Visibility Sensing for Autonomous Streetlight Control for Smart Cities

Krish Hemant Mhatre, Department of Intelligent Systems Engineering, School of Informatics, Computing and Engineering, Indiana University

***Abstract*— Visibility readings can be used to control streetlight system of a city/town. This study aims to build an Autonomous Streetlight Control system for Smart Cities. The main purpose of streetlight for vehicles is to make the lanes visible, therefore the visibility readings is taken by considering the visibility of white and yellow lane lines. This visibility readings could be further used to control the streetlights. It can be further developed to control the intensity too, to save energy.**

I. Introduction

The project involves sensing the visibility on the roads using the lane white and yellow lines using a camera. The data is compared to initial image data (100% visibility) (to detect specific distance for visibility, the 'interested area' part in the visibility.py file could be modified). The visibility percentage obtained after the comparison is then sent to a server, which receives it and updates a CSV File with exact Date and Time (test.mosquitto.org is used as a test server). This data can be further used in many applications, Automatic Streetlight Control could be one of those. In order to test the functioning of this study, Raspberry Pi could be used.

II. Related Work

A. Smart Autonomous Street Light Control System

B. Analysis of Lane Detection Techniques using OpenCV

III. Math

$$H = \begin{cases} 60^\circ \times \left(\frac{G'-B'}{\Delta} mod 6\right) & , Cmax = R' \\ 60^\circ \times \left(\frac{B'-R'}{\Delta} + 2\right) & , Cmax = G' \\ 60^\circ \times \left(\frac{R'-G'}{\Delta} + 4\right) & , Cmax = B' \end{cases}$$

(Equation 1)

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

(Equation 2)

$$V = C_{max}$$

(Equation 3)

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)}$$

(Equation 4)

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

(Equation 5)

Result = Region of Interest Mask && Canny Edges Image

(Equation 6)

visibility = diff.mean()/actual.mean() * 100

(Equation 7)

***Research supported by the Intelligent Systems Engineering Department of School of Informatics, Computing, and Engineering at Indiana University Bloomington**

IV. Visibility Sensing

A. Images Used

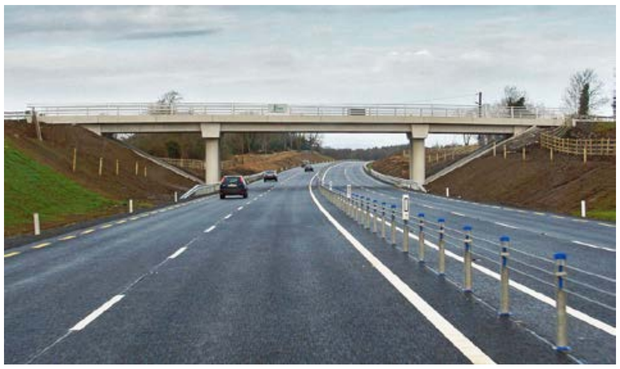

Figure 1.1

This is the image (Figure 1.1) is considered as clear and 100% visible. The rest of the images are compared with this one. The visibility generated is in form of 'percentile'. This image is given as an initial argument. In an actual application model, this image will be provided during the initial setup.

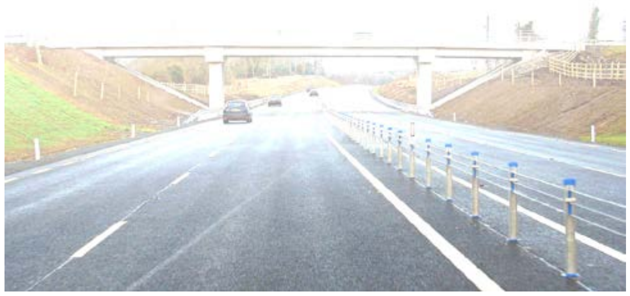

Figure 1.2

This image (Figure 1.2) is compared to the Initial image to calculate visibility. This image is generated using the Python code by blending the test image and a fog image for the purpose of testing. In an actual application model, this will be the image provided by Raspberry Pi every hour as an argument.

B. Grayscale and HSV

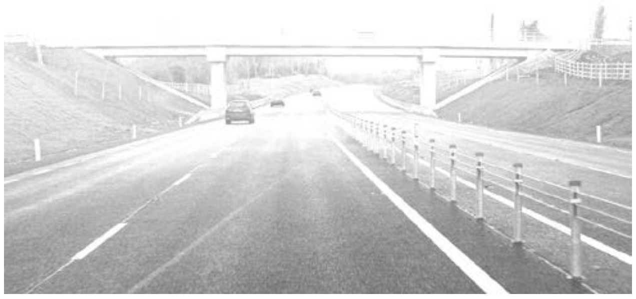

Figure 2.1

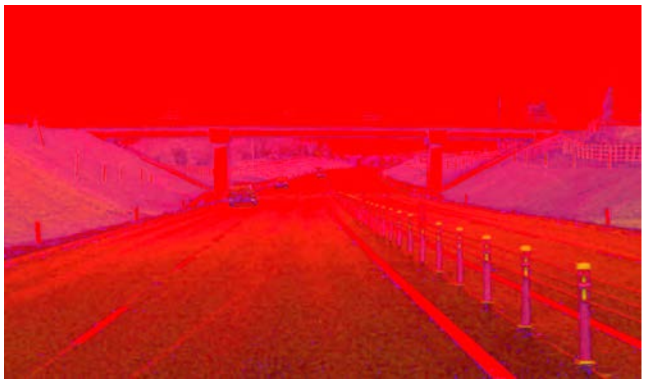

Figure 2.2

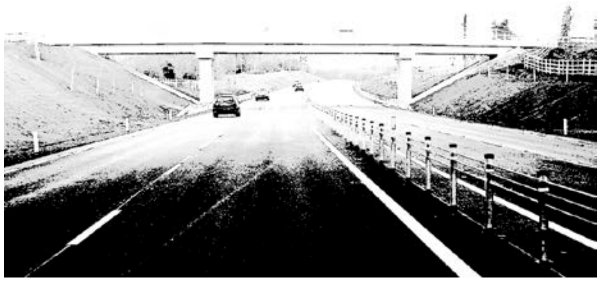

Figure 2.3

The raw image obtained is in 3 channel pixel values i.e. RGB format. In order to process the image to detect the white lane lines, we convert the raw image into a single channel i.e. grayscale format (Figure 2.1). In order to detect yellow lane lines as well, we convert the raw image into a HSV format (Equations (1), (2) and (3), Figure 2.2). We apply a bitwise AND operation on the grayscale image and the HSV image to detect white and yellow color (Figure 2.3).

C. Gaussian Blur

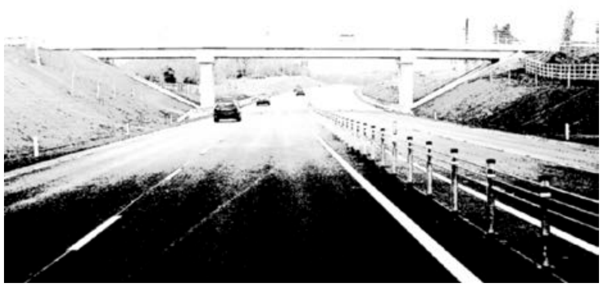

Figure 3.1

The image we obtain now is subject to noise. To deal with such noise, it is essential to apply Gaussian Blur (Equation (4)). Gaussian Blur averages out nearby pixel values which suppresses the noise in the image. We apply Gaussian Blur with a kernel size equal to 5 (Figure 3.1).

D. Canny Edges

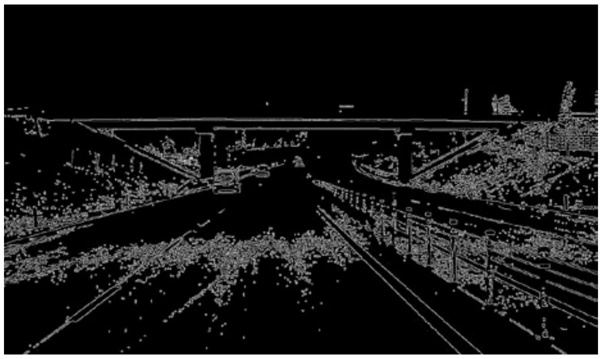
Figure 4.1

The most essential part of the algorithm is detecting canny edges in the image. Canny Edge detector computes the gradient and detects the regions with extreme difference (Equation (5)). We apply canny image detector with minimum threshold of 50 and maximum threshold of 150 (Figure 4.1).

E. Region of Interest

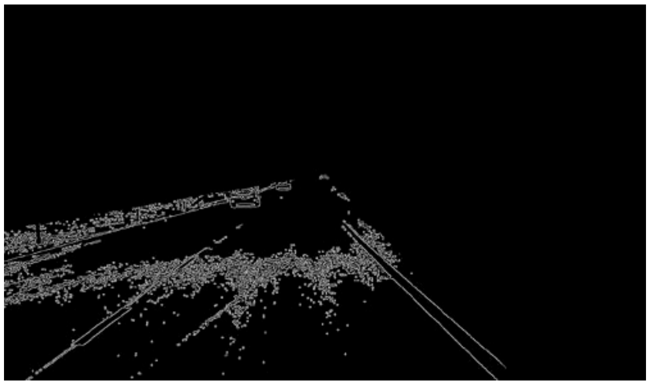
Figure 5.1

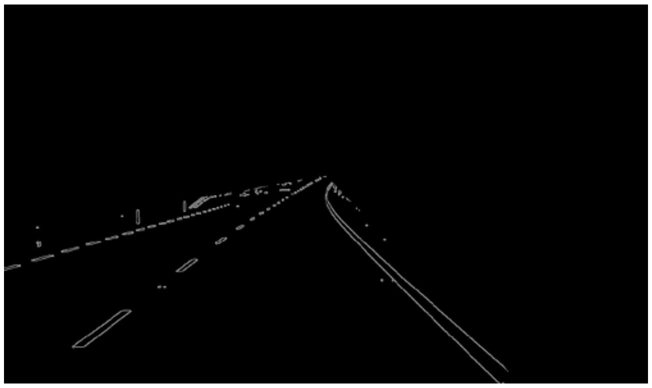
Figure 5.2

The canny image gives clear picture of the regions with steep directional derivatives. We crop out the top-right part of the image at this point to make it easier for the process of lane detection (Figure 5.1).

This is the step where the distance for which the visibility percentage to be calculate, can be set. If the region of interest is set to the upper part of the image, then we test the visibility for a longer distance.

This step could be also done in beginning of the process before processing the image. This gives a clear picture to us for the region where we expect the lanes to exist.

The region of interest of the initial image is also obtained (Figure 5.2).

F. Resulting Image

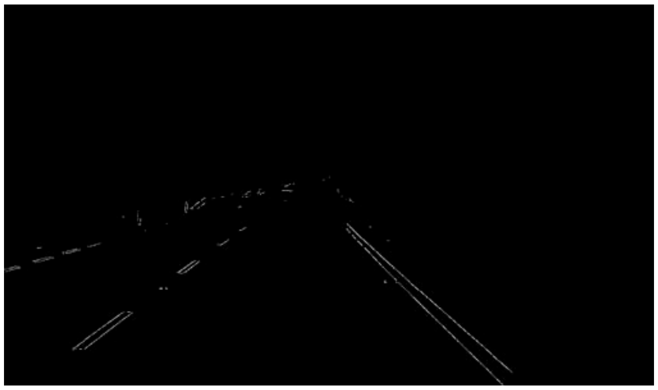
Figure 6.1

We use a bitwise AND on the region of interest of the initial image and the test image (Equation (6)). The resulting image (Figure 6.1) displays the common elements between the two images. This helps us calculate the visibility.

G. Visibility Calculation

The visibility percentage of a certain distance, which is set in the Region of Interest step, can be determined using the mean of the pixel values of the processed initial image (Figure 5.2), which is considered 100% visible and the mean of the pixel values of the processed and compared test image (Figure 6.1). This procedure uses Equation (7) to get the visibility percentage.

V. Transmission of Data

This project uses the MQTT Protocol to transmit the data i.e. the visibility percentages after every 10 minutes. For this purpose, we use MQTT Test Server (Appendix A).

VI. Result

The data containing visibility percentage is received on the server. The data is, then, saved as a .csv file which also contains the date and time of the readings taken.

VII. Future Work

A. Use of Raspberry Pi

In order to use/test this system, Raspberry Pi (Model 3A, 3B, 3B+, Zero or Zero W) can be used. If any other microcontroller is to be used which is not compatible with Python, the image can be transmitted by the device and the image processing can be done on the server.

The device should be deployed, preferably on the right side of the street for which data needs to be transmitted. The device should be placed at least 2 meters above the ground.

B. Testing

The testing for this system can only be done when the device (Raspberry Pi) is deployed. The testing will also require good WiFi connection.

C. Building a Network

In order to build a network of such devices for further use, each device needs to be assigned with a Unique Identity Number. The network can collect data every 10 minutes and will be posted on the server.

D. GUI Support

The data can be displayed on the server side using GUI Support. The project, in future, aims to build a Geo-spatial Information Systems (GIS) based GUI Support. The visibility data could be displayed on a live map using ArcGIS (preferably ArcPy).

VIII. Applications

A. Streetlight Control

The main application aimed at the beginning of this project was Autonomous Streetlight Control. That is the main reason why lane detection is used to calculate visibility. Autonomous Streetlight System will be extremely useful in the cities or towns where day time changes frequently, areas where fog is prevalent, cities or towns on the mountain regions.

B. Other Applications

The system could be also used in various fields with some variation. Some areas where the system could be used are as follows:

1. Crisis Prevention on Highways
2. Airport Visibility
3. Speed Control for Railways

Appendix A

Server – test.mosquitto.org
Port – 1883
Topic – Visibility

Appendix B

Python scripts can be found at
https://github.iu.edu/kmhatre/Visibility_Sensing

Acknowledgement

I would like to thank Indiana University, especially the Department of Intelligent Systems Engineering of School of Informatics, Computing and Engineering for providing the resources to work on this project. I would also like to thank Professor Bryce Himebaugh for providing constant support, guidance and encouragement.

References

[1] Kunjal Nanavati, “Smart Autonomous Street Light Control System,” *LJSTE Volume 2.*
[2] Sunil Kumar Vishwakrma, “Analysis of Lane Detection Techniques using OpenCV,” *2015 Annual IEEE India Conference.*