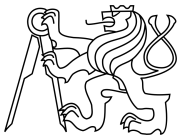
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF MASTER'S THESIS

Title:	Detection of a license plate position from camera records of moving cars
Student:	Bc. Vladislav Jásek
Supervisor:	doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Study Programme:	Informatics
Study Branch:	System Programming
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2017/18

Instructions

The aim is to design and implement an algorithm that enables detection of license plates on moving cars from camera to records (videos). The car that has installed the camera is also moving.

1) Study the task of detecting license plates and approaches used to detect them.
2) Propose a robust algorithm to detect a position of a license plate from a camera record. The camera is installed in a moving car.
3) Implement the proposed algorithm in an appropriate programming language. Utilize the OpenCV library.
4) Verify the implemented algorithm on real data. Evaluate the results and discuss the advantages and disadvantages of the algorithm used.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague February 7, 2017

Czech Technical University in Prague
Faculty of Information Technology
Department of computer science

Master's thesis

Detection of a license plate position from camera records of moving car

Bc. Vladislav Jásek

Supervisor: doc. RNDr. Ing. Marcel Jiřina, Ph.D.

9th May 2017

Acknowledgements

I would like to thank doc. RNDr. Ing. Marcel Jiřina, Ph.D. and Ing. Jakub Novák for their help, support and patience during our consultations.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 9th May 2017 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Jásek, Vladislav. *Detection of a license plate position from camera records of moving car.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Tato práce se zabývá problémem detekce registračních značek z kamery jedoucího auta. Některé stávající přístupy k problematice jsou v této práci shrnuty. Dále je představeno několik state-of-the-art algoritmů pro detekci a tracking objektů. Je navržen, implementován a otestován detekční framework řešící daný problém.

Klíčová slova počítačové vidění, rozpoznávání registračních značek, trackování objektů, onboard kamera, zpracování obrazu, detekce

Abstract

This thesis deals with the problem of license plates detection in moving scene. Some of the current approaches to the problem are summarized in this thesis. Several state-of-the-art techniques of tracking and detection are introduced and detection framework solving the problem is proposed. Practical implementation is described and experimentally evaluated.

Keywords computer vision, licence plate recognition, object tracking, onboard camera, image processing, detection

Contents

List of Figures

Introduction

Computer vision is a very promising and quickly growing interdisciplinary field that finds its uses in a very large scale of applications, for example:

OCR is a process of transferring images containing characters to textual form of data, which is useful for preserving written books.

Another field in which computer vision has very broad and useful application is medicine where image processing algorithms are comprised, for example, of very complex scans of human organs that are subsequently used for preventive monitoring and eventual diagnosis of diseases.

Next field with important use of computer vision is biometrics, the authentication method based on the unique physiological features of an authenticated living organism (typically human), for example, by fingerprint or eye cornea.

Motion capture is a technology that finds widespread use when shooting movies, in the advertising industry, but also in purely practical tasks such as military. The goal of its application is to create a 3D model of a moving object. This is achieved by the fact that the object being filmed (typically an actor or animal) is covered by small contrasting markers, whose trajectories are then captured by cameras from different angles. Subsequently, a 3D object model is created using photogrammetric algorithms.

We can not forget the use in photography and in art, thanks to which today our camera and mobile phones can create panoramas, detect smiles and have many other features.

Another promising field of computer vision is automobile safety. Because of high level of active safety, cars can recognize brands, lanes and keep themselves away from other cars. There is even the first car with almost complete version of autopilot. Nothing like that would be possible without computer vision.

We can only say that everybody in our neighborhood is in contact with computer vision (often unknowingly) every day and a very large number of applications can also be tested by a person who does not have a great technical knowledge.

In this thesis, we propose methods for moving object detection and object

tracking with applications in visual surveillance and consider robustness and computational cost as the major design goals of our work.

Problem statement

The aim is to design and implement an algorithm that enables detection of license plates on moving cars from camera to records (videos). The car that has installed the camera is also moving.

Goals of the thesis

- Study the task of detecting license plates and approaches used to detect them.
- Propose a robust algorithm to detect a position of a license plate from a camera record. The camera is installed in a moving car.
- Implement the proposed algorithm in an appropriate programming language. Utilize the OpenCV library.
- Verify the implemented algorithm on real data. Evaluate the results and discuss the advantages and disadvantages of the algorithm used.

Thesis outline

In order to describe the accomplished goals, the thesis will be structured to chapters in the following manner:

- **Basics**
  In this chapter we will explain some formal definitions, basic terms and fundamental concepts used in the thesis, mainly related to image processing, computer vision and data mining.
- **Analysis**
  In this section the reader will be introduced to the problematics of license plate recognition, typical schema of automatic license plate recognition system (ALPRS) will be introduced.

  Current research in the field of license plate detection and car detection will be summarized.

  Several state-of-the-art algorithms and frameworks that can be useful for object detection and tracking will be presented.
- **Implementation**
  In this chapter, the implementation based on the design proposed in

previous chapter will be presented. First the use cases and requirements will be summarized. Then available technologies will be presented and compared. In following section, the learning phase of the detector will be discussed. Structure of the program will be presented and all used classes will be described.

- **Testing**
  During the development, every finished component underwent extensive unit testing. These tests are summarized in this chapter.

- **Experimental results**
  Experimental results of the final product will be reviewed in this chapter.

Chapter 1

Basics

In this chapter we will explain some formal definitions, basic terms and fundamental concepts used in the thesis, mainly related to image processing, computer vision and data mining.

1.1 Image processing

The purpose of this section is to present selection of some basic image processing techniques. Image processing is processing of images or series of images, using mathematical operations.

Most image-processing techniques involve isolating the individual signal components creating the image and applying standard signal-processing techniques to them.

Images are also often processed as three-dimensional signals with the third-dimension being time or luminance/chrominance.

The output of image processing is also an image or some characteristics or features, extracted from the image.

1.1.1 Image

By the term image, in the scope of this thesis, we will always mean bitmap (also called raster) image.

- From the mathematical point of view, the image is continuous function of two variables:

$$i(x, y) : R^2 \rightarrow R^n \tag{1.1}$$

- In computer representation, the term image typically means two dimensional matrix of equidistant samples:

$$i(x, y) : N^2 \rightarrow R^n \tag{1.2}$$

Where R^n is a vector of features, that denote position of the color in color space.

1.1.2 Color space

Is a useful conceptual tool for defining the color capabilities of image processing device or data file. A color space relates numbers, vectors, names etc. to actually produced color, either being mapped to palette or organized by topology, based on some mathematical features, thus allows to reproduce representations of color between various devices and data formats.

Color spaces can be either device dependent, expressing color relative to some other space or device independent, expressing color in absolute terms against some defined color standard.

Here we will explain some most common color spaces with applications in image processing:

- **RGB**
  Is an additive color model, in which the components of Red, Blue and Green light are mixed together. Three-dimensional volume is described by treating the component values as ordinary Cartesian coordinates in a euclidean space.

- **HSV**
  This space is a cylindrical-coordinate space. The abbreviation stands for Hue, Saturation, Value and is also called HSB [6] (Hue, Saturation, Brightness).

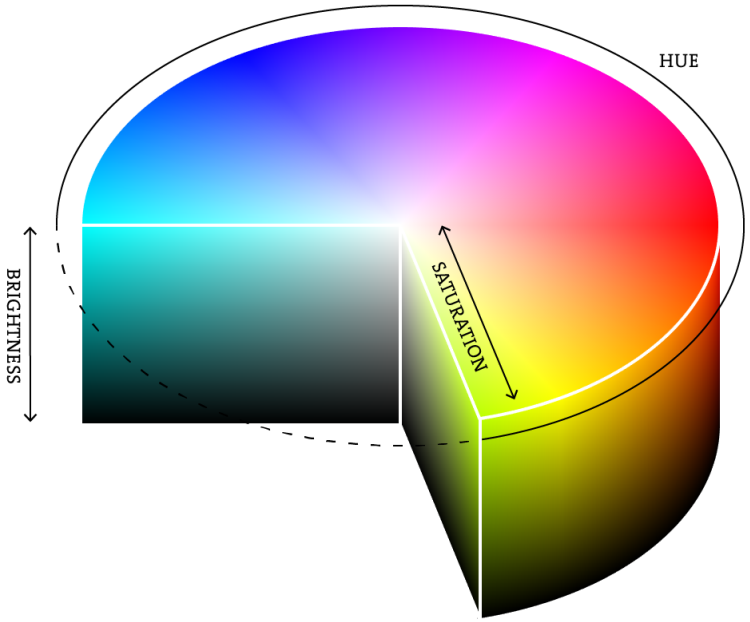


Figure 1.1: HSB color space [1]

The angle around the cylinder axis defines Hue, which is the wavelength within the visible-light spectrum at which the energy output from a source is greatest. Saturation is the relative bandwidth of the visible output from a light source, and is expressed by distance from the cylinder center. Brightness is the amplitude at the wavelength where the intensity is greatest [7].

- **HSL**
  This space is very similar to HSV. The Hue has the same definition as in HSV. S also stands for Saturation, however Lightness stands for brightness relative to the brightness of a similarly illuminated white.

  Both HSL and HSV are simple linear transformations of RGB models.

- **YUV**
  The Y component determines the brightness. The color information is separated out into 2 channels– U and V are blue luminance and red luminance differences respectively [8] . This color space was invented for color television broadcast because of the backward compatibility with monochromatic devices.

1.1.3 Object tracking

Is the task of computing trajectory of the given object in a sequence of video frames [9] . The input of the **tracker** is the appearance of the object in the

first frame. Output is the curve or the sequence of coordinates of the object in consecutive frames.

Trackers typically assume that the object is visible through- out the sequence. If the tracker allows the temporary disappearance of the object from the scene, then the task is called **long term object tracking**.

1.1.4 Object detection

Is the task of detecting all objects of interest in a single video frame [10] . **Detectors** typically require offline training stage, so they cannot be applied to unknown objects. Output from the detector is the set of coordinates of all detected objects.

Object detection is typically much more computationally expensive task than object tracking.

1.1.5 Image segmentation

Image segmentation is the process of partitioning the image into logical regions. There are many different ways to perform image segmentation based on for example:

- Color properties (Thresholding)
- Spatial properties
- Gradient domain properties (edge/corner detection)

1.1.6 Histogram

In the image processing histogram shows frequency of pixels intensity values, thus is probability density function of pixel luminance (or other pixel feature). The x axis shows the range of values of the feature and the y axis shows the frequency of these values.

1.1.7 Histogram equalization

Is the process of transforming cumulative distribution function of pixel luminance to linear function, which has the effect of increasing contrast.

1.2 Converting image to grayscale

There are several different approaches to convert image to grayscale. The most typical are [11]:

Algorithm 1 Histogram equalization

```
Is[0] = P[0]
for k = 1 to L do
    T[k] = T[k-1] + P[k]
end for
for r = 0 to L do
    S[r] = round(T[r]*L)
end for
for y = 0 to M do
    for x = 0 to N do
        r = I[y, x]
        He[y, x] = S[r]
    end for
end for
```

- **Averaging RGB values**
  is the most trivial approach, however it generally does not preserve contours and does not represent shades of gray relative to the way humans perceive luminosity.

- **Weighted averaging of RGB values**
  Because the human eye does not perceive red, green and blue intensities equally, the better way of conversion is to multiply the values by the relative weights of eye sensitivity to particular color. The most common formula is:

$$L = R * 0.3 + G * 0.59 + B * 0.11 \tag{1.3}$$

  although slightly different constants can be used.

- **Desaturation**
  Consists in setting the saturation in HSV image to zero.

- **Selecting a channel**
  This is what most digital cameras use for taking gray scale photos. The channel typically selected is green, because the human eye is most sensitive to it.

- **Converting in gradient domain**
  Is computationally expensive, however preserves all the contours in the original image.

1.2.1 Image normalization

Is the process of rescaling and shifting the luminance values of the image to a desired value range. Linear normalization can be computed as:

$$I' = (I - I_{min})\frac{I'_{max} - I'_{min}}{I_{max} - I_{min}} + I'_{min} \tag{1.4}$$

Where I is original luminance of the pixel, I' is the new luminance of the pixel, $I_{max}, I'_{max}, I_{min}, I'_{min}$ are maximal and minimal allowed intensities in new respective old image.

Nonlinear normalization also exists.

1.2.2 Convolution and Correlation

Convolution ($*$) is a weighted average with one function(discrete or analog) constituting the weights and another the function to be averaged [12]. In the scope of the thesis, we will consider only discrete convolution in 2D, which is commonly used in image filtering. 2D convolution is mathematically defined as:

$$G[i,j] = \sum_{u=-k}^{k}\sum_{v=-k}^{k} h[u,v]F[i+u][j+v] \tag{1.5}$$

It describes how our image responds to a filter. Correlation is just convolution with the filter reversed and describes similarity of two signals/images:

$$G[i,j] = \sum_{u=-k}^{k}\sum_{v=-k}^{k} h[u,v]F[i-u][j-v] \tag{1.6}$$

Note that convolution is both associative and commutative, in contrast to correlation.

In image processing, kernel filter is typically defined by a finite matrix [13] and the origin point. That is the position of the kernel which is above the current output pixel. This could be outside of the actual kernel, though usually it corresponds to one of the kernel elements. For a symmetric kernel, the origin is usually the center element. The effect of filtering is obtained by convolving the image matrix with the kernel matrix.

1.3 Data Mining

Data mining is the process of discovering patterns, dependencies and extracting useful features from seemingly unstructured and complex data. The data are analyzed from different perspectives and summarized into useful information. The basic two problems of data mining are classification and regression.

1.3.1 Classification problem [14]

Suppose we have a (potentially infinite) set of vectors X, that can be covered by finite number of disjunct classes:

$$X \subset C_1 \cup C_2 \cup \ldots \cup C_n \& C_i \cap C_j = \emptyset, i \neq j \tag{1.7}$$

And we want to classify any $x \in X$, based only on finite set of known x and their corresponding class C.

Thus our goal to construct a **classifier**, the projection from:

$$f : X \to Y; Y = \{C_1 \ldots C_n\} \tag{1.8}$$

Examples of some simple classifiers are:

- **k-NN**
- **Decision tree**
- **Naive Bayes classifier**

1.3.2 Regression problem

Regression [14] can be seen as an approximation of an unknown mapping

$$g : X \leftarrow Y \tag{1.9}$$

, using a known mapping $f \in F$, where X and Y are some vector spaces and the mapping f is chosen from F based on a sequence:

$$f = (x^{(1)}), y^{(1)}), \ldots, (x^{(1)}), y^{(1)}); f \in X * Y \tag{1.10}$$

Examples of some simple regression techniques are:

- **Least Mean Squares**
- **MLP**

1.3.3 Ensembling

In machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone.

1.4 Spatial domain indexing algorithms

This section is dedicated to introduction of some basic spatial indexing algorithm. These algorithms allow us to effectively compute collisions, matches and other useful properties and relations between objects placed in (typically) euclidean space.

1.4.1 Grid

Probably the simplest non trivial indexing structure available. The image is divided into equidistant samples along both axes. Their intersections then define the bounding boxes of our grid.

The reference to indexed object is simply put into correct bounding box, which is calculated by normalizing the object coordinates by the number of samples in each dimension.

1.4.2 Quad Tree

Quad tree is a sophisticated technique of spatial domain indexing. Every node of the tree represents a certain bounding box of our image and is either a node or has exactly four children [15].

Children divide bounding box of their parent into four quadrants, that are obtained by splitting the parent bounding box along both x and y axis in half, which means that all the nodes on the same level have exactly same size of their bounding box.

The references to indexed objects are stored only in leaves.

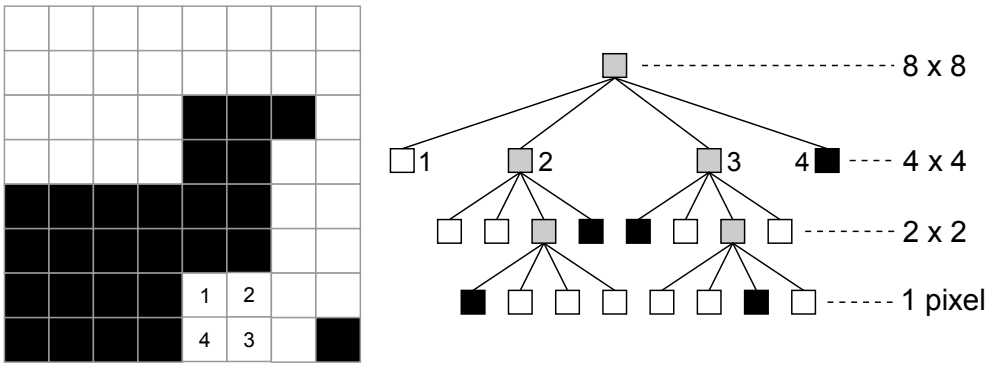


Figure 1.2: Example of quad tree structure[2]

- **Insertion**
  To insert a reference to point into a tree only means to traverse from the root to the certain leaf node, deciding to which bounding box of the children our node belongs. If the inserted objects spans along more quadrants, the references are put in each of them.

  If a certain number of objects is put into one quadrant, this quadrant is then split into four and all its points are passed to the children.

- **Query**
  To query a quad tree for a single object, we only need to traverse to the leaf quadrant like when inserting and then compare all the objects referenced in this quadrant.

Analogous structure in 3D is called **Octree**.

1.4.3 K-D Tree

K-D Tree is another spatial indexing tree. It is a binary tree, where every node contains exactly one point. Each level of a K-D tree splits all children along a specific dimension, using a hyperplane that is perpendicular to the corresponding axis. The corresponding point coordinate in this axis equals median of the inserted set. The two divided subsets are then passed to the children, where are recursively split along next axis and so on.

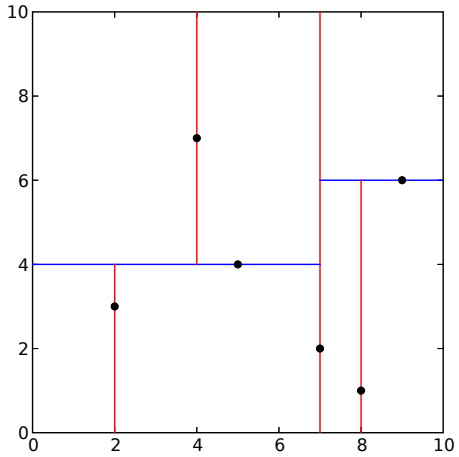


Figure 1.3: k-d tree decomposition for the point set (2,3), (5,4), (9,6), (4,7), (8,1), (7,2) [3]

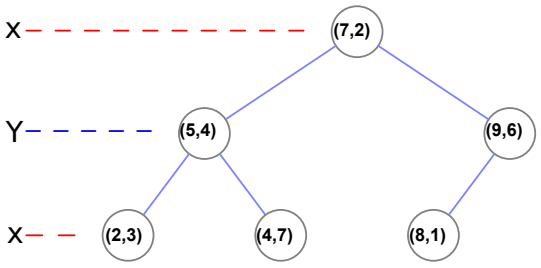


Figure 1.4: Corresponding k-d tree [4]

Chapter 2

Analysis

In this section the reader will be introduced to the problematics of license plate recognition, typical schema of automatic license plate recognition system (ALPRS) will be introduced.

Current research in the field of license plate detection and car detection will be summarized.

Several state-of-the-art algorithms and frameworks that can be useful for object detection and tracking will be presented. We will explain their basic concepts, define their typical applications and summarize their pros and cons.

2.1 Typical ALPR system

Most of the automatic license plates recognition systems are multi-stage systems [16], that get the image of the car and then process it according the following pipeline:

1. The license plate position is localized in the picture of the car.
2. The plate is cropped from the picture, deskewed and resized to some normalized size.
3. The resulting image undergoes normalization process, which typically includes histogram equalization and some thresholding. The result is typically binary image.
4. The individual characters are then extracted from the image, most obvious method of separation being morphological operations and floodfill algorithm.
5. The characters are then recognized by some classifier.
6. Syntactic and geometrical check of the plate is then performed.

7. Recognized character of the license plate are then typically aggregated over several frames in order to improve robustness.

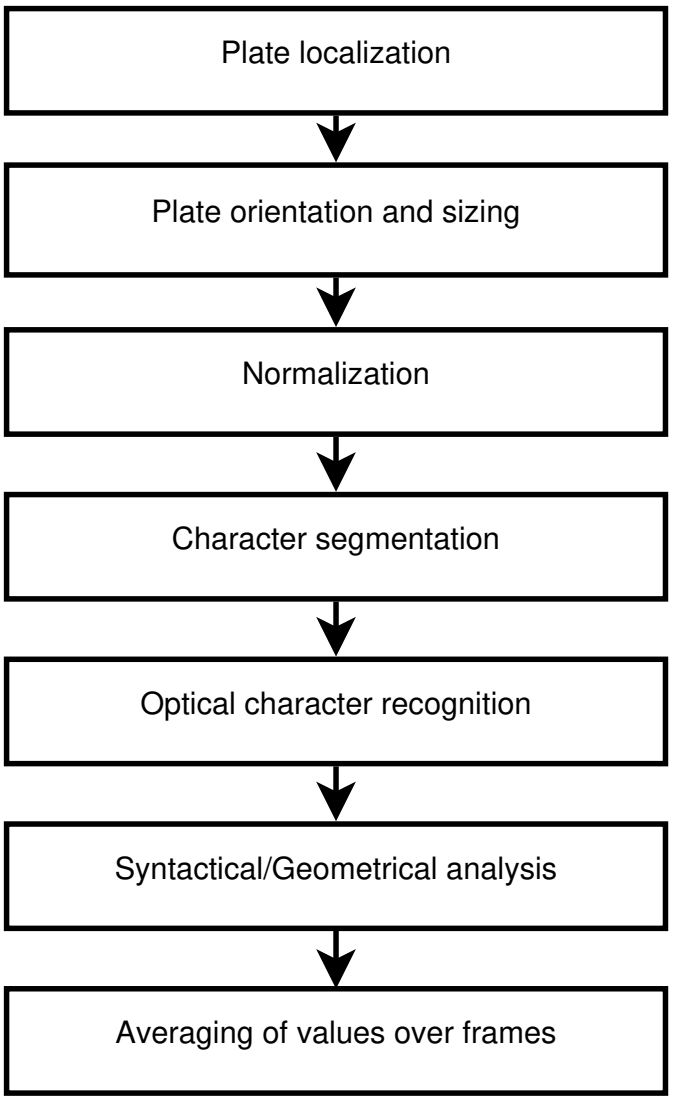


Figure 2.1: Typical automatic license plate recognition system pipeline

All of these steps are well known problems,so this thesis will rather focus on producing the input to standard ALPR system, locating the cars from the scene recorded by onboard camera, that is placed in a moving car.

This may look similar to the localization of the plate procedure, however, the scene is generally much more complex and changes dynamically. Also the objects of interest (cars) can have variety of colors, shapes and sizes as opposed to license plates, whose appearance follow fixed (and typically very

simple) form. So while sharing some similarities with plates localization this problem is much more complex.

2.2 Licence plate detection

Because there is a lot of similarities and overall large problem intersection with the first stage of ALPRS we would like to summarize related work in this field of research. Most of the existing solutions are based either on image segmentation or on detection of the edges.

2.2.1 Based on image segmentation

In this paper [17], a region-based license plate detection method is proposed. Using mean shift procedure in the joint spatial-range domain, the input color vehicle images are segmented into many regions. The selection criteria is related to the proportions of the size of license plates to that of vehicle images. The resulted regions are called candidate region. The following features are then extracted: rectangularity, aspect ratio and edge density and the decision is based on the Mahalanobis classifier.

Authors of this article [18] have proposed another method of license plate localization that is based on the connected components analysis. Color space of the image is changed to the YUV model but only the luminance is recorded.

The neighborhood of every point of the image is analyzed in the form of a square (5x5) to specify the biggest difference in brightness. If the specified contrast exceeds the threshold for a given point, it means that a sort of edge was found in the image which may mean some kind of border between the character and the background and this point is specially marked. If the contrast in the given point is too small then only the threshold operation takes place. White and black areas which are the results of the thresholding are then labeled.

The next step is the elimination stage, which goal is to leave in the picture only these spots which are most likely to be license plate characters, assuming that each spot represents a single character and is not connected with any other object in the image. The spots are eliminated on spatial properties, such as width and height.

Their neighborhood is then analyzed and the spots are grouped into segments, which are further eliminated.

2.2.2 Based on edge detection

Another common approach [19] to the problem is to find the position of the plate by analyzing features of the edges, as proposed in article. Preprocessing part incorporate the adjustment of rotated image. Then after preprocessing, Harris corner algorithm is applied to extract the feature from the image. After

extracting all the corner points, the sliding window approach is applied to find the most likely number plate region. Soft thresholding is taken as part of sliding window with the goal that it works for the majority of pictures. Aspect ratio (AR) limit is set to restrict the LP viably subsequent to separating every single corner point and amid SW approach.

In the article [20], the image is first converted into grayscale, then the density of both horizontal and vertical edges is measured by Sobel detector. In contrast to article, where instead of Sobel detector, the Laplacian of Gaussian (Mexican hat operator) is used as convolution kernel.

In order to make the edges continuous the morphological dilation operation is used.

$$K_{LoG} = \begin{bmatrix} 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -23 & -40 & -23 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \end{bmatrix} \tag{2.1}$$

Figure 2.2: Example of 7x7 Laplacian of Gaussian kernel (the Mexican hat operator)

Another edge based approach, mentioned [21] is based on intersection of areas with high vertical and high horizontal edge density. First, vertical edges are detected from the image and binarized. Then, license plate candidates are extracted by the two-stage detection process. In this process a sliding-window technique is used to mark all windows which satisfied edge density conditions. Edge density conditions are computed on integral edge image allowing us to significantly increase the processing speed of the method. To better distinguish between license plates and complex backgrounds, the edge analysis is performed to remove specific edges. Finally, false candidates are filtered out based on geometrical and textural properties. The proposed method can detect multiple license plates with different sizes in a complex background.

2.3 Vehicle detection

Another field of research similar to the problem of this thesis is vehicle detection. Authors of this article [22] summarized the possible features to look for in the vehicle detection as follows: symmetry, color, shadow, corners, edges,

vehicle lights and disparity (the difference in the left and right halves of the image between corresponding pixels).

In the article [23] is proposed approach based on applying the background subtraction method based on CS (compressive sensing), the measurements of the video is firstly obtained through the compressive sample operated on the input video images. The measurements of the background image will be achieved from the estimation of the former measurements. Besides, the background image needs real-time update about the changes in external environment. When conducting the background subtraction, the differential threshold operation should be undertaken on the measurements of background model and measurements of the real-time video frame image to determine whether there existed moving vehicle in the frame image.

In another article [24], approach with background motion compensation via background subtractor is combined with optical flow tracking to detect general moving objects from moving car.

Another approach is combined WaldBoost detector and the TLD tracker that are scheduled so that a real-time performance is achieved [25].

2.4 Viola-Jones algorithm

Proposed in 2001, Viola-Jones detection framework is the first algorithm being able to detect faces in real time on contemporary hardware [26]. Today, it is still considered state-of-the-art algorithm.

This detector works with very simple image features, called Haar-like features, because of they conceptual similarity with Haar wavelets, used in discrete wavelet transforms (DWT).

2.4.1 Haar-like features

A Haar-like feature is obtained by taking two or more adjacent rectangular, equally sized regions in the specific section of the grayscale image, summing pixel intensities of each region and then computing difference between each sum. Which is equal to applying convolution on the particular section of image with simple kernel that has predefined shape.

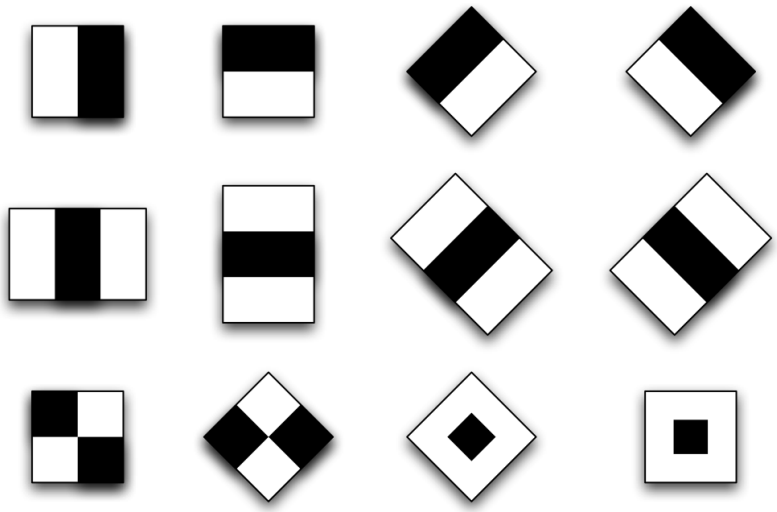


Figure 2.3: All Haar-like features [5]

Computed difference is then used as a feature for further categorization of the classified image.

Rotated Haar-like features also exist [27] , however, they are scarcely used, because in practical usage, the image (or more often the classifier, the effect being the same) is typically rescaled to some very small resolution, where multiplication with rotation matrix produces rounding errors.

2.4.2 Sliding window

In the context of computer vision (and as the name suggests), a sliding window is rectangular region of fixed width and height that "slides" across an image. The algorithm performs exhaustive search of the image, using sliding windows on the whole image with all possible scales, deciding whether the actual window contains desired features. When a certain amount of features match, the detector indicates a hit.

However, simple computing of the features of each window would be computationally expensive. However, a simple technique from dynamic programming allows us to compute every feature in constant time. That trick is called integral image and is considered a contribution of the authors of the detector.

2.4.3 Integral images

Also called **summed area table**, is a data structure for very efficient computing the sum of values in any rectangular subsegment of the image:

$$I(x,y) = \sum_{i=0}^{i \leq x'} \sum_{j=0}^{j \leq y'} i(x,y) \tag{2.2}$$

It can be computed by single pass over the image, the value at any point is just sum of all the pixels above and left (inclusive) [28].

Once the integral image is generated, sum of any rectangular subsegment can be computed in constant time ($O(1)$), using only values at four positions (corners of the).

$$\sum_{i=x_0}^{i\leq x'}\sum_{j=y_0}^{j\leq y'} i(x,y) = I(x_0,y_0) + I(x',y') - I(x_0,y') + I(x',y_0) \tag{2.3}$$

So, now we know, which features are used by the detector and how the detector evaluates them during detection stage. The promising features are selected by Adaboost algorithm.

2.4.4 AdaBoost

Adaboost (short for Adaptive Boosting) represents one of state-of-the art ensembling machine learning techniques [29].

Weak classifier is a classifier whose decision abilities are only slightly better than tossing a coin (0.5). Decision stumb, also called **1-rule**, is defined as a decision tree with only one internal node (root). Definitively is considered a weak classifier.

We will discuss the classic version of Adaboost, that operates only on binary classifiers (typically decision stumps), however version for multiple classification and regression problems also exists.

The key idea is that it combines several weak learners into one strong learner. This is achieved by generating multiple models from the training data, purpose of every new model is to correct the errorneous classification from the previous one. The process continues until all the training samples are correctly classified by the last model or maximum number of models is reached.

Each used decision stump is build during the training phase from one haar-like feature.

Algorithm 2 Adaboost algorithm

```
1: procedure AdaBoost(a, b)
2:     set uniform example weights
3:     for each base learner i do
4:         train i with weighted sample
5:         test i on all data
6:         set weight of i with weighted error
7:         update example weights
8:     end for
9: end procedure
```

2.4.5 Haar cascades

The exhaustive classification of all selected haar-like features would be very time expensive. In order to make the detector work with satisfactory speed in real-time, there must be introduced some hierarchy.

Multiple strong classifiers, each operating on subset of selected haar-like features are combined into sequential cascade. The image must pass every stage of the cascade to be positively classified as desired object.

2.5 HOG detector

HOG stands for Histogram of Oriented Gradients and today is one of the most common algorithms used in tasks that involve person detection.

The HOG detector again uses the sliding window approach, like the Viola-Jones algorithm did, however, instead of evaluating a set of local features, it relies only on one global feature of the actual window, the HOG descriptor [30].

2.5.1 Gradient image

Let us consider 8-bit grayscale image with intensity range $(0; 255)$. A gradient vector for each given pixel is simply a measure of change in the intensity along the x and y directions:

$$\Delta_{i,j} = \begin{bmatrix} x_{i-1} - x_{i+1} \\ y_{j-1} - y_{j+1} \end{bmatrix} \tag{2.4}$$

The magnitude $\mid \Delta_{i,j} \mid$ and angle $\varphi_{\Delta_{i,j}}$ then can be computed as:

$$\mid \Delta_{i,j} \mid = \sqrt{x_{i,j}^2 + y_{i,j}^2} \tag{2.5}$$

$$\varphi_{\Delta_{i,j}} = \arctan \frac{x_{i,j}}{y_{i,j}} \tag{2.6}$$

The difference between two adjacent pixels can be in range $(-255; 255)$, so the range must rescaled to fit the gradient image into two 8-bit images of original size.

The key insight here is that the gradient is invariant to absolute values of pixels in the original image. Normalization of gradient vector also makes it invariant to multiplication of pixel values (constrast adjustments) [30].

2.5.2 HOG descriptor

To compute the HOG descriptor, the window must be resampled to a predefined size (the size used in original algorithm is 64x128).

Then, we operate on fixed size square pixel cells (originally 8x8) within the detection window, the cells being organized in the manner that two adjacent cells complete overlay the middle cell (50% overlap of two cells).

Within a cell, we compute the gradient vector at each pixel and start putting these gradients into quantized histogram, according to $\varphi_{\Delta_{i,j}}$, the bin size of the histogram is typically 20°.

For each gradient vector, it's contribution to the histogram is given by the magnitude of the vector (so stronger gradients have a bigger impact on the histogram). We split the contribution between the two closest bins. The histogram is then normalized.

The HOG descriptor is the vector of histograms of all cells from the image.

The histogram will be very similar for the same object under different lighting conditions, making it easier to recognize the object despite changes in lighting [30].

The underlying classifier is Support Vector Machine [31], separating the classes by optimal hyperplane.

2.5.3 SVM

The purpose is to:

- Separate the data by two parallel hyperplanes.
- Maximize their margin.

Assume that our dataset D is linearly separable to two classes C_+ and C_- . Given a hyperplane H_0, separating our dataset:

$$w^T x + b = 0, \tag{2.7}$$

we can choose two hyperplanes H_- and H_+, that can also separate our data and satisfy:

$$H_- : w^T x + b = -1 \tag{2.8}$$
$$H_+ : w^T x + b = \quad 1 \tag{2.9}$$

from the definition, they are equidistant to H_0. The points from must met the following condition, to ensure that H_- and H_+ really separate our data:

$$w^T x_i + b \geq \quad 1; x_i \in C_+ \tag{2.10}$$
$$w^T x_i + b \leq -1; x_i \in C_- \tag{2.11}$$

These two constraints can be combined:

$$y_i(w^T x_i + b) \geq 1 \tag{2.12}$$

Let m be the perpendicular distance of H_- and H_+, let x be the point on H_-, then, because w is perpendicular to H_+, we can denote vector k as:

$$k = m\frac{w}{\|w\|} \tag{2.13}$$

If we add the vector k to x, we will get point on the hyperplane H_+. Thus:

$$w^T(x + k) + b = 1 \tag{2.14}$$

and after substitution of k we get:

$$w^T(x + m\frac{w}{\|w\|}) + b = 1 \tag{2.15}$$

Dot product of vector with self yields the square of its norm:

$$w^T x + b + m\frac{\|w\|^2}{\|w\|} = 1 \tag{2.16}$$

Because x belongs to H_-, then $w^T x + b + m = -1$:

$$-1 = 1 - m\|w\| \tag{2.17}$$

$$m = \frac{2}{\|w\|} \tag{2.18}$$

Thus maximizing the margin is equivalent to minimizing the norm of w.

2.6 Lucas-Kanade Optical flow tracker

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. It is 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second.

Algorithms based on optical flow make three implicit assumptions [32]:

- All intensity changes can be explained by intensity gradients.
- Neighboring pixels have similar motion.
- The time interval between two frames is short enough for the objects to do not displace significantly.

Now, consider a pixel in the first frame of the video. It moves by some distance in the next frame:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{2.19}$$

By approximating the function by first order Taylor polynomial, we obtain:

$$I(x+dx, y+dy, t+dt) \approx I(x,y,t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \tag{2.20}$$

And so:

$$\frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy \approx -\frac{\partial I}{\partial t}dt \tag{2.21}$$

Dividing through by dt, we obtain [33]:

$$\frac{\partial x}{\partial t} = u \tag{2.22}$$

$$\frac{\partial y}{\partial t} = v \tag{2.23}$$

Because u and v are unknown, we cannot solve this equation, however, we have set an assumption that neighboring pixels have similar motion, so we can take a 3x3 patch around the pixel of interest and find their equations:

$$S \begin{bmatrix} u \\ v \end{bmatrix} = t \tag{2.24}$$

Now the problem is overdetermined, but we can apply least squares fit method if the matrix S is invertible:

$$S^T S \begin{bmatrix} u \\ v \end{bmatrix} = S^T t \tag{2.25}$$

To get the best approximate solution:

$$\begin{bmatrix} u \\ v \end{bmatrix} = (S^T S)^{-1} S^T t \tag{2.26}$$

Lucas-Kanade method computes optical flow for a sparse feature set.

2.7 Tracking-Learning-Detection

Tracking-Learning-Detection algorithm (also called Predator tracker) is state-of-the-art method of long term tracking of one object [34].

The input to the algorithm is a video stream and user defined bounding box, defining the object of interest in the first frame. When run, the algorithm will then try to continuously track the object in the video stream, learning from incoming frames how the object's appearance changes.

As the name suggests, the algorithm is build from three cooperating blocks, running simultaneously:

2.7.1 Tracker

The object is tracked frame-to-frame. The used method of tracking is called **template tracking**. The object is represented by template (a image patch and its histogram). The motion is defined as a transformation, that minimizes mismatch between template and candidate patch.

The template tracking can either be static (the template does not change), adaptive (the template updates with each frame) or these two approaches can be combined.

2.7.2 Detector

Detector uses the sliding window approach (with reduced number of possible scales and shifts), each window is passed to cascaded classifier. The classifier has three stages [34]:

- **Patch variance**:
  This stage rejects all patches, for which gray-value variance is smaller by a user defined threshold than variance of the tracked patch . The gray-value variance can be expressed as:

  $$\mathbf{E}p^2 - \mathbf{E}^2 p \tag{2.27}$$

  And can be computed in constant time, using integral images. The purpose of this stage is to quickly reject non-promising patches.

- **Ensemble classifier**:
  The ensemble is composed of number of classifiers, each of them performs a number of pixel comparisons on the patch.

- **NN**:
  As the last stage, the nearest neighbor of the patch from the data model space is found. If the relative similarity is above certain threshold, the object detector scores a hit. The similarity between two patches is expressed as

  $$S(p_i, p_j) = \frac{1}{2}(NCC(p_i, p_j)) + 1, \tag{2.28}$$

  where NCC stands for Normalized Correlation Coefficient.

  Relative similarity is defined as:

  $$S^r(p, M) = \frac{S^+}{S^+ + S^-}, \tag{2.29}$$

  $S+$ and $S-$ stand for similarity with positive and negative neighbors from the object model M.

2.7.3 Learning

The purpose of the component is to improve performance of the detector by evaluating it in every frame of the stream.

The performance is evaluated by two functions, so called P and N experts. The P-expert and N-expert are capable to identify false positives and false negatives respectively.

Both of the experts make errors themselves, however, their independence enables mutual compensation of their errors.

Chapter 3

Proposal

In this chapter the framework that enables detection of the license plates from camera installed in a moving car will be proposed, based on some of the algorithms described in previous chapter.

3.1 Obtaining sample data

In order to get relevant data for our problem, we needed some video samples for continuous verification of our proposal, these were obtained by recording some typical scenes in which the proposed algorithm should be used.

The camera used was TrueCam A4, which is a low-to-middle class onboard camera, charged from 12V charging plug. The records are stored on micro SD card.

Figure 3.1: The TrueCam A4 camera

Features relevant for the testing are:

- FullHD 1080i (1980x1080)
- 30 FPS (frames pes second)

- 130° FOV (field of view)
- Output in H264-MPEG-4 AVC format (24 bit depth, RGB color space).

Figure 3.2: Typical frame from the onboard camera

3.2 Outline of the detection framework structure

We decided to combine the advantages of both detection and object tracking approach in order to make the whole proposed framework faster and more reliable. The main idea is to decide for each frame which approach use and switch between them in order to achieve real time performance.

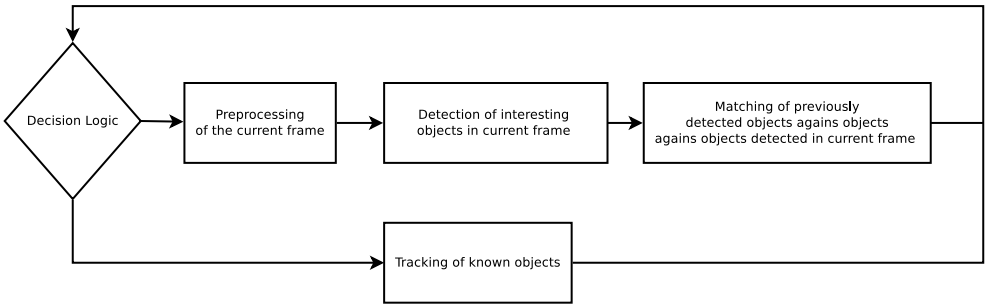


Figure 3.3: Simplified scheme of the framework

3.3 Preprocessing

In order for our detection system to work under different lighting and outdoor conditions, the whole frame of the scene must be preprocessed before it will be passed to the detector.

Because of the presumed complexity of the scene and the requirement for real time detection, it would be unwise to use some computationally expensive preprocessing techniques such as complex segmentation or bilateral filtering.

Instead, convolution with a small gaussian smoothing kernel will be used for reducing the noise.

Afterwards, the image will be converted to YUV color space and then the chrominance channels will be discarded.

After this procedure the histogram equalization will be performed, in order to prepare normalized data for the detector stage.

3.3.1 Detector

Viola Jones detection framework will be used as a detector because of sufficient performance. The detector will be trained to recognize our objects (cars) for initial detection in the whole preprocessed frame.

The detector will periodically check the scene to detect objects of interests, which will then be passed to matching stage.

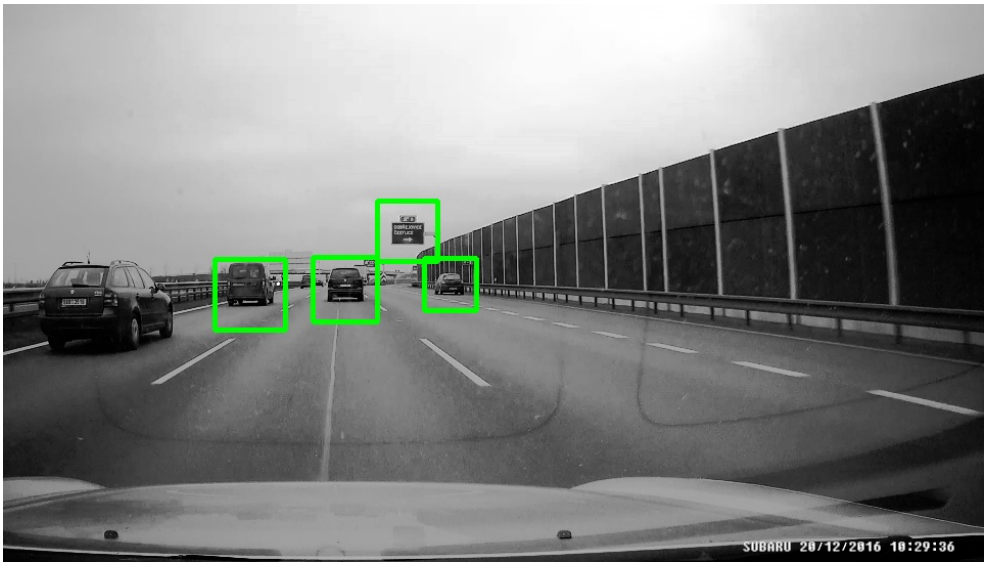


Figure 3.4: Image after the histogram equalization and grayscale conversion. The green rectangles are objects of interest recognized by Viola-Jones detection framework. Notice the false alarm in the center and undetected car to the left.

3.4 Matching

After detection of all objects of interest in the single frame, these regions will be passed to matching phase, in order to classify them against objects known from previous frames as one of:

- Newly detected objects
- Candidates for recognized objects
- Recognized known objects

Because we presume that our detector will be working with sufficient framerate for our objects to do not displace significantly, our matching phase will be based only on property of position and scale.

We will suppose that there can be many objects in the scene. In order to match them with better than quadratic time complexity, we will use the quad tree spatial indexing algorithm, the brute force matching will then take place only within each leaf of the tree.

The criteria for matching will be:

- **Percentage of intersection**
  Because of our constrain on sufficient frame rate, we can assume that one object has a fairly high intersection with its appearance in previous frame.
- **Scale**
  Also, the scale between two consecutive appearances of the object does not change drastically.

Another more sophisticated methods of matching could be proposed, for example histogram matching, however these methods generally have higher computational complexity and would not yield much better results according to our assumptions.

3.5 Object classes

Based on their appearance in previous frames, each detected object's state can be classified into one of the three classes:

3.5.1 Newly detected objects

If the object can not be matched against any known object, in order to filter possible false alarms, the newly detected objects will be put into pool of possible positive detections for further evaluation.

3.5.2 Candidates for recognized objects

In the pool of possible recognized objects, each object has its own time to live property. With processing next frames in the sequence, this property decreases. After this property reaches zero, the object is treated as false alarm and is destroyed.

However, each object in the pool also stores how many times was matched against objects detected in consecutive frames. When this value reaches a certain threshold, the object is considered to be verified and is relocated to the pool of recognized object.

3.5.3 Recognized known objects

When the object is passed to this pool, it will be assigned with unique ID. TTL property is raised to higher constant, allowing the object to survive more unsuccessful detection passes. Every time detector matches this object against entity from current frame the TTL is reset to this higher constant.

3.6 Tracker

The object also has a new tracker entity assigned, that will update the object position when the detector is not active or does not score a hit against.

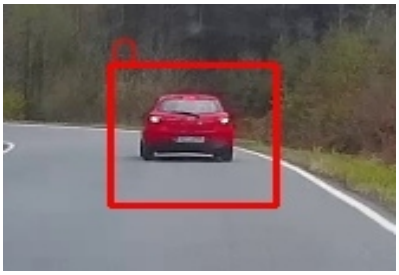


Figure 3.5: Example of KCF tracker drift. The second image is took 480 frames after the first.

Because the trackers have the tendency to drift away from the object in time, the position of the tracker is corrected to position from detector output when hit is scored.

Due to superior performance and quality, the chosen tracker is KCF.

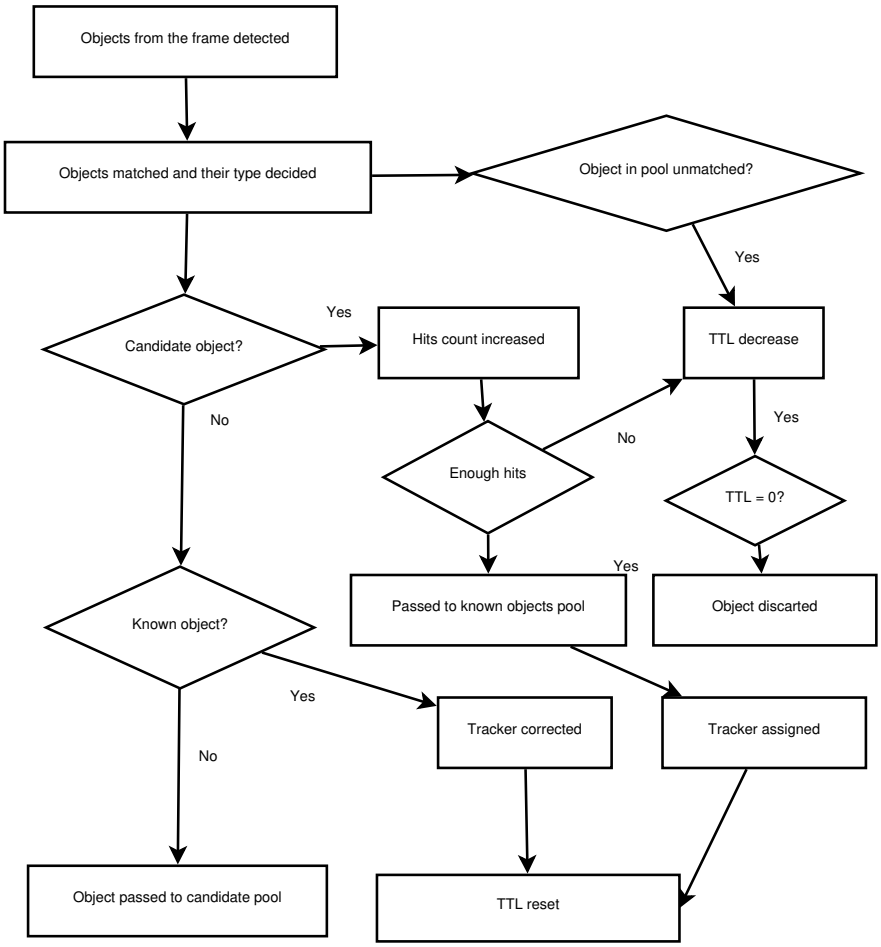


Figure 3.6: Overall schema of the detection phase of the proposal

Chapter 4

Realization

In this chapter, the implementation based on the design proposed in previous chapter will be presented. First the use cases and requirements will be summarized. Then available technologies will be presented and compared. In following section, the learning phase of the detector will be discussed. Structure of the program will be presented and all used classes will be described.

4.1 Use cases

Folowing use cases specify interaction between the user and finished application.

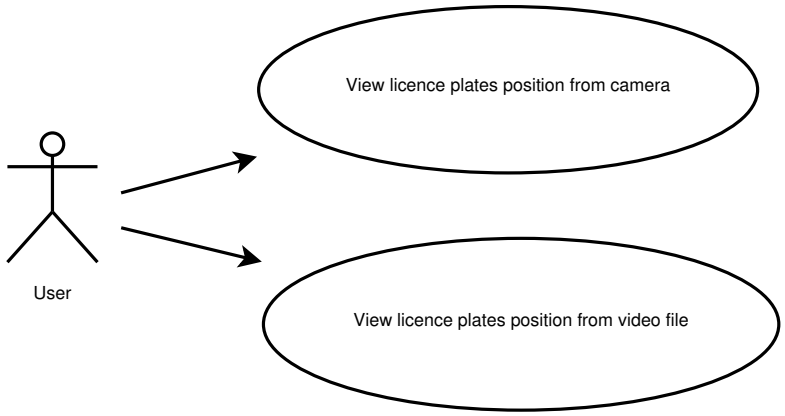


Figure 4.1: Use cases summarization

4.2 Requirements

In this section i would like to summarize both the functional and non-functional requirements. These define how the application should work and specify possible constraints.

4.2.1 Functional requirements

- Detect the positions of license plates.
- Assign the unique ID to detected entities across multiple frames.

4.2.2 Non-functional requirements

- Command line interface
- Visual output
- Detect the license plates positions from video camera stream in real time
- Robustness of detection and tracking

4.3 Used technologies

In this section the available implementation technologies will be summarized and compared.

4.3.1 C++

Available selection of the programming languages was narrowed by the requirement of using OpenCV library to Java, Python, C and C++.

From these languages C does not support object oriented programming paradigm, which perfectly fits the purpose.

Python is interpreted language, thus will generally be slower and less appealing for use in real-time applications.

Java is also interpreted language, but it provides JIT (just in time) compiler to transform performance critical parts of application to native machine code. The great disadvantage is very poor documentation for OpenCV in version for Java.

After all these consideration the final choice went to C++, which is compiled language and has relatively good documentation for the OpenCV library, which is also natively written in C++.

4.3.2 OpenCV library

Is a BSD licensed open source library, containing over 2500 image processing, machine learning and computer vision algorithms.

This software is used by many well-established companies, some examples are Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda and Toyota.

OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

The used version will be OpenCV 3.1, which is latest stable release available.

4.4 Detector learning phase

The OpenCV library provides a command line tool for training of the Haar Cascades, the opencv_traincascade. This tools requires two separate collections of images, serving as positive and negative examples.

Application opencv_createsamples then generates a large number of positive samples from our positive images, by applying transformations and distortions.

As a positive samples 250 random image from this dataset were used. 500 random images from my phone served as a negative samples.

However, the trained cascade did not perform well, so we decided to use OpenCV's supplied cascade for detection of cars instead (**cars.xml**).

It should be noted that the training of the detector took about 14 hours, thus it would be very time consuming to train the classifier on the whole dataset.

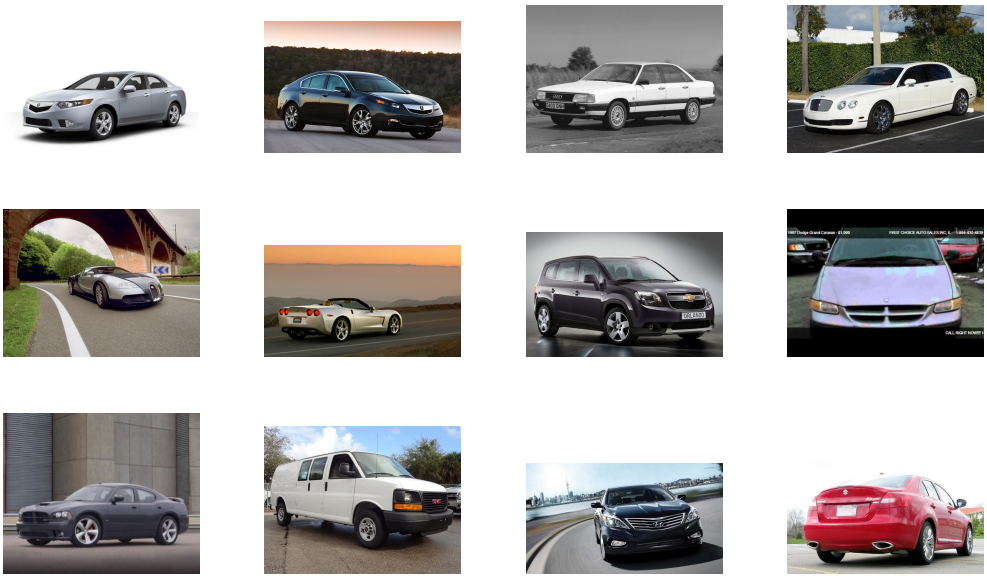


Figure 4.2: Examples some of positive samples from the dataset

4.5 Class Car

This class represents one car and is responsible for the tracking phase of the underlying object.

4.5.1 Constants

- **HITS**
  The constant defines how many successful detections must the car pass be moved into pool of recognized objects.

- **TTL**
  This value defines how many detection cycles the object survives when in pool of possible objects.

- **KNOWN_OBJECT_TTL**
  This value defines how many detection cycles the object survives when in pool of recognized objects.

- **INTERSECTION_UNION_RATIO**
  Defines the minimal overlay of the objects between frames to be considered the same object.

4.5.2 Methods

- **Car** (constructor)

- **Init**
  Performed when the objects passes the continuous detection phase. This method initializes and assigns the instance of KCF tracker to the car. Also gives unique ID to the car.

- **Correct**
  Effectively reattaches tracker to the new position estimated by output from the detector, thus avoiding tracker drift.

- **Update**
  Notifies the tracker to track the position of the car on the current frame.

- **Hit**
  Notifies the class that the detector scored a hit against contained object.

- **UpdateTTL**
  Decreases the Time To Live property. When TTL reaches zero, the tracker is detached and whole instance of object is destroyed.

- **ResetTTL**
  Resets the Time To Live property to constant KNOWN_OBJECT_TTL.

- **Match**
  Decides whether a specific region in the frame contains actual car.

4.5.3 Members

- **m_IdPool**
  Static variable, the unique IDs of all known objects are generating by incrementing this variable.

- **m_Id**
  Unique ID of the object.

- **m_Tracker**
  Pointer to the attached KCF tracker object. KFC tracker is implemented in OpenCv library tracking API.

- **m_Alive**
  Indicates validity of the object. Negative value means either tracker failed, the object did not passed the initial detection stage or TTL has expired.

- **m_Rect**
  This variable contains actual bounding of the object.

- **m_Hits**
  Value indicates how many matches from the detector hits the object needs to be evaluated as recognized object. Its decreased every time detector scores a hit agains this object.

- **m_TTL**
  This value indicates how many unsuccessful detections the object endures before is destroyed. If the object passed the evaluation phase, this value is continuously updated when the object is detected by detector.

4.6 Class CarTracker

This class represents the whole detection and tracking framework.

4.6.1 Constants

- **CASCADE_FILE**
  The path to the XML file containing trained Haar Cascade, that will be used by detector.

- **DETECTION_PERIOD**
  Defines the period of frames in which the detection is performed when the pool of candidate objects is empty.

4.6.2 Methods

- **Process**
  This function is the main routine and completely processes one frame. It performs the detection stage when the frame matches the period defined in constant DETECTION_PERIOD. Also performs the tracking stage of known objects.

- **Track**
  Performs tracking of all members of the known objects pool and displays their bounding boxes. Destroys the objects with tracker failure.

- **Detect**
  This function first converts the frame into grayscale image and performs histogram equalization.

  Then tries to detect possible cars in the frame. All detector hits are then matched against both possible and recognized objects, their status being updated. The possible objects that accumulated enough hits are moved to pool of known objects and their trackers are initialized.

  Eventual new objects are then inserted into possible objects pool. Status of members of both pools is updated and outlived objects are destroyed.

4.6.3 Members

- **m_FrameNr**
  This variable is frame counter and is incremented with each Process function call.

- **m_Cars**
  This linked list serves as a pool for successfully recognized cars.

- **m_Possible**
  This linked list serves as a pool for the candidates for recognized cars. Every object with successful hit from the detector underwents its evaluation phase there, after which is either discarded or moved to m_Cars.

- **m_Classifier**
  This is a reference to Haar classifier, using the cascade specified in constant CASCADE_FILE. This classifier is used to perform the detection phase and is implemented in the OpenCV's object detection library.

4.7 Class QuadTree

This class realizes the quad tree structure for matching phase of the algorithm.

4.7.1 Constants

- **SPLIT**
  Defines the capacity of one leaf. If the number of objects in the leaf exceeds this constant, the leaf splits (if not in maximum depth).

- **MAX_DEPTH**
  Defines maximum depth of the tree.

4.7.2 Methods

- **QuadTree (constructor)**
  Constructs empty node of quadtree.
- **Fits**
  Checks if object fits into the current node.
- **Split**
  Splits the leaf and creates its children.
- **AddToChilds**
  Finds the fitting child of the current node and adds the object into it.
- **Add**
  Adds object to the tree, eventually calls AddToChilds and Split to preserve correct structure of the tree.
- **Match**
  Checks whether any object in the tree matches the input object.

4.7.3 Members

- **m_Bounds**
  Boundary box of the node.
- **m_Childs**
  Pointer to the childs of the node.
- **m_Objects**
  Pointers to the contained objects.
- **m_ObjectsNr**
  Number of the objects stored.
- **m_ObjectsLen**
  m_Objects allocated size.

Chapter 5

Testing

During the development, every finished component underwent extensive unit testing. These tests are summarized in this chapter

5.1 Class Car

- **Tracker failure test**
  The underlying tracker was supplied with random sequences of images and regions of interest outside of the image borders. The class was tested to deal with such issues.

- **Tracker correction test**
  Randomly shifted targets were preserved to the tracker. The class was tested to deal with this condition with and without the detector help.

- **Matching test**
  Extensive matching using the Quad Tree structure was tested.

5.2 Class CarTracker

- **Detector test**
  The performance of the detector was tested with different preprocessing, for example with added gaussian noise.

- **Idle/busy mode test**
  Random regions of the image were inserted into pool of possible objects and the detector behavior according to the setting of constant DETECTION_PERIOD was checked.

5.3 Class QuadTree

- **Addition/Removal test**
  Multiple objects were inserted into the tree and the correctness of the structure was validated.

Chapter 6

Experimental results

In this section, i would like to present experimental results of my implementation of proposed algorithm. For these purposes, i have recorded several different scenes under diverse conditions.

6.1 Metrics

In order to evaluate the results, following metrics will be introduced:

- **Car count**
  Number of distinguishable cars in the video. Percentage is always 1.0.
- **True positives**
  Count of successfully classified cars. Percentage is computed as Car Count divided by True positives.
- **False positives**
  Number of false alarms. Percentage is computed as False positives divided by True positives.
- **Multiply classified**
  Count of multiply classified (relabeled) cars. Percentage is computed relatively to True positives.

6.2 Program settings

Program constants were set as follows:

- Car :: HITS = 5;
- Car :: TTL = 10;
- Car :: KNOWN_OBJECT_TTL = 30;

- Car :: INTERSECT_UNION_RATIO = 0.1;
- CarTracker :: CASCADE_FILE = ”cars.xml”;
- CarTracker :: DETECTION_PERIOD = 10;

The Haar cascade file used was the supplied OpenCV cascade classifier for cars detection (**cars.xml**).

6.3 Highway

This scene is placed on the highway, in good weather conditions and broad daylight. Because the road is straight, most of the cars are facing the camera with their backs.

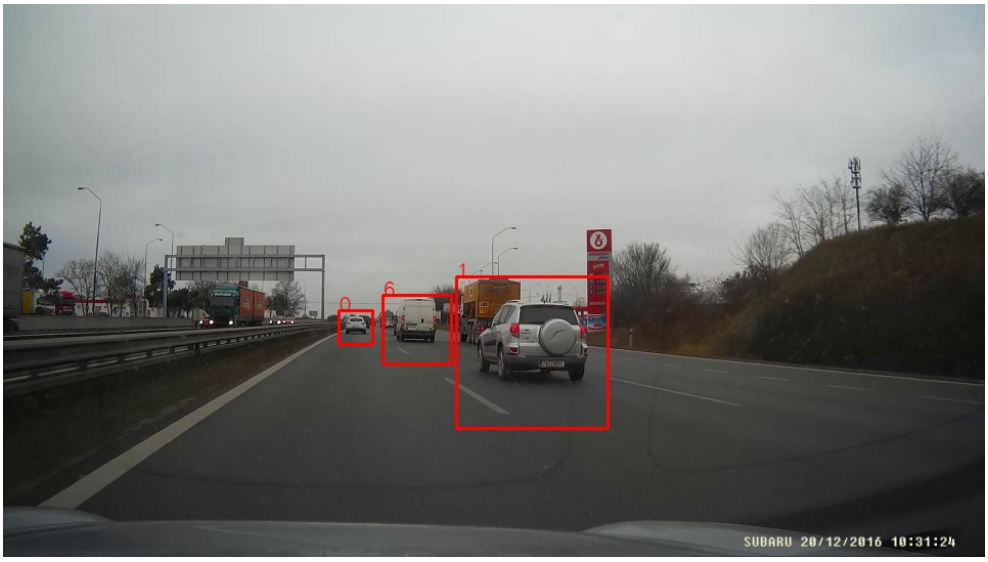


Figure 6.1: The highway

The detection of cars in opposite direction is irrelevant, because they are not entirely visible, due to barriers separating traffic lanes.

Also trucks, although mostly correctly detected are not taken into account.

6.3.1 Results

metric	**count**	**percentage**
cars in the video	5	1.0
true positives	5	1.0
false positives	1	0.16
multiply classified	0	0.0

6.4 One target

In this video, for most of the time, the recording car is following another car on entirely empty route. One parking lot with several other cars is then approached. The video ends with approach of the nearby town, where several cars are met in opposite direction.

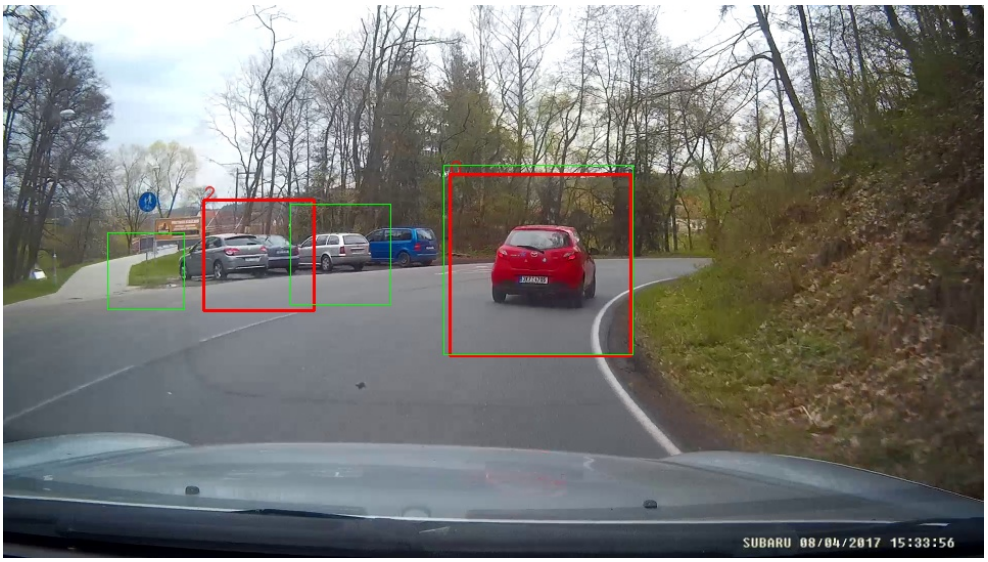


Figure 6.2: Chasing one target

Cars at the pump station are not counted, because they are not in the center of the scene.

6.4.1 Results

metric	**count**	**percentage**
cars in the video	10	1.0
true positives	8	0.8
false positives	3	0.27
multiply classified	1	0.1

6.5 Tunnel

In this scene, the recording car moves in a tunnel, so the scene is relatively dark, with artificial lighting. There are clusters of cars in the same direction. In the evaluation of the results only cars whose contours are distinguishable by human eye count.

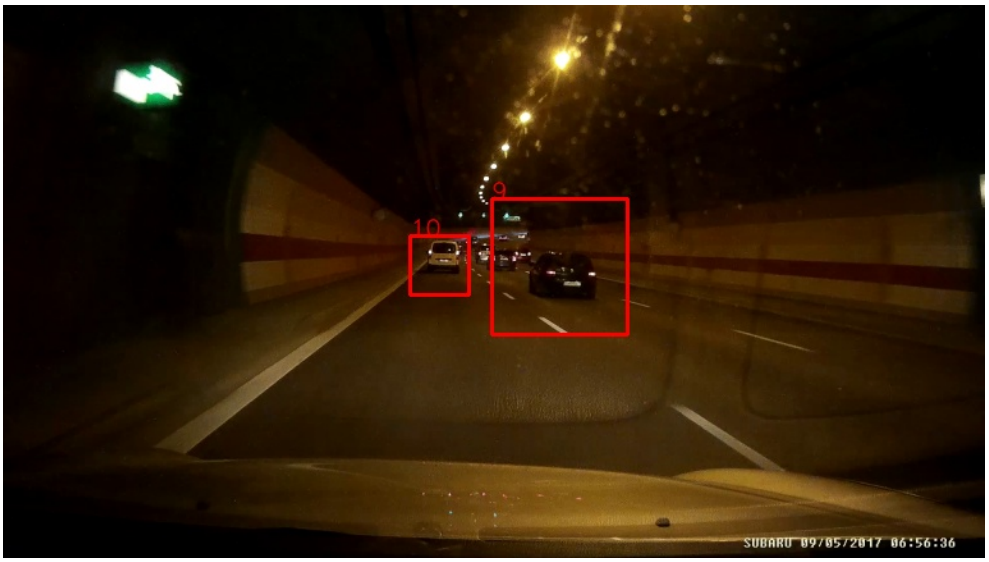


Figure 6.3: The tunnel

6.5.1 Results

metric	count	percentage
cars in the video	7	1.0
true positives	5	0.71
false positives	3	0.3
multiply classified	2	0.28

6.6 Night chase

This scene is filmed in very bad lighting conditions, rendering the detection unreliable. The detector was almost unable to detect cars in opposite direction, so there is a major weakness in the proposal.

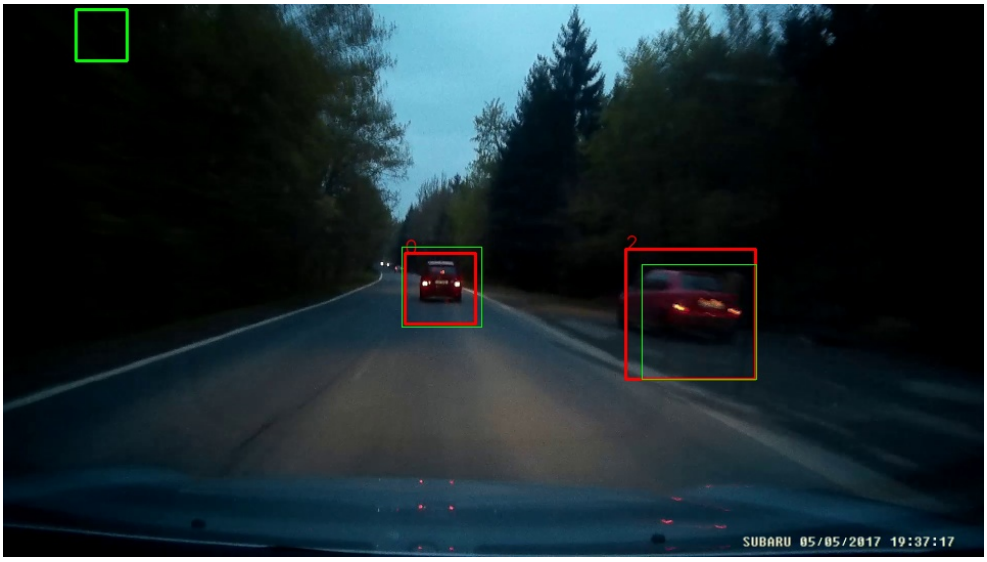


Figure 6.4: Very bad lighting conditions

6.6.1 Results

metric	count	percentage
cars in the video	11	1.0
true positives	5	0.45
false positives	5	1.0
multiply classified	2	0.18

Conclusion

This thesis introduced some of the current approaches to the problem of license plates detection from camera of moving car. Several state-of-the-art techniques of tracking and detection are introduced and detection framework solving the problem is proposed. This framework was successfully implement, evaluated and proof of concept was verified.

Future research

However, there are several imperfections that can be subject to future research, for example more robust matching of overlaying objects or better optimization for scenes with many objects.

Also, as implied by the testing phase, some better preprocessing should be proposed, in order to improve objects detection in bad lighting conditions.

Training of the Haar cascade classifiers is another object of possible future research.

Bibliography

[1] Shiffman, D. HSB Color Space. 2008. Available from: https://processing.org/tutorials/color/

[2] Mula, W. Bitmap and its compressed quadtree representation. 2008. Available from: https://en.wikipedia.org/wiki/Quadtree#/media/File:Quad_tree_bitmap.svg

[3] Wikipedia. k-d tree decomposition for the point set (2,3), (5,4), (9,6), (4,7), (8,1), (7,2). 2006. Available from: https://en.wikipedia.org/wiki/K-d_tree#/media/File:Kdtree_2d.svg

[4] Guel, M. Y. The resulting k-d tree. 2008. Available from: https://en.wikipedia.org/wiki/K-d_tree#/media/File:Kdtree_2d.svg

[5] Berggren, K.; Gregersson, P. Haar Wavelets. 2008. Available from: http://fileadmin.cs.lth.se/graphics/theses/projects/facerecognition/

[6] Vargas, J. HSL, HSB and HSV color: differences and conversion. oct 2013. Available from: http://codeitdown.com/hsl-hsb-hsv-color/

[7] Rouse, M. Definition - hue, saturation, and brightness. 2014. Available from: http://whatis.techtarget.com/definition/hue-saturation-and-brightness

[8] Wright, C. YUV Colorspace. 2004. Available from: http://softpixel.com/~cwright/programming/colorspace/yuv/

[9] Alper Yilmaz, M. S. Object Tracking: A Survey. dec 2006. Available from: https://www.ppgia.pucpr.br/~alceu/pdi/Video%20Segmentation%20and%20Tracking/Yilmaz.pdf

[10] Yali Amit, P. F. Object Detection. Available from: https://cs.brown.edu/~pff/papers/detection.pdf

[11] Helland, T. Seven grayscale conversion algorithms (with pseudocode and VB6 source code). oct 2011. Available from: http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/

[12] Yarlagadd, R. Convolution and Correlation. *Analog and Digital Signals and System*, 2010.

[13] Powell, V. Image Kernels Explained Visually. 2015. Available from: http://setosa.io/ev/image-kernels/

[14] Demut, R. Common characteristics of classification and regression. 2010. Available from: https://edux.fit.cvut.cz/courses/MI-ADM/_media/lectures/02/skripta1-v2.pdf

[15] Johnson, N. Damn Cool Algorithms: Spatial indexing with Quadtrees and Hilbert Curves. Available from: http://blog.notdot.net/2009/11/Damn-Cool-Algorithms-Spatial-indexing-with-Quadtrees-and-Hilbert-Curves

[16] Sheetal Rani, P. K. D. A Review of Recognition Technique u sed Automatic License Plate Recognition System. 2015. Available from: http://research.ijcaonline.org/volume121/number17/pxc3904938.pdf

[17] Wenjing Jia, X. H., Huaifeng Zhang. Region-based license plate detection. nov 2007. Available from: http://www.sciencedirect.com/science/article/pii/S1084804506000762

[18] Muhammad H Dashtban, H. B., Zahra Dashtban. A Novel Approach for Vehicle License Plate Localization and Recognition. jul 2011. Available from: http://www.ijcaonline.org/volume26/number11/pxc3874382.pdf

[19] Tejendra Panchal, A. P., Hetal Patel. License Plate Detection Using Harris Corner and Character Segmentation by Integrated Approach from an Image. 2016. Available from: http://www.sciencedirect.com/science/article/pii/S187705091600185X

[20] Bulugu, I. Algorithm for License Plate Localization and Recognition for Tanzania Car Plate Numbers. 2013. Available from: https://www.ijsr.net/archive/v2i5/IJSRON2013933.pdf

[21] Tarabek, P. A real-time license plate localization method based on vertical edge analysis. 2012. Available from: https://fedcsis.org/proceedings/2012/pliks/354.pdf

[22] Wang, G.; Xiao, D.; et al. Review on vehicle detection based on video for traffic surveillance. sep 2008. Available from: http://ieeexplore.ieee.org/document/4636684/

[23] Cao, Y.; Lei, Z.; et al. A Vehicle Detection Algorithm Based on Compressive Sensing and Background Subtraction. 2012. Available from: http://www.sciencedirect.com/science/article/pii/S2212671612000765

[24] Kim, D.-S.; Kwon, J. Moving Object Detection on a Vehicle Mounted Back-Up Camera. dec 2015. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4732056/

[25] Caraffi, C.; Vojíř, T.; et al. *A System for Real-time Detection and Tracking of Vehicles from a Single Car-mounted Camera*. Master's thesis, Czech Technical University in Prague Faculty of Electrical Engineering, 2012.

[26] de Souza, C. Haar-feature Object Detection in C#. dec 2014. Available from: https://www.codeproject.com/articles/441226/haar-feature-object-detection-in-csharp

[27] Messom, C.; Barczak, A. Fast and Efficient Rotated Haar-like Features using Rotated Integral Images.

[28] Kelly, M. Computer Vision – The Integral Image. sep 2010. Available from: https://computersciencesource.wordpress.com/2010/09/03/computer-vision-the-integral-image/

[29] Brownlee, J. Boosting and AdaBoost for Machine Learning. apr 2016. Available from: http://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/

[30] McCormick, C. HOG Person Detector Tutorial. may 2013. Available from: http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/

[31] KOWALCZYK, A. SVMs - An overview of Support Vector Machine. 2017. Available from: https://www.svm-tutorial.com/2017/02/svms-overview-support-vector-machines/

[32] Rojas, P. D. R. Lucas-Kanade in a Nutshell.

[33] Marshall, D. Optical Flow Constraint Equation. 1997. Available from: http://users.cs.cf.ac.uk/Dave.Marshall/Vision_lecture/node47.html

[34] Zdenek Kalal, J. M., Krystian Mikolajczyk. Tracking-Learning-Detection. jan 2010. Available from: http://kahlan.eps.surrey.ac.uk/featurespace/tld/Publications/2011_tpami

Appendix A

Acronyms

CV Computer vision

OCR Optical character recognition

GUI Graphical user interface

XML Extensible markup language

NN Nearest Neigbor

SVM Support Vector Machine

KCF Kernelized Correlation Filters

FPS Frames per second

FOV Field of view

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `exe` ..... the directory with executables
- `src` ..... the directory of source codes
  - `wbdcm` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.ps` ..... the thesis text in PS format