# Computer Vision at Intersections: Explorations in Driver Assistance Systems and Data Reduction for Naturalistic Driving Studies

Mark Philip Philipsen & Morten Bornø Jensen
Autumn, 2014 - Spring, 2015


AALBORG UNIVERSITY
DENMARK

UC San Diego
Jacobs School of Engineering

**AALBORG UNIVERSITY**
MASTER'S THESIS

**School of Information and Communication Technology**
Selma Lagerlöfsvej 300
9220 Aalborg Ø , Denmark
Phone: +45 9940 7228
http://www.aau.dk

**Title:**
Computer Vision at Intersections: Explorations in Driver Assistance Systems and Data Reduction for Naturalistic Driving Studies

**Theme:**
Master's Thesis: Vision, Graphics, and Interactive Systems

**Project Period:**
P9 and P10, fall 2014 and spring 2015 (Long Thesis)

**Project Group:**
VGIS 1086

**Participants:**
Mark Philip Philipsen
Morten Bornø Jensen

**Supervisors:**
Andreas Møgelmose
Mohan M. Trivedi
Thomas B. Moeslund

**Copies:** 5

**Enclosed:** 4 appendices, 1 CD

**Page Numbers:** 144

**Date of Completion:**
June 3, 2015

**Abstract:**

This report constitutes a long master thesis in Vision, Graphics, and Interactive Systems. It details the work done during almost two semesters abroad at UC San Diego. The work has been research oriented, the report is therefore structured in separate parts in contrast to a standard linear product development flow. The scope of the research has primary been traffic light detection, but also the development of a vehicle detector at intersection used for naturalistic driving studies. A comprehensive survey of traffic light recognition (TLR) systems inducing both academic and industrial research has been done. From this it is clear that TLR research lack challenging public datasets and standardized evaluation methodology. Thus, the largest public dataset in the world with around 110,000 annotated traffic lights has been created and made available. Finally a comparative analysis of a learning-based versus heuristic model-based approaches in relation to traffic light detection is done.
The work has resulted in submission of one conference paper to Intelligent Vehicles Symposium 2015, two conferences papers to Intelligent Transportation Systems Conference 2015, and finally a journal paper for Transactions on Intelligent Transportation Systems.

# Preface

This thesis is composed by Mark Philip Philipsen and Morten Bornø Jensen from November 2014 to June 2015. It constitutes a master's thesis in computer engineering with specialization in "Vision, Graphics and Interactive Systems" from Aalborg University, Denmark. The majority of the work has been carried out abroad at the Computer Vision and Robotics Research (CVRR) Laboratory and Laboratory for Intelligent and Safe Automobiles (LISA), at University of California, San Diego (UCSD), USA.
Throughout the report several external sources are used, these will be referred to when used the first time. The reference is seen as a number in square brackets like this: [#]. The number refers to the source in the bibliography, which is attached at the end of this report. In the bibliography the books are described with its author, title, pages, publishers, edition, and year. Online sources, like web pages, will be described with its author, title, URL, and the date it has been read and downloaded. This bibliography is auto generated by BibTex.
The enclosed CD contains a PDF copy of the project and the source code of the developed systems.
Figures and tables are numbered according to which chapter they appear in, i.e. the first figure in Chapter 7 has the number 7.1, the second figure has number 7.2 etc. Appendices are found at the end of the report, and are referred to with A,B,C,...

Aalborg University, June 3, 2015

Mark Philip Philipsen
<markpp@gmail.com>

Morten Bornø Jensen
<mbornoe@gmail.com>

# Acknowledgments

First of all, we wish to thank professor Mohan Trivedi, head of CVRR and LISA at University of California, San Diego (UCSD), for hosting us in his lab, and for his always enthusiastic supervision and great faith in our abilities.
We would like to thank our supervisors at AAU, professor Thomas Moeslund for opening our eyes to the great opportunity for studying at UCSD and setting up the visit through his connections. Andreas Møgelmose for impressively thorough and patient commenting and helpful criticism.
We would also like to thank our colleagues in the CVRR Lab at UCSD: Eshed always being up for an interesting chat and Sujitha Martin for helping out when stuff hit the fan. Ravi Satzoda for discussing and supervising vehicle detection related research, and Ashish Tawari helping us out when we had a question.

We would like to thank our very good friend and roommate Lars for helping us be social and for our discussions about projects, programming, writing, universities, travel, working, and everything else.
Finally, thanks goes out to all the visitors we have had from home while being over here: Sarah, Louise, Morten's brother, mom, and dad.

# Contents

# Chapter 1

# Introduction

The introduction begins with a motivation for this master's thesis. This is followed by an overview of our work and contributions. The work has been research oriented, therefore the report is structured in separate parts rather than in an ordinary linear product development flow.

## 1.1 Motivation

The automobile revolution in the early 20th century led to a massive increase in road transportation and contemporary road networks was incapable of handling the rapidly increasing traffic load. To allow for efficient and safe transportation, traffic control devices (TCD) were developed for guiding, regulating, and warning drivers. TCDs are elements of the infrastructure that communicate to the drivers, examples are: signs, signaling lights, and pavement markings [17]. In 2011 a total of 253,108,389 vehicles were registered in the USA [49], such a high number of vehicles on the road will result in crashes. Intersections are especially prone to crashes, since vehicles from two or more roads need to cross paths. According to [50] approximately 40 percent of the estimated 5,811,000 crashes that occurred in the USA in 2008 were intersection-related crashes. In 733,000 of these, one or more occupants suffered injuries and 7,421 were fatal crashes. Awareness and safety at intersections is therefore very important and improvements can potentially save a substantial number of lives and reduce the $1.16 trillion bill associated with motor-vehicle related accidents [51].

Since the automobile revolution, safety systems have been given gradually more and more attention. Some of the first most notable systems introduced were power steering and anti-blocking systems, which helped the drivers by increasing the maneuverability during both regular driving but also during emergency

procedures. Ultimately companies such as Google, Daimler, and Tesla wish to make the driving experience fully autonomous. In fact Chris Urmson, director of Google's self-drive car, has fully autonomous cars on the road within the next five years as an ambition [3]. Regards of whether autonomous vehicles are five years, or decades away, the problems and challenges that are present right now, such as the many accidents at intersections, must be addressed. Therefore until autonomous vehicles becomes a reality, it is important to address how safety can be improved by developing Driver Assistance Systems (DAS).

Transportation is a major part of peoples lives, their health and well being is directly related to the efficiency, safety, and cleanness of the transportation systems. To improve the current transportation systems, innovation, and development in sensing, communication, and processing is necessary [46, 37]. The future perspectives of intelligent transportation systems are autonomous networks of vehicles and infrastructure, where transportation is ordered on demand from transport service providers. The high initial expenditure, worries, and effort required when owning and driving private vehicles will no longer be necessary [45]. In the meantime and as long as humans are in the loop, DAS are an important part of reducing the number of fatalities and injuries. Some DAS are already implemented in vehicles available to consumers. Examples of these are adaptive cruise control, collision avoidance with active braking, and blind spot detection. Google's self driving cars have safely driven more than 700,000 miles, which raises the public question whether the technology challenges and issues are solved. Though some of the above problems have been solved, there still remain a large set of problems and challenges that remains to be solved. Google's cars rely significantly on data that have been collected multiple times on the exact test route prior to the test. This data is examined by both humans and computers to provide a prior map of the location's key features in the traffic scene which is essential for self driving [47]. This also lead to a main problem that these prior maps must be collected all around the world, and even more challenging, kept updated. The weather has a great impact on the driving performance for humans and adverse weather conditions challenges the car's video cameras. In fact Google is yet to drive in snow and reach reliability during heavy rain. A self driving car must also be able to robustly detect and recognize traffic lights in all kinds of weather and light conditions.

A large set of problems and challenges remains to be solved. Some of these are retaining performance under less then ideal conditions, such as adverse weather conditions and during both day and night. These are some of the computer vision problems that are not yet solved: vehicle detection, traffic sign recognition, pedestrian detection, traffic light recognition, and traffic scene understanding.

Intersections are some of the most demanding challenges drivers must handle. Navigating intersections requires awareness of surrounding objects, selecting lane, awareness of signs and signals, making stop, or proceed decision while maintaining lane position, appropriate speed, and turn rate. Unlike for traffic sign recognition (TLR) and pedestrian detection, no surveys of TLR research exist. Furthermore, most published TLR systems are evaluated based on local datasets with a limited number of traffic lights (TLs) and little variation. This makes comparison between existing methods and new contributions difficult. Inspiration for further improvements can be found by looking at research done on similar computer vision problems. For traffic sign recognition [29, 26] explains how the focus has shifted from heuristic model-based detection to learning-based approaches and the problem is considered solved on a subset of signs. The same is the case with pedestrian detection, where [12] shows how a learning-based detectors based on Integral Channel Features (ICF) or the even faster and slightly better Aggregated Channel Features (ACF) outperform the other approaches. While research on sign and pedestrian detection has mostly moved on, the same is not the case for TL detection where the majority rely on some sort of color and/or shape filter for detection.

Apart from active safety, some research in causes for accidents is done offline by the processing of comprehensive collections of data. These types of studies are designated Naturalistic Driving Studies (NDS) when based on data captured in naturalistic settings. NDS are the study of data from everyday driving, where a wide range of data are collected and analyzed. From the collected data researchers tries to identify parameters that may cause traffic accidents, this is primarily done by manual annotation of the data. Since this very is expensive and time consuming the extent of such studies are currently limited. By putting effort into automatic data reduction in NDS the workload can be substantially lightened and the scope and quality of studies extended. The purpose is to provide insight and identify the patterns and behaviors of drivers leading up to, and during, near-crashes and crashes.

## 1.2 Scope of Project

The scope of this project is solving problems and challenges at intersections. The scope is threefold:

- Develop a system that can do vehicle detection using stereo vision for automatic data reduction for five NDS events mainly related to intersections.

- Provide an overview of research in TLR. Collect and annotate a large diverse dataset for TLR and propose a standardized evaluation approach.
- Apply a state of the art detector to the TL detection problem and compare it to conventional heuristic model-based TL detectors.

## 1.3 Work and Contributions

This section provides an overview and introduction to the work and contributions made in this project.

### 1.3.1 Vehicle Detection and Tracking for NDS at intersections

In chapter 3 we introduce our conference paper on *Stereo-based Event Detection for Naturalistic Driving Studies* which was accepted for the 2015 IEEE Intelligent Vehicles Symposium (IV) in Seoul, South Korea. In it we propose stereo vision based vehicle detection and tracking in intersections for data reduction with NDS in mind. From the tracked vehicles' location in 3D, five NDS events are automatically detected and a NDS rapport is generated.
The main contributions are:

- Using stereo vision for automatic data reduction for NDS on both day and nighttime data, with focus on intersections.
- Introducing a new NDS event: Average distance to vehicles directly in front of the ego vehicle.

The paper along with related information can be found in chapter 3.

### 1.3.2 Traffic light recognition surveys

In chapter 4 we introduce our journal paper *Vision for Looking at Traffic Lights: Issues, Survey, and Perspectives*, which is submitted for IEEE Transactions on Intelligent Transportation Systems (ITS). In the journal paper relevant TLR research back to 2004 is surveyed. The survey is however primarily focused on research made from 2009 and onward. We provide an overview of the methods employed for detection, classification, and tracking as well as the used color spaces and features. In the conference edition we limit the scope to the problem of detecting TLs, which we consider the main challenge in TLR. By looking at the evaluation of current research in TLR we conclude that the introduction

of a common evaluation practice will significantly help advancement in the area. This includes settling on descriptive and meaningful evaluation metrics as well as the introduction of challenging public training and test datasets. In order to get these practices and the datasets public as fast as possible, they are first described in our conference paper *Emerging Trends for Traffic Lights: Detection and Evaluation*, which is submitted for the 2015 IEEE 18th International Conference on Intelligent Transportation Systems (ITSC) in Las Palmas de Gran Canaria, Spain. This serves as a predecessor to the journal paper which provides a more in-depth review of current research and description of evaluation methodology and proposal for future evaluation.

**Vision for Looking at Traffic Lights: Issues, Survey, and Perspectives**

The main contributions of the journal paper are:

1. Provide an overview of current TLR papers' method choices and contributions.

2. Introduce a common evaluation procedure for future TLR systems.

**Emerging Trends for Traffic Lights: Detection and Evaluation**

The main contributions of the conference paper are:

1. Provide an overview of 4 recent TLR papers' approaches to detection .

2. Introduce a common evaluation procedure for TL detectors.

3. Publish an extensive high resolution, annotated, stereo video database, with day and night video sequences.

The papers along with relevant information can be found in chapter 4.

### 1.3.3 Traffic Light Detection for DAS

In chapter 5 we introduce our conference paper on *Traffic Light Detection: A Learning Algorithm and Evaluations on Challenging Dataset*, which is submitted for the 2015 IEEE 18th International Conference on Intelligent Transportation Systems (ITSC) in Las Palmas de Gran Canaria, Spain. The main contributions are:

- Recognizing traffic light in an DAS use case under adverse conditions.

- Be the first to apply the ACF learning-based traffic light detector successfully.
- Using depth from stereo vision to improve performance of traffic light recognition system.

The paper along with relevant information can be found in chapter 5.

### 1.3.4 Database

Common for all four papers is our collected high resolution, stereo video dataset, with both day and night video sequences. The collection, processing, and organization of this large amount of data are described in chapter 6. The main contributions are:

- Collection of a comprehensive stereo vision video database with both day and night time sequences.
- Annotation of ground truth in both training and evaluation video sequences.

## 1.4 Appendices

In addition to the contributions from the papers listed above, much of the work which constitutes the basis for our research is described in the appendices. The most noteworthy work will be presented in this section.

### 1.4.1 Hardware

Throughout the development of this project, various hardware have been used. In appendix B these are listed together with some Triclops API code snippets used together with Point Grey's Bumblebee XB3. The main work in the hardware chapter is:

- CUDA implementation of V-disparity generation.
- Capture and on the fly rectification of RGB stereo image pairs.

### 1.4.2 Software

In appendix C an introduction to the toolboxes used for conducting the research done with this project is presented.

## 1.5 Reader's Guide

This master's thesis revolves around computer vision and machine learning applied to solve problems in the traffic scene. All of the considered problems are seen from the perspective of an ego vehicle, therefore the proposed are meant to function on a moving platform. Our research covers state of the art visual DAS and NDS. With the purpose of DAS in mind we specifically look at TLR at intersections. For NDS automatic event registration we detect and track vehicles using passive stereo vision in both day and night scenes. As part of our research, hours of stereo video material has been collected under varying conditions for training and testing the developed systems. Our research has spawned four papers, one on NDS, one where a state of the art learning based detector is applied to TLD and two surveys on TLD and TLR, respectively. The papers will be presented in self-contained chapters, where they are introduced and motivated for. Relevant methods are described in depth in the theory chapter 2. The methods will be referenced from their respective chapters: 3, 4, and 5. For the TLD and TLR survey papers, a large stereo image database is collected and presented in chapter 6.

# Chapter 2

# Theory

Through the research done as part of the master's thesis a wide range of interesting methods and concepts have been encountered. A selection of the most essential methods and concepts for our thesis are explained in detail in this chapter. The chapter will thus be referenced from the chapters which include work that rely on the methods and concepts explained here.

## 2.1 Color Spaces

Image segmentation is a central part of computer vision, the purpose is dividing an image into meaningful regions [8]. In this section segmentation of color images will be discussed along with a number of relevant color representations. In digital imaging, color is described using color models. The models represent colors using three numbers withing ranges, specified according to the needed precision. There exists many representations of color, each with their own forces. An important part of successful image processing is selecting a suitable color space. Table 2.1 gives an overview of some essential terminology when dealing with light and color. Throughout this section [40] and [36] are used extensively.

### BGR/RGB

BGR is a reordering of the channels of the well known RGB color space. When reading images into a program using OpenCV this is the default color space and channel ordering. RGB is an additive color model where the addition of the red, green, and blue channels is used to represent a wide range or colors. RGB is the color model used in most color input and output devices, such as, displays and digital cameras, where pixels consists of red, green, and blue light emitters or photocells.

**Table 2.1:** Light and color terminology.

| Name | Description |
|---|---|
| Intensity | Linear measure of radiated energy, in a relevant spectrum |
| Luminance | Perceived power of light, described by a non-linear function of the spectral sensitivity of human vision |
| Hue | Perceived color, dependents on the most prominent wavelengths |
| Saturation | Perceived colorfulness, dependents on the concentration around the prominent wavelengths |

**(a)** BGR. **(b)** Channel B. **(c)** Channel G. **(d)** Channel R.

**Figure 2.1:** Traffic scene represented in the BGR(RGB) color space.

## XYZ

The XYZ color space is defined by the International Commission on Illumination (CIE). It consists of X, Y (luminance) and Z. The value of each of the components are positive and proportional to the radiated energy.

## CIE LUV

The LUV color space is defined by the International Commission on Illumination (CIE). L (Luminance), U and V.

**(a)** XYZ. **(b)** Channel X. **(c)** Channel Y. **(d)** Channel Z.

**Figure 2.2:** Traffic scene represented in the XYZ color space.

**(a)** LUV. **(b)** Channel L. **(c)** Channel U. **(d)** Channel V.

**Figure 2.3:** Traffic scene represented in the LUV color space.

**(a)** YCrCb. **(b)** Channel Y. **(c)** Channel Cr. **(d)** Channel Cb.

**Figure 2.4:** Traffic scene represented in the YCrCb color space.

### YCrCb

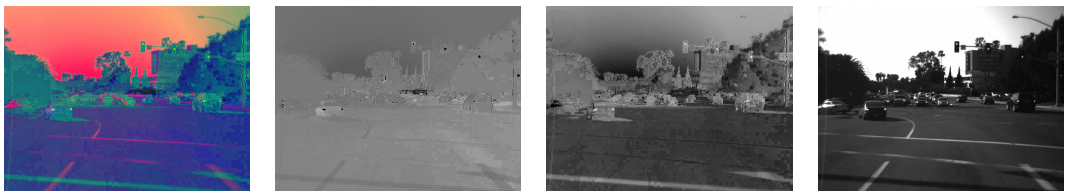
In YCrCb, Y is the luma, Cr is the red difference and Cb is the blue difference.

### HSV

HSV specify hue, saturation and value in a cylindrical color representation.

## 2.2 Stereo Vision

Depth from stereo vision forms the basis for most of the work done in this thesis, hence it is essential to have a thorough understanding of the underlying theory. In this chapter the theory behind the inner workings of a stereo camera will be explained, followed by an explanation of calibration process. Hereafter,



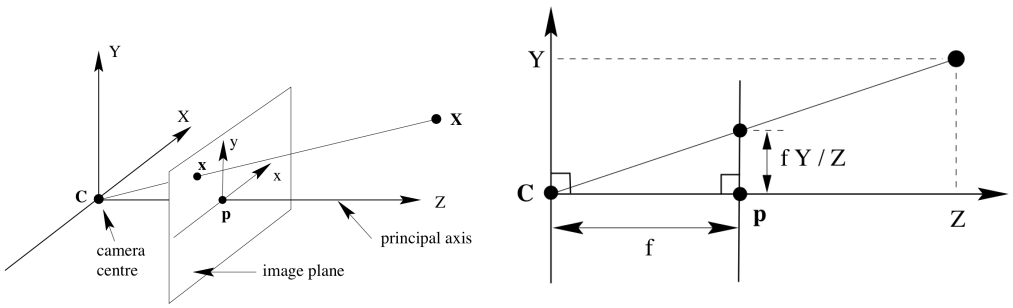
**(a)** HSV. **(b)** Channel H. **(c)** Channel S. **(d)** Channel V.

**Figure 2.5:** Traffic scene represented in the HSV color space.

some of the potential issues that may arise when capturing stereo data are discussed. Finally, a number of open source stereo matching algorithms are analyzed and compared.

### 2.2.1 Camera Model

The pinhole model is used for projecting 3D world coordinates onto a 2D image plane, where the effect of a lens is disregarded and for simplicity's sake the only light allowed to pass through to the image plane is light from a point traveling along a single straight path through a pinhole. The pinhole model is illustrated in Figure 2.9, where Figure 2.6a shows the principal from a angle showing all 3 dimensions, and Figure 2.6b shows only 2 dimensions. The camera center is defined as the origin in a euclidean coordinate system, which is denoted as $C$ in Figure 2.6a, in same figure the image plane is shown which has z-distance from the origin on the focal length, $f$, also described as $Z = f$. This is also indicated in Figure 2.6b where the image plane is denoted as $p$, and the distance from the center to image plane is $f$. A point in world coordinates, i.e $X = (X, Y, Z)^T$, is projected onto the image plane where the line from camera center to the point X meet the image plane. In Figure 2.6a this is seen as the line between, $C$ and $X$, and the point $x$ is the point where the line meets the image plane, also referred to as the principal point. [19]




**(a)** **(b)**

**Figure 2.6:** [19, pp. 154]

By simple trigonometry, the remapping point $X$ from world coordinates to image plane coordinates can be described as seen in equation (2.1).

$$(X, Y, Z)^T \Rightarrow (f\frac{X}{Z}, f\frac{Y}{Z}, f\frac{Z}{Z})^T \tag{2.1}$$

The last image coordinate in the equation on the left can be left ignored, as it can be reduced to $f$, which will remain a constant indicating the image plan

will remain constant on the Z-axis. A final equation description a projection from world to image plane coordinates is there defined in equation (2.2).

$$(X, Y, Z)^T \Rightarrow (f\frac{X}{Z}, f\frac{Y}{Z})^T \tag{2.2}$$

The homogeneous coordinates for mapping between homogeneous vectors of world and image points can be written as the matrix multiplication seen in equation (2.3).

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & f & 0 \\ & & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.3}$$

### 2.2.2 Camera calibration

Throughout this section, cite [19] and [1] are heavily used.

## Intrinsic

The intrinsic parameters are parameters describing the camera internally.

**Principal point offset** Equation (2.2) assumes the origin coordinate in the image plane is the principal point. However, the center might be not be located there due to manufacturer's assembling accuracy. An offset is therefore introduced to equation (2.2), as seen in equation (2.4).

$$(X, Y, Z)^T \Rightarrow (f\frac{X}{Z} + p_x, f\frac{Y}{Z} + p_y)^T \tag{2.4}$$

Which can be rewritten to equation (2.5). This expression is also present in equation (2.3).

$$M = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \tag{2.5}$$

**Non-equal scales in both axial directions** Depending on the camera, there might be some unequal scale factors present in each direction, resulting in the images being distorted. For preventing unequal axis' directions, the pixel dimensions in the x and y direction is taking into consideration as $m_x$ and $m_y$ in the x and y direction, respectively. A new remapping expression can

be derived, seen in equation, by multiplying the pixel dimensions to equation (2.5).

$$M = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \cdot \begin{bmatrix} m_x & & \\ & m_y & \\ & & 1 \end{bmatrix} = \begin{bmatrix} f \cdot m_x & & p_x \cdot m_x \\ & f \cdot m_y & p_y \cdot m_y \\ & & 1 \end{bmatrix} \tag{2.6}$$

Finally, equation (2.6) can be shortened to equation (2.7).

$$M = \begin{bmatrix} \alpha_x & & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix} \tag{2.7}$$

**Skew** Though most cameras have almost perfectly rectangular pixels, their might be case where the pixels are skewed. An skew, $s$, factor is therefore introduced for taking the skew from non-rectangular pixels into account. A final intrinsic paramter matrix, or camera matrix $m$, is seen in equation (2.8).

$$M = \begin{bmatrix} \alpha_x & s & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix} \tag{2.8}$$

## Extrinsic

The extrinsic parameters are parameters describing the relation between the camera coordinates and the world coordinates. These are the rotation matrix, $R$, which describes the rotation between the world coordinates and the camera coordinates. The vector $t$ which describes the translation between the origin of the world coordinate system to the origin of the camera coordinates. The transformation to world coordinates is needed to compare the camera coordinates with camera coordinates from a different camera. A combination of both intrinsic and extrinsic parameters gives the project P where P = M[R|t], which transforms world coordinates to homogeneous image pixel.

## Calibration

When using multiple cameras, and the goal is to find points in a different camera coordinate, space results in equation (2.3) no longer being applicable as this is used for world point in the same camera coordinates. However, if points from a different camera, such as in the case of a stereo camera rig, one point must be used in the other cameras coordinate space. The camera's coordinate space from camera 2 must therefore be scaled and translated to be represented in the

coordinate space of camera 1. The rotation and translation is also defined as the *extrinsic parameters*, which maps world space to camera space as previous mentioned. This is seen in the 3x4 matrix, denoted in equation (2.9). [19, Equation (6.3)]

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & f & 0 \\ & & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.9}$$

Finally the intrinsic parameters is included, and a final mapping expression can therefore be denoted as seen in equation 2.10. [19, Equation (6.4)]. The

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} \alpha_x & s & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix} \cdot \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.10}$$

Equation 2.10 is sometimes reffered to as the projection matrix, which can be denoted as seen in equation 2.11

$$P = M[R|t] \tag{2.11}$$

An illustration of all the steps involved is seen in Figure 2.7, where the entire process is called projection matrix and is similarly denoted as seen in equation (2.11).

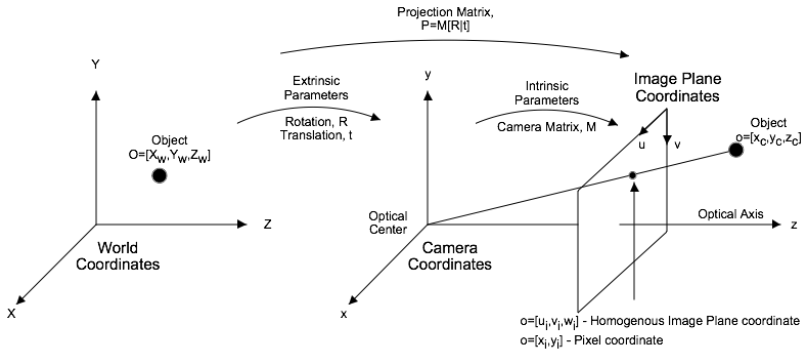



**Figure 2.7**

## Rectification

The purpose of the rectification in two-view geometry is to simplify the corresponding point problem when trying to find two points in two stereo images.

By rectifying a set of images has alligned epipolar lines such they are horizontal lines in the set of image.
In Figure 2.8 epipoles are denoted as $e$ and $e'$ which is placed on the line between two origins where it meets the left and right image plane, respectively. The epipolar line, $l$, is the line between the epipole and the principal point, $x$. In Figure 2.8 the corresponding epipolar line in the right image plane is marked with a purple line. The corresponding coordinate in the right image plane must lie on the purple epipolar line. For easier searching for the corresponding

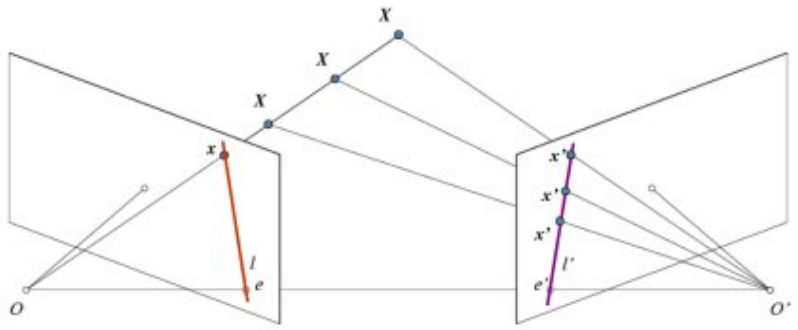



**Figure 2.8:** Illustration of epipolar line.

pixel in two image planes, it is desirable to rectify the image planes as seen in Figure 2.9a, such that the corresponding points problem becomes an 1D search problem on the horizontal epipolar lines. This is illustrated in Figure 2.9a as the golden image plane, which have aligned epipolar lines from the two gray image planes. An example of this is seen in Figure 2.9b.

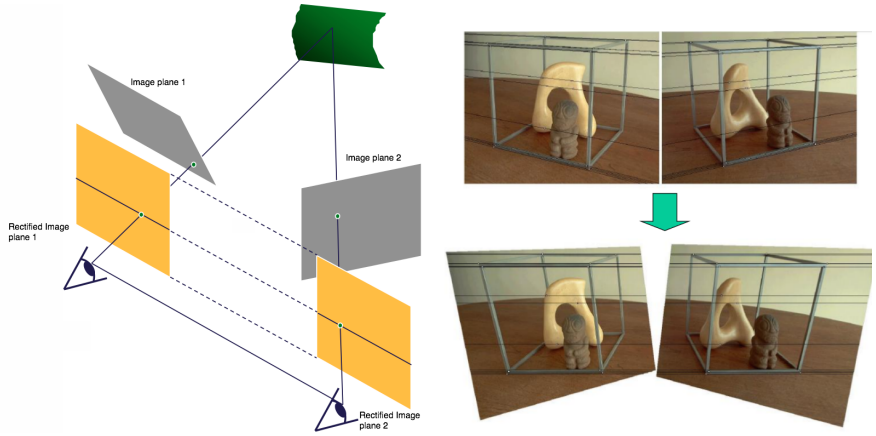
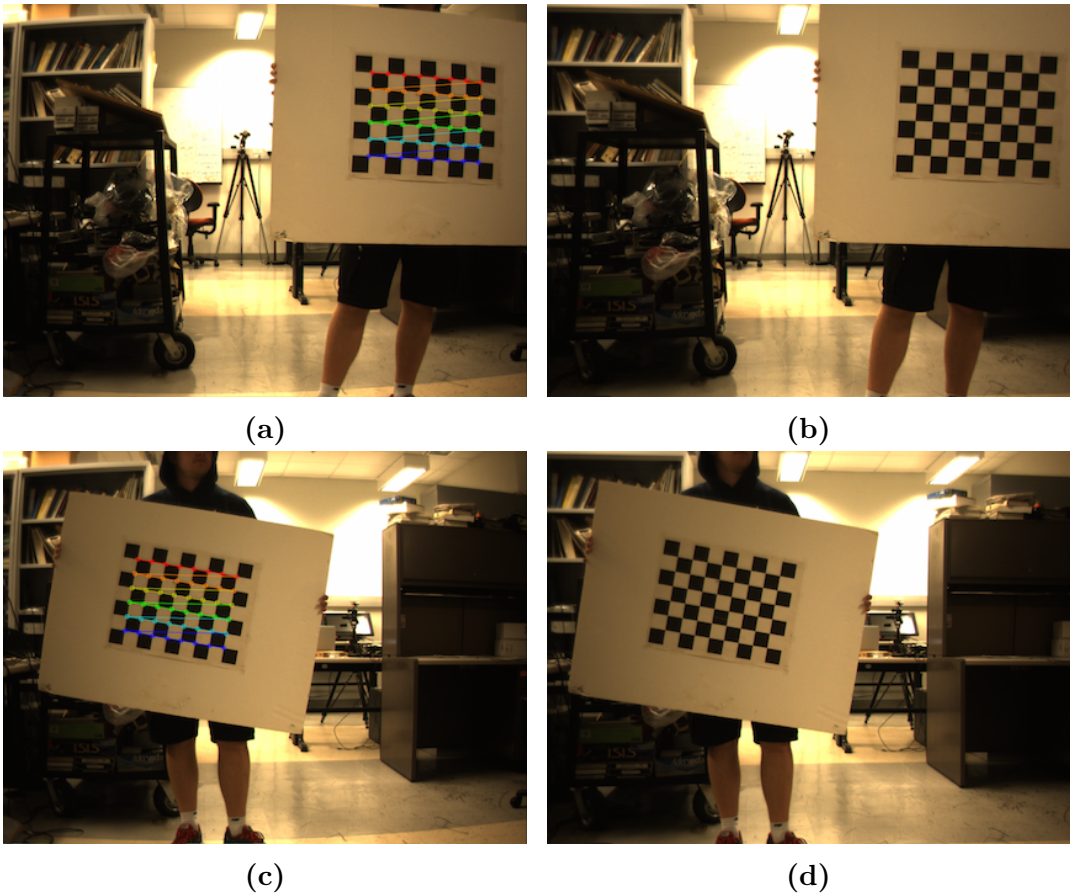



**Figure 2.9:** (a) Illustration of rectification. (b) Example of rectification. [2]

### 2.2.3 Automatic camera calibration

For finding the intrinsic and extrinsic paramteres needed for rectification, various toolboxes are available for doing so. Generally a camera calibration is done by using a planar checkerboard with known dimensions, boxes size, and number of boxes in the given checkerboard. An example is seen in Figure , where there are For calculating a accurate camera calibration it is very important that these definitions are accurate. Furthermore, experiences show that it is advantageously to first calibrate the individual camera one at the time, to make sure that the entire image plane is being calibrated. A series of calibration image is seen in Figure 2.10. Where the Figure 2.10a and Figure 2.10c has line drawn upon the checkerboard to indicate where corners have been found. By finding the corners, which relation is known due to the prior definitions of the checkerboard, the intrinsic and extrinsic parameters are found.



**(a)** **(b)**

**(c)** **(d)**

**Figure 2.10:** (a) Image 0 (b) Image 0 rectified (c) Image 1 (d) Image 1 rectified.

**OpenCV** OpenCV provides a function for automatic corner detection, as well as a function that can do the calibration based on the found corner points. This makes the calibration process really simple with OpenCV. [34]

**Camera Calibration Toolbox for Matlab** This toolbox is not as automated as the approach described for OpenCV, as the user must manually select the area of the checkerboard in each image, hereafter the corners are extracted. This might be considered a little comprehensive if including a large set of calibration images. [5]

**Triclops API** Triclops does not provide a toolbox for automatic camera calibration, instead it provides access to the Bumblebee camera's factory calibration located in the cameras own memory. Additionally, it provides functions for rectifying, disparity calculation and projection to 3D, using the very precises factory calibration [42]. Throughout this project the Triclops API has been used.

### 2.2.4 Stereo Matching Algorithms

Two types of stereo vision exist, sparse and dense stereo. Sparse stereo is created based on matches between a few points in the stereo pair whereas dense stereo is calculated from matches between every pixel in the stereo pair. In this rapport we will primarily concern with dense stereo vision. Finding stereo correspondence is the basis for determining depth from two or more images, it essentially consist of matching locations in these images and assigning a label, the disparity, based on the offset between the matches. Matching is done based on one of the following image location descriptors:

**Pixel descriptor** A pixel descriptor use the intensity or color of single pixels when matching pixel pairs along the epipolar lines. Relying on information from a single pixel makes the methods that uses a pixel descriptor fast but also sensitive to noise.

**Block descriptor** A block descriptor use the intensity or color of multiple pixels located within a window around a center pixel. Blocks in the reference image are then each matched with blocks along the epipolar lines in the search image. Relying on information a block of pixels makes the methods that uses a block descriptor slow but less sensitive to noise, with the drawback of less details and staircase artifacts.

**Feature descriptor** A feature descriptor is found in information rich areas where transitions in intensity or color are found. Methods that use feature descriptors for matching are usually fast because only a fraction of matches needs to be done. However, this only holds if the feature extraction is also fast. A drawback of feature based correspondence methods is a rather spares depth map, especially in monotone areas of the images.

Scharstein and Szeliski makes a thorough analysis of existing dense stereo correspondence algorithms in [43]. As part of this analysis they establish a taxonomy and standard structure that most stereo correspondence algorithms follows.

**Matching cost computation** Matching cost is a measure of the similarity between a location in the reference image and each of the candidate locations in the search image.

**Cost aggregation** Local methods take into consideration the matching cost of nearby pixels. Where global methods often skips this step and instead rely on minimizing the global energy measure in the next step.

**Disparity computation/optimization** For local methods this step simply consist of choosing the disparity with the lowest cost between a pixel in the reference image and a pixel in the search image. Since a matches are found using a winner takes all strategy is used, the same pixel in the search image may be assigned as the match for several pixels in the reference image. This will be a problem in regions with little variation and in case of occlusion, where the object is only visible to one of the cameras.
Global methods can e.g look for the optimal matches for all descriptors along the epipolar line as well as apply a smoothness constrain, which rely on the assumption that disparity does not vary much on surfaces, this is effective at limiting outliers in the disparity estimates in textureless regions.

**Disparity refinement** Since most stereo correspondence algorithms returns discrete disparity estimates, the resulting disparity map will appear layered. This can be mitigated by using more discrete disparity levels. Alternatively, refinements can be done in postprocessing by e.g. curve fitting. Median filters can smooth spurious disparity estimates and holes from e.g. occlusion can be filled by propagating neighboring disparity estimates.

The stereo correspondence algorithms that are studied in this thesis are selected based on the availability of their implementation and their performance. Each

algorithm will be described using Scharstein and Szeliski's taxonomy.

### 2.2.5 Stereo correspondence algorithms

Common for most stereo correspondence algorithms is that they search for matches along the corresponding image rows in the rectified image pairs. This search is usually limited to an interval defined to be between the minimum number of disparities that are expected, translating to the furthest distance that disparities should be calculated for, and the maximum disparity, translating to the closest distance where disparities should be calculated. This defines the horopter, which is the feasible volume of the disparity map. Other than the disparity range, the horopter's size is set by the some of the stereo camera's physical properties, baseline, focal length and pixel size.

#### OpenCV Block Matching (BM) for CPU and GPU

BM is one of the most basic matching algorithm, this results in high speed execution but poor results in difficult scenarios. Before computing matching costs, the image pair is prefiltered with a sobel filter in order to eliminate differences in brightness and enhance texture[6].

**Matching cost computation** A number of small blocks consisting of pixels within a fixed window $W$ in the right image $I_R$ are each compared with the corresponding reference block in the left image $I_L$ by using the sum of absolute differences(SAD). The blocks in the right image all have different offsets $u$ compared to the reference block. The SAD function can be seen in (2.12).

$$\sum_{(i,j)\in W} |I_L(i,j) - I_R(i+u,j)| \tag{2.12}$$

Matches are found along the epipolar lines, which after rectification corresponds to image rows. The best match is the match with the smallest SAD. the offset $u$ for this match is set to be the disparity at that match's coordinate $(i,j)$. To save CPU cycles, the algorithm does not calculate SAD from scratch every time, instead it can often rely on the fact that each reference block is offset by one column of pixels compared to the previous one and simply add the new column of pixel values while also subtracting the now irrelevant column belonging to previous block.

**Cost aggregation** Pixel intensity is aggregated within each block.

**Disparity computation/optimization** The reference block resulting in the smallest SAD is selected as the match in a winner takes all strategy. Matching is only done in regions with strong texturing, low texture regions are left black. This is determined by a texture threshold placed on the SAD window.

**Disparity refinement** False matches are filtered out if (2.13) is not satisfied.

$$uniquenessRatio > \frac{matchvalue - minvalue}{minvalue} \tag{2.13}$$

Because BM is using block matching, matches in boundary regions where the blocks contain both for- and background will alternate between high and low disparity, resulting in so called speckles. These are filtered by only retaining a match if the minimum and maximum disparity within a speckle window lies within a small range.

OpenCV's CPU and GPU implementations of K. Konolige's BM stereo correspondence algorithm [24] takes the parameters seen in table 2.2.

**Table 2.2:** OpenCV BM parameters.

| Parameter | Description | Used setting |
|---|---|---|
| NumDisparities | Search range in pixels corresponding to maximum disparity - minimum disparity | 64 |
| MinDisparity | All parameters can be set automatically by using one of three available presets | 96 |
| PreFilterCap | Truncation value for prefiltered image pixels | 31 |
| TextureThreshold | Determine if region is textured enough for matching | 3 |
| UniquenessRatio | Accept the computed disparity d* only if SAD(d) >= SAD(d*)*(1 + uniquenessRatio/100.) for any d != d*+/-1 within the search range | 15 |
| SpeckleWindowSize | Size of speckle window, which is used to eliminate noisy transitions | 60 |
| SpeckleRange | Tolerance to variation within speckle window | 24 |
| Disp12MaxDiff | Max difference in the left-right consistency check | 4 |
| BlockSize | Size of block used for matching. Must be odd, since the block is centered on the current pixel | 9 |

The results of running OpenCV's CPU implementation of block matching on two different sets of images can be seen in Figure 2.11 and 2.12.

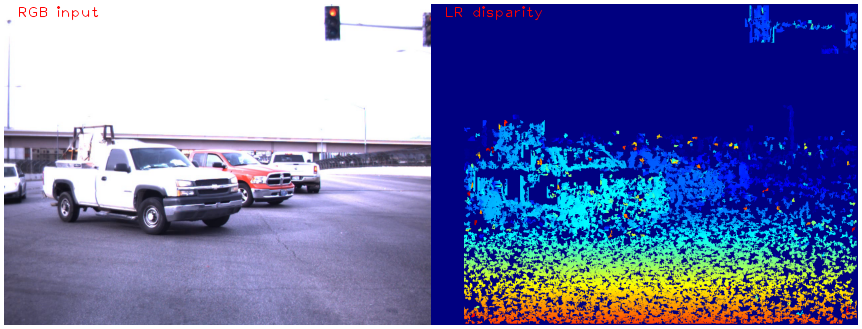



**Figure 2.11:** Disparity from day scene using OpenCV's CPU BM implementation.

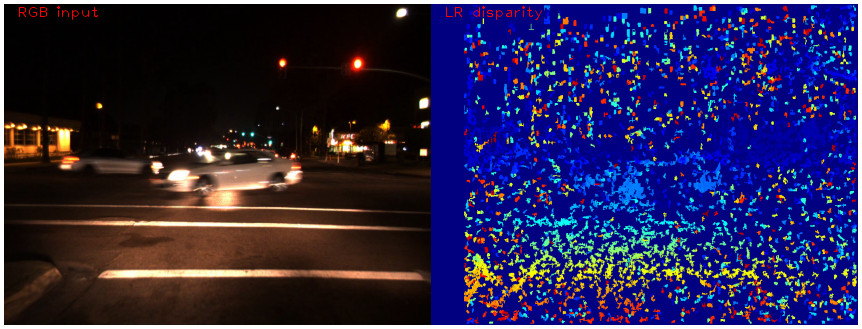



**Figure 2.12:** Disparity from night scene using OpenCV's CPU BM implementation.

The results of running OpenCV's GPU implementation of block matching on two different sets of images can be seen in Figure 2.13 and 2.14.


**Figure 2.13:** Disparity from day scene using OpenCV's GPU BM implementation.

From Figure 2.11, 2.12, 2.13 and 2.14 it is clear that the GPU implementation produce a significantly better disparity map.

Table 2.3 shows the execution time of the two BM implementations on two different sets of 1280x960 images.


**Figure 2.14:** Disparity from night scene using OpenCV's GPU BM implementation.

**Table 2.3:** OpenCV BM timing on two different sets of 1280x960 images.

| Processing unit | Scene | Computation time |
|---|---|---|
| CPU | Day scene | 32.3 ms |
| CPU | Night scene | 46.4 ms |
| GPU | Day scene | 63.6 ms |
| GPU | Night scene | 64.9 ms |

### OpenCV Semi-Global Block Matching (SGBM) for CPU

SGBM is probably the most successful matching algorithm because of it's speed and performance. It uses point wise matching cost and a smoothness term.

**Matching cost computation** Pixel blocks correspondence cost is calculated for all disparities in the search space.

**Cost aggregation** The smoothed path cost is calculated in a number of directions for each disparity.

**Disparity computation/optimization** The disparity resulting in the smallest path cost is selected for each pixel.

**Disparity refinement** A speckle filter and a left-right consistency check can be applied.

OpenCV has a modified CPU implementation of H. Hirschmuller's SGBM stereo correspondence algorithm [20, 21], it takes the following parameters:
The results of running OpenCV's CPU implementation of semi-global block matching can be seen in Figure 2.15 and 2.16.
Table 2.5 shows the execution time of the SGBM implementations on two

**Table 2.4:** OpenCV SGBM(CPU).

| Parameter | Description | Used setting |
|---|---|---|
| NumDisparities | Search range in pixels corresponding to maximum disparity - minimum disparity | 64 |
| MinDisparity | All parameters can be set automatically by using one of three available presets | 96 |
| PreFilterCap | Truncation value for prefiltered image pixels | 31 |
| TextureThreshold | Determine if region is textured enough for matching | 3 |
| UniquenessRatio | Accept the computed disparity d* only if SAD(d) >= SAD(d*)*(1 + uniquenessRatio/100.) for any d != d*+/-1 within the search range | 15 |
| SpeckleWindowSize | Size of speckle window, which is used to eliminate noisy transitions | 60 |
| SpeckleRange | Tolerance to variation within speckle window | 24 |
| Disp12MaxDiff | Max difference in the left-right consistency check | 4 |
| BlockSize | Size of block used for matching. Must be odd, since the block is centered on the current pixel | 9 |
| Mode | Determine of the algorithm considers 5 or 8 directions use MODE_HH for 8 | MODE_SGBM |
| P1 | Controls disparity smoothness, large values gives a smoother disparity, P2 > P1 is required | $8 * 3 * BlockSize * BlockSize$ |
| P2 | Controls disparity smoothness, large values gives a smoother disparity, P2 > P1 is required | $32 * 3 * BlockSize * BlockSize$ |

different sets of 1280x960 images.

### 2.2.6 Post-processing of disparity map

Disparity maps are usually filled with noise and holed regions, caused by bad matches and occlusion. Some of these errors can be cleaned up with processing. The BM and SGBM implementations above employ speckle filters and left-right

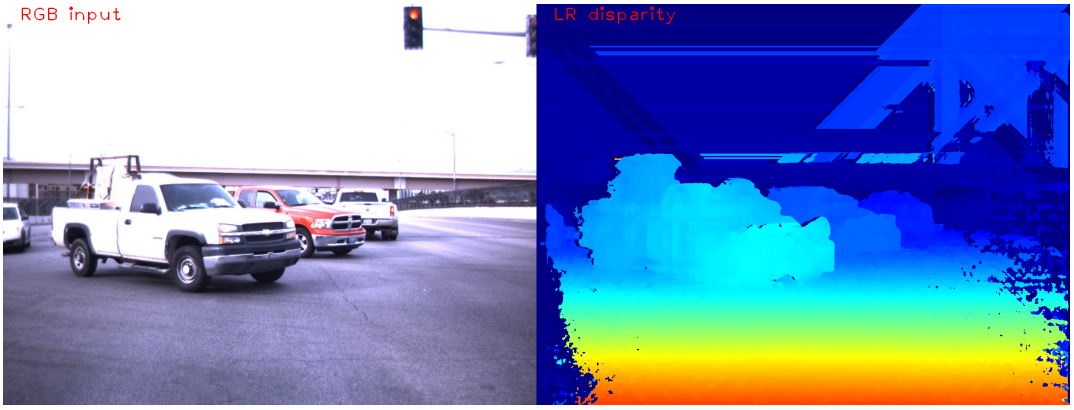



**Figure 2.15:** Disparity from day scene using OpenCV's CPU SGBM implementation.

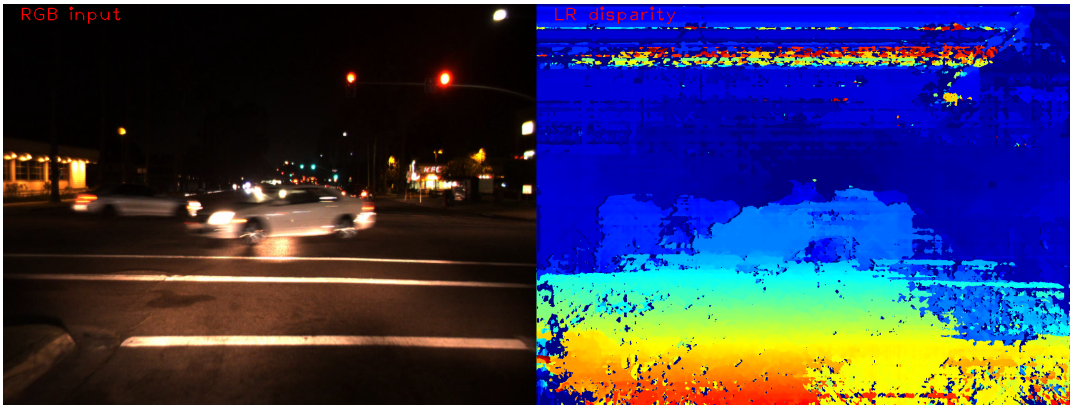



**Figure 2.16:** Disparity from night scene using OpenCV's CPU SGBM implementation.

**Table 2.5:** OpenCV SGBM timing on two different sets of 1280x960 images.

| Processing unit | Scene | Computation time |
|---|---|---|
| CPU | Day scene | 483.677 ms |
| CPU | Night scene | 477.119 ms |

consistency checks for this.

### Pixel consistency check

By looking at the value of a pixel in the previous frame and comparing it with the value in the current frame, a significant amount of noise can be removed. This check is based on the assumption that it is uncommon for correctly matched disparity pixels to erratically change value from frame to frame. In

case of movement, this check will result in a dark edge around the moving objects in the disparity map.

#### LRRL-consistency check

By searching for a match in both direction and determining if the match is consistent in both directions, the issues with the winner takes all strategy can be somewhat mitigated, but at the significant cost of double the amount of searches.

### Stereo image pyramid

A dense disparity map is estimated for each level in the stereo image pyramid independently from each other. Then the reliable disparity estimates selected from each pyramid level are combined into one disparity image. The selection is based on a disparity validity measure that reflects the disparity estimation quality. The validity measure is calculated based on the disparity deviations D in a rectangular shaped window [16].

### 2.2.7 Segmentation methods of disparity maps

For the calculated disparity map to be useful it must be segmented. The following methods can be used for extrapolating meaning from disparity maps.

#### K-means

K-means is a fast and simple unsupervised clustering algorithm, that finds a location for the K center points, which if the correct K was assigned, usually end up in the ideal centers of the distribution of points. However, it requires initialization in the form of a number K center points.

#### Mean-shift

Mean-shift, locates the maxima of a density function. It can be used to detect the modes of a density, which means it can cluster the feature vectors very effectively. I can be prone to return outliers.

#### Spectral clustering

Spectral clustering with Nyströms method is based on pairwise comparison between all possible couples of points. It performs better than the other methods, but is very expensive in terms of resources. The measurement between

all points requires the construction of a graph with nodes for each point and edges between all couples. With Nyströms method the graph is approximated based on the integral eigenvalue problem, this allows the method to approach the speed of the other methods.

### UV Map

An UV map can be generated from a regular map, where of the V-disparity map examines the vertical coordinates in an (u,v) input image coordinate image and is constructed using an disparity map, e.g. from the SGM algorithm. V-disparity was proposed in [25] for aiding obstacle detection from stereo vision by generating a good representation of the road surface, regardless of it's gradient. V-disparity is calculated by accumulating the occurrences of pixel value in images rows in a (u,v) disparity map. The results is essentially a 255 binned histogram for each row. These histograms are represented in a matrix measuring 255 times number of rows. In this matrix distinct planes from the disparity map will be visible as clear lines. Many other objects will stand out as small vertical lines from the dominant diagonal line originating from the disparity pixels belonging to the road surface.

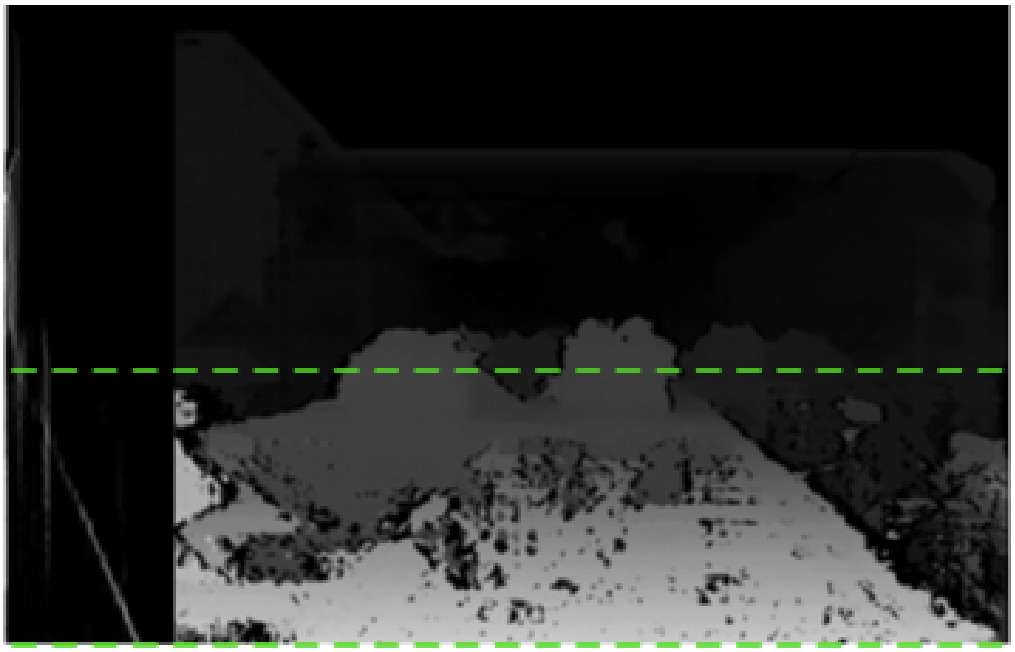


**Figure 2.17:** Correspondence between V-disparity and input disparity map.

The U-disparity map is generated using the horizontal coordinates. Histograms

are calculated for each row in the disparity. Significant surfaces in the disparity map, e.g. roads surfaces, will then show up as lines in the v-disparity. By locating this exact line, segmentation of objects becomes easier. The line can be found using Hough Line, which is a voting scheme for finding lines in a distribution of points or with RANSAC which is an iterative method for estimating parameters based on a number of samples

## 2.3 Learning-based Traffic Light Detection

For both pedestrian[14] and traffic sign recognition [29] integral channel features(ICF) and aggregated channel features(ACF) have provided great results. For our learning based detector we utilize an ACF, but as ACF is based on ICF, ICF is examined in order to give a better understand for ACF. In this section we describe both features. The descriptions are based on [12, 14, 13, 15].

### 2.3.1 Integral Channel Features

ICF is using the integral image for speeding up computations. An integral image is seen in Figure 2.18, where the concept is illustrated. The integral image functions as a look-up table where each entry corresponds to the summation of all pixels above and to the left of the same index in the original image. The purpose is to reduce the number of operations needed for calculating the area of a rectangle.

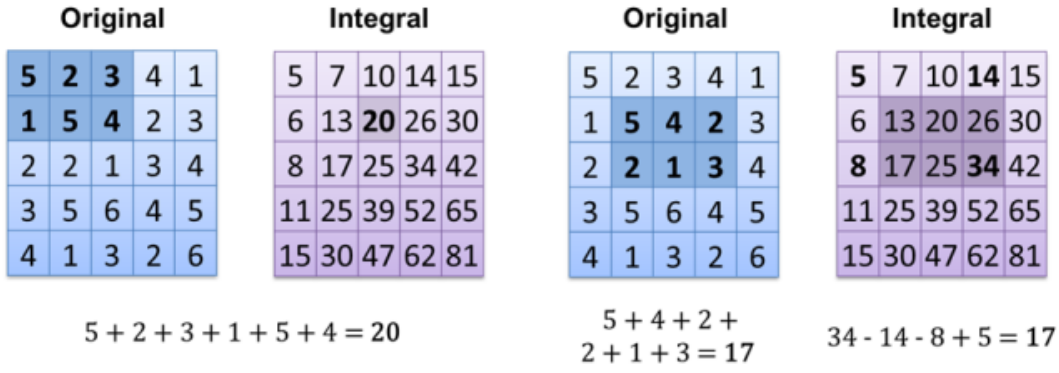



**Figure 2.18:** Example of integral image. [10]

Before using the integral image to extract features, we first need to look into the channels from where they are going to be extracted. A channel is basically a transformation of the original image. Examples are shown in Figure 2.19,

where an input image is converted into multiple channels including different color spaces, edges, and thresholds.

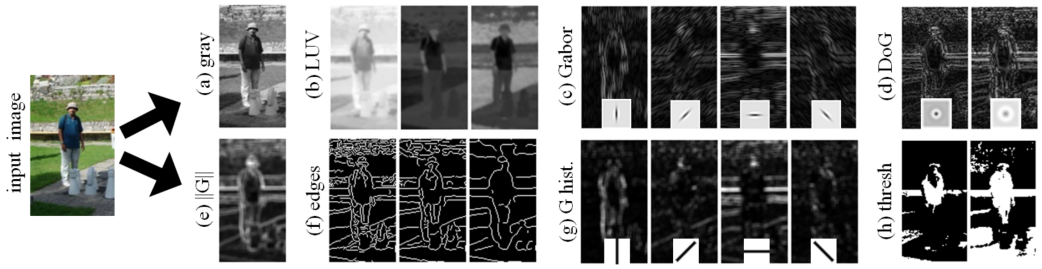



**Figure 2.19:** Examples of possible channels. [15]

The features are then extracted from each of these channels using sums over local rectangular areas. This idea is very similar to the Haar-like features introduced in [52]. In Figure 2.20 (a) a first-order integral channel feature is seen, which is the sum inside a given rectangular area. A second-order Haar-like feature is seen in Figure 2.20 (b), where two first-order areas are combined, e.g. the difference between two first-order features. A higher order of Haar-like feature is seen in 2.20 (c). Finally, 2.20 (d) shows the generation of a histogram based on multiple channels, e.g. histogram of oriented gradients.

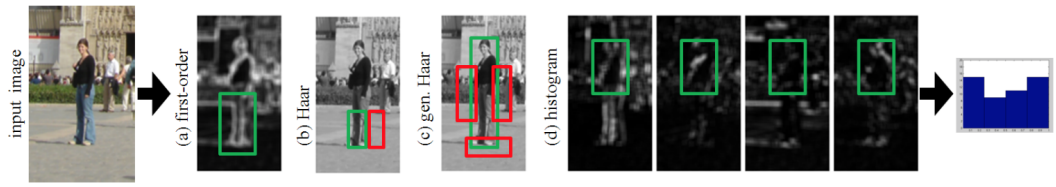



**Figure 2.20:** ICF channels features. [15]

It is clear that there are a large set of possible channels, in practice for ICF a large pool of randomly selected first-order candidate features are created based on the e.g. the following channels: grayscale, LUV, gradient histograms, and gradient magnitude. Higher order features can optionally be calculated and used with the first-order features.

### 2.3.2 Aggregated Channel Features

As mentioned in the beginning of this section ACF is based ICF, and it is therefore conceptually the same in most aspects. ACF is presented to work on the following 10 channels: normalized gradient magnitude, histogram of oriented

gradients (6 channels), and LUV color channels. The key difference between ACF and ICF is the way they construct the features. Instead of computing the relatively heavy integral image, ACF simply smooth and downsample the channels and features can be found with single pixel look-ups in the "aggregated" channels. Which also result in ACF being faster than ICF, as the ICF must construct the integral images and compute the sum of rectangular areas which are more expensive than single pixel lookups. The flow of ACF detector training is seen in Figure 2.21.

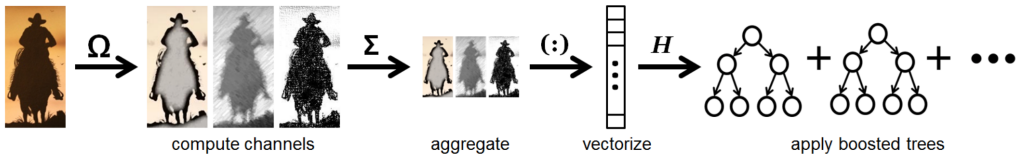



**Figure 2.21:** Flow of ACF detector training. [12]

Traditionally a feature pyramid is constructed by scaling the entire image and then computing all channels for every scale. This is illustrated in Figure 2.22, where scales are sampled with 4 scales per octave. This is a very costly, especially considering scenarios where the number of scales reach more than 10 per octave.

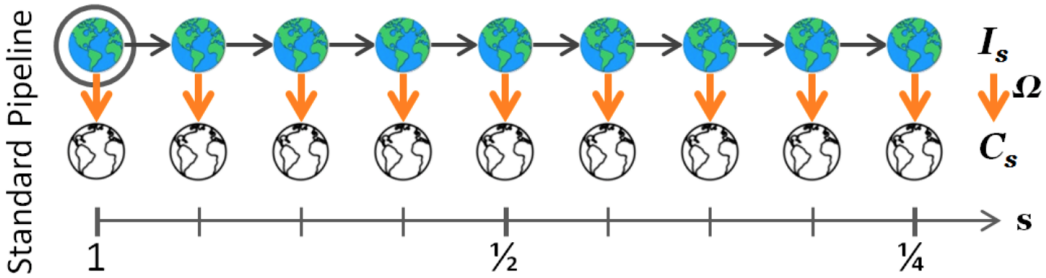



**Figure 2.22:** Traditionally pipeline for a feature pyramid. [12]

The feature pyramid proposed for ACF in [12], is created slightly different. Instead of downsampling the image and then recalculating the channels at every single scale, the fast feature pyramid only downsamples and recalculate the channels once per octave $(1, 1/2, 1/4)$. This approach is seen in Figure 2.23 and can also be used with ICF with improved FPS performance. The remaining scales between two calculated octaves, are computed by approximating the features calculated at the octaves.

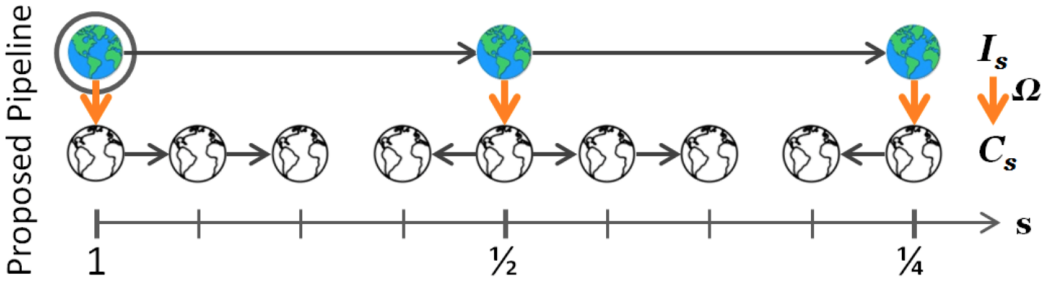
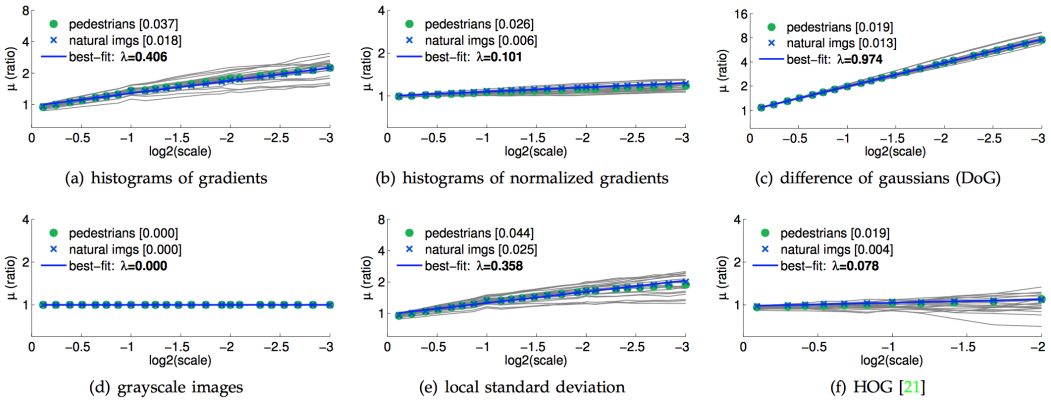



**Figure 2.23:** Proposed pipeline for fast feature pyramids. [12]

The computation of the remaining scales is done using the power law for scaling. For estimating the scaling factor, $\lambda$, one must perform a series of experiments as $\lambda$ is different for each channel, $\Omega$. This is done using Equation (2.14), where N is the number of object/natural image patches. The idea is to plot $\mu_s$ versus scale, $s$, in log-log plot, which result in a straight line which corresponds to the scaling factor, $\lambda$.

$$\mu_s = \frac{1}{N}\sum_{i=1}^{N}\frac{f_\Omega(I_s)}{f_\Omega(I)} \tag{2.14}$$

In [12], this is done for pedestrian detection. Figure **??** shows the result of this for six different channels across 24 scales. By examining the figures it is clear that all of the channels comply with the power law, as a straight slope on a log-log plot is strong indication hereof.




(a) histograms of gradients (b) histograms of normalized gradients (c) difference of gaussians (DoG)

(d) grayscale images (e) local standard deviation (f) HOG [21]

**Figure 2.24:** Estimating the power law feature scaling. [12]

The procedure for training and using the ACF cascade in practice based on Piotr's Computer Vision Matlab Toolbox can be found in appendix C.1.6.

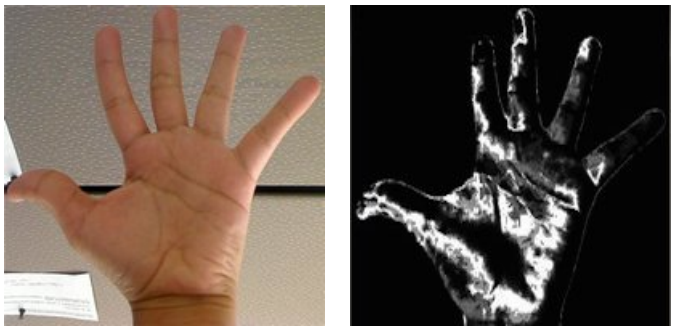
For both ICF and ACF, AdaBoost is usually used for selecting the learners that are best at distinguishing the object [31]. For applying the detector to an image, a sliding window of a predefined size is used across multiple scales.

## 2.4 Heuristic Model-based Traffic Light Detection

In the paper we present two heuristic model-based approaches for solving the traffic light detection problem. In addition to back projection and spotlight, we also looked into using a Gaussian mixture model. In this section, all three methods will be described.

### 2.4.1 Back Projection

Back projection is usually used for image segmentation of finding objects of interest in an image. The main idea is to create a histogram of an image containing the object of interest. This histogram is then used to back-project over a different input image. The output of this is a new image with same dimension as the input image. The output image will however be grayscale, where more white pixels indicates a higher probability of the object of interest being at this exact pixel. The idea is illustrated with a skin segmentation of a hand in Figure 2.25, and in Figure 2.25b, the probabilities of a pixel belonging to the skin is clearly illustrated by varying intensity of white. Similarly, a histogram of specific colors can be used to segment specific objects from the remainder of the image.



**(a)** **(b)**

**Figure 2.25:** (a) Test Image (c) Back projected test image.[33]

### 2.4.2 Spotlight

TL detection by spotlight detection rely on the top-hat transform proposed by [27]. It is used to isolate areas that are either brighter(Top Hat) or dimmer(Black Hat) than surrounding pixels. The operations works by finding the difference between an image and the same image subjected to either the morphological open or close operation shown in equation (2.15) and (2.16) [6].

$$topHat(img) = img - Open(img) \tag{2.15}$$

The open operation enlarges small holes and dips, so by subtracting Open(img) from img, all peaks are revealed.

$$blackHat(img) = Close(img) - img \tag{2.16}$$

The close operation enlarges small peaks and raises, so by subtracting img from Close(img), all valleys are revealed.

The Top Hat operation is used for spotlight detection, as we are interested in isolating bright spots. The size of bright spots found by Top Hat is determined by the structural element size and the number of iterations with which the open operation is applied. The result of using two different parameter settings can be seen in Figure 2.26.
The output of the Top Hat operation usually provide a lot of candidates. These must be filtered using BLOB analysis and clever tricks. This could e.g. be by weighting candidates based on their position and by filtering based on BLOB metrics such as:

- Ratio between width and height of bounding box
- Ratio between the convex area of BLOB and area of bounding box
- Ratio between area of floodfilled BLOB and area of bounding box

### 2.4.3 Gaussian Mixture Model

The idea of using a Gaussian Mixture Model(GMM) is to create a GMM based upon a set of training images with variation of given object. A normal distribution with a mean and variance is found by doing so. If we have multiple objects, multiple mixture components are created. We do in other words train a supervised GMM classifier. The GMM classifier is used for examining each pixel in the test image. The classifier will determine the probabilities of the pixel belonging to each component. If this probability complies with a defined

**(a)** Input image.

**(b)** Top-Hat with structuring element size set to 5x5 and using 3 iterations.

**(c)** Top-Hat with structuring element size set to 5x5 and using 6 iterations.

**Figure 2.26:** Examples of Top Hat operation applied with two different sets of parameters.

threshold, the pixel is labeled to belong to a given component. For doing this, OpenCV's implementation is used. Interestingly their GMM implementation

is called expectation-maximization(EM) algorithm, which, by theory, is an algorithm to obtain maximum likelihood on unsupervised data and thereby able to estimate GMM. [44, 4, 9]. An example of a GMM with three components is seen in Figure 2.27.

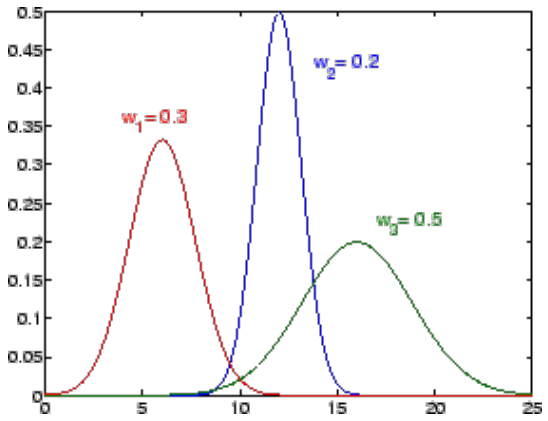



**Figure 2.27:** Example of GMM with three components.

## 2.5 Tracking

All of the tracking used in each stage will be described sequentially throughout this section.Tracking is a central issue in computer vision[28, Ch. 9]. The task of following an object over time is needed in many applications. Following an object over time in a video-feed can be seen as a number of points in a coordinate system. When connecting these points, a curve is created over time. This curve is denoted as a trajectory. A trajectory may not only be based on a position of an object, but can also contain such entries as velocity, acceleration, size, shape etc. This can be denoted as a state vector. The formal definition of tracking is then to find the trajectory of the object's state. This can be formulated as labeling a new object to the existing trajectories. A known framework for tracking is the predict-match-update framework. The idea is to first detect an object in one image and then predict where it approximately will be in the next image. To estimate where the next position will be, two typical types of info can be used. This is either motion or appearance. Motion is information about object movements. If the state is known at time, $t$, we wish to predict the state of time, $t + 1$. For this a motion model, also called a predictor, is introduced. A predictor explains how an object is moving. The simplest predictor is a 0th order predictor, which uses the object's position at time $t$. The 0th order predictor can predict a region in the next image where

the object must be within. By adding velocity and acceleration, it is possible to create a 1st and 2nd order predictor. Examples of a 0th and 1st order predictor are seen in Equation (2.19). By using the motion model, a ROI is found in the next image. If tracking multiple objects the estimated ROI can be used to match objects, with previous objects.

$$p_{t+1} = v_{max} + p_t \quad \text{0th Order predictor} \tag{2.17}$$

$$p_{t+1} = v_t \Delta_t + p_t \quad \text{1st Order predictor} \tag{2.18}$$

$$p_{t+1} = \frac{1}{2} a_t \Delta_t^2 + v_t \Delta_t + p(t) \quad \text{2nd Order predictor} \tag{2.19}$$

Appearance is the second type of information which can be used for tracking. This includes using features to track objects. This can be used in the match step, to find the best match for a detected object. If an object from time $t+1$ is matched with an object from time $t$, the information of the object is then updated. This finalizes the predict-match-update framework. Tracking has uses in many parts of our research but saw only limited use. Some of the algorithms that was investigated are explained in this section.

### 2.5.1 Kalman Filter

Kalman Filter is an algorithm that uses a series of measurements observed over time, to recursively estimate the next outcome. Throughout the report, the theory about Kalman Filter is based on [53], and has previously been used in [38]. As illustrated in Figure 2.28, the main idea can be divided into an iterative process containing two steps: *predict* and *update*. The first step, *predict*, is to predict or projecting forward in time based on the current information, or in other words to obtain the prior estimates for the next time step. The second step, *update*, is using the information gained from the prior estimate and the actual measurement to obtain an improved posterior estimate.
By iterating between these steps, one is able to estimate the next time state, $k$, based on the previous state, $k-1$.

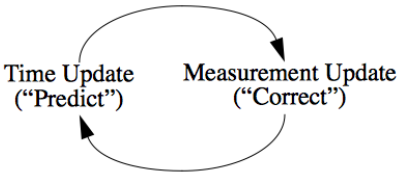



**Figure 2.28:** Main idea of a Kalman filter.[53, Figure 1-1]

**Predict** The system will use Kalman Filter to predict position in the next image. Kalman Filter assumes a state at time $k$ to be evolved from time $(k-1)$. A state can be expressed by the state equation (process model) seen in Equation (2.20).

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} + w_{k-1} \tag{2.20}$$

$$\hat{x}_k^- = A\hat{x}_{k-1} \tag{2.21}$$

The use of Kalman Filter in this system will use no control input and assumes the noise to be gaussian and equal in the entire system. Therefore, the state equation can be reduced to Equation (2.21). This equation is used to obtain an estimate for the next time step.

**Update** The update step uses new observation to update the knowledge to reach a posterior estimate. Equation (2.22) shows the measurement equation. This is the measurement of a new observation, which is used to correct the current prediction model. $H$ is the observation matrix, used to map the state space into observation space. In the system $H = 1$, as there is no mapping between the spaces.

$$z_k = Hx_k + v_k \tag{2.22}$$

Based on prior and posterior estimates, a correction of the estimated state at time $k$ is made. The correction equation is seen in Equation (2.23) where $\hat{x}_k^-$ is the prior estimation from Equation (2.21). $z_k - H\hat{x}_k^-$ gives the difference of the predicted and the actual measurements.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{2.23}$$

$K_k$ is the Kalman gain which is used to decide how much trust you have in the the prior estimate $\hat{x}_k^-$, as it is used to determine how much the estimate should be changed according to the new measurement. The Kalman gain is calculated based on the prior covariance error. If the prior error variance, $P_k^-$, is large, this would imply that the variation of the state is large, and the prior estimation should be discarded and a new estimation made based on new measurements as it is trusted more. Oppositely, if the prior error variance is small or zero, the predicted estimate is not varying much, which would imply that the estimate should not change much.

After the correction step, the posterior covariance error is calculated, which in next iteration is used to calculate the new prior error covariance.

### 2.5.2 Meanshift

The meanshift algorithm tries to find the center of a pixel distribution by calculating the mean pixel location within a window and iterating until the new mean location does not shift any more. Figure 2.29 shows how the meanshift algorithm iterates towards the center of the distribution. The initial window is marked with blue and denoted C1, and has a center point marked with a blue rectangle denoted $C1_o$. The actual centroid of the points within the initial window is however not located at $C1_o$. The centroid of the points is denoted $C1_r$ and is seen with blue filled circle. The idea of meanshift is to move the search window to this new center point. This procedure is done until the center point of the search windows is converged towards the center of distribution.

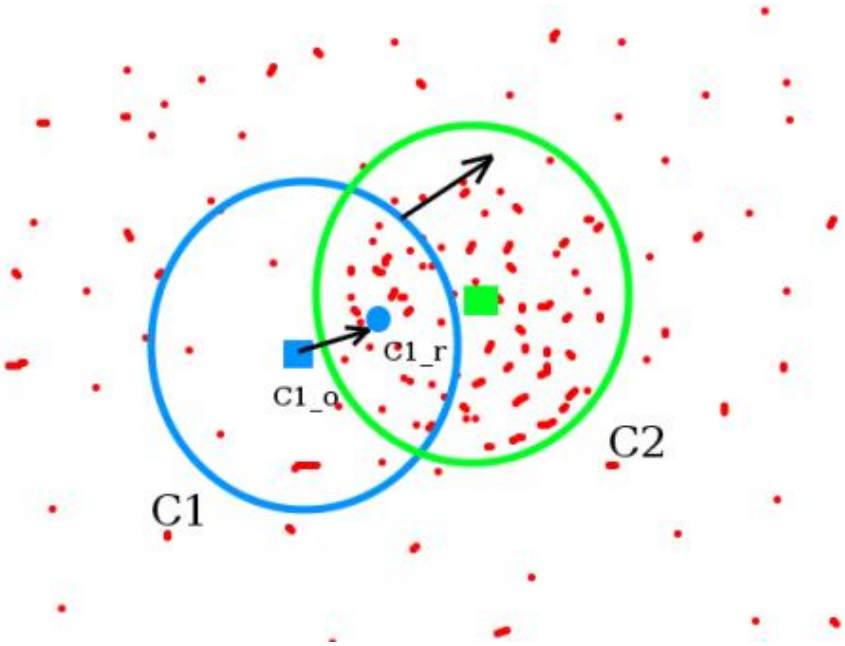



**Figure 2.29:** Meanshift window converging to the center of the distribution.[30]

The pixel distribution can e.g. be created using histogram back projection. This is done by creating a histogram based on the selection of color pixels that belong to the object that must be tracked. The histogram is then used to determine the probability that a certain pixel, in the whole image, belongs to the object from which the sample pixels come from. This leaves a one-channel image where whiter pixels form the distribution of possible object pixels. To

improve the result of the back projection, the probabilities of pixels can be weighed based on how rare the color is in the entire image. More theory of back projection can be found in subsection 2.4.1. An example of using meanshift for tracking an object during a video sequence is seen in Figure 2.30.

(a) (b) (c)

**Figure 2.30:** Example of meanshift tracking. [35]

### 2.5.3 Camshift

Continuously Adaptive Meanshift (camshift) is an extension of the meanshift algorithm that allow the search window size and rotation to adapt accordingly to the target [7]. The first step of camshift is to apply meanshift with a given initial window and wait until the algorithm converges. The idea of camshift is to adjust the search window size such as cases where objects moves closer or further away from the camera we adjust the search window size accordingly. [7] introduces camshift to be able to track faces, and adjusts the search window to be scaled in an elliptical matter as the face is somewhat elliptical. The search window is being scaled according to the current window size, but for each iteration where a defined accuracy is not meet, the window size will be kept increased and mean shift is executed again with updated search window size. An example of using the camshift algorithm for tracking is seen in Figure 2.31. By comparing it with Figure 2.30 it is clear that the detection window scale according to the object.

(a) . (b) (c)

**Figure 2.31:** Example of camshift tracking. [35]

# Chapter 3

# Vehicle Detection and Tracking for NDS at intersections

Our first task upon arrival at the LISA lab was to investigate detection based on depth from a passive stereo vision system would be able to outperform monocular approaches at vehicle detection. Ongoing research in the lab was addressing this problem using in standard monocular video. But as monocular detectors are limited to appearance based detection, obstacles, in this case vehicles, can be difficult to detect in certain situations. Problematic cases are e.g. partial occlusion, unfamiliar vehicles, unfamiliar viewpoints and changing illumination. This leads to late or missed detections. Before reaching a conclusion on this question, we shifted onto working on a more specific variety of the vehicle detection problem. The focus would be on vehicle detection in intersection using stereo vision under challenging conditions, where many of the issues that monocular detectors normally struggles with are found. This research then resulted in the paper *Day and Night-Time Drive Analysis using Stereo Vision for Naturalistic Driving Studies*, which was accepted for the 2015 IEEE Intelligent Vehicles Symposium (IV) in Seoul, South Korea.

## 3.1 Day and Night-Time Drive Analysis using Stereo Vision for Naturalistic Driving Studies

In the paper *Day and Night-Time Drive Analysis using Stereo Vision for Naturalistic Driving Studies* we propose doing object detection in 3D and incorporate ego-motion compensation when tracking and mapping of moving vehicles. The purpose is to use these tracks for detecting NDS events.

# Day and Night-Time Drive Analysis using Stereo Vision for Naturalistic Driving Studies

Mark P. Philipsen[1,2], Morten B. Jensen[1,2], Ravi K. Satzoda[1],
Mohan M. Trivedi[1], Andreas Møgelmose[2], and Thomas B. Moeslund[2]

***Abstract*— In order to understand dangerous situations in the driving environment, naturalistic driving studies (NDS) are conducted by collecting and analyzing data from sensors looking inside and outside of the car. Manually processing the overwhelming amounts of data that are generated in such studies is very comprehensive. We propose a method for automatic data reduction for NDS based on stereo vision vehicle detection and tracking during day- and nighttime. The developed system can automatically register five NDS events, mainly related to intersections, from an existing NDS dictionary. We propose a new drive event which takes advantage of the extra dimension provided by stereo vision.In total, six drive events are selected on the basis of them being problematic to detect automatically using conventional monocular computer vision approaches. The proposed system is evaluated on day- and nighttime data, resulting in drive analysis report. The proposed system reach an overall precision of 0.78 and an overall recall of 0.72.**

## I. INTRODUCTION & MOTIVATION

In 2011 a total of 253,108,389 vehicles were registered in the USA [1], with this many vehicles on the road, crashes are going to happen. In order to analyze the causes of crashes, a lot of research has been done in naturalistic driving studies (NDS). NDS are the study of data from everyday driving, where a wide range of data are collected and analyzed. The purpose is to provide insight into the patterns and behaviors of drivers leading up to and during near-crashes and crashes. [2] defines the scope of understanding vehicular traffic behavior from video as being able to analyze and recognize moving behavioral patterns and being able to describe them using natural language.

The two most notable NDS are the 100-car study [3] and the Strategic Highway Research Program(SHRP) [4], [5], which try to determine patterns and factors impacting the driver's behavior on the road. Discovering these patterns and factors could provide an indication of what sometimes leads to crashes. The previously mentioned 100-car study is based on data captured from 2 million miles on the road, corresponding to nearly 4,300 hours of data, captured from 240 different drivers in the span of 1 year. In the more recent SHRP study, nearly 3,100 drivers have been collecting data from six different places in the USA during the 3-year study. The information is collected with a variety of sensors, providing data such as speed, GPS-position, vehicle dynamics, and video footage covering 360°around the vehicle. All this data is not straightforward to analyze. In [6] the NDS data is categorized into:

- Low-level: sensor data e.g. position from GPS.
- Mid-level: drive events e.g. lane changes or traffic density information.
- High-level: high level semantics e.g. driver behavior or driving styles.

In [7], [8], data reduction for NDS is carried out manually by trained personnel, e.g. hand-labeling a sequence of video footage. The hand-labeling of events must be consistent, therefore a dictionary, such as the one found in [9] is used. This NDS dictionary was adapted in SHRP. If processing of the collected data could be automated, it would allow for more comprehensive studies. 61% of daytime and 38% of night time crashes involve multiple vehicles [10], therefore NDS events based on detection and tracking of other vehicles are essential. Since a disproportionate number of crashes happen at night [11], it is important to also to detect vehicles during nighttime. Intersection are especially prone to crashes, since vehicles need to cross paths. To the best of the authors' knowledge, non of the current published work in the area of data reduction for NDS, utilize stereo vision for automatic drive analysis. [12] detects objects' 3D location and orientation in intersection traffic scenes using scene information from stereo images. However, it is not used for NDS, nor evaluated on nighttime data. In [13], the symmetry and color rear-lamp pairs are utilized for detection and tracking. A similar approach is used in [14], where

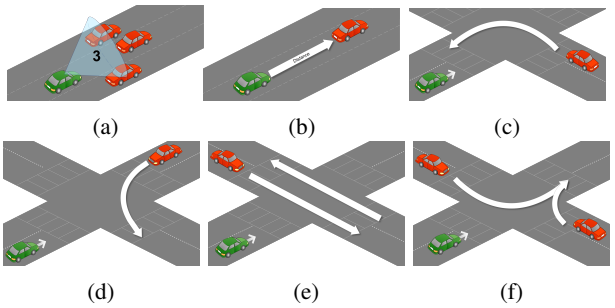



Fig. 1: Automatically detectable data and events. Green is ego-vehicle, red is other vehicles. **(a)** Avg. vehicles in front of ego-vehicle **(b)** Distance to rear-end of front vehicle. **(c)** Vehicle turn to opposite direction. **(d)** Vehicle turn left across path. **(e)** Vehicle drive straight across path. **(f)** Vehicle turn to same direction.

[1]Computer Vision and Robotics Research Laboratory, UC San Diego, La Jolla, CA 92093-0434, USA
[2]Visual Analysis of People Laboratory, Aalborg University, 9000 Aalborg, Denmark.

the proposed system is limited to only consider scenarios where vehicles are fully visible. The same paper provides a brief overview of how monocular vehicle detection has trouble dealing with variations in lighting, weather, changing vehicle shape, and color. The most notable issue with these approaches is that the detection is highly depended on both rear-lamps being visible. Stereo vision is used in [15] to track a vehicle after detecting it using monocular vision. The purpose of introducing stereo vision is to make their system more robust to occlusion. This is also discussed in [16], where the introduction of 3D shows promising results and alleviate some occlusion issues. Most vehicle detection is done in the monocular domain, as e.g in [17] where lane detection is included for localizing and tracking the vehicles and in [18] where parts of vehicles are detected and tracking, the same paper remarks that detecting the turning vehicles in intersection is especially challenging. Monocular systems which aspire to detect vehicle from all possible viewpoints and under changing light conditions, requires a large amount of varied training data.

In this paper, we introduce a stereo vision system for automatic NDS data reduction on both day- and nighttime data. This enables detection and tracking of vehicles in scenarios that would be problematic for monocular detectors. The proposed system will handle a handful of NDS events, which especially benefit from the extra dimension in stereo vision. The treated NDS events are illustrated in Figure 1.

The contributions made in this paper are:

- Using stereo vision for automatic data reduction for NDS on both day- and nighttime data, with focus on intersections (Figure 1c, 1d, 1e, 1f).
- Using stereo vision for determining the average number of vehicles in front of the ego-vehicle. (Figure 1a).
- Introducing a new NDS event: Average distance to vehicles directly in front of the ego-vehicle. (Figure 1b).

The rest of the paper is organized as follows: In Section II, related work in the area of data reduction for NDS is presented followed by an overview of recent work on stereo vision for obstacle detection on roads. An overview of the developed system and the drive semantics that we wish to label automatically are presented in Section III. In section IV the methods which constitute the system are presented. The results are discussed in Section V, followed by concluding remarks in Section VI.

## II. RELATED WORK

Before developing a stereo vision system to be used for generating a drive analysis report, we briefly present some of the recent research published with regards to automatic NDS data reduction. Furthermore, we introduce some of the recent and most notable work in the field of object detection with stereo vision. This is limited only consider stereo vision with the purpose of either improving the disparity with regard to object detection in traffic scenes and work that utilizes stereo vision for vehicle detection.

### A. NDS Data Reduction

In [19], the statistics of near-crashes and crashes are used to identify factors related to driver associated risks, such as age, experience, gender, and demographic. Similar studies are conducted in [20], where results show that near-crash and crash percentage among teenagers were 75 % lower in the presence of adult passengers and 96 % higher among teenagers with risky friends. In [21] NDS are used to quantify distracting activities from e.g. a mobile phone, that results in loss of concentration. Results show that a driver in average is engaged in a distracting activity every 6 minutes, which could results in a near-crash or crash.

The process for manual data reduction for NDS is quite comprehensive and time consuming, it is therefore desirable to automate it by e.g. applying computer vision to understand the traffic scene. An example of one such study is seen in [6], where monocular computer vision and information from the CAN bus is used to automatically detect 23 drive events, including lane position, vehicle localization within lanes, vehicle speed, traffic density, and road curvature. In [22], a system is developed for automatic labeling driver behavior events with regards to overtaking and receding vehicle detection. This type of system is categorized as Looking-In and Looking-Out(LiLo), which is discussed in depth in [23]. LiLo fits well with using multiple inputs to understand the driver's behavior. An example of Li is [24] where driver behavior with respect to hand activity is evaluated.

### B. Object Detection Using Stereo Vision

In [25], a review of the research conducted since 2005 in both monocular and stereo vision with respect to vision-based vehicle detection is presented. Before 2005, [26] conducted a review of on-road vehicle detection. Most of the research published with regard to vehicle detection is evaluating the methods on data, representing a limited part of the challenges. In [27] the accurate and efficient Semi-Global Matching(SGM) is introduced. SGM make use of epipolar geometry, and in most cases a set of rectified stereo images. Horizontal lines in the images are used as a scan lines, matches are then found with a 1D disparity search. A match for a pixel in the left image is found in the right image by searching through the corresponding horizontal line and locating the most similar block to a reference block around the original pixel in the left image. The offset between these pixels is known as the disparity, which is directly related to the distance to the corresponding object. In the same paper, the LR-RL-consistency check is proposed for reducing noise in the calculated disparity image.

In [28], [29], the so-called v-disparity is generated and used for separating objects from the ground/road surface. The v-disparity examines the vertical coordinates in a (u,v) image coordinate system and is constructed using a disparity map from, e.g. the SGM algorithm. What is especially histograms are calculated for each row in the disparity. Significant surfaces in the disparity map will then show up as lines in the v-disparity.

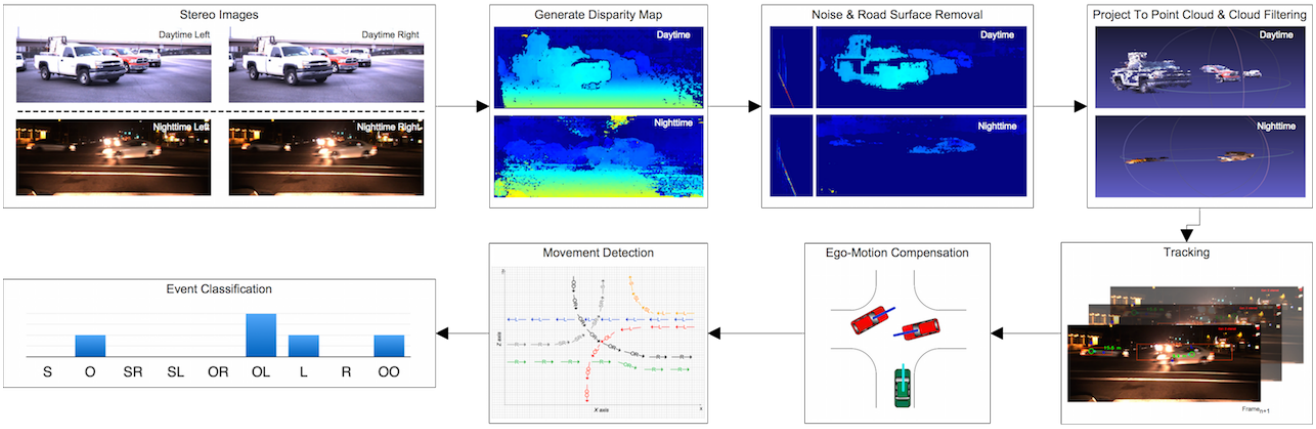



Fig. 2: Flow diagram of the developed system, illustrated with both day- and nighttime data.

In [30], 6D-vision is introduced, where features are found in the left monocular image and then located in 3D by using the stereo images. For each feature point, a Kalman filter is used to estimate a 6D vector consisting of the 3D position and 3D motion vector. [30] is able to do ego-motion compensation by identifying the static 6D points. The predicted static world points are then compared to the remaining points to isolate the ego-motion. In [31], this work is continued and by using 6D-vision, tracked feature points are represented as stixels, which is a vertical areas of equal disparity. The tracked objects are being classified using prior knowledge of vehicle shapes. Alternatively, objects can be classified using clustering in the disparity map, as seen in [28]. In [32], [33], temporal and scene priors from good conditions are used with the purpose of improving the disparity map in adverse weather conditions, such as night, rain, and snow. Using these priors, the object detection rate improves on a database of 3000 frames including bad weather while reducing the false positive rate.

## III. SYSTEM OVERVIEW

The flow of the system is shown in Figure 2. A Bumblebee XB3 stereo camera is used to acquire stereo image pairs with a capture speed of 16 FPS in an resolution of 1280x960. These images are rectified using the factory calibration. A disparity map is generated and noisy pixels are removed. The road surface is removed using the detected line in a corresponding v-disparity. The remaining pixels are considered vehicle candidate pixels. These pixels are projected into 3D world coordinates and outliers are removed. Clusters are found by grouping points that are close neighbors. Each sufficiently big cluster is regarded as a detected vehicle. The clusters' center points are used for nearest neighbor tracking between frames and for determining the distances from ego-vehicle to detected vehicles. Before the detected vehicles' movement between frames is determined, their movement is adjusted according to the ego-vehicles motion. Finally, the vehicles are tracked, and NDS event are detected and logged in a drive analysis report.

## IV. METHODS

In this section all of the methods which are used in the proposed system are presented. The section is divided into subsections corresponding to the stages seen in Figure 2.

### A. Generate Disparity Map & Noise Removal

Stereo images captured under poor conditions may contain very dark, very bright, or otherwise untextured regions causing noise and artifacts in the disparity map. Besides this, inconsistencies between the image pairs may arise in an uncontrolled environment e.g. because of reflections in the camera lenses. A noisy disparity map is bound to lead to problems later in the system, therefore the disparity map is post-processed in an attempt to reduce the number of noisy pixels. Figure 3 shows an examples of a two noisy disparity image in the left box. In the right box, the two corresponding noise reduced disparity images are seen.

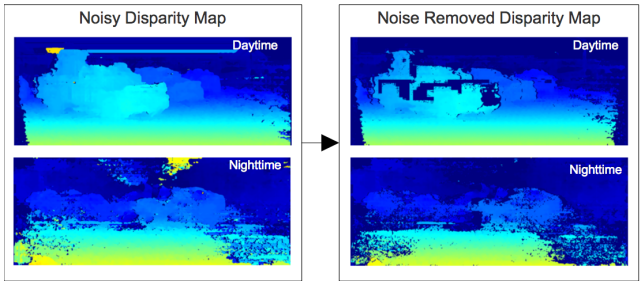
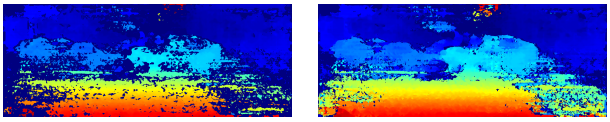



Fig. 3: Examples of raw day and night-time disparity maps and the corresponding noise reduced disparity maps.

*1) LR-RL consistency check:* By using first left and then right image as reference, when searching for matches in the stereo pair, a so called LR-RL consistency check is done. The LR-RL consistency check is used a little differently compared to the original proposal in [27] and the implementation provided by OpenCV. We wish to keep as many disparity pixels as possible for later stages. In case of a disparity difference in the consistency check, the lowest

value is simply selected as the output pixel, rather than discarding the pixel entirely. The result this modified LR-RL consistency check versus the traditional, with a maximum disparity difference of 5 is seen in Figure 4.



(a) (b)

Fig. 4: (a) Traditional LR-RL consistency check. (b) Proposed LR-RL consistency check.

*2) Temporal consistency check:* By looking at the value of a pixel in the previous frame and comparing it with the value in the current frame, a significant amount of noise can be removed. This check is based on the assumption that it is uncommon for correctly matched disparity pixels to erratically change value from frame to frame. In case of movement, this check will result in a dark edge around the moving objects in the disparity map.

*3) Monocular color check:* Large uniform regions are problematic when calculating the disparity map as they may result in artifacts and noise. Especially in night scenes, a large part of the images consist of dark regions. The same is the case in brightly lit scenes. Over and under exposure can be handled to some degree by adjusting the camera accordingly. Regions are removed by searching for pixels with values very close to either of the extremes in the left monocular RGB image. As these pixels are considered as sources of noise, they are simply masked out in the disparity map.

### B. Road Surface Removal

The noise reduced disparity map contains disparities for road surface pixels and obstacle pixels. Since the vehicles that must be detected are found among the obstacle pixels, the disparity pixels associated with the road surface must be removed.

*1) Road surface detection using v-disparity:* The road surface is found by searching for the most significant line in the v-disparity using RANSAC. This method will not work well unless parts of the road surface is visible. In cases where a satisfactory line cannot be found, the last good line is used for road surface removal. Additionally, the line parameters are filtered using a Kalman filter to smooth out faulty road surface estimations. The calculated line is then used as a threshold for determining if pixels belong to objects above the road surface.

### C. Project To Point Cloud & Cloud Filtering

*1) Projecting disparity image to 3D point cloud:* Using the camera's focal length $f$(in pixels) and baseline $b$(in meters), along with the calculated disparity $d$(in pixels), the actual distance $z$ to objects in the camera's view can be found. The remaining $x$ and $y$ components of the world coordinate are the column and row index of the disparity pixel. Equation (1) shows the relations that make this possible.

$$z = \frac{f \cdot B}{d} \quad x = \frac{col \cdot z}{f} \quad y = \frac{row \cdot z}{f} \tag{1}$$

Only disparity map pixels above a certain threshold are projected to 3D world points. This is due to the exponentially increasing inaccuracy with lower disparity values, which makes it difficult to determine whether point is noise, background, or in fact belong to an object.

### D. Vehicle Segmentation in Point Cloud

The acquired point cloud is preprocessed in the following steps in order to clean up the data points for segmentation into clusters constituting vehicles.

1) A pass through filter removes near and distant points. These point degrade performance and can be removed since they are of little interest.
2) Remaining points are downsampled using a voxel grid, insuring an even distribution of points and greatly reduces the number of points, resulting in a reduced processing time.
3) Outlier removal based on mean distance to neighboring points compared to a global mean and standard deviation.

Clusters are found by creating a k-d tree which organizes points according to their distance to neighbors, this enables efficient searches in the 3D point cloud.

### E. Vehicle Tracking

In order to understand what other drivers are doing in relation to the ego-vehicle, their vehicles must be tracked. This is done based on the center point of each of the segmented clusters, these center points are filtered using a Kalman filter in order to make the prediction of a match in the following frame more accurate. The best match is found using the euclidean distance between the predicted center point and the segmented center point in the next frame.

### F. Visual Ego-Motion Compensation

For compensating for the subject vehicle's ego-motion while e.g. approaching an intersection, we utilize the *LIBVISO2: C++ Library for Visual Odometry 2* which can calculate the translation and rotation between moving frames captured by stereo camera. For further explanation we refer to [34].

### G. NDS Event Detection

For all detected vehicles, individual frame to frame movements are categorized to form a basis for determining which NDS events have occurred. Figure 5 shows the different movements that are detected. Based on the movements, a histogram is created and used for classifying the NDS event type. The arrows symbolizes vectors between vehicle center points and the attached letters indicate the movement type.

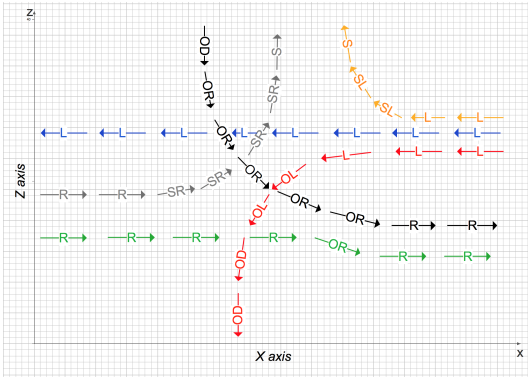



Fig. 5: Movements detected by the system. Every movement is seen from the viewpoint of the ego-vehicle. S (same direction), SR (same direction and towards the right), SL (same direction and towards the left), O (opposite direction), OR (opposite direction and towards the right), OL (opposite direction and towards the left), R (right), L (left).

In addition to the movements seen in Figure 5, a category is created for cases where little or no movement are detected. This category is labeled OO. A movement histogram is created for each vehicle, where a bin corresponds to one type of movement. The histogram is matched to a training histogram for each of the NDS events. Figure 6 shows an example on one such histogram.

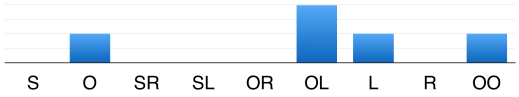



Fig. 6: Example of histogram of detected movements.

The following four NDS event are each described by specific training histograms.

- Other vehicle turn left across path.
- Other vehicle turn onto opposite direction.
- Other vehicle drive straight across path.
- Other vehicle turn onto same direction.

Additionally, we quantify the existing NDS event, *Average number of cars in front of ego-vehicle*. The total number of detected vehicles is recorded for each frame throughout the entire video sequence and the average is calculated when the NDS rapport is generated. The purpose is to quantify the traffic scene density, by the average number of vehicles in front for the ego-vehicle. Finally we propose a new NDS event which is the *average distance to the rear end of the vehicle directly in front* of the ego-vehicle. This is meant as a measure of the ego-vehicle driver's aggressiveness.

## V. RESULTS

The system is evaluated based on 14 day and 12 night video sequences. In each sequence the occurrence of each event, also referred to as the ground truth (GT) is manually hand-labeled. Precision and recall are found using the system output(SO), correctly detected events(TP), incorrectly detected events(FP), and finally missed events(FN). Precision and recall are calculated according to (2).

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \tag{2}$$

Precision is the ratio of correct event detections compared to the actual number of events. Recall is the ratio of correct event detections compared to the total number of detections.

In Figure 7, an example of the NDS event *left turn across Path* is seen. The blue cross represents the cluster's center of mass, the blue text, next to the blue circle, is the measured distance from ego-vehicle to the vehicle's closest point. The blue text stating, "R" and "OR" next to the green circle, tells that the detected vehicles' movements are determined to be right and opposite right movement, respectively.

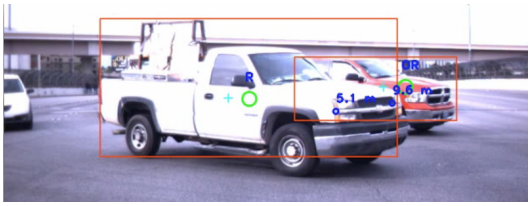


Fig. 7: Partially occluded vehicle detected in a left turn.

In Table I the results are shown. The 12 nighttime video sequences consists of a total of 3933 frames. The 14 daytime videos consists a total of 4992 frames. A result of e.g 35/32 corresponds to 32 manually annotated GT events and 35 system detections. For calculating the precision and recall, each classification done by the system is manually defined as either a TP, FP, or FN. The listed precision and recall numbers are a total from both day- and nighttime data.

TABLE I: Summary of drive analysis from NDS. Results are listed as [SO/GT]. P and R are abbreviations for precision and recall.

| ***Drive Behavior Event*** | **Daytime** | **Nighttime** | **P** | **R** |
|---|---|---|---|---|
| Right - straight across path | 35/32 | 5/19 | 0.95 | 0.63 |
| Left - straight across path | 45/34 | 11/33 | 0.87 | 0.67 |
| Left turn across path | 5/5 | 20/1 | 0.75 | 1 |
| Turn onto opposite dir. | 32/37 | 41/15 | 0.68 | 0.93 |
| Short turn onto same dir. | 7/5 | 9/5 | 0.63 | 1 |
| Long turn onto same dir. | 1/16 | 1/8 | 1 | 0.09 |
| Avg. number of cars | 1.67/1.74 | 1.6/1.3 | NA | NA |
| Avg. distance to car | 8.73 m | 10.98 m | NA | NA |

Table I indicates that the proposed system is able to handle most of the events. However, the system often had difficulties with detecting vehicle far away, especially at night, resulting in poor performance for vehicles turning onto the same direction. The average distance to the vehicle in front of the ego-vehicle was found to be shorter during the daytime, which could indicate people are using a larger safety distance when the visibility is decreased, or it might be a result of lower traffic density in those hours.

Future improvements to the system includes creating a mask using top-hat morphology instead of the monocular

color check for removing big texture-less regions. The color check method removes big parts of black and white vehicles. It was not possible to verify the accuracy of the measured average distance to vehicles, based on the video clips that were used for testing. Finding the ground truth using a laser range finder would be interesting to establish the accuracy of the distance measurements. The ego-motion compensation proved useful, but the estimates were noisy at times. The exact motion of the vehicle can alternatively be extracted from the CAN bus. More significant was the impact from the tracked center points of vehicles, which would shift back and fourth based on the quality of the disparity map. These two issues were the main contributors to wrong event classifications. The varying performance can be attributed to the difficult conditions under which much of the image data was collected. Most false negatives were a result of occlusions or incorrect road surface detection.

## VI. CONCLUSION

We presented a system using stereo vision for automatic data reduction in NDS with focus on intersections and distance measurements. The use of stereo vision is considered beneficial, especially in scenarios such as the one seen in Figure 7. The system proved to work in both day and nighttime conditions with a drop in overall performance for the night sequences. Experiments show very promising detection, trajectory, and automatic event classification rates with an overall precision of 0.78 and recall of 0.72.

Future work includes looking at additional NDS events where stereo vision can be utilized and identifying the exact accuracy of the distance measurements achieved with stereo vision.

## REFERENCES


[1] U.S. Department of Transportation - Bureau of Transportation Systems, "National transportation statistics," 2011.

[2] B. T. Morris and M. M. Trivedi, "Understanding vehicular traffic behavior from video: a survey of unsupervised approaches," *Journal of Electronic Imaging*, vol. 22, pp. 041 113–041 113, 2013.

[3] V. L. Neale, T. A. Dingus, S. G. Klauer, J. Sudweeks, and M. Goodman, "An overview of the 100-car naturalistic study and findings," *National Highway Traffic Safety Administration, Paper*, 2005.

[4] L. N. Boyle, S. Hallmark, J. D. Lee, D. V. McGehee, and N. J. Ward, "Integration of analysis methods and development of analysis plan," *Transportation Research Board of the National Academics, Tech. Rep.*, 2012.

[5] G. Davis and J. Hourdos, "Development of analysis methods using recent data: Shrp2 safety research," *Transportation Research Board of the National Academics, Tech. Rep.*, 2012.

[6] R. Satzoda and M. Trivedi, "Drive analysis using vehicle dynamics and vision-based lane semantics," *Intelligent Transportation Systems, IEEE Transactions on*, pp. 1–10, 2014.

[7] D. LeBlanc, "Road departure crash warning system field operational test: methodology and results. volume 1: technical report," 2006.

[8] T. A. Dingus, S. Klauer, V. L. Neale, A. Petersen, S. E. Lee, J. Sudweeks, M. A. Perez, J. Hankey, D. Ramsey, S. Gupta, C. Bucher, Z. R. Doerzaph, J. Jermeland, and R. Knipling, "The 100-car naturalistic driving study phase ii, results of the 100-car field experiment," 2006.

[9] Virginia Tech Transportation Institute (VITTI), "Researcher dictionary for video reduction data," Tech. Rep., 2012.

[10] National Highway Traffic Safety Administration, "Passenger vehicle occupant fatalities by day and night a contrast," 2012.

[11] US National Library of Medicine National Institutes of Health, "Road traffic casualties understanding the nighttime death toll," 2012.

[12] A. Geiger, M. Lauer, C. Wojek, C. Stiller, and R. Urtasun, "3d traffic scene understanding from movable platforms," *Pattern Analysis and Machine Intelligence (PAMI)*, 2014.

[13] R. O'Malley, E. Jones, and M. Glavin, "Rear-lamp vehicle detection and tracking in low-exposure color video for night conditions," *Intelligent Transportation Systems, IEEE*, vol. 11, pp. 453–462, June 2010.

[14] R. O'Malley, M. Glavin, and E. Jones, "Vehicle detection at night based on tail-light detection," 2010.

[15] E. Ohn-Bar, S. Sivaraman, and M. Trivedi, "Partially occluded vehicle recognition and tracking in 3d," in *Intelligent Vehicles Symposium (IV), 2013 IEEE*. IEEE, 2013, pp. 1350–1355.

[16] N. Buch, S. A. Velastin, and J. Orwell, "A review of computer vision techniques for the analysis of urban traffic," *Intelligent Transportation Systems, IEEE*, vol. 12, pp. 920–939, 2011.

[17] S. Sivaraman and M. M. Trivedi, "Integrated lane and vehicle detection, localization, and tracking: A synergistic approach," *Intelligent Transportation Systems, IEEE*, vol. 14, pp. 906–917, 2013.

[18] ——, "Vehicle detection by independent parts for urban driver assistance," *Intelligent Transportation Systems, IEEE*, vol. 14, pp. 1597–1608, 2013.

[19] F. Guo and Y. Fang, "Individual driver risk assessment using naturalistic driving data," *Accident Analysis & Prevention*, vol. 61, pp. 3 – 9, 2013, emerging Research Methods and Their Application to Road Safety Emerging Issues in Safe and Sustainable Mobility for Older Persons The Candrive/Ozcandrive Prospective Older Driver Study: Methodology and Early Study Findings.

[20] B. G. Simons-Morton, M. C. Ouimet, Z. Zhang, S. E. Klauer, S. E. Lee, J. Wang, R. Chen, P. Albert, and T. A. Dingus, "The effect of passengers and risk-taking friends on risky driving and crashes/near crashes among novice teenagers," *Journal of Adolescent Health*, vol. 49, pp. 587 – 593, 2011.

[21] S. P. McEvoy, M. R. Stevenson, and M. Woodward, "The impact of driver distraction on road safety: results from a representative survey in two australian states," *Injury prevention*, vol. 12, pp. 242–247, 2006.

[22] R. Satzoda and M. Trivedi, "Overtaking amp; receding vehicle detection for driver assistance and naturalistic driving studies," in *Intelligent Transportation Systems (ITSC), IEEE*, 2014, pp. 697–702.

[23] M. M. Trivedi, T. Gandhi, and J. McCall, "Looking-in and looking-out of a vehicle: Computer-vision-based enhanced vehicle safety," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 8, pp. 108–120, 2007.

[24] E. Ohn-Bar, S. Martin, and M. M. Trivedi, "Driver hand activity analysis in naturalistic driving studies: challenges, algorithms, and experimental studies," *Journal of Electronic Imaging*, vol. 22, pp. 041 119–041 119, 2013.

[25] S. Sivaraman and M. M. Trivedi, "A review of recent developments in vision-based vehicle detection," *Intelligent Vehicles Symposium (IV), IEEE*, pp. 310–315, 2013.

[26] Z. Sun, G. Bebis, and R. Miller, "On-road vehicle detection: a review," *Pattern Analysis and Machine Intelligence, IEEE*, pp. 694–711, 2006.

[27] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Computer Vision and Pattern Recognition (CVPR), IEEE*, 2005, pp. 807–814.

[28] A. Broggi, A. Cappalunga, C. Caraffi, S. Cattani, S. Ghidoni, P. Grisleri, P. Porta, M. Posterli, and P. Zani, "Terramax vision at the urban challenge 2007," *Intelligent Transportation Systems, IEEE*, pp. 194–205, 2010.

[29] R. Labayrade, D. Aubert, and S. S. Ieng, "Onboard road obstacles detection in night condition using binocular ccd cameras," in *in ESV*, 2003.

[30] C. Rabe, U. Franke, and S. Gehrig, "Fast detection of moving objects in complex scenarios," in *Intelligent Vehicles Symposium, IEEE*, 2007, pp. 398–403.

[31] F. Erbs, A. Barth, and U. Franke, "Moving vehicle detection by optimal segmentation of the dynamic stixel world," in *Intelligent Vehicles Symposium (IV), IEEE*, 2011, pp. 951–956.

[32] S. Gehrig, M. Reznitskii, N. Schneider, U. Franke, and J. Weickert, "Priors for stereo vision under adverse weather conditions," in *Computer Vision Workshops (ICCVW), IEEE*, Dec 2013, pp. 238–245.

[33] S. Gehrig, N. Schneider, and U. Franke, "Exploiting traffic scene disparity statistics for stereo vision," in *Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE*, 2014, pp. 688–695.

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

# Chapter 4

# Traffic Light Recognition Survey

The literature survey which forms the basis for our two TLR survey papers is done by literature search using the following keywords: traffic, light, signal, detection, recognition. Additional research not captured by the search is found by looking up references in the found articles.

## 4.1 Vision for Looking at Traffic Lights: Issues, Survey, and Perspectives

Currently no survey of TLR research exists. We provide an overview of the methods employed for detection, classification and tracking as well as the used color spaces and features in relevant TLR research back to 2004, but primarily focused on research made from 2009 and onward. By looking at the evaluation of current research in TLR we conclude that the introduction of a common evaluation practice will significantly help advancement in the area. This includes settling on descriptive and meaningful evaluation metrics as well as the introduction of a challenging public training and testset. Our journal paper: *Vision for Looking at Traffic Lights: Issues, Survey, and Perspectives*, which is submitted for IEEE Transactions on Intelligent Transportation Systems (ITS), gives an extensive overview of existing research and provides an in-depth review of the current evaluation methodology as well introduce our proposal for a common future evaluation procedure.

# Vision for Looking at Traffic Lights: Issues, Survey, and Perspectives

Mark P. Philipsen, Morten B. Jensen, Andreas Møgelmose, Thomas B. Moeslund, and Mohan M. Trivedi

***Abstract*—This paper provides an overview of recent work on the problem of traffic light recognition (TLR). The aim is to elucidate which areas have been thoroughly researched and which have not, thereby uncovering opportunities for further improvement. An overview of the applied methods and noteworthy contributions from a wide range of recent papers is presented, along with the corresponding evaluation results. The evaluation of TLR systems is studied and discussed in depth and we propose a common evaluation procedure, which will improve future evaluation and ease comparison. To provide a shared basis for comparing future TLR systems we publish an extensive public database based on footage from US roads. The database contains annotated stereo sequences, captured under varying light and weather conditions, which should challenge future TLR systems.**

***Index Terms*—Traffic light recognition, traffic signal recognition, object detection, computer vision, machine learning, intelligent transportation system, active safety, driver assistance systems**

## I. Introduction

THE automobile revolution in the early 20th century led to a massive increase in road transportation and contemporary road network was incapable of handling the rapidly increasing traffic load. To allow for efficient and safe transportation, traffic control devices (TCD) were developed for guiding, regulating, and warning drivers. TCDs are elements of the infrastructure that communicate to the drivers, examples are: signs, signaling lights and pavement markings [1]. Figure 1 shows an example of a road scene with some of the many TCDs. TCDs are especially important in complex

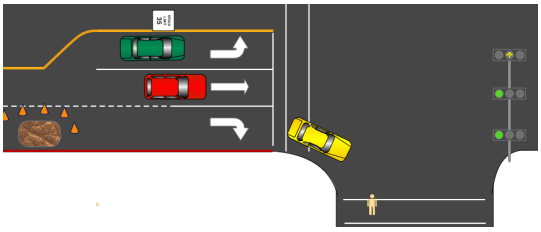


Fig. 1: Traffic Control Devices (TCD) for safe and efficient traffic flow.

environments such as intersections, where a lot of information needs to be communicated. Informing drivers is a balance between providing sufficient information while avoiding to burden and distract the drivers excessively. A driver's cognitive ability is limited by the amount of information and the time available to comprehend the information. Thereby, high speed and overwhelming amounts of information can lead to errors from stress and oversights [1]. TCDs are installed to make traffic run smoothly and safely by guiding road users and assigning the road resource fairly. For TCDs to function properly, all road users are required to abide, otherwise dangerous situations occur. In some cases drivers purposely disregarded the TCDs as e.g. over 1/3 of Americans admit to having purposefully run a red light during the past month [2]. In many cases failure to comply is unintentional and caused by misunderstandings, negligence, or faulty TCDs. Most of the time when driving, a person's attention is divided among driving tasks such as changing lane, activating turn signals, adjusting the speed, reading signs etc. while also listening to the radio, snacking, and thinking about what is for dinner. For all of this to be possible at the same time, much of it happens unconsciously and based on expectations of what will happen. When the unexpected suddenly happens it is perceived with an added delay. In emergency situations a decision of whether to dodge or break must be made. The additional mental load associated with making decisions increases the reaction time further. A considerable share of crashes are a result of delayed reaction time caused by failure to recognize a hazardous situation before a crash becomes inevitable. The complex task of driving is easy most of the time to an experienced driver, because many driving subtasks are automated. The tasks have been practiced over and over with little variation and they become a fixed sequence of actions. When such tasks have be come automated, they become effortless. When driving is easy and the driver is unfocused, critical events are easy to miss. Stressful driving on the contrary means that the driver is very focused and attentive. Therefore a lot of effort is put into the task of driving which quickly leads to fatigue. Inexperienced drivers under complex driving conditions are in particular exposed to this and may become overloaded resulting in missed critical events [3].

Unlike for sign recognition and pedestrian detection, no surveys of TLR research exist. Furthermore, most published TLR systems are evaluated based on local datasets with a limited number of traffic lights(TLs) and little variation. This makes comparison between existing methods and new contributions difficult. The contributions made in this survey

M.P. Philipsen and M.B. Jensen are with the Computer Vision and Robotics Research Laboratory, University of California, San Diego, La Jolla, CA 92093-0434, USA and the Visual Analysis of People Laboratory, Aalborg University, 9000 Aalborg, Denmark.

A. Møgelmose and Prof. T. B. Moeslund are with the Visual Analysis of People Laboratory, Aalborg University, 9000 Aalborg, Denmark.

Prof. M.M. Trivedi is with the Computer Vision and Robotics Research Laboratory, University of California, San Diego, La Jolla, CA 92093-0434, USA.

paper are thus threefold:

1) Provide an overview of current TLR papers' method choices and contributions.
2) Introduce a common evaluation procedure for future TLR systems.
3) Publish an expanded high resolution, annotated, stereo video database, with day and night video sequences.

The paper is organized as follows: Section II provides an introduction to driver assistance systems (DAS). Section III talks in general about computer vision systems for TLR. Section IV, gives an overview of the possible appearances of TLs, along with common challenges that TLR systems are subject to. In section V a decomposition of TLR systems is presented, which enables easy comparison between the recent work. In section VI recent work is examined and the applied methods are summarized, followed by overview tables with methods and evaluation results. Section VII reviews the way TLR system performance has been evaluated up to this point. Following this we propose five metrics that we suggest should be used to evaluate TLR systems. In section VIII, we present a new stereo database for evaluation of future TLR systems. In section IX, experiences, and future possibilities are discussed. Finally, section X rounds of with summarizing and concluding on the findings made through out the paper.

## II. Driver Assistance Systems

The efficiency and advancement of surface transportation systems is an influencing factor in the productivity of nations. They fundamentally affect the time spend on transportation and the mobility of the workforce. They impact the environment and energy consumption, which in turn dictate foreign policy. Since transportation is a major part of peoples lives, their health and well being is directly related to the efficiency, safety and cleanness of the transportation systems. To improve the current transportation systems, innovation and development in sensing, communication, and processing is necessary [4], [5]. The future perspective of intelligent transportation systems is autonomous networks of vehicles and infrastructure, were transportation is ordered on demand from transport service providers. The high initial expenditure, worries, and effort required when owning and driving private vehicles will no longer be necessary [6]. In the meantime and as long as humans are in the loop, DAS is an important part of reducing the number of fatalities and injuries.

Traffic accidents are one of the major causes of death around the world. In addition to the distressed caused by the loss of life and mobility, the monetary cost runs in billions of dollars [7]. The countries with the highest fatality rates are middle-income countries, which recently have experienced a rapid motorization. High income countries are experienced steadily falling fatality rates for half a century due to the implementation of a long list of legislative, emergency care, vehicle safety, and road environment measures. At first people needed to be forced to be safe through regulations, which was something that began in the 1960s [8]. With the introduction of New Car Assessment Program (NCAP) in 1978, safety became a significant parameter for people when buying cars. Since then, car manufacturers have fought to reach high scores in safety tests [3]. DAS are increasingly becoming part of safety ratings and are valuable selling points. Since people's attention tend to be divided and occasionally fails to sense potential dangers in time, lives can be saved by having DAS monitor the environment and depending on the situation either warn or intervene. Since the purpose of DAS it to support the driver, it must make up for the deficiencies of the driver. An example of a driver deficiency where DAS potentially can support the driver is noticing and recognizing TCDs. Studies show that drivers notice some TCDs better than others, speed limit signs are almost always noticed, while signs warning about pedestrian crossings are mostly overlooked [9]. A traffic sign recognition system for DAS should therefore foremost be warning about pedestrian crossings.

Currently the focus of research in computer vision systems for vehicles is divided in two. Major industrial research groups, such as Daimlar and Google, are investing heavily in autonomous vehicles and attempt to make computer vision based system for the existing infrastructure. Other research done by academic institutions, such as the LISA lab at UC San Diego and LaRA at ParisTech, are targeting DAS, which is already available to consumers in some high-end models from car manufacturers such as Audi, BMW, Mercedes-Benz, Tesla, Volvo, etc.. Existing commercial DAS capabilities include, warning of impending collisions, emergency breaking, automatic lane changing, keeping the advertised speed limit, and adaptive cruise control. In addition to DAS directly implemented as part of the car, DAS has been seen implemented on mobile platforms. As smartphones are becoming more powerful both in terms of processing power and an expanding array of powerful sensors such as, cameras, GPS, accelerometers, and gyroscopes. In [10], [11], driver advisory applications are introduced providing several services using an iPhone mounted inside the car, with the camera facing the road in front of the car. DAS related research can also be applied elsewhere, e.g. to help visual impaired navigate in the traffic scene as seen in [12].

## III. Role of Computer Vision in Traffic Light Recognition

For all parts of DAS the urban environment possesses a wealth of challenges, especially to systems that rely on computer vision. An important challenge is recognizing TLs at intersections. In 2012, 683 people died and 133,000 people were injured in crashes that involved red light running in the USA [13]. Ideally, TLs should be able to communicate both visually and using infrastructure to vehicle (I2V) by means of radio communication. Introducing I2V on a large scale requires substantial investments in infrastructure, which is unlikely in the near future. When some form of computer controlled automation is involved in controlling dangerous objects such as vehicles, safety and reliability are of utmost importance. Examples of dangerous scenarios, where DAS

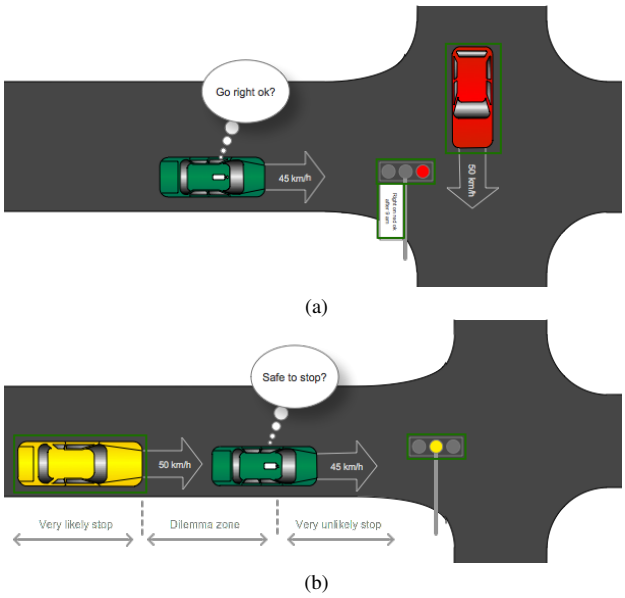



Fig. 2: A fused DAS system in intersection scenarios. (a) safe/legal to turn right on red?. (b) Assistance in the dilemma zone.

is involved, could be a false positive caused by e.g., a tail light or pedestrian crossing light, resulting in the system determining that a red light is imminent, when this is not the case. This might lead to unnecessary distraction of the driver, or even affecting the driver to perform an emergency braking operation. Worse yet, if a red light is missed or miss classified as green, it could potentially lead to the driver accidentally running a red light. Intersections are some of the most complex challenges that drivers encounter, making recognition of TLs an integral part of DAS in the transitional period between manually controlled cars and the introduction of I2V for TL or a fully autonomous networks of cars.

Intersections are some of the most demanding challenges the driver must navigate. Navigating intersections requires awareness of surrounding objects, selecting lane, awareness of signs and signals, making stop or proceed decision while maintaining lane position, appropriate speed and turn rate. At intersections with TLs, the yellow light dilemma present drivers with a decision of whether it is safe/possible to stop before entering the intersection or to keep going and cross the intersection. The interval where this decision have to be made for most people were found to be in the range of 2.5-5.5 seconds before entering the intersection [14]. Outside the interval the decision it typically quite clear. The reaction times of drivers is slowest in the center of the interval, where the decision is the most difficult. Common reasons for unintentional red light running are: distraction from e.g. conversing or manipulating infotainment equipment, speeding to make it through the intersection in time, and aggressive driving by closely following the car in front [15]. Figure 2 shows two scenarios where information from different sensors and intelligent systems can provide improved DAS.

Before delving into the research made in TLR, it is interesting to examine the state of related computer vision problems: traffic sign recognition, pedestrian detection, taillight detection, and headlight detection. When identifying challenges and solutions for TLR, the challenges, methods, and experience from published work in related computer vision problems can be helpful. The related computer vision problems often, include comparable challenges to those found in TLR. Traffic sign recognition and pedestrian detection are research topics that have been addressed extensively. For traffic sign recognition, [9] provides an overview up until 2012. Recently, the focus has shifted from heuristic model based detection to learning based approaches and the problem is considered solved on a subset of signs [16], [17]. Detection of traffic signs, pedestrians, and vehicles during varying lighting, viewpoints and weather conditions is challenging. Traffic signs usually consist of retroreflective materials [18], which make them visible at night by reflecting light from headlights. Retroreflective materials are difficult to capture well with cameras and many of the useful features may be lost, making detection and recognition of the sign impossible. Additionally, the perceived colors of objects change with the lighting and models must therefore tolerate large variations in color. All of these issues suggest that relying solely on color is problematic, therefore shape information is useful for sign detection. An example of the use of shape information is seen in [19]. Though TLs do not have the same retroreflective material problems, optimal camera settings are essential for all computer vision systems. A robust vision based DAS system that works under changing lighting, at varying distances, and under mixed weather conditions is a difficult task as stated in [20], where a concluding remark is to look into a cross-over procedures for handling environmental transitions. For both traffic signs and TLs, the angle between the ego-vehicle and the sign or TL will impact the perceived shape of the object, resulting in a new shape variation.

Determining the relevance of both TLs and traffic signs is a major challenge and the solution rely on incorporating information from other systems, such as lane detection and GPS. Most vehicle detection and brake light detection at night utilize monocular cameras and rely on the symmetry of tail and head lights for detecting vehicles as in [21], [22], [23], [24], [25], [26]. In [27] head and tail light detection is done using cues from lane detection, the purpose is to make a system that can automatically switch between high-beam to low-beam. Vehicle detection using tail and headlight detection is facing similar challenges as TLR with regards to false detections from atypical vehicles, billboards, and pedestrian crossing lights. A recent paper on lane detection is [28], where a context aware framework for lane detection is introduced, this can significantly reduce the required computational demand by scaling the detection algorithm based of the state of the ego-vehicle and the road context. The same paper references several comprehensive surveys of lane estimation techniques, one being [29], where work done across multiple modalities is reviewed. In [30], [31] the gaze and attention of the driver is determined. This is essential information

(a) (b)

Fig. 3: (a) San Diego, California. (b) Cincinnati, Ohio.

for DAS, since it can be used to determine if the driver should be notified as e.g. in [32] where the driver is alerted and safety systems are engaged if the driver is inattentive for a prolonged period of time. While research on sign recognition and pedestrian detection has mostly shifted from heuristic models to learning based detection, as [33] shows how learning based detectors using Integral Channel Features (ICF) or the even faster and slightly better Aggregated Channel Features (ACF) outperform other approaches. The elsewhere successful learning based approaches have not yet been thoroughly tested for TLR where the vast majority of published work, rely on manually specified color and shape models.

## IV. Traffic Lights: Conventions, Structure, Dynamics, and Challenges

Worldwide there are large variations in TL designs; however, all follow a few general guidelines. A TL consists of a box that holds differently colored, and sometimes differently shaped lamps. The orientation, color, size, and shape of the box will vary country to country and even city to city. An example of differently oriented and colored TLs within the USA is seen in Figure 3. As evident in Figure 3(a), there are two methods for mounting TLs, suspended and supported. The latter has proven the most difficult for existing TLR systems, this is discussed in subsection IV-A.

TLs are by design made to stand out and be easily visible by using bright uniformly colored lamps surrounded by a uniform, often dark box. The purpose of a TL is the same across the world, it must safely regulate the traffic flow, while warning drivers about the state of the intersection ahead. The most common TL configuration is the basic red-yellow-green signal, where each state indicates whether a driver should stop, be prepared to stop, or keep driving. A variety of other TLs have been created as a result of more complex intersections. Figure 4 shows some of the allowed vertical configurations of TLs in California.

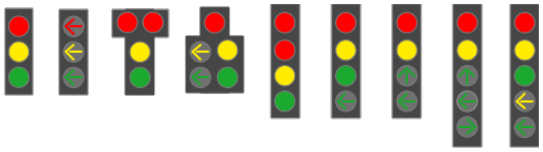


Fig. 4: Examples of vertical TLs found in California. [34]

For increasing road safety and making it easier for drivers when driving across states, the TLs in the USA are regulated by the Federal Highway Administration in the *Manual on Uniform Traffic Control Devices*[35]. Most countries in Europe have signed the *Vienna Convention on Road Signs and Signals* [36], requiring TLs to meet a common international standard.

Besides the various configurations of TLs, the state sequence is an important characteristics of a TL. An example of a state sequence for the basic red-yellow-green light is shown in Figure 5.

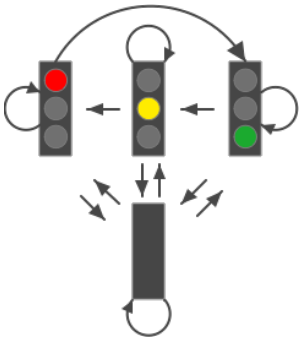


Fig. 5: Basic TL sequence for states: green, yellow, red, and no TL.

A TLR system for DAS must, in addition to detecting and recognizing TLs, be able to figure out which of the recognized TLs have relevance to the ego-vehicle. Figure 6 shows an example of a complex traffic scene where three upcoming intersections are all visible at the same time. One of the intersections contains turn lanes that are accompanied by their own independent TLs. Ideally, the TLR system should be able to incorporate detailed information about the specific TLs in it's evaluation of relevance. An example would be for the system to disregard TLs that only apply to buses or determine whether, at specific time slots during the day, intersections allow for turning right on red. To make this possible, and to enable TLR for use in DAS, the position and the planed route of the ego-vehicle must be taken into account.

Fig. 6: Complex traffic scene with three visible upcoming intersections and turn lanes, each with their associated TLs.

### A. Challenges in recognizing traffic lights

Although TLs are made to be easily recognizable, influences from the environment and sometimes sub-optimal placement

can make successful detection and recognition difficult, if not impossible. Issues include:

- Color tone shifting and halo disturbances because of influences from the atmosphere and glass that the light passes through[37]. Fig. 7(c).
- Occlusion and partial occlusion because of other objects or oblique viewing angles[37]. This is especially a problem with supported TLs [38], [39], [40]. Fig. 7(e),(f),(g).
- Incomplete shapes because of malfunctioning lights[37] or dirty lamps 7. Fig. 7(a),(b).
- False positives from, brake lights, reflections, billboards[41], [42], and pedestrian crossing lamps. Fig. 7(h).
- Changes in lighting due to adverse weather conditions and the positioning of the sun and other light sources. Fig. 7(d),(k),(l).
- Mismatch between camera's shutter speed and TL LED's duty cycle. Fig. 7(i),(j).

Inconsistencies in TL lamps can be caused by dirt, defects, or the relatively slow duty cycle of the LEDs. The duty cycle is high enough for the human eye not to notice that the lights are actually blinking. Issues arise when a camera uses fast shutter speeds, leading to some frames not contain a lit TL lamp. Saturation is another aspect that can influence the appearance of the lights. With transition between day and night, the camera parameters must be adjusted to let the optimal amount of light in and avoid under or over-saturation. [43] introduces an adaptive camera setting system, that change the shutter and gain settings based upon based on the luminosity of the pixels in the upper part of the image.

## V. Traffic Light Recognition for Driver Assistance Systems

Most computer vision problems can be divided into three sub problems, detection, classification, and tracking. The flow of a typical computer vision system, which handles each of the sub problems is illustrated in Figure 8. A similar breakdown is done for traffic sign recognition in [9].

The detection and classification stages are executed sequentially on each frame, whereas the tracking stage feeds back spatial and temporal information between frames. For TLR both the detection and classification stages are comparable to the equivalent stages in traffic sign recognition. Tracking of TLs differs, since signs are static and TLs change states, as it is illustrated in Figure 5. The detection problem is concerned with locating TL candidates. Classification is done based on features extracted from the detected candidates. Tracking uses temporal information about location and TL state when tracking TLs through a sequence of frames. A TLR system that addresses the mentioned problems can therefore be broken into 4 stages: detection, feature extraction, classification, and tracking.

In addition to recognizing TLs, a TLR system for DAS must communicate the gathered information to the driver, preferably in a way that is non-intrusive and adds as little as possible to the cognitive load of the driver. Information

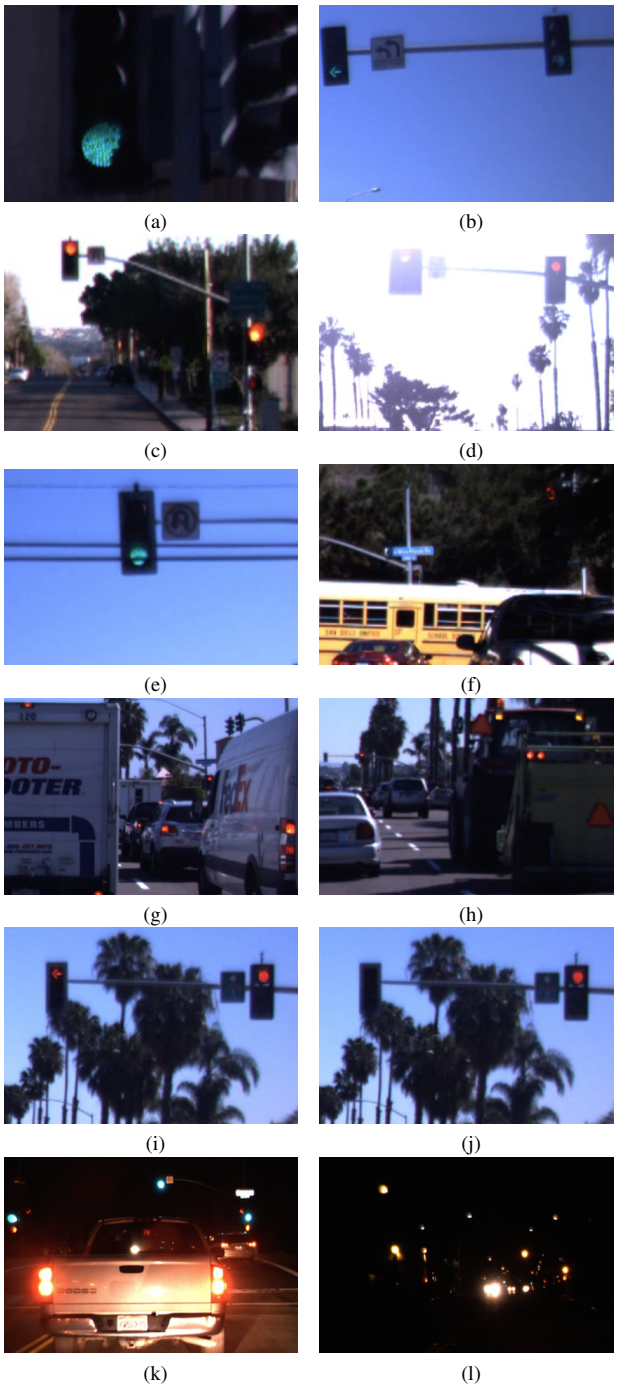



Fig. 7: (a) Examples of frames from the collected dataset.

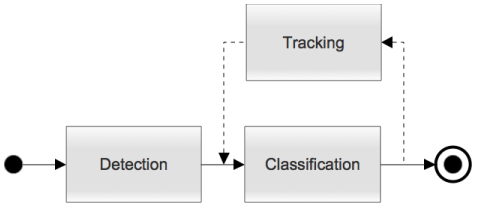



Fig. 8: A common breakdown of a DAS computer vision system.

about the driver's attention can be used to activate a given safety system in case driver is inattentive or to determine whether a driver has noticed a specific object and should be made aware of it. Hence, fusion of data from looking-in and looking-out sensors can be used [44]. In [45] a large set of looking-in activities: head pose estimation, hand and foot tracking; and looking-out activities: vehicle parameters, lane and road geometry analysis, and surround vehicle trajectories, are fused together to predict driver behavior. The presentation aspect of DAS is outside the scope of this paper.

A major challenge for TLR systems in relation to DAS is determining whether a TL is relevant to the ego-vehicle. The relevance of a TL is closely connected to it's placement. There might at one time be several TLs visible to the driver, each possibly in different light states. In such complex cases the assistance system needs to be able to determine which TL is relevant to the driver, a task that can be difficult, even to a human. For this to work, information about the location and direction of the ego-vehicle must be matched with the locations of the TLs. The most advanced system for solving this problem is seen in [41], where a guess is made based on the intersection width and the estimated orientation of the TLs. An alternative and less dynamic approach is used in [38], where the route is recorded beforehand and relevant TLs are manually annotated offline. Features are extracted in the annotated regions, and the system is then able to recognize the relevant TLs on that specific route.

## VI. Traffic Light Recognition: State-of-the-Art

In this section, we present an overview the approaches used for TLR in the surveyed work. TLR systems are divided into four stages; detection, feature extraction, classification, and tracking, as the breakdown was described in chapter V. In addition to the mentioned four stages, we break down papers based on applied color spaces, since some related work e.g. [43] emphasizes color space analysis and choices.

Table I shows an overview of recent and notable research in relation to TLR done by academic institutions. Table II show a similar overview for research introduced from the industry. It should be noted that some of the papers presents more than one approach, whereof only the best performing approach is listed. Since some of the papers focus on only parts of the problem, all fields are not consistently filled in, also in a few cases it is not apparent which exact methods were used. The paper overview covers papers from 2009-2014, with a single exception of an interesting 2004 paper [46] which forms the basis for the more recent paper [38].

### A. Color space

As color is essential for understanding TLs, it is used extensively in most recent work, usually for segmenting ROI and classifying the TL's state. A wide variety of color spaces are used, the RGB color space is often discussed, as it usually is the color space in which input images are represented. Because color and intensity information are mixed in the all the channels of the RGB color space, the information is usually converted to another color space for processing. [42], [11] are the only studies, where RGB is the primary color space. The same author group also utilizes the YCbCr color space in [47]. [48] uses both RGB and YCbCr, in two separate stages, RGB is used for localizing the box of the TL, whereas YCbCr is used for detecting the arrow TL. Normalized RGB is used both by it self, as in [49] and combined with RGB as in [43], [50], [51].

[52], [53], [40], [39] use grayscale for initial ROI segmentation in the form of spot light detection. The HSV and HSI color spaces are well represented by their use in [54], [55], [56], [52], [57], and [58], [59], respectively. [37] uses IHLS which is an improved HLS color model that separates chromatic and achromatic objects well. A few papers, namely [53], [40], [39], rely purely on grayscale, hence, their systems must function using only intensity and shape information. Other work that use grayscale, is [60], where normalized grayscale is used in addition to CIELab and HSV. [61] intially use grayscale for finding ROIs, inside the segmented ROIs they then use HSV to detect the state of the TL.

There is no clear tendency towards using one particular color space. Some recent work has begun combining channels from multiple color spaces, as seen in [60], [43] and to some degree [62]. In [63], [64], the CieLab color space is used to create a new channel by multiplying the Lightness with the sum of the a and b channels.

### B. Detection

TLR systems usually look for a selection of TL components. In Figure 9, components such as the colored lamps and various shapes are seen. The structural relationship between the components is also used.

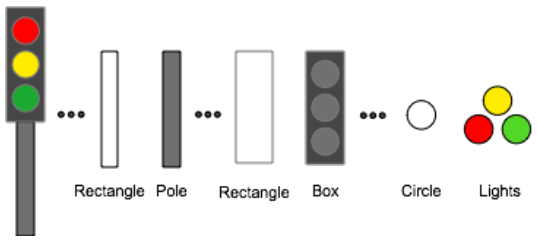



Fig. 9: A supported TL and the different components that TLR systems search for in a basic TL.

A simple and widely used approach to detection is using predefined color density thresholds for each TL state color, this is done in [42], [47], [51], [55], [64], [52], [49], [41], [57], [59], [37], [11]. As mentioned in the color space subsection, a new tendency combining several channels from different color spaces to provide additional information. In [43], [50], fuzzy clustering is introduced to generate unique representations of a given color. Opposite to regular clustering, sometimes called hard clustering, data points in fuzzy clustering can belong to more than one cluster; the association degree to clusters can vary, meaning that a data point can have a higher

probability of belonging to one cluster. This approach provides a different way of grouping data points [65]. A promising detection approach is spotlight detection using the white top hat operation, this is done on grey scale image in [52], [53], [40], [39]. In [54], the V channel from the HSV color space is used instead. In [38], [46], various detection approaches are presented. The best performing a trained Gaussian-distribution pixel classifier, created based on several thousands manually annotated color images. Other approaches include finding circles using generalized Hough transform on Sobel based gradient directions and matched filter detection based on template matching. Since the last two methods cannot by themselves determine the color of the light, a gray value detector is used to determine which light is turned on. Lastly, [38], [46] also test a cascading classifier based on Haar features, which outperform their other approaches, except for the Gaussian color classifier. A popular approach to improving the detection, can be seen in [38], [46], [56], [41], [54], where an off-line database containing prior knowledge of TL locations is used. This is done using accurate GPS measurements and manual annotation of areas with TLs in pre-captured image sequences. [11] only use the GPS coordinates to activate the detection system when approaching a preannotated intersections. In [56] they store hue and saturation histograms of each TL during the initial TL mapping. This helps with handling differences in the light emitting properties of individual TLs. In [41] they, also annotate the possible states of the individual lights to further reduce false positives. Preliminary segmentation leaves BLOB candidates. This is followed by interpretation or further segmentation in the form of filtering. Most recent work applies BLOB analysis in various degrees to either remove noise or calculate BLOB properties. [53], [40], [42], [39], [47], [48], [52], [58], [59] removes noise by looking at a selection of BLOB features, from relative position of elements such as circles, squares, rectangles, spots, containers, to size, aspect ratio, shape, etc.. An example is [52] where they look at the BLOBs' size, aspect ratio, circular shape, holed regions, and the estimated height in world coordinates. [59] employs region growing using seed points from their found BLOBs to perform a border property check between found BLOBs and their corresponding grown regions. Other BLOB analysis includes doing bounding box analysis as in [43], [50], [51], [54], [55], where the goal is to locate the TL box such that the state within it can be estimated. Alternatively to finding the shape from BLOBs, [49], [61] applies a Sobel Kernel to get an edge map and applies Hough transform in order to find either circular shapes or boxes. Hough transform is also used in [11] on an edge map generating using a Laplace edge detection filter.

The contribution of [49] is a modified version of the circular Hough transform, which looks for filled circles, and outperforms the conventional circular Hough transform in locating TL lamps. In [62] they improve this idea further by also looking for circles around active lights. [37] first apply a Laplacian filter that extracts a clear boundary, disregarding halo effects, before looking for approximate ellipses in the canny edge pixels around candidate BLOBs. Their approximate ellipse detection is the main contribution. In [64], [63], fast radial symmetry is used for finding circles, followed by local maximum and minimum for finding the exact parameters of the given circle. [61] finds object boundaries using morphological operations and thresholding, from these borders they use topological analysis to locate rectangles that represent TL candidates.

Generally, the first step of segmentation is to isolate ROI by using either, clustering, distributions, or thresholds. This is followed by either looking for circular objects using Hough transform or fast radial symmetry, or BLOB analysis to isolate candidate TLs. Furthermore, using prior knowledge of the route, geographical and temporal information can drastically reduce the ROI, hence reduce the computational requirements and the number of bad candidates.

### *C. Feature Extraction*

A common feature for classification is color as seen in [43], [50], [51], [38], [46], [61], [54], [60], [41], [58], [49], [62], where the color densities from segmented areas are used. In [59] color from the HSI space is used for template matching. In [56], [52] the features are based on HSV histograms. Besides color, other common features are the shape and structural information from the TL. Shape information includes a wide variety of features, such as aspect ratio, size, and area. Structural information requires some prior knowledge of the shapes that constitute the entire TL, both the case container and their belonging lamps. The structural information is the relatives position between the circles, rectangles, spots, and containers that constitute the entire TL. A TL lamp and the surrounding container is, in many cases, easily distinguishable from the background, making shape, and structural information popular features. Shape information is both combined with structural information in [53], [40], [39], but also color features as in [43], [50], [51], [60], [49], [62]. In [54], [11], color, shape, and structural information are used as features. In [42], a mix of BLOB width, height, center coordinate, area, extent, sum of pixels, brightness moment, and geometric moment are used as features for their SVM classifier.

More advanced feature descriptors are seen in [55], where edge information in the form of Histogram of Oriented Gradients (HoG), in nine directions, are used as features for image regions containing TL containers. [48] does the same but uses 2D Gabor Wavelets features instead of HoG and [57], [47] uses Haar features. [37] relies on spatial texture layout for classification, specifically they calculate a Local Binary Pattern (LBP) histogram for the entire TL as well as for five equally sized regions in each color channel, before creating a feature vector consisting of the concatenated LBP histograms.

The majority of papers employ empirically determined models for TL color, shape, and structural information for finding TL candidates and estimating their states. A few papers use extracted features based on image transforms and classifying the resulting feature vectors against a database of learned models.

### D. Classification

As described in the features subsection, much of the recent work uses color, shape, and structural information in the segmented areas as features for determining the state of TL candidates. [54] utilize a fusion between scores from structure, shape, color, and geolocation information to determine whether a TL should exist at the found position. [61] simply estimate the state to be the winner of a majority count on the number of pixels within empirically determined thresholds. [58] decides on a TL state for the entire segmented frame based on a contradiction scheme that selects the optimal light from TL position and size.

In [38], [46], a neural network is used for determining the state of the found TLs. [59] applies template matching by normalized cross correlation. [40], [39], [53] use adaptive template matching. [52] uses SVM for classification based on HSV histograms. [40] compares their proposed adaptive template matching system with a learning based system, which uses a cascade classifier trained with AdaBoost on Haar features. Their model based approach proved to substantially outperform their learning based approach. [57] also uses an AdaBoost trained classifier based on Haar features. [37] applies SVM to classify LBP feature vectors in order to determine the state of a TL from it's spatial texture layout. [47] tries two learning based approach, with cascading classifiers using Haar features and a model based approach which uses Haar-like features for determining the state for the found TLs. Also here, the learning based approaches lose.

The majority of papers apply a classifier on extracted features and finds the best match by comparison with a number of trained states. A smaller, but still significant, amount of papers detect state by matching templates.

### E. Tracking

As evident in Table I and II, 13 out of 27 papers make use of some form of tracking. Tracking is commonly used for noise reduction by suppressing false positives and handling dropout due to e.g. occlusion. The correlation tracking used in [54], [38], [46], relies on the fact that a given detected TL's state is unlikely to shift sporadically in a sequence of frames. E.g., when a series of red states is detected, it is most likely that the next state in the upcoming frame will be red again. Similarly, if the state has changed and is green for a series of frames, it is likely that the state has in fact changed to green.

Temporal tracking is used to examine previous frames and determine whether a candidate has been found in the same area earlier and whether it may have the same properties as a given candidate in the current frame. This is a simple and straightforward approach used in [50], [51]. To reduce computational load CAMSHIFT is used In [57] to track a given candidate across frames based on its appearance. [52] employs multiple target temporal tracking using predictions of the location of TL from the speed of the ego vehicle. This would allow for validation of state changes and smoothed recognition confidence. Additionally, they modify top hat kernel size, saturation, and intensity thresholds when TLs are about to disappear from the FOV, in order to enable recognition for as long as possible. [58] propose a temporal decision scheme that makes a final classification based on the temporal consistency of their classification. This proved to reduce wrong detections by a third.

Before reaching the final state verdict, [52] inputs the detected state from their classifier into a range of HMMs, one for each possible type of TL and one for a non-TL objects. The model which best fits the detected sequence of states is then selected as the final estimated state. [61] also employs HMM, although only for a single TL type. [43] estimates the distance to TLs using inverse perspective mapping and tests both a Kalman filter and a particle filter for tracking the relative movement between vehicle and TL. Tls are then filtered based on their consistency in position and color. In [53], an Interacting Multiple Model filter is used for keeping track of both state and position of a given TL. The prediction in the model is using Kalman filters to keep track of the state and the position in time. For establishing the state, a Markov chain with weighted probabilities is used for finding the current state based on posterior states, originally introduced in [66]. [56] uses prior maps and a histogram filter to adjust the localization mismatch between predict and actual TL area.

Tracking is mostly used to filter out noise and handle lone failed detections, caused e.g. by occlusion. In most of the surveyed papers, this is done by a simple temporal consistency check. A few of the papers use tracking in a more advanced manner by incorporating prior probabilities. A single paper use tracking to locate previous detections in future frames.

## VII. Evaluation

In the reviewed work on TLR, a wide variety of approaches are used to evaluate performance of the proposed systems. Often, the exact evaluation criteria are not clearly described, making a direct comparison impossible. In addition, the data the proposed systems are evaluated on, is usually a local collection gathered by the authors themselves and not a publicly available dataset. The local datasets are often small in size and contain little variation in environmental conditions. Challenges similar to the *VIVA Challenge* [67] or *The KITTI Vision Benchmark Suite* [68], are not present for TLR.

### A. System evaluation

In the traffic sign detection survey presented in [9], an overview of the results from all the surveyed traffic sign detection papers are summarized with the number of positives and negatives in the evaluation datasets and four measures, namely: Best detection rate, false positives for best detection, mean detection rate, and mean false positives rates. However, the recent and noteworthy papers reviewed in this paper are frequently using the three measures: precision, recall, and accuracy. For summarizing the results with regards to TLR, these three measures are selected and defined in equation (1), (2) and (3). TP, FP, FN are abbreviations for true positives, false positives and false negatives. Table III shows the results and the specifications of the evaluation data from the recent

TABLE I: recent Academic studies in TLR. Colors indicate paper group affiliation. Corresponding evaluation results and datasets for each paper are available in Table III. Abbreviations: Connected component analysis (CCA), support vector machine (SVM), hidden Markov model (HMM)

| Paper | Year | Color Space(s) | Detection | Features | Classification | Tracking |
|---|---|---|---|---|---|---|
| [43] | 2014 | RGB, RGB-N | Fuzzy clustering, CCA, BLOB analysis | Color, shape | - | Kalman filter, particle filter |
| [50] | 2013 | RGB, RGB-N | Fuzzy clustering, CCA, BLOB analysis | Color, shape | - | Temporal filtering |
| [51] | 2012 | RGB, RGB-N | Clustering, BLOB analysis | Color, shape | - | Temporal filtering |
| [53] | 2014 | Grayscale | Top-hat spot light detection, BLOB analysis | Shape, structure | Adaptive template matching | Interacting Multiple Model |
| [40] | 2009 | Grayscale | Top-hat spot light detection, BLOB analysis | Shape, structure | Adaptive template matching | - |
| [39] | 2009 | Grayscale | Top-hat spot light detection, BLOB analysis | Shape, structure | Adaptive template matching | - |
| [42] | 2013 | RGB | Color thresholding, BLOB shape filtering | BLOB features, brightness moments, geometric moments | SVM | - |
| [47] | 2011 | YCbCr | Color thresholding, BLOB shape filtering(width to height ratio, sum of pixels, BLOB area to bounding rectangle ratio) | Haar-like features | Adaptive multi-class classifier trained using AdaBoost | - |
| [62] | 2010 | RGB, RGB-N | Pixel clustering in RGB/RGB-N, edge map with Sobel kernel, modified(filled) circle Hough transform for neighborhood | Color, shape | Color of best circle | - |
| [49] | 2009 | RGB-N | Color thresholding, edge map with Sobel kernel, modified(filled) circle Hough transform | Color, shape | Color of best circle | - |
| [61] | 2014 | Grayscale, HSV | Rectangles from topological analysis of edges | Color | Majority pixel count | HMM |
| [54] | 2014 | HSV | Top-hat spot light detection, BLOB analysis | Color, shape, structure | Fusion of color, shape, structure scores, and geolocation information | Correlation tracking |
| [55] | 2014 | HSV | Color thresholding, BLOB analysis | HoG | SVM | - |
| [60] | 2014 | Norm. grayscale, CIELab, HSV | Prior knowledge of TL location | Color, shape | Convolutional neural network | - |
| [64] | 2014 | CIELab | Color thresholding, radial symmetry, local maximum and minimum, shape filtering | - | - | - |
| [48] | 2012 | RGB, YCbCr | BLOB analysis | 2D Gabor wavelet | Nearest neighbor | - |
| [63] | 2012 | CIELab | Color difference enhancement, neighborhood image filling, radial Symmetry | Color | - | - |
| [11] | 2012 | RGB | Color thresholding, edge map with Laplace filter, circle Hough transform, shape filtering | Color, shape, structure | Color | - |
| [59] | 2011 | HSI | Color thresholding, dimensionality and border property check | Color | Normalized cross correlation template matching | - |
| [37] | 2011 | IHLS | Color thresholding, Laplacian filter for boundary extraction, approximate ellipses based on edge pixels from canny | LBP features | SVM | - |
| [56] | 2011 | HSV | Prior knowledge of traffic light location | HS histograms | Histogram back-projection scores | Histogram filter |
| [52] | 2010 | HSV | Top-hat spot light detection, Color thresholding, CCA, BLOB analysis | Concatenated HSV histogram | SVM | HMM and temporal tracking using ego motion |
| [57] | 2010 | HSV | Color thresholding, morphological operation | Haar features | AdaBoost trained classifier | CAMSHIFT |
| [58] | 2009 | HSI | Gaussian-distributed classifier, BLOB analysis, temporal information | Color | Global contradiction solving scheme | Temporal filtering/decision scheme |

TABLE II: Recent studies in TLR from industry. Colors indicate paper group affiliation. Corresponding evaluation results and datasets for each paper are available in Table IV.

| Paper | Year | Color Space(s) | Detection | Features | Classification | Tracking |
|---|---|---|---|---|---|---|
| [38] | 2013 | - | Prior knowledge of TL location, Gaussian-distribution classifier | Color | Neural network | Correlation tracking |
| [46] | 2004 | - | Prior knowledge of TL location, Gaussian-distribution classifier | Color | Neural network | Correlation tracking |
| [41] | 2011 | - | Prior knowledge of TL location and state sequence | Color, shape | Color and BLOB geometry | Temporal filtering |

noteworthy TLR papers from academic institutions. Table IV show similar results and specifications from noteworthy research contributed from the industry.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

A Precision close to one indicates that all the recognized TL states are in fact correctly recognized.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

A recall close to one indicates that all the TL states, in a given video sequence, were correctly recognized by the proposed system.

$$Accuracy = \frac{TP}{TP + FN + FP} \tag{3}$$

The recognition rate, or accuracy, is the overall recognition rate taking true positives, false negatives, and false positives into consideration. An accuracy close to one indicates that the system detects all TLs with no false positives. Traditionally, true negatives are also included in the calculated accuracy as follows from equation (3), but true negatives are rarely used in evaluations of TLR systems.

Since papers have different goals, the proposed systems can be evaluated very differently. In [38] the presented system includes lane recognition, localization, motion estimation, pedestrian recognition, vehicle detection, and TLR. The system was evaluated on the 100 km long *Bertha Benz Memorial Route*, under real traffic conditions. Their conclusion states that the presented system drove the route fully autonomously. As impressive as this is, they do not disclose any quantitative measure of the TLR system's performance.

Direct comparison of the results between all of the surveyed papers is not feasible, due to the differences in evaluation methodology and testing data. One example of this is papers such as [38], where the scope is a complete autonomous system that must complete a specific challenge.Other examples is that many papers do not clearly define detection rate, recognition rate, etc., when reporting results. Generally, the detection rate is mentioned as the term describing the ratio between correct TL recognitions and the ground truth in a given video sequence. This term is equal to the recall term as seen in equation (2). The recognition rate is considered the same as overall accuracy as seen in equation (3). When this term is used without a clear definition, the recognition rate results are published as accuracy in Table III.

Besides the problems with lacking definition of terminology, different criteria are sometimes used for deciding when a TL recognized and registered as a true positive. An example of this is seen in [40], [39], where a TL is noted as a true positive if a TL is recognized once in the series of frames where it appears. Hence, for a false negative to be noted a TL must not be detected once in the entire video sequence. This aspect is an important notation, as such differences in evaluation practice leads to skewed results, since it is a much harder task to recognize a TL in every frame it appears in. For future evaluations, we suggest evaluating FPs, TPs on a frame by frame basis, as this gives a more complete representation of a given system's performance.

Table III indicates that many of the surveyed systems performs in the high 90 % in recall, precision, and accuracy. Some of the best performing systems rely on high detail maps of TL position and type. In addition to significantly easing the task of detecting TLs, prior maps can be matched with an exact route. As long as the ego-vehicle follows this route, the problem of determining which TLs are relevant to the ego-vehicle can be disregarded. In relation to using such systems for DAS in consumer vehicles, an overwhelming amount of data needs to be collected, maintained, and distributed.

The surveyed papers are in most cases evaluated on data captured by themselves, hence the data cannot be directly compared. The conditions of the captured data are in most cases poorly described which is also greatly visible in Table III where some just present they used data captured during the day. Furthermore, the table show that some datasets contain as little as 35 TL, which do not constitute a valid basis for evaluating a system for DAS. Nearly half of the papers that have defined their dataset specifications are using a dataset with under 3,000 positives. The current best performing is therefore hard to conclude as the exact conditions of all the dataset is not known. [43] is the best system based on dataset size, ground truth, and conditions information, but also as the system are utilizing adaptive image acquisition and tracking. Another notable system is presented in [38], where a autonomous driving car using TLR has driven 100 km fully automatically in public traffic. The performance of only the TLR system is not published making it impossible to directly compare it to other work.

### B. Future evaluation

A variety of ways can be used to evaluate TLR systems, examples are recognition rate, detection rate, recall, precision, true positive rate, false positive rate, false positives per frame, confusion matrix, F1-score, etc.. When evaluating a system, it is important to define how a given term is mathematically defined, preventing any doubt of what is meant with recognition rate, detection rate, and accuracy. By doing so, the comparability with others work becomes easier. Furthermore, specific details about the dataset a system was evaluated on are important for making a fair assessment possible. A very important term to define clearly is what a true positive is. We suggest using PASCAL overlap criterion introduced in [69], which define a true positive as a detection where the detected bounding box area cover atleast 50 % of the annotated ground truth bounding box area, which is also seen in equation (4).

$$a_0 = \frac{\text{area}(B_d \cap B_{gt})}{\text{area}(B_d \cup B_{gt})} \geq 0.5 \tag{4}$$

Where $a_0$ denotes the *overlap ratio* between the detected bounding box $B_d$ and the ground truth bounding box $B_{gt}$. $B_d \cap B_{gt}$ denotes the intersection of the detected and ground truth bounding boxes, and $B_d \cup B_{gt}$ denotes their union. The PASCAL overlap criterion is also illustrated in Figure 10, where the red detection area do not satisfy the requirement of an overlapping ratio of 50 %. This requirement is however satisfied with the green detection area.

TABLE III: Evaluation datasets and corresponding result of recent academic studies in TLR. Colors indicate paper group affiliation. The abbreviation RCMP indicates evaluation on the *Robotics Centre of Mines ParisTech* dataset, RCMP- indicates evaluation on part of the dataset and RCMP+ indicates that additional private datasets was used for evaluation. For *Ground Truth*, # of frames indicates # of frames with a minimum of 1 TL in it. The terms *Recognition Rate* and *Detection Rate* are considered equivalent to Accuracy and Recall, respectively.

| Paper | Year | Dataset | Size [Frames] | Ground Truth | Resolution [Pixels] | Conditions | Pr [%] | Re [%] | Ac [%] |
|---|---|---|---|---|---|---|---|---|---|
| [43] | 2014 | Local | 75,258 | 19,083 frames | 752x480 | Day, night | 99.38 | 98.24 | 99.39 |
| [50] | 2013 | Local | 16,176 | 4,600 frames | 752x480 | Night | 98.04 | - | - |
| [51] | 2012 | Local | 27,000 | 14,000 frames | 752x480 | Day, night | 90.32 | - | - |
| [53] | 2014 | Local | - | - | - | - | 97.6 | 87.57 | 97.6 |
| [40] | 2009 | RCMP+ | >11,179 | 10,339 TLs | 640x480 | Day | 84.5 | 53.5 | - |
| [39] | 2009 | RCMP+ | >11,179 | >9,168 TLs | 640x480 | Day | 95.38 | 98.41 | - |
| [42] | 2013 | Local | 16,080 | 12,703 frames | 620x480 | Night | - | 93.53 | - |
| [47] | 2011 | Local | 30,540 | 16,561 frames | 620x480 | - | - | 93.80 | - |
| [62] | 2010 | Local | 35 | 35 TLs | - | - | - | - | 89 |
| [49] | 2009 | Local | 30 | 30 TLs | - | - | - | - | 86.67 |
| [61] | 2014 | Local | 649 | 446 TLs | 648x488 | Multiple light conditions | 99.59 | 92.19 | 94.45 |
| [54] | 2014 | Local | 3,767 | - | - | Good, challenging, very challenging | - | - | 96.07 |
| [55] | 2014 | Local | - | - | 640x480 | Multiple light conditions and cluttered backgrounds | - | - | - |
| [60] | 2014 | Local | 3,351 | 3,351 TLs | - | Afternoon, dusk | - | - | 97.83 |
| [64] | 2014 | Local | 70 | 142 TLs | 240x320 | - | 84.93 | 87.32 | - |
| [48] | 2012 | Local | 5,000 | - | 1392x1040 | Direct sunlight, backlighting, cloudy and sunny. | - | - | 91.00 |
| [63] | 2012 | RCMP | 11,179 | 9,168 TLs | 640x480 | Day | 61.22 | 93.75 | - |
| [11] | 2012 | Local | 7,311 | - | - | Day | - | 89.9 | - |
| [59] | 2011 | RCMP- | 5,553 | 5,553 frames | 640x480 | - | 96.95 | 94.4 | - |
| [37] | 2011 | Local | 714 | 763 TLs | 640x480 | Sunny, cloudy, rainy | 34.51 | 94.63 | 95.01 |
| [56] | 2011 | Local | - | - | 1280x1024 | Noon, dusk, night | 81.46 | 77.98 | 92.85 |
| [52] | 2010 | Local | - | 2,867 TLs | 512x384 | - | - | - | 89.6 |
| [57] | 2010 | Local | - | - | 780x580 | - | - | - | - |
| [58] | 2009 | Local | 6,630 | - | 640x480 | - | - | - | 98.81 |

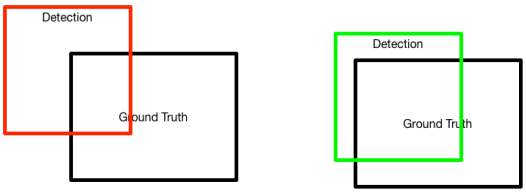



Fig. 10: PASCAL overlap criterion

For visualizing the results, a confusion matrix provide a quick overview of the overall performance of a system, but also the performance for specific object classes, in this case red, yellow, and green TLs. An example of a confusion matrix is seen in Table V, dark grey indicates the variables, and brighter grey indicate either ground truth or system classification output. By using these numbers, recall, precision, and accuracy can be calculated as seen in the blue fields.

In [70], it is stated that using the overall accuracy on an unevenly distributed dataset is misleading. The overall accuracy does not explain much about the smaller classes in

TABLE V: Confusion matrix for evaluation of a skewed dataset.

| | | | System classification | | | |
|---|---|---|---|---|---|---|
| | | | Red | Yellow | Green | Recall |
| | | | 700 | 17 | 170 | |
| Ground Truth | Red | 700 | 100% | 0% | 0% | 100% |
| | Yellow | 17 | 0% | 100% | 0% | 100% |
| | Green | 170 | 0% | 0% | 100% | 100% |
| | Precision | | 100% | 100% | 100% | 100% |

the dataset. Because of the variation in TLs, which is discussed in section IV, it is unlikely that any collected dataset will have an even distribution between the classes of TLs. The standard red and green TLs would have many more occurrences in a dataset compared to yellow and all of the arrow lights. To illustrate the problem, an example would be a system that reaches an overall accuracy of 85% on the two biggest classes,

TABLE IV: Evaluation datasets and corresponding result from recent industrial studies in TLR. Colors indicate paper group affiliation. For *Ground Truth*, # of frames indicates # of frames with a minimum of 1 TL in it. The terms *Recognition Rate* and *Detection Rate* are considered equivalent to Accuracy and Recall, respectively.

| Paper | Year | Dataset | Size [Frames] | Ground Truth | Resolution [Pixels] | Conditions | Pr [%] | Re [%] | Ac [%] |
|---|---|---|---|---|---|---|---|---|---|
| [38] | 2013 | Local | - | - | - | 100 km in real world | - | - | - |
| [46] | 2004 | Local | - | - | - | - | - | >95 | - |
| [41] | 2011 | Local | - | 1,383 frames | 2040x1080 | Morning, afternoon, night | 99.65 | 61.94 | 93.63 |

and the remaining 15% are spread out on several minor and less frequent classes, such as the arrow variations. In such a case, a system that can reach high recall and precision rates on major classes, will also reach a high accuracy even though the system might is not able to recognize the less frequent classes. A result indicating this is highly problematic as a FP in smaller classes, such as red arrow left, might cause critical and deadly accidents. In [70] various methods for coping with skewed datasets are presented. These are precision, recall, F-measure, *Receiver Operator Characteristic* (ROC) curve, cost curve, and *Precision-Recall* (PR) curve, which can be used to interpret if a given system has problems with less frequent classes. In [71], the relationship between ROC curves and PR curves is discussed. The common and widely used ROC curve is used for publishing results from a given system with a binary, *positive or negative*, decision approach. A ROC curve is generated by plotting the *True Positive Rate* (TPR) as function of the *False Positive Rate* (FPR), where the TPR is equivalent to recall. The FPR indicates the ratio of false classifications that are mistakenly registered as a TL. The TPR and FPR definitions are seen in equation (5) and (6).

$$\text{True Positive Rate} = \frac{TP}{TP + FN} \tag{5}$$

$$\text{False Positive Rate} = \frac{FP}{FP + TN} \tag{6}$$

The PR curve is generated by using precision and recall defined in equation (1) and (2). Unlike with ROC curves, PR curves do not use true negatives, as it simply plots the precision as a function to the recall. The following is a simple example used for comparing ROC and PR curves. Consider two different approaches for recognizing TL, both approaches are evaluated on the same video sequence of 10,000 frames, which contains a total of 2000 frames with one visible TL in each frame providing a ground truth of 2000. Approach 1 recognizes 2000 TLs in the sequence whereof 1900 are correctly recognized. Approach 2 recognizes 2500 TLs in the sequence whereof 1900 are correct. In Table VI and VII the corresponding results from each results are seen.

TABLE VI: Results from approach 1.

| | |
|---|---|
| TP: 1,900 | FN: 100 |
| FP: 100 | TN: 7900 |

TABLE VII: Results from approach 2.

| | |
|---|---|
| TP: 1,900 | FN: 100 |
| FP: 600 | TN: 7400 |

From the information above the TPR and FPR for ROC, and the precision and recall for PR are calculated:

| ***Approach 1*** | | ***Approach 2*** | |
|---|---|---|---|
| TPR | 0.95 | TPR | 0.95 |
| FPR | **0.0125** | FPR | **0.075** |
| Recall | 0.95 | Recall | 0.95 |
| Precision | **0.95** | Precision | **0.76** |

By comparing the results marked with bold from above approaches, it is clear that the difference of 0.0625 using FPR is quite small, compared to a difference of 0.19 using precision.The increased number of false positives has a higher impact on the PR curve than on the FPR used in ROC curves. For evaluation of systems where the amount of true negative is either not available or not interesting, the PR curves should be used, otherwise ROC. For a evaluation of TLR the true negative would in most cases not be present nor interesting. This will change if considering a system that recognize all TLs, and needs to determine which are of interest to the driver. The number of true negatives will be the TLs that are rejected as being of interest to the user. For a final results, both [71] and [70] mention the *Area-Under-Curve* (AUC) as an alternative to the traditional accuracy measure. The AUC can be used as a final measure for both ROC and PR curves, where a higher number indicates a better performance. Using AUC for measuring the accuracy of a given system gives an insight into the ability for recognizing TLs. AN AUC of 100% would indicate the system works perfectly. The results should also include at what pixel area each TL was first detected. Such a metric is essential for TLR to be useful for DAS, as it must be able to detect and recognize TLs at a proper distance, providing time to react upon the information. The pixel area is calculated according to equation (7).

$$\text{Pixel area} = \text{area}(B_d \cap B_{gt}) \tag{7}$$

For future publications, we suggest evaluating the proposed TLR systems with DAS in mind. Therefore we propose that DAS related TLR systems should be evaluated as on a frame-to-frame basis instead TL over the visible timeline. Furthermore, a proposed system accuracy should be calculated using AUC on a PR curve. The proposed evaluation terms are listed below:

- True positives are defined according to the PASCAL overlap criterion.
- Precision, as seen in equation (1)
- Recall, as seen in equation (2)
- Area-Under-Curve on Precision-Recall curves
- Confusion matrix.

- Pixel area at which the true positive is first recognized, calculated as seen in equation (7).

## VIII. Traffic Light Database

As it was concluded in the traffic sign survey paper [9], the general approach for testing and validating a proposed method is use a privately collected dataset. This is considered sufficient for preliminary testing and validation. But, when trying to estimate the contribution made in a paper, it becomes difficult to compare the work with others'.

The only publicly available dataset is provided by *Robotics Centre of Mines ParisTech* in France. The dataset consists of 11,179 frames from a single 8m 49s long video. It contains 9,168 hand-labeled instances of TLs. More information about this dataset can be found in Table VIII. An alternative to using either of the two TL databases found in Table VIII, could be to use existing traffic sign databases, but as the name suggests, these are focused on traffic signs rather than TLs. In [9] it is stated that as of 2012, at least six different traffic sign databases were publicly available, of which two contain video recordings.

In [73] an overview of vehicle detection systems based on both monocular and stereo vision since 2005 is provided. Both monocular and stereo vision are widely used for solving this problem, but an interesting finding in this work is the stereo vision bottom-up paradigm which consist of visual odometry, feature points in 3D, and distinguishing static from moving points, which is also mentioned in [74].The motivation for having a public TL database with stereo images is therefore high as all three parts in this paradigm can help reduce the amount of false positives. This notion is reinforced in [75] where the main technical challenges in urban environments are occlusions, shadow silhouettes, and dense traffic. The introduction of stereo has shown promising result in relation to solving these challenges.

The database that is collected and released together with this paper is focused on TLR and contains TLs that are found in San Diego, California, USA. The database provides four day and two nighttime sequences for testing. These test sequences 23 minutes and 9 seconds of driving in Pacific Beach and La Jolla, San Diego. The stereo image pairs are acquired using the Point Grey's Bumblebee XB3 (BBX3-13S2C-60) which contains three lenses which capture images with a resolution of 1280 x 960, each with a Field of View(FoV) of 60°.The stereo camera supports two different baselines, 12 and 24 cm, whereof a baseline of 24 cm is used for the LISA TL database. The stereo images are uncompressed and rectified on the fly. Bumblebee XB3 is mounted in the center of the roof of the capturing vehicle which is connected to a laptop through the interface standard IEEE-1394b, sometimes refereed to as FireWire 800.

Besides the five test sequences, we provide twelve shorter video sequences consisting of TLs collected in the northern part of San Diego. These are intended for training and additional testing. They are organized as seen in Table IX, which gives a detailed overview of all the video sequences that are made available with this paper. The gain and shutter speed were manually set to avoid over saturation as well as limit flickering of the TLs. For all day clips, a shutter speed of 1/5000 sec and a gain of 0 are used. For all night clips, a shutter speed of 1/100 sec and a gain of 8 are used. A Triclops calibration file is provided along with the stereo images, this file contains the factory calibration of the used Bumblebee XB3 camera, for use with the Point Grey's Triclops SDK.

Each sequence in the database comes with hand labeled annotations for the left frame. Annotations for a given video sequence contain the following information: frame number, rectangular area around the lit TL lamp, and it's state. In [38], ground truths are labeled down to 2x2 pixels following that a vehicle approaching a TL with 70 km/h should time to maneuver 80 meters before reaching the TL. Their system should therefore detection TLs as far as 100 meters, corresponding to 2x2 pixels with the specified camera setup. Similar to [38], we suggest recognizing TLs at 80 meter, based on the an approach at 50 km/h. Given that time should be allowed for detecting TLs for 5 consecutive frames with the camera capturing at 16 FPS, the first detection of TLs should be at 85 meters. This corresponds to detecting TLs of sizes down to 4x4 pixels. Labeling is therefore done for every visible TL lamp if it meets this size criteria. An example of annotated TL lamps is seen in Figure 11.

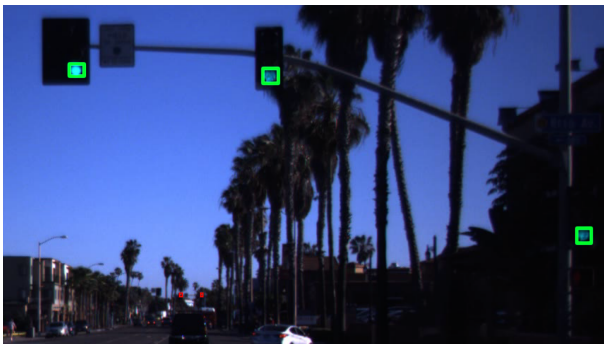


Fig. 11: Green rectangles illustrate annotated green TLs, red rectangles illustrate annotated red TLs.

The LISA Traffic Light Database is made freely available at http://cvrr.ucsd.edu/LISA/datasets.html for educational, research, and non-profit purposes.

## IX. Discussion and Perspectives

In this section we discuss the current trends and perspectives based on the surveyed work on TLR.

The current state of TLR is difficult to determine as evaluation is done on local datasets and with different evaluation methodology. For maintaining and advancing research on TLR, the introduction and use of public TL datasets and common evaluation methodology is essential. This will enable newcomers and established research groups in both the academic and industrial domain to efficiently compare new approaches to already published work. Until now, it required substantial effort to gain an overview of the state of TLR research, since no survey existed. The scope of current work vary significantly, spanning from simple TL detection in the

TABLE VIII: Overview of current public TL databases. Ambiguous means that it could not be decided whether the light was a TL during annotation, these are ignored when evaluating.

| | ***Robotics Centre of Mines ParisTech*[72]** | ***LISA Traffic Light Database*** |
|---|---|---|
| #Classes | 4 (green, orange, red & ambiguous) | 7 (go, go forward, go left, warning, warning left, stop, & stop left) |
| #Frames / #Annotations | 11,179 / 9,168 | 46,416 / 89,388 |
| Image resolution | 640 x 480 8-bit RGB | 1280 x 960 8-bit RGB |
| Stereo images | No | Yes |
| Place of origin | Paris, France | San Diego, USA |
| Video included | Yes 8min 49s. @25FPS | Yes 44min 24s. @16FPS |
| Description | 1 urban day time sequence | 4 test sequences $\geq$ 5min and 18 clips $\leq$ 2min 49s, morning, evening, night |

TABLE IX: Overview of the video sequences in LISA Traffic Light Database.

| **Sequence name** | **Description** | **# Frames** | **# Annotations** | **# TLs** | **Length** | **Classes** |
|---|---|---|---|---|---|---|
| Day seq. 1 | morning, urban, backlight | 4,800 | 10,267 | 25 | 5min | Go, warning, warning left, stop, stop left |
| Day seq. 2 | evening, urban | 9,586 | 11,154 | 29 | 6min 10s | Go, go forward, go left, warning, stop, stop left |
| Night seq. 1 | night, urban | 4,992 | 18,889 | 25 | 5min 11s | Go, go left, warning, stop, stop left |
| Night seq. 2 | night, urban | 6,533 | 23,776 | 54 | 6min 48s | Go, go left, warning, stop, stop left |
| Day clip 1 | evening, urban, lens flare | 2,161 | 6,474 | 10 | 2min 15s | Go, stop |
| Day clip 2 | evening, urban | 1,031 | 2,230 | 6 | 1min 4s | Go, go left, warning left, stop, stop left |
| Day clip 3 | evening, urban | 643 | 1,087 | 3 | 40s | Go, warning, stop |
| Day clip 4 | evening, urban | 397 | 859 | 8 | 24s | Go |
| Day clip 5 | morning, urban | 2,667 | 9,717 | 8 | 2min 46s | Go, go left, warning, warning left, stop, stop left |
| Day clip 6 | morning, urban | 468 | 1,215 | 4 | 29s | Go, stop, stop left |
| Day clip 7 | morning, urban | 2,718 | 8,189 | 10 | 2min 49s | Go, go left, warning, warning left, stop, stop left |
| Day clip 8 | morning, urban | 1,040 | 2,025 | 8 | 1min 4s | Go, go left, stop, stop left |
| Day clip 9 | morning, urban | 960 | 1,264 | 4 | 59s | Go, go left, warning left, stop, stop left |
| Day clip 10 | morning, urban | 48 | 109 | 4 | 3s | Go, stop |
| Day clip 11 | morning, urban | 1,052 | 1,268 | 6 | 1min 5s | Go, stop |
| Day clip 12 | morning, urban | 152 | 229 | 3 | 9s | Go |
| Day clip 13 | evening, urban | 693 | 873 | 8 | 43s | Go, warning, stop |
| Night clip 1 | night, urban | 591 | 1,885 | 8 | 36s | Go |
| Night clip 2 | night, urban | 2,299 | 4,205 | 25 | 2min 24s | Go, go left, warning, stop, stop left |
| Night clip 3 | night, urban | 1,051 | 1,476 | 14 | 1min 6s | Go, go left, warning left, stop, stop left |
| Night clip 4 | night, urban | 1,104 | 2,538 | 9 | 1min 9s | Go, warning, stop |
| Night clip 5 | night, urban | 1,453 | 3,242 | 19 | 1min 31s | Go, go left, warning, stop, stop left |
| | | 46,418 | 112,971 | 290 | 44min 24s | |

academic domain, to development from industry of systems robust enough to be used in autonomous systems as seen with [38]. In the academic domain a large set approaches for solving the issues related to TLR have been introduced. This have been beneficial to the industry which has been able to adapt some of these findings with success. As a consequence of the different scopes, current evaluation data is captured under different conditions and for different purposes. The ideal TLR benchmark should provide a large variation in environmental conditions, similar to *The KITTI Vision Benchmark Suite* presented in [68] for e.g. stereo evaluation, and the *VIVA challenge* [67] for hands, face, and traffic signs. Currently [72] provides a benchmark with the only publicly available dataset. The dataset is not widely used and is limited to one environment type. A comparison between the dataset and the LISA TL dataset released along with this paper is located in Table VIII. When TLR systems eventually matures, the evaluation metrics should evolve to include weighted penalties for missed or wrong recognitions based on the severity of the error. Furthermore, the distance where TLR systems are first able to successfully recognize a TL is very relevant and should also be part of the evaluation.

The papers that seems to perform the best and are widely cited are [46], [38], [41] from the industry, and [56], [54] from academic institutions. The approaches from these papers rely on prior maps of TL location and properties, which makes it possible to achieve solid performance under challenging conditions. Such systems can reduce the number of false positives substantially, because they know where TLs should and should not be. Systems relying on precise maps have a big advantage over conventional systems. The price is less flexibility and high cost, since the maps must be kept up to date for the systems to function.

The paradigm change seen in traffic sign recognition, and pedestrian detection from model-based to learning based detection has not yet happened for TLR. This is underlined by examining Table I where detection of candidate TLs is

TABLE X: DAS applications proposed in recent studies on TLR.

| DAS feature | Description | Requirements | References |
|---|---|---|---|
| Dashboard visualization | Help driver by showing TL state in dashboard, where will be easily visible. Particularly useful for people with color vision deficiency. | TLR recognition and lane understanding | [52], [43] |
| Get going alert | Help driver by drawing attention to the recently switched green light | TLR recognition | [52] |
| Warn driver of red TL | Help driver by drawing attention to upcoming red TL in intersections | TLR recognition, intersection, and lane understanding | [64], [54], [42] |
| Stop at red lights | Enable autonomous vehicle to stop at red light | TLR recognition and lane understanding | [52] |
| Accurate stop at red lights | Smooth braking towards stop line at red TL | TLR recognition, stop line, and lane understanding | [43] |
| Stop and go | Enable vehicle to stop engine when waiting at a red light | TLR recognition | [52], [43] |

entirely based on heuristic models. This raises the question whether such model-based approaches outperforms learning based approaches. Since only the best performing approach from each paper is listed in Table I, it is not apparent that [40], [46] developed learning based TL detectors to compared with their model-based systems. The learning based approaches utilize cascading classifiers based on Haar features. In [46] the approach based on the learning based detector reaches a recognition rate of 90 % against 95 % for the best performing color based approach. Their model-based approaches are able to run in real time, while the cascading classifier is approximately a factor of ten slower. In [40] the model-based approach consistently achieves significantly better precision and similar recall. Detection of TL candidates using learning based detectors has not been researched sufficiently. Other features, that have proved useful elsewhere should be investigated e.g. HoG, LBP, ICF, and ACF.

The additional information that stereo vision provides is rarely used, one exception to this is [46] where stereo vision is used to measure real world distance and size of detected objects. Doing this resulted in a ten fold decrease in false candidates, along with the benefits that knowing distance and size gives to tracking. Stereo vision cues could be considered as an additional feature channel, or for rejecting false positives from e.g. tail lights and reflections. In other areas stereo vision has proven useful, in [75] it is discussed how stereo vision can improve the robustness of computer vision systems. In [76] vehicle detection at night is assisted by a stereo vision 3D edges extractor, while in [77], vehicle detection rely solely on stereo vision for both day- and nighttime data.

Less than half of the TLR papers include tracking. The most popular tracking method is a simple temporal consistency check. This efficiently suppress FPs and lone FNs. A few papers uses more advanced and sophisticated tracking, such as HMM, IMM, and CAMSHIFT. This is an area that must be researched further as tracking is known to increase performance as seen in [16] where introduction of tracking of traffic signs significantly reduced the number of FPs. For vehicle detection, [78] has similarly increased performance based on vehicle tracking fused with lane localization and tracking.

*DAS TLR Applications*

There are several applications in which TLR can be used in DAS. In Table X, DAS and autonomous applications of TLR which are mentioned in the surveyed papers are listed along with their requirements. Fusion of systems and data from multiple sensors can greatly improve the overall capabilities of DAS. In [32], [31] the driver's attention is measured using cameras looking inside the car. In [30] a first person view camera is used for capturing. The driver's registered attention can e.g. be used to activate safety systems in case of the driver being inattentive. By fusing TL recognition with looking-in systems which e.g detect the driver's eye gaze, it can be determined whether or not the driver have noticed the TL and if the driver should be made aware of it. Other properties to decide if the driver should be informed are the TL's detectability and discriminability as discussed in [79]. Velocity information from the CAN bus can also be obtained to help determining if the vehicle is slowing down while approaching the TL. A lot of applications require fusion of information from multiple systems, this includes most of the application seen in Table X.

Understanding the traffic scene is necessary as seen with the use of intersection and lane information. A major challenge for TLR in complex intersections is to determine which TLs are relevant to the driver. Selecting the biggest and closest one, as in [12], is a simplistic way of determining which lights to adhere to. In complex intersections, this will not be sufficient and more intelligent approaches must be applied. So far the most intelligent systems for solving this problem is seen in [41], where a guess is made based on the intersection width and the estimated orientation of the TLs. An alternative and less dynamic approach was seen in [38], where relevant TLs are manually annotated before hand. Features are extracted in the annotated regions and the system is then able to recognize the relevant TLs on that specific route.

TLR can potentially help people who tend to experience visual fatigue and decrease the stress level while driving. This is especially true for people with color vision deficiency or similar challenges. As mentioned in the introduction, a large portion of accidents are connected intersections and red light running. Integration of TLR systems in cars can reduce these accidents. Furthermore, the integration of TLR

system and DAS in cars could to some degree be implemented on smartphones as seen with [11]. Other applications for a developed TLR system is to use it for naturalistic driving studies (NDS) analysis by automatic detection of events related to e.g. red running at intersections. Something similar is done with lane detection in [80], where a set of NDS events are identified and quantified.

*Directions*

Even though the Daimler group in Germany and the VisLab group in Italy, have successfully managed to make autonomous vehicles drive on public roads, the TLR problem is not considered solved. Under many conditions, TLR systems will be challenged and experience errors. This is mainly due to changing weather and light conditions. To overcome these challenges, TLR systems should be able to adapt parameters throughout the TLR pipeline to the changing conditions. Another major problem that still remains to be solved is determining the relevance of recognized TLs. More research should be made into extending TLR for DAS with lane detection, detailed maps and other traffic scene information.

## X. Concluding Remarks

This survey presented an overview of the current state of traffic light recognition (TLR) research in relation to driver assistance systems. The approaches from the surveyed paper was divided into color spaces, detection, features, state detection, and tracking. There is no clear tendency towards using one particular color space. Some recent work has begun combining channels from multiple color spaces to create a combined color space that separates the relevant traffic light (TL) colors well. Most detection approaches rely on color or shape for finding TL candidates other rely on spotlight detection in a single intensity channel. BLOB analysis is generally used to remove bad TL candidates, this is done based on prior knowledge of the properties of TL BLOBs. Furthermore, some of the best performing approaches use detailed maps of the route and temporal information to improve performance. Many papers utilize manually specified models of TLs, which consist of color, shape, and structural features, to do state detection of TL candidates. Other use trained features such as HoG, LBP and 2D Gabor wavelets, classified using SVM. A few rely on template matching or neural networks using the color and/or shape. The tracking stage is dominated by temporal filtering, while more advanced approaches include HMM, IMM, and CAMSHIFT.

TLR is dominated by model based approaches, especially for finding TL candidates. This raises the question of whether model based approaches outperform learning based approaches for TLR. Based on the limited experiences with learning based detection this question cannot yet be answered. Additionally, because the systems are evaluated using different methodology and on very different datasets it is not clear which approaches are the best. Only one public database with TLs is currently available and, it is not widely used. We have therefore contributed a new database, the LISA Traffic Light Database, which contains TLs captured with a stereo camera in San Diego, USA under varying conditions. The database is suppose to enable comparable evaluation on a large and varied dataset, and provides the possibility of including stereo vision for improving TLR. The dataset should eventually be included in a challenge or benchmark similar to the *VIVA Challenge* [67] or *The KITTI Vision Benchmark Suite* [68].

## Acknowledgment

The authors would like to thank our colleagues at the LISA-CVRR Laboratory.

## References

[1] Federal Highway Administration. (2009) Traffic control devices: Uses and misuses. [Online]. Available: http://safety.fhwa.dot.gov/intersection/resources/fhwasa10005/brief_3.cfm

[2] AAA Foundation for Traffic Safety. (2014) 2014 traffic safety culture index. [Online]. Available: https://www.aaafoundation.org/sites/default/files/2014TSCIreport.pdf

[3] D. Shinar, *Traffic Safety and Human Behavior*, ser. Traffic Safety and Human Behavior. Emerald, 2007, no. vb. 5620.

[4] J. Sussman, *Perspectives on Intelligent Transportation Systems (ITS)*. Springer US, 2005.

[5] P. Papadimitratos, A. La Fortelle, K. Evenssen, R. Brignolo, and S. Cosenza, "Vehicular communication systems: Enabling technologies, applications, and future outlook on intelligent transportation," *IEEE Communications Magazine*, vol. 47, pp. 84–95, 2009.

[6] STT Nehterlands Study Centre for Technology Trends. (2009) Tomorrow's transport starts today. [Online]. Available: http://stt.nl/wp/wp-content/uploads/2014/05/STT_SIV_ENG_LRspreads.pdf

[7] World Health Organization. (2013) Global status report onroad safety 2013. [Online]. Available: http://www.who.int/violence_injury_prevention/road_safety_status/2013/en/

[8] America on the Move. (2015) Auto safety history. [Online]. Available: http://amhistory.si.edu/onthemove/themes/story_86_1.html

[9] A. Mogelmose, M. M. Trivedi, and T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, pp. 1484–1497, 2012.

[10] E. Koukoumidis, L.-S. Peh, and M. R. Martonosi, "Signalguru: leveraging mobile phones for collaborative traffic signal schedule advisory," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 127–140.

[11] E. Koukoumidis, M. Martonosi, and L.-S. Peh, "Leveraging smartphone cameras for collaborative road advisories," *IEEE Transactions on Mobile Computing*, vol. 11, pp. 707–723, 2012.

[12] Y. Jie, C. Xiaomin, G. Pengfei, and X. Zhonglong, "A new traffic light detection and recognition algorithm for electronic travel aid," in *Fourth International Conference on Intelligent Control and Information Processing (ICICIP)*, 2013, pp. 644–648.

[13] The Insurance Institute for Highway Safety (IIHS). (2015) Red light running. [Online]. Available: http://www.iihs.org/iihs/topics/t/red-light-running/topicoverview

[14] Vermont Agency of Transportation. (2009) An evaluation of dilemma zone protection practices for signalized intersection control. [Online]. Available: http://vtransplanning.vermont.gov/sites/aot_program_development/files/documents/materialsandresearch/completedprojects/Dilemma_Zone_Final_Report_6_19_09.pdf

[15] Federal Highway Administration. (2009) Engineering countermeasures to reduce red-light running. [Online]. Available: http://safety.fhwa.dot.gov/intersection/resources/fhwasa09027/resources/intersection\%20safety\%20issue\%20brief\%206.pdf

[16] A. Mogelmose, D. Liu, and M. M. Trivedi, "Traffic sign detection for us roads: Remaining challenges and a case for tracking," in *IEEE Transactions on Intelligent Transportation Systems*. IEEE, 2014, pp. 1394–1399.

[17] M. Mathias, R. Timofte, R. Benenson, and L. Van Gool, "Traffic sign recognition - how far are we from the solution?" in *ICJNN*, 2013.

[18] Federal Highway Administration. (2015) Nighttime visibility. [Online]. Available: http://safety.fhwa.dot.gov/roadway_dept/night_visib/

[19] H. Fleyeh and M. Dougherty, "Road and traffic sign detection and recognition," in *Proceedings of the 16th Mini-EURO Conference and 10th Meeting of EWGT*, 2005, pp. 644–653.

[20] R. O'Malley, M. Glavin, and E. Jones, "Vehicle detection at night based on tail-light detection," in *1st international symposium on vehicular computing systems, Trinity College Dublin*, 2008.

[21] Y.-L. Chen, C.-T. Lin, C.-J. Fan, C.-M. Hsieh, and B.-F. Wu, "Vision-based nighttime vehicle detection and range estimation for driver assistance," in *IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2008, pp. 2988–2993.

[22] C. Idler, R. Schweiger, D. Paulus, M. Mahlisch, and W. Ritter, "Realtime vision based multi-target-tracking with particle filters in automotive applications," in *IEEE Intelligent Vehicles Symposium*. IEEE, 2006, pp. 188–193.

[23] S. Gormer, D. Muller, S. Hold, M. Meuter, and A. Kummert, "Vehicle recognition and ttc estimation at night based on spotlight pairing," in *12th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 2009, pp. 1–6.

[24] R. O'Malley, E. Jones, and M. Glavin, "Rear-lamp vehicle detection and tracking in low-exposure color video for night conditions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, pp. 453–462, 2010.

[25] J. C. Rubio, J. Serrat, A. M. López, and D. Ponsa, "Multiple-target tracking for intelligent headlights control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, pp. 594–605, 2012.

[26] R. O'malley, M. Glavin, and E. Jones, "Vision-based detection and tracking of vehicles to the rear with perspective correction in low-light conditions," *IET Intelligent Transport Systems*, vol. 5, pp. 1–10, 2011.

[27] S. Eum and H. G. Jung, "Enhancing light blob detection for intelligent headlight control using lane detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, pp. 1003–1011, 2013.

[28] R. Satzoda and M. Trivedi, "On enhancing lane estimation using contextual cues," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. PP, no. 99, pp. 1–1, 2015.

[29] A. Bar Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: a survey," *Machine Vision and Applications*, pp. 1–19, 2012. [Online]. Available: http://dx.doi.org/10.1007/s00138-011-0404-2

[30] A. Tawari, A. Mogelmose, S. Martin, T. B. Moeslund, and M. M. Trivedi, "Attention estimation by simultaneous analysis of viewer and view," in *IEEE 17th International Conference on Intelligent Transportation Systems*. IEEE, 2014, pp. 1381–1387.

[31] A. Tawari, K. H. Chen, and M. M. Trivedi, "Where is the driver looking: Analysis of head, eye and iris for robust gaze zone estimation," in *IEEE 17th International Conference on Intelligent Transportation Systems*. IEEE, 2014, pp. 988–994.

[32] A. Tawari, S. Sivaraman, M. Trivedi, T. Shannon, and M. Tippelhofer, "Looking-in and looking-out vision for urban intelligent assistance: Estimation of driver attentive state and dynamic surround for safe merging and braking," in *IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 115–120.

[33] P. Dollar, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 8, pp. 1532–1545, Aug 2014.

[34] California Department of Transportation. (2015) California manual on uniform traffic control devices. [Online]. Available: http://www.dot.ca.gov/hq/traffops/engineering/control-devices/trafficmanual-current.htm

[35] Federal Highway Administration. (2015) Manual on uniform traffic control devices. [Online]. Available: http://mutcd.fhwa.dot.gov/

[36] United Nations. (2006) Vienna convention on road signs and signals. [Online]. Available: www.unece.org/trans/conventn/signalse.pdf

[37] C.-C. Chiang, M.-C. Ho, H.-S. Liao, A. Pratama, and W.-C. Syu, "Detecting and recognizing traffic lights by genetic approximate ellipse detection and spatial texture layouts," *International Journal of Innovative Computing, Information and Control*, vol. 7, no. 12, pp. 6919–6934, 2011.

[38] U. Franke, D. Pfeiffer, C. Rabe, C. Knoeppel, M. Enzweiler, F. Stein, and R. Herrtwich, "Making bertha see," in *IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2013, pp. 214–221.

[39] R. de Charette and F. Nashashibi, "Real time visual traffic lights recognition based on spot light detection and adaptive traffic lights templates," in *IEEE Intelligent Vehicles Symposium*, 2009, pp. 358–363.

[40] R. Charette and F. Nashashibi, "Traffic light recognition using image processing compared to learning processes," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 333–338.

[41] N. Fairfield and C. Urmson, "Traffic light mapping and detection," in *Proceedings of ICRA 2011*, 2011.

[42] H.-K. Kim, Y.-N. Shin, S.-g. Kuk, J. H. Park, and H.-Y. Jung, "Nighttime traffic light detection based on svm with geometric moment features," *World Academy of Science, Engineering and Technology 76th*, pp. 571–574, 2013.

[43] M. Diaz-Cabrera, P. Cerri, and P. Medici, "Robust real-time traffic light detection and distance estimation using a single camera," *Expert Systems with Applications*, pp. 3911–3923, 2014.

[44] M. M. Trivedi, T. Gandhi, and J. McCall, "Looking-in and looking-out of a vehicle: Computer-vision-based enhanced vehicle safety," *IEEE Transactions on Intelligent Transportation Systems*, pp. 108–120, 2007.

[45] E. Ohn-Bar, A. Tawari, S. Martin, and M. M. Trivedi, "On surveillance for safety critical events: In-vehicle video networks for predictive driver assistance systems," *Computer Vision and Image Understanding*, vol. 134, no. 0, pp. 130 – 140, 2015, image Understanding for Real-world Distributed Video Networks. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314214002070

[46] F. Lindner, U. Kressel, and S. Kaelberer, "Robust recognition of traffic signals," in *IEEE Intelligent Vehicles Symposium*, 2004, pp. 49–53.

[47] H.-K. Kim, J. H. Park, and H.-Y. Jung, "Effective traffic lights recognition method for real time driving assistance system in the daytime," *World Academy of Science, Engineering and Technology 59th*, 2011.

[48] Z. Cai, Y. Li, and M. Gu, "Real-time recognition system of traffic light in urban environment," in *IEEE Symposium on Computational Intelligence for Security and Defence Applications (CISDA)*. IEEE, 2012, pp. 1–6.

[49] M. Omachi and S. Omachi, "Traffic light detection with color and edge information," in *2nd IEEE International Conference on Computer Science and Information Technology*. IEEE, 2009, pp. 284–287.

[50] M. Diaz-Cabrera and P. Cerri, "Traffic light recognition during the night based on fuzzy logic clustering," in *Computer Aided Systems Theory-EUROCAST 2013*. Springer Berlin Heidelberg, 2013, pp. 93–100.

[51] M. Diaz-Cabrera, P. Cerri, and J. Sanchez-Medina, "Suspended traffic lights detection and distance estimation using color features," in *15th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 2012, pp. 1315–1320.

[52] D. Nienhuser, M. Drescher, and J. Zollner, "Visual state estimation of traffic lights using hidden markov models," in *13th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 2010, pp. 1705–1710.

[53] G. Trehard, E. Pollard, B. Bradai, and F. Nashashibi, "Tracking both pose and status of a traffic light via an interacting multiple model filter," in *17th International Conference on Information Fusion (FUSION)*. IEEE, 2014, pp. 1–7.

[54] Y. Zhang, J. Xue, G. Zhang, Y. Zhang, and N. Zheng, "A multi-feature fusion based traffic light recognition algorithm for intelligent vehicles," in *33rd Chinese Control Conference (CCC)*. IEEE, 2014, pp. 4924–4929.

[55] C. Jang, C. Kim, D. Kim, M. Lee, and M. Sunwoo, "Multiple exposure images based traffic light recognition," in *IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 2014, pp. 1313–1318.

[56] J. Levinson, J. Askeland, J. Dolson, and S. Thrun, "Traffic light mapping, localization, and state detection for autonomous vehicles," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 5784–5791.

[57] J. Gong, Y. Jiang, G. Xiong, C. Guan, G. Tao, and H. Chen, "The recognition and tracking of traffic lights based on color segmentation and camshift for intelligent vehicles," in *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2010, pp. 431–435.

[58] Y. Shen, U. Ozguner, K. Redmill, and J. Liu, "A robust video based traffic light detection algorithm for intelligent vehicles," in *IEEE Intelligent Vehicles Symposium*, 2009, pp. 521–526.

[59] C. Wang, T. Jin, M. Yang, and B. Wang, "Robust and real-time traffic lights recognition in complex urban environments," *International Journal of Computational Intelligence Systems*, vol. 4, no. 6, pp. 1383–1390, 2011.

[60] V. John, K. Yoneda, B. Qi, Z. Liu, and S. Mita, "Traffic light recognition in varying illumination using deep learning and saliency map," in *IEEE 17th International Conference on Intelligent Transportation Systems*. IEEE, 2014, pp. 2286–2291.

[61] A. Gomez, F. Alencar, P. Prado, F. Osorio, and D. Wolf, "Traffic lights detection and state estimation using hidden markov models," in *IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 750–755.

[62] M. Omachi and S. Omachi, "Detection of traffic light using structural information," in *IEEE 10th International Conference on Signal Processing (ICSP)*, 2010, pp. 809–812.

[63] G. Siogkas, E. Skodras, and E. Dermatas, "Traffic lights detection in adverse conditions using color, symmetry and spatiotemporal information." in *VISAPP (1)*, 2012, pp. 620–627.

[64] S. Sooksatra and T. Kondo, "Red traffic light detection using fast radial symmetry transform," in *11th International Conference on Electrical*

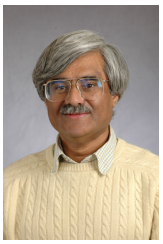
*Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. IEEE, 2014, pp. 1–6.
[65] S. Naz, H. Majeed, and H. Irshad, "Image segmentation using fuzzy clustering: A survey," in *Emerging Technologies (ICET), 2010 6th International Conference on*, 2010, pp. 181–186.
[66] H. A. Blom and Y. Bar-Shalom, "The interacting multiple model algorithm for systems with markovian switching coefficients," *IEEE Transactions on Automatic Control*, vol. 33, pp. 780–783, 1988.
[67] U. Laboratory for Intelligent and Safe Automobiles . (2015) Vision for intelligent vehicles and applications (viva) challenge. [Online]. Available: http://cvrr.ucsd.edu/vivachallenge/
[68] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
[69] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, pp. 303–338, 2010.
[70] C. G. Weng and J. Poon, "A new evaluation measure for imbalanced datasets," in *Proceedings of the 7th Australasian Data Mining Conference-Volume 87*. Australian Computer Society, Inc., 2008, pp. 27–32.
[71] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 233–240.
[72] Robotics Centre of Mines ParisTech. (2015) Traffic lights recognition (tlr) public benchmarks. [Online]. Available: http://www.lara.prd.fr/benchmarks/trafficlightsrecognition
[73] S. Sivaraman and M. M. Trivedi, "A review of recent developments in vision-based vehicle detection." in *Intelligent Vehicles Symposium*, 2013, pp. 310–315.
[74] R. Danescu, F. Oniga, and S. Nedevschi, "Modeling and tracking the driving environment with a particle-based occupancy grid," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, pp. 1331–1342, 2011.
[75] N. Buch, S. Velastin, and J. Orwell, "A review of computer vision techniques for the analysis of urban traffic," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, pp. 920–939, 2011.
[76] I. Cabani, G. Toulminet, and A. Bensrhair, "Color-based detection of vehicle lights," in *IEEE Intelligent Vehicles Symposium*. IEEE, 2005, pp. 278–283.
[77] M. P. Philipsen, M. B. Jensen, R. K. Satzoda, M. M. Trivedi, A. Møgelmose, and T. B. Moeslund, "Night-time drive analysis using stereo-vision for data reduction in naturalistic driving studies," in *Intelligent Vehicle Symposium*. IEEE, 2015.
[78] S. Sivaraman and M. M. Trivedi, "Integrated lane and vehicle detection, localization, and tracking: A synergistic approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, pp. 906–917, 2013.
[79] F. Kimura, Y. Mekada, T. Takahashi, I. Ide, H. Murase, and Y. Tamatsu, "Method to quantify the visibility of traffic signals for driver assistance," *IEEJ Transactions on Electronics, Information and Systems*, vol. 130, pp. 1034–1041, 2010.
[80] R. Satzoda and M. Trivedi, "Drive analysis using vehicle dynamics and vision-based lane semantics," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, pp. 9–18, 2015.

**Mark Philip Philipsen** received the B.Sc. degree in Computer Engineering with specialization in Informatics from Aalborg University, Aalborg, Denmark, in 2013. He is currently working toward the M.Sc. degree in Vision, Graphics, and Interactive systems from Aalborg University.
He has been a Visiting Scholar with the Computer Vision and Robotics Research Laboratory, University of California, San Diego. His research interests are computer vision and machine learning.

**Morten Bornø Jensen** received the B.Sc. degree in Computer Engineering with specialization in Informatics from Aalborg University, Aalborg, Denmark, in 2013. He is currently working toward the M.Sc. degree in Vision, Graphics, and Interactive Systems from Aalborg University.
He has been a Visiting Scholar with the Computer Vision and Robotics Research Laboratory, University of California, San Diego. His research interests are computer vision and machine learning.

**Andreas Møgelmose** received the B.Sc. degree in computer engineering on the topic of information processing systems and the Master's degree in Vision, Graphics, and Interactive Systems from Aalborg University, Aalborg, Denmark, in 2010 and 2012, respectively. He is currently working toward the Ph.D. degree with the Visual Analysis of People Laboratory, Aalborg University.
He has been a Visiting Scholar with the Computer Vision and Robotics Research Laboratory, University of California, San Diego. His research interests are computer vision and machine learning.

**Thomas Baltzer Moeslund** received the M.Sc.E.E. and Ph.D. degrees from Aalborg University, Aalborg, Denmark, in 1996 and 2003, respectively. He is currently a Professor and the Head of the Visual Analysis of People Laboratory with Aalborg University. He has been involved in ten national and international research projects, as a Coordinator, Work Package leader, and Researcher. He is the author of about 100 peer-reviewed papers. His research interests include all aspects of computer vision, with a special focus on automatic analysis of people. Prof. Moeslund has been a Co-chair of four international workshops/tutorials and a member of the Program Committee for a number of conferences and workshops. He serves as an Associate Editor and as a member of the Editorial Board for four international journals. He received the Most Cited Paper Award in 2009, the Best IEEE Paper Award in 2010, and the Teacher of the Year Award in 2010.



**Mohan Manubhai Trivedi** received the B.E. (with honors) degree in electronics from Birla Institute of Technology and Science, Pilani, India, in 1974 and the M.S. and Ph.D. degrees in electrical engineering from Utah State University, Logan, UT, USA, in 1976 and 1979, respectively. He is currently a Professor of Electrical and Computer Engineering and the Founding Director of the Computer Vision and Robotics Research Laboratory and the Laboratory for Intelligent and Safe Automobiles (LISA) at the University of California, San Diego. He and his team are currently pursuing research in machine and human perception, multimodal interfaces and interactivity, machine learning, intelligent vehicles, driver assistance and active safety systems. Prof. Trivedi is serving on the Board of Governors of the IEEE Intelligent Society and on the Editorial Board of the IEEE Transportation Systems Transactions on ITS. Prof. Trivedi is a Fellow of the IEEE (for contributions to Intelligent Transportation Systems), Fellow of the IAPR (for contributions to vision systems for situational awareness and human centered vehicle safety), and Fellow of the SPIE (for distinguished contributions to the field of optical engineering).

A major part of our work on TLR is the extensive stereo dataset of TLs which is described in both survey papers and more in depth in Chapter 6.

## 4.2 Emerging Trends for Traffic Lights: Detection and Evaluation

The conference paper *Emerging Trends for Traffic Lights: Detection and Evaluation*, which is submitted to the IEEE Intelligent Transportation Systems Conference (ITSC), 2015. It focuses on the problem of detecting TLs, which we consider the main challenge in TLR. An overview of four recent approaches to detection are presented in order to give a feel for the state of the field. We introduce our preferred evaluation procedure of detectors together with the LISA traffic light dataset, in order to encourage researchers to train, evaluate and compare their work using this dataset.

# Emerging Trends for Traffic Lights: Detection and Evaluation

Mark P. Philipsen[1,2], Morten B. Jensen[1,2],
Andreas Møgelmose[1], Thomas B. Moeslund[1], and Mohan M. Trivedi[2]

***Abstract*— Research in traffic light recognition (TLR) has stagnated compared to related computer vision areas, such as pedestrian detection and and traffic sign recognition. We focus on the detection sub-problem, since this is the most challenging problem and solving this is the key to a successful TLR system. Evaluation of existing systems is currently limited primarily to small local datasets. In order to provide a common basis for comparison of future TLR research an extensive public database is collected based on footage from US roads. The database consists of both test and training data, totaling 48,048 frames and 90,767 annotated traffic lights, captured in both night and day scenarios with varying light and weather conditions.**

## I. Introduction

Driver assistance systems are gaining a lot of momentum currently, as evident in top models from many prominent car manufacturers. Recognition of traffic lights (TLs) could be a desirable addition but judging by the state of current research in this area, consumer ready systems are not on the immediate horizon. Traffic light recognition (TLR) consists of three sub-problems, detection, classification, and tracking. Figure 1 illustrates the typical flow of such a computer vision system. A similar breakdown is done for traffic sign recognition in [1].

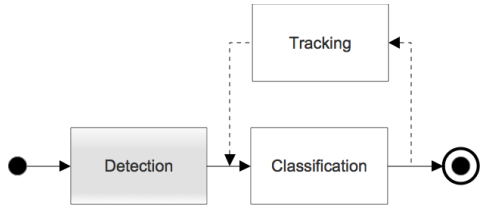



Fig. 1: Breakdown of a vision based TLR system.

The detection and classification stages are executed sequentially on each frame, whereas the tracking stage feeds back spatial and temporal information between frames. For TLR both the detection and classification stages are comparable to the equivalent stages in traffic sign recognition. Tracking of TLs differs, since signs are static and TLs change states. More about the coventions, structure and dynamics of TLs in section II. The detection problem covers locating desired candidate TLs. Candidates are either rejected or accepted in the classification stage based on features extracted from the detected candidates. Furthermore, the state of accepted candidates is determined. In tracking the location and state of TLs are tracked through a frame sequence. Since detection of TL candidates is the foundation for a successful classification and tracking we will focus exclusively on this for the remainder of this paper. Unlike for sign recognition and pedestrian detection, no surveys of TLR research exist. The purpose of this paper is to highlight some prominent approaches to TL detection from a few recent papers, as well as describe a common procedure for evaluation of such detectors. Most of the research current published is evaluated based on local datasets with a limited number of TLs and little variation. This makes comparison between existing methods and new contributions difficult. We introduce a comprehensive TL dataset along with a proposal for a common evaluation procedure for traffic light detectors. The KITTI Vision Benchmark Suite[2] benchmark various applications, such as stereo, object detection, and odometry, with the purpose of reducing the bias and providing real-world test scenarios with a high set of difficulties. The recently introduced *VIVA Challenge* is also used for to benchmarking a large set of difficult tasks associated with drivers, occupants, vehicle dynamics, and vehicle surroundings. The *VIVA Challenge* include hands, face, and sign datasets captured under challenging naturalistic driver settings. The TL dataset published with this work is eventually going to be included in the *VIVA Challenge*.

The contributions made in this survey paper are thus threefold:

1) Provide an overview of 4 recent TLR papers' approaches to detection .
2) Introduce a common evaluation procedure for TL detectors.
3) Publish an extensive high resolution, annotated, stereo video database, with day and night video sequences.

The paper is organized as follows: Section II, gives an overview of the possible appearances of TLs, along with common challenges that TL detection systems are subject to. Related research is summarized in section III. In section IV, we present a new database for evaluation of TL detection systems. Finally, section VI rounds of with some concluding remarks.

## II. Traffic Lights: Structure and Challenges

Traffic lights are by design made to stand out and be easily visible by using bright uniformly colored lamps surrounded by a uniform, often dark box. The purpose of a TL is the same across the world, it must safely regulate the traffic flow, while warning drivers about the state of the intersection

[1]Visual Analysis of People Laboratory, Aalborg University, 9000 Aalborg, Denmark.
[2]Computer Vision and Robotics Research Laboratory, UC San Diego, La Jolla, CA 92093-0434, USA

ahead. The most common TL configuration is the basic red-yellow-green signal, where each state indicates whether a driver should stop, be prepared to stop, or keep driving. Worldwide there are large variations in TL designs; however, all follow a few general guidelines. A TL consists of a box that holds differently colored, and sometimes differently shaped lamps. The orientation, color, size, and shape of the box will vary country to country and even city to city. In the U.S. TLs are regulated by the Federal Highway Administration in the *Manual on Uniform Traffic Control Devices* [3] and most European countries have signed the *Vienna Convention on Road Signs and Signals* [4], requiring TLs to meet a common international standard.

### A. *Challenges in recognizing traffic lights*

Although TLs are made to be easily recognizable, influences from the environment and sometimes sub-optimal placement can make successful detection difficult, if not impossible. Issues include:

- Color tone shifting and halo disturbances because of influences from the atmosphere and glass that the light passes through[5]. Fig. 2(c).
- Occlusion and partial occlusion because of other objects or oblique viewing angles[5]. This is especially a problem with supported TLs [6], [7], [8]. Fig. 2(e),(f),(g).
- Incomplete shapes because of malfunctioning lights[5] or dirty lamps 2. Fig. 2(a),(b).
- False positives from, brake lights, reflections, billboards[9], [10], and pedestrian crossing lamps. Fig. 2(h).
- Changes in lighting due to adverse weather conditions and the positioning of the sun and other light sources. Fig. 2(d),(k),(l).
- Mismatch between camera's shutter speed and TL LED's duty cycle. Fig. 2(i),(j).

Inconsistencies in TL lamps can be caused by dirt, defects, or the relatively slow duty cycle of the LEDs. The duty cycle is high enough for the human eye not to notice that the lights are actually blinking. Issues arise when a camera uses fast shutter speeds, leading to some frames not contain a lit TL lamp. Saturation is another aspect that can influence the appearance of the lights. When transitioning between day and night, the camera parameters must be adjusted to let the optimal amount of light in and avoid under or over-saturation. [11] introduces an adaptive camera setting system, that change the shutter and gain settings based upon based on the luminosity of the pixels in the upper part of the image.

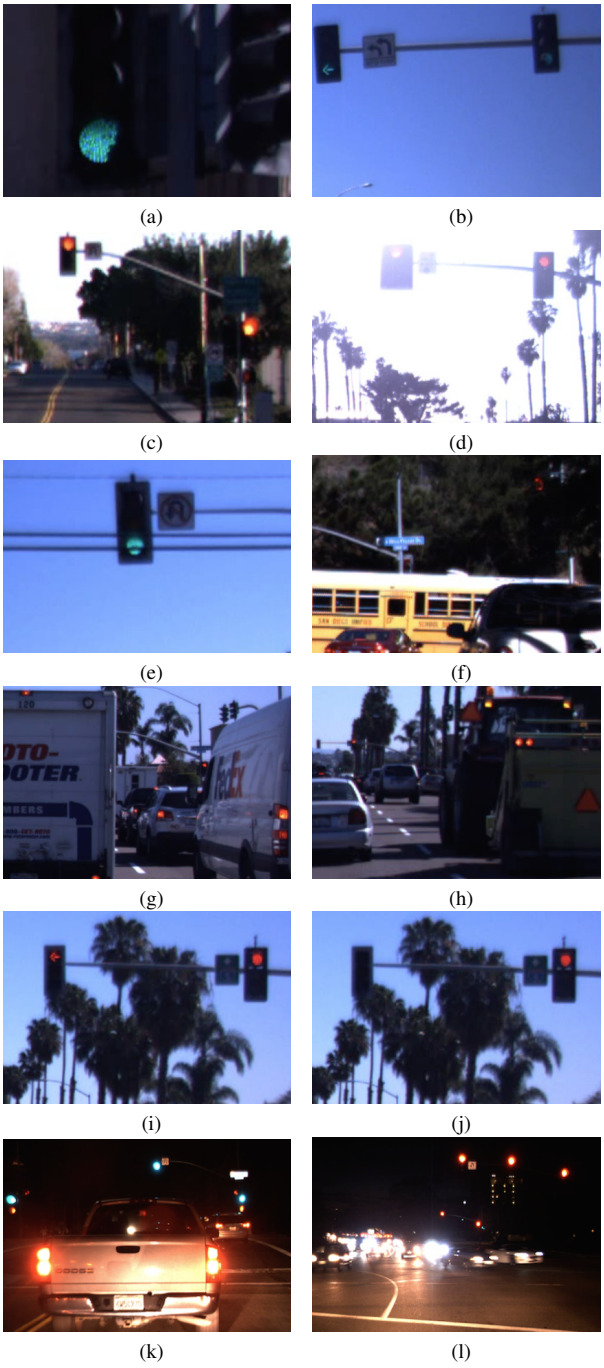



Fig. 2: (a) Examples of TLs from the collected dataset.

## III. Related Work

Common for [12], [8], [7] is a TL detector which relies purely on intensity from grayscale images. This has the advantage of being more robust to color distortion. Areas brighter than their surroundings are segmented using the white top-hat mophology operation, which leads to an initial high number of false candidates. False candidates are filtered out based on the shape information. Specifically, rejection is done based on criteria such as, dimension ratio, the BLOB being hole free and approximately convex. Furthermore, the areas of BLOBs are compared to the areas of regions grown from extrema in the original grayscale image. This is especially effective for removing false candidates big bright areas such as the sky. This detector relies heavily on a competent classifier for further rejection and state estimation, since the number of false candidates is very high and color information is not available. The detector manages to find 90% of all TLs in the testset.

[13] begins by detecting the vanishing line and thereby

reducing the search area considerably, relying on the assumption that TLs will only appear above this line. They then apply the the white top-hat operation as [12], [8], [7] did, on the intensity channel V from a HSV image. What is left is filter based on statistical measurements of the hue and saturation ranges of red and green lights. All pixels outside these ranges are rejected while the remaining pixels are selected as candidates. Remaining BLOBs are filtered based on size and height-width ratio. They then look for black bounding boxes around the BLOBs based on gradient information and the blackness of the inside of box candidates. Their system reaches an accuracy of 85%.

[10] extracts candidate BLOBs from RGB images by applying a color distance transform proposed in [14]. The transform emphasizes the chosen color in an intensity image, which is thresholded to remove to suppressed colors. This is followed by shape filtering to reduce noise using width/height ratio and the solidity of BLOBs. The solidity is calculated based on the ratio between the area of the BLOB and it's bounding box. When evaluating their system, they count a success if the TL was detected just once in the sequence, this allows them to reach a detection rate of 93.53%.

An easy way of improving the segmentation is to reduce the search area. This could be achieved by only looking in the upper half of the input image. A more precise way to reduce the searching window is seen in [9], where an off-line database containing prior knowledge of TL locations is used. The off-line database is created using the input image combined with accurate GPS measurements, and then manually hand-label the areas with TLs on a pre-captured image sequences. Given that the route which the TLR system is used upon is hand-labeled, such an off-line database can reduce the searching window and the detection problem significantly. However, collecting and maintaining such a database is very comprehensive. Besides using prior knowledge, [9] use the color and shape information for filtering.

## IV. LISA Traffic Light Database

As it was concluded in the traffic sign survey paper [1], the general approach for testing and validating a proposed method is to use a privately collected dataset. This is considered sufficient for preliminary testing and validation. But, when trying to estimate the performance of a contribution, it becomes difficult to compare the work with others'. The only currently public dataset is provided by *Robotics Centre of Mines ParisTech* in France. The dataset consists of 11,179 frames from a single 8m 49s long video. It contains 9,168 hand-labeled instances of TLs. More information about this dataset can be found in Table I.

A public TL database should support the three part stereo vision bottom-up paradigm described in [16]. [16] provides an overview of vehicle detection systems based on both monocular and stereo vision since 2005. Both monocular and stereo vision are widely used for solving this problem, but an interesting finding in this work is the stereo vision bottom-up paradigm which consist of visual odometry, feature points in 3D, and distinguishing static from moving points, which is also mentioned in [17]. The motivation for having a public TL database with stereo images is therefore that all three parts in this paradigm can help reduce the amount of false positives. This notion is reinforced in [18] where the main technical challenges in urban environments are occlusions, shadow silhouettes, and dense traffic. The introduction of stereo has shown promising result in relation to solving these challenges.

The database that is collected and released together with this paper is focused on TLR and contains TLs that are found in San Diego, California, USA. The database provides two day and two nighttime sequences for testing. The stereo image pairs are acquired using the Point Grey's Bumblebee XB3 (BBX3-13S2C-60) with a resolution of 1280 x 960, each with a Field of View(FoV) of 60°. The stereo camera supports two different baselines, 12 and 24 cm, whereof a baseline of 24 cm is used for the LISA TL database. The stereo images are uncompressed and rectified on the fly, and captured with a frame rate of 16 FPS. Capturing was done by mounting the stereo camera in the center of the roof on the capturing vehicle.

Besides the four test sequences, we provide 18 shorter video sequences consisting of TLs collected in the northern part of San Diego. These are intended for training and additional testing. They are organized as seen in Table II, which gives a detailed overview of all the video sequences that are made available with this paper. The number of annotations is the accumulated number of hand-labeled TLs on a frame-by-frame bases. The number of traffic lights is the physical number of TLs in the real world. The gain and shutter speed were manually set to avoid oversaturation as well as limit flickering of the TLs. For all day clips, a shutter speed of 1/5000 sec and a gain of 0 are used. For all night clips, a shutter speed of 0.0625 and a gain of 8 are used. A Triclops calibration file is provided along with the stereo images. This file contains the factory calibration of the used Bumblebee XB3 camera, to be used with the Point Grey's Triclops SDK.

Each sequence in the database comes with hand labeled annotations for the left frame stereo frame. Annotations for a given video sequence contains the following information: frame number, rectangular area around the lit traffic light lamp, and the state of that area. Labeling is done for every visibly lit TL lamp.

An example of annotated TL lamps is seen in Figure 3.

The LISA Traffic Light Database is made freely available at `http://cvrr.ucsd.edu/LISA/datasets.html` for educational, research, and non-profit purposes.

## V. Evaluation

A wide variety of approaches and metrics have been used to evaluate detector performance. A standardized methodology would make comparison more straightforward. For evaluations on the LISA Traffic Light database we suggest using: precision and recall, which are defined in equation (1) and (2). TP, FP, and FN are abbreviations for true positives,

TABLE I: Overview of current public TL databases.

| | *Robotics Centre of Mines ParisTech*[15] | *LISA Traffic Light Database* |
|---|---|---|
| # Classes | 4 (green, orange, red & ambiguous) | 7 (go, go forward, go left, warning, warning left, stop, & stop left) |
| # Images | 11,179 | 48,048 |
| Annotations | 9,168 | 90,767 |
| Image resolution | 640 x 480 8-bit RGB | 1280 x 960 8-bit RGB |
| Color | Yes | Yes |
| Stereo images | No | Yes |
| Place of origin | Paris, France | San Diego, USA |
| Video included | Yes | Yes |
| Description | 1 day time sequence in urban environment(downtown Paris) | 4 sequences and 18 clips, each with different environment, e.g. Morning, evening, and night |

TABLE II: Overview of the video sequences in LISA Traffic Light Database.

| **Sequence name** | **Description** | **# Frames** | **# Annotations** | **# TLs** | **Length** | **Classes** |
|---|---|---|---|---|---|---|
| Day sequence 1 | morning, urban | 4,800 | 10,267 | 25 | 5.00 min | Go, warning, warning left, stop, stop left |
| Day sequence 2 | evening, urban | 9,586 | 11,154 | 29 | 6.10 min | Go, go forward, go left, warning, stop, stop left |
| Night sequence 1 | night, urban | 4,976 | 7,949 | 17 | 5.11 min | Go, warning, stop |
| Night sequence 2 | night, urban | 6,528 | 8,693 | 21 | 6.48 min | Go, warning, stop |
| Day clip 1 | evening, urban, lens flare | 2,161 | 6,474 | 10 | 2.15 min | Go, stop |
| Day clip 2 | evening, urban | 1,031 | 2,230 | 6 | 1.04 min | Go, go left, warning left, stop, stop left |
| Day clip 3 | evening, urban | 643 | 1,087 | 3 | 0.40 min | Go, warning, stop |
| Day clip 4 | evening, urban | 397 | 859 | 8 | 0.24 min | Go |
| Day clip 5 | morning, urban | 2,667 | 9,717 | 8 | 2.46 min | Go, go left, warning, warning left, stop, stop left |
| Day clip 6 | morning, urban | 468 | 1,215 | 4 | 0.29 min | Go, stop, stop left |
| Day clip 7 | morning, urban | 2,718 | 8,189 | 10 | 2.49 min | Go, go left, warning, warning left, stop, stop left |
| Day clip 8 | morning, urban | 1,040 | 2,025 | 8 | 1.04 min | Go, go left, stop, stop left |
| Day clip 9 | morning, urban | 960 | 1,264 | 4 | 0.59 min | Go, go left, warning left, stop, stop left |
| Day clip 10 | morning, urban | 48 | 109 | 4 | 0.02 min | Go, stop |
| Day clip 11 | morning, urban | 1,052 | 1,268 | 6 | 1.05 min | Go, stop |
| Day clip 12 | morning, urban | 152 | 229 | 3 | 0.09 min | Go |
| Day clip 13 | evening, urban | 693 | 873 | 8 | 0.43 min | Go, warning, stop |
| Night clip 1 | night, urban | 2,560 | 4,231 | 9 | 2.40 min | Go, warning, stop |
| Night clip 2 | night, urban | 1,264 | 2,946 | 5 | 1.19 min | Go, warning, stop |
| Night clip 3 | night, urban | 352 | 539 | 3 | 0.22 min | Go, warning, stop |
| Night clip 4 | night, urban | 2,560 | 5,578 | 11 | 2.40 min | Go, warning, stop |
| Night clip 5 | night, urban | 1,392 | 3,871 | 6 | 1.27 min | Go, warning, stop |
| | | 48,048 | 90,767 | 208 | 46.1 min | |

Fig. 3: Example of annotated traffic light lamps.

false positives and false negatives. The TPs, FPs and FNs should be evaluated on a per frame basis.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Precision is the ratio of correct TL detections compared to the actual number of TLs.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Recall is the ratio of correct TL detections compared to the total number of detections.

For presenting and evaluating the overall system performance, we suggest generating a precision-recall curve and using the area-under-curve (AUC) as measure. A high AUC indicates good performance, an AUC of 100% indicates a perfect system for the testset. An example of a precision-recall curve is seen in Figure 4.

To determine TPs, the Pascal criterion is used:

$$a_0 = \frac{\text{area}(B_d \cap B_{gt})}{\text{area}(B_d \cup B_{gt})} \geq 0.5 \tag{3}$$

$a_0$ denotes the *overlap ratio* between the detected bounding box $B_d$ and the ground truth bounding box $B_{gt}$. ($B_d \cap$

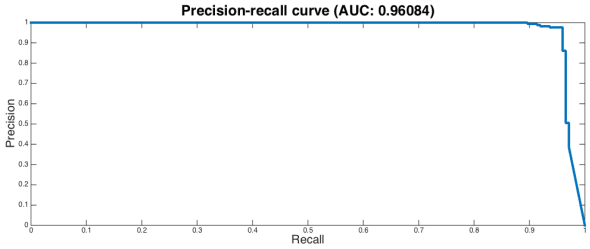



Fig. 4: Example of a precision-recall curve

$B_{gt}$) denotes the intersection of the detected and ground truth bounding boxes, and $(B_d \cup B_{gt})$ denotes their union.

For future publications, we suggest evaluating the proposed TLR systems with DAS in mind. Therefore we propose that DAS related TLR systems are evaluated on a frame-by-frame basis. Furthermore, a proposed system accuracy should be calculated using AUC on a PR curve. The proposed evaluation terms are listed below:

- True positives are defined according to the PASCAL overlap criterion.
- Precision, as seen in equation (1)
- Recall, as seen in equation (2)
- Area-under-curve on Precision-Recall curves

## VI. Concluding Remarks

We have presented an overview of four state of the art traffic light detection methods published in recent papers on traffic light recognition (TLR). Three of these method rely on color or shape information for detecting TL candidates and one rely on spotlight detection in a single intensity channel.

Because the systems are evaluated using different methodology and on very different datasets it is not clear which approaches are the best. Only one public database with TLs is currently available and, it is not widely used. We have therefore contributed with a new database, the LISA Traffic Light Database, which contains TLs captured using a stereo camera on roads in San Diego, USA under varying conditions. The database is supposed to enable comparable evaluation on a large and varied dataset, and provides the possibility of including stereo vision for improving TLR. The dataset will eventually be included in in the *VIVA Challenge* [19].

## References

[1] A. Mogelmose, M. M. Trivedi, and T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *Intelligent Transportation Systems, IEEE*, vol. 13, pp. 1484–1497, 2012.

[2] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[3] Federal Highway Administration. (2015) Manual on uniform traffic control devices. [Online]. Available: http://mutcd.fhwa.dot.gov/

[4] United Nations. (2006) Vienna convention on road signs and signals. [Online]. Available: www.unece.org/trans/conventn/signalse.pdf

[5] H.-S. L. A. P. Cheng-Chin Chinang, Ming-Che Ho and W.-C. Syu, "Traffic lights by genetic approximate ellipse detection and spatial texture layouts."

[6] U. Franke, D. Pfeiffer, C. Rabe, C. Knoeppel, M. Enzweiler, F. Stein, and R. Herrtwich, "Making bertha see," in *Computer Vision Workshops (ICCVW), IEEE*, 2013, pp. 214–221.

[7] R. de Charette and F. Nashashibi, "Real time visual traffic lights recognition based on spot light detection and adaptive traffic lights templates," in *Intelligent Vehicles Symposium, IEEE*, 2009, pp. 358–363.

[8] R. Charette and F. Nashashibi, "Traffic light recognition using image processing compared to learning processes," in *Intelligent Robots and Systems, IEEE/RSJ*. IEEE, 2009, pp. 333–338.

[9] N. Fairfield and C. Urmson, "Traffic light mapping and detection," in *Proceedings of ICRA 2011*, 2011.

[10] H.-K. Kim, Y.-N. Shin, S.-g. Kuk, J. H. Park, and H.-Y. Jung, "Night-time traffic light detection based on svm with geometric moment features," *World Academy of Science, Engineering and Technology 76th*, pp. 571–574, 2013.

[11] M. Diaz-Cabrera, P. Cerri, and P. Medici, "Robust real-time traffic light detection and distance estimation using a single camera," *Expert Systems with Applications*, pp. 3911–3923, 2014.

[12] G. Trehard, E. Pollard, B. Bradai, and F. Nashashibi, "Tracking both pose and status of a traffic light via an interacting multiple model filter," in *Information Fusion (FUSION)*. IEEE, 2014, pp. 1–7.

[13] Y. Zhang, J. Xue, G. Zhang, Y. Zhang, and N. Zheng, "A multi-feature fusion based traffic light recognition algorithm for intelligent vehicles," in *Control Conference (CCC)*. IEEE, 2014, pp. 4924–4929.

[14] A. Ruta, Y. Li, and X. Liu, "Towards real-time traffic sign recognition by class-specific discriminative features," 2009.

[15] Robotics Centre of Mines ParisTech. (2015) Traffic lights recognition (tlr) public benchmarks. [Online]. Available: http://www.lara.prd.fr/benchmarks/trafficlightsrecognition

[16] S. Sivaraman and M. M. Trivedi, "A review of recent developments in vision-based vehicle detection." in *Intelligent Vehicles Symposium*, 2013, pp. 310–315.

[17] R. Danescu, F. Oniga, and S. Nedevschi, "Modeling and tracking the driving environment with a particle-based occupancy grid," *Intelligent Transportation Systems, IEEE*, vol. 12, pp. 1331–1342, 2011.

[18] N. Buch, S. Velastin, and J. Orwell, "A review of computer vision techniques for the analysis of urban traffic," *Intelligent Transportation Systems, IEEE*, vol. 12, pp. 920–939, Sept 2011.

[19] U. Laboratory for Intelligent and Safe Automobiles . (2015) Vision for intelligent vehicles and applications (viva) challenge. [Online]. Available: http://cvrr.ucsd.edu/vivachallenge/

# Chapter 5

# Traffic Light Detection for DAS

After surveying recent research on TLR we wanted to implement our own TL detectors to test on the new LISA Traffic Light Database. The most notable detector is a state of the art learning based detector based on aggregated channel features. In addition to the learning based detector we implemented three heuristic model-based detectors inspired by experience and the surveyed research. Only two of these were included in the paper for comparison with the learning based detector. Since the LISA Traffic Light dataset contains stereo image pairs, we developed a stereo vision based candidate filter based on TL candidates' height above the road surface. This extension to our detectors was not included in the paper, but it is described and evaluated in section 5.2.

## 5.1 Traffic Light Detection: A Learning Algorithm and Evaluations on Challenging Dataset

The majority of recent work in the domain of TLD is using heuristic model-based approaches. Previous attempts at using learning-based detectors based on Haar-like features have been unsuccessful. In related areas learning-based approaches have gained traction. Examples of areas where learning-based approaches are now the state of the art are pedestrian detection and traffic sign recognition. It would therefore be interesting to apply a state of the art object detector to the TL detection problem. One learning-based and two model-based approaches have been developed and compared using the LISA Traffic Light Database. We have submitted the paper *Traffic Light Detection: A Learning Algorithm and Evaluations on Challenging Dataset*, to the IEEE Intelligent Transportation Systems Conference (ITSC), 2015, based on our findings.

Furthermore, we have been looking into utilizing stereo vision for filtering TL candidates based on their height above the road surface. This extension was not included in the paper but the system is described in section 5.2 along with the evaluation results. For any introductory theory about stereo vision, refer to section 2.2. The hardware used throughout this paper is described in appendix B. The learning-based approaches are implemented in Matlab, and the model-based and stereo-vision approaches are developed in c++ using PCL and OpenCV. For more information about the used libraries, refer to appendix C.

The paper is found on the following pages.

# Traffic Light Detection: A Learning Algorithm and Evaluations on Challenging Dataset

Mark P. Philipsen[1,2], Morten B. Jensen[1,2],
Andreas Møgelmose[1], Thomas B. Moeslund[1], and Mohan M. Trivedi[2]

*Abstract*— **Traffic light recognition (TLR) is an integral part of any intelligent vehicle, which must function in the existing infrastructure. Pedestrian and sign detection have recently seen great improvements due to the introduction of learning based detectors using integral channel features. A similar push have not yet been seen for the detection sub-problem of TLR, where detection is dominated by methods based on heuristic models.**

**Evaluation of existing systems is currently limited primarily to small local datasets. In order to provide a common basis for comparing future TLR research an extensive public database is collected based on footage from US roads. The database consists of both test and training data, totaling 48,048 frames and 90,767 annotated traffic lights, captured under a varying light and weather conditions.**

**The learning based detector achieves an AUC of around 0.5, which is more than an order of magnitude better than the two heuristic model-based detectors.**

## I. Introduction

Recognition of traffic lights (TLs) is an integral part of Driver Assistance Systems(DAS) in the transitional period between manually controlled cars and a fully autonomous network of cars. Currently the focus of research in computer vision systems for vehicles is divided in two. Major industrial research groups, such as Daimlar and Google, are investing heavily in autonomous vehicles and attempt to make computer vision based system for the existing infrastructure. Other research done by academic institutions, such as the LISA lab at UC San Diego and LaRA at ParisTech, are targeting DAS, which is already available to consumers in some high-end models. Existing commercial DAS capabilities include, warning of impending collisions, emergency breaking, automatic lane changing, keeping the advertised speed limit, and adaptive cruise control. For all parts of DAS the urban environment posses a lot of challenges, especially to the systems that rely on computer vision. One of the most important challenge here is detecting and recognizing TLs at intersections. Ideally, the TLs should be able to communicate both visually and using radio communication. However, this requires investments in infrastructure, something that is usually not a high priority.

When some form of computer controlled automation is involved with dangerous objects such as cars, safety and reliability is of utmost importance. The worst case scenarios would be a false positive from e.g. a tail light resulting in the assistance system determining that a red light is imminent when it is not the case and unnecessarily distracting the driver, or worse affecting the driver to perform an emergency braking operation. Most current research is focused on detection and recognition during day time with plenty of light, which makes it much easier to reject false positives, from e.g. tail lights, street lights and various reflections. An exception is a system proposed by Google in [1], where a prior map of the location of TLs makes it possible for their system to achieve solid performance even at night. The same system is able to reduce the number of false positives substantially when it knows where the traffic signal should, and should not be.

Inspiration for further improvements can be found by looking at research done on similar computer vision problems. For sign recognition [2], [3] explain how the focus has shifted from heuristic model-based detection to learning based approaches and the problem is considered solved on a subset of signs. The same is the case with pedestrian detection, where [4] shows how a learning based detectors based on Integral Channel Features(ICF) or the even faster and slightly better Aggregated Channel Features(ACF) outperform the other approaches. While research on sign and pedestrian detection has mostly moved on, the same is not the case for TL detection where the majority rely on some sort of color and/or shape filter for detection.

Research related to pedestrian and traffic signs have benefited greatly from high amount of public datasets made available through various benchmarks, such as the *KITTI Vision Benchmark Suite*[5] and *VIVA Challenge* [6]. Currently only one public TL dataset is available, which is the dataset published by LaRA at ParisTech. The dataset consist of 11,179 frames from a 8min and 49sec long drive in Paris. In order to provide a common basis for comparing future TLR research an extensive public database is collected based on footage from US roads captured under varying light and weather conditions. Each test sequence consists of a continuous drive in an urban environment providing lots of frames with and without TLs.

The purpose of this paper is to compare two heuristic TL detection methods to a state-of-the-art learning based detector relying on ACF. Learning based detectors relying on Haar features have been applied in earlier research [7], [8], [9], without much success. This is therefore the first successful learning based detector applied to the TL detection problem. Evaluation and comparison between the three

[1]Visual Analysis of People Laboratory, Aalborg University, 9000 Aalborg, Denmark.
[2]Computer Vision and Robotics Research Laboratory, UC San Diego, La Jolla, CA 92093-0434, USA

approaches is done on daytime sequences from the extensive and difficult LISA Traffic Light Database. The contributions are thus threefold:

1) First successful application of a state-of-the-art learning based detector for TL detection.
2) Comparison between two heuristic TL detection approaches and a learning based detector using ACF.
3) Introduce the first evaluation based on the public LISA Traffic Light Database.

The paper is organized as follows: Relevant research is summarized in section II. In section III we present the proposed methods, followed by evaluation of the TL detectors in section IV. Finally, section V rounds of with some concluding remarks.

## II. Related Work

Recent work published in the area of traffic light recognition is reviewed, before developing a traffic recognition system to be used for DAS. For a more extensive overview of the TLR domain, we refer to [10].

### A. Traffic Light Recognition

Common for [11], [9], [12] is a TL detector which relies purely on intensity from grayscale images. This has the advantage of being more robust to color distortion. Areas brighter than their surroundings are segmented using the white top-hat mophology operation, which leads to an initial high number of candidates. False candidates are filtered out based on shape information. Specifically, rejection is done based on criteria such as, dimension ratio, the BLOB being free of holes and approximately convex. Furthermore, the areas of BLOBs are compared to the areas of regions grown from extrema in the original grayscale image. This is especially effective for removing false candidates big bright areas such as the sky. This detector relies heavily on a competent classifier for further rejection and state estimation, since the number of false candidates is very high and color information is not available. The detector manages to find 90% of all TLs in the testset.

[13] begins by detecting the vanishing line and thereby reducing the search area considerably, relying on the assumption that TLs will only appear above this line. They then apply the the white top-hat operation as [11], [9], [12] did, on the intensity channel V from a HSV image. What is left is filter based on statistical measurements of the hue and saturation ranges of red and green lights. All pixels outside these ranges are rejected while the remaining pixels are selected as candidates. Remaining BLOBs are filtered based on size and height-width ratio. They then look for black bounding boxes around the BLOBs based on gradient information and the blackness of the inside of box candidates. Their system reaches an accuracy of 85%.

[14] extracts candidate BLOBs from RGB images by applying a color distance transform proposed in [15]. The transform emphasizes the chosen color in an intensity image, which is thresholded to remove to suppressed colors. This is followed by shape filtering to reduce noise using width/height ratio and the solidity of BLOBs. The solidity is calculated based on the ratio between the area of the BLOB and it's bounding box. When evaluating their system, they count a success if the TL was detected just once in the sequence, this allows them to reach a detection rate of 93,53%.

## III. Methods

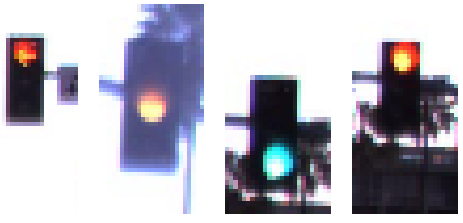
In this section all of the methods which are used in the proposed system are presented. The section is divided into two subsection. In the first subsection the learning based detector is described. The second subsection explains the tracking used for improving the output of the detector.

### A. Learning based detection

In this subsection we apply the successful ACF detector to the TL detection problem. The learning based detection is similar to the approach seen in [16] for traffic signs. We use the Matlab toolbox provided by [17]. The learning based detection system is described in the following three parts:

*1) Features:* The learning based detector is based on features from 10 channels as described in [18]. A channel refers to different representations of the input image. The 10 different channels include 6 gradient histogram channels, 1 for unoriented gradient magnitude, and 3 for the channels in the CIE-LUV color space. First-order Haar-like features are used for all channels, which basically are summations of rectangular image regions in the channels. This procedure is speeded up using the widely known integral image. Finally, the features are evaluated using a modified AdaBoost classifier with depth-2 decision trees as weak learners.

*2) Training:* Training is done using 14,106 positive TL samples with a resolution of 20x40 and 42,125 negative samples from 200 carefully selected frames without TLs. In Figure 1 four examples of the positives used for the learning based detector are seen. Similarly, Figure 2 shows two examples of frames used for negatives.

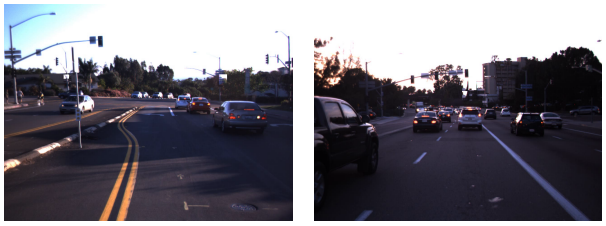


(a) (b) (c) (d)

Fig. 1: Positive samples for learning based detector.

The classifier is trained with Adaboost based on the features extracted from the positive samples. We train 4 cascade stages, 1st stage consists of 10 weak learners, 2nd stages of 100, 3rd stage of 1000, and 4th stage of 20000. In the 4th stage, the training algorithm convergent at 3136 weak learners.

*3) Detection:* We use a 20x40 sliding window across an integral image of each of the 10 channels in the test image.



(a) (b)

Fig. 2: Negative samples for learning based detector.

### B. Heuristic model based detection

We want to compare the learning based detector to more conventional detectors based on heuristic models. The first approach is based on back projection of trained color histograms of the three TL colors. The second approach is purely relying on intensity information for spotlight detection.

*1) Detection by Back Projection:* Back projection begins with the generation of color distribution histograms. These histograms are created from 10 specifically selected training samples for each color, green, yellow, and red. Based on the U and V channels of the LUV color space a 2D histogram is created for each of the colors. The histograms are min-max normalized before they are used for back projection. The resulting back projection is thresholded to remove low probability pixels. TLs are found using BLOB analysis, and size, shape information is used to generate confidence scores for each BLOB. The specific metrics are listed here:

- Ratio between width and height of bounding box
- Mean value inside bounding box in the back projection image
- Mean value inside bounding box in the intensity image
- Ratio between area of floodfilled BLOB and area of bounding box

*2) Detection by Spotlight Detection:* Spotlights are found in the intensity channel L from the LUV colorspace using the white top-hat morphology operation. This method has been used in a significant fraction of recent TLR papers [11], [9], [12], [13], [19]. The found spotlight are scored based on the listed metrics.

- Ratio between width and height of bounding box
- Ratio between the convex area of BLOB and area of bounding box
- Ratio between area of floodfilled BLOB and area of bounding box

## IV. Evaluation

The systems are evaluated based upon the following five criteria:

- True positives are defined according to the PASCAL overlap criterion.
- Precision, as seen in equation (1)
- Recall, as seen in equation (2)
- Area-under-curve on Precision-Recall curves

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Precision is the ratio of correct TL detections compared to the actual number of TLs.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Recall is the ratio of correct TL detections compared to the total number of detections.

For presenting and evaluating the overall system performance, we use a precision-recall curve and using the area-under-curve (AUC) as measure. A high AUC indicates good performance, an AUC of 100% indicates a perfect system for the testset.

All systems are evaluated on the two test day sequences from the LISA Traffic Light Database[1]. This provides a total of frame number of 14,386, and a total ground truth of 21,421 annotated TLs. Additional information of the video sequences can be found in Table I. The resolution of the LISA Traffic Light Database is 1280x960. We however only use the upper half of the images, which gives a system evaluation time of an average 0.67 seconds per frame.

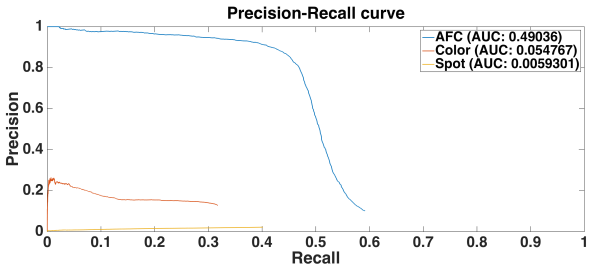



Fig. 3: Precision-Recall curve of day sequence 1.

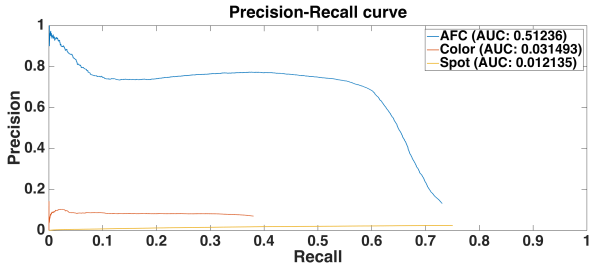



Fig. 4: Precision-Recall curve of day sequence 2.

In Figures 3 and 4 show the precision-recall curves for day sequences 1 and 2. From Figure 3 it is clear that the learning based detector far outperforms the other detectors in both precision and recall when evaluated on sequence 1. The same is also seen in Figure 4, though the spotlight approach reach a higher recall than both ACF and color back projection. This however comes with very low precision because the detector detects a large number of false positives. From both

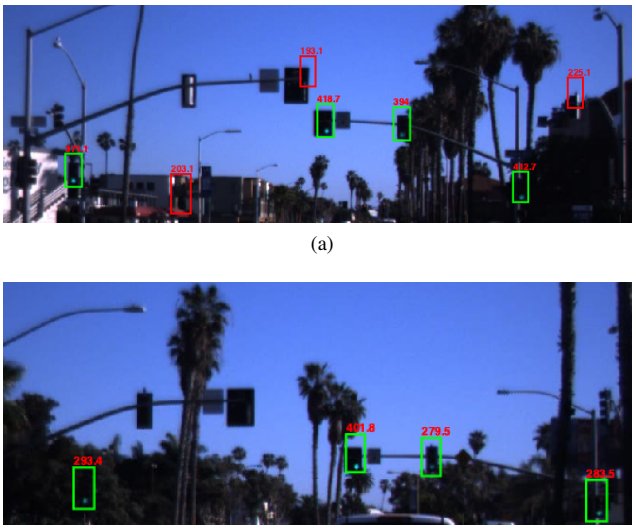
[1]Freely available at `http://cvrr.ucsd.edu/LISA/datasets.html` for educational, research, and non-profit purposes.

TABLE I: Overview of the daytime test sequences in LISA Traffic Light Database.

| Sequence name | Description | # Frames | # Annotations | # TLs | Length |
|---|---|---|---|---|---|
| Day sequence 1 | morning, urban | 4,800 | 10,267 | 25 | 5.00 min |
| Day sequence 2 | evening, urban | 9,586 | 11,154 | 29 | 6.10 min |
| | | 14,386 | 21,421 | 54 | 11.1 min |

Figure 3 and 4 we can also see that the confidence metrics defined in subsection III-B for the model-based detectors are bad at discriminating between TLs and non TL spotlights. It is apparent that for especially the spotlight detector false candidates obtain a better score than actual TLs.

In Figure 5 two detection images from the learning based system is seen. The green bounding box is the positive detected TLs, and the red bounding box is false positives. The true positive detected TLs have a score around 400, and the false positives have a score around 200 making it easy to discard them.




(a)




(b)

Fig. 5: Detections by the learning based detector.

## V. Concluding Remarks

We have compared a learning based detector based on aggregated channel features to two detectors based on heuristic models. The learning based detector reached the best AUC, because of the high precision, while the spotlight detection achieved the highest recall a the cost of a very low precision. For detectors recall is usually the most important parameter, since many of the false positives can be removed in later stages, but false negatives are lost for good. The learning based detector achieves an AUC of around 0.5, which is more than an order of magnitude better than the two heuristic model-based detectors.

On top of the detectors we would like to implement tracking to reduce the number of false positives and false negatives. Stereo vision could be used to filter out false positives by looking at the detected TL candidates' height above the road surface as well as their size and shape. 3D information can also be used to improve tracking precision.

## References


[1] N. Fairfield and C. Urmson, "Traffic light mapping and detection," in *Proceedings of ICRA 2011*, 2011.

[2] A. Mogelmose, D. Liu, and M. M. Trivedi, "Traffic sign detection for us roads: Remaining challenges and a case for tracking," in *Intelligent Transportation Systems (ITSC), IEEE*. IEEE, 2014, pp. 1394–1399.

[3] M. Mathias, R. Timofte, R. Benenson, and L. Van Gool, "Traffic sign recognition - how far are we from the solution?" in *ICJNN*, 2013.

[4] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *PAMI*, 2014.

[5] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[6] Laboratory for Intelligent and Safe Automobiles UC San Diego. (2015) Vision for intelligent vehicles and applications (viva) challenge. [Online]. Available: http://cvrr.ucsd.edu/vivachallenge/

[7] U. Franke, D. Pfeiffer, C. Rabe, C. Knoeppel, M. Enzweiler, F. Stein, and R. Herrtwich, "Making bertha see," in *Computer Vision Workshops (ICCVW), IEEE*, 2013, pp. 214–221.

[8] F. Lindner, U. Kressel, and S. Kaelberer, "Robust recognition of traffic signals," in *Intelligent Vehicles Symposium, IEEE*, 2004, pp. 49–53.

[9] R. Charette and F. Nashashibi, "Traffic light recognition using image processing compared to learning processes," in *Intelligent Robots and Systems, IEEE/RSJ*. IEEE, 2009, pp. 333–338.

[10] M. P. Philipsen, M. B. Jensen, M. M. Trivedi, A. Møgelmose, and T. B. Moeslund, "Vision for looking at traffic lights: Issues, survey, and perspectives," in *Intelligent Transportation Systems, IEEE Transactions [In submission]*. IEEE, 2015.

[11] G. Trehard, E. Pollard, B. Bradai, and F. Nashashibi, "Tracking both pose and status of a traffic light via an interacting multiple model filter," in *Information Fusion (FUSION)*. IEEE, 2014, pp. 1–7.

[12] R. de Charette and F. Nashashibi, "Real time visual traffic lights recognition based on spot light detection and adaptive traffic lights templates," in *Intelligent Vehicles Symposium, IEEE*, 2009.

[13] Y. Zhang, J. Xue, G. Zhang, Y. Zhang, and N. Zheng, "A multi-feature fusion based traffic light recognition algorithm for intelligent vehicles," in *Control Conference (CCC), 2014 33rd Chinese*. IEEE, 2014.

[14] H.-K. Kim, Y.-N. Shin, S.-g. Kuk, J. H. Park, and H.-Y. Jung, "Night-time traffic light detection based on svm with geometric moment features," *World Academy of Science, Engineering and Technology 76th*, pp. 571–574, 2013.

[15] A. Ruta, Y. Li, and X. Liu, "Towards real-time traffic sign recognition by class-specific discriminative features," 2009.

[16] A. Mogelmose, D. Liu, and M. Trivedi, "Traffic sign detection for u.s. roads: Remaining challenges and a case for tracking," in *Intelligent Transportation Systems (ITSC), IEEE*, 2014, pp. 1394–1399.

[17] P. Dollár, "Piotr's Computer Vision Matlab Toolbox (PMT)," http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html.

[18] P. Dollár, Z. Tu, P. Perona, and S. Belongie, "Integral channel features," in *BMVC*, vol. 2, 2009, p. 5.

[19] D. Nienhuser, M. Drescher, and J. Zollner, "Visual state estimation of traffic lights using hidden markov models," in *Intelligent Transportation Systems (ITSC), IEEE*. IEEE, 2010, pp. 1705–1710.

## 5.2 Stereo Vision Extension

In addition to the learning- and model-based approaches, we also looked into utilizing stereo vision for rejecting false TL candidates. This did not make it into a paper but could be useful in future TLR systems. The system system rejects candidates outside a feasible height above the road. Moreover, utilizing stereo vision enables candidate filtering based on 3D shape and context, something which would be interesting to investigate further. The stereo extension is integrated with each of the two model based detectors described in *Traffic Light Detection: A Learning Algorithm and Evaluations on Challenging Dataset*. Figure 5.1 shows the composition our preliminary stereo vision aided TL detectors.

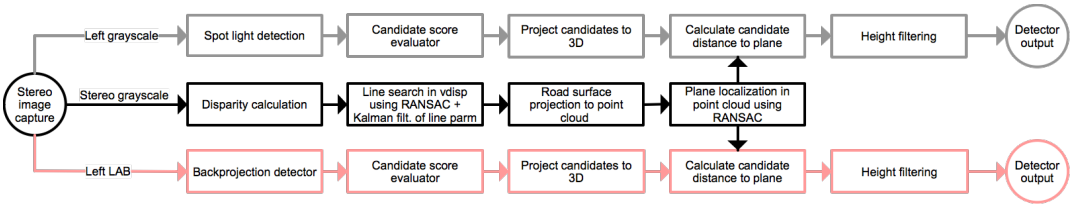
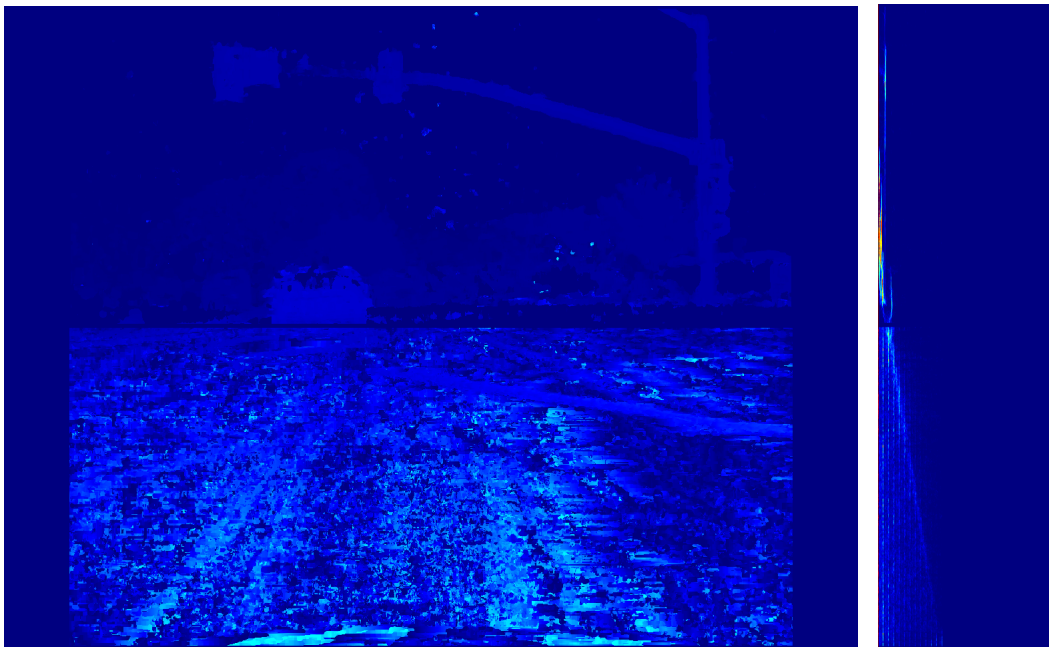



**Figure 5.1:** System composition of stereo vision aided TL detector.

The spotlight and backprojection detectors as well as their candidate evaluators are described in the paper. Additionally, the techniques behind the detectors are explained in the theory chapter 2. What remains is the stereo extension which is described below.

### 5.2.1 Disparity map generation

The introduction of depth information is in this case done using a disparity map generated from stereo image pairs. The stereo correspondences are found using OpenCV's GPU implementation of the block matching algorithm proposed by K. Konolige's BM in [24]. We use different parameter for calculating the top and bottom half of the input image pairs. This is done because the bottom half is used for road surface localization where a relatively dense disparity map is necessary, since the road surface is mostly uniform we avoid using prefiltering and strict matching criteria. For the upper half is usually less uniform therefore the filtering and matching criteria can be more strict without losing too much information. Figure 5.2a shows the upper and lower parts of the disparity map combined in one.

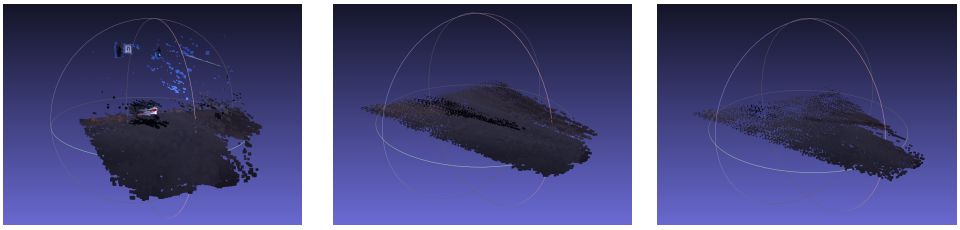


**(a)** Combined upper and lower disparity map. **(b)** v-disp.

### 5.2.2 Road surface localization

For determining the TLs' height above the road surface the road surface is located in the point cloud, but only after getting a rough localization in the v-disparity and using it to remove potentially distracting measurements. The v-disparity is calculated from the disparity map on the GPU as it is described in appendix B. Figure 5.2b shows the v-disparity the disparity map.

In the v-disparity the most prominent line is found using RANSAC. In cases where a satisfactory line cannot be found, the latest good line is used. Additionally, the line parameters are filtered using a Kalman filter to smooth out faulty road surface estimations. The calculated line is then used as a threshold for determining if disparity pixels belongs to the road surface. What remains is a disparity map containing only disparity pixels from the road surface and objects above it. These remaining disparity pixels if projected into 3D, would result in the point cloud seen in Figure 5.3. Since we assume that the road surface is located in the bottom half of the image, only pixels from this region are projected to 3D. The result can be seen in Figure 5.3b. The ground plane is estimated by running RANSAC on the point cloud, resulting in a plane consisting of the points seen in Figure 5.3c.



(a) (b) (c)

**Figure 5.3:** Visualization of point clouds: (a) Full scene, only for scene understanding. (b) Lower half of scene, used for the actual road surface localization. (c) Resulting best estimate of road surface.

### 5.2.3 Projecting candidates to 3D and distance to plane

When TL candidates are detected their ROI is projected to 3D and the voxel, in this small point cloud, which is closest to the camera is used as the 3D coordinate for the TL candidate. The distance between this point and the localized road surface is then found.

### 5.2.4 Height filtering

Based on the found height candidates are filtered by only keeping TL candidates which are between 2 and 8 meters above the road surface. Figure 5.4 shows the detected TL candidates and their height above the road surface. The size of the circles around detections signifies the detector confidence. The lines from the detections shows the length from the detection to the projection onto the road surface.

**Figure 5.4:** Blue: spotlight detections, red: histogram back-projection detections, purple: detetions by both spotlight and backprojection. Lines from detections to road surface illustrates the height of detected TL candidates.

## 5.3 Results

Figure 5.5 shows the height filtered TL candidates.

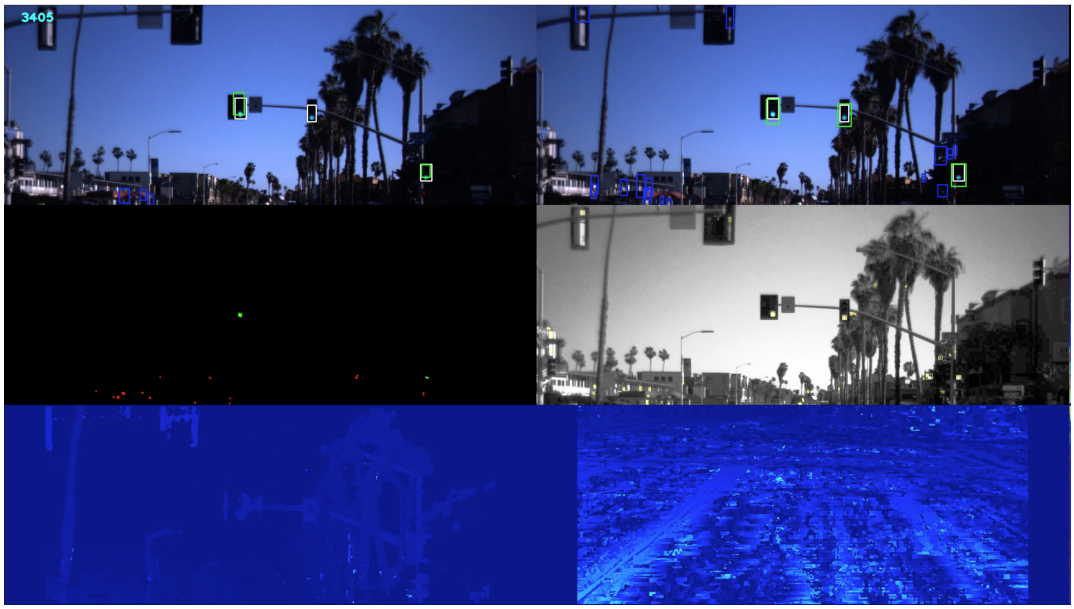



**Figure 5.5:** Results(first row): Back-projection(left), Spotlight(right). Raw detector output(middel row): Back-projection(left), Spotlight(right). Disparity map(bottom row): Top half(left), Bottom half(right).

Figure 5.6 and 5.7 show the result of adding the stereo extension to the two heuristic detectors.

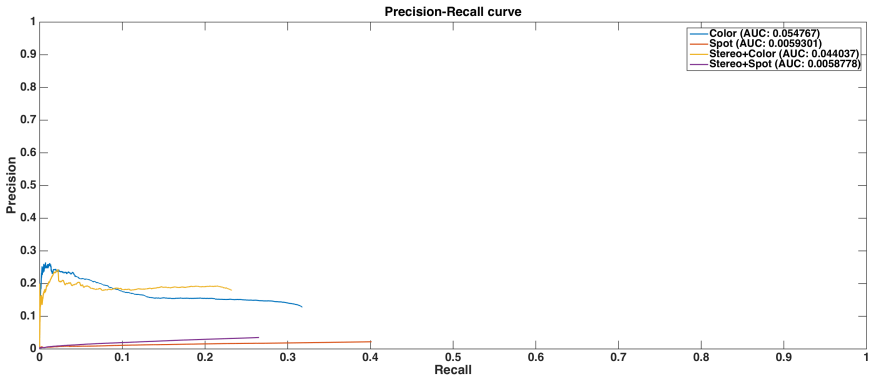



**Figure 5.6:** PR curve comparing regular detectors to detectors with the stereo extension.

It is clear that the added height criterion removes a significant part of the FPs but it also rejects a substantial amount of actual TLs.

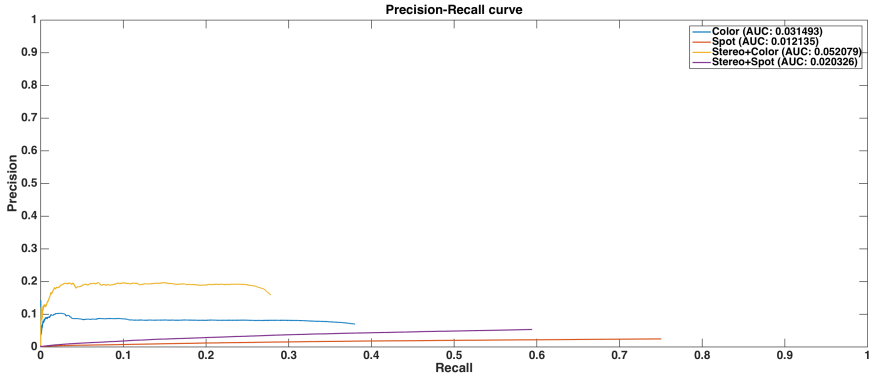



**Figure 5.7:** PR curve comparing regular detectors to detectors with the stereo extension.

# Chapter 6

# Database

Image data is fundamental for research in computer vision. Since insufficient data is publicly available it was necessary to capture several hours of video data for this master thesis. Each of the four papers which were submitted during the thesis period relies on our own data in varying degrees. In this chapter we present each datasets and the structure of the accompanying annotations as well as the setup for acquiring the data.

## 6.1 Datasets

The datasets are mostly organized and annotated using a rather simple structure. This structure was used in an existing vehicle rear end dataset, which we were to extend. Since our annotation tool was build with this structure in mind, we kept it when later annotating TLs. However, since our TL dataset is supposed to be released in the VIVA challenge, the accompanying annotations will also be available in a format which is in line with the annotations for the VIVA traffic sign database. It is preferable to streamline datasets in the challenge, since it will enable tool reuse and compatibility.

The simple annotation structure which is stored in txt files exemplified in listing 6.1. Each frame which contain annotations gets a line beginning with the frame number from the video sequence. Values are tab-separated so after the first tab comes the number of annotation in the frame, followed by the tab-separated coordinates for the upper left corner of the first annotation and the width and height of the annotation. If there is more than 1 annotation in the frame, the coordinates and dimensions are inserted one after the other. This simplified annotation structure, which does not take TL class into consideration, is intended to keep the complexity of our annotation tool down. Similarly

we wanted to keep the data collected in a few files. Therefore frames are not extracted into separate image files, but are kept in video containers.

```
1 frame     N annotations  Upper left x-coord1 Upper left y-coord1 Width1
    Height1 ... Upper x-coordN    Upper left y-coordN WidthN    HeightN
```

**Listing 6.1:** Iterator(safe) method for copying image elements to 1D array.

Since the TLR dataset is meant to be published in the VIVA challenge, we adapt the more complex structure which is already used for the LISA traffic sign database. Here the annotations are stored as 1 annotation per line with the addition of information such as class tag and file path to individual image files. With this structure the annotations are stored in a csv file which is compatible with the tools made for the traffic sign datasbase. The structure of the comma separated files are exemplified in listing 6.2.

```
1 Filename;Annotation tag;Upper left corner X;Upper left corner Y;Lower right
    corner X;Lower right corner Y;Occluded,On another road;Origin file;Origin
    frame number;Origin track;Origin track frame number
```

**Listing 6.2:** Iterator(safe) method for copying image elements to 1D array.

The database is mostly captured in San Diego, California, USA. A single clip used for NDS is captured in Las Vegas, Nevada, USA. The stereo image pairs are acquired using the Point Grey's Bumblebee XB3 (BBX3-13S2C-60) which is constructed with three lenses which each capture images with a resolution of 1280 x 960. Each lens has a with a horizontal Field of View(FoV) of 66°. The stereo camera supports two different baselines, 12 and 24 cm, whereof a baseline of 24 cm is used for the LISA TL database. The stereo images are uncompressed and rectified on the fly. The Bumblebee XB3 is mounted in the center of the roof of the capturing vehicle and is connected to a laptop through the IEEE-1394b interface, sometimes refereed to as FireWire 800.

A Triclops calibration file is provided along with the database, this file contains the factory calibration of our Bumblebee XB3 camera. This calibration is provided for use with the Point Grey's Triclops SDK, but is unnecessary for most purposes since the image pairs are provided in rectified state. The focal length in pixels of the camera is calculated[41] to be 1612.77 pixels in equation (6.1) and the wide baseline is 0.239813 meters.

$$1.25998 * 1280 pixels = 1612.77 pixels \tag{6.1}$$

The collected database is divided into three parts, each for a specific purpose:

### 6.1.1 Rear ends of vehicles at night

As part of ongoing research in the lab, we were tasked with collecting and annotating rear ends of vehicles at night, primarily on highways. This served as additional data for training and testing a monocular detector for nighttime detection of rear-end of vehicles on highways. In this process we selected 8,922 frames of which 5,069 were annotated with 9,761 annotations of rear ends. To do this we began the development of our own annotation tool. Figure 6.1 shows the improvements the Ravi's monocular detector caused by the addition of our training data.

**Figure 6.1:** White squares are GT, red are detections from detector trained on our data, yellow are detections from detector trained on other data.

### 6.1.2 Intersecting vehicles at day and night-time

The dataset for evaluating stereo vehicle detection for NDS events during both day and nighttime, is recorded with the purpose of capturing vehicles and road surfaces while preserving as much information as possible. Therefore the camera shutter speed was set to be as long as possible(max. 1/16 sec.) and gain was set medium-high at 8 when recording at night. For day-time recordings the shutter speed was adjusted to let in the maximum amount of light, while avoiding overexposure, gain was set low. The video clips are primarily recorded leading up to intersections and while stopped at red lights, which is where most of relevant NDS event takes place. This dataset is unannotated and evaluation must therefore be done under manual supervision. Evaluation was done on 13 day clips totaling 1269.9 MB and 16 night totaling 856.7 MB, this translates in to roughly 18 minutes of video.

### 6.1.3 LISA Traffic Light Database

This is the most prominent of our collected databases. It contributes waste amounts of annotated stereo imaging for training and testing TLR systems. The database contains two day and two nighttime sequences for testing. These test sequences amounts to a total of 23 minutes and 9 seconds of driving in Pacific Beach, San Diego. The long continuous test sequences serves to give a realistic understanding of the performance of TLR systems, since they must function well hen multiple TLs are visible at widely different distances as well as when no TL is insight. Besides the four test sequences, we provide 18 shorter video clips consisting of TLs collected in the northern part of San Diego. These are intended for training and initial testing. They are organized as seen in Table 6.1, which gives a detailed overview of all the annotated video sequences and clips. Camera gain and shutter speed were manually set to avoid over exposure as well as limit flickering of the TLs. For all day-time recordings, a shutter speed of 1/5000 sec and a gain of 0 was used. For all night-time recordings, a shutter speed of 1/100 sec and a gain of 8 was used.

#### Publication

The LISA Traffic Light Database is made public at `http://cvrr.ucsd.edu/LISA/datasets.html` and in the second half of 2015 it will be integrated as part of the second iteration of the VIVA challenge which will be located at `http://cvrr.ucsd.edu/vivachallenge/`.

**Table 6.1:** Overview of the video sequences in LISA Traffic Light Database.

| Sequence name | Description | # Frames | # Annotations | # TLs | Length | Classes |
|---|---|---|---|---|---|---|
| Day seq. 1 | morning, urban, backlight | 4,800 | 10,267 | 25 | 5min | Go, warning, warning left, stop, stop left |
| Day seq. 2 | evening, urban | 9,586 | 11,154 | 29 | 6min 10s | Go, go forward, go left, warning, stop, stop left |
| Night seq. 1 | night, urban | 4,992 | 18,889 | 25 | 5min 11s | Go, go left, warning, stop, stop left |
| Night seq. 2 | night, urban | 6,533 | 23,776 | 54 | 6min 48s | Go, go left, warning, stop, stop left |
| Day clip 1 | evening, urban, lens flare | 2,161 | 6,474 | 10 | 2min 15s | Go, stop |
| Day clip 2 | evening, urban | 1,031 | 2,230 | 6 | 1min 4s | Go, go left, warning left, stop, stop left |
| Day clip 3 | evening, urban | 643 | 1,087 | 3 | 40s | Go, warning, stop |
| Day clip 4 | evening, urban | 397 | 859 | 8 | 24s | Go |
| Day clip 5 | morning, urban | 2,667 | 9,717 | 8 | 2min 46s | Go, go left, warning, warning left, stop, stop left |
| Day clip 6 | morning, urban | 468 | 1,215 | 4 | 29s | Go, stop, stop left |
| Day clip 7 | morning, urban | 2,718 | 8,189 | 10 | 2min 49s | Go, go left, warning, warning left, stop, stop left |
| Day clip 8 | morning, urban | 1,040 | 2,025 | 8 | 1min 4s | Go, go left, stop, stop left |
| Day clip 9 | morning, urban | 960 | 1,264 | 4 | 59s | Go, go left, warning left, stop, stop left |
| Day clip 10 | morning, urban | 48 | 109 | 4 | 3s | Go, stop |
| Day clip 11 | morning, urban | 1,052 | 1,268 | 6 | 1min 5s | Go, stop |
| Day clip 12 | morning, urban | 152 | 229 | 3 | 9s | Go |
| Day clip 13 | evening, urban | 693 | 873 | 8 | 43s | Go, warning, stop |
| Night clip 1 | night, urban | 591 | 1,885 | 8 | 36s | Go |
| Night clip 2 | night, urban | 2,299 | 4,205 | 25 | 2min 24s | Go, go left, warning, stop, stop left |
| Night clip 3 | night, urban | 1,051 | 1,476 | 14 | 1min 6s | Go, go left, warning left, stop, stop left |
| Night clip 4 | night, urban | 1,104 | 2,538 | 9 | 1min 9s | Go, warning, stop |
| Night clip 5 | night, urban | 1,453 | 3,242 | 19 | 1min 31s | Go, go left, warning, stop, stop left |
| | | 46,418 | 112,971 | 290 | 44min 24s | |

## 6.2 Video Annotator

For annotating the large number of video captured, a semi-automatic video annotation tool has been developed. Originally the annotation tool was developed to annotate vehicle's rear-end as this was the first task during this project. This only required a few main functions of the tool: Annotate a video on a frame-by-frame basis, use left and right mouse click to mark a rectangular area in the frame, and finally convert all the annotation into a common syntax and print them to an annotation file. Later the annotation tool was developed further and optimized towards annotating traffic lights. For increasing the annotation time, the color-based tracking method meanshift and it's extension, CAMSHIFT, were used to predict the position of a given annotation in the next frame. Furthermore, the tool utilizes the middle mouse button for creating a rectangular box based on the flood filling algorithm. Such that by moving the mouse over a lit traffic light followed by a middle mouse click.

To start annotating a video sequence, one must compile the c++ program, by using the CMakeLists.txt. When this is done, the annotation program is started using the *annotateVideo* executable. The first argument for *annotateVideo* is the path to the video sequences. The second argument is from which frame the user wish to start annotating from. An example of this is seen in Figure 6.2, where a few frames have been annotated and the string that is outputted to

```
Mortens-MacBook-Pro:src Morten$ ./annotateVideo dayClip2Shutter0.000800-Gain-0.mov 0

This is a tool developed for annotating rectangles in a video sequence.
You select an area by clicking left in the frame where you wish the top left corner of the rect to be.
This is follwed by a right click for lower right corner of the rect.
Additionally, the middle button can be used in the center of a color to use floodfill to create a rectangle around it.
Usage:
   ./annotateVideo [Direction to video to annotate] [Start annotation number]


Hot keys:
        ESC or q- quit the program
        s - Save annotated andApply CAMSHIFT on previous rects in current frame.
        z or i - If area is not good. 'z' or 'i' can be used for ignoring it.
        d - Save to file, and proceed to next frame.
Naming frames from: 0
init done
opengl support available
String saved to file: 0 3       764     248     10      10      910     246     10      11      1226    338     12      10
Annotated frame was: 0
String saved to file: 1 3       766     252     4       4       911     246     9       11      1228    339     9       8
Annotated frame was: 1
String saved to file: 2 3       765     250     7       7       910     246     10      11      1226    338     11      9
Annotated frame was: 2
String saved to file: 3 3       765     250     5       7       910     246     10      11      1226    338     11      9
Annotated frame was: 3
String saved to file: 4 3       765     250     7       7       910     246     10      11      1226    337     11      10
Annotated frame was: 4
String saved to file: 5 3       766     252     5       4       910     246     10      11      1226    338     10      9
Annotated frame was: 5
```

**Figure 6.2:** Syntax for starting the annotation tool and annotation prints in terminal.

the annotation file is printed in the terminal.
An annotation proposed by either prediction, manually annotation with left and right click, or flood filling annotation is marked with a purple outline in the frame, and as seen with the keyboard shortcuts in Table 6.3, the current proposed annotation can be accepted or rejected. This will result in respectively, a green outline or black outline of the proposed annotation as seen in Figure 6.3. In same figure, a small window is visible in upper left corner. This window contain a small zoomed area around the mouse position, such it becomes easier to be more accurate while annotating manually.
When all the desired objects are annotated in a frame, the user can either save annotation and proceed to next frame by pressing 'd', save annotation; and proceed and predict annotations in next frame by pressing 's', or finally quit the program by pressing 'q'. If s is pressed, the tool will proceed to next frame and draw white rectangles on the frame which represent the predicted rectangle based on previous annotations. These are accepted or rejected by sequentially marking them with a purple color and either accepting or rejecting them, which will make them green or black. After this the user can reannotate objects with the mouse if some of the predicted annotations are not accurate, an then proceed to next frame by using same procedure as just described.
Finally when all the frames have been annotated, the tool will close down and an annotation file can be found in the annotation tool directory.
The frames used for training and testing a monocular classifier for nighttime detection of rear-end of vehicles on highways.

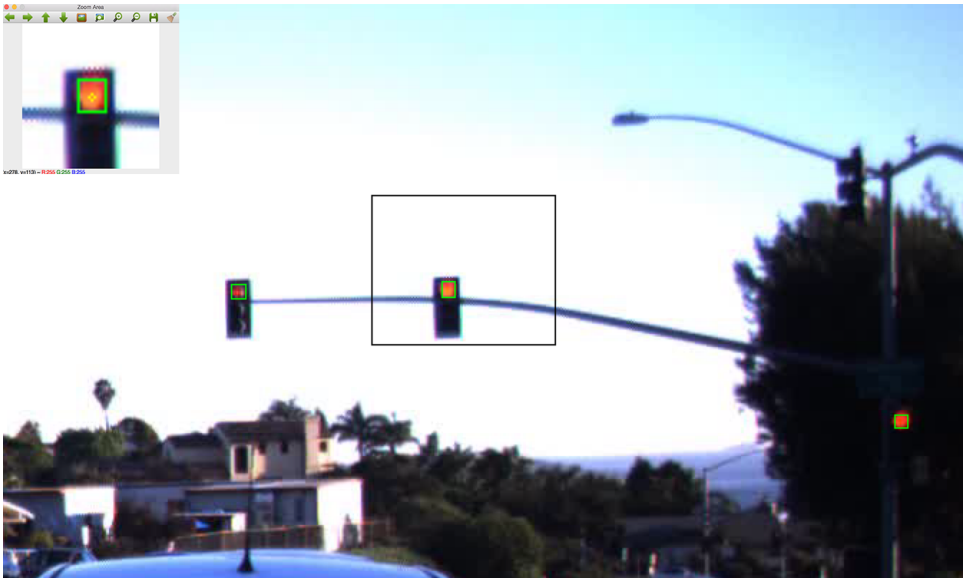


**Figure 6.3:** Three saved annotation and one rejected annotation.

**Table 6.2:** Keyboards shortcuts for video annotator tool.

| Operation | Shortcut |
|---|---|
| Save current annotation(s) and proceed to next frame | d |
| Save current annotation(s), proceed to next frame, and apply CAMSHIFT to previous annotations in current frame | s |
| Ignore selected area | z |
| Ignore selected area | i |
| Quit the program | ESC |
| Quit the program | q |

**Table 6.3:** Mouse usage for video annotator tool.

| Operation | Shortcut |
|---|---|
| Mark top left corner of the rectangle | Left click |
| Mark bottom right corner of the rectangle | Right click |
| Use flood fill in center of color to create rectangle around it | Middle click |

# Chapter 7

# Conclusion

This master's thesis revolves around computer vision and machine learning applied to solve problems in the traffic scene. Specifically problems and challenges related to intersections as mentioned in the scope of the project in section 1.2. Most of the work have been carried out during our 7-month stay with the Laboratory of Intelligent & Safe Automobiles (LISA), and the Computer Vision and Robotics Research Laboratory (CVRR) at University of California, San Diego (UCSD). Our work has been research oriented rather than the traditional problem oriented approach affiliated with previous semesters at Aalborg University. The report structure is therefore more loose with rather independent chapters.

The main theme in this report has been traffic light detection (TLD), but the report has also included the use of stereo vision for vehicle detection at intersections with the purpose of automatic data reduction for Naturalistic Driving Studies (NDS). The work resulted in four papers, whereof one has been been accepted and the remaining three are submitted for review.

A major problem for TLD compared to other traffic related problems, such as traffic sign detection, has been the absence of a proper overview of the current traffic light recognition (TLR) research. To address this problem, a comprehensive journal survey paper has been compiled containing an overview of state of the art research. From the review of existing literature it was clear that there is a need for a large public traffic light dataset with mixed weather and light conditions. It is also nessesary to establish a standardized procedure for evaluating TLD systems, in order to facilitate comparison of methods. Therefore the LISA Traffic Light Database has been captured and features more than 110,000 annotations of traffic lights captured in stereo under both day and night conditions. The dataset is captured on southern Californian roads, and contains full stereo video tracks making tracking and detection using stereo

vision algorithms possible.

Finally, several traffic light detectors have been developed featuring two heuristic model based, one learning based. Additionally a stereo vision based extension have been tested for rejecting false traffic light candidates based on their height above the road surface. A comparative analysis of the detectors is done and shows that our learning based approaches outperforms our model based approaches.

# Bibliography


[1] Chris Bahnsen. Thermal-visible-depth image registration. Master's thesis, Aalborg University, Denmark, 2013.

[2] Chris Bahnsen. Stereo vision. University Lecture, 2014. Course: Robot Vision.

[3] BBC. TED 2015: Google boss wants self-drive cars 'for son'. http://www.bbc.com/news/technology-31931914, March 2015. Accessed: 2015-05-25.

[4] Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[5] J. Y. Bouguet. Camera calibration toolbox for Matlab. http://www.vision.caltech.edu/bouguetj/calib_doc/, 2008. Accessed: 2015-02-09.

[6] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning Opencv, 1st Edition.* O'Reilly Media, Inc., first edition, 2008.

[7] Gary R Bradski. Computer vision face tracking for use in a perceptual user interface. 1998.

[8] H. D. Cheng, X. H. Jiang, Y. Sun, and Jing Li Wang. Color image segmentation: Advances and prospects. *Pattern Recognition*, 34:2259–2281, 2001.

[9] Ana Claudia Oliveira de Melo, Ronei Marcos de Moraes, and Liliane dos Santos Machado. Gaussian mixture models for supervised classification of remote sensing multispectral images. In *Progress in pattern recognition, speech and image analysis*, pages 440–447. Springer, 2003.

[10] César de Souza. Haar-feature object detection in c, 2014.

[11] Piotr Dollár. Piotr's Computer Vision Matlab Toolbox (PMT). http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html.

[12] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. *PAMI*, 2014.

[13] Piotr Dollár, Ron Appel, and Wolf Kienzle. Crosstalk cascades for frame-rate pedestrian detection. In *Computer Vision–ECCV 2012*, pages 645–659. Springer Berlin Heidelberg, 2012.

[14] Piotr Dollár, Serge Belongie, and Pietro Perona. The fastest pedestrian detector in the west. *BMVC*, 2(3):7, 2010.

[15] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. *BMVC*, 2(3):5, 2009.

[16] G. Dubbelman, W. van der Mark, J.C. van den Heuvel, and F.C.A. Groen. Obstacle detection during day and night conditions using stereo vision. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 109–116, 2007.

[17] Federal Highway Administration. Traffic control devices: Uses and misuses, 2009.

[18] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *Intelligent Vehicles Symposium (IV)*, 2011.

[19] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[20] H. Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 807–814 vol. 2, June 2005.

[21] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):328–341, Feb 2008.

[22] H. Hirschmuller and D. Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(9):1582–1599, 2009.

[23] Cambridge in Colour. Digital camera image noise, 2015.

[24] Kurt Konolige. Small vision systems: Hardware and implementation. In *Robotics Research*, pages 203–212. Springer London, 1998.

[25] R. Labayrade, D. Aubert, and J.-P. Tarel. Real time obstacle detection in stereovision on non flat road geometry through "v-disparity" representation. In *Intelligent Vehicle Symposium, 2002. IEEE*, volume 2, pages 646–651 vol.2, 2002.

[26] M. Mathias, R. Timofte, R. Benenson, and L. Van Gool. Traffic sign recognition - how far are we from the solution? In *ICJNN*, 2013.

[27] F Meyer. *Contrast features extraction, Special Issues of Practical Metallography, Vol. 8*. Riederer Verlag GmbH, Stuttgart, 1977.

[28] Thomas B. Moeslund. *Introduction to Video and Image Processing - Building Real Systems and Applications*. Undergraduate Topics in Computer Science. Springer, 2012.

[29] Andreas Mogelmose, Dongran Liu, and Mohan M Trivedi. Traffic sign detection for us roads: Remaining challenges and a case for tracking. In *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, pages 1394–1399. IEEE, 2014.

[30] Alexander Mordvintsev and Abid K. Meanshift and camshift. http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_video/py_meanshift/py_meanshift.html. Accessed: 2015-06-02.

[31] Andreas Møgelmose. Advanced Detection. HOG, ICF and rea-lworld use. Slides. Accessed: 2015-05-04.

[32] OpenCV. About OpenCV. http://opencv.org/about.html. Accessed: 2015-02-12.

[33] OpenCV. Back projection. http://docs.opencv.org/doc/tutorials/imgproc/histograms/back_projection/back_projection.html. Accessed: 2015-06-02.

[34] OpenCV. Camera Calibration and 3D Reconstruction. http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html. Accessed: 2015-02-09.

[35] OpenCV. Meanshift and camshift. http://docs.opencv.org/master/db/df8/tutorial_py_meanshift.html. Accessed: 2015-06-02.

[36] OpenCV. Miscellaneous image transformations. http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html. Accessed: 2015-06-02.

[37] P. Papadimitratos, A. La Fortelle, K. Evenssen, R. Brignolo, and S. Cosenza. Vehicular communication systems: Enabling technologies, applications, and future outlook on intelligent transportation. *IEEE Communications Magazine*, 47:84–95, 2009.

[38] Jesper B. Pedersen, Jonas B. Markussen, Mark P. Philipsen, and Morten B. Jensen. Counting the crowd at a carnival. 8th semester University project, 2014.

[39] Saarland University Piotr Danilewski. Cuda memory hierarchy, 2014.

[40] Charles Poynton. Frequently asked questions about color, 1997.

[41] Point Grey Research. Determining the focal length, image center, baseline and field of view measurements for pgr stereo vision cameras. https://www.ptgrey.com/KB/10315. Accessed: 2015-06-02.

[42] Point Grey Reserach. Triclops SDK. http://www.ptgrey.com/triclops. Accessed: 2015-02-09.

[43] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[44] Roy Shil. Bust out your own graphcut based image segmentation with opencv. http://www.morethantechnical.com/2010/05/05/bust-out-your-own-graphcut-based-image-segmentation-with-opencv-w-code/. Accessed: 2015-06-02.

[45] STT Nehterlands Study Centre for Technology Trends. Tomorrow's transport starts today, 2009.

[46] J.S. Sussman. *Perspectives on Intelligent Transportation Systems (ITS)*. Springer US, 2005.

[47] Technologyreview. Hidden obstacles for google's self-driving cars. http://www.technologyreview.com/news/530276/hidden-obstacles-for-googles-self-driving-cars/. Accessed: 2015-06-01.

[48] Udacity. Intro to parallel programming, 2014.

[49] U.S. Department of Transportation - Bureau of Transportation Systems. National transportation statistics. 2011.

[50] U.S Department of Transportation - National Highway Traffic Safety Administration. Crash Factors in Intersection-Related Crashes: An On-Scene Perspective. http://www-nrd.nhtsa.dot.gov/Pubs/811366.pdf, September 2010. Accessed: 2015-02-09.

[51] U.S Department of Transportation - National Highway Traffic Safety Administration. The Economic and Societal Impact Of Motor Vehicle Crashes, 2010. http://www-nrd.nhtsa.dot.gov/pubs/812013.pdf, May 2014. Accessed: 2015-05-25.

[52] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[53] Greg Welch and Gary Bishop. An introduction to the kalman filter. 1995.

[54] Nicholas Wilt. *The CUDA handbook : a comprehensive guide to GPU programming*. Addison-Wesley Professional, ISBN: 0321809467, first edition, 2011.

# Appendix A

# Jacobs Research Expo Poster

Below is an attachment the poster presented at UC San Diego Jacobs School of Engineering 32nd Annual Research Expo on April 16, 2015, in San Diego.

# Vehicle and Critical Event Detection in Night by Robust Stereo Analysis

*Mark P. Philipsen & Morten B. Jensen*

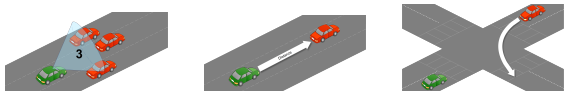
*Advisors: Mohan M. Trivedi & Thomas B. Moeslund*

AALBORG UNIVERSITY
DENMARK

UC San Diego
Jacobs School of Engineering
CVRR Computer Vision and Robotics Research Laboratory
LISA: Laboratory for Intelligent & Safe Automobiles

## Introduction

We introduce a stereo-vision based system that enables automatic event detection based on detection and tracking of vehicles in scenarios that usually would be highly problematic for monocular detectors. The purpose is to provides insight into patterns and behaviors of drivers during near-crashes and crashes. The proposed system will be limited to only handling a handful of events, which especially benefit from the extra dimension gained with stereo-vision. The events considered in this paper are illustrated in Figure 1. Monocular systems usually have problems dealing with occlusions and are in some cases using classifiers based on appearance, which require a large amount of training data for dealing with detection of vehicles from the many possible viewpoints and light conditions.




(a) Average number of cars in front of ego-vehicle.

(b) Distance to rear-end of vehicle directly in front.

(c) Other vehicle entering intersection - left turn across path.

(d) Other vehicle entering intersection - turning onto opposite direction.

(e) Other vehicle entering intersection - straight across path.

(f) Other vehicle entering intersection - turning same direction.

**Figure 1:** Critical events that can be automatically detected by the proposed method. Green car is ego vehicle. Red cars are other vehicles.[1]

Our contributions are:

- Using stereo-vision for automatic event detection in both day and nighttime, with focus on intersections (Figure 1c, 1d, 1e, 1f).
- Introducing a new event: Average number of vehicles in front of the ego vehicle. (Figure 1a).
- Introducing a new event: Average distance to vehicles directly in front of the ego vehicle. (Figure 1b).

## References

[1] M. P. Philipsen, M. B. Jensen, R. K. Satzoda, M. M. Trivedi, A. Møgelmose, and T. B. Moeslund, "Night-time drive analysis using stereo-vision for data reduction in naturalistic driving studies," in *IEEE, Intelligent Vehicle Symposium*, 2015.

[2] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *IEEE CVPR 2005*.

[3] R. Labayrade, D. Aubert, and J.-P. Tarel, "Real time obstacle detection in stereovision on non flat road geometry through" v-disparity" representation," in *IEEE, Intelligent Vehicle Symposium, 2002.*

[4] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *Intelligent Vehicles Symposium*, 2011.

## Proposed System

A Bumblebee XB3 stereo camera is used to acquire **stereo images** with an average rate of 15 FPS in an resolution of 1280x960. For generating the **disparity map**, the OpenCV's SGBM[2] implementation is used. **Noise** is removed using LR-RL consistency check, temporal information, and a monocular color check. The **road surface** is found by searching for the most significant line in the V-disparity[3] using RANSAC. Additionally, the line parameters are filtered using a Kalman filter to smooth out faulty road surface detections. Objects are projected to 3D using the camera's properties, which result in 3D **point cloud** representations of the segmented objects. The acquired point clouds are post processed using a band-bass filter to remove near and distant points, downsampled using a voxel grid, and outliers are removed. **Clusters** are found by creating a k-d tree, which organize points according to distance to neighbors.

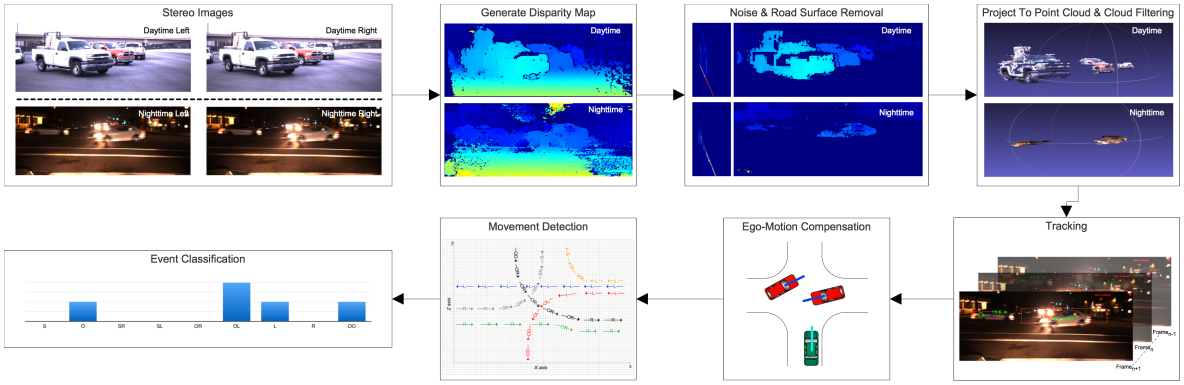



**Figure 2:** Critical event detection process flow.

The clusters' center points are used for nearest neighbor **tracking** between frames and for determining the distance from ego-vehicle to detected vehicles. For **ego-motion compensating** while approaching an intersection, we utilize the *LIBVISO2: C++ Library for Visual Odometry 2* [4]. Events are detected by looking at the **movement** history of other vehicles. For all detected vehicles, individual frame to frame movements are categorized to form a histogram of movements for **event classification**.
An overview of the system is shown in Figure 2. The final output is an event report.

## Results

The proposed system is evaluated on 4,992 day and 3,933 night time frames. In Table 1 the results are seen, where GT is the *ground truth* of events manually labeled and SO is the *system output*. The proposed system have an overall precision of 0.78 and recall of 0.72.

**Table 1:** Summary of event report analysis. The syntax for the results are [SO/GT]. P and R are abbreviations for precision and recall, respectively.

| *Drive Behavior Event* | Daytime | Nighttime | P | R |
|---|---|---|---|---|
| Right - straight across path | 35/32 | 5/19 | 0.95 | 0.63 |
| Left - straight across path | 45/34 | 11/33 | 0.87 | 0.67 |
| Left turn across path | 5/5 | 20/1 | 0.75 | 1 |
| Turn onto opposite dir. | 32/37 | 41/15 | 0.68 | 0.93 |
| Short turn onto same dir. | 7/5 | 9/5 | 0.63 | 1 |
| Long turn onto same dir. | 1/16 | 1/8 | 1 | 0.09 |
| Avg. number of cars | 1.67/1.74 | 1.6/1.3 | NA | NA |
| Avg. distance to car | 8.73 m | 10.98 m | NA | NA |

## Concluding Remarks

The use of stereo-vision is considered beneficial, especially in scenarios with partly occluded cars, in such cases most monocular systems will fail. Distance information from the ego vehicle to the surroundings provide useful information with regard to drive patterns before and during a crashes.

- Introduction of a novel stereo based critical event analysis approach.
- Experimental analysis shows very promising detection, trajectory and event classification rates.
- Ongoing research involves extensive experimental validation and a day and night time critical event detection module for public use.

## Acknowledgment

Thanks to Ravi K. Satzoda and Andreas Møgelmose for guidance.

# Appendix B

# Hardware

## B.1 Computer

The all of the software developed as part of this thesis was tested on the hardware described in the tables below.

**Table B.1:** Technical specification of MacBook Pro Retina, 15-inch, Late 2013

| MacBook Pro Retina, 15-inch, Late 2013 | |
|---|---|
| Processor | 2.3 GHz i7-4850HQ |
| Memory | 16 GB 1600 MHz PC3-12800 |
| Storage | 512 GB PCIe 3.0 SSD |
| Graphics | Intel Iris Pro 5200 1536 MB shared<br>Nvidia GeForce GT 750M with 2 GB GDDR5 |

Each of these machines run OS X Yosemite, Windows 7 and Ubuntu 12.04/Ubuntu 14.04

## B.2 Camera

Most stereo matching is based on the assumption that pixels belonging to the same objects have the same same intensity/color, in other words, they are radiometrically similar. However, radiometric differences in the image pairs will arias due to differences in the cameras, vignetting, image noise. This is not apparent in the datasets that stereo matching methods are usually compared

based on e.g. the unnaturally clean Middlebury stereo data set, Tsukuba. Some examples of types of image noise are listed below [23]:

- Fixed pattern noise is usually a result of long exposure and differences in the responses of individual sensor cells.
- Random noise is usually significant with low exposure.
  - Gaussian noise caused by poor illumination, high temperature and noise in the electronic circuitry.
  - Salt-and-pepper noise caused by analog-to-digital conversion errors and bit errors in transmission.
  - Shot noise is caused by the discrete nature of photons which follows a Poisson distribution, this type of noise is particularly noticeable in the lighter parts of an image. This effect is proportionally reduced with the amount of photons allowed in.
- Banding noise usually is a result of excessive brightness or due to faulty image sensor readout.

Furthermore differences may arise due to reflective surfaces, which will reflected different amounts of light depend on the viewing angle. This problem will only get bigger with increases in baseline length. Since precalibration cannot fix all of these issues, it can be expected that some radiometric differences will be present when the matching algorithm is put to work. The effects of the radiometric differences can be mitigated by applying filters such as mean and Laplacian of Gaussian, this comes with the drawback of blurred disparity maps [22].
Stereo image data is captured with Point Grey's Bumblebee XB3. The specification of the used Bumblebee camera can be seen in the table below:

### B.2.1 Triclops API

Interaction of the Bumblebee camera is done through the libdc1394 firewire camera library. This library allows for full camera control, video capture, multi-camera/multi-adapter support, multi-platform (Linux/OSX/Windows), colorspace conversion functions and multiple de-bayering algorithms. Point Grey provides a pgrlibdcstereo package which essentially is a wrapper library for libdc1394 that makes it easier to work with the stereo camera. Together with the propitiatory Triclops library the following steps are done during stereo image capture and rectification.

**Table B.2:** Technical specification of Bumblebee XB3(BBX3-13S2C-60)

| Bumblebee XB3(BBX3-13S2C-60) | |
|---|---|
| Resolution | 1280 x 960 |
| FPS | 16 |
| Color | Yes |
| Focal Length | 6 mm |
| Baselines | 12 cm<br>24 cm |
| Sensor | Sony ICX445 CCD |
| Sensor Format | 1/3" |
| Pixel Size | 3.75 $\mu$m square |
| Connection | IEEE-1394b (800Mb/s) |

### B.2.2 Setting Camera type

To work with the camera, we need to initialize an stereocamera object and set the capture mode. This is seen in code example (B.1).

```
1  PGRStereoCamera_t stereoCamera;
2    // query information about this stereo camera
3    err = queryStereoCamera(camera, &stereoCamera);
4    if ( err != DC1394_SUCCESS )
5    {
6      fprintf(stderr, "Cannot query all information from camera\n");
7      cleanup_and_exit(camera);
8    }
9
10   if (stereoCamera.nBytesPerPixel != 2)
11   {
12     // can't handle XB3 3 bytes per pixel
13     fprintf(stderr, "Example has not been updated to work with XB3 in 3 camera
           mode yet!\n");
14     cleanup_and_exit(stereoCamera.camera);
15   }
16
17   // set the capture mode
18   printf("Setting stereo video capture mode\n");
19   err = setStereoVideoCapture(&stereoCamera);
20   if (err != DC1394_SUCCESS)
21   {
22     fprintf(stderr, "Could not set up video capture mode\n");
23     cleanup_and_exit(stereoCamera.camera);
24   }
```

**Listing B.1:** Set the Camera

### B.2.3 Segmentation fault fix

Parts of the Triclops API is apparently not thread safe, therefore it is necessary to limit the number of threads to 1, to avoid segmentation fault 11.

```
1   e= triclopsSetMaxThreadCount( triclops , 1);   //Important to avoid
        segmentation fault
2   if ( e != TriclopsErrorOk )
3   {
4      fprintf( stderr , "triclopsSetMaxThreadCount failed!\n" );
5      triclopsDestroyContext( triclops );
6      cleanup_and_exit( camera );
7      return 1;
8   }
```

**Listing B.2:** limit thread count to 1

### B.2.4 Getting camera context(calibration)

To project into 3D point clouds, one needs to know the focal length in pixels. As we use the factory calibration we need to extract the calibration content from the camera. This is done as seen in code example B.3.

```
1  // get calibration and stuff from camera
2
3   printf("Getting TriclopsContext from camera (slowly)... \n");
4   e = getTriclopsContextFromCamera(&stereoCamera , &triclops);
5   if (e != TriclopsErrorOk)
6   {
7     fprintf(stderr , "Can't get context from camera\n");
8     cleanup_and_exit(camera);
9     return 1;
10  }
11  printf("...done\n");
12
13  printf("Saving TriclopsContext from camera... \n");
14  e = triclopsWriteDefaultContextToFile(triclops , "triclops.cal");
15  if (e != TriclopsErrorOk)
16  {
17    fprintf(stderr , "Can't save context from camera\n");
18    cleanup_and_exit(camera);
19    return 1;
20  }
21  printf("...done\n");
22   }
```

**Listing B.3:** Extract and save camera content

### B.2.5 Setting shutter speed

Through the libdc1394 library it is possible to access and set the camera gain and shutter speed. The exact values will be adjusted as a compromise between the following measures in the resulting images.

- low gain - low intensity but little noise
- high gain - high intensity but lots of noise and artifacts

Image example:

- low shutter speed - high intensity and less prone to problems with AC lighting but lots of blur and oversaturation
- high shutter speed - low intensity and more prone to problems with AC lighting but sharp and less oversaturation

Image example:
The listing below shows how the camera's gain is set using dc1394

```
1 //set gain to manual
2 err = dc1394_feature_set_power(stereoCamera.camera, DC1394_FEATURE_GAIN,
      DC1394_ON);
3 dc1394_feature_set_mode(stereoCamera.camera, DC1394_FEATURE_GAIN,
      DC1394_FEATURE_MODE_MANUAL);
4 err = dc1394_feature_set_absolute_control(stereoCamera.camera,
      DC1394_FEATURE_GAIN, DC1394_ON);
5 err = dc1394_feature_set_absolute_value(stereoCamera.camera,
      DC1394_FEATURE_GAIN, gain);
```

**Listing B.4:** Setting camera gain

The listing below shows how the camera's shutter speed is set using dc1394

```
1 //set shutter to manual [minShut: 0.000003 maxShut: 0.020847]
2 err = dc1394_feature_set_power(stereoCamera.camera, DC1394_FEATURE_SHUTTER,
      DC1394_ON);
3 dc1394_feature_set_mode(stereoCamera.camera, DC1394_FEATURE_SHUTTER,
      DC1394_FEATURE_MODE_MANUAL);
4 err = dc1394_feature_set_absolute_control(stereoCamera.camera,
      DC1394_FEATURE_SHUTTER, DC1394_ON);
5 err = dc1394_feature_set_absolute_value(stereoCamera.camera,
      DC1394_FEATURE_SHUTTER, shutterSpeed);
6
7 err = dc1394_feature_get_absolute_value(stereoCamera.camera,
      DC1394_FEATURE_SHUTTER, &s);
8 err = dc1394_feature_get_absolute_value(stereoCamera.camera,
      DC1394_FEATURE_GAIN, &g);
9 printf("Settings set to; gain: %f shutter: %f\n", g, s);
```

**Listing B.5:** Setting camera shutter speed

### B.2.6 Getting full RGB images from the camera(api buffer fix)

```
// size of buffer for all images at mono8
  unsigned int nBufferSize = stereoCamera.nRows *
                             stereoCamera.nCols *
                             stereoCamera.nBytesPerPixel;
  // allocate a buffer to hold the de-interleaved images
  unsigned char* pucDeInterlacedBuffer = new unsigned char[nBufferSize];
  unsigned char* pucRGBBuffer   = new unsigned char[3 * nBufferSize];
  unsigned char* pucRightBuffer = new unsigned char[3 * nBufferSize];
  unsigned char* pucLeftBuffer  = new unsigned char[3 * nBufferSize];
  unsigned char* pucRightRGB  = NULL;
  unsigned char* pucLeftRGB   = NULL;
  unsigned char* pucCenterRGB = NULL;

  TriclopsInput inputLeft,inputRight;
```

**Listing B.6:** Initializing image buffers

```
// get the images from the capture buffer and do all required processing
      // note: produces a TriclopsInput that can be used for stereo processing
      extractImagesColor(&stereoCamera,
          DC1394_BAYER_METHOD_NEAREST,
            pucDeInterlacedBuffer,
            pucRGBBuffer,
            pucRightBuffer,
          pucLeftBuffer,
            &pucRightRGB,
            &pucLeftRGB,
            &pucCenterRGB,
            &inputRight,
            &inputLeft
          );

    TriclopsColorImage leftTriImg,rightTriImg;

    e = triclopsRectifyColorImage(triclops,TriCam_RIGHT, &inputRight, &
        rightTriImg);
    e = triclopsRectifyColorImage(triclops,TriCam_LEFT, &inputLeft, &
        leftTriImg);

    vector<Mat> channelsLeft,channelsRight;

    cv::Mat blueLeft(stereoImageHight,stereoImageWidth, CV_8UC1, leftTriImg.
        blue);
    cv::Mat greenLeft(stereoImageHight,stereoImageWidth, CV_8UC1, leftTriImg.
        green);
    cv::Mat redLeft(stereoImageHight,stereoImageWidth, CV_8UC1, leftTriImg.
        red);

    cv::Mat blueRight(stereoImageHight,stereoImageWidth, CV_8UC1, rightTriImg
        .blue);
    cv::Mat greenRight(stereoImageHight,stereoImageWidth, CV_8UC1,
        rightTriImg.green);
    cv::Mat redRight(stereoImageHight,stereoImageWidth, CV_8UC1, rightTriImg.
        red);

    channelsLeft.push_back(blueLeft);
    channelsLeft.push_back(greenLeft);
    channelsLeft.push_back(redLeft);

    channelsRight.push_back(blueRight);
```

```
36      channelsRight.push_back(greenRight);
37      channelsRight.push_back(redRight);
38
39      Mat leftImageRect,rightImageRect;
40
41      merge(channelsLeft,leftImageRect);
42      merge(channelsRight,rightImageRect);
```

**Listing B.7:** modified extractImageColor function

By default the function extractImagesColor() in the pgrlibdcstereo library for some reason return pTriclopsInput as a pointer to only the green channel. This means a lot of information is lost. Therefore the library is changes and recompile so the buffer pointed to by pTriclopsInput contain all 3 color channels.

```
1   unsigned char* pucGrabBuffer = frame->image;
2
3      if ( stereoCamera->nBytesPerPixel == 2 )
4      {
5         // de-interlace the 16 bit data into 2 bayer tile pattern images
6         dc1394_deinterlace_stereo( pucGrabBuffer,
7                                    pucDeInterleaved,
8                                    stereoCamera->nCols,
9                                    2*stereoCamera->nRows );
10        // extract color from the bayer tile image
11        // note: this will alias colors on the top and bottom rows
12        dc1394_bayer_decoding_8bit( pucDeInterleaved,
13                                    pucRGB,
14                                    stereoCamera->nCols,
15                                    2*stereoCamera->nRows,
16                                    stereoCamera->bayerTile,
17                                    bayerMethod );
18        // now deinterlace the RGB Buffer to extract the green channel
19        // The green channel is a quick and dirty approximation to the mono
20        // equivalent of the image and can be used for stereo processing
21        dc1394_deinterlace_rgb( pucRGB + 3 * stereoCamera->nRows * stereoCamera
              ->nCols,
22                                pucLeft,
23                                stereoCamera->nCols,
24                                3*stereoCamera->nRows );
25         dc1394_deinterlace_rgb( pucRGB,
26                                pucRight,
27                                stereoCamera->nCols,
28                                3*stereoCamera->nRows );
29
30        *ppucRightRGB  = pucRGB;
31        *ppucLeftRGB   = pucRGB + 3 * stereoCamera->nRows * stereoCamera->nCols;
32        *ppucCenterRGB = *ppucLeftRGB;
33     }
34
35     pTriclopsInputLeft->inputType    = TriInp_RGB;
36     pTriclopsInputLeft->nrows  = stereoCamera->nRows;
37     pTriclopsInputLeft->ncols  = stereoCamera->nCols;
38     pTriclopsInputLeft->rowinc = stereoCamera->nCols;
39     pTriclopsInputLeft->u.rgb.red        = pucLeft +2* stereoCamera->nRows *
           stereoCamera->nCols;
40     pTriclopsInputLeft->u.rgb.green      = pucLeft +1* stereoCamera->nRows *
           stereoCamera->nCols;
41     pTriclopsInputLeft->u.rgb.blue       = pucLeft +0* stereoCamera->nRows *
           stereoCamera->nCols;
```

```
42
43      pTriclopsInputRight->inputType  = TriInp_RGB;
44      pTriclopsInputRight->nrows = stereoCamera->nRows;
45      pTriclopsInputRight->ncols = stereoCamera->nCols;
46      pTriclopsInputRight->rowinc     = stereoCamera->nCols;
47      pTriclopsInputRight->u.rgb.red       = pucRight +2* stereoCamera->nRows *
            stereoCamera->nCols;
48      pTriclopsInputRight->u.rgb.green     = pucRight +1* stereoCamera->nRows *
            stereoCamera->nCols;
49      pTriclopsInputRight->u.rgb.blue      = pucRight +0* stereoCamera->nRows *
            stereoCamera->nCols;;
50
51      // return buffer for use
52      dc1394_capture_enqueue( stereoCamera->camera, frame );
```

**Listing B.8:** changing pgrlibdcstereo to return full 3 channel RGB image

### B.2.7 Rectifying color images

The 3 channel RGB images returned by extractImagesColor() can then be passed to triclopsRectifyColorImage() where they are rectified using the triclops context that was extracted from the camera earlier. Afterwards all that is left is to assemble channels from the rectified TriclopsColorImages to two Mat type RGB images.

```
1       TriclopsColorImage leftTriImg,rightTriImg;
2
3       e = triclopsRectifyColorImage( triclops,TriCam_RIGHT, &inputRight, &
            rightTriImg );
4       e = triclopsRectifyColorImage( triclops,TriCam_LEFT, &inputLeft, &
            leftTriImg );
5
6       vector<Mat> channelsLeft,channelsRight;
7
8       cv::Mat blueLeft(myImageHeight,myImageWidth, CV_8UC1, leftTriImg.blue );
9       cv::Mat greenLeft(myImageHeight,myImageWidth, CV_8UC1, leftTriImg.green);
10      cv::Mat redLeft(myImageHeight,myImageWidth, CV_8UC1, leftTriImg.red);
11
12      cv::Mat blueRight(myImageHeight,myImageWidth, CV_8UC1, rightTriImg.blue);
13      cv::Mat greenRight(myImageHeight,myImageWidth, CV_8UC1, rightTriImg.green
            );
14      cv::Mat redRight(myImageHeight,myImageWidth, CV_8UC1, rightTriImg.red);
15
16      channelsLeft.push_back(blueLeft);
17      channelsLeft.push_back(greenLeft);
18      channelsLeft.push_back(redLeft);
19
20      channelsRight.push_back(blueRight);
21      channelsRight.push_back(greenRight);
22      channelsRight.push_back(redRight);
23
24      Mat leftImageRect,rightImageRect;
25
26      merge(channelsLeft,leftImageRect);
27      merge(channelsRight,rightImageRect);
```

**Listing B.9:** rectifying and composing color images

When the program terminates, wee clean up the buffers.

### B.2.8 Cleaning up

```
1  printf("Stop transmission\n");
2  if (dc1394_video_set_transmission( stereoCamera.camera, DC1394_OFF ) !=
       DC1394_SUCCESS)
3  {
4    fprintf(stderr, "Couldn't stop the camera?\n");
5  }
6
7  if (pucDeInterlacedBuffer)
8    delete[] pucDeInterlacedBuffer;
9  if (pucRGBBuffer)
10   delete[] pucRGBBuffer;
11 if (pucRightBuffer)
12   delete[] pucRightBuffer;
13 if (pucLeftBuffer)
14   delete[] pucLeftBuffer;
```

**Listing B.10:** cleaning up

## B.3 CUDA

A important tool for reducing execution time of expensive algorithms is the GPU. For Nvidia GPUs this can be done using the CUDA toolkit. For speeding up some of the programs developed as part of this thesis, the CUDA capable GPU described in table B.3 was available. Much of the following information originates from a Parallel programming course by Udacity [48].

CUDA allows for programming of both CPU and GPU in the same program. The CUDA compiler splits the program in CPU and GPU instructions.

#### CUDA programming

Programming for GPUs that supports CUDA is done in CUDA C and CUDA C++, which essentially are extended versions of the standard C and C++ languages. The extensions enables execution of special functions, called kernels, in parallel on the GPU. Using the terminology associated with CUDA programming, the CPU and it's associated memory is called the host, whereas GPU and it's memory is referred to as the device. The host manages it's own memory as well as the device memory. This means that in a typical CUDA program the host allocates and initialize data on the host, followed by memory allocation on the device. The data the must be processed on the device is uploaded and kernels are executed on that data. The resulting data is finally downloaded

**Table B.3:** Technical specifications for GeForce GT 750M CUDA Capabilities.

| GeForce GT 750M CUDA Capability | |
|---|---|
| CUDA Driver Version | 7.0 |
| CUDA Capability | 3.0 |
| Total Memory | 2048 MB |
| CUDA Cores | 2 Multiprocessors(MP)<br>192 CUDA Cores per MP |
| GPU Clock | 926 MHz |
| Threads per MP | 2048 |
| Threads per block | 1024 |
| Warp size | 32 |
| Block dimension (x,y,z) | (1024, 1024, 64) |

from device to host memory. Figure B.1 shows the software layers involved in a CUDA program. CUDA Runtime(CUDART) creates an abstraction layer on top of parts of the driver API, this layer hides some of the explicit resource management and facilitates easier integration into C++ code by enabling a simple way for kernel invocation.
When programming for the GPU a kernel program must look like a serial program for one thread. Before kernels can be executed the the structure of the threads that will be executing the kernels must be set. Each thread have it's own memory. Threads are organized in blocks, each with it's own memory shared between the threads. The blocks are the organized in grids which access global memory.

With CUDA there are two additions to the ordinary C and C++ syntax: Triple angle brackets are used when calling device code from host code.

```
1 mykernel<<<N,M,(shared mem per block in bytes)>>>(in1,in2,out1);
```

**Listing B.11:** triple angle brackets function call

N and M are parameter that have to do with the number of blocks and ... when executing the kernel on the GPU. N and M determine the number of times the function is executed e.i the number of blocks and the number of threads per block respectively. Following the angle brackets are input and output variables. These must be allocated on the device by the CPU before

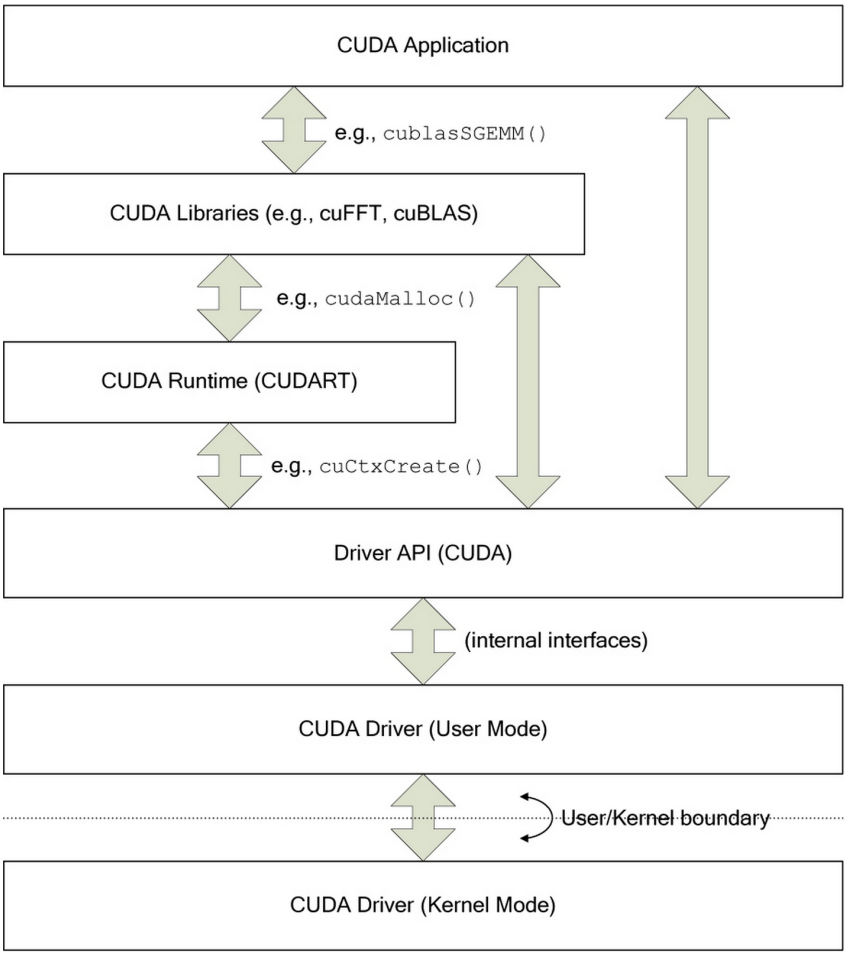



**Figure B.1:** Hierarchy of the layers involved in CUDA programming.[54]

the kernel is launched. N and M can be 1, 2 or 3D grids, it deaults to 1D. A 3D grid of blocks is initialized with dim3(x,y,z). dim3(x,1,1)==(dim3,x)==x Device code is written in functions with the keyword

```
__global__ void mykernel(void)
{
...
}
```

**Listing B.12:** device function

```
__global__ void add(int *a, int *b, int *c)
{
*c = *a + *b;
}
```

**Listing B.13:** device function

Since the function runs on the device, variables must be located in device memory. A typical memory flow of CPU+GPU program:

```
cudaMalloc() // CPU allocates memory on GPU
cudaMemcpy() // Copying data from CPU to GPU
"kernel launch" by CPU onto GPU where it process the data
cudaMemcpy() // Copying data back to CPU from GPU
cudaFree() // Release memory resources on GPU
```

**Listing B.14:** device function

Memory handling is done from the host:

```
int main(void) {
int a, b, c; // host copies of a, b, c
int *d_a, *d_b, *d_c; // device copies of a, b, c
int size = sizeof(int);
// Allocate space for device copies of a, b, c
cudaMalloc((void **)&d_a, size);
cudaMalloc((void **)&d_b, size);
cudaMalloc((void **)&d_c, size);
// Setup input values
a = 2;
b = 7;
// Copy inputs to device
cudaMemcpy(d_a, &a, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, &b, size, cudaMemcpyHostToDevice);
// Launch add() kernel on GPU
add<<<1,1>>>(d_a, d_b, d_c);
// Copy result back to host
cudaMemcpy(&c, d_c, size, cudaMemcpyDeviceToHost);
// Cleanup
cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
return 0;
}
```

**Listing B.15:** device function

Kernels run asynchronously on the GPU so control is returned to the CPU immediately. When results are to be transferred back to the CPU, it must be insured that the GPU is done.

```
cudaMemcpy() //Blocks the CPU until the copy is complete
Copy begins when all preceding CUDA calls have completed
cudaMemcpyAsync() //Asynchronous, does not block the CPU
cudaDeviceSynchronize() //Blocks the CPU until all preceding CUDA calls have
    completed
```

**Listing B.16:** device function

### Parallel communication patterns

When initialization parallelization, there exists communication patterns. These are briefly described below:

**Map:** 1 to 1 correspondence between input and output. E.g. conversion of 3 channel color image to single channel grayscale.

**Gather:** many to 1 correspondence between input and output. E.g. image blurring.

**Scatter:** 1 to many correspondence between input and output. In part of the program the write location is calculated. Problem multiple threads may try to write to the same location at the same time. E.g. histogram calculation from image.

**Stencil:** Stencil is a fixed pattern where neighboring values in an array/matrix are accessed. Lots of data reuse/overlap in memory access. E.g. the sobel filter kernel.

**Reduce:** all to 1 correspondence between input and output. E.g. summation.

**Scan and sort:** all to all correspondence between input and output. E.g. finding the median.
With some of the communication patterns above it is necessary to introduce synchronization between the treads and/or blocks. One form is called a barrier e.g.

```
1 __syncthreads();
```

**Listing B.17:** barriers

CUDA makes no guaranties about when and on which sm a specific thread block will run. This allows for great efficiency and scaleability. CUDA does however guarantee that all thread in a block are executed on the same SM at the same time and all blocks in a kernel finish before blocks in from a following kernel are executed.

### Efficient GPU usage

**Arithmetic intensity** When doing GPU programming desirable to obtain as high arithmetic intensity as possible, this relationship can be seen in equation (B.1).

$$arithmeticintensity = \frac{timespendonmathoperations}{timespendonmemoryoperations} \tag{B.1}$$

Additionally, since threads on the GPU have access to three layers of memory, data should be placed in the closest and fastest when possible. Figure B.2 illustrates the memory hierarchy. Speed wise the memory is ordered local(registers) > shared > global.

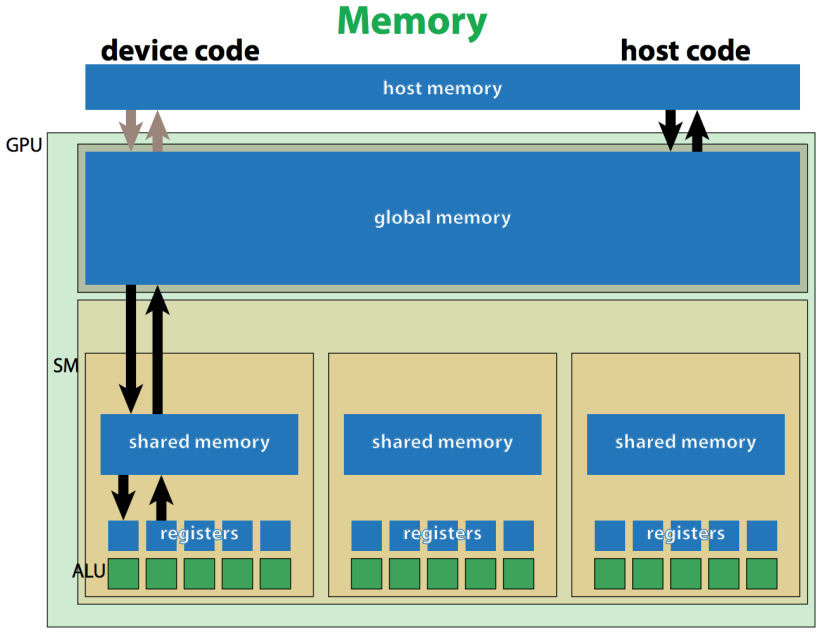



**Figure B.2:** Hierarchy of the memory layers involved in CUDA programming.[39]

Use coalesced global memory access when possible, meaning adjacent threads should access contiguous chunks of memory. This can in many cases be done by transposing the data structure.

**Thread divergence** The threads should follow the same paths through the code when possible. Threads diverge due to if and else and different threads doing loops different number of times.

### CPU vs GPU

The CPU strengths:

1. CPU has higher clock speed, more cycles per second - power and heat limited.

2. CPU does more instructions per cycle - instruction set and cost limited.

CPU uses complex control hardware - flexibility, optimized for low latency.

A CPU analogy would be a rally car, which has a low passenger capacity but is able to quickly get to the finish.

The GPU strengths:

1. GPU used less power per calculation.
2. GPU scales well.

GPU Little control but lots of computation hardware - restricted but power efficient, optimized for high throughput(pixels/second)

A CPU analogy would be a school bus, which has a high passenger capacity but is slow and might need to take a longer and more passable route to the finish. It is said that: "The GPU does not get out of bed for less than 1000 threads [48]".

### Integrating custom CUDA with OpenCV CUDA

Interfacing with OpenCV gpuMat using CUDA data pointer is necessary if GPU memory is access through OpenCv's API and the assigned memory needs to be accessed using standard CUDA.

```
1  //You can create GpuMat object for existed pointer and GpuMat will not delete
       this memory in destructor:
2
3  void* data;
4  size_t step;
5  cudaMalloc2D(&data, &step, width * sizeof(float), height);
6  GpuMat mat(height, width, CV_32FC1, (uchar*) data, step); /cv::gpu::GpuMat dst
       (height, width, CV_32F, d_Ptr);
7  gpu::bitwise_not(mat, mat);
8  cudaFree(data);
9
10 // And you can get device pointer from GpuMat with ptr<> method:
11 float* data = mat.ptr<float>();
12 size_t step = mat.step;
```

**Listing B.18:** gpumatCUDA

### CUDA implementation example for OpenCV

The following CUDA kernel calculates V-disparity.

```
1  __global__
2  void vdispKernel(unsigned char* image,
3                   unsigned int* vdispImage,
```

```
4                           int numRows, int numCols, int vdispNumRows, int
                                vdispNumCols)
5   {
6     int index_x = blockIdx.x * blockDim.x + threadIdx.x;
7     int index_y = blockIdx.y * blockDim.y + threadIdx.y;
8
9     // map the two 2D indices to a single linear, 1D index
10    int grid_width = gridDim.x * blockDim.x;
11    int index = index_y * grid_width + index_x;
12
13    int vdispIdx = ((index/(numCols-1))*255) + image[index];
14    atomicAdd(&vdispImage[vdispIdx],1);
15  }
```

**Listing B.19:** gpumatCUDA

Table B.4 shows the difference in computation time between calculating V-disparity on CPU and GPU.

**Table B.4:** V-disparity timing from CPU and GPU.

| Processing unit | Computation time |
|---|---|
| CPU | 12.9 ms |
| GPU | 1.6 ms |

# Appendix C

# Software

## C.1 Libraries and toolboxes

### C.1.1 OpenCV

OpenCV is an open source library originally started by Intel, it contains a collection of 2500 optimized computer vision and machine learning algorithms. The majority of the algorithms are implemented for CPU, but many are also available for GPU[32]. Everything that is developed as part of this master thesis is developed using OpenCV's C++ API.

### C.1.2 PCL

PCL is an open source library that was started by some of the people at Willow garage.
PCL e.g allows for easy organization of 3D points into a point cloud structure, optimized filtering and visualization.

### C.1.3 libviso

LIBVISO2 is a C++ library for visual odometry. It enables 6 DOF for ego motion estimation for a camera through a sequence of monocular or stereo images. The library is accompanied by publication [18].

### C.1.4 Specialized libraries

In addition to the primary libraries that was just mentioned, the following specialized libraries are used, mostly because they are used extensively by the primary libraries.

**Eigen** Contains SSE optimized linear algebra, vector and matrix operations.

**FLANN** Contains data structures such as kd-tree, which enables efficient nearest neighbor searches.

**QHull** Contains operations for finding convex hull.

### C.1.5 Matlab toolbox

Piotr's Computer Vision Matlab Toolbox [11] has been used together with the regular Matlab environment. The description found on the webpage define the purpose of the toolbox very well: "*This toolbox is meant to facilitate the manipulation of images and video in Matlab. Its purpose is to complement, not replace, Matlab's Image Processing Toolbox, and in fact it requires that the Matlab Image Toolbox be installed.* ". The content of the toolbox consists of 7 parts: channels, classify, detector, filters, images, matlab, and videos. In this project only 2 of these have been used, namely: channels and detector. The aggregated channels and their content are described in [12, 14, 13, 15].

### C.1.6 Creating a Aggregated Channel Features Object Detector

In this project we have utilzed Aggregated Channel Features (ACF) for detecting traffic lights. In order to train a cascading classifier based on these, both positive and negative sample images must be obtained. Positives samples consist of image with the object of interest. Negatives samples should not contain any instances of the desired objects.
Piotr's Computer Vision Matlab Toolbox provides a great and easy tool for training an ACF based detector. In code example C.1 the *opts=acfTrain()* command is seen, which basically is the creation of an options object for the acfTrain function. This is followed by a large set of options settings, whereof only some are used in the same code example. The *modelDs* defines the model's dimensions in terms of height and width. This region can be extended using the model dimension padding, *modelDsPad*. An important notation is that dimensions of *modelDsPad* must be of same size of pre-cropped positive samples. The pBoost argurments defines parameters for AdaBoost. Code example C.1 uses pre-cropped positives samples, but regular full size negative images. Finally, all of the input parameters are used for model training.

```
1 %% Set up ACF Detector
2 opts=acfTrain();
3
```

```
opts.modelDs=[40 20];
opts.modelDsPad=[50 25];
opts.pPyramid.pChns.pColor.smooth=0;
opts.nWeak=[10 100 2000];
opts.pBoost.pTree.maxDepth=2;
opts.pBoost.discrete=0;
opts.pBoost.pTree.fracFtrs=1/16;
opts.nNeg=25000;
opts.nAccNeg=50000;
opts.pPyramid.pChns.pGradHist.softBin=1; opts.pJitter=struct('flip',1);

opts.posWinDir=[dataDir '/train/pos'];
opts.negImgDir=[dataDir '/train/neg'];

opts.pPyramid.pChns.shrink=1;
opts.name='models/Lisa+';

%% Train ACF Detector
 detector = acfTrain(opts);
```

**Listing C.1:** Setup and train the ACF based detector.

Only certain modifications can be done after training the detector. More information can be found in *acfModify.m* from the toolbox. In code example C.2 only *cascThr* and *cascCal* are modified for reaching a better recall. These parameters adjust the threshold and calibrations for the constant soft cascades.

```
%% Modify ACF Detector
 pModify=struct('cascThr',0.5,'cascCal',0.1);
 detector=acfModify(detector,pModify);
```

**Listing C.2:** Modify the detector post training.

Finally, the trained detector is applied on input images. In code example C.3 the trained ACF based detector is applied on all frames from the input video sequence 'dayclip2.avi'. The output of the detector is a set of bounding boxes with a confidence level. For preventing bounding boxes with a large amount of overlap, the type argument *max* is used for discarding bounding boxes with the lowest confidence in case of major overlap.

```
videoObj = VideoReader('dayclip2.avi');
get(videoObj);
nFrames = floor(videoObj.NumberOfFrame);

for frameNumber = 1 : nFrames
      frame = read(videoObj, frameNumber);
      imgLoi = imcrop(frame,[0 0 1280 960/2]); % Cropping to upper half of left
          image.
      bbs = acfDetect(imgLoi,detector);
      bbsOverlap = bbNms(bbs, 'type','max' ); % If large overlap, the bb with
          the lower score is suppressed.
end
```

**Listing C.3:** Loading a video sequence and using the trained ACF based detector.

For plotting the bounding boxes on the frame, the bbApply.m provides a useful drawing function. It can also be used for drawing the ground truth bounding

boxes when these are loaded. This could be done similar to what is seen in code example C.4 where all content from a txt file is read into *array*.

```
%% Fetch ground truth
annotation = textread('enlargedBB-dayclip2.txt','%s', 'delimiter', '\n','
    whitespace', '');
array(size(annotation,1))=0;
for i=1:size(annotation,1)
    line = (strsplit(char(annotation(i,1))));
    for j=1:size(line,2)
        array(i,j) = str2double(line(1,j));
    end
end
```

**Listing C.4:** Load in ground truth

For the LISA Traffic Light database, the ground truth txt file syntax is "<frameNumber> <numberOfObjects> <upperLeftCorner> <upperLeftCornerY> <width> <height>". The four last parameters are scaled according to the number of objects in the frame. An example of plotting the ground truth bounding boxes on the frame is seen code example C.5.

```
    for i = 1 : size(annotation,1);
        if (frameNumber-1)==array(i,1); % We start annos from 0, matlab index
            from 1
            numObjects = array(i,2);
            % Convert to bbApply syntax
            gtBB(numObjects,4)=0;
            for j=1:numObjects
                gtBB(j,1) = array(i,(3+(4*(j-1))));
                gtBB(j,2) = array(i,(4+(4*(j-1))));
                gtBB(j,3) = array(i,(5+(4*(j-1))));
                gtBB(j,4) = array(i,(6+(4*(j-1))));

            end
            bbApply('draw',gtBB,'b');
        end
    end
```

**Listing C.5:** Plotting ground truth bounding boxes

### C.1.7 Optimization for Image Indexing

In this section useful optimizations for operating on images are explained.

#### Image indexation

Indexation and scanning through image matrices is a very common operation in image processing, it is therefore important to do it as efficiently as possible. Code example C.6 shows the simplest and safest way of indexing an image in OpenCV.

```
for (int32_t v=0; v<nCols; v++) {
    for (int32_t u=0; u<nRows; u++) {
        img_data[k]  = (uint8_t)image.at<uchar>(u,v);
        k++;
    }
}
```

**Listing C.6:** Iterator(safe) method for copying image elements to 1D array.

The average execution time of a 1280x500 image was found to be around 5.97 ms.

Code example C.7 shows a faster image indexation approach, which acquire a pointer for each row and increment it for the columns. For continuously allocated images if nCols is can be set to nCols * nRows and nRows is then set to 1.

```
for(int32_t i = 0; i < nRows; ++i) {
    valueP = image.ptr<uchar>(i);
    for (int32_t j = 0; j < nCols; ++j) {
        img_data[k] = (uint8_t)valueP[j];
        k++;
    }
}
```

**Listing C.7:** Pointer(fast) method for copying image elements to 1D array.

The average execution time of a 1280x500 image was found to be around 2.15 ms.

# Appendix D

# Installation guides for thesis essential software

## D.1 Installing OpenCV with CUDA support in OSX

Installing OpenCV with CUDA support on OSX is now possible without serious complications since CUDA 7 beta release. Previous versions of CUDA would only compile with the libstdc++, which was the standard library in earlier versions of OSX before the transition from gcc to llvm, which uses the newer libc++.
In Listing **??** the easy installation of OpenCV without CUDA support is shown, this procedure can also be recommended as a first step before compiling OpenCV with CUDA support, since homebrew will automatically install many of the needed dependencies.

```
1 $ brew tap homebrew/science
2
3 $ brew install opencv --with-ffmpeg --with-tbb --with-tests --with-qt
```

**Listing D.1:** brew install OpenCV.

This should result in a working OpenCV installation without CUDA.
You should now have installed all the required packages for installing OpenCV. Next, you need to download OpenCV from http://opencv.org/downloads.html. For our approach we used OpenCV 2.4.10 for Linux. When you have downloaded your OpenCV distribution, we advice to move the directory from the "Downloads" directory, as you experience show that this folder are sometimes completed erase as a result of a lot of unorganized files.
In order to get OpenCV with CUDA support, first download and install CUDA 7.0 from nvidia's homepage. You can check your compute Capability at https:

//developer.nvidia.com/cuda-gpus. E.g the graphic card used throughout this approach is a NVIDIA GeForce GT 750M, which has a compute capability of 3.0. When done download the source code for OpenCV. Before compiling OpenCV, make sure that earlier installations are gone. If homebrew was used the command 'brew uninstall opencv' will do the job.
Start a terminal (ctrl+t), and go through the steps seen in Listing D.13. Before executing "make -j8", you should look through the makefile to check whether all the modules are correctly selected. E.g check if CUDA is included. Please note, that this process may take a few hours to complete.

```
$ cd opencv-2.4.10
$ mkdir build
$ cd build
$ cmake -D WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D
    INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -
    D WITH_QT=ON -D WITH_VTK=ON -D WITH_OPENGL=ON -D WITH_CUDA=ON -D
    CUDA_ARCH_BIN=3.0 -D WITH_TIFF=OFF -D WITH_OPENEXR=OFF ..
$ make -j8
$ sudo make install
```

**Listing D.2:** Compile and install OpenCV with CUDA.

To install the OpenCV 3.0 Beta along with the experimental modules, download OpenCV and OpenCV_Contrib from https://github.com/Itseez

```
$ cd <opencv_location>/

$ mkdir build

$ cd build

$ cmake -D WITH_TBB=ON  -D WITH_V4L=ON -D WITH_QT=ON -D WITH_VTK=ON -D
    WITH_TIFF=OFF -D WITH_OPENGL=ON -D WITH_CUDA=ON -D CUDA_ARCH_BIN=3.0 -D
    WITH_OPENEXR=OFF -DOPENCV_EXTRA_MODULES_PATH=</opencv_contrib/modules> ..

$ make -j8

$ sudo make install
```

**Listing D.3:** Install OpenCV3 with contrib modules.

## D.2 Installing Windows 7 on recent Macs

Installing Windows 7 through Bootcamp.
Since newer Macs only have USB3 and Windows 7 doesn't support that by default, you will have no mouse or keyboard for installation if the installation isn't done with the following steps:

1. Have a Windows 7 image file ready somewhere on your Mac as well as a <8GB USB drive.

2. Open Boot Camp Assistant, check all 3 boxes – make boot disk, download software and partition disk. Go through screens until it ends at a screen where all you can do is click quit.

3. Reopen Boot Camp Assistant and this time CHECK THE THIRD BOX ONLY, the one to partition your drive. Be sure to leave the first two unchecked. Choose your partition size (make it big <100GB if you later need Ubuntu also). When Boot Camp is done partitioning the disk it will automatically quit Boot Camp and reboot your computer, which it didn't do the first time.

4. As computer is automatically rebooting, hold down alt/option key until you see screen where you can choose to boot up from Mac HD or your windows USB. Select the 'windows' drive.

5. This time when you get to the first Windows 7 screen, you should have a nice big fat white cursor that works.

When you start your new Windows 7 OS you might want to remove the pagefile and disable hibernation, since they each by default take up the amount of RAM your Mac has on your SSD also. With an SSD and plenty of RAM they are really not necessary.
Disable The Paging File:

1. Right-click Computer

2. Select Properties

3. Select Advanced system settings

4. Select the Advanced tab and then the Performance radio button

5. Select the Change box under Virtual memory

6. Un-check Automatically manage paging file size for all drives
7. Select No paging file, and click the Set button
8. Select OK to allow and restart.

Disable Hibernation:

1. Type "cmd" in the Start menu search box
2. Right-click on the cmd program and select Run as Administrator
3. In the command line, type "powercfg -h off"
4. Once completed, the command prompt returns and you can close it.

## D.3 Installing OpenCV with CUDA support in Windows

Installing OpenCV with Cuda support on Windows 7 have the following Prerequisites:

- CMake
- Cuda 6.5 64 bit
- OpenCV 2.4.10
- Visual Studio 2013
- Intel TBB(optional)
- Python(optional)

The procedure is as follows:

1. Install CMake, CUDA, Python and Visual Studio
2. Download Intel TBB and OpenCV to a permanent location
3. Open CMake
   (a) Select Source Folder to opencv/source
   (b) Select Output Folder to opencv/build
   (c) Press 'Configure'
   (d) Choose Visual Studio 12 2013 Win64
   (e) CMake settings round 1.
      i. Cancel BUILD_DOCS and BUILD_EXAMPLES
      ii. CMAKE_LINKER must be Visual Studio 12.0 for vs2013
      iii. Cancel CUDA_ATTACH_VS_BUILD_RULE_TO_CUDA_FILE, to avoid some CUDA related errors
      iv. Choose WITH_CUBLAS, WITH_CUDA, WITH_OPENGL, WITH_TBB
      v. Press Configure to refresh
   (f) CMake settings round 2.
      i. Set include path for tbb, to <TBB permanent location>lib\intel64\vc12

ii. Press ‘Configure’ to refresh

(g) Press ‘Generate’ to create OpenCV.sln in output folder

(h) Add ‘#include <algorithm>’ to “opencv-2.4.10\modules\gpu\src\nvidia\core\NCV.cu”. Otherwise you’ll have ‘max’ undefined error.

(i) Compile OpenCV.sln

i. If any of your libraries (OpenCV, tbb, Python, etc.) is in “C:\Program Files”, you need to run Visual Studio 2013 as administrator before you open the solution file with it.

ii. We suggest building the ‘opencv_core’ and ‘opencv_gpu’ first. Otherwise it’ll take hours before you find an error.

iii. You can right click ‘ALL_BUILD’ to build the entire OpenCV.

iv. After that build ‘INSTALL’

v. Build ‘ALL_BUILD’ and ‘INSTALL’ in Release mode again.

vi. The building took around 2 hours for each mode on my PC.

(j) The final compiled OpenCV lib(+CUDA) can be found in ‘Output Folder’\install

Creating a project a OpenCV/CUDA capable VS project

1. Create new Visual C++ Win32 Console Application

2. In the Solution Explorer, right click on the project and select ‘Properties’

(a) Add ‘<dir>\opencv\build\install\include’ to ‘Configuration Properties’ ->‘C/C++’ ->‘Additional Include Directories’

(b) Add <dir>\opencv\build\install\x64\vc12\lib’ to ‘Configuration Properties’ ->‘Linker’ ->‘Additional Library Directories’

(c) Add the following libs to ‘Configuration Properties’ ->‘Linker’ ->‘Input’ ->‘Additional Dependencies’

i. `opencv_calib3d2410d.lib;opencv_contrib2410d.lib;opencv_core2410d.lib;opencv_features2d2410d.lib;opencv_flann2410d.lib;opencv_gpu2410d.lib;opencv_highgui2410d.lib;opencv_imgproc2410d.lib;opencv_legacy2410d.lib;opencv_ml2410d.lib;opencv_nonfree2410d.lib;opencv_objdetect2410d.lib;opencv_photo2410d.lib;opencv_stitching2410d.lib;opencv_superres2410d.lib;opencv_ts2410d.lib;opencv_video2410d.lib;opencv_videostab2410d.lib`

3. Change from Win32 to x64 in the configuration manager

4. OpenCV code including cuda functions should now be able to compile and run

This configuration should be done in VS debug mode, the same configuration needs to be done in release mode if you wish to build release software with the small difference, that the OpenCV libs then should be named e.g. opencv_calib3d2410.lib instead of opencv_calib3d2410d.lib. The d is for some extra debug functionality.

## D.4 Installing Ubuntu through Windows using Wubi

For installing Ubuntu we utilize Wubi. Wubi is an officially supported installer for Windows XP, Vista and 7 users that allows Ubuntu to be installed and uninstalled in a safe, easy way as with any other Windows application. To read more about Wubi, go to https://wiki.ubuntu.com/WubiGuide, whereof most of the content in this section is derived from. This guide contain instructions for two versions of Ubuntu namely, 12.04 and 14.04. The old version 12.04 is the most straight forward to install, but it may have issues on modern SSDs. 14.04 comes with a newer file system and will work better, however 14.04 is not well supported for use with Wubi, therefore a number of hacks must be applied to get it running.
For Ubuntu 12.04 LTS the steps are as follows:

1. Boot up in Windows 7.
2. Go to http://releases.ubuntu.com, and select the version you wish to use. In this case "12.04/".
3. Scroll to the button, and download "wubi.exe".
4. Right click on wubi.exe, and select "Run as administrator".
5. A windows called, "Ubuntu Setup" should now start up. In the top of the Window it should say "You are about to install Ubuntu-12.04".
6. The installation drive should be set to **c:**, which is the startup drive on the Windows 7 partition.
7. Select a proper installation size. Note: Ubuntu recommends 15 GB or minimum of 8 GB.
8. Select what language you wish to have in you Ubuntu OS.
9. Enter a username and password.
10. Press install, and a download of the installation files(Approx. 700 mb) should begin. Afterwards you are requested to reboot.
11. Remember to hold the command key on rebooting until the first boot loader is seen. Select Windows partition in first boot loader, and Ubuntu in second boot loader. The installation will continue for another 10-15 minutes and the machine will reboot again.

For Ubuntu 14.04 LTS the steps are as follows:

1. Boot up in Windows 7.
2. Go to http://releases.ubuntu.com, and select the version you wish to use. In this case "14.04.1/".
3. Scroll to the button, and download "wubi.exe".
4. In the same list locate and download "ubuntu-14.04.1-desktop-amd64.iso".
5. Make sure that the two are located in the same folder and disconnect from the internet".
6. Right click on wubi.exe, and select "Run as administrator".
7. A windows called, "Ubuntu Setup" should now start up. In the top of the Window it should say "You are about to install Ubuntu-14.04".
8. The installation drive should be set to **c:**, which is the startup drive on the Windows 7 partition.
9. Select a proper installation size. Note: Ubuntu recommends 15 GB or minimum of 8 GB.
10. Select what language you wish to have in you Ubuntu OS.
11. Enter a username and password.
12. Press install, and wait a short while until you are requested to reboot, reconnect to the internet before restarting.
13. Remember to hold the command key on rebooting until the first boot loader is showing. Select Windows partition in first boot loader, and Ubuntu in second boot loader. The installation will continue for another 10-15 minutes and the machine will reboot again.
14. Again remember to hold the command key on rebooting until the first boot loader is showing. Select Windows partition followed by Ubuntu in second. At this point you should be presented with a grub menu, where you can either wait/press 'enter' to continue or among one of the alternatives press 'e' to edit the Ubuntu entry. In the entry change permission from ro to rw as shown in the next step.

15. Change: linux\boot\vmlinuz-3.13.0-32-generic root=UUID=55B018A020A3F99A loop=\ubuntu\disks\root.disk ro rootflags=sync quiet splash $ vt_ handoff to: linux\boot\vmlinuz-3.13.0-32-generic root=UUID=55B018A020A3F99A loop=\ubuntu\disks\root.disk rw rootflags=sync quiet splash $ vt_ handoff

16. Alternative: If you were not presented with the option to press 'e' you must hold down 'shift' directly after selecting Ubuntu in the second bootloader to reveal it.

17. When done, press 'control' + 'x' to proceed with boot up.

18. Optional: To avoid having to do the last couple of steps every time you boot, make it semi-permanent by entering to commands seen below.

```
$ sudo sed -i s/' ro '/' rw '/g /etc/grub.d/10_lupin

$ sudo update-grub
```

**Listing D.4:** Semi-permanent 14.04 boot fix

```
$ sudo nano /etc/default/grub

update GRUB_CMDLINE_LINUX to hold:

GRUB_CMDLINE_LINUX="libata.force=noncq"

$ sudo update-grub
```

**Listing D.5:** SSD fix

Ubuntu in one form or another should now be installed on your Windows 7 partition. You can check this by navigating, in Windows 7, to your c: directory and see that a folder called "ubuntu" is created.

Before starting working and compiling your code in Ubuntu, the authors strongly recommends installing MacFan which is a program for controlling fan settings. To do this, follow the steps in Listing D.6.

```
$ sudo add-apt-repository ppa:mactel-support/ppa

$ sudo apt-get update

$ sudo apt-get install macfanctld

$ macfanctld
```

**Listing D.6:** Install MacFan

Note that the authors tried to install a 32-bit version of Ubuntu. There however seemed to be some problems with the drivers, such that the installation of Ubuntu 32-bit could not be finished. The authors therefore recommend using the 64-bit from the beginning.
If you however wish to install the 32-bit version of Ubuntu, you should use the following steps. But please note, that the authors has not been able to get Ubuntu version to work.

1. Boot up in Windows 7.
2. Go to `http://releases.ubuntu.com`, and select the version you wish to use. In this case "12.04/".
3. Scroll to the button, and download "wubi.exe".
4. Right-click Wubi.exe and select "Create Shortcut".
5. Right-click the shortcut, select Properties, and modify the Target line, i.e: `"C:\Documents\<user>\Desktop\wubi.exe"--32bit`
6. Right click the shortcut and select "Run as administrator".
7. Follow the same steps as above (from step 5).

## D.5 Installing CUDA in Ubuntu

The first step is to download the CUDA toolkit repo, and as we have installed a 64-bit version of Ubuntu, a 64-bit version of the Cuda Toolkit is used. Download the DEB file for CUDA 6.5 (cuda-repo-ubuntu1204_6.5-14_amd64.deb) from NVIDIA's homepage. Next, open a terminal and follow the steps seen in Listing D.7. Note, never do an "sudo apt-get upgrade", experience indicates this crashes the touchpad and other drivers in Ubuntu.

```
$ sudo dpkg -i cuda-repo-ubuntu1204_6.5-14_amd64.deb

$ sudo apt-get update

$ sudo apt-get install cuda
```

**Listing D.7:** Install CUDA.

After installing CUDA, we wish to include CUDA in our environment, this is done by adding some paths to your .bashrc file, as seen in Listing D.8.

```
$ sudo gedit .bashrc
Copy following lines into .bashrc:
export PATH=/usr/local/cuda-6.5/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-6.5/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=/usr/local/cuda-6.5/lib64:$LD_LIBRARY_PATH

$ sudo reboot
```

**Listing D.8:** Modify bashrc

You should now reboot your computer into Ubuntu again. To do this, it is important to remember to hold down the alt/option key while rebooting, and select the Windows option in the first boot loader, followed by Ubuntu in the second boot loader. When booted up, we wish to verify that CUDA is correctly installed. To do so, follow the steps seen in Listing D.10.

```
$ cd /usr/local/cuda-6.5/samples/1_Utilities/deviceQuery

$ sudo make

$ ./deviceQuery
```

**Listing D.9:** Verify CUDA installation.

After performing these steps, you should see some specifications of your Nvidia graphics card and the CUDA installation in the terminal.
In some cases desktop elements such as the menu and side bar will disappear in the following restarts after installed a new graphics driver. This problem is still not solved but if it happens it might be helpful to do some of the following.

```

When at the black desktop press 'ctrl'+'alt'+'f1' to access tty1
```

```
3  Install the compiz settings manager if necessary
4  $ sudo apt-get install compizconfig-settings-manager
5
6  Run it:
7  $ export DISPLAY=:0
8  $ ccsm
9
10 To access it press  'ctrl'+'alt'+'f7' to get back to the desktop
11
12 find and enable the Unity plugin
13
14 everything should spring back into place but it does not.
```

**Listing D.10:** Graphics problems.

## References

The following citations have been used in this section:

http://www.cimat.mx/~fcoj23/Tutorials/Install_OpenCV&CUDA_PSIVT2013.pdf
http://jasonjuang.blogspot.com/2013/09/how-to-install-cuda-55-on-ubuntu-1204.html

## D.6 Installing OpenCV with CUDA support in Ubuntu

Installing OpenCV with Cuda support on Ubuntu should be rather straightforward, experiments however show that a lot of problems come up. To prevent issues, two main things must be in order. First of all, you will need to install quite a lot of packages on your computer BEFORE installing OpenCV. Naturally you will need to make sure that your installing of CUDA is already working (see section D.5). OpenCV should work with both cpp and python compiler.
The approach that we have had success with is seen in below listnings. It should be noted, that all the packages may not be required for your specific application, this approach is directed for our Master Thesis.
In Listing D.12 the installation of all the preliminary packages are seen.

```
$ sudo apt-get update

$ sudo apt-get install build-essential cmake pkg-config

$ sudo apt-get install libtiff4-dev libjasper-dev libgtk2.0-dev libswscale-dev
libv4l-dev libxine-dev libdc1394-22-dev zlib1g-dev libpng-dev libopenexr-dev
libgdal-dev libtheora-dev libx264-dev yasm libfaac-dev libv4l-dev libxine-dev
libtbb-dev libeigen3-dev python-numpy python3-dev python3-tk python3-numpy
```

**Listing D.11:** Install required package.

ffmpeg

```
$ sudo apt-add-repository ppa:mc3man/trusty-media
$ sudo apt-get update
$ sudo apt-get install ffmpeg gstreamer0.10-ffmpeg
```

**Listing D.12:** Install required package.

You should now have installed all the required packages for installing OpenCV. Next, you need to download OpenCV from `http://opencv.org/downloads.html`. For our approach we used OpenCV 2.4.9 for Linux. When you have downloaded your OpenCV distribution, we advice to move the directory from the "Downloads" directory, as you experience show that this folder are sometimes completed erase as a result of a lot of unorganized files.
Start a terminal (ctrl+t), and go through the steps seen in Listing D.13. Before executing "make -j8", you should look through the makefile to check whether all the modules are correctly selected. I.e check if CUDA is included. Please note, that this process may take a few hours to complete.

```
$ cd <opencv_location>/

$ mkdir build

```

```
$ cd build

$ cmake -D CMAKE_CXX_COMPILER=/usr/bin/g++ -D WITH_CUDA=ON -D
  BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON
  -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_QT=ON -D
  WITH_OPENGL=ON -D BUILD_DOCS=ON BUILD_EXAMPLES=ON -D BUILD_opencv_python=ON
  -D BUILD_opencv_java=OFF ..

$ make -j8

$ sudo make install
```

**Listing D.13:** Install OpenCV.

OpenCV should now successfully be installed on your Ubuntu distribution. For checking and verifying this, open a terminal and go through the steps seen in Listing D.14, or alternatively open a terminal, import cv in a python shell.

```
$ cd <opencv_location>/samples/c

& ./build_all.sh

& ./facedetect lena.jpg
```

**Listing D.14:** Verify OpenCV is correctly installed.

If you are having problems with missing dependencies, take look at Listing D.15. This Listing show 6 dependencies our approach could not locate, so we found them manually and created a symbolic link for them.

```
$ sudo ln -s /usr/local/cuda-6.5/lib64/libcufft.so /usr/lib/libcufft.so
$ sudo ln -s /usr/local/cuda-6.5/lib64/libnpps.so /usr/lib/libnpps.so
$ sudo ln -s /usr/local/cuda-6.5/lib64/libnppi.so /usr/lib/libnppi.so
$ sudo ln -s /usr/local/cuda-6.5/lib64/libnppc.so /usr/lib/libnppc.so
$ sudo ln -s /usr/local/cuda-6.5/lib64/libcudart.so /usr/lib/libcudart.so

$ sudo ldconfig
```

**Listing D.15:** Create symbolic links for missing dependencies.

Finally, verify that CUDA is correctly working with OpenCV. To do this, follow the steps seen in Listing D.16. Please note, that you have to copy and modify the build_all.sh file to the samples/gpu directory.

```
$ cd <opencv_location>/samples/gpu

& ./build_all.sh

& ./stereo_match <image_left> <image_right>
```

**Listing D.16:** Verify OpenCV is correctly installed.

A final note, be aware that graphic cards has a varying CUDA compute capability, and all graphic cards are therefore not able to do utilize all methods on the GPU. You can check your compute Capability at `https://developer.`

nvidia.com/cuda-gpus. I.e the graphic card used throughout this approach is a NVIDIA GeForce GT 750M, which has a compute capability of 3.0.

## D.7 Installing Flycapture and Triclops APIs form Point Grey

These are the steps needed for getting the proprietary Flycapture and Triclops libraries working on Ubuntu 12.04 64 bit.

### D.7.1 Flycapture

Before installing Triclops, Flycapture needs to be installed. However, it also have some dependencies which should be installed first as shown in Listing

```
1 Install flycapture prerequisites:
2 $ sudo apt-get install libraw1394-11 libgtk2.0-0 libgtkmm-2.4-dev libglademm
      -2.4-dev libgtkglextmm-x11-1.2-dev libusb-1.0-0
3
4 Execute flycapture install script:
5 $ sudo sh install_flycapture.sh
```

**Listing D.17:** installing Flycapture and dependencies

### D.7.2 Triclops

```
1 $ sudo dpkg -i triclops-3.4.0.1_amd64.deb
```

**Listing D.18:** installing triclops

In addition to the two required Point Grey libraries, a program called Coriander can be useful for testing camera settings.

### D.7.3 Coriander

```
1 $ sudo apt-get install coriander
```

**Listing D.19:** installing coriander

## D.8 Installing PCL

PCL is short for Point Cloud Library..

### D.8.1 PCL

Before installing Triclops, Flycapture needs to be installed. However, it also have some dependencies which should be installed first as shown in Listing

```
1  sudo apt-get update && sudo apt-get install build-essential
2
3  sudo apt-get install g++ libboost-all-dev libeigen3-dev libflann-dev libvtk5-
       dev
4  libqhull-dev libgomp1 openni-dev
5
6  Install prebuilt binaries:
7
8  sudo add-apt-repository ppa:v-launchpad-jochen-sprickerhof-de/pcl
9  sudo apt-get update
10 sudo apt-get install libpcl-all
```

**Listing D.20:** installing PCL and dependencies