# DETECTION OF TRIANGULAR AND RECTANGULAR OBJECTS IN DIGITAL IMAGES

Selcan Kaplan Berkaya

Master of Science Thesis

Computer Engineering Program

June, 2014

**JÜRİ VE ENSTİTÜ ONAYI**

**Selcan Kaplan Berkaya**'nın **"Detection of Triangular and Rectangular Objects in Digital Images"** başlıklı **Bilgisayar Mühendisliği** Anabilim Dalındaki, Yüksek Lisans Tezi 23.06.2014 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

| | | Adı Soyadı | İmza |
|---|---|---|---|
| **Üye (Tez Danışmanı)** | **:** | **Doç. Dr. Serkan GÜNAL** | **…………..** |
| **Üye** | **:** | **Doç. Dr. Cüneyt AKINLAR** | **…………..** |
| **Üye** | **:** | **Yard. Doç. Dr. Semih ERGİN** | **…………..** |

**Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun …………….. tarih ve …………….. sayılı kararıyla onaylanmıştır.**

**Enstitü Müdürü**

## ABSTRACT

**Master of Science Thesis**

**DETECTION OF TRIANGULAR AND RECTANGULAR OBJECTS IN DIGITAL IMAGES**

**Selcan KAPLAN BERKAYA**

**Anadolu University**

**Graduate School of Sciences**

**Computer Engineering Program**

**Supervisor: Assoc. Prof. Dr. Serkan GUNAL**

**2014, 49 pages**

In this dissertation, novel methods are proposed for the detection of rotated triangular and rectangular objects in digital images. The proposed methods utilize recently developed and successful edge detection algorithm, and consist of detection and validation stages. In the detection stage, the proposed methods use line segments and construct triangular and rectangular shapes from those segments. The line segments detected by using edge detection algorithm are converted into line pairs according to their angles and distance between each two lines. The candidate line pairs are first combined with each other. If the triangular or rectangular shapes are not constructed, for triangular shapes these line pairs are combined with a single line segment, for rectangular shapes two line pairs combined, and then these pairs are combined with a single line segment by following the appropriate criteria. Finally, in the validation stage, the candidate triangles and rectangles are validated using Helmholtz principle and Number of False Alarms (NFA) computation. According to the results of the experimental studies, the proposed methods offer higher detection performances than Open Source Computer Vision (OpenCV) triangle and rectangle detection algorithms which are commonly used in computer vision field.

**Keywords:** Geometrical Shape Detection, Triangular Object Detection, Rectangular Object Detection, Shape Analysis, Hough Transform

# ÖZET

**Yüksek Lisans Tezi**

**SAYISAL İMGELERDE ÜÇGENSEL VE DİKDÖRTGENSEL NESNELERİN BELİRLENMESİ**

**Selcan KAPLAN BERKAYA**

**Anadolu Üniversitesi**
**Fen Bilimleri Enstitüsü**
**Bilgisayar Mühendisliği Anabilim Dalı**
**Danışman: Doç. Dr. Serkan GÜNAL**

**2014, 49 sayfa**

Bu tez çalışmasında, sayısal imgelerdeki döndürülmüş üçgensel ve dikdörtgensel nesnelerin tespiti için yeni yöntemler önerilmiştir. Önerilen yöntemler, yakın zamanda geliştirilmiş olan başarılı bir kenar tespit algoritmasından faydalanmakta olup, tespit ve doğrulama olmak üzere iki aşamadan oluşmaktadır. Tespit aşamasında, önerilen yöntemler çizgi parçalarını kullanarak üçgensel ve dikdörtgensel şekillerin tespitini yapmaktadır. Kenar tespit algoritmasıyla tespit edilen çizgi parçaları, açıları ve aralarındaki uzaklığa göre ikili çizgi parçalarına dönüştürülmüştür. Aday ikili çizgi parçaları önce kendi aralarında birleştirilmiştir. Eğer üçgen veya dikdörtgen oluşmamışsa, üçgensel şekiller için tek bir çizgi parçasıyla, dikdörtgensel şekiller için bu iki ikili çizgi birleştirildikten sonra tek bir çizgi parçasıyla karşılaştırılarak uygun ölçütlerde birleştirilmiştir. Son olarak, doğrulama aşamasında, Helmholtz prensibi ve Yanlış Alarm Sayısı (YAS) hesaplaması kullanılarak onaylanmıştır. Deneysel çalışmaların sonucuna göre, önerilen yöntemler, bilgisayarla görü alanında sıkça kullanılan Açık Kaynaklı Bilgisayarla Görü (OpenCV) üçgen ve dikdörtgen tespit algoritmalarına göre daha yüksek tespit başarımı sağlamaktadır.

**Anahtar Kelimeler:** Geometrik Şekil Tespiti, Üçgensel Nesne Tespiti, Dikdörtgensel Nesne Tespiti, Şekil Analizi, Hough Dönüşümü

## ACKNOWLEDGEMENTS

I would like to thank to my advisor Assoc. Prof. Dr. Serkan GUNAL and Assoc. Prof. Dr. Cuneyt AKINLAR for their guidence and support during my study. It was my pleasure to work with them during this study. Also, I would like to thank to Scientific and Technical Research Council of Turkey (TUBITAK) for giving me financial support through its scholarship program during this study. Finally, I would like to express my eternal gratitude to my husband and my parents for their everlasting love and support.

Selcan Kaplan Berkaya

June, 2014

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

## ABBREVIATIONS

| | | |
|---|---|---|
| $A_{min}$ and $A_{max}$ | : | Angle tolerance |
| $D_{max}$ | : | Distance tolerance |
| DT | : | Distance Transform |
| ED | : | Edge Drawing |
| EDPF | : | Edge Drawing Parameter Free |
| FN | : | False Negative |
| FP | : | False Positive |
| GHT | : | Generalized Hough Transform |
| HSI | : | Hue Saturation Intensity |
| HT | : | Hough Transform |
| $L_{max}$ | : | Length tolerance |
| MRF | : | Markov Random Field |
| ms | : | Milliseconds |
| NFA | : | Number of False Alarms |
| Non-HT | : | Non-Hough Transform |
| OpenCV | : | Open Source Computer Vision |
| PHT | : | Probabilistic Hough Transform |
| RGB | : | Red Green Blue |
| $R_{min}$ and $R_{max}$ | : | Ratio tolerance |
| SHT | : | Standard Hough Transform |
| TP | : | True Positive |

# 1. INTRODUCTION

Detection of polygonal objects in digital images is an important problem in image processing and computer vision, and has many applications especially in such building detection for geographical information systems, construction planning, environmental investigation (Park et al., 2009), vehicle detection (Moon et al., 2002), license plates recognition (Gao and Zhou, 2000; Xu et al., 2004; Llorens et al., 2005), triangular and rectangular traffic signs detection (delaEscalera et al., 1997; Cyganek, 2007; Maldonado-Bascon et al., 2007; Xu, 2009; Ruta et al., 2010; Khan et al., 2011; Bruno et al., 2012; Zaklouta and Stanciulescu, 2014), and many others. The most common geometrical shapes are triangle and rectangle. While triangular and rectangular objects can be observed in ordinary images, they can be also encountered in particular images such as road scenes including traffic signs. So far, it has made many rectangular objects detection methods. Detection of triangular objects is mainly used in traffic sign recognition and there are many studies in this field. However, there are a few papers about diverse kinds of triangular objects detection from color images.

The common methods for shape detection are based on HT and its improved version (Zhu and Quingzhi, 2011). Although these methods may offer satisfactory performance, their main disadvantages are high computational complexity and large storage requirement. These methods are mostly used in circle and line detection problems (Liu et al., 2006, Bradski and Kaehler, 2008).

In the HT based methods (described in detail in Section 1.2), most of the methods that used edge and line primitives to form shapes like rectangles or triangles use Canny edge detector for finding contours, and then apply Hough line transform for finding straight lines. In (He and Ma, 2009) and (Jung and Schramm, 2004), techniques based on windowed HT are used for image transform and triangle and rectangle detection, respectively.

The other methods are based on linear combination of methods (Non-HT). Some samples of them are information of corners, the property of inscribed circle, color information and template matching, etc. Non-HT methods based on edge and line pirimitives were provided to avoid the defects of the HT method. These methods of detection efficiency and detection accuracy depend on the number of the lines obtained from the line detection algorithm (Gunduz et al., 2013).

In (Liu and Wang, 2014), the approach based on the property of the inscribed circle of triangles is proposed for triangle detection. In (Garlipp and Muller, 2006), the authors establish a model with a regression function after detecting the edges. Then, they detect the triangular and rectangular objects if the model meets the triangularity or rectangularity conditions. (Liu et al., 2006a; Liu et al., 2006b) presented Markov Random Field (MRF) model-based algorithms for triangular and rectangular shaped objects detection in color images, respectively. In (Barnes et al., 2010), they proposed a regular polygon detector using a posteriori probability approach based on locality and gradient information. However, the proposed technique is capable of detecting only equilateral triangles and rectangles.

In the methods used information of corners, firstly corners must be found. A corner can be defined as the intersection of two or more edges. Polygons can detect using localization of corners in image. Distance Transform (DT) is one of these. In this method, firstly the corner points are found in the image (Maldonado-Bascon et al., 2007; Moomivand and Abolfazli, 2011). DT image is obtained using distance of each pixel to the nearest corner of the image. This method takes a long time to create the feature vector. Therefore, it is not suitable for real-time applications.

Color based detection methods aim to segment the given color in order to provide a region of interest for further processing. Color information is easy to be influenced by illumination changes and different weathers. With color based methods, researchers choose different color spaces and thresholds. Escalera et al. used the normalized Red Green Blue (RGB) space with fixed threshold in which red component was chosen as reference (delaEscalera et al., 1997), and (Yalic and Can, 2011; Xu et al., 2012) used RGB, too. International Commission on Illumination, L* stands for luminance, a* is the red-green axis, and b* is the blue-yellow axis (CIELab or CIE L*a*b*) is used in other work (Khan et al., 2011) to represent the color image because this space can independently control color and intensity information. Most researches (Xu, 2008, Zhu et al., 2006, Ruta et al., 2010, Maldonado et al., 2007) used Hue Saturation Intensity (HSI) space which is regarded as more immune to lighting changes. In (Garcia-Gorrido et al., 2006), comparing the methods based on colour segmentation with the ones based on shape analysis it can be concluded that color provides a faster focusing on the seeking areas but in practice precision is lower. In conclusion, the methods based on shape analysis are more robust against changes in lighting conditions.

Defining a measure or a cost measuring the “distance” or the “similarity” between the (known) reference patterns and the (unknown) test pattern, in order to perform the matching operation known as template matching (described in detail in Section 1.1). Matching based algorithms are standard methods for polygons detection; these start by recovering line segments, and construct shapes from these. This is inherently slow as the possible match candidates grow exponentially with the number of edges. It is also non-robust, performing poorly under partial occlusions.

Some of major algorithms that commonly used for shape detection are mentioned in continued Section 1.1 and Section 1.2.

## 1.1. Template Matching (OpenCV Template Matching)

Template Matching is a method for searching and finding the location of a template image in a larger (source) image. Template matching via cvMatchTemplate() matches an actual image patch against an input image by “sliding” the patch over the input image using one of the matching methods described in this section (Bradski and Kaehler, 2008). If, as in Figure 1.1, there is an image patch containing a triangular traffic sign, then that sign can be slid over an input image looking for strong matches that would indicate another sign is present. The function call is given below:

```
void cvMatchTemplate(
const CvArr* image,
const CvArr* templ,
CvArr* result,
int method );
```

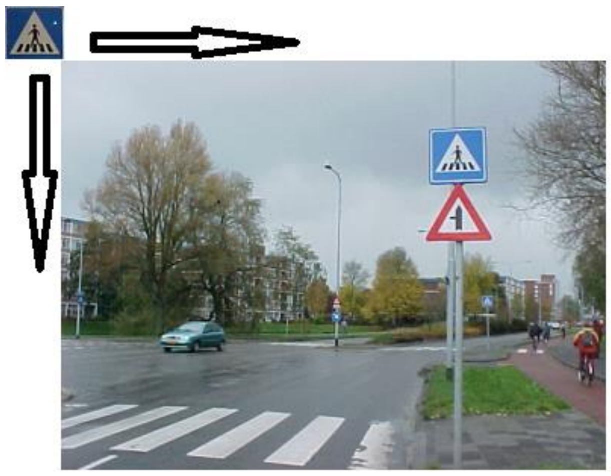


**Figure 1.1.** cvMatchTemplate() sweeps a template image patch across another image

The matching model in templ is just a patch from a similar image containing the object for which you are searching. The input image is used as I, T the template, and R the result.

### 1.1.1. Square difference matching method (method = CV_TM_SQDIFF)

These methods match the squared difference, so a perfect match will be 0 and bad matches will be large:

$$R_{sq_{diff}}(x,y) = \sum_{x'y'}[T(x',y') - I(x+x',y+y')]^2 \qquad (1.1)$$

### 1.1.2. Correlation matching methods (method = CV_TM_CCORR)

These methods multiplicatively match the template against the image, so a perfect match will be large and bad matches will be small or 0.

$$R_{ccorr}(x,y) = \sum_{x'y'}[T(x',y').I(x+x',y+y')]^2 \qquad (1.2)$$

### 1.1.3. Correlation coefficient matching methods (method=CV_TM_CCOEFF)

These methods match a template relative to its mean against the image relative to its mean, so a perfect match will be 1 and a perfect mismatch will be –1; a value of 0 simply means that there is no correlation (random alignments).

$$R_{ccoeff}(x,y) = \sum_{x'y'}[T(x',y').I(x+x',y+y')]^2 \tag{1.3}$$

$$T(x',y') = T(x',y') - \frac{1}{(w.h)\sum_{x''y''}T(x'',y'')} \tag{1.4}$$

$$I(x+x',y+y') = I(x+x',y+y') - \frac{1}{(w.h)\sum_{x''y''}I(x+x'',y+y'')} \tag{1.5}$$

### 1.1.4. Normalized methods

The normalized methods are useful because they can help reduce the effects of lighting differences between the template and the image. In each case, the normalization coefficient is the same:

$$Z(x,y) = \sqrt{\sum_{x'y'}T(x',y')^2.\sum_{x'y'}I(x+x',y+y')^2} \tag{1.6}$$

The values for methods that give the normalized computations are listed in Table 1.1. Figure 1.2 shows the results of sweeping the sign template over the source image (shown in Figure 1.1) using each of cvMatchTemplate()’s available matching methods. In the first column, the darkest is the better match, for the other two columns, the brighter a location, the higher the match.

**Table 1.1.** Normalized computations

| Value of method parameter | Computed result |
|---|---|
| **CV_TM_SQDIFF_NORMED** | $R_{sq\_diff\_normed}(x,y) = \frac{R_{sq\_diff}(x,y)}{Z(x,y)}$ |
| **CV_TM_CCORR_NORMED** | $R_{ccorr\_normed}(x,y) = \frac{R_{ccorr}(x,y)}{Z(x,y)}$ |
| **CV_TM_CCOEFF_NORMED** | $R_{ccoeff\_normed}(x,y) = \frac{R_{ccoeff}(x,y)}{Z(x,y)}$ |

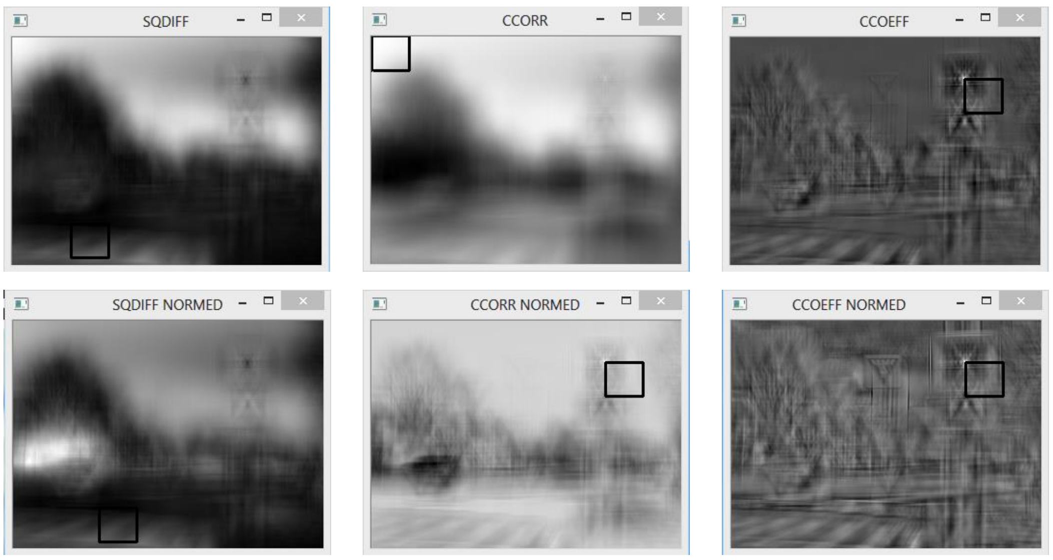


**Figure 1.2.** Match results of six matching methods for the template search depicted in Figure 1.1, first row are the standard methods SQDIFF, CCORR and CCOEFF, second row are the same methods in its normalized version

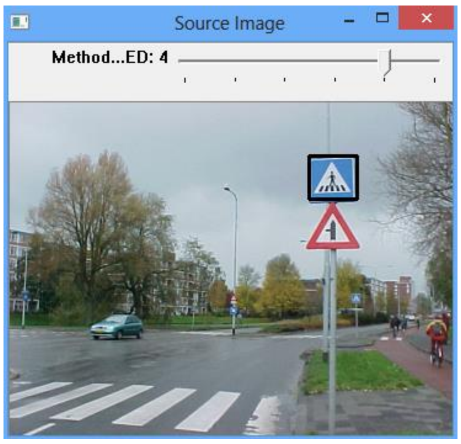


**Figure 1.3.** The right match for a sample image given in Figure 1.1 (CCORR NORMED, CCOEFF, COEFF NORMED give the correct matches)

The right match is shown above (black rectangle around the sign at the top). For this image, CCORR NORMED, CCOEFF and CCOEFF NORMED give the best matches.

### 1.2. Hough Transform (OpenCV Hough Transform)

The HT (Hough, 1962) was proposed by Paul Hough who patented the method in 1962. It transforms between the Cartesian space and a parameter space in which a straight line (or any parameterized curve) can be defined. The original HT was a line transform (Bradski and Kaehler, 2008; Ginkel et al.,2004), which is defined to detect straight lines in binary images, and seemingly inherently discrete. The transform can be further generalized to cases other than just simple lines (to other shapes and grayscale images).

The advantages of the HT:

- The Generalized Hough Transform (GHT) (Ballard, 1981) is essentially a method for object recognition.
- It is robust to partial or slightly deformed shapes (i.e., robust to recognition under occlusion). The pixels lying on one line need not all be contiguous. This can be very useful when trying to detect lines with short breaks in them due to noise, or when objects are partially occluded.
- It is robust to the presence of additional structures in the image (i.e., other lines, curves, etc.).
- It is tolerant to noise. The main advantage of the HT technique is that it is tolerant of gaps in feature boundary descriptions and is relatively unaffected by image noise (Fisher et al., 2000).
- It can find multiple occurences of a shape during the same processing pass.

The disadvantages of the HT:

- It requires a lot of storage and extensive computation (but it is inherently parallelizable!).
- Faster variations have been proposed in the literature.

- It can give misleading results when objects happen to be aligned by chance.
- Detected lines are infinite lines described by their ($a$, $b$) values, rather than finite lines with defined end points.

### 1.2.1. Hough Line Transform

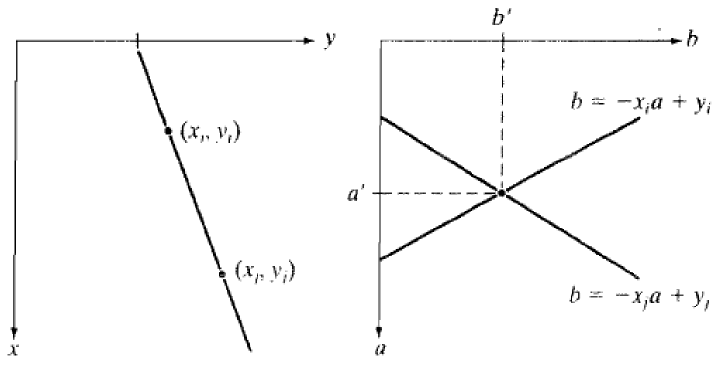
The basic theory of the Hough line transform is that any point in a binary image could be part of some set of possible lines. Given a single isolated edge point ($x_i$, $y_i$), there are an infinite number of lines that could pass through the points and each of these lines can be characterized by some particular equation (varying values of a and b) presented in Equation 1.7 (Gonzalez and Woods, 2008).

$$y_i = ax_i + b \tag{1.7}$$

These lines can be represented as a line in parameter space. Given a set of collinear edge points, each of them have associated a line in parameter space. These lines intersect at the point where $a$ is the slope and $b$ the intercept of the lines corresponding to the parameters of the line in the image space. In fact, all the points on this line (shown in Figure 1.4a) have lines in parameter space that intersect at ($a^{'}$, $b^{'}$) (shown in Figure 1.4b and Figure 1.5b).

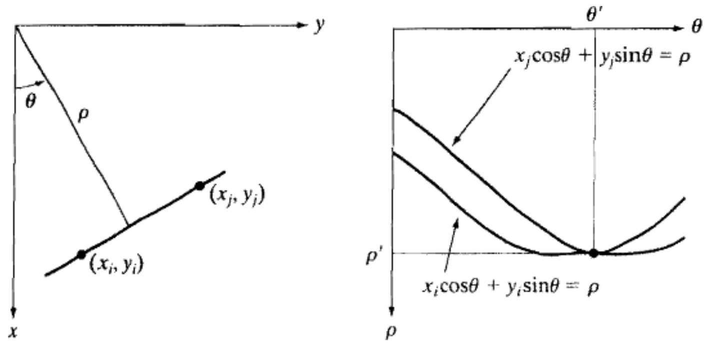
Figure 1.5a shows a line in the Cartesian coordinates (xy-plane). Figure 1.5c shows its HT. The equation of the line is Equation 1.7 and the corresponding equation in Polar coordinates (parameter space) is Equation 1.8:

$$\rho = x \cos\theta + y \sin\theta \tag{1.8}$$

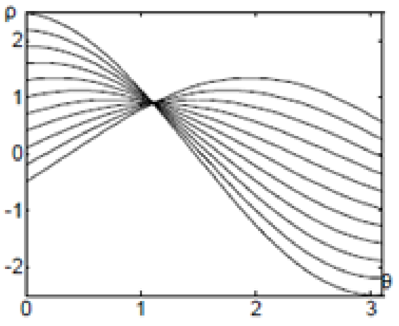


(a) (b)

**Figure 1.4.** A sample line in image and parameter spaces (Gonzalez and Woods, 2008). (a) xy-plane. (b) Parameter space


y
$\theta$
$\rho$
$(x_j, y_j)$
$(x_i, y_i)$
x
$\theta'$
$\theta$
$x_j\cos\theta + y_j\sin\theta = \rho$
$\rho'$
$x_i\cos\theta + y_i\sin\theta = \rho$
$\rho$

(a) (b)



(c)

**Figure 1.5.** A sample straight line and its HT. (a) (p, θ) parameterization of line in the xy-plane (A straight line in the Cartesian coordinates). (b) Sinusodial curves in the pθ-plane, the point of intersection ($p^{'}$, $\theta^{'}$) corresponds to the line passing through points ($x_i$, $y_i$) and ($x_j$, $y_j$) in the xy-plane (Gonzalez and Woods, 2008). (c) HT of the straight line (Figure 1.5a) (He and Ma, 2009)

OpenCV supports two different kinds of Hough line transform: the standard Hough transform (SHT) (Duda and Hart, 1972) and the probabilistic Hough transform (PHT) (Kiryati et al., 1991). Both of these algorithms are tested in this thesis and accessed with the same OpenCV function, though the meanings of some of the arguments depend on which method is being used.

```
CvSeq* cvHoughLines2(
CvArr* image,
void* line_storage,
int method,
double rho,
double theta,
int threshold,
double param1 = 0,
double param2 = 0
);
```

The first argument is the input image. The second argument is a pointer to a place where the results can be stored, which can be either a memory storage or a plain N-by-1 matrix array (the number of rows, N, will serve to limit the maximum number of lines returned). The next argument, method, can be CV_HOUGH_STANDARD, CV_HOUGH_PROBABILISTIC, or CV_HOUGH_MULTI_SCALE for (respectively) SHT, PHT, or a multiscale variant of SHT. The next two arguments, rho and theta, set the resolution desired for the lines (i.e., the resolution of the accumulator plane). The threshold value is the value in the accumulator plane that must be reached for the routine to report a line (Bradski and Kaehler, 2008).

The results of these two functions compared with the proposed algorithms in this thesis are not good enough in terms of accuracy and processing time. Even only to obtain the straight lines with SHT or PHT takes longer than the proposed algorithms.

A sample image and its PHT are illustrated in Figure 1.6. Only PHT is shown, because it is observed that PHT gives better results than SHT.

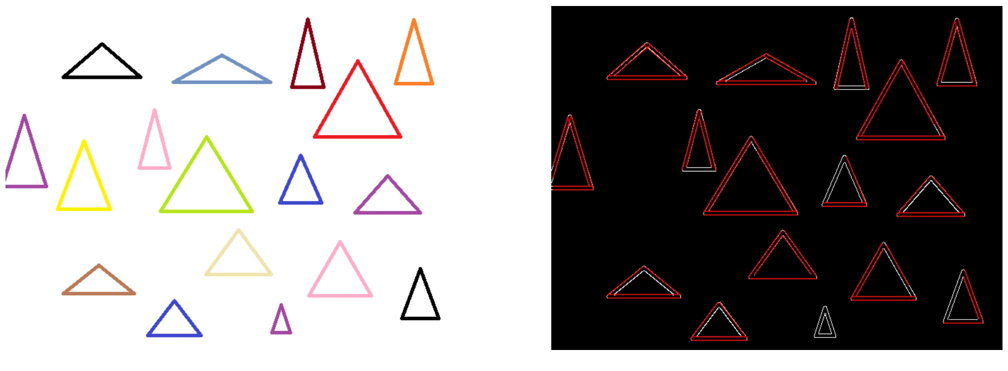

(a) (b)

**Figure 1.6.** A sample synthetic image and its PHT. (a) A sample image. (b) The obtained lines using OpenCV PHT (10.59 ms) (overlayed by red color)

## 1.3. The Proposed Approaches

In this thesis, novel and efficient algorithms are proposed to detect various types of triangular and rectangular shapes under different ligthing conditions. The proposed methods mainly consist of a detection stage and a validation stage. In the detection stage, the algorithm first utilizes the recently developed, real-time edge segment detection algorithm, named Edge Drawing (ED) (Topal and Akinlar, 2012), which, given a grayscale image, produces a set of edge segments each of which is a contiguous pixel chain. Nextly, the extracted edge segments are converted into line segments and these line segments are further processed to detect triangles or rectangles. In case an extracted edge segment is closed, i.e., the first and the last pixel of the edge segment is very close to each other on the image, the edge segment most likely traces the boundary of a polygonal object; thus, triangular and rectangular shapes can easily and quickly be detected from such closed edge segments. If, on the other hand, the edge segment is a non-closed, triangular and rectangular shapes can still be detected in a reasonable time using pairwise relationships of line segments that meet certain criteria. In the validation stage, the candidate shapes are validated using Helmholtz principle (Desolneux et. al., 2004; Desolneux et. al., 2008) which eliminates false detections leaving only perceptually meaningful triangles or rectangles. In order to evaluate the performance of the proposed algorithm, for triangle detection four datasets, which are composed of synthetic and real images containing various types of triangles (e.g. regular, overlapped, partially occluded, discontiguous, etc.), and for rectangles three datasets are employed. The results of the experimental works reveal that both of the proposed methods offer higher detection performances than that of widely used OpenCV triangle or rectangle detection algorithm.

## 1.4. Organization of the Thesis

The organization of this thesis is as follows: Section 2 introduces the proposed triangle detection algorithm in detail. In addition, the parts used in common with the proposed rectangle detection algorithm are provided in this section. Also, experimental work and related results in terms of accuracy and processing time of the proposed triangle detection algorithm are presented in this section.

In Section 3, the proposed rectangle detection algorithm is introduced. Also, in a similar manner to the proposed triangle detection algorithm, the experimental work and related results are presented in this section.

Finally, some concluding remarks and future directions are provided in Section 4.

# 2. THE TRIANGLE DETECTION ALGORITHM

Triangular and rectangular shape detection algorithms work with grayscale images and follow several steps to compute the triangle(s) and rectangle(s) in a given image. General outline of these algorithms is presented in Algorithm 1 and each step of the algorithm is described detailly within the following subsections.

**Algorithm 1:**

1. Detect edge segments by Edge Drawing Parameter Free (EDPF).
2. Convert edge segments into line segments.
3. Find closed edge segments and extract complete triangles/rectangles.
4. Detect line pairs and straight lines by combining remaining line segments.
5. Combine line pairs and straight lines to detect triangle/rectangle candidates.
6. Validate the candidate triangles/rectangles using Helmholtz principle.
7. Output the remaining valid triangles/rectangles.

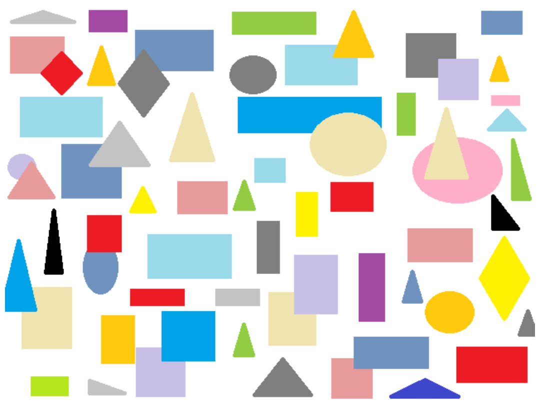
1 and 2 steps are common for both triangle and rectangle detection algorithms.

## 2.1. Edge Segment Detection by EDPF

Edge Drawing (ED) (Topal and Akinlar, 2012), a real-time edge/edge segment detector is proposed recently. ED outputs not only a binary edge map similar to those output by traditional edge detectors, but it also outputs the result as a set of edge segments each of which is a contiguous (connected) pixel chain (Akinlar and Topal, 2013).

ED has many parameters that must be set by the user, which requires the tuning of ED’s parameters for different types of images. Ideally, one would want to have a real-time edge/edge segment detector which runs with a fixed set of internal parameters for all types of images and requires no parameter tuning. To achieve this goal, ED is recently incorporated with the “a contrario” edge validation mechanism due to the Helmholtz principle, and obtained a real-time parameter-free edge segment detector, which named EDPF (Akinlar and Topal, 2012). EDPF works by running ED with all ED’s parameters at their extremes, which detects all possible edge segments in a given image with many false positives. Then the extracted edge segments are validated by the Helmholtz principle, which eliminates false detections leaving only perceptually meaningful edge segments.

Figure 2.1a shows a 640 × 480 synthetic image containing various kinds of colored rectangles and triangles obstructed by ellipses, circles or each other. When this image is fed into EDPF, the edge segments shown in Figure 2.1b are produced. Each color in the edge map represents a different edge segment, each of which is a contiguous chain of pixels. For this image, EDPF outputs 66 edge segments in just 18.68 ms in a PC with a 3.5GHz Intel Core i7-3770K CPU and 16GB RAM.


(a)

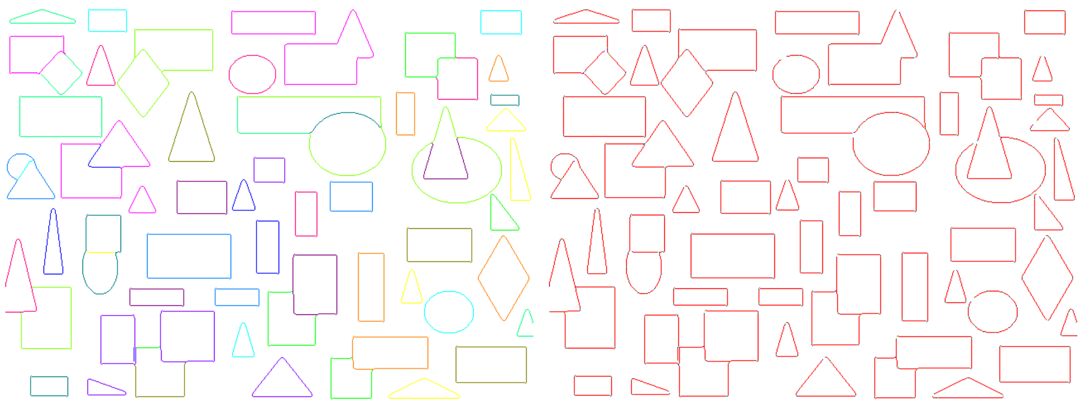

(b) (c)

**Figure 2.1.** A sample synthetic image and results of EDPF (Akinlar and Topal, 2012). (a) A sample image (640 × 480). (b) Edge segments extracted by EDPF (Akinlar and Topal, 2012). EDPF (Akinlar and Topal, 2012) outputs 66 edge segments in 18.68 ms. (c) Lines approximating the edge segments. A total of 310 lines are extracted

## 2.2. Conversion of Edge Segments into Line Segments

Conversion of an edge segment into a set of lines follows the algorithm, EDLines given in (Akinlar and Topal, 2011). The proposed algorithm is to start with a short line that satisfies a certain straightness criterion, and extend the line for as long as the mean square error is smaller than a certain threshold, i.e., 1 pixel error. Here is the algorithm (Akinlar and Topal, 2011) for converting an edge segment into several lines:

**Algorithm 2:**

```
LineFit(Pixel *pixelChain, int noPixels){
 double lineFitError = INFINITY;     // current line fit error
LineEquation lineEquation;           // y = ax+b OR x = ay+b

while (noPixels > MIN_LINE_LENGTH){
   LeastSquaresLineFit(pixelChain, MIN_LINE_LENGTH, &lineEquation, &lineFitError);
   if (lineFitError <= 1.0) break; // OK. An initial line segment detected
   pixelChain ++;  // Skip the first pixel & try with the remaining pixels
   noPixels--;         // One less pixel
} // end-while

if (lineFitError > 1.0) return;  // no initial line segment. Done.

// An initial line segment detected. Try to extend this line segment
int lineLen = MIN_LINE_LENGTH;
while (lineLen < noPixels){
   double d = ComputePointDistance2Line(lineEquation, pixelChain[lineLen]);
   if (d > 1.0) break;
   lineLen++;
} //end-while

// End of the current line segment. Compute the final line equation & output it.
 LeastSquaresLineFit(pixelChain, lineLen, &lineEquation);
 Output "lineEquation"

// Extract line segments from the remaining pixels
LineFit(pixelChain+lineLen, noPixels-lineLen);
}  //end-LineFit
```

Using Algorithm 2, it is very fast to convert an edge segment made up of a chain of pixels into a set of lines (Akinlar and Topal, 2011). Consecutive set of lines are shown in Figure 2.1c. After conversion these line segments are turned into straight lines by checking collinearity. The edges of the corners of the regular- shaped objects are straight line segments (Zhang et al., 2010). Due to effect of noise or illumination variation, line segments may be highly fragmented and grouping process is necessary. Moreover, line segments combination will reduce the time complexity of later rectangle boundary detection process (Liu et al., 2006a, Liu et al., 2006b). Therefore, some straight line segments are merged into a single segment according to several perceptual grouping criteria:

1. The merged line segments must belong to the same edge segment.
2. The distance between line segments should be very short.
3. The angle between line segments should be smaller than five degree to check collinearity.

In this step, straight lines in both closed and non-closed edge segments are obtained according to meet the requirements mentioned above. The remaining steps are described in following sections.

## 2.3. Finding Closed Edge Segments and Extraction of Complete Triangles

The easiest case for triangle detection is when the entire boundary of an object in the image is returned as a closed curve; that is, the edge segment starts at a pixel on the boundary of an object, traces its entire boundary and ends at where it starts. In other words, the first and last pixels of the edge segment are neighbors of each other (Akinlar and Topal, 2013).

If an edge segment forms such a closed pixel chain, a check is performed to see if the segment traces the entire boundary of a triangle. For this purpose, the edge segment is first converted into line segments and a check is performed to see if the number of straight line segments is equal to three.

In the second step, the three interior angles of candidate triangle are computed. Angle tolerance ($A_{min}$ and $A_{max}$), distance tolerance ($D_{max}$) and ratio tolerance ($R_{min}$ and $R_{max}$) are used to verify that those three lines are the sides of the triangle. If the summation of the angles are close to 180 ([180-$\varepsilon$, 180+$\varepsilon$]) and each angle is between $A_{min}$ and $A_{max}$, intersection points of each pair of lines are found.  Thus, the corner coordinates of the candidate triangle are obtained. Then, the distances between these corners are computed. If the ratios of these distances to the actual lengths of straight lines are not between $R_{min}$ and $R_{max}$, it is concluded that these three lines do not form a triangle. Otherwise, these three lines are forwarded to the validation stage. The algorithm for triangle processing of a closed edge segment is provided in Algorithm 3.

**Algorithm 3:**

```
if(the segment of interest is closed) {
    bool triangle=false;
    if(number of lines in the edge segment==3){
    Find the angles between lines;
        if(175<summation of three angles<185 ){
            if(15<the angles<165)){
                Find intersection point of three lines;
                Assign these intersection points to the triangles structure as the
            corners of candidate triangle;
                if(the start and end coordinates of lines are not between corner
                coordinates ){tri=false;}
                else{
                    Compute distances between corners;
                    if( distance between corners)>9){
                if(actual lengths of lines/distances(ratios)<0.35 && ratios >2)
                                { triangle=false; }//end-if
                    else{
                        if(Validation of Triangles==true){
                            numberofTriangles++;
                    triangle=true;
                    }//end-if}//end-else
        }//end-if}//end-else}//end-if}//end-if}//end-if}//end-if}//end-if
```

If a complete triangle cannot be constructed in previous steps, then a sorting operation is processed on available line segments, and the longest three lines are selected. Later on, the selected lines are processed as described in Algorithm 3. If a triangle still cannot be obtained by both of these methods, Algorithm 4 in Section 2.5 is used.

## 2.4. Extraction of Complete Triangles from Nonclosed Edge Segments

As distinct from extraction of complete rectangles, complete triangles are extracted from nonclosed edge segment in a separate stage due to triangle detection algorithm. In this stage, if the edge segment of interest is not closed, the start and end points of the segment are not close enough to each other; candidate triangle cannot be constructed from this segment. In this case, the lines belonging to the segment are sorted according to their lengths. Then, the previous steps in detection of complete triangles from the closed edge segments (Section 2.3) are repeated in the same order.

## 2.5. Line Pairs Detection

In this step, appropriate line pairs are determined. In order to achieve this goal, an algorithm, namely FindPairs (Algorithm 4), is proposed. FindPairs is used when a triangle or a rectangle is not constructed from closed edge segments or the number of lines in the edge segment of interest is smaller than 3 for triangle, 4 for rectangle.

Here, the lines in the edge segment of interest and the number of lines are sent as input parameters to the algorithm. If the lengths of both of the lines compared are longer than $L_{max}$ (because longer line segments give more accurate results than others), the distance between these two lines can be $D_{max1}$ at most.

Otherwise, the distance can be up to $D_{max2}$. If the distance between two lines is smaller than either $D_{max1}$ or $D_{max2}$, and the angle between these two lines is between $A_{min}$ and $A_{max}$, these lines are accepted as an appropriate pair. Also, the angle obtained in this step is used as the corner angle whereas the intersection point of the two lines computed is used as the corner coordinate for further processing in Section 2.6 for triangle detection or Section 3.2 for rectangle detection.

**Algorithm 4:**

```
void FindPairs (int index, LineSegment *ls){
 int dist=0;
double *xInt = new double[100]; //intersection point x coordinate
 double *yInt = new double[100]; //intersection point y coordinate
for(int i=0;i<index;i++){
        for(int j=i+1;j<index;j++){
                dist=0;
                if( (ls[i].len>10 || ls[j].len>10)){
                        if(ls[i].len<80 || ls[j].len<80) dist=5;
                        else if (ls[i].len>=80 && ls[j].len>=80) dist=15;
                Find intersection points of these lines;
                    if(distance between the start and end coordinates of lines and intersection
                    point < dist) {
                        double angle=Compute angle between two lines; //(ls[i],ls[j]);
                                if ((angle>15 && angle<165) ) {
                                Assign these lines, intersection point and angle to the arrays;
                }/end-if }//end-if}/end-if }//end-innerfor }/end-for }//end-FindPairs
```

## 2.6. Combining Line Pairs and Straight Lines

While combining line pairs, the angles between the line segments in appropriate line pairs are computed. Four angles are found in this step. Three of them are the interior angles of the candidate triangle, whereas the fourth one is either the angle between the two lines that are supposed to be on the same side of the candidate triangle, or the angle between the same lines itself. These cases are illustrated in Figures 3.1a and 3.1b by $P_{12}$ and $P_{22}$, respectively. The values of the regarding angle can be up to $A_{min}$ as in Figure 2.2a, or equal to infinity as in Figure 2.2b. In the first case, it is assumed that a corner between two lines does not exist. In the second case, the corner between these lines was already found in Section 2.5. As alternative to these cases, Figure 2.2c illustrates another case that a line pair is combined with a single straight line. Here, there is no suitable line pair that can be combined with this line pair; therefore, it is compared with every single straight line in a given image.

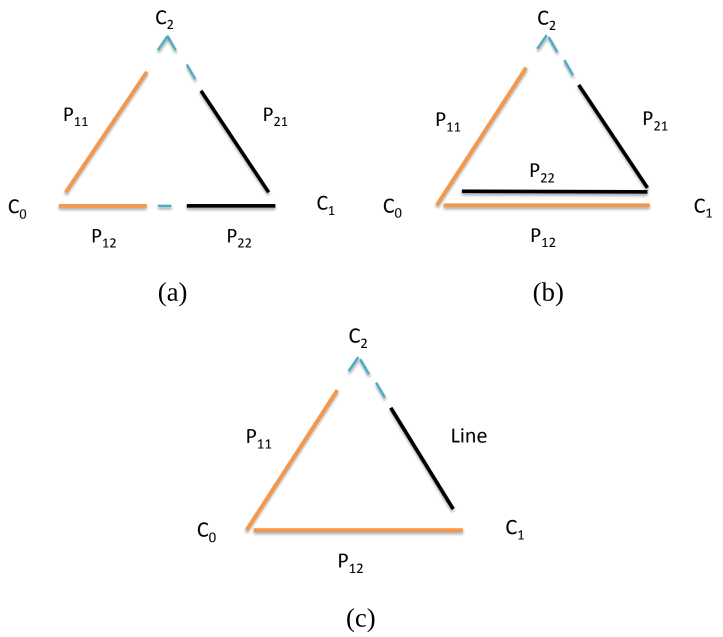


**Figure 2.2.** Line pairs positions for triangles. (a) Line pairs in different edge segments, (b) Line pairs have the same straight line. (c) A line pair joining with a single straight line

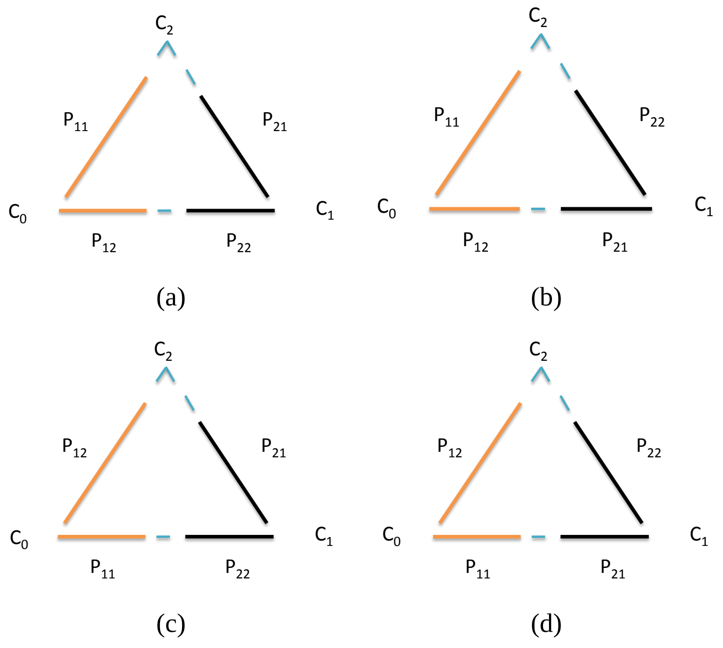


**Figure 2.3.** Line segments positions in line pairs for triangles (a, b, c, d)

In the first two cases described above shown in Figure 2.2, there can be four line segments positions as illustrated in Figure 2.3. In this figure, {$P_{11}$, $P_{12}$} and {$P_{21}$, $P_{22}$} compose possible line pairs, respectively.

In all those positions, intersection points of the lines are first computed. There are total of four intersection points, namely $IP_0$, $IP_1$, $IP_2$ and $IP_3$. Three of these points are corners of the candidate triangle. Since $C_0$ and $C_1$ are already obtained within the line pair detection step, $C_2$ needs to be computed now.

In the first position illustrated in Figure 2.3a, if the distance between the lines {$P_{12}$, $P_{22}$} is smaller than 3, these lines are collinear; hence, they do not constitute a corner. The distance between the obtained intersection point and $C_0$, which is called as $L_1$, is set to zero. Similarly, the obtained intersection point and $C_1$, represented by $L_2$, is set to zero as well. If the obtained intersection point is out of image boundaries, these lines are discarded, and {$L_1$, $L_2$} are set to zero. Otherwise, the distance between the obtained intersection point and the corners and the angles between the lines {$P_{12}$, $P_{22}$} are computed, and assigned to $L_1$, $L_2$, and $A_0$, respectively. The aforementioned process is repeated for the line pairs {$P_{12}$, $P_{21}$}, {$P_{11}$, $P_{22}$}, and {$P_{11}$, $P_{21}$}. In this way, {$L_3$, $L_4$, $A_1$}, {$L_5$, $L_6$, $A_2$}, {$L_7$, $L_8$, $A_3$} values are respectively obtained. Then, the resulting angles are checked to see if they are between $A_{min}$ and $A_{max}$. If there are exactly 3 angles meeting this criterion, the corners that had been found before ($C_0$ and $C_1$) are assigned as the candidate corners. In order to find the third and last corner ($C_2$), the lines forming this corner should be determined.

Considering the case illustrated in Figure 2.3a, if $A_0$ is lower than $A_{min}$, four requirements must be met.

- The distances between $IP_3$ and $C_0$, $IP_3$ and $C_1$ must be longer than $D_{max1}$ (because minimum line length is equal to 6).
- The ratio of the actual length of $P_{11}$ to $L_7$, and $P_{21}$ to $L_8$ must be between $R_{min}$ and $R_{max}$. The ratio of the summation of the lengths of $P_{12}$ and $P_{22}$ to $L_3$ must be between $R_{min}$ and $R_{max}$, too.
- $A_3$ must be between $A_{min}$ and $A_{max}$.
- The ratio between the summation of length of lines and circumference of candidate triangle must bigger than 0.6.

The second and fourth requirements are here necessary to overcome partial occlusion. If the start and end points of the lines corresponded to the actual corners of the triangle, this ratio would be 1.0. Thus, 40 percent margin of error is allowed. If these four requirements are satisfied, and the summation of $A_1$, $A_2$ and $A_3$ are between [180-ε, 180+ε], $C_2$ is found and it is equal to $IP_3$. If the lines lie between the candidate corner locations, they are fed into the validation stage. The abovementioned processes can be repeated in a similar manner for the other cases illustrated in Figures 2.3b, c and d. As a reference, Table 2.1 lists all possible lengths (visualized in Figure 2.4) in four line positions that are illustrated in Figure 2.3.

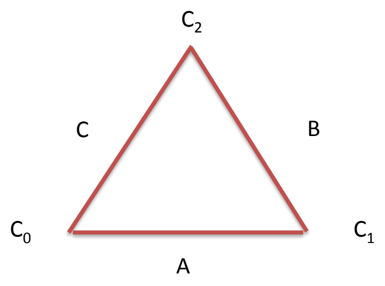


**Figure 2.4.** A sample triangle with lengths of edges

**Table 2.1.** Lengths in four line positions for triangles

| Lengths | Position 1 | Position 2 | Position 3 | Position 4 |
|---|---|---|---|---|
| **Lines that are the same side** | {$P_{12}$-$P_{22}$} | {$P_{12}$-$P_{21}$} | {$P_{11}$-$P_{22}$} | {$P_{11}$-$P_{21}$} |
| **$L_1$** | 0 | A | A | C |
| **$L_2$** | 0 | 0 | 0 | B |
| **$L_3$** | A | 0 | C | 0 |
| **$L_4$** | 0 | 0 | B | A |
| **$L_5$** | 0 | C | 0 | A |
| **$L_6$** | A | B | 0 | 0 |
| **$L_7$** | C | 0 | 0 | 0 |
| **$L_8$** | B | A | A | 0 |

In the case of combining line pair with a straight line segment process (Figure 2.2c); a straight line is selected in the different edge segment. The selected line segment must be in a certain area such that it is determined by longer line segments in the line pairs. If the start and end points of the line segment are inside the boundary of the circle, which has a radius that is equal to twice the length of the longer line segment, this line segment is selected and forwarded to further processing as described in the second step of Section 2.3.

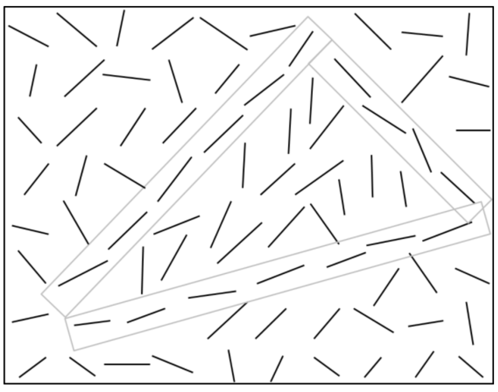
## 2.7. Validation of the Proposed Triangle Detection Algorithm

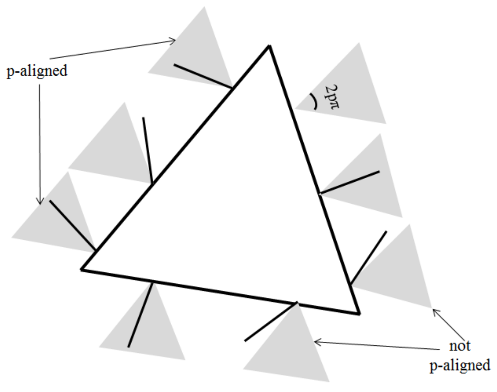
In the previous section, how the line segments are combined together to generate candidate triangles is described. As one would expect, some of the candidate triangles would be false detections and need to be eliminated before the final result is returned to the user.

To eliminate false detections, the proposed triangle detection algorithm employs the Helmholtz principle, which states that for a geometric structure to be perceptually meaningful, the expectation of this structure (grouping or Gestalt) by chance must be very low in a random situation (Desolneux et. al., 2004; Desolneux et. al., 2008). This is an '*a contrario*' approach, where the objects are detected as outliers of the background model. As shown in (Desolneux et. al., 2008), a suitable background model is one in which all pixels are independent.

They show that the simplest such model is the Gaussian white noise. In other words, no meaningful structure is perceptible in a Gaussian white noise image (Desolneux et. al., 2008).

Desolneux et al. use the Helmholtz principle to find meaning alignments, i.e., line segments, in a given image without requiring any parameters (Desolneux et. al., 2000). Their idea is to compute the level line orientation field (which is orthogonal to the gradient orientation field) of a given image, and look for a contiguous set of pixels having similar level line orientation. Figure 2.5a shows the level line orientation field for an image, where the aligned pixels that make up for a line segment (which represents one side of a triangle) are marked inside a rectangle. The authors define what aligned means as follows: Two points (or line segments) P and Q have the same direction, i.e., aligned, with precision p=1/8 if angle(P) and angle(Q) are within pπ = π/n degrees of each other. In the proposed triangle detection algorithm, the value of *'n'* is fixed to 8 and thus, '*p*', the precision or the accuracy of direction between two pixels, is equal to 1/8=0.125 and two points are aligned (or p-aligned) if their angles are within pπ = π/8 = 22.5 degrees of each other.


(a)



(b)

**Figure 2.5.** (a) Level line orientation field (orthogonal to the gradient orientation) of an image. Aligned pixels (inside each rectangle) clearly make up for one line segment of the triangle. With all three lines together, the triangle is clearly visible, (b) Illustration of several p-aligned and not p-aligned gradients. Observe that the aligned gradients are perpendicular to the level-line angle and are inside the tolerance cone of 2pπ

The gradient magnitude and the level line angle at a pixel (x, y) are computed using a 2x2 mask as follows (Desolneux et. al., 2000):

$$g_x(x,y) = \frac{I(x+1,y)-I(x,y)+I(x+1,y+1)-I(x,y+1)}{2} \quad (2.1)$$

$$g_y(x,y) = \frac{I(x,y+1)-I(x,y)+I(x+1,y+1)-I(x+1,y)}{2} \quad (2.2)$$

$$g(x,y) = \sqrt{g_x(x,y)^2 + g_y(x,y)^2} \quad (2.3)$$

$$angle(x,y) = \arctan\left(\frac{g_x(x,y)}{-g_y(x,y)}\right) \quad (2.4)$$

where I(x, y) is the intensity of the input image at pixel (x, y), g(x, y) is the gradient magnitude, and angle(x, y) is the level line angle. The reason for using such a simple gradient operator is to reduce the dependency of the computed gradient, and thus, preserve pixel independence as much as possible (Desolneux et al., 2000).

To make validation by the Helmholtz principle concrete, Desolneux et al. define what is called the NFA of a line segment as follows (Desolneux et. al., 2000): Let L be a line segment of length *'n'* with at least *'k'* points having their directions aligned with the direction of L in an image of size NxN pixels. Define NFA of L as (von Gioi et al., 2008; von Gioi et al., 2010):

$$NFA(n,k) = N^4 \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{n-i} \quad (2.5)$$

where $N^4$ represents the number of potential line segments in an NxN image. This is due to the fact that a line segment has two end-points, each of which can be located in any of the $N^2$ pixels of the image; thus, a total of $N^2$x$N^2$ line segments. The probability '*p*' used in the computation of the binomial tail is the accuracy of the alignment between the pixel's level line angle and the line segment.

An event (a line segment in this case) is called ε-meaningful if its NFA($n, k$) $\leq$ ε. Desolneux et al. (Desolneux et. al., 2004; Desolneux et. al., 2008) advises setting ε to 1, which corresponds to one false detection per image. Given these definitions, a line segment is validated as follows: For a line segment of length '$n$', compute the level line angle of each pixel along the line and count the number of aligned pixels '$k$'. Then, compute NFA($n, k$) and accept the line segment as valid if NFA($n, k$) $\leq 1$. Otherwise, reject the line segment.

Desolneux et. al.'s line segment validation framework outlined above can easily be adapted to triangle validation. Since a triangle consists of three line segments each representing one side of the triangle, the sides of the triangle are just needed to treat as an extended line segment, with each side having a different level line orientation. Figure 2.5a shows the level line orientation field of an image, where the aligned pixels that make up for each side of the triangle are marked inside three rectangles. Figure 2.5 shows the gradient directions (perpendicular to the level line angles) of several points on the boundary of a triangle, some of which are p-aligned with the triangle, and some of which are not. The gray triangle illustrates the tolerance cone between the ideal gradient direction and the observed gradient direction. If the observed gradient direction is inside the cone, then the point is assumed to be aligned with the side of the triangle, otherwise the point is assumed to be non-aligned with the side of the triangle.

The definition of the NFA of a line segment to a triangle is adapted as follows: Let T be a triangle having a perimeter of '$n$' points with at least '$k$' points having their directions aligned with T in an image of size NxN pixels. Define NFA of T as:

$$NFA(n,k) = N^6 \sum_{i=k}^{n} \binom{n}{i} p^i \, (1-p)^{n-i} \tag{2.6}$$

where $N^6$ represents the total number of potential triangles in an NxN image. This is due to the fact that a triangle has 3 corners and each corner can be located in any of the $N^2$ points of the image for a total of $N^6$ triangles.

Given this NFA definition, a triangle is validated as follows: For a triangle having a perimeter of '*n*' pixels, compute the level line angle of each pixel along the sides of the triangle and count the number of aligned pixels '*k*'. Compute NFA(*n, k*) and accept the triangle as valid if NFA(*n, k*) $\leq \varepsilon$; otherwise the triangle is rejected. It is important to point out that the gradient computation is performed in the validation step over the original non-filtered image as required by the *'a contrario'* framework.

The epsilon ($\varepsilon$) in the above comparison denotes the expected number of detections under the background model. In other words, if gaussian white noise image is fed into the algorithm, at most $\varepsilon$ many detections shold be get. $\varepsilon$ is set to 1 as advised by Desolneux et al. (Desolneux et. al., 2004; Desolneux et. al., 2008), which corresponds to one false detection per image.

The candidate rectangles are validated by a similar procedure to this section.

## 2.8. Experimental Work

Performance of the proposed triangle detection algorithm was evaluated through a comprehensive experimental work. Specifically, accuracy and processing time of the algorithm was measured using four distinct datasets containing both synthetic and natural images. During the experiments, all results were presented with a comparison to well-known OpenCV triangle detection algorithm (a derivative of squares.cpp that occurs in the OpenCV samples in …/opencv/samples/c/) (Bradski and Kaehler, 2008) in terms of accuracy and processing time. In the following subsections, the employed image datasets are described; accuracy and processing time analysis are comparatively provided.

### 2.8.1. Datasets

In order to measure the performance of the proposed triangle detection algorithm, four datasets (Table 2.2) were employed. The first two datasets are respectively composed of synthetic and handmade images containing overlapped, partially occluded, discontiguous and arbitrary triangles. The remaining two datasets contain natural images. While one of them has various scenes, the other has specifically road scenes with traffic signs. All triangles within the datasets are manually tagged. Sample images from all four datasets are provided in Figures 2.6-2.9a.

**Table 2.2.** The triangle datasets description

| Dataset | Image Type | Image Size | # of images | # of Triangles |
|---|---|---|---|---|
| **1** | Synthetic | 640 × 480 | 11 | 300 |
| **2** | Handmade | 640 × 480 | 14 | 323 |
| **3** | Real (Various Scenes) | 640 × 480 | 36 | 116 |
| **4** | Real (Road Scenes) | 360 × 270 | 48 | 43 |

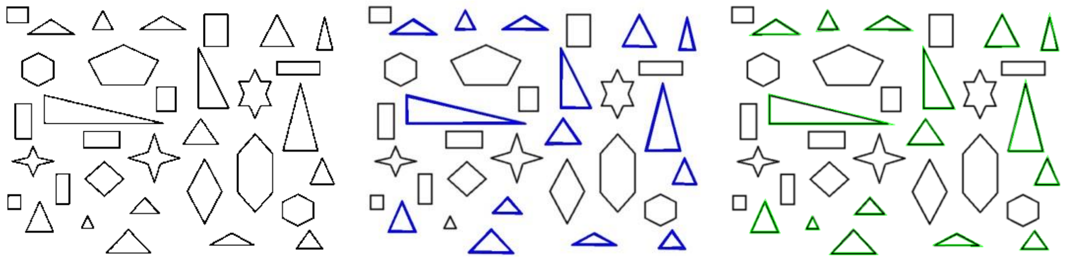

(a) (b) (c)

**Figure 2.6.** A sample synthetic image and results of triangle detection algorithms. (a) A sample synthetic image from Dataset 1. (b) The triangles detected by OpenCV triangle detection algorithm (overlayed by blue color). (c) The triangles detected by the proposed algorithm

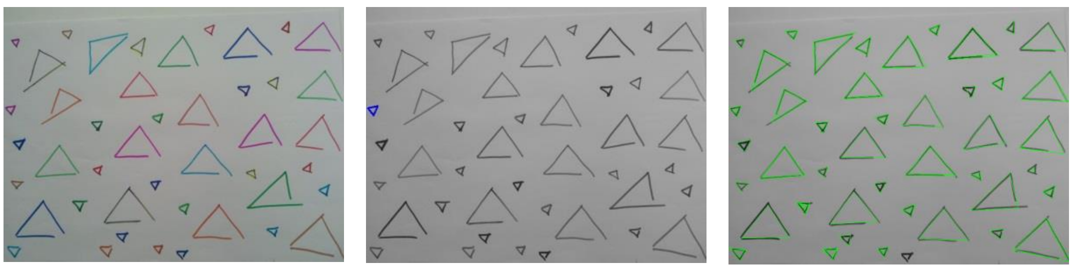
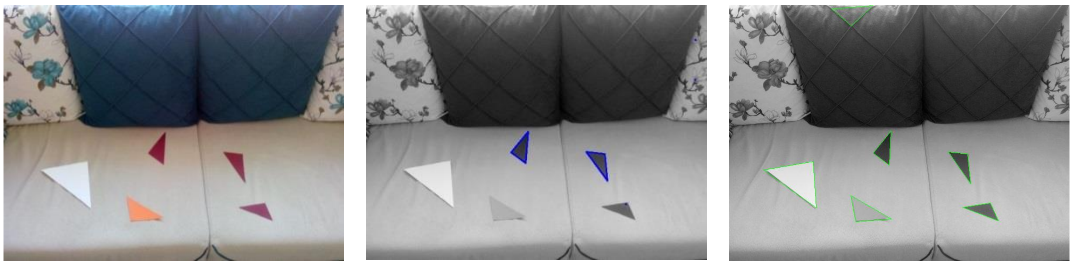
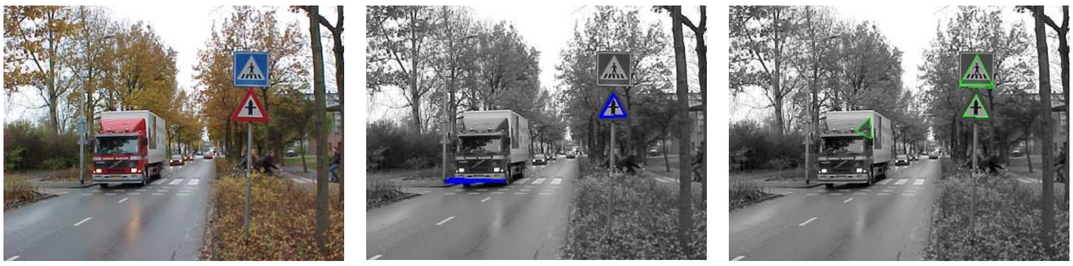

(a) (b) (c)

**Figure 2.7.** A sample handmade image and results of triangle detection algorithms. (a) A sample handmade image from Dataset 2. (b) The triangles detected by OpenCV triangle detection algorithm. (c) The triangles detected by the proposed algorithm



(a) (b) (c)

**Figure 2.8.** A sample real image and results of triangle detection algorithms. (a) A sample real image from Dataset 3. (b) The triangles detected by OpenCV triangle detection algorithm. (c) The triangles detected by the proposed algorithm



(a) (b) (c)

**Figure 2.9.** A sample road scene image and results of triangle detection algorithms. (a) A sample road scene image from Dataset 4. (b) The triangles detected by OpenCV triangle detection algorithm. (c) The triangles detected by the proposed algorithm

### 2.8.2. Accuracy analysis

Since both precision and recall scores of the detection algorithm are of concern, well-known F-score, which takes both into consideration, was used for evaluation. The detection results of the proposed algorithm and OpenCV counterpart are comparatively given in Table 2.3 where the highest F-scores for each dataset are indicated in bold.

**Table 2.3.** Detection results of the proposed triangle detection algorithm / OpenCV

| **Dataset** | **TP** | **FP** | **FN** | **Precision** | **Recall** | **F-score** |
|---|---|---|---|---|---|---|
| **1** | 298 / 268 | 4 / 0 | 2 / 32 | 0.98 / 1.00 | 0.99 / 0.89 | **0.99** / 0.94 |
| **2** | 295 / 113 | 4 / 2 | 28 / 210 | 0.98 / 0.98 | 0.91 / 0.34 | **0.94** / 0.50 |
| **3** | 103 / 44 | 27 / 27 | 13 / 72 | 0.79 / 0.61 | 0.88 / 0.37 | **0.83** / 0.46 |
| **4** | 39 / 9 | 2 / 4 | 4 / 34 | 0.95 / 0.69 | 0.90 / 0.20 | **0.92** / 0.32 |

One can easily see that the proposed algorithm surpassed OpenCV algorithm for all datasets. The success of the proposed algorithm is more obvious especially in the datasets 2, 3 and 4. Since OpenCV triangle detection algorithm relies on complete contours, it fails in real images where complete contours are hardly found. On the other hand, the proposed algorithm successfully detects triangles even in such conditions. While Figures 2.6-2.9b display the triangles detected by OpenCV algorithm on sample images from each dataset, Figures 2.6-2.9c display the ones detected by the proposed algorithm.

However, there are a few cases where the proposed triangle detection algorithm fails to detect some of the valid triangles in the image. Figure 2.10 shows some of those cases. Since the performance of the proposed algorithm thoroughly depends on the outcome of the edge segment detection algorithm, ED, a triangle cannot be constructed if there is no edge between the corners. This can be seen in Figure 2.10b, where the triangles with dotted edges are not detected. This is expected since the proposed algorithm works on an edge segment that must trace the boundary of the triangle, whereas there would be no edge segments extracted from the sides of a triangle if the side consists of very short dashed lines. So it is impossible for the proposed algorithm to detect such triangles and it is seen in Figure 2.10b. The other failure case is when the interior angle of the triangle is very small as shown Figure 2.10a, and it remains out of the bounds of the preset angle tolerance interval. In such cases, the proposed algorithm would not be able to construct any candidate triangles, and no detections would be possible. By changing the pre-determined parameters for each image, it may be possible to detect such triangles; but it is believed that using a single set of fixed parameters for all images is an important property of the proposed algorithm even if this would mean that the proposed algorithm would miss some valid triangles present in the image.

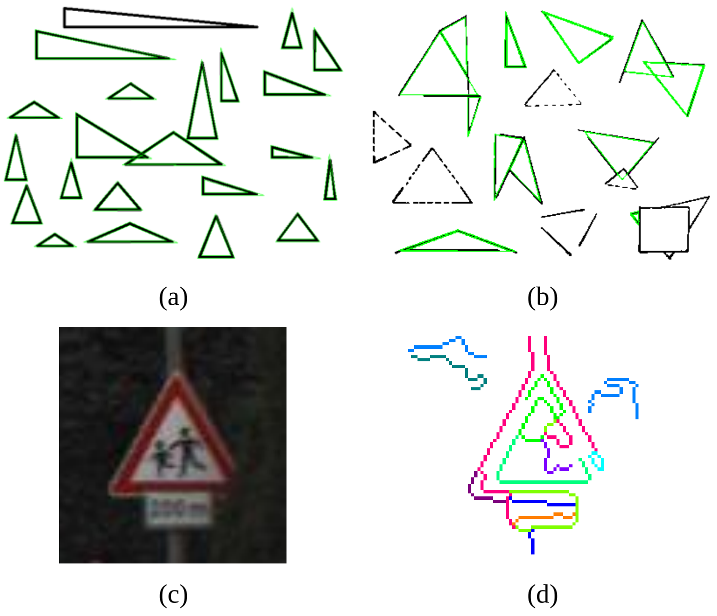


**Figure 2.10.** Sample images on which the proposed algorithm fails to detect triangles

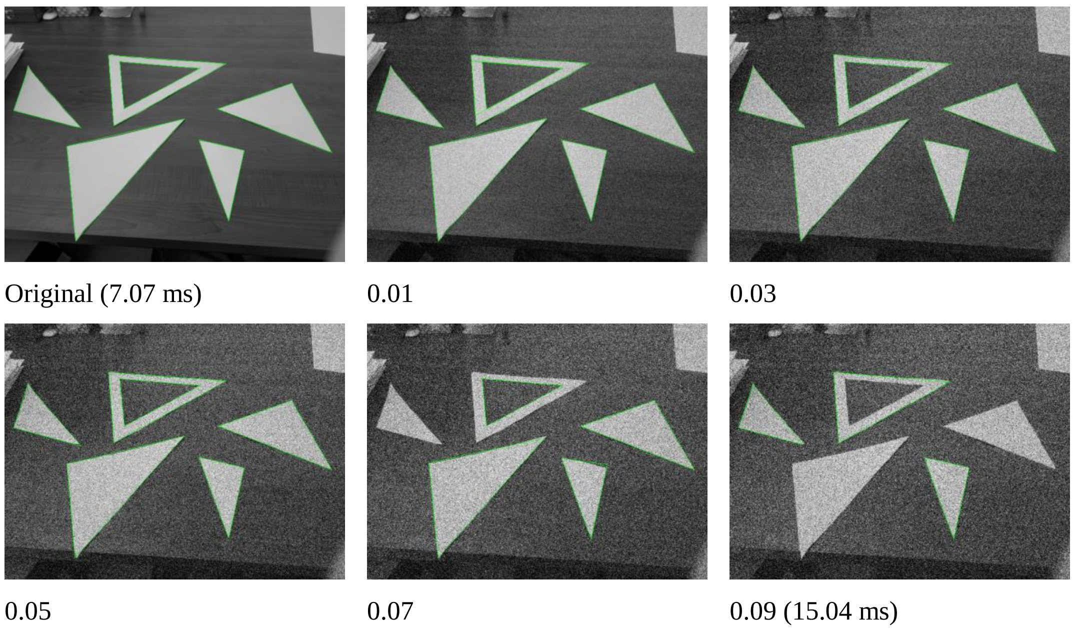


**Figure 2.11.** The performance of the proposed triangle detection algorithm the increasing Gaussian white noise on a sample image. The noise added images were obtained by using the MATLAB function imnoise(img, 'gaussian', mean, variance) with mean=0 and variance increasing from 0.01 to 0.09. Increasing the noise further causes complete detection failure

The last experiment is to measure the performance of the proposed algorithms in noisy images. To this end, an image containing several small and big triangles for the proposed triangle detection algorithm and rectangles for the proposed rectangle detection algorithm  is taken, added varying levels of Gaussian white noise to the image, and fed the images to the proposed algorithm, respectively.

Figure 2.11 demonstrates the performance of the proposed triangle detection algorithm as the increasing Gaussian white noise on a sample image. The noise added images were obtained by using the MATLAB function imnoise (img, 'gaussian', mean, variance) with mean=0 and variance increasing from 0.01 to 0.09.

Increasing the noise further causes complete detection failure. The reason for the detection failure comes from the fact that as the noise is increased, the boundaries of the triangles are approximated by many short edge segments instead of a few long edge segments as would be the case in less noisy images. It is important to stress that there are no false detections in none of the images, which is also very important. It is also observe that the running time of proposed triangle detection algorithm increases in noisy images. The reason for this is the increased edge segment detection time by EDPF. As the amount of noise increases in an image, EDPF takes more time to compute the edge segments because many pixels start becoming potential edge elements. Triangle detection after edge segment detection remains pretty constant across all images.

### 2.8.3. Processing time analysis

The proposed algorithm was implemented in C++ language, and evaluated on a PC equipped with an Intel Core i7-3770K 3.5 GHz CPU and 16 GB of RAM. Average processing times per image are comparatively listed in Table 2.4.

Although the proposed algorithm fails to surpass OpenCV in terms of processing time, the required processing time is tolerable considering the superior detection performance offered by the proposed algorithm. Besides, these values are in still in compatible with real time operations such that more than 45 frames per second can be easily processed in the worst case scenario.

**Table 2.4.** Average processing times (ms) of the triangle detection algorithms

| Dataset | OpenCV | Proposed Algorithm |
|---|---|---|
| **1** | 1.07 | 15.88 |
| **2** | 1.31 | 11.82 |
| **3** | 4.26 | 21.37 |
| **4** | 2.31 | 5.43 |

## 3. THE RECTANGLE DETECTION ALGORITHM

After obtaining straight lines in both closed and non-closed edge segments according to meet the requirements mentioned in Section 2.2, the further operations for finding rectangles are processed. Those are described in detail between Sections 3.1 and 3.2.

### 3.1. Finding Closed Edge Segments and Extraction of Complete Rectangles

The first step after the detection of the edge segments, all edge segments are checked, take the closed ones and see if the closed edge segment traces the entire boundary of a rectangle. To decide whether four lines form a quadrangle, their mutual location, parallelism and gap value in the corners must be analyzed (Lagunovsky and Ablameyko, 1999). In order to achive this, a rotated rectangle is fitted to the pixels making up the edge segment and a check is performed to see if the number of straight lines in the segment is equal to four (because rectangles have four sides). If the number of lines is bigger than four, sort the lengths of these straight lines according to descending order.

In the next step, the interior four angles of candidate rectangle are computed. As used in the proposed triangle detection algorithm, $A_{min}$ and $A_{max}$, $D_{max}$ and $R_{min}$ and $R_{max}$ are used to verify that those four lines are the sides of the rectangle. If the summation of the angles are close to 360 ([360-ε, 360+ε]), each angle is between $A_{min}$ and $A_{max}$ and the summation of consecutive two angles is close to 180 ([180-ε, 180+ε]), intersection points of each pair of lines are found. The usage of $R_{min}$ and $R_{max}$ is the same as the Algorithm 3 in the Section 2.3. In the Algorithm 3, there are there lines for the determination of the sides of the candidate triangle; in this case there are four lines for the determination. The algorithm for rectangle processing of a closed edge segment is provided in Algorithm 5.

**Algorithm 5:**

```
if(number of lines>=4){
```

```
if(segment of interest is closed){
    bool quadrangle=false;
    if(number of lines in the edge segment==4){
    Find the angles between lines;
    if(345<summation of these angles<375){
        if(20<the angles<160 && 160<summation of consecutive two angles<200){
            Find intersection points of four lines;
            Assign these intersection points to the rectangles struct as the corners of
            candidate quadrangle;
            if( the start and end coordinates of lines are not between corner
        coordinates)  {quadrangle=false;}
            else{
                Compute distances between corners;
                if( distance between corners)>9){
                    if(actual lengths of lines/distances(ratios)<0.35 &&
                ratios>2)
                    {quadrangle=false;}
                    else{
                        if(Validatation of Quadrangles==true){
                        numberOfQuadrangles++;
                        quadrangle=true;
                    }//end-if}//end-else  }//end-if}//end-else
}/end-if}//end-if}/end-if}/end-if}//end-if
```

## 3.2. Combining Line Pairs and Straight Lines

In this step of proposed algorithm, the first idea is to form a rectangle with combining two appropriate line pairs. In combining two line pairs process, the angles between the line segments in these line pairs are computed. Four angles are found in this step.

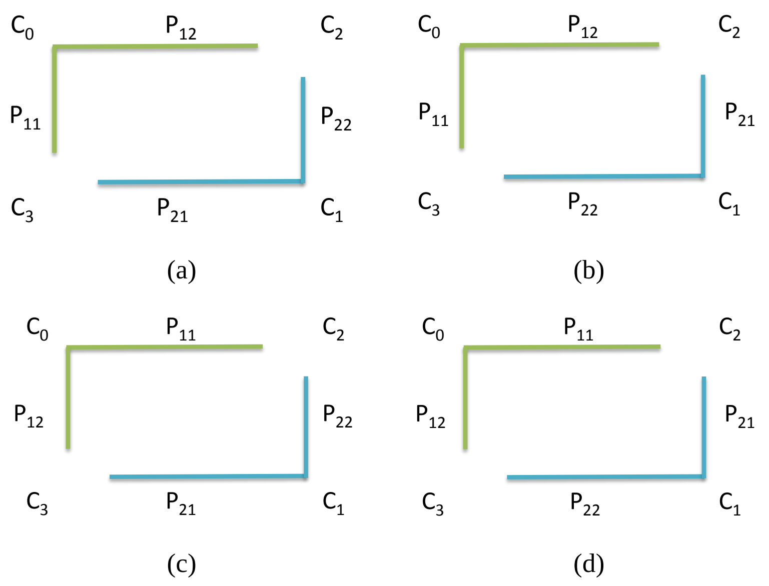


**Figure 3.1.** Line segments positions in joining two line pairs for rectangles

There can be four line segments positions as illustrated in Figure 3.1. In this figure, $\{P_{11}, P_{12}\}$ and $\{P_{21}, P_{22}\}$ compose possible line pairs, respectively.

In all those positions, firstly intersection points of lines are computed. There are total of four intersection points, namely $IP_0$, $IP_1$, $IP_2$ and $IP_3$. If specified conditions are satisfied in all positions, for the first position as shown in Figure 3.1a, $C_2$ and $C_3$ are $IP_0$, $IP_3$, for the second position as shown in Figure 3.1b, $C_2$ and $C_3$ are $IP_1$, $IP_2$, for the third position as shown in Figure 3.1c, $C_2$ and $C_3$ are $IP_2$, $IP_1$, and for the last position, line segments in the first line pair as shown in Figure 3.1d, $C_2$ and $C_3$ are $IP_3$, $IP_0$, respectively.

In the first position illustrated in Figure 3.1a, if the distance between the lines $\{P_{12}, P_{22}\}$ is smaller than 3, these lines are collinear; hence, they do not constitute a corner. The distance between the obtained intersection point and $C_0$, which is called as $L_1$, is set to zero. Similarly, the obtained intersection point and $C_1$, represented by $L_2$, is set to zero as well. If the obtained intersection point is out of image boundaries, these lines are discarded, and $\{L_1, L_2\}$ are set to zero. Otherwise, the distance between the obtained intersection point and the corners and the angles between the lines $\{P_{12}, P_{22}\}$ are computed, and assigned to $L_1$, $L_2$, and $A_0$, respectively. The aforementioned process is repeated for the line pairs

{$P_{12}$, $P_{21}$}, {$P_{11}$, $P_{22}$}, and {$P_{11}$, $P_{21}$}. In this way, {$L_3$, $L_4$, $A_1$}, {$L_5$, $L_6$, $A_2$}, {$L_7$, $L_8$, $A_3$} values are respectively obtained. Then, the resulting angles are checked to see if they are between $A_{min}$ and $A_{max}$. If there are exactly 2 angles meeting this criterion, the corners that had been found before ($C_0$ and $C_1$) are assigned as the candidate corners. In order to find the third and fourth corner ($C_2$ and $C_3$), the lines forming these corner should be determined.

Considering the case illustrated in Figure 3.1a, if $IP_0$ is not equal to $C_0$ and $C_1$, three requirements must be met.

- $L_1$, $L_2$, $L_7$, $L_8$ must be longer than $D_{max1}$ (because minimum line length is equal to 6).
- The ratio of the actual length of $P_{11}$ to $L_7$, $P_{12}$ to $L_1$, $P_{22}$ to $L_2$, and $P_{21}$ to $L_8$ must be between $R_{min}$ and $R_{max}$.
- The ratio between the summation of length of lines and circumference of candidate rectangle must bigger than 0.6.

The last two requirements are here necessary to overcome partial occlusion. If the start and end points of the lines corresponded to the actual corners of the rectangle, this ratio would be 1.0. Thus, 40 percent margin of error is allowed. If these three requirements are satisfied, $C_2$ is found and it is equal to $IP_0$, and if the summation of $A_0$, $A_1$, $A_2$ and $A_3$ are between [360-ε, 360+ε], and the consecutive angles are between [180-ε, 180+ε], and $IP_3$ is not equal to $C_0$ and $C_1$, $C_3$ are found and it is equal to $IP_3$. If the lines lie between the candidate corner locations and differences between $L_1$, $L_2$ and $L_7$, $L_8$ are smaller than $L_{max1}$ (parallelism tolerance), they are fed into the validation stage. The abovementioned processes can be repeated in a similar manner for the other cases illustrated in Figures 3.1b, c and d. As a reference, Table 3.1 lists all possible lengths (visualized in Figure 3.2) in four line positions that are illustrated in Figure 3.1.

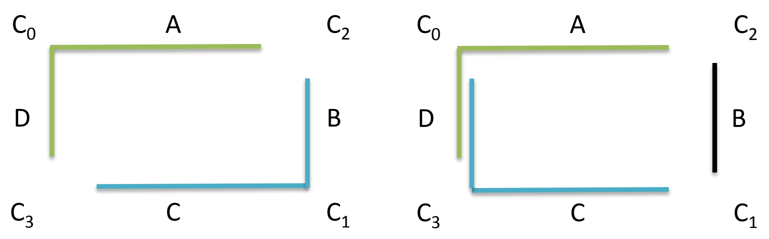


**Figure 3.2.** Sample rectangles formed line pairs. (a) Formed two line pairs. (b) Formed two line pairs and a single straight line

**Table 3.1.** Lengths in four line positions for rectangles

| Lengths | Position 1 | Position 2 | Position 3 | Position 4 | Interval |
|---|---|---|---|---|---|
| | {$P_{12}$-$P_{22}$} | {$P_{12}$-$P_{21}$} | {$P_{11}$-$P_{22}$} | {$P_{11}$-$P_{21}$} | (intersection points-corner) |
| **$L_1$** | A | ∞ | ∞ | D | { $P_{12}$-$P_{22}$ } - $C_0$ |
| **$L_2$** | B | ∞ | ∞ | C | { $P_{12}$-$P_{22}$ } - $C_1$ |
| **$L_3$** | ∞ | A | D | ∞ | { $P_{12}$-$P_{21}$ } - $C_0$ |
| **$L_4$** | ∞ | B | C | ∞ | { $P_{12}$-$P_{21}$ } - $C_1$ |
| **$L_5$** | ∞ | D | A | ∞ | { $P_{11}$-$P_{22}$ } - $C_0$ |
| **$L_6$** | ∞ | C | B | ∞ | { $P_{11}$-$P_{22}$ } - $C_1$ |
| **$L_7$** | D | ∞ | ∞ | A | {$P_{11}$-$P_{21}$ } - $C_0$ |
| **$L_8$** | C | ∞ | ∞ | B | {$P_{11}$-$P_{21}$ } - $C_1$ |

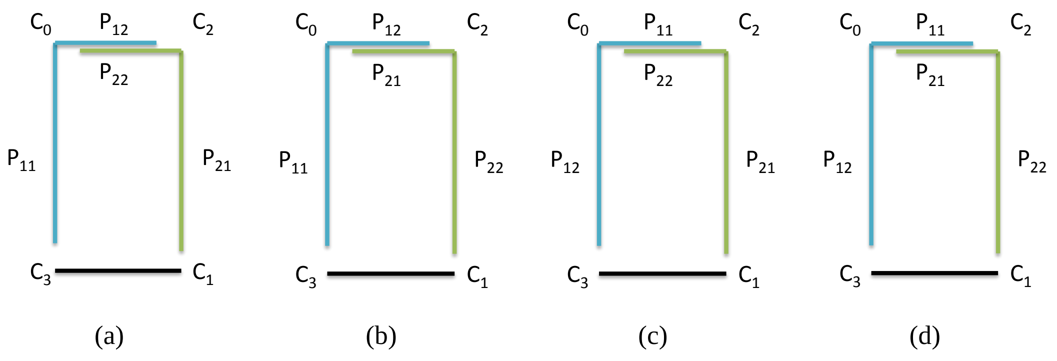


**Figure 3.3.** Line segments positions in joining a single line segment (1)

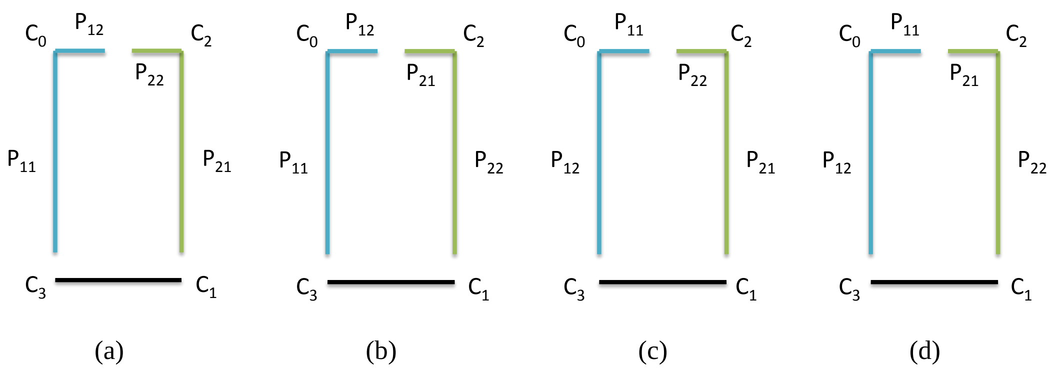


**Figure 3.4.** Line segments positions in joining a single line segment (2)

In this step, the second idea is to combine two line pairs with a single line segment process as shown in Figure 3.2b; there can be four line segments positions similar to the previous process. In the first position shown in Figure 3.3a or Figure 3.4a, the selected straight line is compared with lines $P_{11}$, $P_{21}$. In the second position shown in Figure 3.3b or Figure 3.4b, the selected straight line is compared with lines $P_{11}$, $P_{22}$. In the third position shown in Figure 3.3c or Figure 3.4c, the selected straight line is compared with lines $P_{12}$, $P_{21}$. In the last position shown in Figure 3.3d or Figure 3.4d, the selected straight line is compared with lines $P_{12}$, $P_{22}$.

In Figures 3.3-3.4(a-d) combining two line pairs with a straight line segment process, a straight line is selected in the different edge segment. The selected line segment must be in a certain area. This area is determined by longer line segment. This is $P_{11}$ or $P_{21}$ in the first position shown in Figures 3.3a and 3.4a. Then $P_{11}$, $P_{21}$, selected line, $P_{12}$ or $P_{22}$ (the longer one is selected.) are fed into rectangle detection algorithm. If start and end points of line segment are inside the boundary of circle which has a radius that is equal to twice the length of longer line segment, this line segment is selected and forwarded to further processing as described in the second step of Section 3.1.

## 3.3. Experimental Work

Performance of the proposed rectangle detection algorithm was evaluated in a similar manner to that of triangle detection algorithm mentioned previously. In this case, specifically, accuracy and processing time of the algorithm was measured using three distinct datasets containing both synthetic and natural images. In the following subsections, the employed image datasets are described; accuracy and processing time analysis are comparatively provided.

### 3.3.1. Datasets

In order to measure the performance of the proposed rectangle detection algorithm, three datasets (Table 3.2) were employed. The first two datasets are respectively composed of synthetic and handmade images containing overlapped, partially occluded, discontiguous and various types of rectangles. The remaining dataset contains natural images has various scenes. All rectangles within the datasets are manually tagged as in the triangle detection algorithm. Sample images from all three datasets are provided in Figures 3.5-3.7a.

**Table 3.2.** The rectangle datasets description

| Dataset | Image Type | Image Size | # of images | # of Rectangles |
|---|---|---|---|---|
| **1** | Synthetic | 640 × 480 | 10 | 199 |
| **2** | Handmade | 640 × 480 | 7 | 176 |
| **3** | Real (Various Scenes) | 640 × 480 | 36 | 186 |

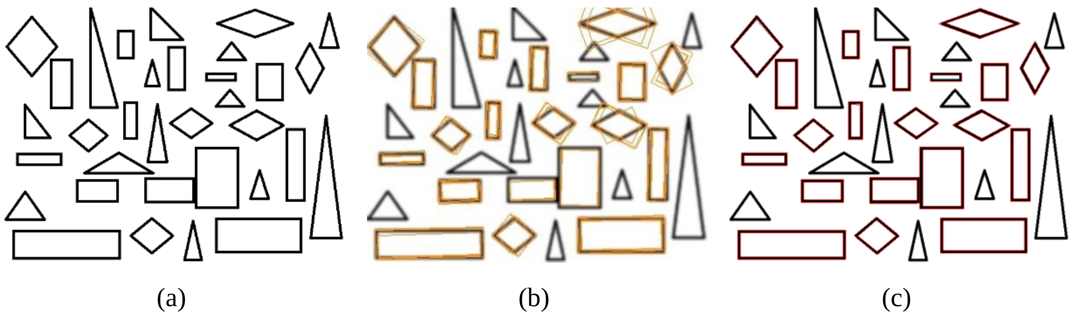
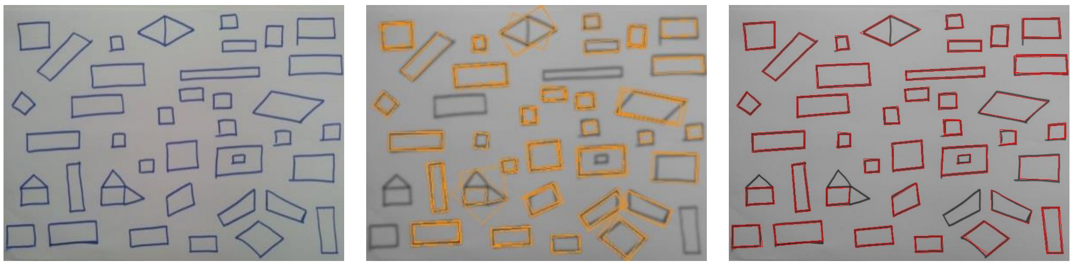
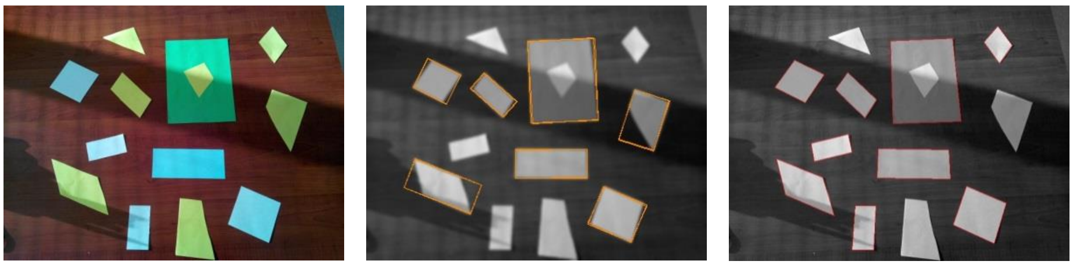


**Figure 3.5.** A sample synthetic image and results of rectangle detection algorithms. (a) A sample synthetic image from Dataset 1. (b) The rectangles extracted by OpenCV rectangle detection algorithm. (c) The rectangles detected by the proposed algorithm



(a) (b) (c)

**Figure 3.6.** A sample handmade image and results of rectangle detection algorithms. (a) A sample handmade image from Dataset 2. (b) The rectangles extracted by OpenCV rectangle detection algorithm. (c) The rectangles detected by the proposed algorithm



(a) (b) (c)

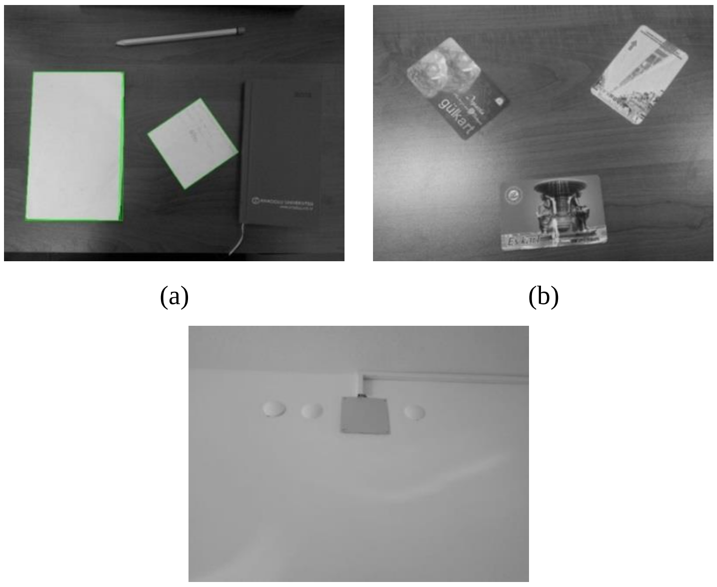
**Figure 3.7.** A sample real image and results of rectangle detection algorithms. (a) A sample real image from Dataset 3. (b) The rectangles extracted by OpenCV rectangle detection algorithm. (c) The rectangles detected by the proposed algorithm

### 3.3.2. Accuracy analysis

Since both precision and recall scores of the detection algorithm are of concern, well-known F-score, which takes both into consideration, was used for evaluation. The detection results of the proposed algorithm and OpenCV counterpart by performing Canny edge detector and Hough Line using Emgu CV using the Contour class to detect rectangular objects is comparatively given in Table 3.3 where the highest F-scores for each dataset are indicated in bold.

**Table 3.3.** Detection results of the proposed rectangle detection algorithm / OpenCV

| Dataset | TP | FP | FN | Precision | Recall | F-score |
|---|---|---|---|---|---|---|
| **1** | 198 /186 | 1 / 8 | 1 / 13 | 0.99 / 0.96 | 0.99 / 0.93 | **0.99** / 0.95 |
| **2** | 176 / 132 | 6 / 16 | 0 / 44 | 0.96 / 0.89 | 1.00 / 0.75 | **0.98** / 0.81 |
| **3** | 160 / 61 | 8 / 5 | 26 / 125 | 0.95 / 0.92 | 0.86/ 0.32 | **0.90** / 0.48 |


(a) (b)

(c)

**Figure 3.8**. Sample images on which the proposed algorithm fails to detect rectangles

It is clearly visible that the proposed algorithm surpassed OpenCV algorithm for all datasets. The success of the proposed algorithm is more obvious especially in the datasets 2 and 3. While Figures 3.5-3.7b display the rectangles detected by OpenCV algorithm on sample images from each dataset, Figures 3.5-3.7c display the ones detected by the proposed algorithm.

However, there are few cases where the proposed algorithm fails to detect. Figure 3.8 shows some of those cases. Since the algorithm thoroughly depends on the edge detection algorithm, a rectangle cannot be constructed if there is no edge between the corners.

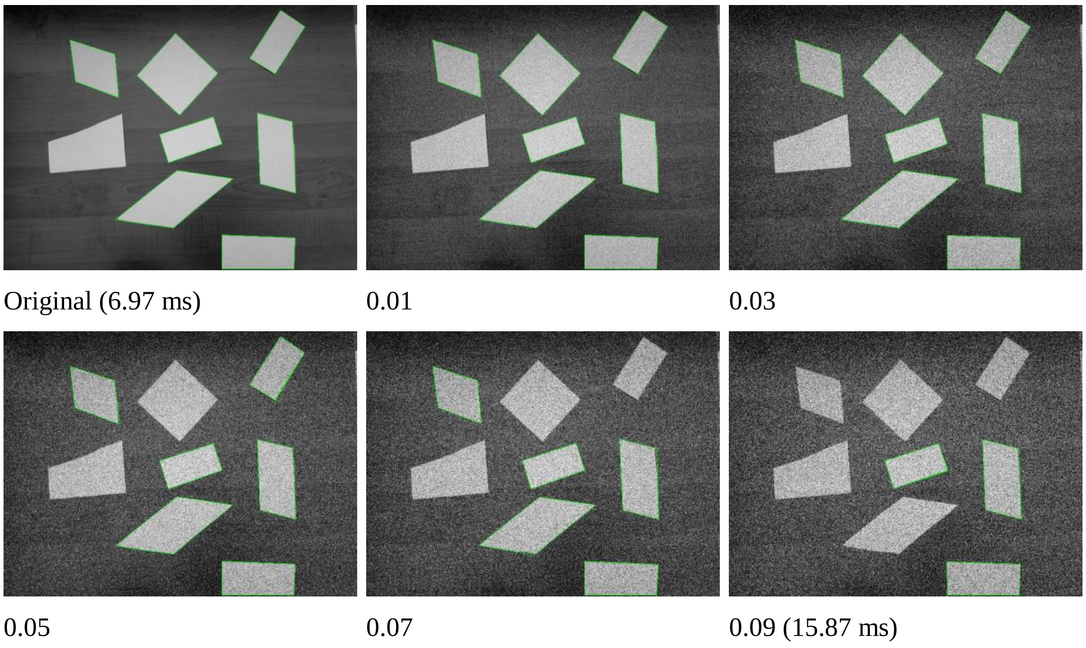


**Figure 3.9.** The performance of the proposed rectangle detection algorithm as the increasing Gaussian white noise on a sample image. The noise added images were obtained by using the MATLAB function imnoise(img, 'gaussian', mean, variance) with mean=0 and variance increasing from 0.01 to 0.09. Increasing the noise further causes complete detection failure

The performance of the proposed rectangle detection algorithm in noisy images is shown in Figure 3.9. To obtain the noisy images, the processes are used as in the Section 2.7.2. After obtaining noisy images, they are fed into the proposed algorithm.

In a similar manner to the proposed triangle detection algorithm, increasing the noise further causes complete detection failure and increases the processing time. In the noisy images, the rectangle sides forming the line segments obtained are divided into many different edge segments, for this reason the proposed algorithm can not reach the right result.

### 3.3.3. Processing time analysis

The proposed rectangle detection algorithm was implemented in the similar way proposed triangle detection algorithm. Average processing times per image are comparatively listed in Table 3.4. Although the proposed algorithm fails to surpass OpenCV in terms of processing time in the real images, the processing time of the proposed rectangle detection algorithm permits the real time operations. More than 40 frames per second can be easily processed with this algorithm in the worst case scenario.

**Table 3.4.** Average processing times (ms) of the rectangle detection algorithms

| Dataset | OpenCV | Proposed Algorithm |
|---|---|---|
| **1** | 22,2 | 13,21 |
| **2** | 21,57 | 25,64 |
| **3** | 14,80 | 16,14 |

# 4. CONCLUSIONS

In this thesis dissertation, novel methods are proposed for rotated rectangle and triangle detection in digital images. Both of the proposed methods use line segments derived from a recently developed edge detection algorithm. The edge detection algorithm used first converts each edge segments, which is a contiguous chain of pixels, into a set of line segments. Some straight line segments obtained are then merged into a single segment according to several perceptual grouping criteria.  The straight line segments are compared with each other to obtain the appropriate pairwise.

The proposed algorithms are capable of quickly, accurately and simultaneously detecting various types of triangles such as wide-angled, narrow-angled, right-angled and rectangles such as square, rhombus, parallelogram, etc. on both synthetic and real images. Geometrically distorted, poorly illuminated and even occluded triangles and rectangles can be also detected with high accuracies.

The proposed algorithms are tested with synthetic and natural images. Both of the algorithms are compared with OpenCV triangle and rectangle detection algorithms, which are quite common methods used in computer vision field. The proposed algorithms clearly surpass OpenCV algorithms in terms of accuracy. Even if the processing time of the proposed triangle detection algorithm is higher than that of OpenCV counterpart, it still allows real-time operations.

Both of the proposed algorithms fail to detect in some cases. Some of those cases are described as follows. A side forming more than one line segment obtained can be divided into many different edge segments. In the proposed algorithms, there are only the line segments in the same edge segment are merged each other. For this reason, the proposed algorithm can not reach the right result in such cases. If all lines which are parallel to each other are merged regardless of whether they belong to which edge segments, all the triangles or rectangles in the image, for example nested triangles or rectangles may not be detected.

In order to compute the angles and to determine the sides of the candidate triangles and rectangles, some parameters such as angle tolerance, distance tolerance and ratio tolerance are used. These parameters are determined to give the best results as far as possible. The other case is mentioned above is to remain outside the determined tolerances. If you give different parameter values, you can get the different results.

Adaptation of the proposed algorithms for the detection of other polygonal shapes remains as an interesting future work.

## REFERENCES

Akinlar, C. and Topal, C. (2011), "EDLines: A real-time line segment detector with a false detection control," *Pattern Recognition Letters*, **32**(13), 1633-1642.

Akinlar, C. and Topal, C. (2012), "Edpf: A Real-Time Parameter-Free Edge Segment Detector with a False Detection Control," *International Journal of Pattern Recognition and Artificial Intelligence*, **26**(1).

Akinlar, C. and Topal, C. (2013), "EDCircles: A real-time circle detector with a false detection control," *Pattern Recognition*, **46**(3), 725-740.

Ballard, D. H. (1981), "Generalizing the Hough Transform to Detect Arbitrary Shapes," *Pattern Recognition*, **13**(2), 111-122.

Barnes, N., Loy, G. and Shaw, D. (2010), "The regular polygon detector," *Pattern Recognition*, **43**(3), 5,92-602.

Bradski, G. and Kaehler, A. (2008), "*Laerning OpenCV: Computer Vision with the OpenCV Library*," O'Reilly Media.

Bruno, L., Parla, G. and Celauro, C. (2012), "Improved Traffic Signal Detection and Classification via Image Processing Algorithms," *Siiv-5th International Congress - Sustainability of Road Infrastructures 2012*, **53**, 811-821.

Cyganek, B. (2007), "Real-time detection of the triangular and rectangular shape road signs," *Advanced Concepts for Intelligent Vision Systems, Proceedings*, **4678**, 744-755.

delaEscalera, A., Moreno, L. E., Salichs, M. A. and Armingol, J. M. (1997), "Road traffic sign detection and classification," *Ieee Transactions on Industrial Electronics*, **44**(6), 848-859.

Duda, R. O. and Hart, P. E. (1972), "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the Association for Computing Machinery* **15**(1), 11-15.

Fisher, R., Perkins, S., Walker, A. and Wolfart, E. (2000), "Hough Transform," Hypermedia Image Processing Reference (the HIPR Copyright), http://www.cse.iitd.ac.in/~pkalra/csl783/GHT-notes.pdf

Gao, D. S. and Zhou, J. (2000), "Car license plates detection from complex scene," *2000 5th International Conference on Signal Processing Proceedings, Vols I-Iii*, 1409-1414.

Garcia-Garrido, M.A., Sotelo, M.A., and Martin-Gorostiza, E. (2006), "Fast traffic sign detection and recognition under changing lighting conditions," *In Proc. ITSC*, 811–816.

Garlipp, T. and Muller, C. H. (2006), "Detection of linear and circular shapes in image analysis," *Computational Statistics & Data Analysis*, **51**(3), 1479-1490.

Ginkel, M., Luengo Hendricks, C. L. and Vliet, L. J. (2004), "A short introduction to the Radon and Hough transforms and how they relate to each other," Number QI-2004-01 in the Quantitative Imaging Group Technical Report Series.

Gonzalez, R. C. and Woods, R. E. (2008), "*Digital Image Processing*," Prentice Hall.

Gunduz, H., Kaplan, S., Gunal, S. and Akinlar, C. (2013), "Circular Traffic Sign Recognition empowered by Circle Detection Algorithm," *2013 21st Signal Processing and Communications Applications Conference (Siu)*.

He, J. P. and Ma, Y. (2009), "Triangle Detection Based on Windowed Hough Transform," *Proceedings of 2009 International Conference on Wavelet Analysis and Pattern Recognition*, 95-100.

Hough, P.V.C. (1962), "Method and means for recognizing complex patterns," US patent nr. 3069654.

Jung, C. R. and Schramm, R. (2004), "Rectangle detection based on a windowed Hough Transform," *Xvii Brazilian Symposium on Computer Graphics and Image Processing, Proceedings*, 113-120.

Khan, J. F., Bhuiyan, S. M. A. and Adhami, R. R. (2011), "Image Segmentation and Shape Analysis for Road-Sign Detection," *Ieee Transactions on Intelligent Transportation Systems*, **12**(1), 83-96.

Kiryati, N., Eldar, Y. and Bruckstein, A. M. (1991), "A Probabilistic Hough Transform," *Pattern Recognition*, **24**(4), 303-316.

Lagunovsky, D. and Ablameyko, S. (1999), "Straight-line-based primitive extraction in grey-scale object recognition," *Pattern Recognition Letters*, **20**(10), 1005-1014.

Liu, H. and Wang, Z. (2014), "PLDD: Point-lines distance distribution for detection of arbitrary triangles, regular polygons and circles," *Journal of Visual Communication and Image Representation*, **25**, 273-284.

Liu, Y. X., Goto, S. and Ikenaga, T. (2006a), "An MRF Model Based Algorithm of Triangular Shape Object Detection in Color Images," *International Journal of Information Technology*, **12**(2).

Liu, Y. X., Ikenaga, T. and Goto, S. (2006b), "A novel approach of rectangular shape object detection in color images based on an MRF model," *Proceedings of the Fifth IEEE International Conference on Cognitive Informatics, Vols 1 and 2*, 386-393.

Llorens, D., Marzal, A., Palazon, V. and Vilar, J. M. (2005), "Car license plates extraction and recognition based on connected components analysis and HMM decoding," *Pattern Recognition and Image Analysis, Pt 1, Proceedings*, **3522**, 571-578.

Maldonado-Bascon, S., Lafuente-Arroyo, S., Gil-Jimenez, P., Gomez-Moreno, H. and Lopez-Ferreras, F. (2007), "Road-sign detection and recognition based on support vector machines," *Ieee Transactions on Intelligent Transportation Systems*, **8**(2), 264-278.

Moomivand E., Abolfazli E. (2011), "A Modified Structural Method for Shape Recognition," *IEEE Symposium on Industrial Electronics and Applications* (ISIEA2011), September 25-28, Langkawi, Malaysia.

Moon, H., Chellappa, R. and Rosenfeld, A. (2002), "Performance analysis of a simple vehicle detection algorithm," *Image and Vision Computing*, **20**(1), 1-13.

Park, D. C., Huong, V. T. L., Woo, D. M. and Lee, Y. (2009), "Extraction of Rectangular Boundaries from Aerial Image Data," *2009 International Conference on Computer Engineering and Technology, Vol Ii, Proceedings*, 473-477.

Ruta, A., Li, Y. M. and Liu, X. H. (2010), "Real-time traffic sign recognition from video by class-specific discriminative features," *Pattern Recognition*, **43**(1), 416-430.

Topal, C. and Akinlar, C. (2012), "Edge Drawing: A combined real-time edge and segment detector," *Journal of Visual Communication and Image Representation*, **23**(6), 862-872.

Xu D., Tang Z., and Yan X. (2012), "Real Time Road Sign Detection Based on Rotational Center Voting and Shape Analysis," *Proceedings of 2012 IEEE International Conference on Mechatronics and Automation*, 1972-1977.

Xu, J. F., Li, S. F. and Yu, M. S. (2004), "Car license plate extraction using color and edge information," *Proceedings of the 2004 International Conference on Machine Learning and Cybernetics, Vols 1-7*, 3904-3907.

Xu, S. (2009), "Robust traffic sign shape recognition using geometric matching," *Iet Intelligent Transport Systems*, **3**(1), 10-18.

Yalic H. Y., and Can A. B. (2011), "Automatic Recognition of Traffic Signs in Turkey Roads," *2011 IEEE 19th Signal Processing and Communications Applications Conference* (SIU2011).

Zaklouta, F. and Stanciulescu, B. (2014), "Real-time traffic sign recognition in three stages," *Robotics and Autonomous Systems*, **62**(1), 16-24.

Zhu, Y., Quingzhi, Z. (2011), "Rectangle Detection by the Chain-code Tracing," *Electrical and Control Engineering (*ICECE), 759- 762.