to both the road and car, and track their motion. The algorithms of this project should be fast enough to alert drivers in real time.

This project is the first step to go towards autopilot car projects. Many companies are making autopilot cars but the equipment used for it is costly. Here in this project we want to make a driver assistance pedestrian and object detection system which is cheap, so that this can be equipped in any car. This project can be extended to a complete autopilot system.

Having defined the problem statement for the project, we now discuss the approach to solve it. First we need to calculate a depth map of the scene from which we can easily locate objects in the 3D world and track them. To calculate depth map, the following approaches exist.

- Axial Stereo: It has a single camera where we use the images at time t and $t - 1$(adjacent images) to calculate the depth map.
- Stereo Rig: It has two cameras which are fixed rigidly to one another. This is similar to binocular vision of a human. Here we calculate depth map using the images from both cameras at the same time.

In axial stereo case, camera motion at t and $t-1$ can be different while car is moving on the road. So we need to calculate the camera motion in order to calculate depth for this case. It is algorithmically hard and takes more time to calculate the camera motion at t and $t-1$. In stereo rig both the cameras undergo same motion at time t. So we need not calculate camera motion in order to compute depth map. Hence, we follow stereo rig approach to solve the problem.

1.1 Previous Works

Ewerth *et al.* (2004) and Wang and Huang (1999) find camera motion based on motion vectors at a pixel or for rigid blocks. Ewerth *et al.* (2004) estimates motion vectors for small patches. Based on location of the patch in the image, they formulated a relation between motion vector of the patch and the camera motion parameters (like zooming, rotation and translation) to estimate the camera motion.

We implemented the technique mentioned in Murphey *et al.* (2000) and Gong *et al.* (2014) to calculate depth map of a real video assuming camera motion in the axial direction alone, as in the case of axial stereo. They formulated the technique which will work for less Depth Of Field (DOF) and small shift of camera in axial direction. However, it is not the case for real world road scenes.

Stixels are defined as vertical stick or object above the flat ground in the corresponding column of the image. Benenson *et al.* (2011) estimated stixels without computing depth map. They estimated the stixel height by formulating cost function which assigns a local minimum value to object. After estimating stixels, (Benenson *et al.*, 2012) used HOG based pedestrian detection. Won *et al.* (2014) estimated stixels by formulating a cost function and guided filtering. Pfeiffer and Franke (2011) estimated stixels using Maximum Aposterior probability (MAP) of a pixel in the image given a stixel.

1.2 Organisation of the thesis

In chapter 2, we begin by explaining the stereo rig set-up and about disparity. We also explain how to calculate the disparity map for given left-right stereo image pair and reduce it's size without affecting performance.

In chapter 3, we discuss how to smooth disparity map and compute ground segmentation. We also discuss how to remove the errors in the ground segmentation.

In chapter 4, we discuss how to segment different objects each in column and row, called as Vertical and Horizontal Segmentation respectively. We also explain how to obtain objects given the segmentation and track them.

In chapter 5, we discuss results and performance of the algorithm.

CHAPTER 2

Introduction To Stereo Vision

2.1 Stereo Rig Setup

A stereo rig is designed to extract 3D information of objects from the images which helps to locate an object in both 3D space and images, calculate height of the object and track it. We follow the stereo rig system explained in Labayrade *et al.* (2002) and Hu and Uchimura (2005). A stereo rig has two cameras separated by a known base distance,b which take pictures of the scene at the same time. We assume that the stereo rig mounted on the vehicle has two coplanar cameras with the same intrinsic parameters and their horizontal co-axis is parallel to the road surface (see Fig.2.1), where the pitch angle to the ground plane is θ. We follow three co-ordinate systems, World Coordinate System, WCS(X_w, Y_w, Z_w) and two Camera Coordinate Systems (Left Camera Coordinate System, LCCS(X_l, Y_l, Z_l) and Right Camera Coordinate System, RCCS (X_r, Y_r, Z_r)). The origin of WCS is center of the origins of LCCS and RCCS. The optical axis of the WCS is parallel to the ground plane and it indicates the vehicle's direction of motion. 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 centre will be denoted by (u_o, v_o), assumed to be at the centre of the image.

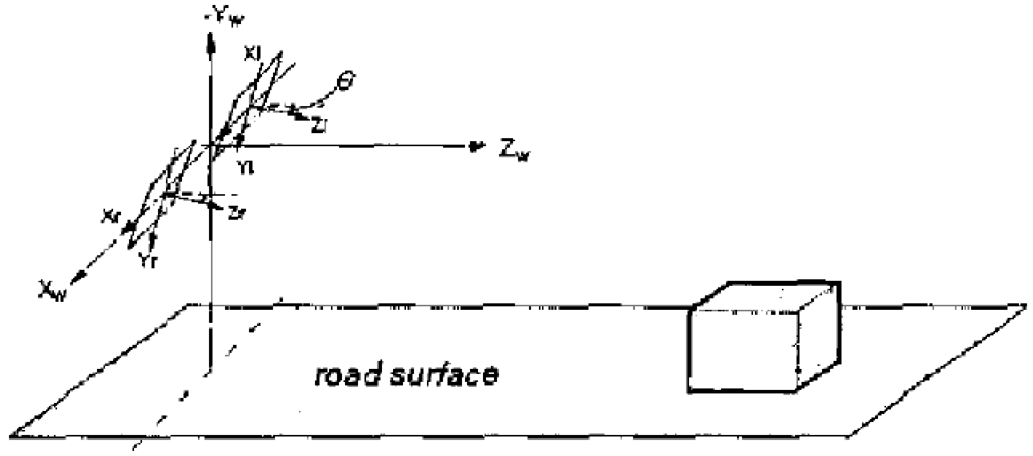


Figure 2.1: Relationship between Camera Coordinate System and World Coordinate System

Let us assume a pin-hole camera model for calculating the projection of any point in 3D world with respect to WCS on the image with camera coordinate system. Assuming the camera's aspect ratio is 1.0 and focal lengths of both the cameras as f, the perspective projection matrix is expressed as follows:

$$P = \begin{bmatrix} f & 0 & u_0 & 0 \\ 0 & f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.1}$$

We can obtain the transformation from WCS to CCS on translating by $\pm\frac{b}{2}$ and rotating by $-\theta$. So the transformation between the WCS homogeneous coordinates $(X_w, Y_w, Z_w, 1)^T$ and the image coordinates $(u, v, 1)^T$ is given by,

$$D_{l,r} = R_{l,r}T_{l,r} = \begin{bmatrix} 1 & 0 & 0 & \pm\frac{b}{2} \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = PD_{l,r} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = M_{l,r} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.3}$$

where λ is scale factor and $M_{l,r}$ is the transformation matrix,

$$M_{l,r} = \begin{bmatrix} f & u_0\sin(\theta) & u_0\cos(\theta) & \pm\frac{fb}{2} \\ 0 & f\cos(\theta) + v_0\sin(\theta) & -f\sin(\theta) + v_0\cos(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \end{bmatrix} \tag{2.4}$$

From equation Eq.2.3 and Eq.2.4, we can simply calculate the camera's image coordinates:

$$\begin{cases} u_{l,r} = & u_0 + f\dfrac{X_w \pm \frac{b}{2}}{Y_w\sin(\theta) + Z_w\cos(\theta)} \\ v = & v_0 + f\dfrac{Y_w\cos(\theta) - Z_w\sin(\theta)}{Y_w\sin(\theta) + Z_w\cos(\theta)} \end{cases} \tag{2.5}$$

2.2 Calibration Of Stereo Rig

Calibration of stereo rig is about estimating intrinsic and extrinsic parameters of two cameras in the rig and hence the transformation between the two camera images. Calibrating the camera gives us some specific values which can be used to measure distances in length units and not in pixels. Before calibrating parameters of stereo rig we should make sure the cameras are rigidly fixed such that relative position and orientation remains same. From Eq. 2.1 in the previous section 2.1, if A be a matrix such that, $P = [A\ O]$ where $O = [0\ 0\ 0]^T$, then

$$A = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

A is called the intrinsic parameter of stereo rig. Intrinsic parameter includes focal length, image format and principal point (u_o, v_o). The matrix of intrinsic parameters, A does not depend on the scene viewed. So, once estimated, it can be re-used as long as the focal length is fixed.

The joint rotation-translation matrix $[R|T]$ is called a matrix of extrinsic parameters. It is used to describe the camera motion around a static scene, or vice versa, rigid motion of an object in front of a still camera. If the poses of an object relative to the first camera and to the second camera, (R_1, T_1) and (R_2, T_2), respectively are computed, then those poses definitely relate to each other. This means that, given (R_1, T_1), it should be possible to compute (R_2, T_2). To relate those, we only need to know the position and orientation of the second camera relative to the first camera, i.e relative rotation and translation (R, T) with respect to first camera. $R_2 = R{*}R_1 T_2 = R{*}T_1 + T$

2.2.1 How To Calibrate

We use the CV::stereoCalibrate() function in OpenCV libraries to calibrate intrinsic and extrinsic parameters of stereo rig. After making sure that the cameras are rigidly fixed, we need to capture atleast 20 images of chessboard pattern in different positions positions. Chessboard pattern should contain minimum of 10 columns and 6 rows with size of the square around 2.5cm. In order to get a good calibration we should place the chessboard pattern in the camera frame such that:

- It is at the left and right edges of the field of view (X calibration)
- It is at the top and bottom edges of the field of view (Y calibration)
- It is detected at various angles to the camera ("Skew")
- It fills the entire field of view (Size calibration)
- It is tilted to the left, right, top and bottom (X,Y, and Size calibration)

We use CV::findChessboardCorners() function present in OpenCV to find corners of the chessboard pattern in the left-right pair of stereo images. These found corners in the left-right pair of stereo images are given as input to the CV::stereoCalibrate() which calculates matrices of intrinsic and extrinsic parameters. This function uses the technique mentioned in (Zhang, 2000) for calculating the parameters.

2.3 Rectification Of Stereo Images

Rectification of stereo images is required before proceeding for further calculations. Rectification of stereo image pair makes epipolar lines parallel to each other which ensures search space for particular pixel is horizontal in the stereo image pair while calculating disparity of pixel. Given the intrinsic and extrinsic parameters of stereo rig we rectify the images using inbuilt function in OpenCV libraries, CV::stereoRectify(). CV::stereoRectify() calculates the wrap which rectifies the left-right stereo image pair. We used "CV::remap()" function to wrap the images such that there is only horizontal shift of the object in the left-right stereo image pair. We can check the performance of these function by calculating and displaying the epipolar lines. For example in Fig.2.2 we can observe the difference in the stereo pair images before and after rectification. The images in Fig.2.2 are captured using the stereo rig set-up built by using web cameras. We fixed two web cameras rigidly to a wooden plank so that there is no relative motion between them.

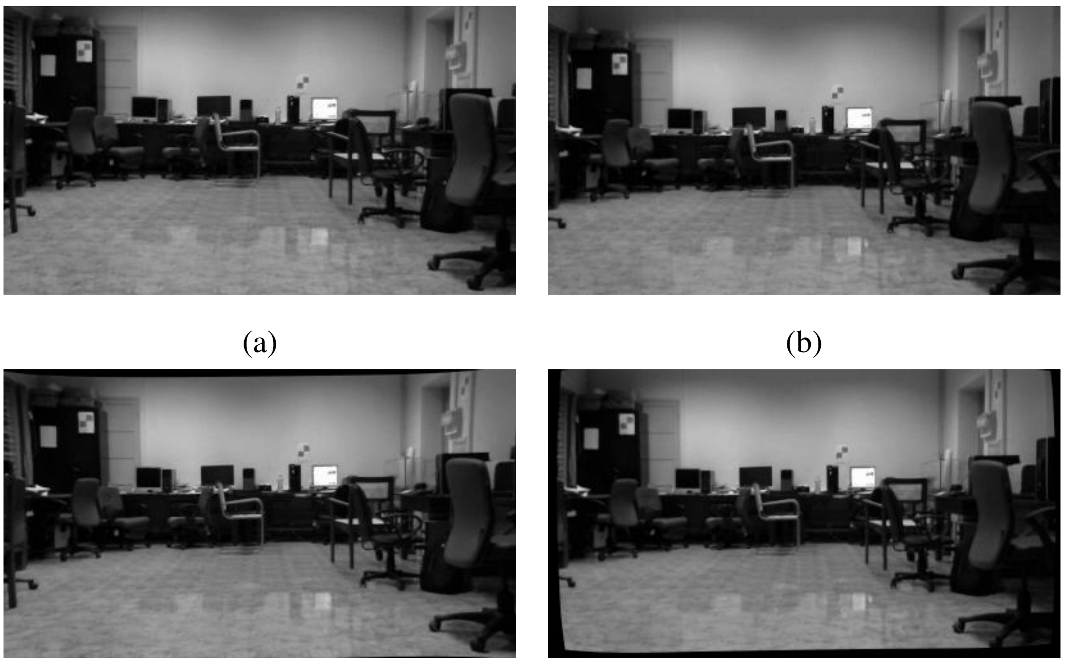

(a) (b) (c) (d)

Figure 2.2: (a),(b) left and right stereo image pair respectively, before rectification. (c),(d) are corresponding left and right stereo image pair after rectification

2.4 Disparity

Disparity is defined as the difference in image location of an object seen by the two cameras of a stereo rig. This is because the stereo rig is designed such that there is only horizontal shift of the object between left and right stereo image pair. For example we observe only horizontal shift of tree and house in the below Fig.2.3 Thus here, the

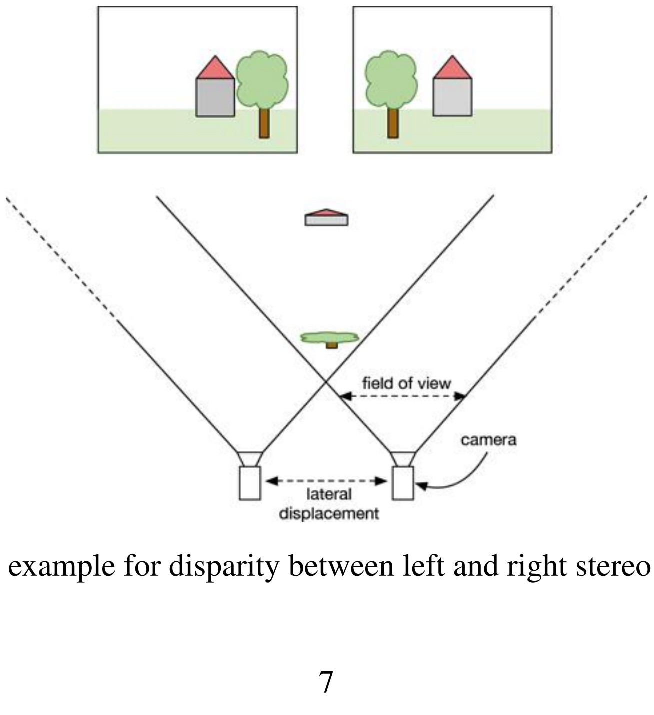


Figure 2.3: example for disparity between left and right stereo image pair

disparity,Δ is equal to change in x-coordinates of corresponding pixel in the left-right stereo image pair. From Eq.2.5 in section 2.1 we have, where (u_l, v) and (u_r, v) are coordinates in LCCS and RCCS of corresponding pixels in left-right stereo image pair.

$$\begin{cases} u_{l,r} = & u_0 + f\dfrac{X_w \pm \dfrac{b}{2}}{Y_w \sin(\theta) + Z_w \cos(\theta)} \\ v = & v_0 + f\dfrac{Y_w \cos(\theta) - Z_w \sin(\theta)}{Y_w \sin(\theta) + Z_w \cos(\theta)} \end{cases} \tag{2.7}$$

For more simplification, we denote new set of image coordinates (U, V) with respect to camera optical center:

$$\begin{cases} U = u_{l,r} - u_0 & = f\dfrac{X_w \pm \dfrac{b}{2}}{Y_w \sin(\theta) + Z_w \cos(\theta)} \\ V = v - v_0 & = f\dfrac{Y_w \cos(\theta) - Z_w \sin(\theta)}{Y_w \sin(\theta) + Z_w \cos(\theta)} \end{cases} \tag{2.8}$$

Thus from Eq.2.7 we have disparity,Δ as,

$$\Delta = u_l - u_r = f\frac{b}{Y_w \sin(\theta) + Z_w \cos(\theta)} \tag{2.9}$$

If $\theta = 0$ then,

$$\Delta = u_l - u_r = f\frac{b}{Z_w} \tag{2.10}$$

2.4.1 Representing Different Planes in U-V domain

Generally in the real world, we can approximate road, buildings and objects to different planes like horizontal, vertical and oblique planes. Here we define all planes with respect to WCS as shown in the Fig.2.4.

As explained in the paper (Hu and Uchimura, 2005) , we can represent different planes of WCS in U-V domain as

- Horizontal Planes
  A horizontal plane in WCS can be described as,

$$Y_w = h \boxed{\text{constant}} \tag{2.11}$$

From Eq.2.9 and Eq.2.8 in the previous section 2.4, we can deduce below equa-

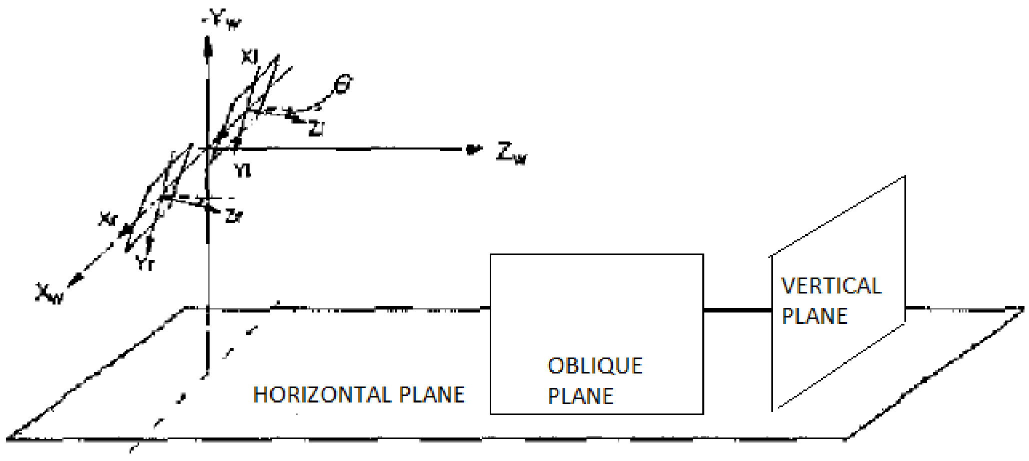


Figure 2.4: Planes in WCS

tion with respect to the left image

$$\frac{h}{b}\Delta = f\sin(\theta) + V\cos(\theta) \tag{2.12}$$

Eq.2.12 shows that horizontal planes in WCS are projected as straight line in V-disparity domain

- Vertical Planes
  A vertical plane in WCS can be described as,

$$Z_w = p\ \boxed{\text{constant}} \tag{2.13}$$

If $\theta = 0$ from Eq.2.10 in the previous section 2.4, then disparity,Δ is

$$\Delta = f\frac{b}{Z_w} = \frac{fb}{p} \cdots \boxed{\text{constant}} \tag{2.14}$$

Disparity is constant in U-V domain for Vertical plane in WCS.

- Oblique Planes
  An oblique plane in WCS can be described as,

$$X_w = \beta\ \boxed{\text{constant}} \tag{2.15}$$

If $\theta = 0$ from Eq.2.10 and Eq.2.8 in the previous section 2.4, we deduce a linear relation between U and Δ with respect to left image.

$$U = f\frac{(X_w + \frac{b}{2})}{Z_w} \qquad and \qquad \Delta = f\frac{b}{Z_w}$$

$$\frac{(\beta + \frac{b}{2})}{b}\Delta = U \tag{2.16}$$

Eq.2.16 shows that Oblique planes which are perpendicular to x-axis in WCS are projected as straight line in U-disparity domain

2.4.2 Disparity Calculation

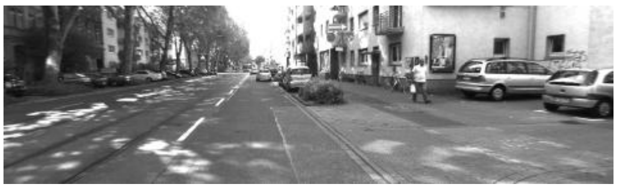
Disparity Calculation at any pixel can be done by using several algorithms. As we are dealing with road scenes, and implementing an algorithm for detection and tracking of pedestrians and objects in driver assistance car, algorithm should be fast enough to alert the driver, in order to prevent accidents. We use Stereo Semi Global Block Matching, (StereoSGBM) algorithm in OpenCV as it performs faster than other algorithms like Stereo Block Matching (StereoBM). StereoSGBM is less accurate than StereoBM, but the performance of StereoSGBM is good enough for further calculations as we smooth the disparity map, as explained in later chapters. StereoSGBM matches the blocks centred around the pixels in the left-right stereo image pair. StereoSGBM uses Boyer Moore's algorithm (as explained in Boyer and Moore (1977)) for block matching purpose. StereoSGBM was constructed from the algorithm in the paper Hirschmuller (2008). StereoSGBM also includes Birchfield and Tomasi (1998) sub-pixel metric.

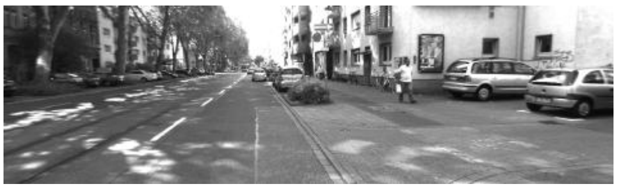
We use StereoSGBM class available in OpenCV for calculating disparity. We should set parameters to get accurate disparity values. The in-built class function sgbm->compute() computes the disparity values at every pixel of given left-right stereo image pair. The parameters set for calculating disparity are:

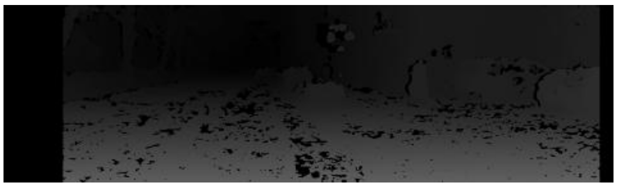
- SADWindowSize = 7; numberOfDisparities = 160; cn = image1.channels(); sgbmWinSize=9;
- sgbm->setPreFilterCap(63);
- sgbm->setBlockSize(sgbmWinSize);
- sgbm->setP1(8*cn*sgbmWinSize*sgbmWinSize);
- sgbm->setP2(32*cn*sgbmWinSize*sgbmWinSize);
- sgbm->setMinDisparity(-31);
- sgbm->setNumDisparities(numberOfDisparities);
- sgbm->setUniquenessRatio(15);
- sgbm->setSpeckleWindowSize(100);
- sgbm->setSpeckleRange(32);
- sgbm->setDisp12MaxDiff(1);

"sgbm->compute(image1, image2, disp);" computes the disparity for left-right stereo image pair of image1 and image2. Disparity value at pixel obtained will range betweem

$[-32\ 128]$. For example see Fig.2.5 disparity for the left-right stereo image pair Fig.2.5 (a) and Fig.2.5 (b).


(a)


(b)


(c)

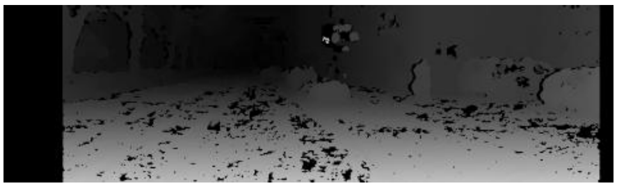

(d)

Figure 2.5: (a),(b) are rectified left and right stereo image pair respectively. (c) is disparity map computed using StereoSGBM. (d) is disparity map where disparity values are mapped to gray values [0 255]

Disparity map contains black patches (as shown in Fig.2.5) where StereoSGBM was unable to match block in the given search space so it will simply assign minimum disparity value to those pixels.

CHAPTER 3

Ground Segmentation

3.1 Introduction

In this chapter we learn how to obtain the road plane from disparity map. The disparity map contains several black patches where StereoSGBM is unable to match blocks for the pixels. So before going for road segmentation, we need to remove the patches. Images used here are of size 1344 columns and 391 rows which implies the size of Disparity is also 1344 columns and 391 rows. If we proceed with the disparity of same size for further calculations the time complexity of the algorithm increases. As our goal of this project is to achieve a fast algorithm that detects and tracks pedestrians on road, we decrease the size of disparity map in a way that does not affect our output results. Let N_{rows} be the number of rows and N_{cols} be the number of columns in left-right stereo image pair.

3.2 Reduction of Disparity Map Size

Time complexity of the overall algorithm can be reduced by reducing the disparity map size. In this report, we reduce disparity map by clubbing 3 columns into 1 column which means stixel width in each column will become 3 pixels. Even if we reduce it this way, performance is not affected as explained in Benenson *et al.* (2012). Benenson *et al.* (2012) has explained different ways of reducing disparity so as to make algorithm fast, among which fixing stixel width as 3 pixels in each column has better performance, so we use it.

Let D be the disparity Matrix. We create M, Marker matrix such that

$$M(i,j) = \begin{cases} 1 & D(i,j) < -8 \\ 0 & otherwise \end{cases} \tag{3.1}$$

We average the pixels whose corresponding marker value is '0', in 3 columns of original disparity to corresponding single column in updated disparity matrix. Let the updated matrix be D_m,

$$D_m(i,j) = \begin{cases} 0 & \sum_{k=0}^{2} M(i,3j+k) = 3 \\ \dfrac{\sum_{k=0}^{2} (1 - M(i,3j+k))\, D(i,3j+k)}{\left(3 - \left(\sum_{k=0}^{2} M(i,3j+k)\right)\right)} & otherwise \end{cases} \tag{3.2}$$

This type of reduction removes small black patches of size less than 5X5 pixels.

3.3 Smoothing

Black patches occur due to occlusions, oversaturation of pixels, parallax errors and insufficient data during computing. Smoothing should be done to remove these errors and to get better results. Here while smoothing we initially made an assumption that the lower 15 rows of the image belong to road. Firstly, we perform row by row smoothing and then we smooth the entire image using column smoothing.

3.3.1 Row Smoothing

We take the Marker matrix,M and the updated matrix D_m from section 3.2, as input and smooth the lower 11 rows. Before smoothing we invert each column such that top goes to bottom and bottom goes to top for easy calculations. The new matrix will be $\hat{D}_m$,

$$\hat{D}_m(i,j) \;=\; D_m(N_{rows} - i, j) \tag{3.3}$$

N_{rows} and N_{th} are number of rows and reduced number of columns of disparity matrix. Here in row smoothing, we calculate average disparity value, Avg of valid pixels in a row whose Marker value is "0". Now we calculate $Avg - \hat{D}_m(i,j)$, difference between disparity value and average disparity value of the corresponding row. If it is greater than 30, then we make pixel disparity value as invalid by updating its marker value to 1 in M (as shown in the Algorithm 1). We join all the valid points by straight lines and linearly interpolate disparity from those straight lines for invalid pixels.

Algorithm 1 Row Smoothing

```
Require: Matrices $\hat{D}_m$, $M$
1: for i= 0 to 10 do
2:     calculate Avg,Average disparity of valid pixel in row
3:     for j=0 to $N_{th}$ do
4:         if $Avg - \hat{D}_m(i,j)$ >30 then $M(i,j) = 1$
5:     end for
6:     join all valid pixels by straight lines and calculate disparity value for invalid
       points
7: end for
```

3.3.2 Column Smoothing

We remove the patches along the columns by calculating possible slope of the line by differentiating cases like occlusion, large patch of missing pixels and errors due to over saturation. Large patch of missing pixels can occur due to mirrors and over saturation of very shiny objects. We find window size of the invalid pixels and decide whether it belongs to same object or may be it is mirror or due to occlusion or it may appear in left but may not appear in right because of parallax vision.

We go along the column, and when we encounter invalid pixel, we start to calculate length of the window which is difference between previously encountered invalid pixel to next valid pixel when we are going along the column. Now we differentiate the cases based on the length of window obtained. Let previously encountered invalid pixel be p_{prev} and next valid pixel to p_{prev} when we are going along the column be p_v. Thus window size is $p_v - p_{prev}$. The cases we consider are

- if the window size > 90, then we check neighbouring pixels of $p_{prev} + k$ (where k>90 and k<110) which are valid. if we get any we use that value and draw a straight from p_{prev} to $p_{prev} + k$. Thus we get disparity value for in between invalid pixels. This case occurs mainly for large transparent mirrors of cars.
- if $(\hat{D}_m(p_{prev}) - \hat{D}_m(p_v)) < 10$ then we draw a straight from p_{prev} to p_v. This case generally occurs for near large objects.
- if $(\hat{D}_m(p_{prev}) - \hat{D}_m(p_v)) > 10$ and window size > 80, then we draw a straight from p_{prev} to p_v. this generally occurs when road surface is occluded or small objects are occluded by the others.
- For all the other cases, we calculate the slope using next 10 to 20 valid pixels from p_v and using $\hat{D}_m(p_v)$ and slope we calculate disparity value for invalid pixels.

D_s is disparity matrix after complete smoothing. In Fig.3.1 we clearly observe the difference in disparity map before and after smoothing.

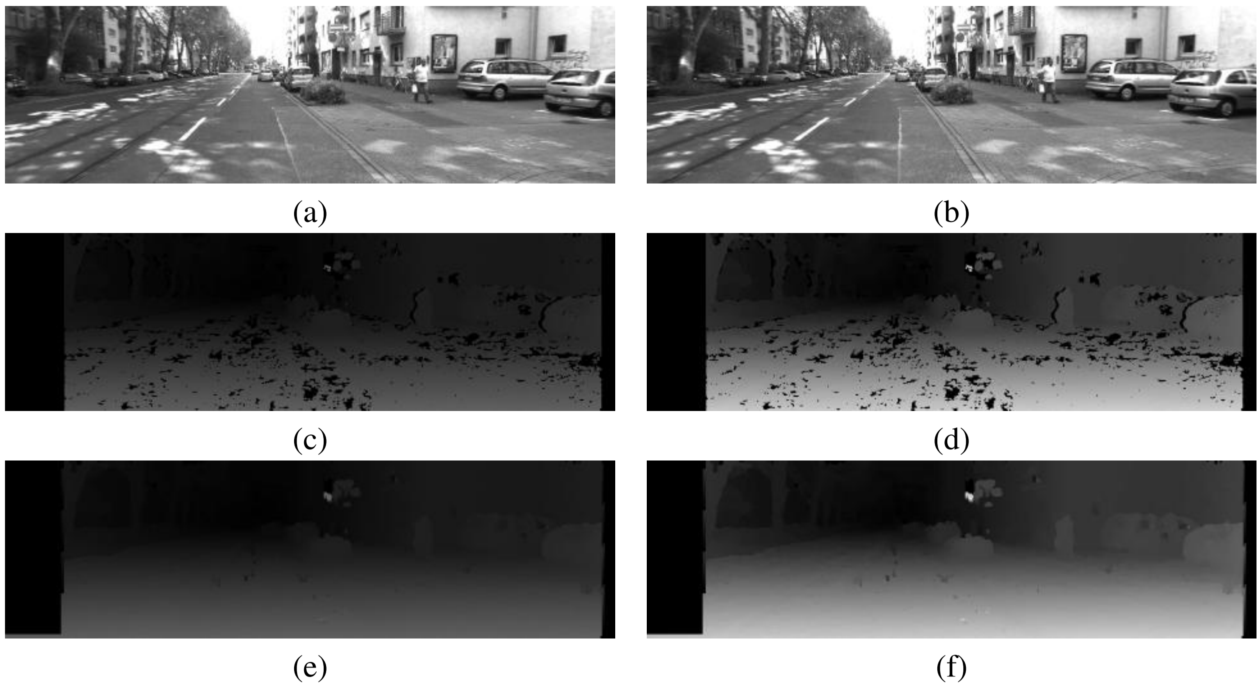


Figure 3.1: (a),(b) are rectified left and right stereo image pair respectively. (c) is disparity map computed using StereoSGBM. (d) is disparity map where disparity values are mapped to gray values [0 255]. (e) is disparity map obtained after smoothing. (f) is disparity map where disparity values of (e) are mapped to gray values [0 255]

3.4 Ground Segmentation

We make the following assumptions about road scene and differentiated road from the objects,

- We assume the lower 15 rows in the image belongs to the road.
- Road cannot exceed $\frac{2N_{rows}}{3}$.
- In general we observe the road in the middle of the image and sides of the image may belong to objects like cars and buildings. So, we assume any pixel p in the columns, from 0 to $\frac{N_{th}}{3}$ and from $\frac{2N_{th}}{3}$ to N_{th} may belong to the object but not the ground. So we check pixels belonging that range of columns for twice or thrice whether it belongs to ground or not.

3.4.1 Algorithm

In a road scene left-right stereo image pair, if we go along the column, objects will have a slope less than 0 or very near to zero (as mentioned in the section 3.3.1). Using this, we separate objects from ground. We calculate the average slope for a block size of 10 pixels, at every pixel in both up and down direction using Dynamic programming. Let $UD(i,j) or UD(p)$ and $DD(i,j) or DD(p)$ be average of slope at pixel 'p'(at location (i,j)) in up and down directions respectively. Let G be the ground segmentation matrix such that $G(i,j) = 0$ if p belongs to road otherwise $G(i,j) = 255$.

Algorithm 2 Ground Segmentation

```
Require: Matrices $UD$, $DD$
 1: for i= 15 to $N_{rows}$ and j=0 to $N_{th}$ do
 2:   if $DD(i,j) < T_g$ and $DD(i+1,j) < T_g$ then
 3:     then it belong to Object
 4:   else
 5:     if $UD(i,j) < T_g$ and $UD(i-1,j) < T_g$ then
 6:       then it belong to Object
 7:     else
 8:       pixel belongs to road. calculate the slope the road
 9:     end if
10:   end if
11: end for
```

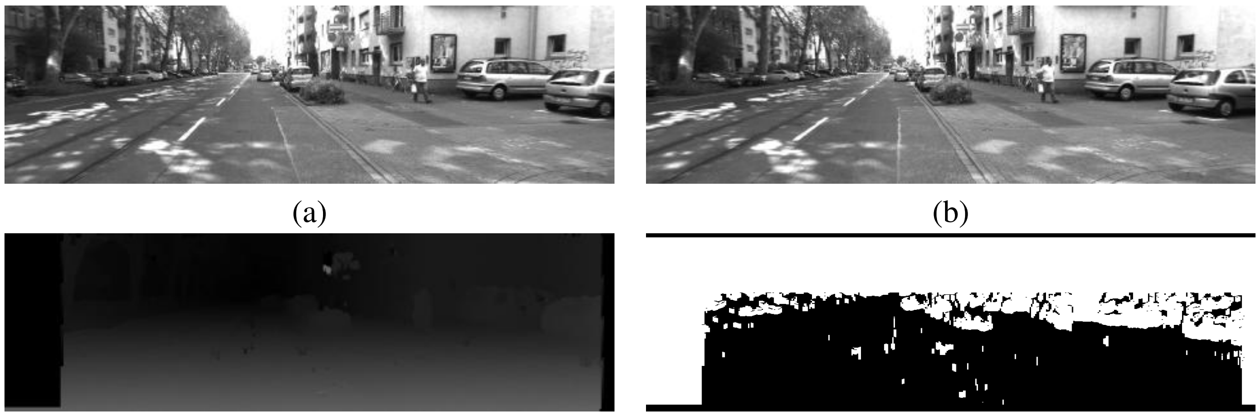


Figure 3.2: (a),(b) are rectified left and right stereo image pair respectively. (c) disparity map after smoothing. (d) resultant G, ground segmentation matrix obtained using this algorithm

Ground segmented image for an example left-right stereo image pair is shown in the above Fig.3.2.

3.4.2 Removal Of Errors

The result obtained in the previous subsection 3.4.1 contains errors. These errors can be removed by the assumptions made earlier in this section. In the following subsection, we discuss two ways of removing the errors,

- based on checking with threshold for each row.
- based on size of patch of the road or object.

Comparing with Threshold

We will calculate average road value for each row by going along left and right from middle column for 10 pixels. Let N_{mid} be middle column and Avg(i) be average disparity value of road pixels in i^{th} row.

$$\boxed{Avg(i) = \frac{\sum_{j=N_{mid}-10}^{N_{mid}+10} \left(\frac{255 - G(i,j)}{255}\right) D_s(i,j)}{\sum_{j=N_{mid}-10}^{N_{mid}+10} \left(\frac{255 - G(i,j)}{255}\right)}} \tag{3.4}$$

Using $Avg(i)$ we calculate average slope of this road scene. Let $rlimit$ be,

$$\boxed{rlimit = 11 + \left(\frac{Avg(11)}{averageslope}\right)} \tag{3.5}$$

We set threshold $T_r(i)$ for i^{th} row as,

$$T_r(i) = \begin{cases} 7.5 & i < (\frac{rlimit}{2}) \\ 5(\frac{rlimit - i}{rlimit} + 1) & (\frac{rlimit}{2}) < i < rlimit \end{cases} \tag{3.6}$$

Based on the experiments on the video frames collected from publication Geiger *et al.* (2011), we modelled average disparity value of road for each row, $\hat{Avg}(i)$ (as shown in the Eq.3.7) in the road scenes for those videos and average slope value as 0.4.

$$\boxed{\hat{Avg}(i) = 90 - (0.4 * (i - 15))} \tag{3.7}$$

We update the G matrix as,

$$G(i,j) = \begin{cases} 255 & if\ G(i,j) = 0\ and\ |D_s(i,j) - Avg(i)| > 5 \\ 0 & if\ G(i,j) = 255, |D_s(i,j) - Avg(i)| < T_r(i)\ and\ j \in \left[\frac{N_{th}}{3}\ \frac{2N_{th}}{3}\right] \\ 0 & if\ G(i,j) = 255, |D_s(i,j) - \hat{A}vg(i)| < T_r(i)\ and\ j \in \left[\frac{N_{th}}{3}\ \frac{2N_{th}}{3}\right] \\ G(i,j) & otherwise \end{cases} \tag{3.8}$$

Using Patch Size

We take the G matrix obtained after removal of errors by comparing with the threshold and build a graph such that we add an edge between p and q, if $q \in Ne(p)$ and $G(q) = G(p)$, where p,q are pixels in the image and q is the neighbour p. $Ne(p)$, Neighbour of p is defined as if $p = (i,j)$ then $q \in \{(i,j-1),(i-1,j),(i+1,j),(i,j+1)\}$. All the pixels in the image are nodes in the graph. We use Depth First search Traversal, DFS as explained in King and Launchbury (1995), to find the size of a patch i.e we go from a pixel p to its all connected nodes and count the size of the patch.

Algorithm 3 Removing Error using DFS

```
Require: Matrices $G$, Vs(Visited matrix) and St(empty stack)
 1: for i= 0 to $N_{rows}$ and j=0 to $N_{th}$ do
 2:   if Vs(i,j)==FALSE then
 3:     initiate count=0
 4:     DFS algo:
 5:     push $p_{itr}$=(i,j) to stack St
 6:     while St not empty do
 7:       pop the top most element in the St and assign it to $p$
 8:       $Vs(p) = TRUE$ and increment count by 1
 9:       if $\forall q \in Ne(p)$ and $G(q) = G(p_{itr})$ then
10:         push $q$ to the stack
11:       end if
12:     end while
13:     if $G(p_{itr}) = 255$ and count<400 then
14:       using DFS inverse values of the nodes connected to $p_{itr}$
15:     end if
16:     if $G(p_{itr}) = 0$ and count<5000 then
17:       using DFS inverse values of the nodes connected to $p_{itr}$
18:     end if
19:   end if
20: end for
```

In the algorithm 3, we initialize bool matrix Vs to "0 or FALSE" and St to a empty stack which can store pixels. We will iterate the "for" loop until the value of every point in the matrix Vs becomes "1 or TRUE". While iterating the loop we check the value of Vs at every point, if Vs(p)=FALSE then we use DFS and find the size of the patch by visiting all the connected nodes to p. After calculating the patch size we decide it as error or not by comparing it with threshold and update the value of pixels in matrix G correspondingly. In Fig.3.3 for the given left-right stereo image pair, we can clearly observe the white and black error patches are removed using the algorithms discussed in this subsection.

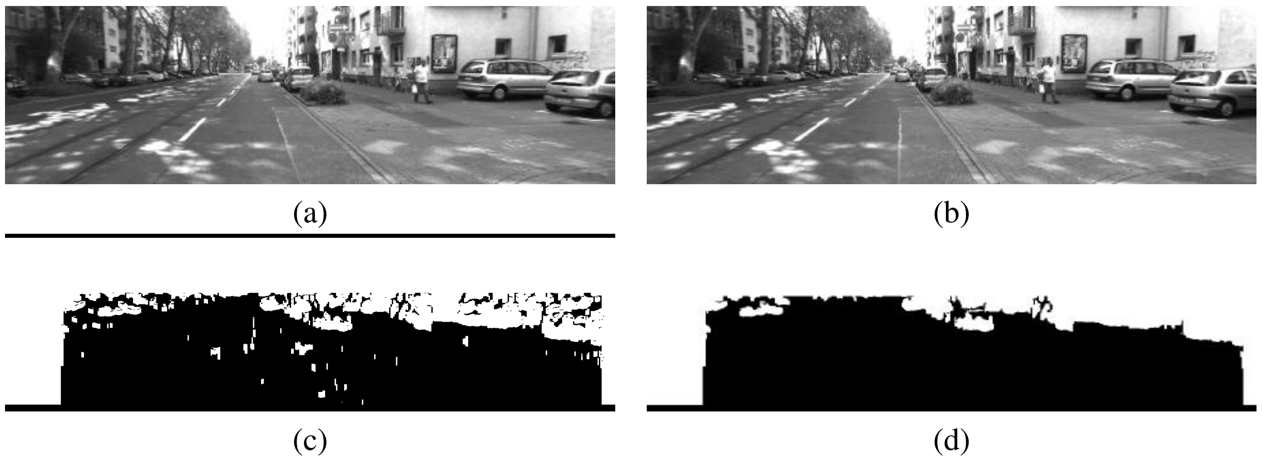


Figure 3.3: (a),(b) are rectified left and right stereo image pair respectively. (c)G, ground segmentation matrix or image before removing errors. (d) resultant G, ground segmentation matrix or image after removing errors using the algorithms in this subsection

CHAPTER 4

Object Detection And Labelling

4.1 Introduction

In this chapter we discuss how to detect the objects and label them using the disparity matrix, D_s and ground segmented matrix, G calculated in the chapter 3. As mentioned in section 2.4, we approximate real world objects with several planes and detect them for object segmentation. Disparity value behaviour for each plane will be different and there will be either smooth jumps or sharp jumps in disparity map at the boundaries of two different objects. We segment the objects in each row and each column both vertically and horizontally. We use these results for further object segmentation using our algorithm which is similar to graph-cut algorithms in Ding *et al.* (2001). We use different classifiers to classify different objects like cars and pedestrians. We label the objects and track them using optical flow.

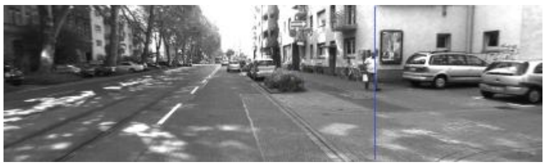
4.2 Vertical Segmentation

In this section we discuss how to segment the objects in each column by using the matrices D_s and G. In general when we go along the column from bottom to top in the image and corresponding top to bottom in the matrices D_s and G, we observe sudden jumps or disparity window in between the road map which implies an object. Here we detect those jumps by calculating difference between averages of disparity values in the windows in up and down directions. Let the matrix V_{seg} represent the vertical segmentation matrix. The following algorithm is used for segmenting objects in vertical direction in each column, where T_{v1} and T_{v2} are thresholds.

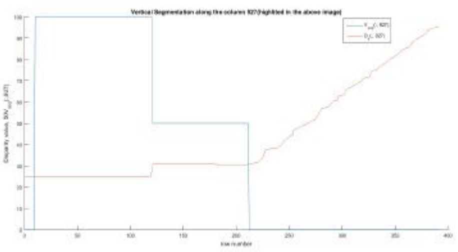
Algorithm 4 Vertical Segmentation

```
Require: Matrices $D_s$ and $G$
 1: for j= 1 to $N_{th}$ do
 2:     initialize count=0
 3:     for i= 0 to $N_{rows} - 9$ do
 4:         if $G(i, j) = 255$ then
 5:             if $G(i - 1, j) = 0$ then
 6:                 increment count by 1 and update $V_{seg}(i, j) = count$
 7:             else
 8:                 if $|D_s(i, j) - D_s(i - 1, j)| < T_{v1}$ then
 9:                     $V_{seg}(i, j) = V_{seg}(i - 1, j)$ as it belongs to same object
10:                 else
11:                     calculate average disparity for 5 pixels in the up and down direction.
                        Let those averages are $Up_{avg}$ and $Dn_{avg}$
12:                     if $|Up_{avg} - Up_{avg}| > T_{v2}$ then
13:                         if in the previous iteration same step was performed then don't in-
                            crement count otherwise increment count by 1 and update $V_{seg}(i, j)$
                            accordingly
14:                     else
15:                         it belongs to the same object so don't increment count and update
                            $V_{seg}(i, j) = V_{seg}(i - 1, j)$
16:                     end if
17:                 end if
18:             end if
19:         end if
20:     end for
21: end for
```

In Fig.4.1 we shown the resultant vertical segmentation along the column 927.



(a)



(b)

Figure 4.1: (a) left camera image (b) $50V_{seg}$, vertical segmentation matrix values along the column 927.

4.3 Horizontal Segmentation

We use a similar approach to the one in the previous section to segment objects horizontally in each row. We calculate average disparity value at every pixel for window size of 5 pixels in the left and right directions and obtain L_{avg} and R_{avg} as shown in Eq. 4.1. We use Dynamic programming to calculate these matrices.

$$L_{avg}(i,j) = \left(\frac{1}{5}\right) \sum_{k=j-4}^{j} D_s(i,k)$$
$$R_{avg}(i,j) = \left(\frac{1}{5}\right) \sum_{k=j}^{j+4} D_s(i,k) \tag{4.1}$$

Now we define LR_{avg} as,

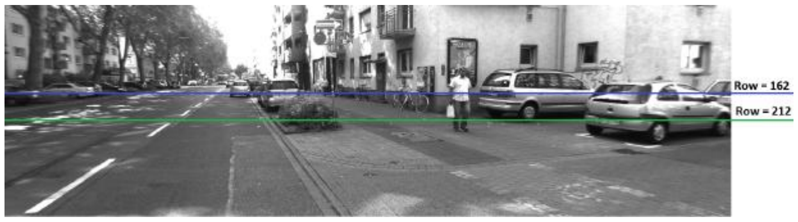
$$LR_{avg}(i,j) = R_{avg}(i,j) - L_{avg}(i,j) \tag{4.2}$$

Let H_{seg} be the horizontal segmentation matrix which is obtained using the following algorithm. The algorithm for segmenting objects along horizontal in each row, in given next,

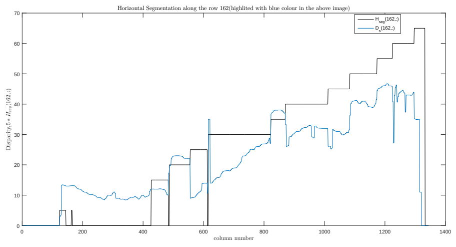
Algorithm 5 Horizontal Segmentation

```
Require: Matrices $LR_{avg}$, $D_s$, $G$ and $Tmp$ (temporary matrix)
 1: for i= 0 to $N_{rows}$ do
 2:     initialize count=0
 3:     for j= 4 to $N_{th} - 4$ do
 4:         if $|D_s(i,j) - D_s(i,j-1)| < T_{h1}$ then
 5:             (i,j) belongs to the same object of (i,j-1). $Tmp(i,j) = Tmp(i,j-1)$
 6:         else
 7:             if $|LR_{avg}(i,j)| > T_{h2}$ then
 8:                 if in previous iteration same step was performed then don't increment
                    count as this pixel belongs to boundary, otherwise increment the count as
                    this doesn't belong the same object. Thus update $Tmp(i,j)$ accordingly.
 9:             else
10:                 (i,j) belongs to the same object of (i,j-1). $Tmp(i,j) = Tmp(i,j-1)$
11:             end if
12:         end if
13:     end for
14: end for
15: Combine $G$ and $Tmp$ to get updated horizontal segmentation matrix, $H_{seg}$
```

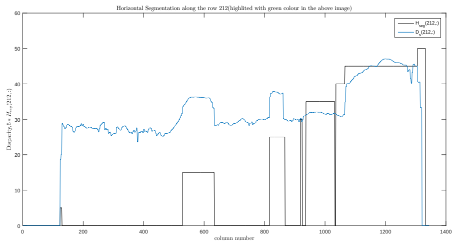
where T_{h1} and T_{h2} are thresholds. In Fig.4.2 we shown the horizontal segmentation along the rows 162 and 212.



(a)



(b)



(c)

Figure 4.2: (a) left camera image (b) H_{seg}, horizontal segmentation along the row 162 in the image. (c) H_{seg}, horizontal segmentation along the row 212 in the image.

4.4 Object Segmentation

In this section we use matrices D_s, G, V_{seg} and H_{seg} to detect the objects by constructing a graph and using a cost function that cuts edges in the graph similar to graph cut algorithms. We construct the graph with all pixels in the image as nodes and add undirected edges between any two nodes p and q if $q \in Ne(p)$, $G(q) = G(p)$ and either $V_{seg}(q) = V_{seg}(p)$ if p and q are in same column or $H_{seg}(q) = H_{seg}(p)$ if p and q are in same row. Let $\mathcal{O}$ represent the list of the objects to be detected in the image and $\mathcal{O}(i)$ represent i^{th} object. We use the below cost function,$\mathcal{C}$ to cut the edge between p and q, where already $p \in \mathcal{O}(i)$. Let c_i, be the centroid of the object,$\mathcal{O}(i)$.

$$\mathcal{C}(q, \mathcal{O}(i)) = |D_s(q) - D_s(c_i)| + (\mu * (Dist(q, c_i))) \tag{4.3}$$

where $Dist(q, c_i)$ calculates the distance between q and c_i.

We use Breadth First Search, BFS traversal explained in Silvela and Portillo (2001) and cost function in Eq. 4.3 to segment object in the constructed graph. Let O_{seg}, object segmentation matrix obtained using this algorithm, where, if $O_{seg}(p) = O_{seg}(q)$

for pixels p and q then it implies that they belong to same object. The algorithm for object segmentation, in given next, where T_{ob1} and T_{ob2} are thresholds.

Algorithm 6 Object Segmentation

Require: (a)Matrices D_s, G, V_{seg} and H_{seg}. (b) Functions $\mathcal{C}()$ and $Dist()$ (distance function). (c) Qu_o (empty queue to store pixels in BFS algo), $Vist$ (visited matrix)

```
1:  initialize count=0 (object number)
2:  for i= 15 to $N_{rows} - 9$ do
3:      for j= 4 to $N_{th} - 4$ do
4:          initialize obcount=0 (number of pixel in object)
5:          if Vist(i,j)=0 then
6:              $p_{itr} = (i, j)$ , initialize $c_{itr} = (0, 0)$(centroid) and push $p_{itr}$ to $Qu_o$
7:              increment count and add $p_{itr}$ to $\mathcal{O}(count)$
8:              BFS algorithm:
9:              while $Qu_o$ is not empty do
10:                 assign top element in $Qu_o$ to p and pop it from it
11:                 $Vist(p) = 1$ and add $p$ to $\mathcal{O}(count)$
12:                 calculate centroid $c_{itr}$ of the object $\mathcal{O}(count)$
13:                 increment obcount and $O_{seg}(p) = count$
14:                 if $\forall q \in Ne(p)$ and $G(q) = G(p)$ then
15:                     if p and q are in same row, and $H_{seg}(q) = H_{seg}(p)$ then
16:                         calculate $\mathcal{C}(q, \mathcal{O}(count))$
17:                     end if
18:                     if p and q are in same column, and $V_{seg}(q) = V_{seg}(p)$ then
19:                         calculate $\mathcal{C}(q, \mathcal{O}(count))$
20:                     end if
21:                     if $\mathcal{C}(q, \mathcal{O}(count)) < T_{ob1}$ then push q into $Qu_o$
22:                 end if
23:             end while
24:             if obcount<$T_{ob2}$ then
25:                 use BFS algorithm change $O_{seg}$ value for all the pixels belonging to
                    $\mathcal{O}(count)$ and remove them from $\mathcal{O}(count)$
26:                 decrement the count
27:             end if
28:         end if
29:     end for
30: end for
```

All the pixels belonging to the same object will have same value in the matrix, O_{seg}. In Fig.4.3, to observe different objects clearly, values of O_{seg} are mapped to gray scale values [0 255].

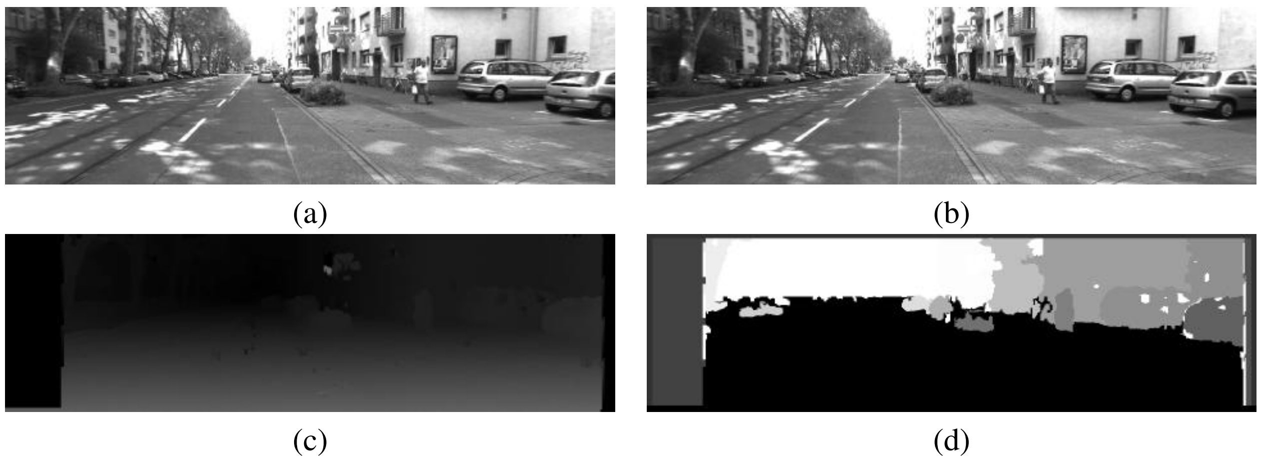


Figure 4.3: (a),(b) are rectified left and right stereo image pair respectively. (c) disparity map after smoothing. (d) O_{seg}, object segmentation matrix which was obtained using the algorithm mapped to gray scale values [0 255]

4.5 Labelling Objects

In this section, we assign labels to detected objects in section 4.4. We pass the image to different classifiers for detection of pedestrians and cars which gives a rectangle as output surrounding the detected elements. We will match the detected rectangles to the object in $\mathcal{O}$ with maximum overlap region with detected rectangle.

4.5.1 Pedestrian Detection

Pedestrians in the image are detected using HOG based pedestrian method explained in Dalal and Triggs (2005) . We have inbuilt class "HOGDescriptor()" where we set parameters and the following two steps yield pedestrian detection.

- hog.setSVMDetector(cv::HOGDescriptor:: getDefaultPeopleDetector());
- hog.detectMultiScale(); (you should provide parameter, inputs and outputs to this function)

After getting the FR, the list of detected rectangles of pedestrians, we use the following algorithm to match them with the objects in $\mathcal{O}$.

Algorithm 7 Pedestrian Labeling

```
Require: Lists $FR$, $\mathcal{O}$
 1: for i=1 to size of $FR$ do
 2:     intialize maxarea=0 and outindex=-1
 3:     for k=1 to size of $\mathcal{O}$ do
 4:         r= insterction rectangle of $FR(i)$ and $Rect(\mathcal{O}(k))$
 5:         if maxarea < area of r then update maxarea= area of r and outindex=k
 6:         if area of r > 0.6*(area of $FR(i)$) and area of r > 0.8*(area of $\mathcal{O}(k)$) then
 7:             update maxarea= area of r and outindex=k and break the loop
 8:         end if
 9:     end for
10:     label the object $\mathcal{O}(outindex)$ as pedestrian
11: end for
```

4.5.2 Car Detection

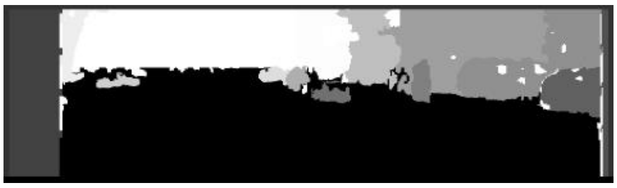
Cars in the image are detected using Haar Cascade Classifier in the OpenCV libraries. Haar Cascade Classifier in OpenCV uses the techniques explained in Viola and Jones (2001) and Lienhart and Maydt (2002). We need to provide trained ".xml" file for detection of cars. "(CvHaarClassifierCascade*) cvLoad()" in built function loads trained vectors stored in the ".xml" file into the variable cascade. "cvHaarDetectObjects()" in built function in OpenCV gives the output list of cars, FC. We match every detected car in the list, FC with the objects which are not labelled as pedestrian in the list $\mathcal{O}$.

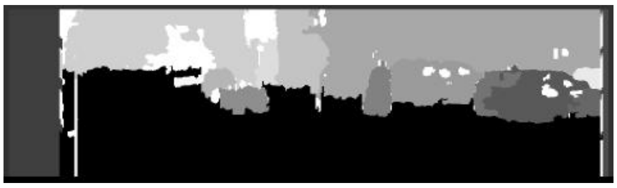
Algorithm 8 Car Labeling

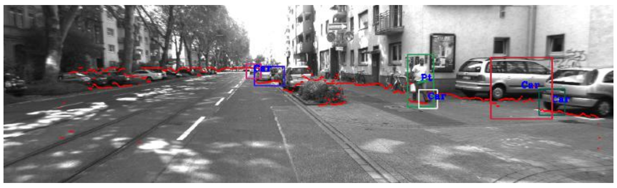
```
Require: Lists $FC$, $\mathcal{O}$
 1: for i=1 to size of $FC$ do
 2:     intialize maxarea=0 and outindex=-1
 3:     for k=1 to size of $\mathcal{O}$ do
 4:         if label of $\mathcal{O}(k)$ is not pedestrian then
 5:             r= insterction rectangle of $FC(i)$ and $Rect(\mathcal{O}(k))$
 6:             if maxarea < area of r then update maxarea= area of r and outindex=k
 7:             if area of r ==(area of $\mathcal{O}(k)$) and (area of $\mathcal{O}(k)$)>3000 then
 8:                 update maxarea= area of r and outindex=k and break the loop
 9:             end if
10:         end if
11:     end for
12:     if (maxarea>0.25*area of $FC(i)$)label the object $\mathcal{O}(outindex)$ as car
13: end for
```

4.6 Tracking

We track the objects and calculate their velocities using feature points and optical flow. Let $I_{l,t-1}$ be the previous and $I_{l,t}$ be the current left images of the left-right stereo image pairs. Using "goodFeaturesToTrack()" inbuilt function in OpenCV we detect good features in previous left image, $I_{l,t-1}$ which can be tracked. Let these good features be $\mathcal{F}p_{prev}$. We use $\mathcal{F}p_{prev}$ and in built function "calcOpticalFlowPyrLK()" in OpenCV to compute optical flow and tracked features in the current left image for $\mathcal{F}p_{prev}$. "calcOpticalFlowPyrLK()" function uses the iterative Lucas-Kanade method with pyramids to compute optical flow (which was explained in Bouguet (1999)). Let those tracked features be $\mathcal{F}p_{curr}$ and objects detected for the previous left-right stereo image pair is $\mathcal{PO}$. We link the objects in both $\mathcal{PO}$ and $\mathcal{O}$ with help of $\mathcal{F}p_{prev}$ and $\mathcal{F}p_{curr}$ by matching maximum number of features common between the objects in previous and current stereo image pairs. In Fig.4.4 we show the final results of a scene where tracking is detected clearly for the objects like pedestrian and cars.


(a)


(b)


(c)

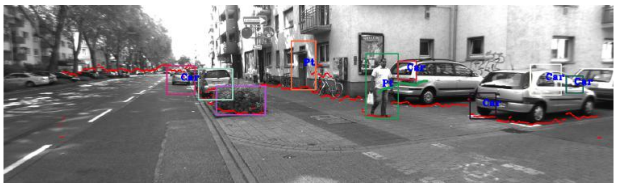

(d)

Figure 4.4: (a) O_{seg}, object segmentation matrix at time=2,(b) O_{seg}, object segmentation matrix at time=20. (c) final result at time=2. (d) final result at time=20

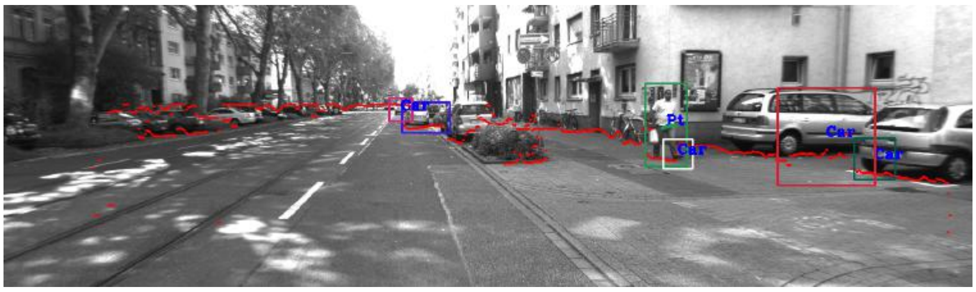
CHAPTER 5

Displaying Results And Performance Of Algorithm

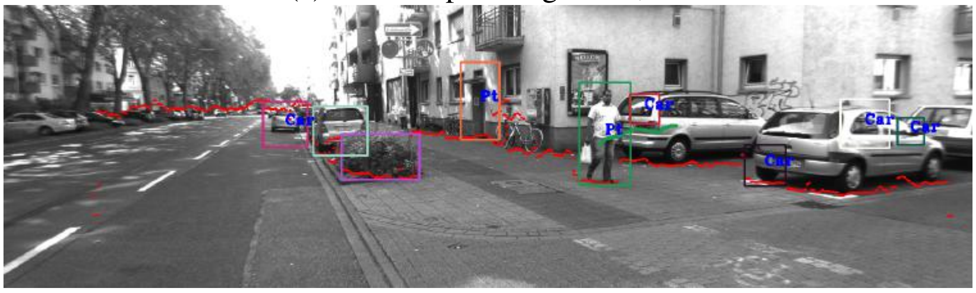
Left-right stereo images used to conduct experiments are collected from the publication Geiger *et al.* (2011). We ran our algorithm on four videos of each around one and half minutes. Video-1 contains 1200 frames, video-2 contains 1000 frames, video-3 contains 300 frames and video-4 contains 1200 frames.

5.1 Displaying Results

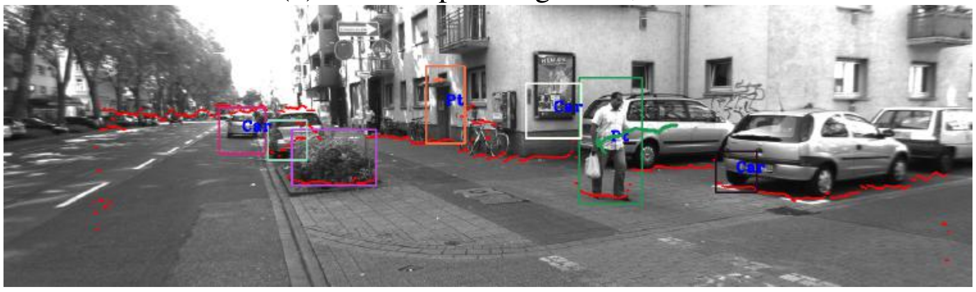
We use G calculated in chapter 3, and display the road upto where car can move safely (as shown in Fig. 5.1). We display the objects which are labelled in the current left-right stereo image pair. Before displaying them we assign colour to the object if not assigned earlier. We also display objects which are labelled as pedestrians or cars in the earlier images and not labelled in the current left-right stereo image pair, but on a condition that their size should be less than 15000 pixels, to differentiate cars and pedestrians from building and background. For example in the first three images of Fig. 5.1 pedestrian was labelled as "Pt" and in next two images pedestrian was not labelled in next two, but in all images the pedestrian was surrounded by same green colour box. We displayed pedestrian in the last two images in Fig. 5.1, although the pedestrian was not detected by the classifier as pedestrian size is than 15000 pixels and detected by classifier in the previous frames. We can clearly observe in Fig. 5.1 building was not displayed even though it was detected as car in some images since it's size greater than 15000 pixels.



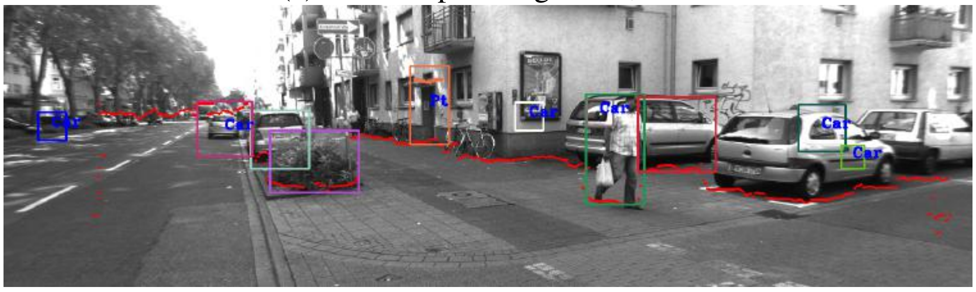
(a) Final output image time,t=2



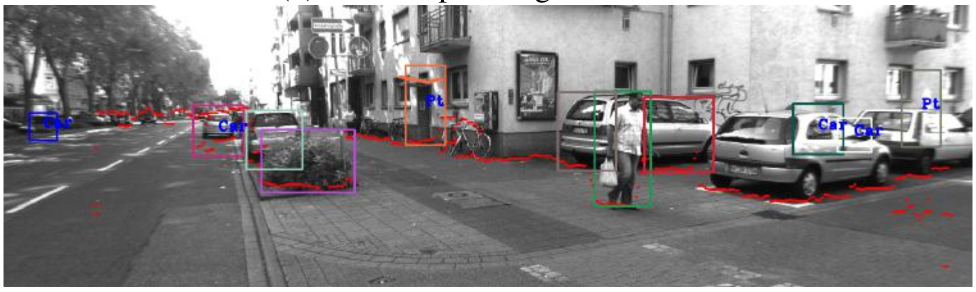
(b) Final output image time,t=20



(c) Final output image time,t=30



(d) Final output image time,t=35

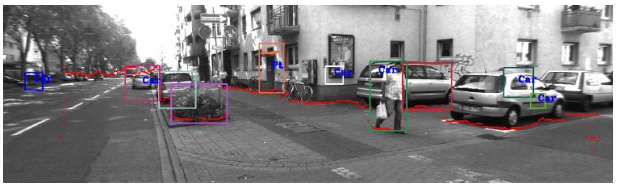


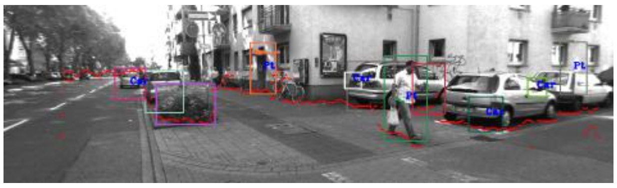
(e) Final output image time,t=38

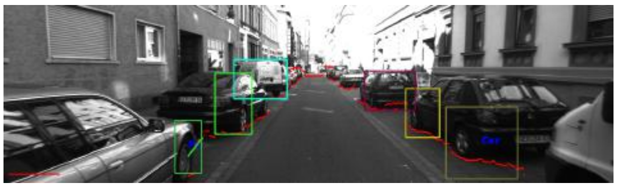
Figure 5.1: Final output images of a scene at different time instances

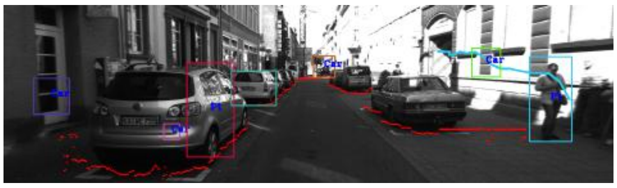
Results for Video-1

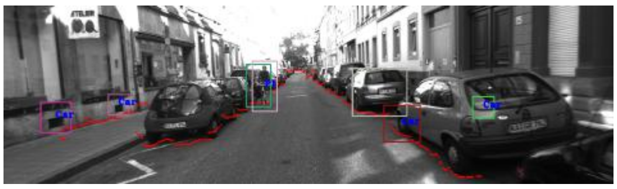
Video-1 was taken in narrow urban roads during day time. In Fig. 5.2 (a) and (b) images, the pedestrians and cars have the same colour of rectangle by which they are circumscribed. In the image Fig. 5.2 (d), we displayed tracking of the pedestrian walking on the foot path. We tracked the cyclist who is coming in the opposite direction on the road as shown in Fig. 5.2 (e) and (f).

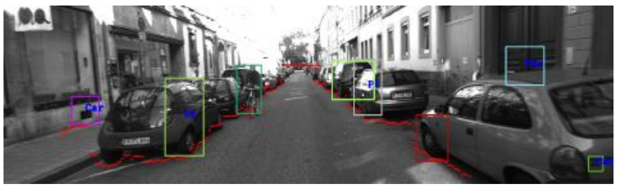

(a)


(b)


(c)


(d)


(e)


(f)

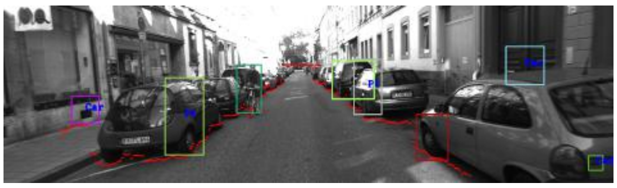
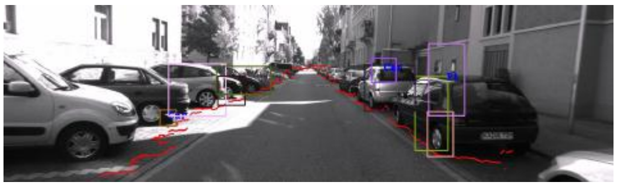

(g)

Figure 5.2: results for video-1

Results for Video-2

Video-2 was taken in narrow urban roads during day time. In Fig. 5.3, we can observe that the occluded cars are detected as different cars by the algorithm.


(a)

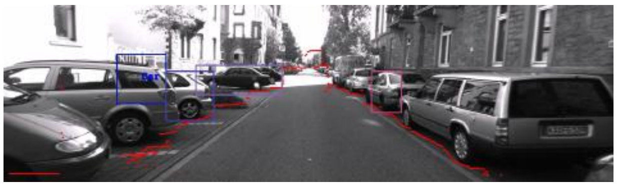
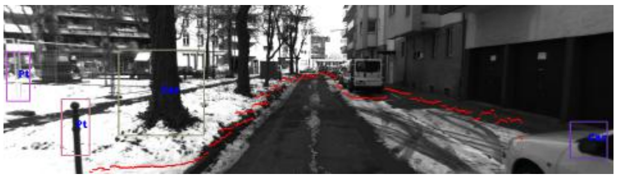
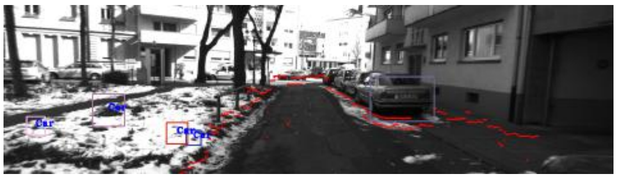

(b)

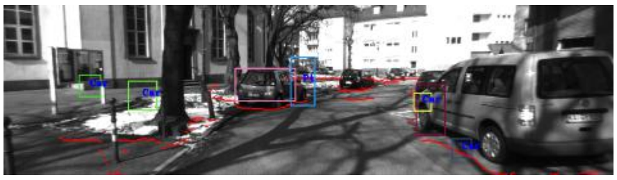
Figure 5.3: results for video-2

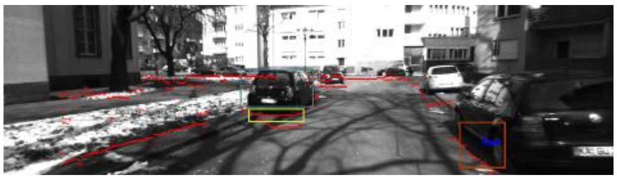
Results for Video-3

Video-3 was taken in urban roads of cold countries during winter. In Fig. 5.4, we can observe snow in the sides of the road is not include the road. So it is alerting the driver about snow region in the sides of the road. By the results in Fig. 5.4 we can conclude that this algorithm will work in the urban roads of cold countries.


(a)


(b)


(c)


(d)

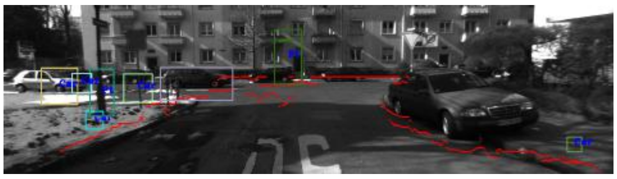
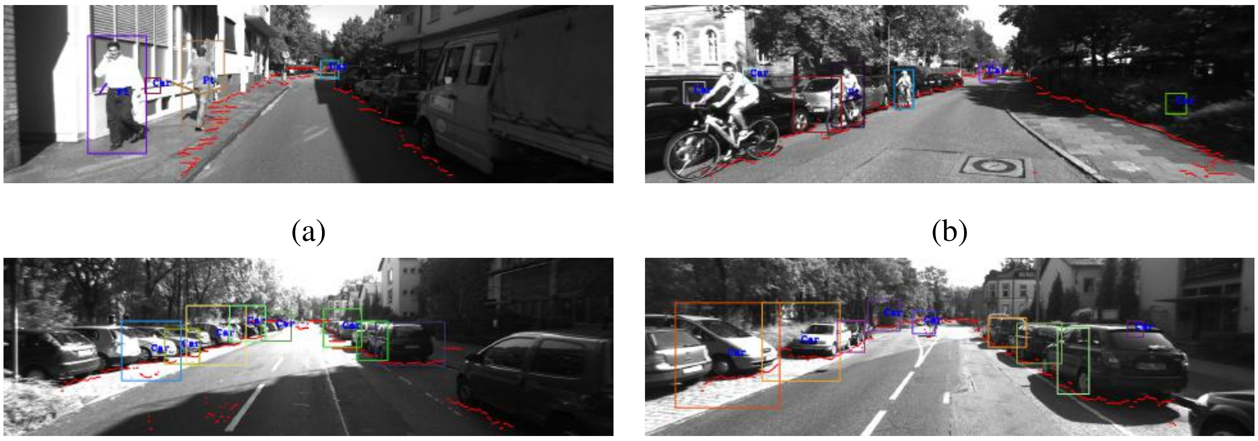

(e)

Figure 5.4: results for video-3

Results for Video-4

Video-4 was taken in wide urban roads during day time. In Fig. 5.5(a), footpath was not included in the road, and we can observe that pedestrians on the road are detected and tracked. The cyclists in Fig. 5.5(b) are also detected and tracked by the algorithm.

In Fig. 5.5(c) and (d) divider in the middle of the road is not included in the road, and cars parked on the side of the are detected by the algorithm.



(a) (b)

(c) (d)

Figure 5.5: results for video-4

5.2 Performance

5.2.1 Time complexity

In this section we discuss about time complexity of the algorithm. As we saw this algorithm contains different steps like column smoothing, ground, vertical, horizontal and object segmentation. The time complexity of all these is $O\left(N_{rows}N_{th}\right)$. The algorithm also contains tracking and its time complexity is $O\left(2 * (size\ of\ \mathcal{F}p_{curr}) * (\boxed{size\ of\ \mathcal{O}})\right)$ where for the images used here $size\ of\ \mathcal{F}p_{curr} \leq 1500$ and $size\ of\ \mathcal{O} \leq 40$.

We ran the algorithm on the videos mentioned in the section 5.1 in our laptop which has 8GB RAM and intel-i7 processor. If no task is running in the back ground, here are the time taken by different modules of the algorithm.

- For getting disparity using StereoSGBM : 467.179ms
- For row smooth algorithm : 0.346ms
- For column smooth algorithm : 8.352ms
- For ground segmentation algorithm (including removing errors) : 193.302ms
- For vertical segmentation algorithm : 5.242ms
- For horizontal segmentation algorithm : 14.679ms

- For object segmentation algorithm : 75.800ms
- HOG and Haar cascade classifiers : 467.624ms
- For optical flow and tracking algorithm : 45.707ms
- Total time taken by the algorithm : 1291.497ms

If processor was running some other tasks in the background, following are the time taken by different modules of the algorithm.

- For getting disparity using StereoSGBM : 1294.991ms
- For row smooth algorithm : 1.176ms
- For column smooth algorithm : 28.857ms
- For ground segmentation algorithm (including removing errors) : 720.876ms
- For vertical segmentation algorithm : 19.962ms
- For horizontal segmentation algorithm : 53.047ms
- For object segmentation algorithm : 280.652ms
- HOG and Haar cascade classifiers : 1457.303ms
- For optical flow and tracking algorithm : 125.439ms
- Total time taken by the algorithm : 4027.046ms

5.2.2 Failures

This algorithm shows errors when there is a transition from shadow region to bright sunny region as shown in the Fig.5.6

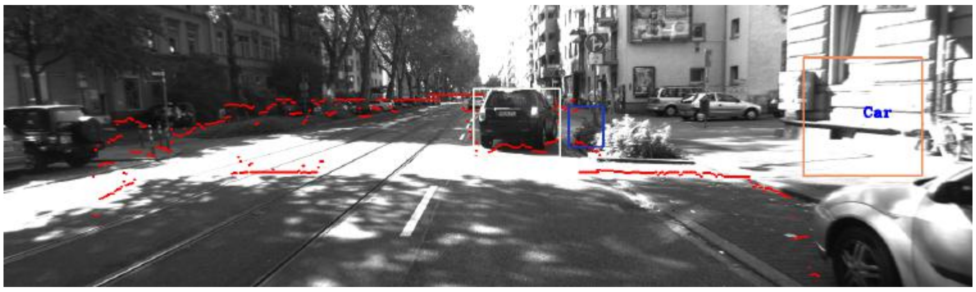


Figure 5.6: error in ground detection due to transition between shadow and bright regions.

We made an assumption in chapter 3 that middle of the image is road. So this algorithm can not remove errors in ground segmentation while car is turning right or left, as shown in the Fig. 5.7 we can clearly observe errors in ground estimation in the middle of the image i.e it included some parts of the car in ground.

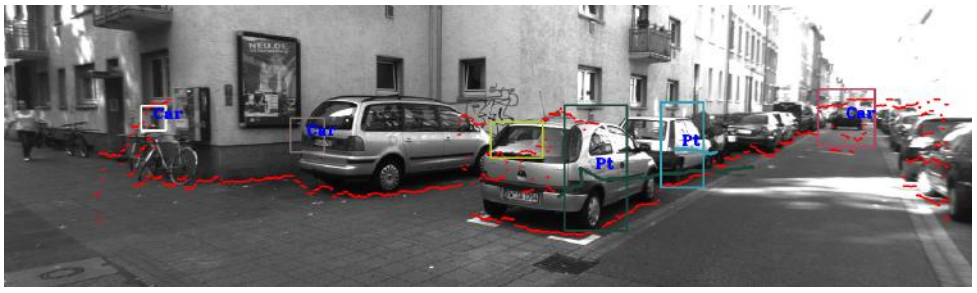


Figure 5.7: error in ground detection while car is turning right

This algorithm divides an object into 2 or 3 objects if they are very near. In the below Fig.5.8 car highlighted in the box is divided into two objects.

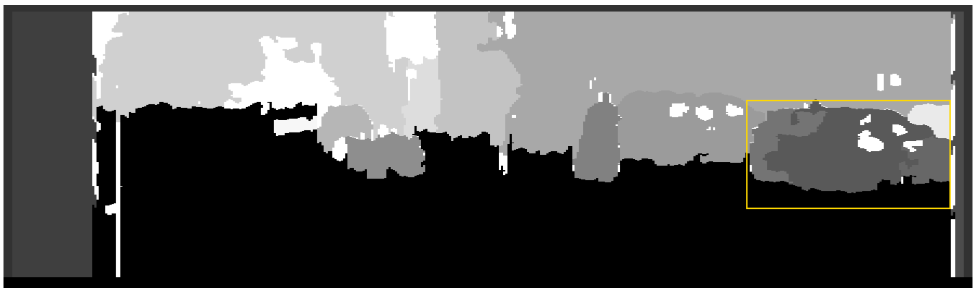


Figure 5.8: error in object detection

CHAPTER 6

Conclusions

This project is an attempt to build a system for car which can detect pedestrians, cars, obstacles and show the safe road area to the driver. As discussed in the section 5.1, this algorithm will yield good results in narrow road, wide road and on the road of cold countries. Time taken by the algorithm can be reduced from 2 seconds to $\frac{1}{20}$ seconds. As explained OpenCV performance of different functions can be improved if we use GPU instead CPU. In Fig. 6.1 we can observe speed-up factor of different modules when we use GPU instead of CPU.

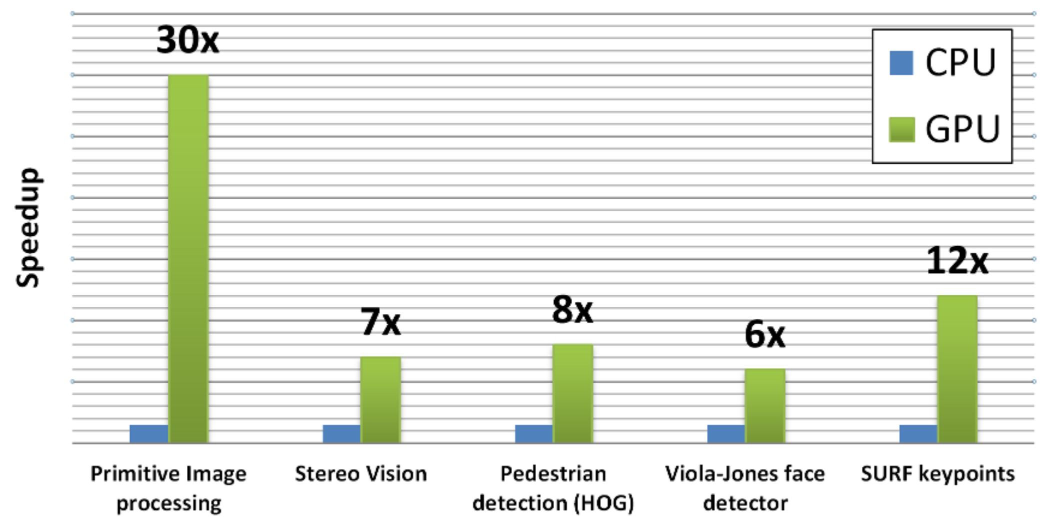


Figure 6.1: performance of CPU Vs GPU for different modules

Thus, by moving from CPU to GPU we can we can make algorithm 8 times faster. Another advantage in using GPU, we can run the modules like Vertical and Horizontal segmentation, and Object segmentation, classifier and tracking algorithm in parallel. By running modules in parallel we can make the algorithm 5 times faster. Hence, we can run the algorithm in $\frac{1}{20}$ for a single left-right stereo image pair.

We can conclude that in the real time this algorithm can run at 20 frames per second.

REFERENCES

1. **Benenson, R.**, **M. Mathias**, **R. Timofte**, and **L. V. Gool** (2012). Fast stixel computation for fast pedestrian detection. *Computer Vision-ECCV*, 11–20.

2. **Benenson, R.**, **R. Timofte**, and **L. V. Gool** (2011). Stixels estimation without depth map computation. *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference*, 2010–2017.

3. **Birchfield, S.** and **C. Tomasi** (1998). A fast string searching algorithm. *Proc. Sixth IEEE Int',l Conf. Computer Vision*, 1073–1080.

4. **Bouguet, J. Y.** (1999). Pyramidal implementation of the lucas kanade feature tracker description of the algorithm. *technical report*.

5. **Boyer, R. S.** and **J. S. Moore** (1977). A fast string searching algorithm. *Communications of the ACM*, **20**, 762–772.

6. **Dalal, N.** and **B. Triggs** (2005). Histograms of oriented gradients for human detection. *Computer Vision and Pattern Recognition, CVPR*, **1**, 886 – 893.

7. **Ding, C. H. Q.**, **X. He**, **H. Zha**, **M. Gu**, and **H. D. Simon** (2001). A min-max cut algorithm for graph partitioning and data clustering. *Proceedings IEEE International Conference on Data Mining, ICDM 2001*, 107 – 114.

8. **Ewerth, R.**, **M. Schwalb**, **P. Tessmann**, and **B. Freisleben** (2004). Estimation of arbitrary camera motion in mpeg videos. *Pattern Recognition, ICPR 2004. Proceedings of the 17th International Conference*, **1**, 512–515.

9. **Geiger, A.**, **J. Ziegler**, and **C. Stiller** (2011). Stereoscan: Dense 3d reconstruction in real-time. *Intelligent Vehicles Symposium (IV)*.

10. **Gong, Y.**, **R. S. Johnston**, **C. D. Melville**, and **E. J. Seibel** (2014). Axial-stereo 3d optical metrology of internally machined parts using high-quality imaging from a scanning laser endoscope. *Optomechatronic Technologies (ISOT), 2014 International Symposium*, 228 –231.

11. **Hirschmuller, H.** (2008). Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **30**, 328 – 341.

12. **Hu, Z.** and **K. Uchimura** (2005). U-v-disparity: an efficient algorithm for stereovision based scene analysis. *Intelligent Vehicles Symposium, IEEE*, 48 – 54.

13. **King, D. J.** and **J. Launchbury** (1995). Structuring depth-first search algorithms in haskell. *POPL '95 Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 344–354.

14. **Labayrade, R.**, **D. Aubert**, and **J. P. Tarel** (2002). Real time obstacle detection in stereovision on non flat road geometry through v-disparity representation. *Intelligent Vehicle Symposium, 2002. IEEE*, **2**, 646–651.

15. **Lienhart, R.** and **J. Maydt** (2002). An extended set of haar-like features for rapid object detection. *IEEE ICIP*, **1**, 900–903.

16. **Murphey, Y. L.**, **J. Chen**, **J. Crossman**, **J. Zhang**, **P. Richardson**, and **L. Sieh** (2000). Depthfinder, a real-time depth detection system for aided driving. *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE*, 122 –127.

17. **OpenCV** (). documentation. `http://opencv.org/platforms/cuda.html`.

18. **Pfeiffer, D.** and **U. Franke** (2011). Towards a global optimal multi-layer stixel representation of dense 3d data. *BMVC*.

19. **Silvela, J.** and **J. Portillo** (2001). Breadth-first search and its application to image processing problems. *IEEE Transactions on Image Processing*, **10**, 1194 – 1199.

20. **Viola, P.** and **M. J. Jones** (2001). Rapid object detection using a boosted cascade of simple features. *IEEE CVPR, 2001*, **1**, 511–518.

21. **Wang, R.** and **T. Huang** (1999). Fast camera motion analysis in mpeg domain. *Image Processing, ICIP 99. Proceedings. 1999 International Conference*, **3**, 691–694.

22. **Won, K. H.**, **J. Son**, and **S. K. Jung** (2014). Stixels estimation through stereo matching of road scenes. *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, 116–120.

23. **Zhang, Z.** (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1330 – 1334.