Fast Radial Symmetry Detection for Traffic Sign Recognition

Matias Alejandro Valdenegro Toro

Publisher: Dean Prof. Dr. Wolfgang Heiden

University of Applied Sciences Bonn-Rhein-Sieg,
Department of Computer Science

Sankt Augustin, Germany

August 2015

Technical Report 04-2015

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

ISSN 1869-5272

Uniform Resource Name urn:nbn:de:hbz:1044-opus-15922
URN-Resolver at the German National Library http://nbn-resolving.de

Radial Symmetry Detection for Traffic Sign Recognition
Report on R&D
Corrected edition

Matias Valdenegro*
B-IT Master Studies Autonomous Systems

University of Applied Sciences Bonn-Rhein-Sieg
Fraunhofer Institute for Intelligent Analysis and Information Systems

Advisors: Dr. Stefan Eickeler ** and Prof. Dr. Paul Plöger ††

Originally delivered on January 15, 2014.
Corrected on November 21, 2014.

*matias.valdenegro@smail.inf.h-brs.de
**stefan.eickeler@iais.fraunhoffer.de
††paul.ploeger@h-brs.de

Abstract

Advanced driver assistance systems (ADAS) are technology systems and devices designed as an aid to the driver of a vehicle. One of the critical components of any ADAS is the traffic sign recognition module. For this module to achieve real-time performance, some preprocessing of input images must be done, which consists of a traffic sign detection (TSD) algorithm to reduce the possible hypothesis space. Performance of TSD algorithm is critical.

One of the best algorithms used for TSD is the Radial Symmetry Detector (RSD), which can detect both Circular [7] and Polygonal traffic signs [5]. This algorithm runs in real-time on high end personal computers, but computational performance of must be improved in order to be able to run in real-time in embedded computer platforms.

To improve the computational performance of the RSD, we propose a multiscale approach and the removal of a gaussian smoothing filter used in this algorithm. We evaluate the performance on both computation times, detection and false positive rates on a synthetic image dataset and on the german traffic sign detection benchmark [29].

We observed significant speedups compared to the original algorithm. Our Improved Radial Symmetry Detector is up to 5.8 times faster than the original on detecting Circles, up to 3.8 times faster on Triangle detection, 2.9 times faster on Square detection and 2.4 times faster on Octagon detection. All of this measurements were observed with better detection and false positive rates than the original RSD.

When evaluated on the GTSDB, we observed smaller speedups, in the range of 1.6 to 2.3 times faster for Circle and Regular Polygon detection, but for Circle detection we observed a decreased detection rate than the original algorithm, while for Regular Polygon detection we always observed better detection rates. False positive rates were high, in the range of 80% to 90%.

We conclude that our Improved Radial Symmetry Detector is a significant improvement of the Radial Symmetry Detector, both for Circle and Regular polygon detection. We expect that our improved algorithm will lead the way to obtain real-time traffic sign detection and recognition in embedded computer platforms.

Corrections in this edition

- Removed the mention to a non-existing figure in the Radial Symmetry section.
- Equation 34 has been corrected. In the original version, the sum limits were not correct.

Contents

List of Figures

List of Tables

List of Abbreviations

ADAS	Advanced Driver Assistance Systems
DR	Detection Rate
FPR	False Positive Rate
GTSDB	German Traffic Sign Detection Benchmark [29]
NA	N-Angular Image (Equiangular Image)
ROC	Receiver-Operating Characteristic
RPD	Regular Polygon Detector
RSD	Radial Symmetry Detector
TSD	Traffic Sign Detection
TSR	Traffic Sign Recognition

1 Introduction

All developed countries have issues with traffic-related accidents, such as exceeding speed limits, driving under the influence of alcohol, or simply dangers inherently associated with roads and highways, such as other cars, animals on the road and reckless driving.

Traffic accidents produce unnecessary human (and animal) casualties. Many governments aim to reduce this kind of casualties, and different measures are taken: traffic regulations, banning the use of distractive devices (cellphones), restricting the availability of alcohol, etc.

But technology can also be used to save human lives. Many devices have been built with this purpose, and Advanced Driver Assistance Systems (ADAS) are one type of devices [30]. ADAS are designed as an aid to the driver of a vehicle, with the purpose to avoid "dangerous situations".

In theory, a properly designed ADAS can make a vehicle be aware that the driver is under the influence of alcohol and refuse to start, or notice that the driver is drowsy and take actions to wake him or her up. It can tell the driver that he is driving over the speed limit, or provide more sophisticate features like night vision and collision avoidance.

One very desirable component of commercial ADAS is the Traffic Sign Recognition module (TSR) [21]. The purpose of this module is to recognize Traffic Signs in a video feed from a camera installed in the vehicle, then provide this information to other ADAS modules. This kind of information is required for many tasks, such as knowing the current speed limit, guiding the driver through traffic, and general knowledge of potentially dangerous zones.

Common architectural implementations of TSR split this process into two stages: Detection and Recognition [21]. First the detection stage "detects" candidate traffic signs in the input image, and then the recognition stage is run over the candidate traffic signs, where some candidates might not contain traffic signs.

This is done to increase computational performance, since other wise the recognition algorithm would have to be executed over the entire input image, and this could lead to problems such as a very high number of false positives, or a very poor computational performance.

For TSR systems to be commercially usable in automotive grade hardware, they require to run with real-time performance, and have a very low number of false positives [8]. This means running in real-time in automotive grade embedded systems, rather than high end personal computers.

In the present work we develop improvements to the Radial Symmetry Detector [7], a traffic sign detection algorithm, which can run in real-time on high end personal computers. More computational optimizations are needed to achieve real-time performance in automotive grade embedded systems, which usually have low power requirements, and thus lower computational power than personal computers.

2 Related Work

We divided this State of the Art into different subsections, each one describing the state of the art of a different subtopic, but all of this subtopics form part of the bigger picture of Advanced Driver Assistance Systems and the subproblem of Traffic Sign Recognition and Detection.

2.1 Advanced Driver Assistance Systems

Advanced Driver Assistance Systems (ADAS) are technological systems and devices to aid the driver in the task of driving the vehicle to his destination. The help to the driver consists of multiple aids inherent to the driving task, such as to increase awareness of situations on the road, protect the driver from potential harms, and protect others from the driver.

The basic idea of ADAS is to increase road and vehicle safety and to avoid accidents on the road [47]. There are multiple types of ADAS systems that cover a range of different use cases:

(Adaptive) Cruise Control Cruise control is a system that keeps a constant speed of the vehicle, which is set by the driver. Adaptive Cruise Control adds speed control with consideration of the vehicle in front in the same lane, to keep a constant distance to it [40].

Navigation Give information to the driver about vehicle position and route guidance. The classic implementation of this functionality are GPS navigation devices.

Enhanced Information Systems Under this category we consider systems that provide enhanced information about the road environment to the driver, such as night vision,

adverse lighting enhancement, legal speed limits, distance to neighboring vehicles, and blind spot monitoring.

Collision Avoidance This consists of systems that help the driver avoid collisions with other vehicles and/or the environment [55].

Lane Change Assistance and Control Systems that aid the driver to change lanes, and it could also notify the driver if he or she is drifting from the lane markings on the road [56].

Drowsiness Detection Detect if the driver is able to drive without sleeping during the driving process. Also can be considered as detect if the driver is going to fall asleep and take the appropiate protective actions [53].

Autonomous Parking Park the vehicle without human intervention. The driver should be able to indicate where does he want the vehicle to be parked, and the vehicle should perform the process automatically [57].

Autonomous Driving Drive the vehicle without human intervention other than setting the destination. This is a very complex problem, since many real world conditions must be considered, such as traffic, other non-autonomous vehicles, traffic laws and legal speed limits, varying weather and lighting conditions, etc. There are some big advances on this topic triggered by the DARPA Challenge competition such as [54].

ADAS are very complex systems, since they consider a great number of variables that can be measured on the vehicle, and usually are implemented as embedded systems, which also introduces computing power constraints.

One important component is the Traffic Sign Recognition module. Many research has been done about this topic has been done, which is covered in the next section.

2.2 Traffic Sign Recognition

Traffic Sign Recognition (TSR) is the task of recognizing the type and semantic meaning of a traffic sign from an image. For this process the usual implementation is done using Machine Learning and Pattern Recognition algorithms, trained using a suitable set of traffic signs from the target country's traffic signs. This means that the classificator algorithm must be trained with a different set of traffic signs and semantic information for each country or zone.

The recognition process is usually split into 3 stages [21]:

1. **Detection**: Possible Traffic Sign candidates are detected in the input image, and are given to the Recognition stage. This stage is used to reduce the number of possible Traffic Sign candidates that the Recognition stage must classify.

2. **Tracking**: The purpose of this stage is to track Traffic Signs over a certain number of video frames. This information can be used to aid the Detection stage of the process.

3. **Recognition**: The final stage recognizes and classifies Traffic Signs using a suitable Machine Learning or Pattern Recognition algorithm.

Much research has been devoted to the problem of Traffic Sign Recognition [21]. There are several conditions that make this problem harder:

- Lighting conditions vary (day, night, dawn, dusk, etc), as well as the weather conditions such as rain, fog and snow.
- Traffic Signs can be Occluded by Trees, Cars, Trucks and Pedestrians.
- People often vandalize Traffic Signs, and they degrade naturally due to sunlight and paint quality.
- Traffic Signs are usually perpendicular to the road, but the orientation can vary, as well as size and the height of the pole where it is placed.
- Since ADAS are installed in moving vehicles, camera frames could contain motion blur.

Since the color of Traffic Signs is standarized [18], many TSR techniques are based on color information, while the competing technique is to use shape information to detect Traffic Signs.

The Radial Symmetry Detector (RSD) [7] and Regular Polygon Detector (RPD) [5] have been developed for Traffic Sign Detection. Barnes at al use cross correlation for TSR and classification [7] [8].

A Hardware platforms for real-time TSR is presented in [48]. The authors use a FPGA to implement a TSR system, with the Hough Transform and color segmentation to detect Circular Red signs but performance is not real-time.

Evaluating the performance of TSR systems is also a problem, [44] proposed a methodology to evaluate performance, based in a metholody in the field of Visual Surveillance,

but the authors did not provide a dataset. The German Traffic Sign Recognition Benchmark (GTSRB) has provided a big dataset of 50000 images for TSR evaluation [49].

A Rectangle Detector which is an Extension of the RSD was presented in [35], tuned specifically for rectangular US speed signs. The authors combined Viola-Jones classification to remove false detections with Linear Discriminant Analysis for recognition/classification to obtain a 98.75% detection rate, 97.5% classification rate and a global recognition rate of 96.25%.

SIFT and SURF have been used for Traffic Sign Recognition , in conjunction with a 4-stage System Architecture for robust Circular Traffic Sign Recognition. Höferlin & Zimmermann [31] introduce the refinement stage which refines location estimates to aid the classification process. For Detection they use the RSD and SIFT, Tracking is done with the Kalman Filter and classification with a Multilayer Neural Network over SIFT and SURF features. They obtained 96.4% recognition rate.

In a competition done using the GTSRB [49], with respect to correct classification rate (CCR), Committee of Convolution Neural Networks (CNN) [13] were the best algorithm, with a 99.46% CCR. Second came a human classificator with average CCR of 98.84% (with a max of 99.22%). Third came Multiscale Convolutional Neural Networks [46] with 98.31%, fourth came Random Forests [62] with 96.14% and 5th to 7th came Linear Discriminant Analysis [32] with 95.68%, 93.18% and 92.34%. Linear Discriminant Analysis can run in real-time, but Convolutional Neural Networks cannot.

2.3 Traffic Sign Detection

Performance of many TSR systems depends on the ability of correctly detecting Traffic Signs in a image, with the purpose of posterior recognition and/or classification. Then for Traffic Sign Detection (TSD) many algorithms and techniques have been developed.

Most detection techniques can be divided into 2 types [43]:

Color Segmentation

Detects Traffic Signs according to their color. Since Traffic Sign color is standarized [18], this kind of segmentation is possible, but color perception in a camera is not lighting invariant, does not account for sign degradation due to weather and sunlight, and can fail when there is occlusion on the signs.

Shape Segmentation

Detects Traffic Signs according to their shape, by using a specific shape detector.

This is possible due to the standarization of Traffic Sign shapes in most countries [18]. This kind of methods are based on gradient information, and thus are lighting invariant, but they are sensitive to noise in the image.

Fang & Chen [19] used 2 Neural Networks to extract color and shape features from an image by using fuzzy logic sets, then use a Kalman Filter to track the detected signs over several frames. Their method works with Circular and Polygonal Traffic Signs, and color segments the image with the HSI color space, since the hue value is invariant to lighting. Computation time is in the range of 1 to 2 seconds for a 320x240 image on a standard Pentium 4 PC (1 to 0.5 Hz).

Barnes et al adapted the Radial Symmetry Detector and Regular Polygon Detector for Traffic Sign Detection and reported their results in [7], [4], [5] and [8]. More about this detector is covered on the Circle and Polygon detection sections.

Bahlmann et al [2] used Haar wavelet computed by Adaboost training with color and shape information, without the need of manually tuning thresholds. With their system, they reduce the false positive rate by an order of magnitude (0.3% to 0.03%), which is quite an improvement. Computation times allow for a processing frequency of 10 Hz.

Garcia et al [22] used an Hough-based algorithm to detect Circles and Lines that form Triangles, with a processing speed of 5 to 50 Hz. To achieve such high processing rate with the Hough Transform, they preprocess contour information and reject contourns that do not match a Traffic Sign, such as with aspect ratio and closedness.

Edge Orientation Histogram has also been used to detect Traffic Signs [1], where features can be learned and then used to compare against computed features in the image, yielding detections. 80% to 90% detection rate can be achieved with 0.1% false positive rate, with a processing rate of 7.5 Hz.

Maldonado-Bascon et al [41] used Support Vector Machines (SVM) for detection, by doing color segmentation in HSI color space and then classify shape blobs with a linear SVM, to finally recognize them with a Gaussian Kernel SVM. Their method can detect Circular, Rectangular, Triangular and Octagonal Traffic Signs, and has a detection rate of 93.24% and a computation time of 1.77 seconds over 720x576 pixel images (0.56 Hz).

The Bilateral Chinese Transform [10] is a variation of Hough-like algorithms such as the RSD, but only gradient edges that are 180° from each other vote for the middle point

as a center, which has the advantage of not requiring radius information about the shape being detected. Authors report 86% detection rate with 25 false positives over 89 images at 640x480 resolution, with processing time of 30 milliseconds (33 Hz), but in a subsequent paper [9] the same transform takes 0.8 seconds to process 960x1080 images, which is an inconsistency.

The Radial Symmetry Detector has been implemented in a NVIDIA GeForce 9400M GPU [23] on a embedded computer platform with a Atom CPU. Glavtchev et al achieved 88% detection rate at 33 Hz processing rate with this setup, which is closer to the ideal implementation of TSD on embedded platforms.

Chen et al [11] used pure color segmentation in HSV color space, then detected Traffic Signs using Haar wavelets from Adaboost training, from where they achieve a 90% combined recognition rate with 50 millisecond processing time for detection (20 Hz).

A generic way to detect Traffic Signs is to use a generic object detector. Since Traffic Signs have known color and shapes, using a generic shape detector, such as Circle, Rectangle and Triangle Detectors is a good way to obtain high correct detection rates. This approach has the advantage of not being Traffic Sign-specific, so detector research can also be used for other purposes.

2.4 Circle Detection

The most used algorithm for Circle Detection is the Generalized Hough Transform (GHT) [3] and the Circular Hough Transform (CHT) [17] [37] and their variants, which can be used to detect any kind of shape that can be described with an analytical equation $f(\mathbf{x}, \mathbf{a}) = 0$, where $\mathbf{x}$ is a coordinate vector, and $\mathbf{a}$ is a parameter vector.

The Hough Transform (HT) is an algorithm that takes an input image with a target shape to be detected and produces an accumulator array by successive "voting" of parameters of the shape in the parameter space represented by the accumulator array. Then the detection of the shape is performed by finding maxima in the accumulator array, which is usually implemented by thresholding.

The Generalized Hough Transform has exponential computational complexity, in the order of $O(M^m)$, where m is the number of parameters in the parameter vector, and M is the number of values of the parameters. For Circles there are 3 parameters, so the GHT has complexity $O(n^3)$, where n depends on the discretization of the parameter space. The

accumulator array has exponential space complexity as well.

The GHT has a very costly complexity, which translates into very high computation times, and this is the biggest drawback of using this transformation. Much research has been devoted to reducing the computational complexity of the Hough Transform.

Properties of the shapes being detected can be used to reduce the complexity of the HT, for example for Ellipses, in [52] they remove straight lines from the edge set and then use the property that for 2 parallel tangets to an elipse, the middle point between them must be the center of the ellipse, and this center is incremented, so votes must accumulate in ellipse centers.

Randomness can also be used to decrease computation times, as with the Randomized Hough Transform (RHT) [60], which take n random pixels of the image and use the GHT to vote for this pixels in the accumulator array, and then search for the shape in this reduced accumulator array. This increases performance significantly and reduces the dimension of the accumulator array, but results depend on the precision parameter δ and previous information about the shape being detected.

Computation times of the RHT are in the range of several seconds, while the GHT can take several minutes, depending on the shape being detected and the image size. The dimension of the accumulator array for circles and ellipses has been reduced to 2 [61].

Geometric symmetry has algo been used to reduce the computation time of the HT. In [26] Ho & Chen used the symmetry of Circles and Ellipses to construct the symmetric horizontal and vertical axes when scanning the image in scanline order. Then the intersection of these axes define the center of the Circle or Ellipse, and then the parameters (major and minor axis, and orientation) can be determined with a voting algorithm. This method has been improved to use a 1D accumulator array in [24].

An improvement of the RHT is presented in [12], on which Chen & Chung decided not to use an accumulator array, but a distance metric from which they select 4 edge pixels at random and determine if this points are likely part of a circle. Then they gather evidence for the possibility of a circle by iterating on the edges of the circle hypothesis. This algorithm is fast when compared to the RHT, but still in the range of several seconds to detect a Circle.

Many of this algorithms do not run in real time (at least 10 frames per second), and

take several seconds to detect one Circle. But the HT and its variants have some desirable properties, such as noise resistance and robustness to distortions and discontinuities in the shape

Kim & Kim in [36] used the properties of chords in a circle to compute the center of the circle with a 2 dimensional HT, and then use a 1D HT to compute the radius. This algorithm has computational complexity $O(n^3)+O(n_e^4 n^2)$, where n is the number of pixels in the image and n_e is the maximum number of endpoints of the possible chords. The results of this contribution show computation time between 0.3 to 22 seconds.

Rad et al in [45] presented the Fast Circle Detection (FCD) algorithm that by finding all gradient pair vectors and use them to detect the center and radius of the circle. Two conditions that can be evaluated fast are required to find the gradient pair vectors. The authors report computation times from 0.3 to 1.6 seconds, which is a big improvement from the current state of the art up to this point.

Loy & Zelinsky in [39] proposed the Fast Radial Symmetry algorithm to detect points of interest in images, and then use this technique to build the Radial Symmetry Detector (RSD) [7] where the gradient direction is used to vote for a circle center at a distance r pixels away. If r matches the circle radius, then the votes accumulate at the center and can be detected by simple thresholding of the accumulator array.

The biggest advantage of the Radial Symmetry Detector is that is able to run in real-time, since it has computationa complexity $O(kp)$, where p is the number of pixels of the image, and k is the size of the set if different circle radius values for detection. Barnes et al report in [7] that a computation time of 13.2 milliseconds on a 320x240 image was achieved.

The biggest disadvantage of the Radial Symmetry Detector is that it requires information about the radius of the circles being detected (in pixels), and that performance is sensitive to the gradient magnitude threshold [4], which means that noise will reduce the performance of the detector.

Another deficit of the RSD is that it was tested on a high end computer platform (3.4 GHz Intel Xeon) [8], and thus is not able to run in real-time on embedded or low power platforms, such as the ones used to implement ADAS. More computational optimizations are required for embedded implementations. Also the number of false positives is high (0.3 - 0.8 false positive rate), which is discouraging for a commercial vehicle implementation.

More recent research has been done on Circle Detection without using the HT, such as [15] where a Swarm Intelligence algorithm (Adaptive Bacterial Foraging) is used to detect circles, with computation times in the range of 300 to 1000 milliseconds. Also Learning Automata [14] has been applied to detect multiple circles, which shows a considerable improvement over previous automata and genetic algorithm techniques, with computation time in the order of 100 to 200 milliseconds.

In [28] Houben developed a image preprocessing technique using color information to improve the detection performance of several traffic sign detectors such as the RSD.

2.5 Regular Polygon Detection

There are not many generic regular polygon detectors in the literature. Detector algorithms are generally dedicated to one kind of regular polygons, commonly Triangles, Squares, Octagons, and some detectors specialize on Rectangle detection.

The HT can also be used to detect polygons in general. In [16] Davies uses the HT to vote for the polygon center, but the number of peaks is $O(n^3)$ where n is the number of sides of the polygon.

Also using the HT polygon sides can be detected as Lines and an algorithm such as [33] can be used to detect polygons, but this takes time $O((n+m)^4)$ and space $O((n+m)^2)$, where n is the number of lines and m is the number of line intersections. This method can run in real-time for small number of lines (6 from the publication), but for for more than 30 it will not be able to achieve real-time performance.

The Radial Symmetry Detector has been extended to a Regular Polygon Detector (RPD) in [38] by constructing a line of votes at r distance from the voting gradient element that is perpendicular to the gradient. The RSD can be considered a specific case of the RPD when the number of sides $n \to \infty$.

The RPD runs in real-time (20 Hz) in a high end Intel processor, but it has a moderately high number of false positives. As with the RSD, computational performance is dependent on the amount of noise in the image.

Square Detection can be considered a special case of Rectangle Detection, [34] used a Windowed Hough Transform and geometric properties of lines of a Rectangle, specifically

that sides of a rectangle are perpendicular to each other, so they form a known pattern in the maximums of the accumulator array. This method does not run in real-time and takes several seconds to compute.

Barnes et al proposed a new robust regular polygon detector [6] that used a 5 dimensional space to represent polygons, where they compress this space to a 3 dimensional space where search can be performed. They use a maximum likelihood approach to recover the 2 lost dimensions. This algorithm runs with a frame time of 50 milliseconds on 320x240 images.

In [5], Barnes & Loy proposed more computational optimizations to the RPD. By removing the equiangular image computation, the RPD can run at more than 20 Hz in a "standard PC", but this increases the number of false positives, but the authors use a multiple frame information to reduce the false positive rate.

Non-HT techniques are also present in the literature, such as [27] where a fuzzy shell clustering algorithm is used to detect rectangles, and this method is improved in [51] where a Neural Network and competitive learning is used to detect Rectangles.

2.6 Discussion

We can see that traffic sign recognition is a very well studied problem, and many recognition and classification algorithms exist. Many of this algorithms do not run in real-time, or they do only when applied to small images.

The optimal way to maximize performance of the recognition algorithm is to run it only in the image section where the traffic sign is located.

Since the detection algorithm must always be run in the full input image, the performance of it is critical. One of the fastest detector algorithm is the Radial Symmetry Detector [7] and its generalization for regular polygons [5]. Both of them can run in real-time by taking some assumptions on sign size and orientation, but only on high end processors oriented for servers (Intel Xeon).

So clearly there is a need to optimize the Radial Symmetry Detector with respect to computational performance. Also this algorithm could have other uses, since its structure is not specific to traffic signs only, and can be used to detect circles and regular polygons, with many more applications than just traffic sign detection.

There is room for improvement of the Radial Symmetry Detector. The use of the gaussian filter in the Circle Detector is necessary to resist noise, but it creates a performance problem, since without doubt is the most expensive part of the algorithm.

3 Background

3.1 Radial Symmetry

Symmetry is a property of mathematical objects. It can be expressed as invariance to transformations, for example, the equation $y = x^2$ is invariant if we replace x by $-x$, which can be considered as a transformation that "mirrors" through the X axis. Symmetry is usually associated to geometric figures, since its very easy to see and understand.

Invariance can be thought as a property of an mathematical object that does not change when a transformation is applied to the object. For example, if we rotate a triangle, the values of the interior angles do not change, and thus they are invariant to rotation. The same for area and length of the sides. But the position of the triangle in some world coordinate frame will change when the rotation is applied, and thus is not invariant to rotation.

Giving a definition of Radial Symmetry is not trivial. For a radially symmetric object, if we cut the object with a plane that intersects the axis of the object, then we get 2 halfs that are mirrored from each other [1]. This property holds for any plane that goes through the axis of the object.

A more formal definition is that a radially symmetric object is constructed radially from a central point (the axis) [2].

All radially symmetric objects have a symmetry axis that usually is the center of the object. Two geometry objects that are relevant for this report are Circles and Regular Polygons, specifically Triangles, Squares and Octagons.

3.2 Image Gradient

The gradient is a generalization of the concept of derivative to functions of multiple variables [50]. Given a function $f(x_1, x_2, ..., x_n) : R^n \rightarrow R$ then the gradient of f is denoted

[1] As defined by the Collins English Dictionary at `http://www.collinsdictionary.com/dictionary/english/radial-symmetry`

[2] As defined by the American Heritage Dictionary of the English Language at `http://www.ahdictionary.com/word/search.html?q=radial+symmetry`

as ∇f and is defined as $\nabla f : R^n \to R^n$:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n} \right)$$

So the gradient of a multivariable function is a vector with the same dimensionality as the number of variables of the function. Some interesting properties of the gradient are that the gradient vector points to the direction of biggest growth of the function.

For 2D images, we can consider a grayscale image as a function $I : \{0, W\} \times \{0, H\} \to D$, where $D = \{0, 1, 2, 3, ..., 255\}$ [3]. The gradient of this image can be computed by using finite differences approximation for the derivative:

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

When h is small, this is a good approximation of the derivative $f'(x)$. When applied to the grayscale pixel values, the gradient $(\mathbf{G}_x, \mathbf{G}_y)$ in the x and y directions can be computed by using the sobel operator [25], where the image I is represented as a $w \times h$ matrix:

$$\mathbf{G}_x = S_x * I \qquad \mathbf{G}_y = S_y * I \tag{1}$$

Where $*$ represents the convolution operation and the convolution kernels K_x and K_y are given by:

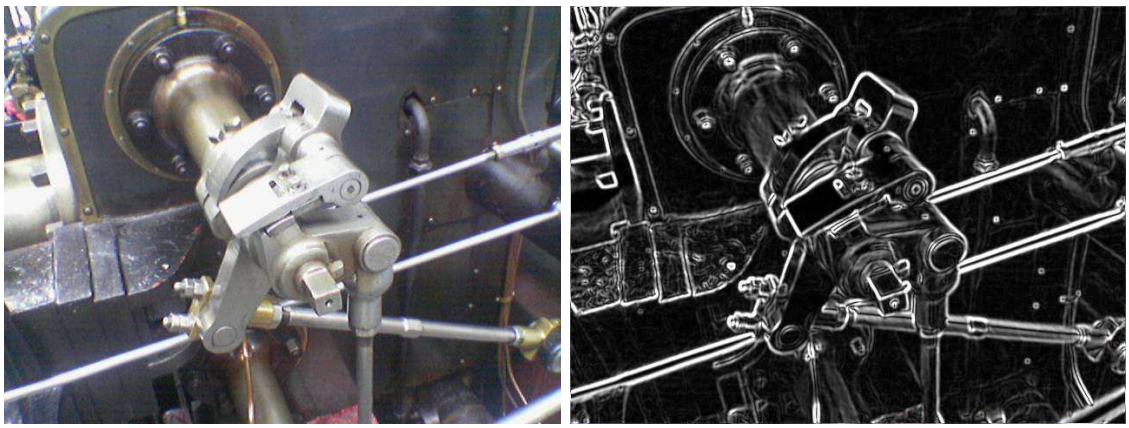
$$S_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \qquad S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{2}$$

After convolving the image I with the sobel operator kernels S_x and S_y, 2 new images of the same size as the original one are obtained, each representing the x and y components of the gradient. The angle-magnitude representation of the gradient can be computed as:

$$|\mathbf{G}| = \sqrt{G_x^2 + G_y^2} \qquad \angle G = \text{atan2}(G_y, G_x) \tag{3}$$

An example result of using the Sobel Operator in a image is shown in Figure 1. The gradient of a grayscale image is usually used to extract edges from the image, but other possible interpretations can be used to extract other type of information from the image.

[3] This is the most common image representation in software, using unsigned byte channels, which have a range from 0 to 255.



(a) Color Image (b) Gradient Magnitude

Figure 1: Example result from the Sobel Operator. **Source**: Wikimedia Commons, Files `Valve_original_(1).PNG and Valve_sobel_(3).PNG`

3.3 Gaussian Smoothing

The purpose of smoothing is to remove noise from the image by blurring the image with a kernel filter that is made with the gaussian function:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}} \tag{4}$$

Since images are 2D, the gaussian smoothing kernel is given by a 2D gaussian:

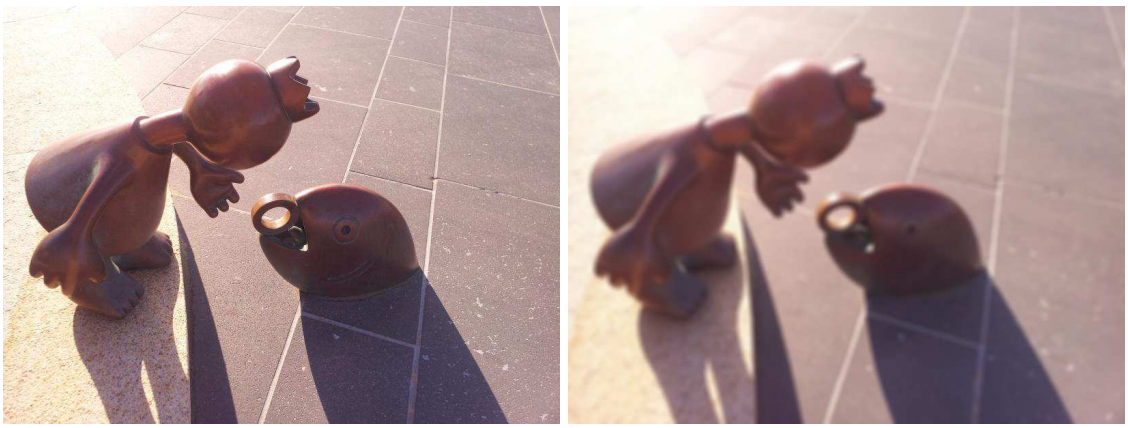
$$G_\sigma(x, y) = f(x)f(y) = \frac{1}{2\pi\sigma} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{5}$$

To construct a gaussian smoothing kernel, we must consider x as the horizontal distance from the kernel center to the element being computed, and y as the vertical distance. The smoothed S image from a given image I is given by:

$$S = G_\sigma * I \tag{6}$$

Where $*$ is the convolution operaton. Gaussian kernels are preferred to other types of smoothing kernels because they are rotationally invariant, which in practice means that the smoothed image will not show the well known ringing effect of mean filtering.

The parameter σ is the standard deviation of the associated normal distribution and controls how far pixels from the center affect the filtered value. Bigger values of σ will make the output image more blurred and "out of focus", while small values of σ will reduce the effect of the filter. This can be seen in Figure 2.



(a) Original Image

(b) 5×5 Gaussian Kernel with $\sigma = 5$

(c) 10×10 Gaussian Kernel with $\sigma = 10$

(d) 20×20 Gaussian Kernel with $\sigma = 20$

Figure 2: Gaussian Smoothing Example

Another advantage of gaussian smoothing is that it can be implemented as a separable filter:

$$S = G(y)_\sigma * (G(x)_\sigma * I) \tag{7}$$

Where $G(y)_\sigma$ is a 1D gaussian kernel in the y direction, and $G(x)_\sigma$ is a 1D gaussian kernel in the x direction. In practice, convolving an image of size $w \times h$ with a filter of size $k \times k$ will take time in the order of $O(whk^2)$, while the separable filter will take time in the order of $O(whk)$, which is clearly an improvement.

3.4 Hough Transform

The Hough Transform is a object detection algorithm that contains the theoretical background of the Radial Symmetry Detector. It works by using a voting scheme, where voting is done by using some properties or equations of the object being detected, which are col-

lected in a array called the accumulator array.

Then the most likely object position and parameters will obtain the biggest number of votes in the accumulator array. Then the object can be detected by finding the maxima in the accumulator array. This also makes the Hough Transform resistant to noise in the original image.

Initially the Hough Transform was developed to detect lines in Images [17]. For this the polar representation of a line is used:

$$r = x\cos\theta + y\sin\theta \tag{8}$$

Given a accumulator array A, an Image indexed by (r, θ) (hough space), usually discretized in some range such that $r \geq 0$ and $\theta \in [0, \pi]$. Then for each point (x, y) of the line (detected by some method, such as edge detection) and for each value in the discretized space of θ the following computation is performed:

$$\begin{aligned} r &= x\cos\theta + y\sin\theta \\ A(r,\theta) &= A(r,\theta) + 1 \end{aligned} \tag{9}$$

Then for each point in the line, a sinusoidal curve is drawn in the accumulator array in hough space. For all points (x, y) that belong to the same line, the corresponding sinusoidal curves will intersect in a point (r, θ) that will correspond to the parameters of the line in polar coordinates.

This means the "votes" will accumulate in the most likely parameters (r, θ) of the line, and thus by finding the maxima in the accumulator array, the line is detected. An example Image and his Accumulator Array is shown in Figure 3. In this image, the peaks in the accumulator array on the right are clearly visible.

The Hough Transform has also been generalized to detect circles [61] and other types of shapes [3].

3.5 Regular Polygons

Regular polygons are polygons that have the following properties:

- All sides have the same length.
- All interior angles have the same value.

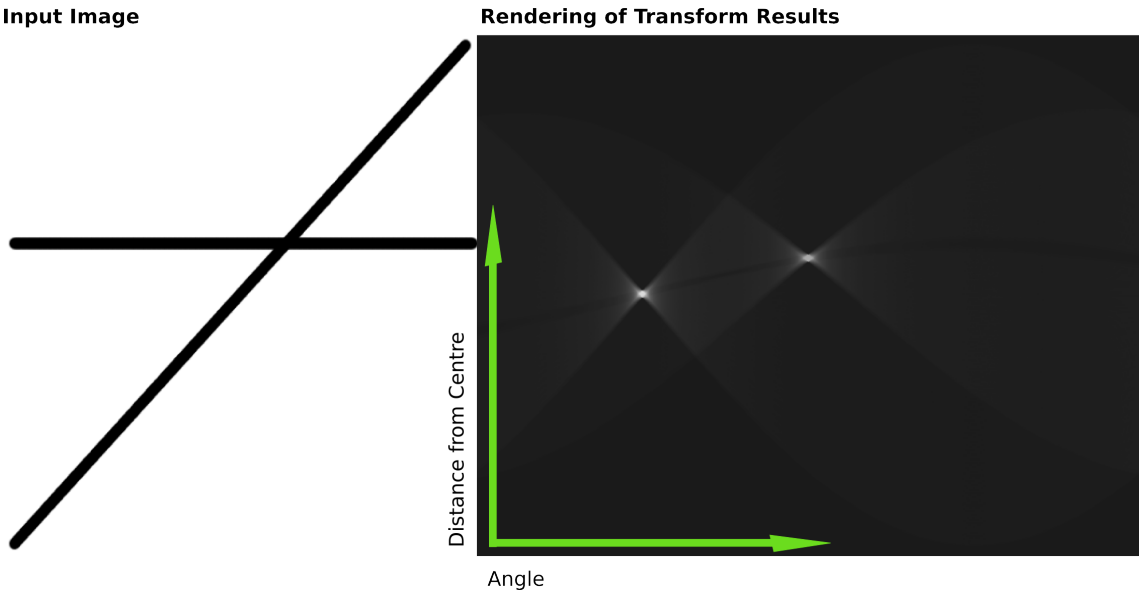


Figure 3: Line (left) and his Accumulator Array (right) after applying the Hough Transform. **Source**: Wikimedia Commons, File `Hough-example-result-en.png`

- All exterior angles have the same value.

Defining what is the radius of a regular polygon is not trivial, and this factor is very important when using radial symmetry detectors. From geometry, there are 2 possible regular polygon radii:

Incircle Is the circle that is inside (inscribed) the polygon, and is tangent to each and every side. The radius of this circle is called the apothem [58].

Circumcircle Is the circle that is outside (circumscribed) the polygon, and touches each and every vertex. The radius of this circle is called the Circumradius or simply the radius of the polygon.

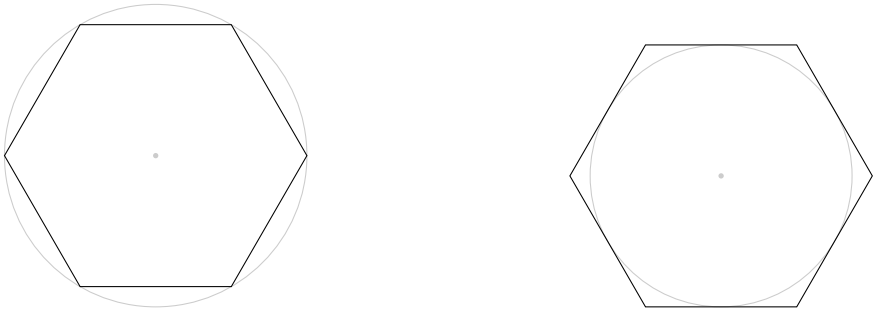
Both Incircle and Circumcircle of a Hexagon can be seen in Figure 4.

Given the number of sides of the regular polygon n and the apothem r, then the length of the polygon's sides l is given by:

$$l = r \tan\left(\frac{\pi}{n}\right) \tag{10}$$

And given the Circumradius R then the apothem r is given by:

$$r = R \cos\left(\frac{\pi}{n}\right) \tag{11}$$



(a) Circumcircle of a Hexagon (b) Incircle of a Hexagon

Figure 4: Circumcircle and Incircle of a Regular Polygon

Data: Number of Sides n and Circumradius R
Result: Regular Polygon Vertices
$\alpha \leftarrow \frac{2\pi}{n}$;
$l \leftarrow \emptyset$;
for $i \leftarrow 0$ **to** n **do**
  $x \leftarrow R\cos(i\alpha)$;
  $y \leftarrow R\sin(i\alpha)$;
  $l \leftarrow l \cup (x, y)$;
end

Figure 5: Regular Polygon Generation Algorithm

To generate the vertices of a n-sided polygon, the algorithm on Figure 5 can be used. This algorithm is used during experimentats to generate regular polygons and paint them in images.

3.6 Traffic Signs

Traffic Signs are information signs posted at the side or above streets, roads and highways with the intention of giving the driver about driving regulations in the specific zone where they are placed.

The design of Traffic signs is country specific, but in general they share some common characteristics:

- They are placed in posts at sides or above the road.
- They are shaped in common geometrical shapes, such as circles, triangles, squares, polygons, rectangles, etc.

- Their color and shape are standarized, but this is standard is different in each country.
- The language and interpretation are country specific.

There are two basic types of traffic signs:

- Ideogram-based Traffic Signs: An ideogram is a graphical symbol that is used to represent an idea or concept [59], so a traffic sign that contains an ideogram is the one that contains a symbol that represents the conveyed message. For example, a Traffic Sign with the symbol of an animal means that there is danger of this kind of animal in the road. Some signs of this kind can be seen in Figure 6.
- Text-based Traffic Signs: They contain text with the information, usually used when there is no other way to convey the message with a symbol, such as speed limits, direction and distances to nearby landmarks. This type of traffic signs are mostly rectangular. Some signs of this kind can be seen in Figure 7.

The Vienna Convention on Road Signs and Signals [18] is a multilateral treaty that standarized traffic signs shape, color and symbols, and has been ratified by 62 countries, mostly in Europe and Asia. For traffic signs, it defines that the possible shapes (depending on their use) are:

Circle
: Stop Sign, Priority for oncoming Traffic, Standard Prohibition, Parking and End of Prohibition. Standard Mandatory signs.

Triangle
: Danger Warning Sign, Yield Sign.

Diamond or Square
: Danger Warning Sign, Priority and End Priority

Rectangle
: Priority over oncoming Traffic, Special Regulation Signs, Information Signs, Motorway and Highway specific signs, and Temporary signs.

Octagon
: Stop Sign

This information eases the process of traffic sign detection, since it can be reduced to the problem of detecting Circles, Triangles, Squares, Rectangles or Octagons in a image.

Figure 6: Ideogram based Traffic Signs. From the GTSDB [29]

Figure 7: Text based Traffic Signs. From the GTSDB [29]

4 The Radial Symmetry Detector

The Radial Symmetry Detector (RSD) is a detector algorithm based on the symmetry around the radii of circles and regular polygons. It is described in a series of papers by Barnes, Loy and Zelinsky [39] [7] [5] and it can detect circles and regular polygons in images.

A common requisite for both Circle and Regular Polygon RSDs is that the radius of the circle/polygon is known. The basic theoretical working of the RSD relates to the fact that the gradient vector at each point of a circle and polygon points to the center of the circle, as seen in Figure 8.

Then, if votes are cast at r units along the gradient vector for each point in the circle, where r is the radius of the circle. Then votes should accumulate exactly at the center of the circle, and thus the circle could be detected by finding maxima in vote space. This assumption indicates why the radius of the circle is required information to use this detection algorithm.

Since the usual requirement is to detect circles of certain radii, and for computational complexity consideratinos, the authors of the RSD decide to use a set of radii R that contains all radii that are of interest for detection. Then the algorithm will detect all

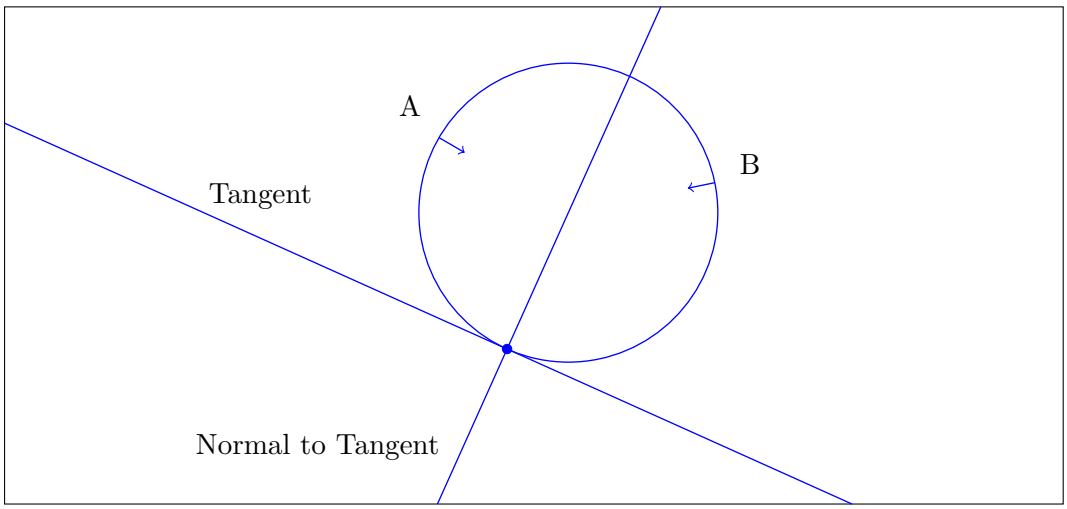


Figure 8: A and B are normal vectors to the circle, and the tangent and normal to the tangent are also shown

radii $r \in R$ circles or regular polygons in the image, and ignore other circles with different radius value.

4.1 Circle Detector

The Circle Detector as described in [39] has the following general structure:

- Compute the gradient of the input image I.
- Compute orientation and magnitude vote images.
- Combine vote images.
- Threshold vote image to obtain detections.

First, the gradient of the input image is computed, and the length of each gradient value is computed for each pixel. If the length of the gradient is smaller than the gradient magnitude threshold, then it is set to $(0, 0)$.

Then, 2 vote images are constructed for each radii $r \in R$ that will be considered for detection. Both vote images are of the same size as the original input image. The first vote image is denominated the orientation image O_r, and has a one channel integer pixel format.

The second vote image is denominated the magnitude image M_r, and has one channel floating point pixel format. Both vote images are initialized with each pixel at value

0. Then for every non-zero gradient $\mathbf{g}(\mathbf{p})$ at position $\mathbf{p} = (x, y)$, two votes are cast, at positions:

$$\begin{aligned} \mathbf{p}_+ &= \mathbf{p} + \text{round}\left(r\frac{\mathbf{g}(\mathbf{p})}{||\mathbf{g}(\mathbf{p})||}\right) \\ \mathbf{p}_- &= \mathbf{p} - \text{round}\left(r\frac{\mathbf{g}(\mathbf{p})}{||\mathbf{g}(\mathbf{p})||}\right) \end{aligned} \tag{12}$$

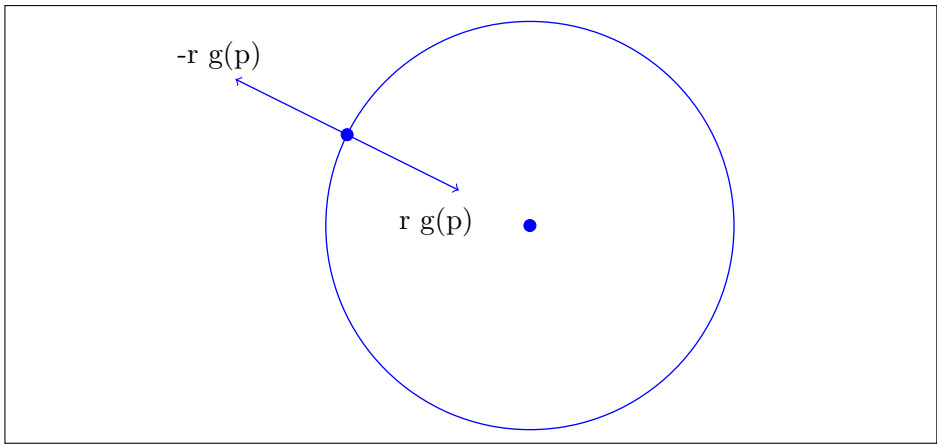


Figure 9: Positive and Negative voting positions

The position $\mathbf{p}_+$ is denominated the positively affected pixel, and position $\mathbf{p}_-$ is denominated the negatively affected pixel [39]. This positions can be seen in Figure 9. Each vote is cast at both vote images, according to:

$$\begin{aligned} O_r(\mathbf{p}_+) &= O_r(\mathbf{p}_+) + 1 \\ O_r(\mathbf{p}_-) &= O_r(\mathbf{p}_-) - 1 \\ M_r(\mathbf{p}_+) &= M_r(\mathbf{p}_+) + ||\mathbf{g}(\mathbf{p})|| \\ M_r(\mathbf{p}_-) &= M_r(\mathbf{p}_-) - ||\mathbf{g}(\mathbf{p})|| \end{aligned} \tag{13}$$

We know that the gradient of a circle always points towards the center, but this also depends on the contrast of the circle in the image. Usually the gradient points from dark to light, and in this case a light circle on a dark background will have their gradient pointing in the right direction (towards the center), but a dark circle on a light background will have their gradient pointing in the opposite direction. To consider both options, the detector votes for both possible center positions (negative and positively affected pixels).

After both vote images have been filled, then both vote images O_r and M_r are combined into a radial symmetry contribution image S_r [39] according to the following formula for each pixel position $\mathbf{p}$:

$$
\begin{aligned}
S_r &= F_r * A_r \\
F_r(\mathbf{p}) &= \frac{M_r(\mathbf{p})}{k_r} \left(\frac{|\tilde{O}_r(\mathbf{p})|}{k_r} \right)^{\alpha} \\
\tilde{O}_r(\mathbf{p}) &= \begin{cases} O_r(\mathbf{p}) & \text{if } O_r(\mathbf{p}) < k_r \\ k_r & \text{otherwise} \end{cases}
\end{aligned}
\tag{14}
$$

Where A_r is a 2D gaussian smoothing kernel of size $r \times r$ and standard deviation $0.5r$. In [39] the authors of the RSD recommended that the gaussian kernel A_r must sum to r, but we found that this property is not strictly necessary and a gaussian that sums to 1 will also work.

The objective of this gaussian smoothing kernel is to smooth the combined image, so that votes are "scattered" and the vote image is smoother, removing or attenuating the noise present in the original image, which is propagated to the vote images. Without doubt, this is the most computationally expensive part of the RSD algorithm.

The factor k_r is used as a normalization factor, so the number of votes across different radius values in the vote image S_r are approximately the same. This normalization factor can be obtained by experimental measurement. The authors of the RSD showed their experimental measurements to come up with the following Normalization value:

$$
k_r = \begin{cases} 8 & \text{if } r = 1 \\ 9.9 & \text{otherwise} \end{cases}
\tag{15}
$$

α is called the radial strictness factor, and according to the authors it "determines how strictly radial the radial symmetry must be for the transform to return a high value" [39].

After computing the radial symmetry contribution image S_r for each radius being considered, is possible to compute a general radial symmetry contribution image S as the average of the contributions between all radii:

$$
S(\mathbf{p}) = \frac{1}{|R|} \sum_{r \in R} S_r(\mathbf{p})
\tag{16}
$$

Finally, maxima must be found in the radial symmetry contribution images. This can be done either on S or S_r, by thresholding the radial symmetry contribution image with a detection threshold D_t, which is the standard method to find maxima on hough-like vote images or accumulator arrays. The value of D_t can be obtained experimentally, and how to do so will be described later in this report.

A algorithmic description of this algorithm is presented in Figure 10. Two results of the execution of the RSD are shown in Figures 11 and 12. The first shows the results on a Synthetic image that contains circles of radius 10 pixels, and the second shows the results from a real traffic sign of radius approximately 25 pixels.

Data: Input image I, Gradient Magnitude Threshold G_t, Detection Threshold D_t, set of radii R
Result: Circle Detections
Compute the gradient G of image I;
for *each pixel* $\boldsymbol{g}(\boldsymbol{p}) \in G$ **do**
  if $||\boldsymbol{g}(\boldsymbol{p})|| < G_t$ **then**
    skip pixel;
  end
  for *each radius value* $r \in R$ **do**
    Compute orientation O_r and magnitude M_r vote images according to equation (13);
    Compute radial symmetry contribution S_r acoording to equation (14);
  end
end
Compute $S = \frac{1}{|R|} \sum_{r \in R} S_r$;
for *each pixel* $s(\boldsymbol{p}) \in S$ **do**
  if $s(\boldsymbol{p}) > D_t$ **then**
    Emit circle detection at pixel position $\mathbf{p}$;
    The radius of the circle can be recovered by checking the maxima in each S_r at position $\mathbf{p}$;
  end
end

Figure 10: Circular Radial Symmetry Detector Algorithm

In [8], Barnes & Zelinsky propose some computational optimizations to use the RSD for traffic sign detection. The modifications relate to the way the vote images are constructed. They propose to use only the orientation vote image O_r and process it as follows:

$$F_r(\mathbf{p}) = \left(\frac{\tilde{O}_r}{k_r} \right)^{\alpha} \tag{17}$$

Where k_r is the normalization constant. Then $F_r(\mathbf{p})$ is smoothed using a gaussian smoothing kernel A_r:

$$S_r(\mathbf{p}) = F_r(\mathbf{p}) * A_r \tag{18}$$

(a) Original Image

(b) Image Gradient

(c) Orientation Vote Image

(d) Magnitude Vote Image

(e) Radial Symmetry Contribution Image

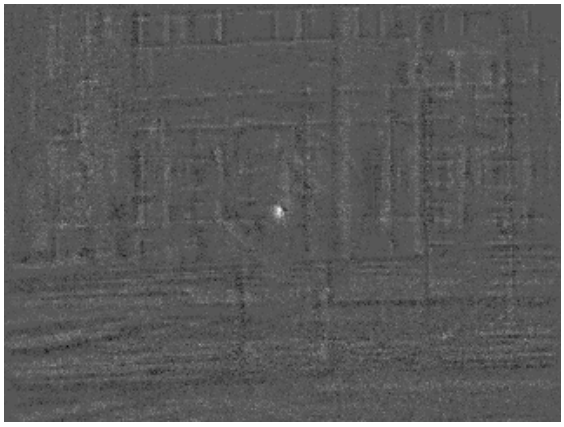
(f) Detected Circles

Figure 11: Results of the Circular RSD on Synthetic Image of $r = 10$, and $D_t = 20$

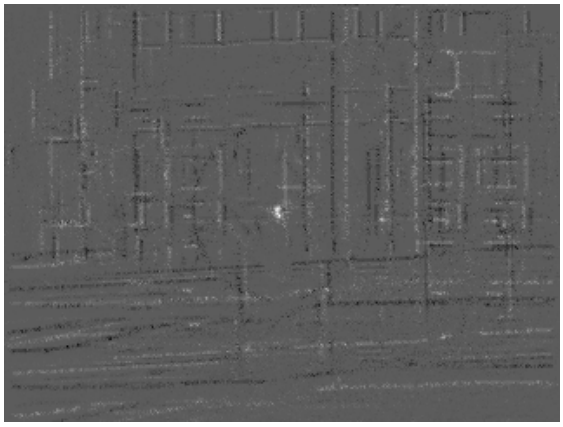
This reduced computation costs increases performance. They also study the sensibility of the detector to the gradient magnitude threshold, since real images are noisy, noise propagates to the vote images and also reduces the computational performance of the

(a) Original Image

(b) Image Gradient



(c) Orientation Vote Image



(d) Magnitude Vote Image

(e) Radial Symmetry Contribution Image

(f) Detected Circles

Figure 12: Results of the Circular RSD on a real Image with $r = 25$, and $D_t = 18$. Traffic Sign Image extracted from the GTSDB [29]

algorithm, since noise will propagate through the sobel operator and it will be amplified, making more noise in the gradient, and less pixels will fail the gradient magnitude thresh-

old test.

Barnes & Zelinsky propose a gradient magnitude threshold of $G_t = \sqrt{11299} \sim 105.05$ [8] that provides a good tradeoff between noise reduction, detector performance, and computational performance.

4.1.1 Parameters

The parameters of the circular RSD are:

Set of radii R The set of radius of the circles to consider for detection, in pixels in the image. This parameter can be experimentally estimated, since for a given camera installed in a vehicle going at a constant velocity, traffic sign images can be capture and statistically processed to obtain the most likely circle radii values. Traffic signs smaller than 5 to 10 pixels cannot be successfully classified (they are too small), and the maximum radius size can be experimentally determined. The GTSDB will later be used to determine limits for the set of radii [29].

Gradient Magnitude Threshold G_t This parameter determines which gradient values will effectively vote for the circle center, and gradients with magnitude smaller than G_t will be ignored. This threshold can be used to skip large sections of the image, with a big computational performance improvement, but also can be used to remove noise and its effect on the vote images. Sensible values of this parameter are 2%-5% of the maximum gradient magnitude in the image [39] [38]. As previously mentioned, an empirical value of $G_t = 105.05$ was proposed in [8].

Radial Strictness α This parameter determines how strict the symmetry must be. The authors recommended a value of $\alpha = 2$ which is "suitable for most applications" [39]. Bigger α values will attenuate non-radial feature points.

Detection Threshold D_t This parameter determines detections, and should have a value that matches the peaks in the vote image S or S_r. This value can be obtained experimentally, and how to do so will be explained later.

4.2 Regular Polygon Detector

The regular polygon detector was initially presented in [38] and consists of a modification of the circular RSD to detect regular polygons. The structure of the algorithm is very similar:

- Compute the gradient of the input image I.

- Compute vote image and equiangular images.
- Combine vote and equiangular images.
- Threshold vote image to obtain detections.

The radius of a regular polygon is difficult to define, and for the polygon RSD, this radius will be considered as the radius of the incircle of the regular polygon (the apothem). The number of sides of the regular polygon is a required information to use this regular polygon detector.

The vote image is computed in a similar way to the circular RSD, but instead of voting on a single point, the votes are laid out in a line perpendicular to the gradient at a distance r units away. Two lines are voted, at the positive and negative directions from the gradient pixel. The length of the line w is given by:

$$w = r \tan\left(\frac{\pi}{n}\right) \tag{19}$$

Where n is the number of sides of the regular polygon being detected. The negatively and positively affected pixel positions are given by Equation (12), and the parametric equation of both positive and negative oriented lines are given by:

$$\begin{aligned} L_+(\mathbf{p}, m) &= \text{round}(\mathbf{p}_+ + m\mathbf{g}_\perp(\mathbf{p})) = \text{round}\left(\mathbf{p} + r\frac{\mathbf{g}(\mathbf{p})}{||\mathbf{g}(\mathbf{p})||} + m\mathbf{g}_\perp(\mathbf{p}\right) \\ L_-(\mathbf{p}, m) &= \text{round}(\mathbf{p}_- - m\mathbf{g}_\perp(\mathbf{p})) = \text{round}\left(\mathbf{p} - r\frac{\mathbf{g}(\mathbf{p})}{||\mathbf{g}(\mathbf{p})||} - m\mathbf{g}_\perp(\mathbf{p}\right) \end{aligned} \tag{20}$$

Where m is the line parameter, and $\mathbf{g}_\perp(\mathbf{p})$ is a unit vector perpendicular to $\mathbf{g}(\mathbf{p})$. If $\mathbf{g}(\mathbf{p}) = (g_x, g_y)$ then:

$$\mathbf{g}_\perp(\mathbf{p}) = (-g_y, g_x) \tag{21}$$

Then, for the positive oriented line, pixels where $m \in [-w, w]$ receive a positive vote (+1), and pixels where $m \in [-2w, -w-1] \cup [w+1, 2w]$ receive a negative vote (-1).

For the negative oriented line, pixels where $m \in [-w, w]$ receive a negative vote (-1), and pixels where $m \in [-2w, -w-1] \cup [w+1, 2w]$ receive a positive vote (+1).

This process is done to construct the vote image O_r, with one channel integer pixel format. The second vote image is called the equiangular vote image B_r. Loy & Barnes in [38] described the equiangular property of the gradient vectors of a regular polygon.

The gradient vectors of a regular polygon on each side are spaced $\frac{360^\circ}{n}$ degrees apart, and when computing the angle of each gradient vector θ, if we compute $n\theta$ and reconstruct the vector $\mathbf{v}$ from its angle-magnitude form then:

$$\mathbf{v}(\mathbf{p}) = (||g(\mathbf{p})|| \cos(n\theta),\ ||g(\mathbf{p})|| \sin(n\theta)) \tag{22}$$

Then for each side of the regular polygon, all vectors $\mathbf{v}_n(\mathbf{p})$ point in the same direction. This is shown in Figure 13. If we compute the vectorial sum $\sum \mathbf{v}_n(\mathbf{p})$, then this sum will have a maximum magnitude, which can be used by the RSD as an additional vote to check for the existence of a regular polygon.

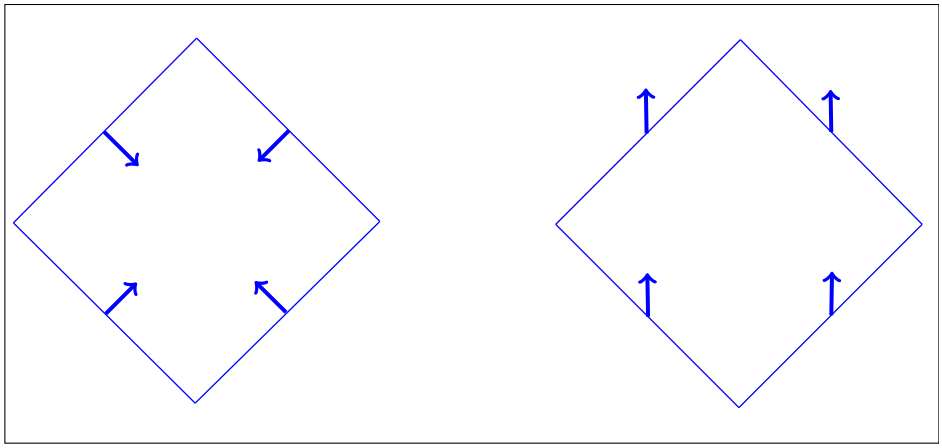


Figure 13: Equiangular Property shown in a Square

The equiangular image B_r is a two channel (x and y) floating point pixel format image and is computed as:

$$B_r(\mathbf{q}) = B_r(\mathbf{q}) + \text{sign}(O_r(\mathbf{q}))\mathbf{v}(\mathbf{p}) \qquad \forall q \in L(p, m) \tag{23}$$

Where $\text{sign}(O_r(\mathbf{q}))$ represents the sign of the vote given to the vote image O_r for the same gradient pixel, so the votes are "projected" with the corresponding negative or positive vote.

The vector value of the equiangular image can also be used to determine the orientation of the regular polygon, but Loy & Barnes mention that if this information is already known, then the equiangular image is not necessary.

Then both vote and equiangular images are combined as:

$$S_r(\mathbf{p}) = \frac{O_r(\mathbf{p})||B_r(\mathbf{p})||}{(2wr)^2} \tag{24}$$

The value $(2wr)^2$ is a normalization factor used to normalize the vote values for different radii.

Data: Input image I, Gradient Magnitude Threshold G_t, Detection Threshold D_t, set of radii R and number of sides n
Result: Circle Detections
Compute the gradient G of image I;
for *each pixel* $\boldsymbol{g}(\boldsymbol{p}) \in G$ **do**
  if $||\boldsymbol{g}(\boldsymbol{p})|| < G_t$ **then**
    skip pixel;
  end
  for *each radius value* $r \in R$ **do**
    Compute vote image O_r according to equation (20);
    Compute equiangular vote image B_r according to equation (23);
    Compute radial symmetry contribution S_r acoording to equation (24);
  end
end
Compute $S = \frac{1}{|R|} \sum_{r \in R} S_r$;
for *each pixel* $s(\boldsymbol{p}) \in S$ **do**
  if $s(\boldsymbol{p}) > D_t$ **then**
    Emit circle detection at pixel position $\mathbf{p}$;
    Radii (apothem) of the regular polygon can be recovered by checking the maxima in each S_r;
    Orientation of the regular polygon can be recovered by checking the orientation of the vector $B_r(\mathbf{p})$
  end
end

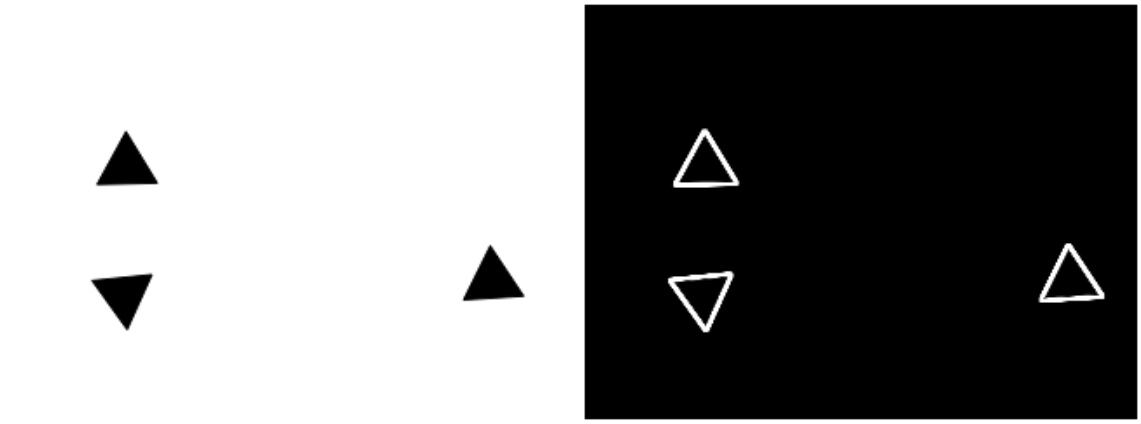
Figure 14: Polygon Radial Symmetry Detector Algorithm

4.2.1 Parameters

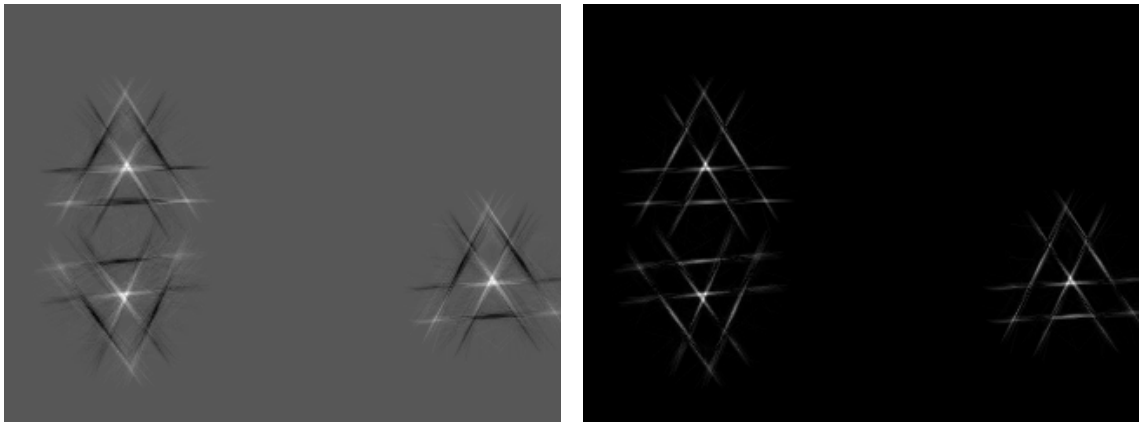
Set of radii R Same as the Circular RSD, but the radius valu of the regular polygon must correspond to the apothem.

Number of sides of regular polygon n Traffic signs have known number of sides, either triangles (3), squares (4) or octagons (8), but the detector will work with any regular polygon, as long as the number of sides is known.

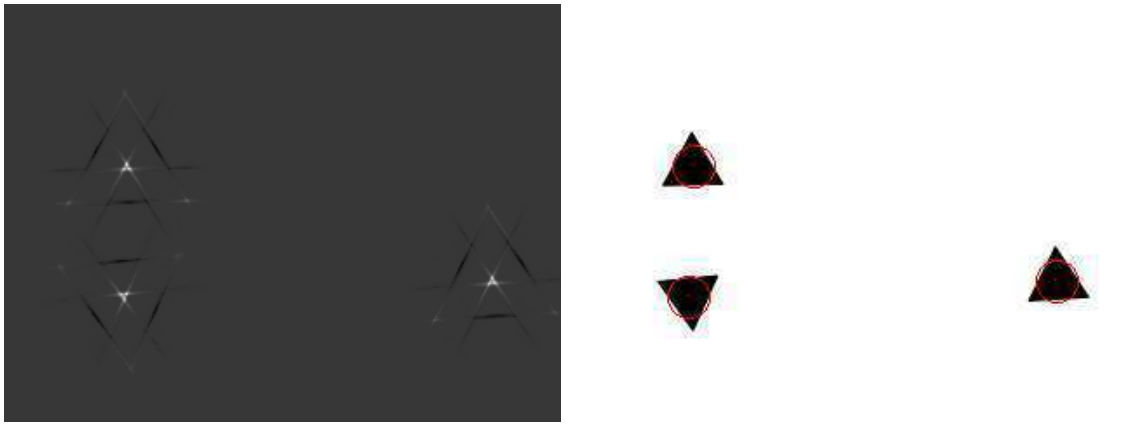
Gradient Magnitude Threshold G_t Same as the Circular RSD.



(a) Original Image (b) Image Gradient



(c) Vote Image (d) Magnitude of the Equiangular Image



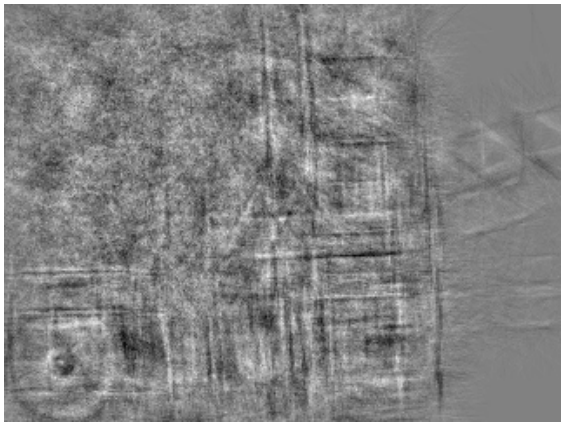
(e) Radial Symmetry Contribution Image (f) Detected Triangles

Figure 15: Results of the Regular Polygon RSD on Synthetic Image of a triangle with $r = 10$, and $D_t = 34$

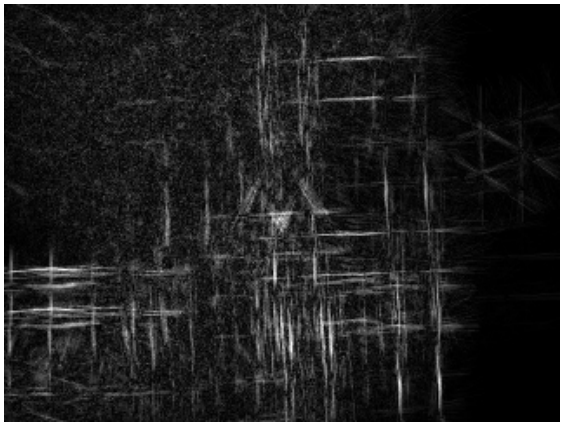
Detection Threshold D_t Same as the Circular RSD.

(a) Original Image

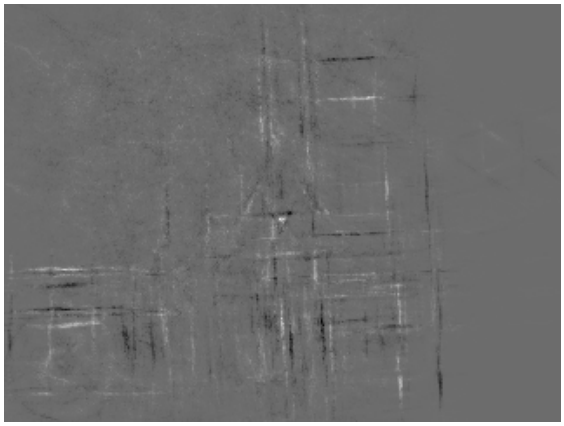
(b) Image Gradient



(c) Vote Image



(d) Magnitude of the Equiangular Image



(e) Radial Symmetry Contribution Image

(f) Detected Triangles

Figure 16: Results of the Regular Polygon RSD on Real Image of a triangular traffic sign with $r = 12$, and $D_t = 8$. Traffic Sign Image extracted from the GTSDB [29]

4.3 Implementation Details

When implementing both the Circular and Regular Polygon RSD, we came up with several implementation details that were not covered in the original publications, so here we will describe them as documentation for the future.

4.3.1 Normalization

The RSD authors proposed in [39] that the orientation vote image can be normalized by dividing by the maximum, which is correct. But they also argued that in their experiments, the value:

$$k_r = \begin{cases} 8 & \text{if } r = 1 \\ 9.9 & \text{otherwise} \end{cases}$$

Can be used to normalize O_r, but in my own experiments, this was not the case. The maximum value of the orientation vote image ranged from 5 to 30, which is out of range from the values given by the RSD authors.

The simple solution was to always compute the maximum value of O_r and use it as the value of k_r.

4.3.2 Detection Merging

Using a threshold to find maxima in the vote image has the disadvantage that unless the threshold is exactly equal to the peak value, multiple detections could happen, since it is very likely that values in the neighborhood of the peak will also have values very close to the peak value.

This is a common problem with Hough Transform based algorithms. A very simple solution is to use a detection merging algorithm, which takes the output list of detections from the detector algorithm and "merges" detections that are "too close" from each other according to some distance merge threshold M_t, and produces a new list of merged detections, that will correspond to the final output of the detector.

A detection object is composed of a center (c_x, c_y) and a radius r, and represents the radius of the detected circle, or the apothem of the detected regular polygon

The detection algorithm that we implemented is very simple. It iterates over all detections, and tries to merge each possible pair of detections. When this is no longer possible,

it stops.

To merge 2 detections $A = \{c_a = (x_a, y_a), r_a\}$ and $A = \{c_b = (x_b, y_b), r_b\}$ into detection $M = \{c_m = (x_m, y_m), r_m\}$, define the following:

$$\begin{aligned} l &= \min\{x_a - r_a,\, x_b - r_b\} \qquad & r &= \max\{x_a + r_a,\, x_b + r_b\} \\ d &= \min\{y_a - r_a,\, y_b - r_b\} \qquad & u &= \max\{y_a + r_a,\, y_b + r_b\} \end{aligned} \tag{25}$$

Then the center c_m and radius r_m of the merged detection are:

$$\begin{aligned} c_m &= \left(\frac{l+r}{2},\, \frac{u+d}{2}\right) \\ r_m &= \max\left\{\frac{r-l}{2},\, \frac{u-d}{2}\right\} \end{aligned} \tag{26}$$

The complete algorithm in pseudocode is presented in Figure 17. This algorithm has computational complexity $O(n^2)$ where $n = |D|$ is the number of detections in the detection list. Currently we are using a value of $M_t = 7$ px.

4.3.3 Threshold Tuning

Finally, the detection threshold D_t must be carefully set to obtain correct detections. If the detection threshold is too low, too many false positives will be detected, and if the detection threshold is too high, then the number of false positives will be low, but the number of correct detections will also be low.

So tuning of an appropiate value for the detection threshold is required, and this is formulated by doing exhaustive search over a subset of the threshold value space, and select the threshold that maximizes the detection rate. This detection rate is defined as:

$$\text{detection rate} = \frac{\text{Number of Correct Detections}}{\text{Number of possible Detections in the Image}} \tag{27}$$

And the false positive rate is defined as:

$$\text{false positive rate} = \frac{\text{Number of Incorrect Detections}}{\text{Number of Detections made by the Detector}} \tag{28}$$

The idea of the threshold tuning algorithm is to execute the detector over some test image set, varying the detection threshold D_t in some defined range of values, and then select the threshold that produces the maximum detection rate (closest to 1.0).

In practice there are many values of the detection threshold that produce a maximum detection rate (or very close to the maximum). To discriminate between them, we chose

Data: List of detections D and distance merging threshold M_t
Result: Merged Detections
ret $\leftarrow \emptyset$;
temp $\leftarrow D$;
if $|D| < 1$ **then**
  return ret;
end
ret = ret $\cup$ temp.firstElement;
for $i \leftarrow 0$ **to** $|temp|$ **do**
  $detectionMerged \leftarrow false$;
  $a = temp_i$;
  for $j \leftarrow 0$ **to** $|D|$ **do**
    $b = ret_j$;
    if $||a.c - b.c|| < M_t$ **then**
      ret_j = merge a and b according to Equation (26);
      $detectionMerged \leftarrow true$;
      break;
    end
  end
  if *detectionMerged is false* **then**
    $ret = ret \cup a$;
  end
end
return ret;

Figure 17: Detection Merging Algorithm

the biggest detection threshold from the subset of thresholds that are % away from the maximum detection rate, and then minimize the false positive rate over this subset.

The Threshold tuning algorithm is shown in Figure 18. Currently we are using a value of $B_t = 0.02$.

Data: Detector d, Number of Images n and best detection rate threshold B_t
Result: Optimal Detection Threshold D_t
Run the detector of a dataset of n images and store the pairs (D_t, detection rate, false positive rate) in list r;
best = max$\{r.detectionRate\}$;
bests = $\{x$such as$|x.detectionRate - best.detectionRate| < B_t\}$;
ret = min$\{bests.falsePositiveRate\}$;

Figure 18: Threshold Tuning Algorithm

5 Improvements to the Radial Symmetry Detector

The main issue with the performance of the Circle Radial Symmetry Detector [39] is the use of the gaussian smoothing filter to "disperse" votes of the F image into the S image. This is done to reduce the influence of noise into the voting process. Since gradient-based algorithms are very sensitive to noise, this is a fundamental step.

Removing the gaussian smoothing filter might greatly increase computational performance, but at the cost of decreasing the noise resistance given by this smoothing.

5.1 General Approach

The general approach (which is credited to Stefan Eickeler) is the following:

- Remove the gaussian smoothing filter.
- Reduce the size of the vote images according to a multiscale approach.
- Use a different thresholding method that considers multiple scales, but this is optional.

5.2 Multiple scale approach

Multiscale approaches are common in Computer Vision and Image Processing [25]. The idea of applying a multiscale approach to the Radial Symmetry Detector is to reduce the size of the vote images, which should reduce the computational costs involved.

Then instead of considering a set of different radii values R, we use a set of scales S, which contains scale factors, which are relative to a base radius value r_{base}. Then the equivalent of the radius values are:

$$R = r_{base} s \qquad \forall s \in S \tag{29}$$

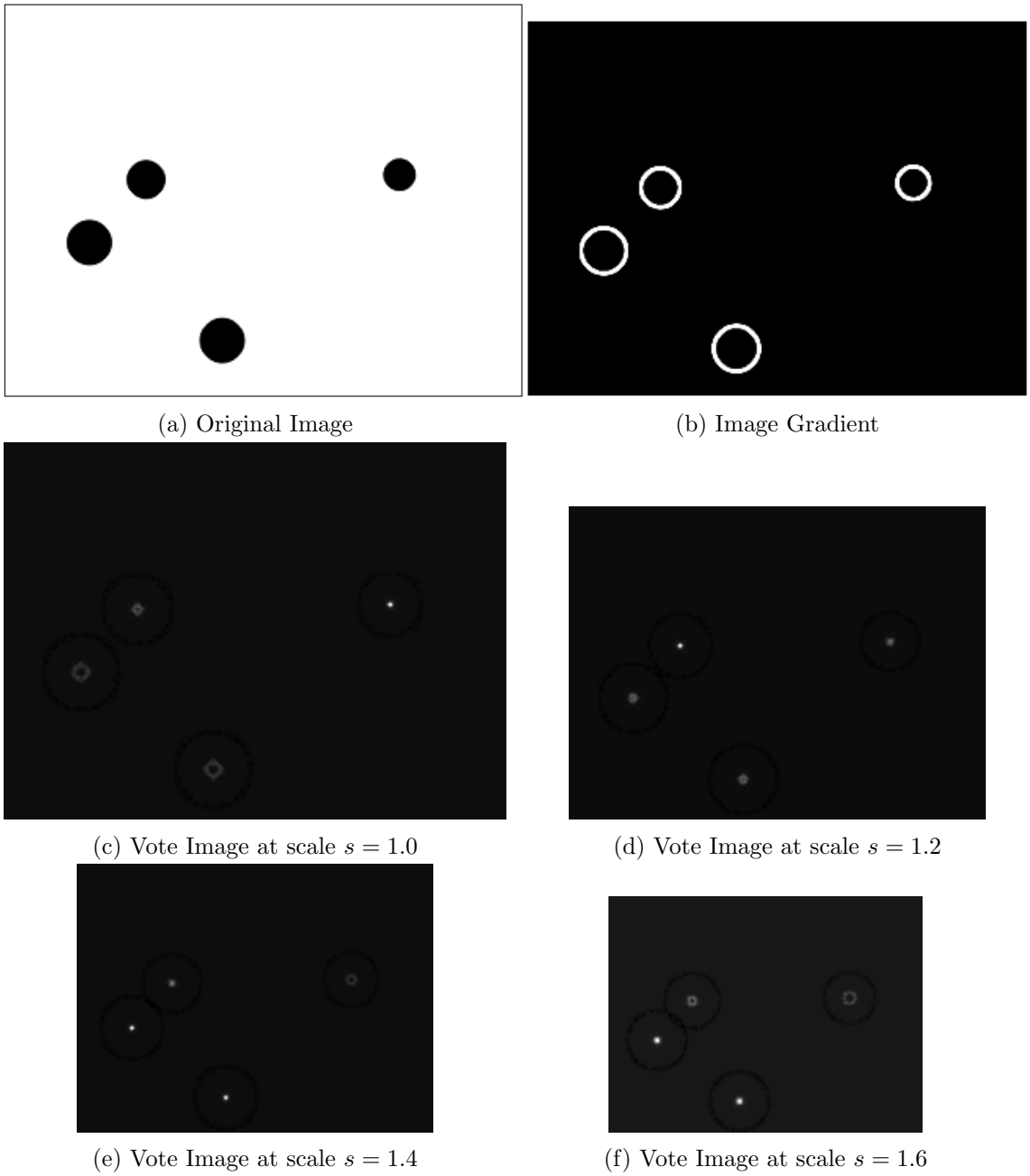
And then at each scale $s \in S$, the corresponding affected pixel positions p_+ and p_- are given by:

$$\begin{aligned} p_+(\mathbf{p}) &= \text{round}\left(\frac{\mathbf{p}}{s} + r_{base}\frac{\mathbf{g}(\mathbf{p})}{||\mathbf{g}(\mathbf{p})||}\right) \\ p_-(\mathbf{p}) &= \text{round}\left(\frac{\mathbf{p}}{s} - r_{base}\frac{\mathbf{g}(\mathbf{p})}{||\mathbf{g}(\mathbf{p})||}\right) \end{aligned} \tag{30}$$

Where again $\mathbf{g}(\mathbf{p})$ is the gradient vector at pixel $\mathbf{p}$. Vote images are computed exactly as in the original RSD algorithm, but using the new vote positions. Given that the

input image I has size $w \times h$ pixels, then for each scale $s \in S$ being considered the size of the vote image is $\lceil \frac{w}{s} \rceil \times \lceil \frac{h}{s} \rceil$. This makes the vote image smaller as the the scale increases.

The number of scales is configurable and there is minimum or maximum value, except for the extreme cases where the vote images might be extremely small.



(a) Original Image

(b) Image Gradient

(c) Vote Image at scale $s = 1.0$

(d) Vote Image at scale $s = 1.2$

(e) Vote Image at scale $s = 1.4$

(f) Vote Image at scale $s = 1.6$

Figure 19: Vote image examples with different scales. Base radius is 10 pixels. Circles have radius of 10, 12, 14 or 16 pixels.

5.3 Thresholding

Now that we have a vote image pyramid according to scale, then we need to devise a new thresholding mechanism that considers this new information. So the following thresholding methodology has been devised by Stefan Eickeler.

For each vote image $v \in V$ generated by the previously mentioned voting method, there is an associated scale v_s. Then for each pixel $p \in v$, we will test the following value with the detection threshold:

$$t(\mathbf{p}) = \frac{v(\mathbf{p})^2}{v_s^2} \tag{31}$$

If $t(\mathbf{p}) > \frac{D_t}{3}$, then we perform a second threshold test by computing a test value t_r over the neighborhood of scale v_s, which means the scale that is immediately lower and bigger than v_s.

We denote the immediately next scale as v_{s+1} and the immediately previous scale v_{s-1}. First, from scale a we have pixel sample position $\mathbf{p}_a$, then we sample scale b at pixel sample position $\mathbf{p}_b$ given by:

$$\mathbf{p}_b = \frac{a}{b}\mathbf{p}_a \tag{32}$$

The value of $\mathbf{p}_b$ should be clamped to the size of the vote image at scale b. Then the second threshold test value for each neighboring scale b is given by:

$$t(\mathbf{p}_b) = \frac{v(\mathbf{p}_b)^2}{b^2} \tag{33}$$

And then we average this values into t_r for the neighboring scales, as well as the sum of squares into t_{rs}:

$$\begin{aligned} t_r(\mathbf{p}) &= \frac{1}{|V|}\sum_{i=s-1}^{s+1} t(\mathbf{p}_{v_i}) \\ t_{rs}(\mathbf{p}) &= \frac{1}{|V|}\sum_{i=1}^{|V|}\left(t(\mathbf{p}_{v_i})\right)^2 - \left(\frac{1}{|V|}\sum_{i=1}^{|V|} t(\mathbf{p}_{v_i})\right)^2 \end{aligned} \tag{34}$$

Where $|V|$ is the number of vote images. We also compute a mean radius value μ_r for the shape:

$$\mu_r = \frac{\sum_{s-1}^{s+1} r_{base}\, v_i\, t(\mathbf{p}_{v_i})}{\sum_{s-1}^{s+1} t(\mathbf{p}_{v_i})} \tag{35}$$

This mean radius value is used in case of a successful detection. A successful detection is emitted if and only if:

$$t_r(\mathbf{p}) > D_t \qquad \textbf{and} \qquad t_{rs}(\mathbf{p}) > S_t \tag{36}$$

Where D_t is the detect threshold and S_t is called the square sum threshold. This detection has center position given by $\mathbf{p}\, v_s$ and radius/apothem given by μ_r.

The full Improved Radial Symmetry Detector algorithm is shown in Figure 20.

Data: Input image I, Gradient Magnitude Threshold G_t, Detection Threshold D_t, Square Sum Threshold S_t, base radius r_{base}, set of scales S
Result: Detections
Compute the gradient G of image I;
Create a vote image with size $\lceil \frac{w}{s} \rceil \times \lceil \frac{h}{s} \rceil$ for each $s \in S$ and store them in V;
for *each pixel* $\boldsymbol{g}(\boldsymbol{p}) \in G$ **do**
  if $||\boldsymbol{g}(\boldsymbol{p})|| < G_t$ **then**
    skip pixel;
  end
  for *each scale value* $s \in S$ **do**
    Cast vote into the vote image v that corresponds to scale s;
  end
end
for *each scale value* $s \in S$ **do**
  for *each pixel* $q(\boldsymbol{p})$ *of the vote image corresponding to scale* s **do**
    if $q(\boldsymbol{p}) > \frac{D_t}{3}$ **then**
      Compute $t_r(\mathbf{p})$, $t_{rs}(\mathbf{p})$ and μ_r from $q(\mathbf{p})$ and neighboring scales of s;
      if $t_r(\boldsymbol{p}) > D_t$ ***and*** $t_{rs}(\boldsymbol{p}) > S_t$ **then**
        Emit shape detection at position $s\,\mathbf{p}$ with radius/apothem μ_r;
      end
    end
  end
end

Figure 20: Improved Radial Symmetry Detector Algorithm

5.4 Circle Detection

To detect Circles, we used the same voting mechanism as Barnes et al in [8], which only uses the votes from the orientation image O_r. Instead we make +1 votes into a 2x2 square and -1 votes into a similar 2x2 square. Vote images generated for Figure 19 were generated by using this technique, and the detections obtained by using the improved thresholding

algorithm are shown in Figure 21.

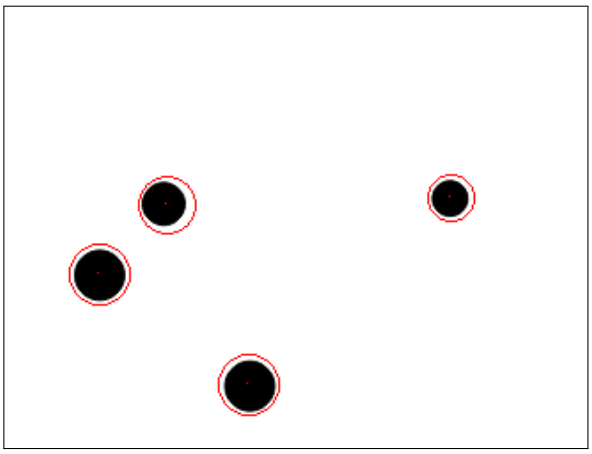


Figure 21: Detections generated by the Improved Circle RSD

5.5 Regular Polygon Detection

To detect regular polygons, we used 2 approaches. Both approaches use the same line voting scheme of Loy & Barnes [38]. The first approach is to use direct voting into a Orientation integer image, in the same way as described in Section 4.2.

The voting line is given by:

$$\begin{aligned} L_+(\mathbf{p}, m) &= \text{round}(\mathbf{p}_+ + m\mathbf{g}_\perp(\mathbf{p})) = \text{round}\left(\frac{\mathbf{p}}{s} + r_{base}\frac{\mathbf{g}(\mathbf{p})}{||\mathbf{g}(\mathbf{p})||} + m\mathbf{g}_\perp(\mathbf{p})\right) \\ L_-(\mathbf{p}, m) &= \text{round}(\mathbf{p}_+ - m\mathbf{g}_\perp(\mathbf{p})) = \text{round}\left(\frac{\mathbf{p}}{s} - r_{base}\frac{\mathbf{g}(\mathbf{p})}{||\mathbf{g}(\mathbf{p})||} - m\mathbf{g}_\perp(\mathbf{p})\right) \end{aligned} \tag{37}$$

The size of the voting line w is slightly different due to scale and is given by w_s, where s is the scale:

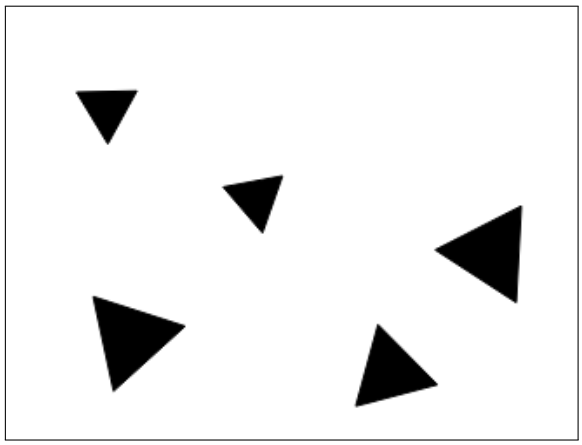
$$w_s = s\, r_{base}\, \tan\frac{\pi}{n} \tag{38}$$

The second approach uses voting into the equiangular image and then combines the equiangular image $\mathbf{B}_s$ into a final scalar vote image F_s using the following equation:

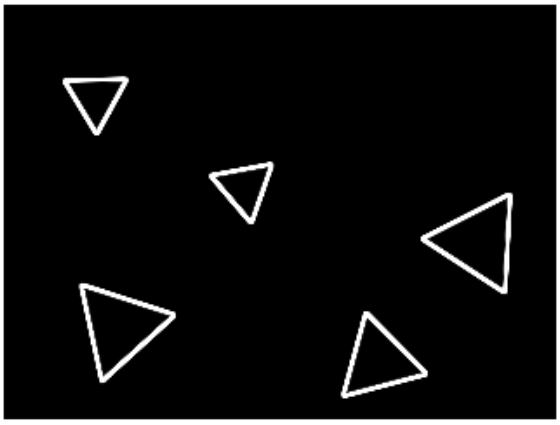
$$F_s(\mathbf{p}) = \frac{||\mathbf{B}_s(\mathbf{p})||}{2s\, w_s\, r_{base}} \tag{39}$$

We should note that for regular polygons, the radius is substituted by the apothem of the corresponding regular polygon. An example triangle detector output can be seen in

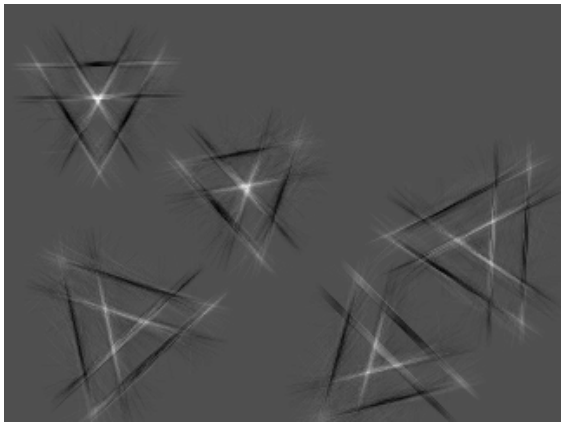
Figure 22, and the corresponding detections are shown in Figure 23.



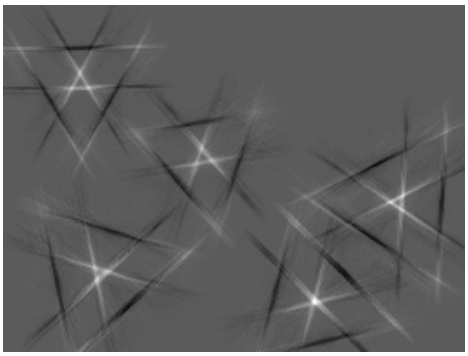
(a) Original Image



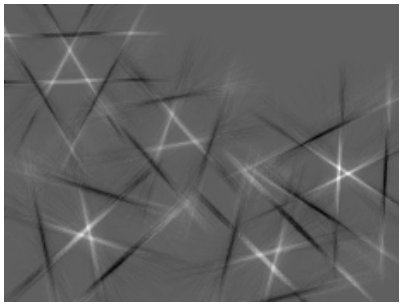
(b) Image Gradient



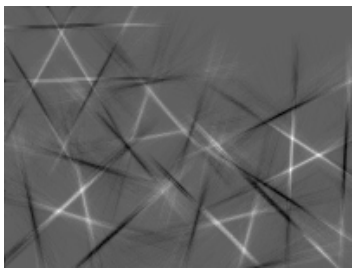
(c) Vote Image at scale $s = 1.0$



(d) Vote Image at scale $s = 1.2$



(e) Vote Image at scale $s = 1.4$



(f) Vote Image at scale $s = 1.6$

Figure 22: Improved Regular Polygon Detector example different scales. Base apothem is 10 pixels. Triangles have apothem of 10, 12, 14 or 16 pixels.

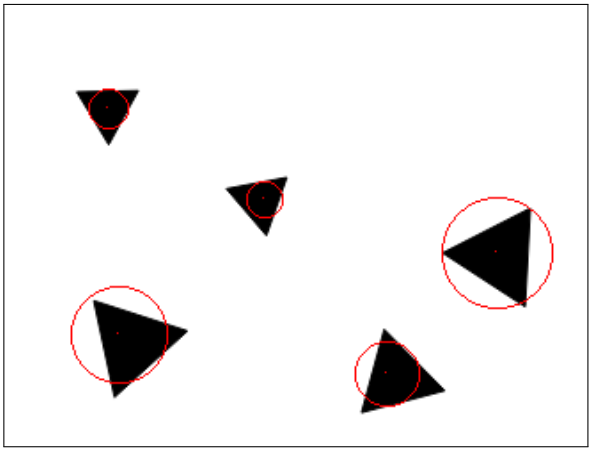


Figure 23: Triangle Detections

5.6 Improving the Original Regular Polygon Detector

Loy & Barnes defined the equiangular (or N-Angular) vote image in [38], but since its expensive to compute it, and if the traffic sign orientation is known, then there is no need to compute this image.

But the equiangular vote image represents very strong features that are only valid for regular polygons. We wanted to test the hypothesis that the equiangular image is better at detecting regular polygons than plain ± 1 votes.

For this we built a Regular Polygon Detector that has the exact same structure as Loy & Barnes Regular Polygon Detector [38], but uses only the equiangular image. This image is then combined into a scalar vote image F_r with the following equation:

$$F_r(\mathbf{p}) = \frac{||\mathbf{B}_s(\mathbf{p})||}{2wr} \tag{40}$$

Then the vote image F_r is then thresholded by using the same simple thresholding algorithm used by Loy & Barnes.

6 Experimental Setup

6.1 General Remarks

We ran the experiment on a Asus Zenbook UX32VD Laptop, with a Core i5-3317U Processor with a clock frequency of 1.7 Ghz and 10 GB of RAM. This processor uses TurboBoost and the clock frequency could go as high as 2.4 Ghz.

The compiler was GCC 4.8.2 and all code was compiled in release mode (without any debug symbols) and using optimization level 3 (`-O3` compiler flag).

To measure computation time, We used the `clock_gettime` with `CLOCK_PROCESS_CPUTIME_ID`, which measures the CPU time used by the process. This function should have nanosecond precision.

6.2 Detector Configuration

We implemented 7 detector algorithms, with different configurations:

Circle Detection

- **OCRSD**: Radial Symmetry Detector as presented in [39]
- **SCRSD**: Radial Symmetry Detector for Circular Traffic Signs as presented in [7].
- **ICRSD**: Improved Circle Radial Symmetry Detector presented in Section 5.4.

Regular Polygon Detector

- **RPD**: Regular Polygon Detector as presented in [38].
- **RPD-NAO**: Regular Polygon Detector as presented in [38], but only using the Equiangular Image as voting information. This was discussed in Section 5.6.
- **IRPD**: Improved Regular Polygon Detector presented in Section 5.5.
- **IRPD-NAO**: Improved Regular Polygon Detector presented in Section 5.5, but only using the Equiangular image as voting information.

To evaluate the different detector algorithms, We took three different approaches. The first approach is to generate synthetic images containing shapes to be detected, with different radius. This way we have a precise knowledge about the ground truth information, and performance of each detector algorithm with different conditions can be evaluated.

6.3 Evaluation on Synthetic Images

Shapes are generated at random positions p, which are uniformly distributed $U(a, b)$ in the X and Y directions, according to:

$$\begin{aligned} p_x &\sim U(\epsilon, w - \epsilon) \\ p_y &\sim U(\epsilon, h - \epsilon) \end{aligned} \tag{41}$$

Where image I has size $w \times h$ pixels. ϵ is a border value to avoid shapes being generated into the border of the image, and currently has a value of:

$$\epsilon = 1.5r + 10 \tag{42}$$

Where r is the radius or apothem of the shape being drawn, with a 10 pixel constant border. For the current experimental results, we used an image size of 320×240 pixels. The orientation θ of each shape is also randomly determined, given by a uniform distribution $\theta \sim U(0, 360)$. Special care was taken to avoid overlapping shapes, since 3 shapes are generated for each test image.

Then we test detectors by generating a image dataset of 100 images, and do threshold tuning for the dataset. We repeat this process for each radius/apothem value in $r \in [10, 30]$, and compute detection rate, false positive rate and computation time for each radius. This range was selected by considering 2 factors. Traffic signs smaller than 10 pixels cannot be correctly classified, so this should be the minimum radius to be evaluated. If we look at the GTSDB [29], the estimated radii distribution for circles is peaked around radius 9, and then the tail gets near 0 at around radius 30, so the bound $r \in [0, 30]$ was chosen.

4 shapes were evaluated independently, Circles, Squares, Triangles and Octagons. This shapes were chosen because they represent the most common traffic sign shapes.

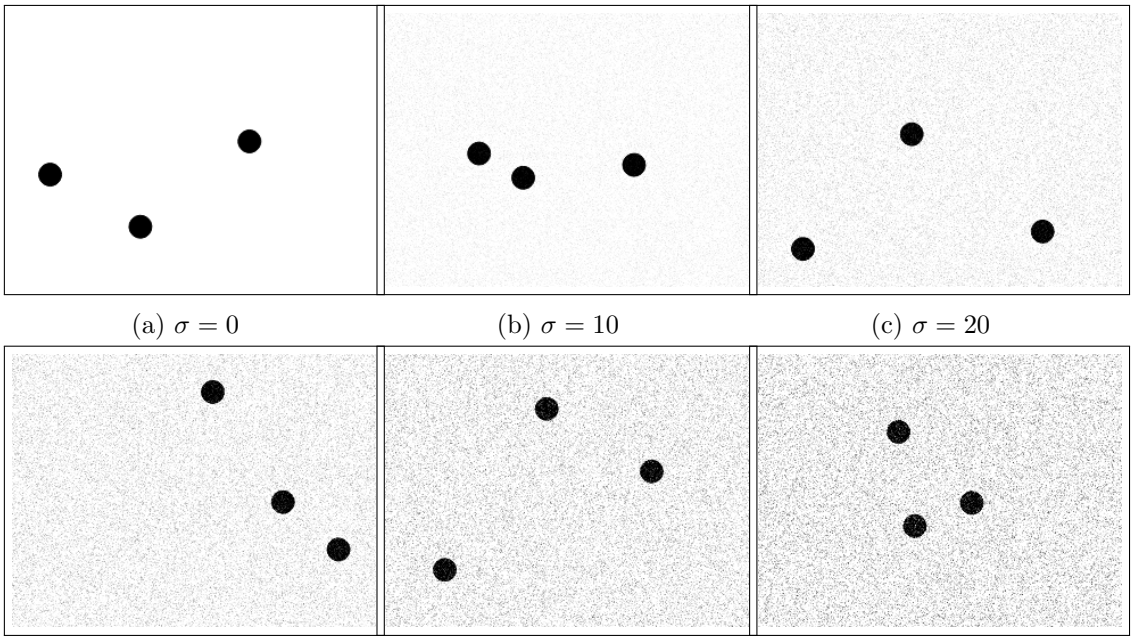
One problem with this approach is that in reality images have noise, and synthetic images are noise-free. To simulate this behaviour, we add gaussian noise to each image pixel [25], according to the Gaussian probability distribution function $N(\mu, \sigma)$ (PDF):

$$P(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{43}$$

This Gaussian noise has zero mean $\mu = 0$, and a given standard deviation σ. We generate random noise values that are Gaussian distributed $e \sim N(0, \sigma)$ and add it to each image pixel $\mathbf{p} \in I$ of the generated image I:

$$I(\mathbf{p}) = I(\mathbf{p}) + e \tag{44}$$

This is done for each image pixel on the image I. This method assumes that the images I are one channel grayscale, so scalar noise values are directly added to the grayscale values and then clamped to the range $[0, 255]$. We should note that this means that for big standard deviation values, noise will be clipped to the maximum pixel value of 255. In



(a) $\sigma = 0$ (b) $\sigma = 10$ (c) $\sigma = 20$

(d) $\sigma = 30$ (e) $\sigma = 40$ (f) $\sigma = 50$

Figure 24: Gaussian Image noise with varying values of standard deviation σ

Figure 24 we can see gaussian image noise with different standard deviations, from $\sigma = 0$ to $\sigma = 50$.

The chosen range of noise was $\sigma \in [0, 10, 20, 30, 40, 50]$. This represents noise ranges from none to an amount that will generate enough variation to be clipped by the maximum range of the image values (255). Gaussian noise of $\sigma = 50$ will be approximately in the range of $[-150, 150]$.

The parameters used to do threshold tuning are given in Table 1.

	Original RSD/RPD	Improved RSD/RPD
Minimum Detection Threshold D_t	1	0
Maximum Detection Threshold D_t	100	75000
Detection Threshold step	1	5000
Minimum Square Sum Threshold S_t	-	0
Maximum Square Sum Threshold S_t	-	1000000
Square Sum Threshold step	-	100000
Gradient Magnitude Threshold G_t	105	105
Scales	-	{ 0.95, 1.0, 1.05 }

Table 1: Threshold tuning parameters used for synthetic image evaluation

Then finally we generated a new dataset of 100 images of the given shape and used it as cross validation set. Then we ran the detector with this new dataset and computed Detection Rates, False Positive Rates and Computation Times. The mean and standard deviation is computed for the whole dataset and set as detection rate for this radius value.

6.4 Evaluation on a Traffic Sign Dataset

The second approach consisted on testing each detector algorithm over the German Traffic Sign Detection Benchmark dataset [29] (GTSDB).

The GTSDB provides 900 real world traffic sign images, as well as ground truth information. Ground truth contains the traffic sign type, and the x and y coordinates of the bounding box of the traffic sign. All traffic signs contained in this dataset are either Circles, Triangles, Squares or Octagons.

Since the GTSDB does not contain ground truth information about the radius or apothem of the traffic sign, it must be estimated from the available information. To do this first we estimate the biggest side of the bounding box, where ul is the upper left point of the bounding box, and lr is the lower right point:

$$\begin{aligned} w &= |ul_x - lr_x| \\ h &= |ul_y - lr_y| \end{aligned} \tag{45}$$

Then the biggest half-side hs is given by:

$$hs = \max\left\{\frac{w}{2}, \frac{h}{2}\right\} \tag{46}$$

Then the radius/apothem r of the traffic sign is given by:

$$r = \frac{hs}{\cos(\frac{\pi}{4})} = \frac{hs}{\sqrt{2}} \sim 0.707hs \tag{47}$$

The histogram of estimated radius/apothem for the 4 basic shapes can be seen in Figure 25. We also split the GTSDB dataset according to shape, and "cut" each traffic sign into his own image, with a size of 320x240 pixels. This is done to normalize the dataset reduce the time needed when doing experiments, since also Barnes et al also used the same image size.

But since the estimated radius/apothem is only an estimation, we still could be several pixels away from the real radius. To compensate for this, we configured each detector with a radii set of $R = \{r-1, r-2, r-3\}$, and for the improved detector, with a scale set of

$S = \{0.95, 1.0, 1.05\}$.

We also do threshold tuning to compute the maximum performance that can be extracted from each detector. The parameters are the same than Synthetic Image Evaluation and are given in Table 1. We split the dataset into batches of shapes with the same radius/apothem and tune them together.

6.5 Multiple Radius Evaluation

The proposed evaluation on synthetic images has a problem, it does not consider the fact that in real life applications, we usually want to detect shapes with varying radiuses, which translates to a detector configured with a set R of radii values, since we do not know what exactly is the radius of the shape being detected on each image.

For this all detector algorithms evaluated use either a multiple radii set R or a multiple scale set S. Then a more realistic situation could be simulated by generating images with different radius/apothem values and running a detector configured with this radii/apothem values appropiately.

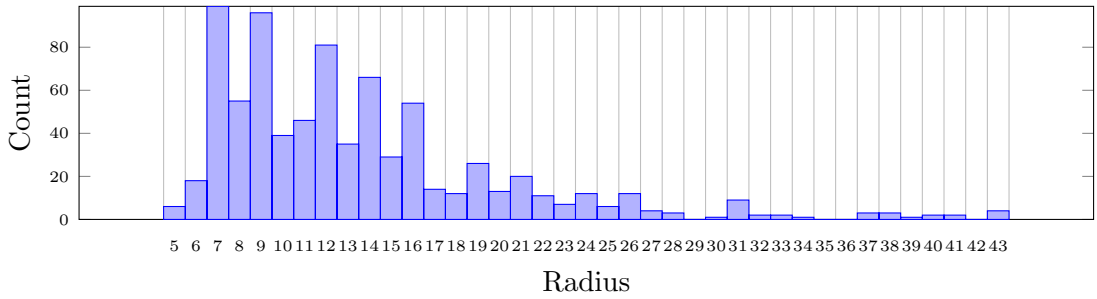
For this we generated 100 image 3 shapes each and with radius/apothem selected randomly from the set $R = \{10, 12, 14, 16, 18, 20\}$. The probability of each radius to be selected is the same (a Uniform distribution).

The Original detectors are configured with a set of radii $R = \{10, 12, 14, 16, 18, 20\}$. The improved detectors are configured with a set of scales $S = \{1.0, 1.2, 1.4, 1.6, 1.8, 2.0\}$ and a base radius value of $r_{base} = 10$. All other detector parameters are the ones specified in Table 1.

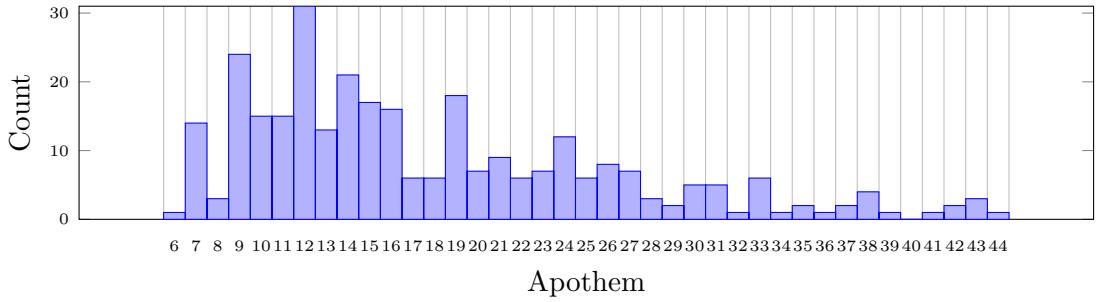
For this experiment we compute mean and standard deviation of the detection rates, false positive rates and computation times, but for all the detection process over the whole 100 image dataset. We also evaluate the effect of noise by adding gaussian noise with different standard deviation values, from $\sigma = 0$ to $sigma = 50$ in steps of 10 units.

6.6 ROC Curve Evaluation

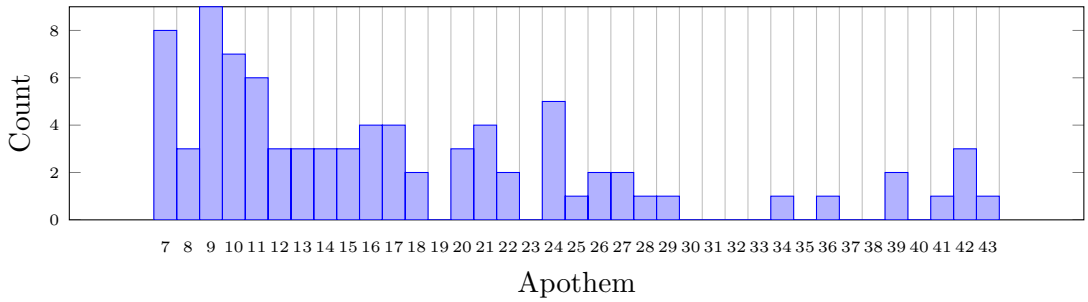
ROC curve is a common visualization technique for detector performance [20]. It plots the False Positive Rate versus the Detection Rate (or True Positive Rate), but each point of the curve is generated by changing the detection threshold inherent to most detector algorithms.



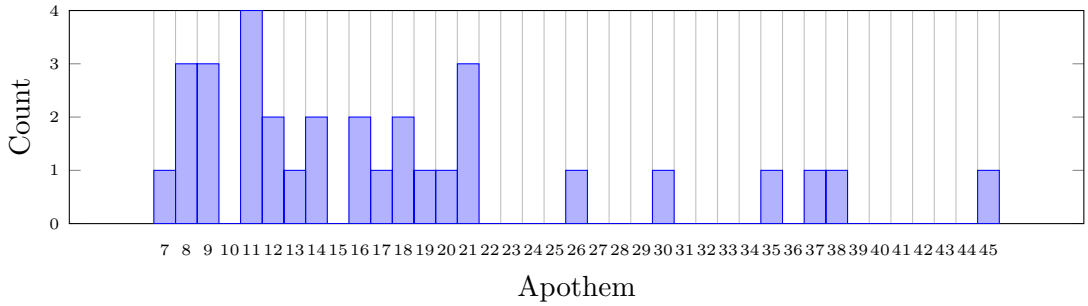
(a) Circles



(b) Triangles



(c) Squares



(d) Octagons

Figure 25: Histogram of estimated radius/apothem distributions of the GTSDB

To compute the ROC curves, we run each detector over a generated dataset of 200 images of the corresponding shape, with a radius/apothem of $r = 10$, and gaussian noise $\sigma = 50$. We vary the detection threshold from $D_t = 0$ to $D_t = 100$, in steps of 1 unit. For the improved detectors, we vary the detection threshold from $D_t = 0$ to $D_t = 10000$ in steps of 100 units. Gradient magnitude threshold is $G_t = 105$.

After generating each pair of false positive and detection rates, we plot them and fit a curve which is draw as the ROC curve [42].

6.7 Speedup Error Propagation

To compare computation times of 2 algorithms the speedup is usually used, which is defined as:

$$S = \frac{T_o}{T_i} \tag{48}$$

Where T_o is the time taken by the original algorithm, and T_i is the time of the improved algorithm. To do error propagation, we use the standard error propagation formula of a function f, where we assume independent variables:

$$\sigma_f^2 = \sum_{i=0}^{n} \left(\frac{\partial f}{\partial x_i} \right)^2 \sigma_{x_i}^2 \tag{49}$$

Where σ_f^2 is the variance of the function f, and $\sigma_{x_i}^2$ is the variance of the i-th variable. If we apply this equation to the speedup function S then we obtain:

$$\sigma_S^2 = \frac{\sigma_{T_o}^2}{T_i^2} + \frac{T_o^2 \sigma_{T_i}^2}{T_i^4} \tag{50}$$

Where σ_{T_o} is the standard deviation of the time taken by the original algorithm, and σ_{T_i} is the standard deviation of the time taken by the improved algorithm. We will use the value of σ_S when computing speedup of computation times.

7 Experimental Results and Analysis

7.1 Synthetic Image Results

7.1.1 Circle Detector

Raw data results can be seen in Figures 29, 30 and 31 in the Appendix. For the Detection rate (DR) we can see that all detectors have very high detection rates, and differences

are minimal. The Improved Circle Detector is clearly the best, and only in few cases its detection rate is smaller than the original circle detectors.

Regarding the False Positive Rate (FPR), all detectors have very low false positive rates, with only a few radius values giving false positive rate bigger than 1%. The original circle detectors have also some variance in the detection rate, while the improved circle detector has a very small variance, which cannot even be seen in the plot.

For Computation Time, clearly the improved circle detector has better performance than the original detectors. But with noise from $\sigma = 40$, performance gets unstable, with a sharp peak around radius $r = 18$. With noise of $\sigma = 50$, the performance benefit is almost lost, with smaller computation time only from radius $r = 22$.

This performance variation can be attributed to the Gradient Magnitude Threshold G_t. Too much noise will make more gradient pixels pass this threshold test, which will degrade performance.

We should mention that the original circle detectors have linear time complexity in the radius r, while our improved circle detector has constant time complexity.

7.1.2 Regular Polygon Detector

Raw data results for Triangles can be seen in Figures 32, 33 and 34 in the Appendix.

In Figure 32 we can see that all detectors have very high detection rates, except for the Improved Polygon Detector with the Equiangular Image (IRPD-NA). Also the Original Polygon Detector (ORPD) has issues when the noise increases, the detection rate decreases significantly from apothem $r = 25$, and for $r = 30$ it is 0 for big noise values.

We should remember that the original RPD algorithm does not use gaussian smoothing, so this could be a sign of the cost of not using this kind of smoothing. This can also be seen in the False Positive Rate in Figure 33, where the False Positive Rate of the original detector is very high for big noise and radius values.

False Positive Rates are low for all detectors except for the Improved Detector with the Equiangular Image. The Improved Detector also has some issues with FPR around $0.2 - 0.4$ in some cases with increased noise, while the original detector does not have this issue.

About Computation Times, we can see in Figure 34 across different noise values the original RPD with only Equiangular Image is the best, while the Improved Detector coming second, but this difference gets smaller as noise increases. Also the Original Detector has better performance than the Improved Detector as noise increases.

About the square detector, Detection Rates (in Figure 35) are again high for all detectors except the Improved detector with Equiangular Image, but this gap gets smaller as the apothem value increases, but this improved detector never reaches detection rate above 90%.

About False positive rates (in Figure 36), when there is no noise they are small, around 20% in average, but as noise increases the FPR decreases for all detectors, except for the Improved Detector with Equiangular Image. This detector has a high false positive rate, around maximum of 90%, which makes it useless for most uses.

Logic says that FPR rates should increase with noise, and not decrease, so this is a extraneous situation. For Computation Times as shown in Figure 35, performance behaviour is the same as the Triangle Detector, where the original RPD with only Equiangular Image has the best performance, followed by the Improved detector, but this trend reverts when noise increases, where the original RPD is faster than the improved RPD.

Finally, about Octagon Detection, the same trend than the previous detector remains, where the Improved detector with Equiangular Image has the smallest detection rate, while the other detectors have very high detection rates. FPR are similar, with low FPR values for all detectors except the Improved detector with Equiangular Image.

About computational performance, again the RPD with Equiangular Image is the best, followed very closely by the Improved detector.

All detector algorithms have linear time complexity in the apothem value. Also the computation times when noise increases degrade very fast, with maximum noise of $\sigma = 50$ have performance in the order of hundreds of milliseconds, which clearly is not appropiate for real-time performance.

Clearly the Improved detector with Equiangular Image is useless as a regular polygon detector, with very high false positive rates, and at least 20-30% lower detection rates than the other polygon detectors.

7.2 Traffic Sign Dataset Results

7.2.1 Circle Detector

Aggregate results from testing each detector with the GTSDB can be seen in Table 2 and specific results divided by Radius can be seen in Figure 41. For this dataset, we can see that in average the detection rate of the Improved Detector is lower than the Original Detector, while the false positive rate of the Improved Detector is slightly better (lower). For most cases the performance of the Sign RSD is better than the Original detector.

Computational performance is clearly better for the Improved detector, with a speedup of 1.62. The Improved Circle Detector has constant time complexity, while the 2 original detectors have linear time complexity in the radius value.

	OCRSD	SCRSD	ICRSD
Average Detection Rate	0.57	0.58	0.40
Detection Rate σ	0.25	0.24	0.18
Average False Positive Rate	0.86	0.86	0.74
False Positive Rate σ	0.15	0.14	0.24
Computation Time (ms)	57	57	35
Computation Time σ	13	14	22

Table 2: Circle Detector Performance under the GTSDB

7.2.2 Regular Polygon Detector

For the Triangle Detector, aggregate results can be seen in Table 3 while results divided by apothem can be seen in Figure 42. We can see that all improved detectors have better detection rates than the RPD, with the IRPD and IRPD with Equiangular Image having the highest detection rates at 64%.

But FPR are high, not being lower than 95% for all detectors, which make this kind of detectors a bit useless for real work. It should be noted that using only the Equiangular Image in the RPD (as in RPD-NAO) practically doubles the detection rate, from 19% to 44%. This is one the hypotheses that were proposed in the present work.

About computational performance, clearly the fastest detector is the IRPD, with a speedup of 2.1. In this case this detector is superior to the RPD in both detection rate and computation times, which makes it a strong improvement over the original.

For the Square Detector, aggregate results can be seen in Table 4 while results divided

by apothem can be seen in Figure 43. Again, all improved detectors have detection rates bigger than the original RPD, with the greatest detection rate going to the RPD-NAO. Clearly the use of the Equiangular Image is a big improvement. The other improved detectors have a slightly lower detection rate.

FPR are high but lower than with Triangle Detection. The lowest detection rate goes with the Improved Detectors. The RPD-NAO might have the biggest detection rate, but the smallest detection rate are given by the IRPD and IRPD-NAO.

All Improved Detectors have better computational performance than the original detectors, and the best speedup is given by the IRPD, with a speedup of 3.1. If we make some sacrifices about the detection rate, the IRPD and IRPD-NAO are the best detectors in this category.

For the Octagon Detector, aggregate results can be seen in Table 5 while results divided by apothem can be seen in Figure 44. Again all Improved detectors have better detection rates than the Original RPD, with the RPD-NAO having the best detection rate at 90%.

About False Positive Rates, the FPR of all Improved detectors is lower at 86%, but the RPD-NAO has a slightly higher FPR than the original detector.

In computational performance, clearly the IRPD and IRPD-NAO are the best algorithms, and if some tradeoff about detection rate could be made, then this two detectors are the best for Octagon detection. The speedup over the original detector is 1.8.

As a general remark, we can see that computational performance always improves as the regular polygon number of sides is increased, due to the line length w being smaller.

We should note that the averages given in Tables 2, 3, 4 and 5 were computed as simple averages for each radius/apothem batch. Since the distribution of radius/apothem values is not uniform, there are more samples of some values than others, which means that the averages could be skewed. A simple solution for this is to compute averages weighted by the amount of radius/apothem values present for each batch.

7.3 Multiple Radius Evaluation

7.3.1 Circle Detector

First, for the Circle detector the raw data plots can be seen in Figure 45. We can see that for all the evaluated circle detectors, the detection rates are very high, almost always 100%, and the false positives are very low, very near 0%.

	RPD	RPD-NAO	IRPD	IRPD-NAO
Average Detection Rate	0.19	0.44	0.64	0.64
Detection Rate σ	0.25	0.28	0.25	0.25
Average False Positive Rate	0.95	0.96	0.98	0.98
False Positive Rate σ	0.08	0.07	0.03	0.03
Computation Time (ms)	383	331	183	187
Computation Time σ	345	268	103	106

Table 3: Triangle Detector Performance under the GTSDB

	RPD	RPD-NAO	IRPD	IRPD-NAO
Average Detection Rate	0.52	0.84	0.78	0.78
Detection Rate σ	0.36	0.23	0.26	0.26
Average False Positive Rate	0.86	0.80	0.71	0.71
False Positive Rate σ	0.17	0.26	0.27	0.27
Computation Time (ms)	267	232	115	117
Computation Time σ	218	177	70	73

Table 4: Square Detector Performance under the GTSDB

	RPD	RPD-NAO	IRPD	IRPD-NAO
Average Detection Rate	0.55	0.90	0.80	0.80
Detection Rate σ	0.43	0.24	0.32	0.32
Average False Positive Rate	0.95	0.97	0.86	0.86
False Positive Rate σ	0.11	0.02	0.22	0.22
Computation Time (ms)	117	108	65	64
Computation Time σ	57	54	20	20

Table 5: Octagon Detector Performance under the GTSDB

Table 6 presents the computation times, and this and subsequent tables contain computation times with standard deviations, as well as speedups with their standard deviations computer via error propagation. Here we can see the raw computation times for each detector (with standard deviations), as well as computed speedups betwen the Sign RSD (SCRSD) and the Improved detector. We can see that for small noise values, up to $\sigma = 20$ the speedups are considerably high, with values between 5 and 6. When noise increases beyond $\sigma = 20$, the speedup decreases but stays at least twice as fast as the original circle detector. Since all other metrics (DR and FPR) are the same, this is clearly a big improvement.

The drop in speedup could be attributed to the fixed value of the gradient magnitude threshold G_t. Too much noise will degrade the performance of the detector.

	$\sigma = 0$	$\sigma = 10$	$\sigma = 20$	$\sigma = 30$	$\sigma = 40$	$\sigma = 50$
OCRSD	78 ± 2	76 ± 1	79 ± 1	90 ± 6	90 ± 1	98 ± 5
SCRSD	74 ± 1	77 ± 2	76 ± 1	83 ± 3	91 ± 5	92 ± 7
ICRSD	13 ± 1	13 ± 1	14 ± 1	31 ± 1	36 ± 1	37 ± 5
Speedup	5.7 ± 0.5	5.9 ± 0.5	5.4 ± 0.4	2.7 ± 0.1	2.5 ± 0.2	2.5 ± 0.4

Table 6: Multiple Circle Detector Evaluation Computation Times (ms)

7.3.2 Regular Polygon Detector

For the Triangle Detector, the raw data plots can be seen in Figure 46. We can see that as noise increases, the detection rate of the Original detector (RPD) decreases. It starts in around 60% and ends in 40%. The improved detector starts with a detection rate of about 50%, and increases as noise increases, ending in around 70%, dominating the other detectors, but only when noise standard deviation is big.

The original detector (RPD) has high false positive rate, in the range of 60%, while the improved detector (IRPD) starts with 80% FPR, but it decreases as noise standard deviation increases, ending at around 30%. Logic dictates that FPR should increase with increasing noise, but this behaviour by the IRPD contradicts this logic.

In Table 7 we can see the computed speedups between the RPD and the IRPD. The IRPD is always faster, with speedups greater than 2.8, and when noise standard deviation increases, the speedup also increases, with a maximum speedup of 3.8.

	$\sigma = 0$	$\sigma = 10$	$\sigma = 20$	$\sigma = 30$	$\sigma = 40$	$\sigma = 50$
RPD	85 ± 6	85 ± 6	236 ± 13	729 ± 36	1240 ± 64	1682 ± 104
RPD-NA	69 ± 8	66 ± 5	179 ± 9	689 ± 68	1132 ± 113	1512 ± 130
IRPD	30 ± 2	30 ± 2	66 ± 4	197 ± 5	340 ± 15	441 ± 10
IRPD-NA	35 ± 3	34 ± 2	69 ± 3	199 ± 9	333 ± 16	451 ± 42
Speedup	2.8 ± 0.3	2.8 ± 0.3	3.6 ± 0.3	3.7 ± 0.2	3.6 ± 0.2	3.8 ± 0.3

Table 7: Multiple Triangle Detector Evaluation Computation Times (ms)

For the Square Detector, in Figure 47 the plots are available. We can see that the Improved detector (IRPD) completely dominates the Original detector (RPD) in detection rates, false positive rates and computation times. About computation times, the IRPD-NA is slighly faster but the detection rates and false positive rates are very bad.

Again the false positive rates of the IRPD decrease as noise standard deviation σ increases. The detection rate of the IRPD in average is close to 70%.

Speedups are available in Table 8. Speedups are very close to 2.0, and they increase with increasing noise standard deviation. The biggest speedup is very close to 3.0, which signals a very nice improvement over the original detector.

	$\sigma = 0$	$\sigma = 10$	$\sigma = 20$	$\sigma = 30$	$\sigma = 40$	$\sigma = 50$
RPD	59 ± 3	59 ± 3	132 ± 7	358 ± 10	598 ± 32	789 ± 45
RPD-NA	40 ± 2	43 ± 3	129 ± 11	353 ± 48	541 ± 51	728 ± 64
IRPD	31 ± 4	26 ± 1	60 ± 2	135 ± 4	213 ± 4	275 ± 4
IRPD-NA	39 ± 5	42 ± 5	55 ± 2	127 ± 17	188 ± 6	245 ± 7
Speedup	1.9 ± 0.3	2.3 ± 0.1	2.2 ± 0.1	2.7 ± 0.1	2.8 ± 0.2	2.9 ± 0.2

Table 8: Multiple Square Detector Evaluation Computation Times (ms)

Finally, about Octagon Detection, we can see the data plots in Figure 48. Detection rates for the original (RPD) and improved detectors (IRPD and RPD-NAO) are very high and degrade with increasing noise (as expected). The detection rates of the improved detector never go below 90%, while the other detectors have slightly lower detection rates.

False positive rates for the improved detector (IRPD) are considerably low (less than 5%), and they slightly increase as noise increases. FPR for the other detectors are always bigger than the IRPD. The IRPD-NAO has very big FPR values, around 70%.

In Table 9 we can see the speedup results. Speedups are in the range of 1.8 to 2.4, and they have no clear increasing or decreasing trend (probably due to experimental randomization). The biggest speedup of 2.4 is gotten with the maximum noise with standard deviation $\sigma = 50$.

	$\sigma = 0$	$\sigma = 10$	$\sigma = 20$	$\sigma = 30$	$\sigma = 40$	$\sigma = 50$
RPD	46 ± 1	46 ± 1	83 ± 4	175 ± 3	217 ± 7	348 ± 27
RPD-NA	32 ± 2	31 ± 2	61 ± 2	167 ± 15	269 ± 27	300 ± 13
IRPD	20 ± 1	21 ± 1	45 ± 1	83 ± 3	118 ± 2	149 ± 2
IRPD-NA	24 ± 1	24 ± 1	36 ± 2	73 ± 6	106 ± 5	130 ± 5
Speedup	2.3 ± 0.1	2.2 ± 0.1	1.8 ± 0.1	2.1 ± 0.1	1.8 ± 0.1	2.4 ± 0.2

Table 9: Multiple Octagon Detector Evaluation Computation Times (ms)

Doing a global comparison of the RPD experiments we can infer that as the number of sides of the shape increases, the performance speedup decreases. With the Triangle Detector we obtained the biggest speedups, nearly 6 times faster than the original detector,

and the smallest speedups were gotten with the Octagon Detector, averaging 2 times faster.

The Improved detector with Equiangular Image (IRPD-NA) has a very bad detection performance, with consistent lower detection rates and bigger false positive rates. Clearly it is not a good choice as a detector. But performance-wise it is at least as good as the IRPD, and in some cases it has a slightly lower computation time.

It should be also mentioned that in pretty much all cases the IRPD had lower variation of the computation times than the Original RPD. This indicates that the IRPD has a more stable behaviour with respect to computation time, but performance is still degraded by noise.

7.4 ROC Curve Results

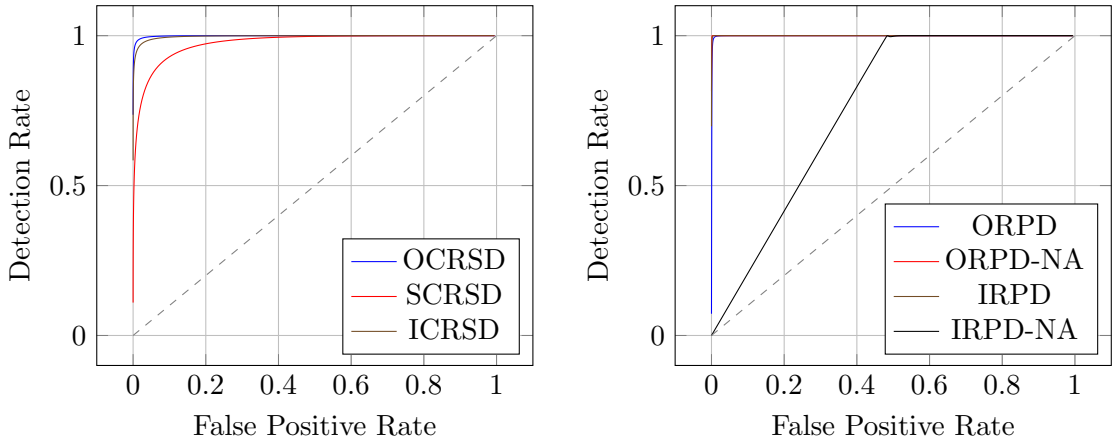
ROC curves can be seen in Figure 26. For the circle detector, all ROC curves are pretty similar and indicate that all detectors can successfully perform 100% detection rate with 0% false positive rate.

For the Regular Polygon Detectors, their shape again is very similar, except for the Improved RPD with Equiangular Image, which clearly cannot perform at 100 % detection rate with 0% false positive rate. The minimum FPR to get 100% detection rate is around 50% false positive rate, which is useless for any serious work.

The ROC curve of the Triangle detector cannot be evaluated on Figure 26 due to scale, so a "zoomed" version is available in Figure 27. Here we can see tha small differences, where the IRPD is in between the curves for the ORPD-NA and the ORPD, but anyways the difference is very small.

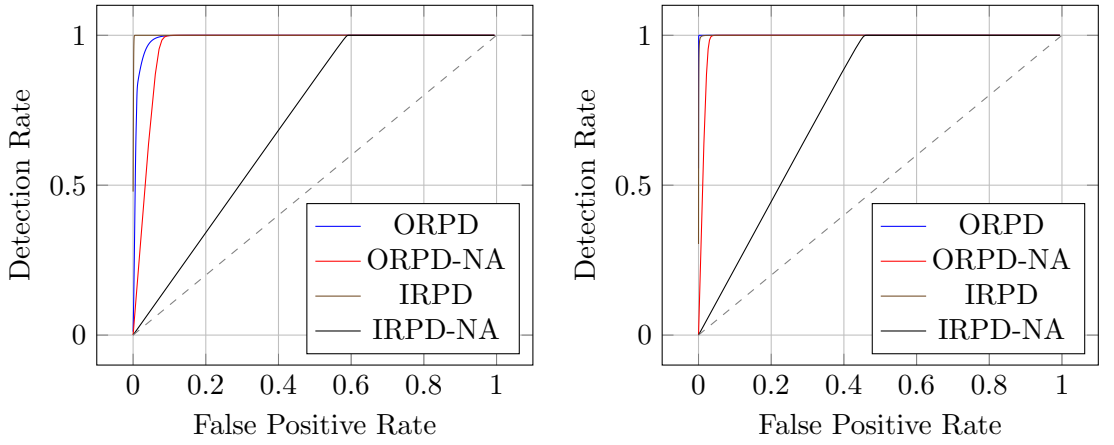
All detectors perform in the "good" side of the plot and they have the possibility of tuning the detection threshold to obtain 100% detection rate with very low false positive rate.

If we compute areas under the ROC curves, areas for all detectors will be very near 1.0, except for the IRPD-NA, which has an approximate area of 0.7 for the Square Detector and an area of 0.75 for the Triangle and Octagon Detectors. This signals that the IRPD-NA has not good detection performance when compared with the other detectors.



(a) Circle Detector

(b) Triangle Detector



(c) Square Detector

(d) Octagon Detector

Figure 26: ROC Curves with $\sigma = 50$

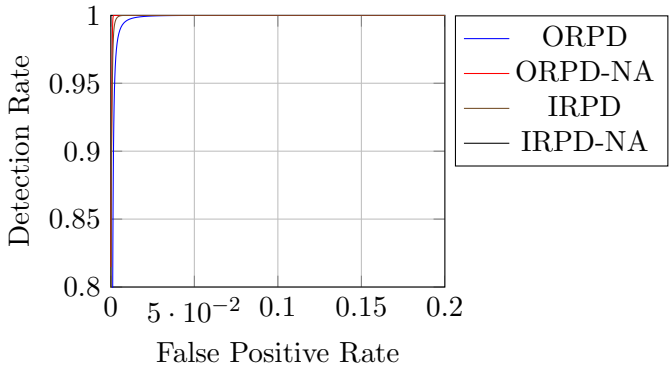


Figure 27: Triangle Detector ROC Curve, Zoom for $x \in [0, 0.2]$ and $y \in [0.8, 1.0]$

8 Conclusions

In the current report we have presented improvements to the Radial Symmetry Detector, both for Circle and Regular Polygon detection.

To improve the RSD algorithm, we used a multiscale approach with the intention of reducing the size of the vote images, and we also removed the gaussian smoothing filter at the end of the voting process. We proposed a new optional thresholding method that uses the multiple scale informatio as a way to improve detection rates and decrease false positive rates.

This approach was successful in improving computational performance, while keeping the same or better detection and false positive rates. To evaluate our new Improved Radial Symmetry Detector, we used synthetic generated images containing shapes, as well as the German Traffic Sign Detection Benchmark [29].

For synthetic generated images, at resolution of 320×240 pixels and in average, we obtained speedups of 6 times faster than the original RSD for a Circle Detector, 3.4 times faster for a Triangle Detector, 2.5 times faster for a Square Detector and 2.1 times faster for a Octagon Detector. All of this improvements were followed by comparable or better detection and false positive rates.

For real world images in the GTSDB, for the Circle Detector we obtained a small improvement of 1.6 times faster, with the cost of a smaller detection rates (20% difference) and a slightly smaller false positive rate.

For the Triangle Detector, we obtained a 2.1 times faster performance improvement, with a better detection rate and a comparable false positive rate. We must mention that detection rates for the original and improved detectors were high, in the range of 90%.

For the Square Detector, a speedup of 2.3 times faster computational performance was observed, with a considerable greater detection rate than the original regular polygon detector, but at the cost of slightly increased false positive rates.

Finally for the Octagon Detector, we observed a speedup of 1.8 times faster, with a increased detection and false positive rates.

Our improved Circle Radial Symmetry Detector has constant time complexity with

respect to radius of the detected circle, and it only depends on the size of the input image, the number of scales being considered, the amount of noise in the image, and the gradient magnitude threshold G_t and detection threshold D_t.

The Improved Regular Polygon Detector has linear time complexity with respect to the apothem of detected regular polygon, and this performance also depends on the same parameters as the Improved Circle RSD.

We can conclude that our improvements to the Radial Symmetry Detector have decreased the computational cost of running the algorithm in a input image, while keeping comparable detection and false positive rates, while for some shapes, this metrics are better than the original detector.

We also tested the sensitivity of the detector to the threshold value by plotting ROC curves. We observed that the ROC curves are very comparable (almost equal), and they all allow the detector to operate with 100% detection rate with a very low false positive rate.

We expect that our improvements to the RSD will help make the way to have real-time traffic sign detection and recognition on embedded computer platforms.

8.1 Future Work

But there is still much work to be done. We did not consider or control the orientation of the shapes or traffic signs, and some detectors appear to be sensitive to this variable. More work is needed to ensure that this variable is taken into account. The gradient orientation along with the shape information could be used to improve the computational performance of the detector, by only voting with the gradient pixels that meet this constraint. This has already been done by Barnes et al [5].

We also did some rudimentary normalization of the vote images among different radii values, but more work is needed in this area to build a strong detector that can successfully compare votes in different radius/apothem scales. This could also improve the detection and false positive rates of our improved detector.

A simple way of improving the performance of the regular polygon detector is to decrease the size of the voting line. We performed some basic experiments that were not included in this report about this, and it shows that the line can be slightly decreased

with no effect in the detector and false positive rates, slightly increasing performance.

Uncertainty of the radius/apothem values must be mentioned. The GTSDB does not contain this information and it has been estimated from the ground truth data, but this will also affect the evaluated detector performance, since we do not know the sensibility of the detector to uncertainty in the radius/apothem. We do not know the ground truth radius of the traffic signs in the GTSDB, so this could be a variable to be considered in the future.

Noise in the image is and will be always a problem in Image Processing and Computer Vision algorithms. For the RSD and the Improved RSD algorithms, noise also affects the computational performance. Currently we use a gradient magnitude threshold to deal with part of the noise, but this is not enough, and more work is needed to make the RSD more noise resistant, so noise does not degrade computational performance.

False positive rates are very high in some cases, and this could pose a problem for some general purpose applications. For traffic sign recognition, this is not a big issue since the candidate traffic signs will be recognized/classified by a posterior recognition stage, which should be able to recognize signs and not signs.

Some false positives can be seen in Figure 28, where the RPD detected correctly the square traffic sign, but it also detected the traffic lights, the direction sign and even a incomplete square made by the traffic light post. But for other kinds of applications that require circle and/or regular polygon detection, this could pose a problem.

Wwe only experimented with normal voting process already established in the literature, but other "voting kernels" could be tested and they might improve performance and/or detection and false positive rates. We already know that voting kernels and smoothing the vote image or accumulator arrays are equivalent [3], so a computational improvement might be along this path.

Finally, more testing and experimentation must be done. Special care must be taken with embedded hardware platforms. Since ADAS must run in automotive grade embedded processors, performance characteristics could be different and might lead to different requirements for a traffic sign detector algorithm. So the Radial Symmetry Detector and the Improved RSD presented in this report must be tested on this kind of platforms, as well as real datasets with continuous image frame from a camera.

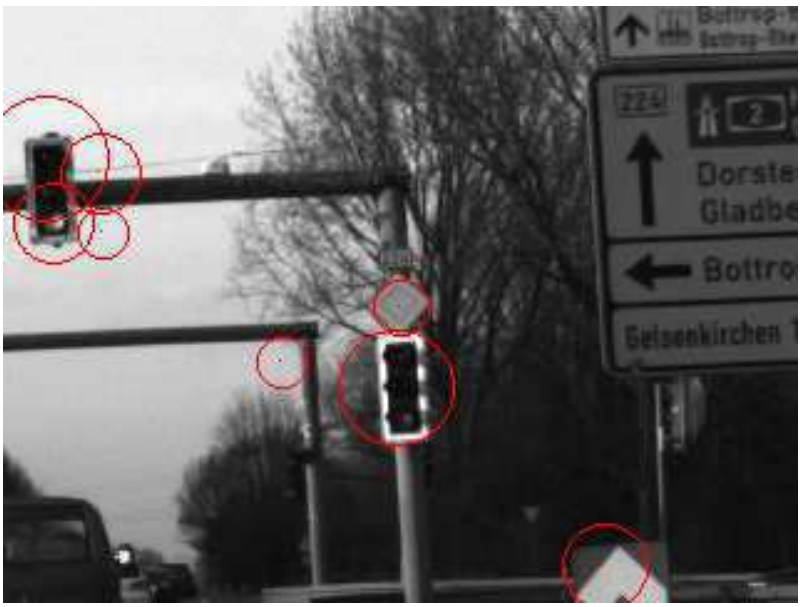


Figure 28: False Positives in a image of the GTSDB

9 References

[1] Bram Alefs, Guy Eschemann, Herbert Ramoser, and Csaba Beleznai. Road sign detection from edge orientation histograms. In *Intelligent Vehicles Symposium, 2007 IEEE*, pages 993–998. IEEE, 2007.

[2] Claus Bahlmann, Ying Zhu, Visvanathan Ramesh, Martin Pellkofer, and Thorsten Koehler. A system for traffic sign detection, tracking, and recognition using color, shape, and motion information. In *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, pages 255–260. IEEE, 2005.

[3] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.

[4] Nick Barnes. Improved signal to noise ratio and computational speed for gradient-based detection algorithms. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 4661–4666. IEEE, 2005.

[5] Nick Barnes and Gareth Loy. Real-time regular polygonal sign detection. In *Field and Service Robotics*, pages 55–66. Springer, 2006.

[6] Nick Barnes, Gareth Loy, David Shaw, and Antonio Robles-Kelly. Regular polygon detection. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 778–785. IEEE, 2005.

[7] Nick Barnes and Alex Zelinsky. Real-time radial symmetry for speed sign detection. In *Intelligent Vehicles Symposium, 2004 IEEE*, pages 566–571. IEEE, 2004.

[8] Nick Barnes, Alexander Zelinsky, and Luck S Fletcher. Real-time speed sign detection using the radial symmetry detector. *Intelligent Transportation Systems, IEEE Transactions on*, 9(2):322–332, 2008.

[9] Rachid Belaroussi, Philippe Foucher, J-P Tarel, Bahman Soheilian, Pierre Charbonnier, and Nicolas Paparoditis. Road sign detection in images: A case study. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 484–488. IEEE, 2010.

[10] Rachid Belaroussi and Jean-Philippe Tarel. A real-time road sign detection using bilateral chinese transform. In *Advances in Visual Computing*, pages 1161–1170. Springer, 2009.

[11] Long Chen, Qingquan Li, Ming Li, and Qingzhou Mao. Traffic sign detection and recognition for intelligent vehicle. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 908–913. IEEE, 2011.

[12] Teh-Chuan Chen and Kuo-Liang Chung. An efficient randomized algorithm for detecting circles. *Computer Vision and Image Understanding*, 83(2):172–191, 2001.

[13] Dan Ciresan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. A committee of neural networks for traffic sign classification. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1918–1921. IEEE, 2011.

[14] E Cuevas, F Wario, V Osuna-Enciso, D Zaldivar, and M Perez-Cisneros. Fast algorithm for multiple-circle detection on images using learning automata. *IET Image Processing*, 6(8):1124–1135, 2012.

[15] Sambarta Dasgupta, Swagatam Das, Arijit Biswas, and Ajith Abraham. Automatic circle detection on digital images with an adaptive bacterial foraging algorithm. *Soft Computing*, 14(11):1151–1164, 2010.

[16] ER Davies. Minimising the search space for polygon detection using the generalised hough transform. *Pattern recognition letters*, 9(3):181–192, 1989.

[17] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[18] Inland Transport Committee Economic Commission for Europe. Convention on Road Signs and Signals done at Vienna on 8 November 1968, 1968. [Online; accessed 28-December-2013].

[19] Chiung-Yao Fang, Sei-Wang Chen, and Chiou-Shann Fuh. Road-sign detection and tracking. *Vehicular Technology, IEEE Transactions on*, 52(5):1329–1341, 2003.

[20] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[21] Meng-Yin Fu and Yuan-Shui Huang. A survey of traffic sign recognition. In *Wavelet Analysis and Pattern Recognition (ICWAPR), 2010 International Conference on*, pages 119–124. IEEE, 2010.

[22] Miguel Angel Garcia-Garrido, Miguel Angel Sotelo, and E Martm-Gorostiza. Fast traffic sign detection and recognition under changing lighting conditions. In *Intelligent Transportation Systems Conference, 2006. ITSC'06. IEEE*, pages 811–816. IEEE, 2006.

[23] Vladimir Glavtchev, Pınar Muyan-Ozçelik, Jeffrey M Ota, and John D Owens. Feature-based speed limit sign detection using a graphics processing unit. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 195–200. IEEE, 2011.

[24] A Goneid, S El-Gindi, and A Sewisy. A method for the hough transform detection of circles and ellipses using a 1-dimensional array. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, volume 4, pages 3154–3157. IEEE, 1997.

[25] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[26] Chun-Ta Ho and Ling-Hwei Chen. A fast ellipse/circle detector using geometric symmetry. *Pattern Recognition*, 28(1):117–124, 1995.

[27] Frank Hoeppner. Fuzzy shell clustering algorithms in image processing: fuzzy c-rectangular and 2-rectangular shells. *Fuzzy Systems, IEEE Transactions on*, 5(4):599–613, 1997.

[28] Sebastian Houben. A single target voting scheme for traffic sign detection. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 124–129. IEEE, 2011.

[29] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks (submitted)*, 2013.

[30] T Hummel, M Kühn, J Bende, and A Lang. Advanced driver assistance systems. an investigation of their potential safety benefits based on an analysis of insurance claims in germany. german insurance association insurers accident research. *German Insurance Association Insurers Accident Research, Research report FS*, 3, 2011.

[31] Benjamin Höferlin and Klaus Zimmermann. Towards reliable traffic sign recognition. In *Intelligent Vehicles Symposium, 2009 IEEE*, pages 324–329. IEEE, 2009.

[32] Christian Igel, Verena Heidrich-Meisner, and Tobias Glasmachers. Shark. *The Journal of Machine Learning Research*, 9:993–996, 2008.

[33] Alfredo Ferreira Manuel J Fonseca Joaquim and A Jorge. Polygon detection from a set of lines. 2003.

[34] Claudio Rosito Jung and Rodrigo Schramm. Rectangle detection based on a windowed hough transform. In *Computer Graphics and Image Processing, 2004. Proceedings. 17th Brazilian Symposium on*, pages 113–120. IEEE, 2004.

[35] Christoph Gustav Keller, Christoph Sprunk, Claus Bahlmann, Jan Giebel, and Gregory Baratoff. Real-time recognition of us speed signs. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 518–523. IEEE, 2008.

[36] Heung-Soo Kim and Jong-Hwan Kim. A two-step circle detection algorithm from the intersecting chords. *Pattern recognition letters*, 22(6):787–798, 2001.

[37] Carolyn Kimme, Dana Ballard, and Jack Sklansky. Finding circles by an array of accumulators. *Communications of the ACM*, 18(2):120–122, 1975.

[38] Gareth Loy and Nick Barnes. Fast shape-based road sign detection for a driver assistance system. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, pages 70–75. IEEE, 2004.

[39] Gareth Loy and Alexander Zelinsky. Fast radial symmetry for detecting points of interest. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8):959–973, 2003.

[40] Meng Lu, Kees Wevers, and Rob Van Der Heijden. Technical feasibility of advanced driver assistance systems (adas) for road traffic safety. *Transportation Planning and Technology*, 28(3):167–187, 2005.

[41] Saturnino Maldonado-Bascon, Sergio Lafuente-Arroyo, Pedro Gil-Jimenez, Hilario Gomez-Moreno, and Francisco López-Ferreras. Road-sign detection and recognition based on support vector machines. *Intelligent Transportation Systems, IEEE Transactions on*, 8(2):264–278, 2007.

[42] RA Maxion and RR Roberts. *Proper use of ROC curves in Intrusion/Anomaly Detection*. University of Newcastle upon Tyne, Computing Science, 2004.

[43] Andreas Møgelmose, Mohan M Trivedi, and Thomas B Moeslund. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. 2012.

[44] S Muller-Schneiders, Christian Nunn, and Mirko Meuter. Performance evaluation of a real time traffic sign recognition system. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 79–84. IEEE, 2008.

[45] Ali Ajdari Rad, Karim Faez, and Navid Qaragozlou. Fast circle detection using gradient pair vectors. In *DICTA*, pages 879–888. Citeseer, 2003.

[46] Pierre Sermanet and Yann LeCun. Traffic sign recognition with multi-scale convolutional networks. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2809–2813. IEEE, 2011.

[47] Adnan Shaout, Dominic Colella, and S Awad. Advanced driver assistance systems-past, present and future. In *Computer Engineering Conference (ICENCO), 2011 Seventh International*, pages 72–82. IEEE, 2011.

[48] MA Souki, L Boussaid, and M Abid. An embedded system for real-time traffic sign recognizing. In *Design and Test Workshop, 2008. IDT 2008. 3rd International*, pages 273–276. IEEE, 2008.

[49] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.

[50] Gilbert Strang. *Calculus*. Wellesley-Cambridge Press, Wellesley, Mass, 1991.

[51] Mu-Chun Su and Chao-Hsin Hung. A neural-network-based approach to detecting rectangular objects. *Neurocomputing*, 71(1):270–283, 2007.

[52] Saburo Tsuji and Fumio Matsumoto. Detection of ellipses by a modified hough transformation. *Computers, IEEE Transactions on*, 100(8):777–781, 1978.

[53] Hiroshi Ueno, Masayuki Kaneda, and Masataka Tsukino. Development of drowsiness detection system. In *Vehicle Navigation and Information Systems Conference, 1994. Proceedings., 1994*, pages 15–20. IEEE, 1994.

[54] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[55] Ardalan Vahidi and Azim Eskandarian. Research advances in intelligent collision avoidance and adaptive cruise control. *Intelligent Transportation Systems, IEEE Transactions on*, 4(3):143–153, 2003.

[56] C Visvikis, TL Smith, M Pitcher, R Smith, et al. Study on lane departure warning and lane change assistant systems. *Study on lane departure warning and lane change assistant systems*, 1(1):1–124, 2013.

[57] Massaki Wada, Kang Sup Yoon, and Hideki Hashimoto. Development of advanced parking assistance system. *Industrial Electronics, IEEE Transactions on*, 50(1):4–17, 2003.

[58] Eric W. Weisstein. Regular polygon. From MathWorld—A Wolfram Web Resource. URL: `http://mathworld.wolfram.com/RegularPolygon.html`. Last visited on 25/12/2013.

[59] Wikipedia. Ideogram — Wikipedia, the free encyclopedia, 2013. [Online; accessed 29-December-2013].

[60] Lei Xu, Erkki Oja, and Pekka Kultanen. A new curve detection method: randomized hough transform (rht). *Pattern recognition letters*, 11(5):331–338, 1990.

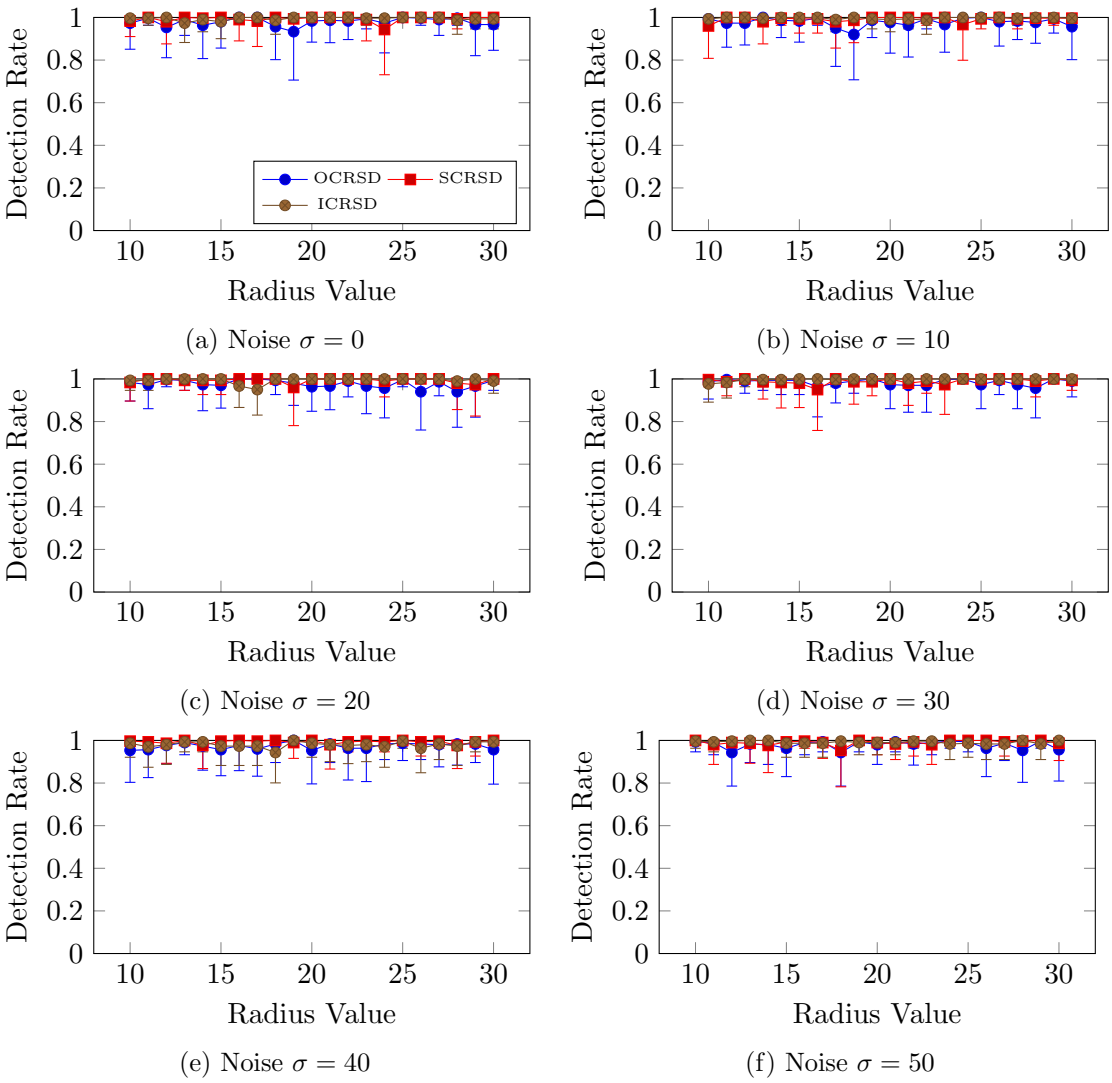
[61] Raymond KK Yip, Peter KS Tam, and Dennis NK Leung. Modification of hough transform for circles and ellipses detection using a 2-dimensional array. *Pattern Recognition*, 25(9):1007–1022, 1992.

[62] Fatin Zaklouta, Bogdan Stanciulescu, and Omar Hamdoun. Traffic sign classification using kd trees and random forests. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2151–2155. IEEE, 2011.

A Experimental Data

A.1 Synthetic Image Evaluation

A.1.1 Circle Detector



(a) Noise $\sigma = 0$

(b) Noise $\sigma = 10$

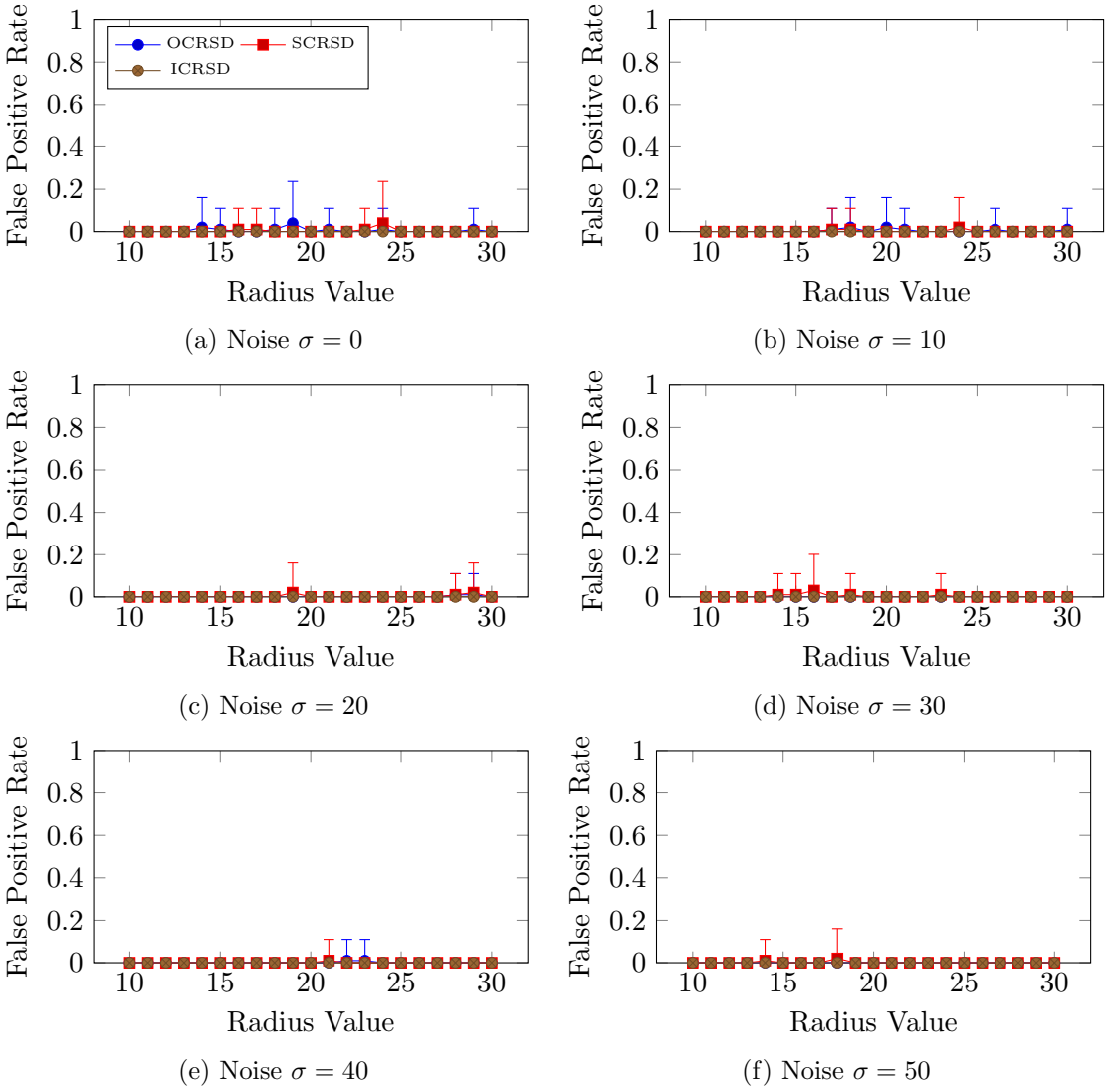
(c) Noise $\sigma = 20$

(d) Noise $\sigma = 30$

(e) Noise $\sigma = 40$

(f) Noise $\sigma = 50$

Figure 29: Circle Detector Detection Rates with Noise from $\sigma = 0$ to $\sigma = 50$



(a) Noise $\sigma = 0$

(b) Noise $\sigma = 10$

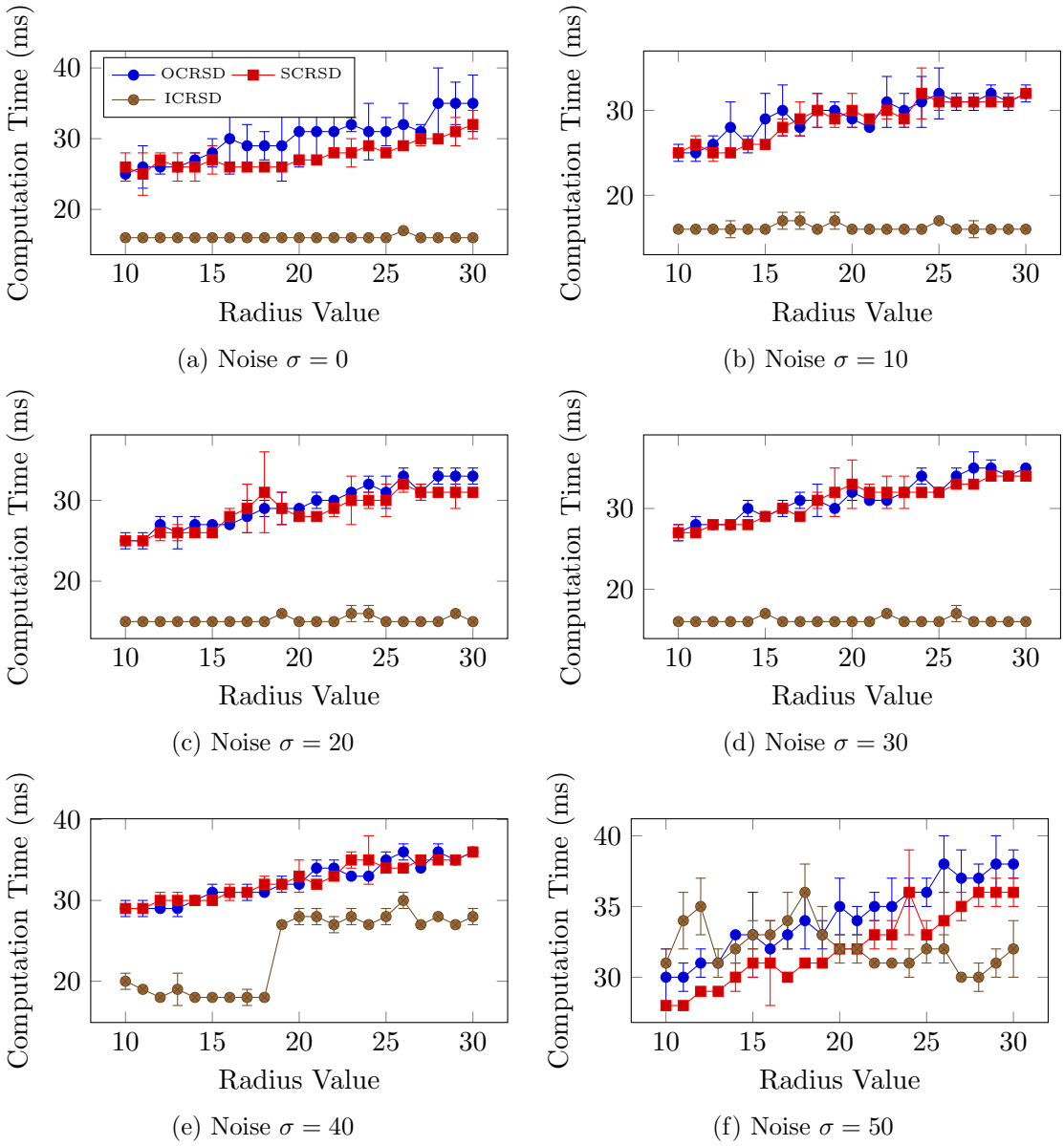
(c) Noise $\sigma = 20$

(d) Noise $\sigma = 30$

(e) Noise $\sigma = 40$

(f) Noise $\sigma = 50$

Figure 30: Circle Detector False Positive Rates with Noise from $\sigma = 0$ to $\sigma = 50$



(a) Noise $\sigma = 0$

(b) Noise $\sigma = 10$

(c) Noise $\sigma = 20$

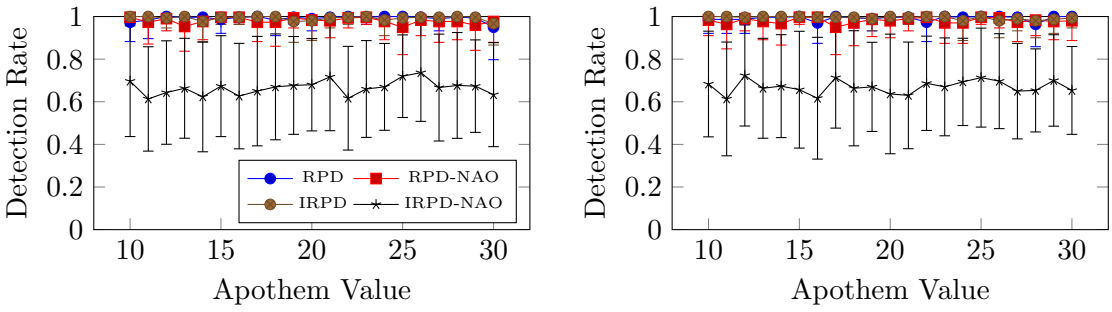
(d) Noise $\sigma = 30$

(e) Noise $\sigma = 40$

(f) Noise $\sigma = 50$

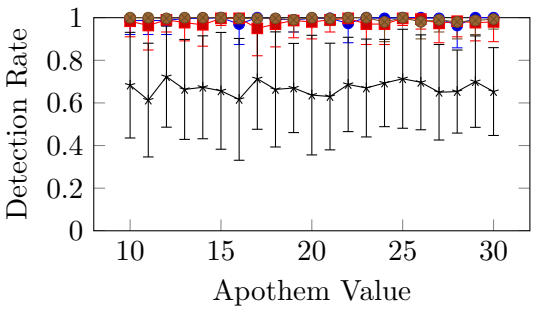
Figure 31: Circle Detector Computation Time (ms) with Noise from $\sigma = 0$ to $\sigma = 50$

A.1.2 Triangle Detector

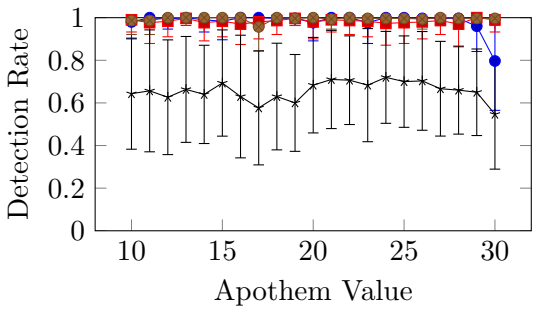


(a) Noise $\sigma = 0$

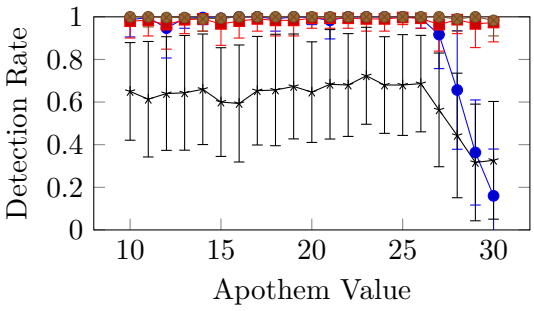
(b) Noise $\sigma = 10$



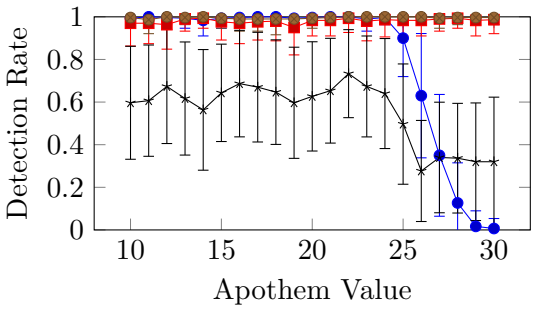
(c) Noise $\sigma = 20$



(d) Noise $\sigma = 30$

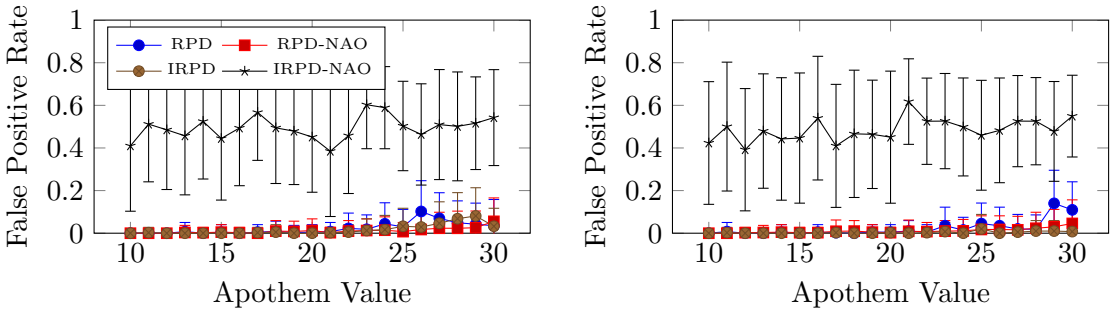


(e) Noise $\sigma = 40$



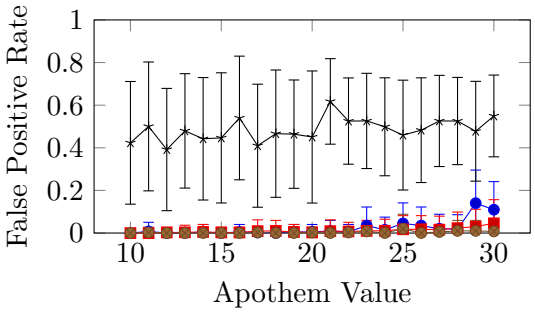
(f) Noise $\sigma = 50$

Figure 32: Triangle Detector Detection Rates with Noise from $\sigma = 0$ to $\sigma = 50$

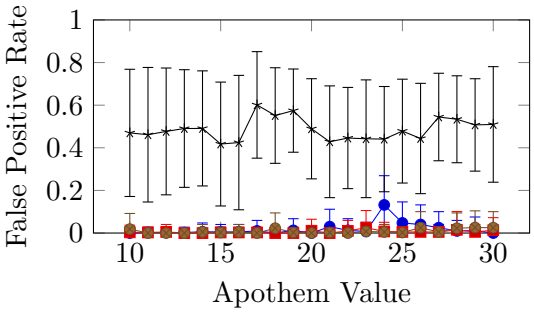


(a) Noise $\sigma = 0$

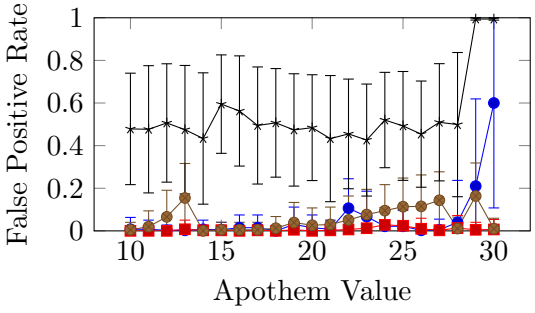
(b) Noise $\sigma = 10$



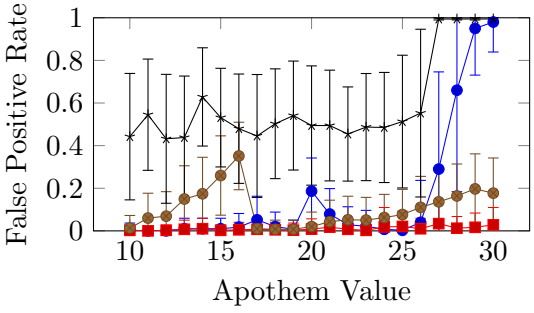
(c) Noise $\sigma = 20$



(d) Noise $\sigma = 30$

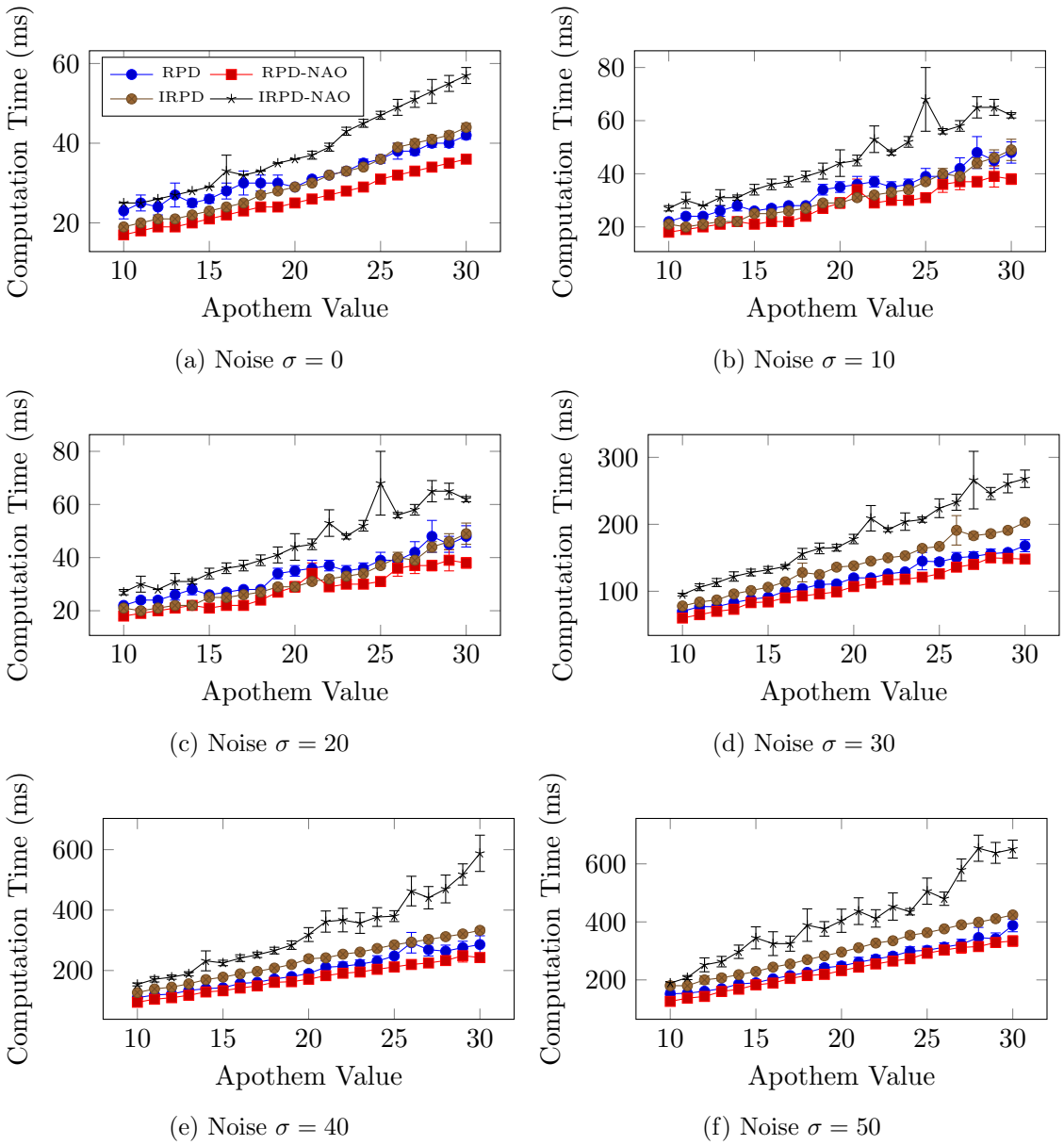


(e) Noise $\sigma = 40$



(f) Noise $\sigma = 50$

Figure 33: Triangle Detector False Positive Rates with Noise from $\sigma = 0$ to $\sigma = 50$



(a) Noise $\sigma = 0$

(b) Noise $\sigma = 10$

(c) Noise $\sigma = 20$

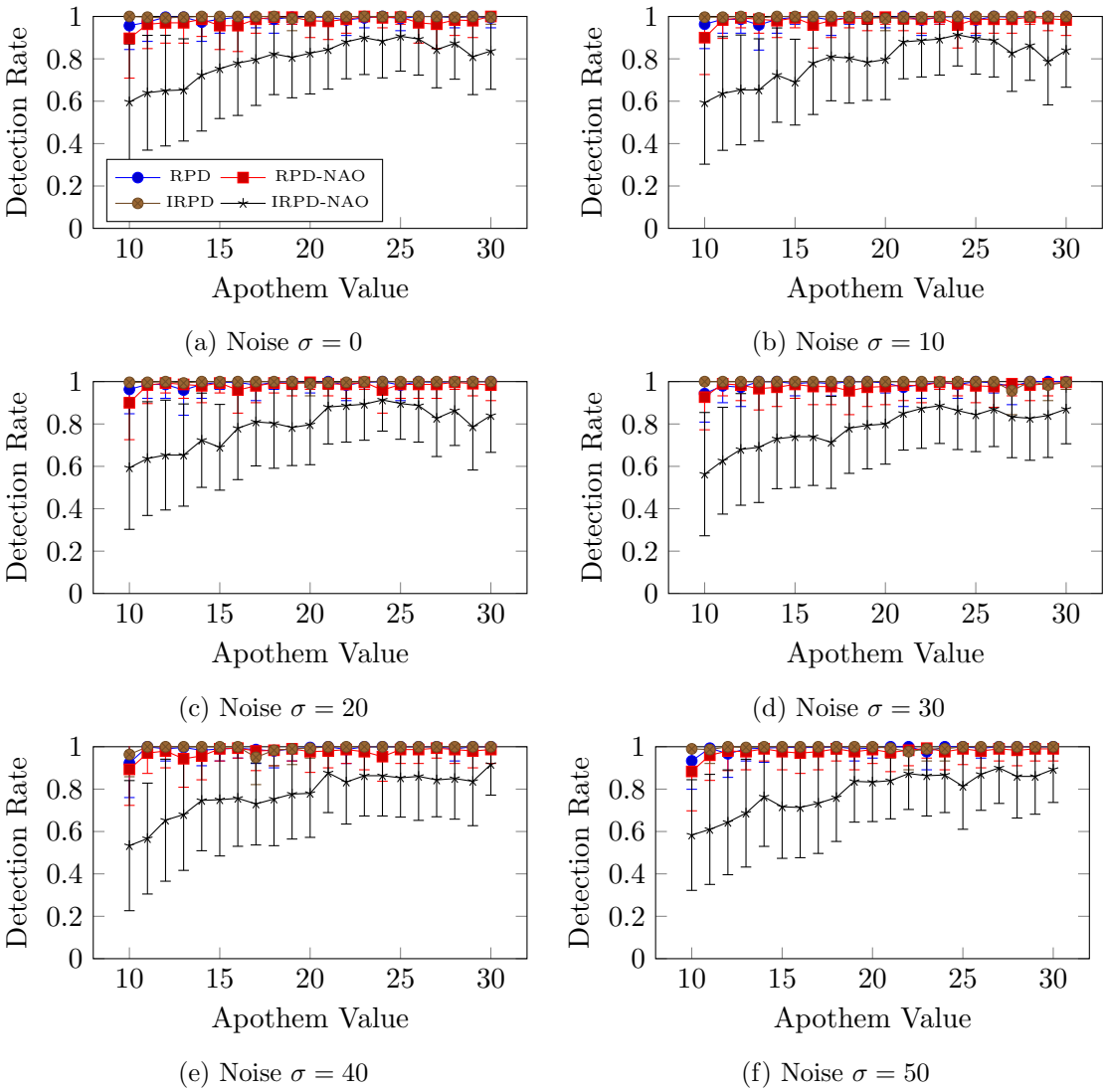
(d) Noise $\sigma = 30$

(e) Noise $\sigma = 40$

(f) Noise $\sigma = 50$

Figure 34: Triangle Detector Computation Time (ms) with Noise from $\sigma = 0$ to $\sigma = 50$

A.1.3 Square Detector



(a) Noise $\sigma = 0$

(b) Noise $\sigma = 10$

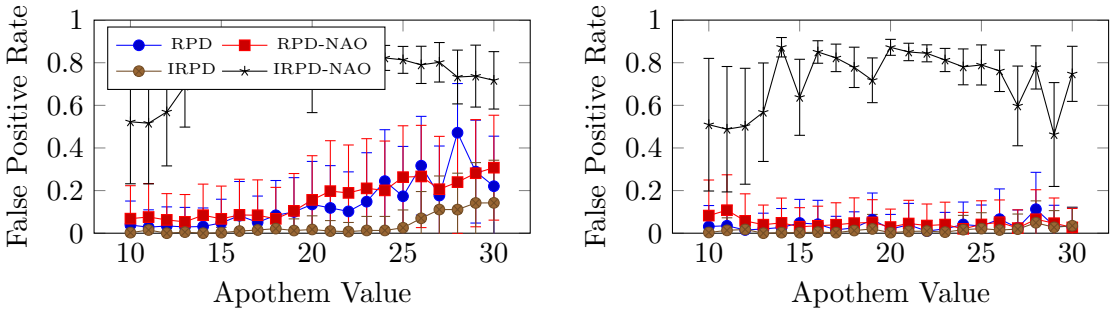
(c) Noise $\sigma = 20$

(d) Noise $\sigma = 30$

(e) Noise $\sigma = 40$

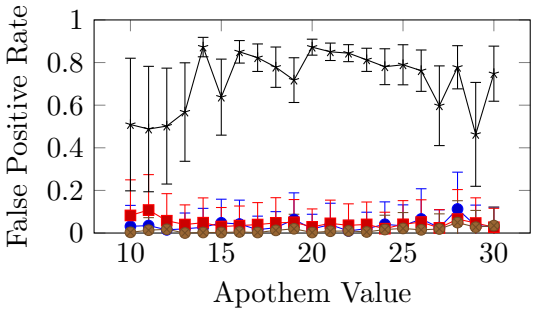
(f) Noise $\sigma = 50$

Figure 35: Square Detector Detection Rates with Noise from $\sigma = 0$ to $\sigma = 50$

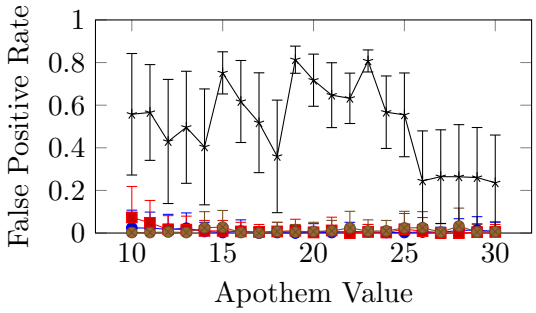


(a) Noise $\sigma = 0$

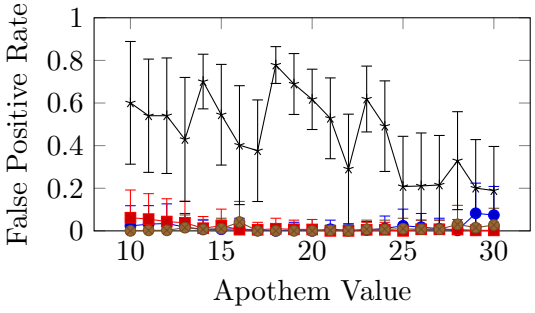
(b) Noise $\sigma = 10$



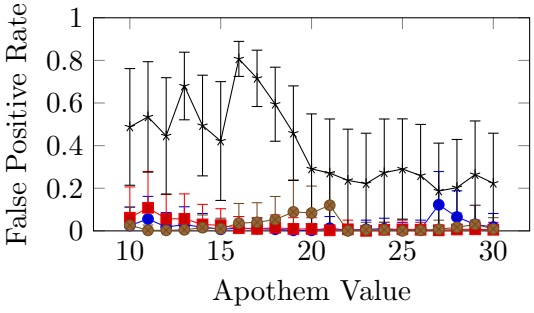
(c) Noise $\sigma = 20$



(d) Noise $\sigma = 30$

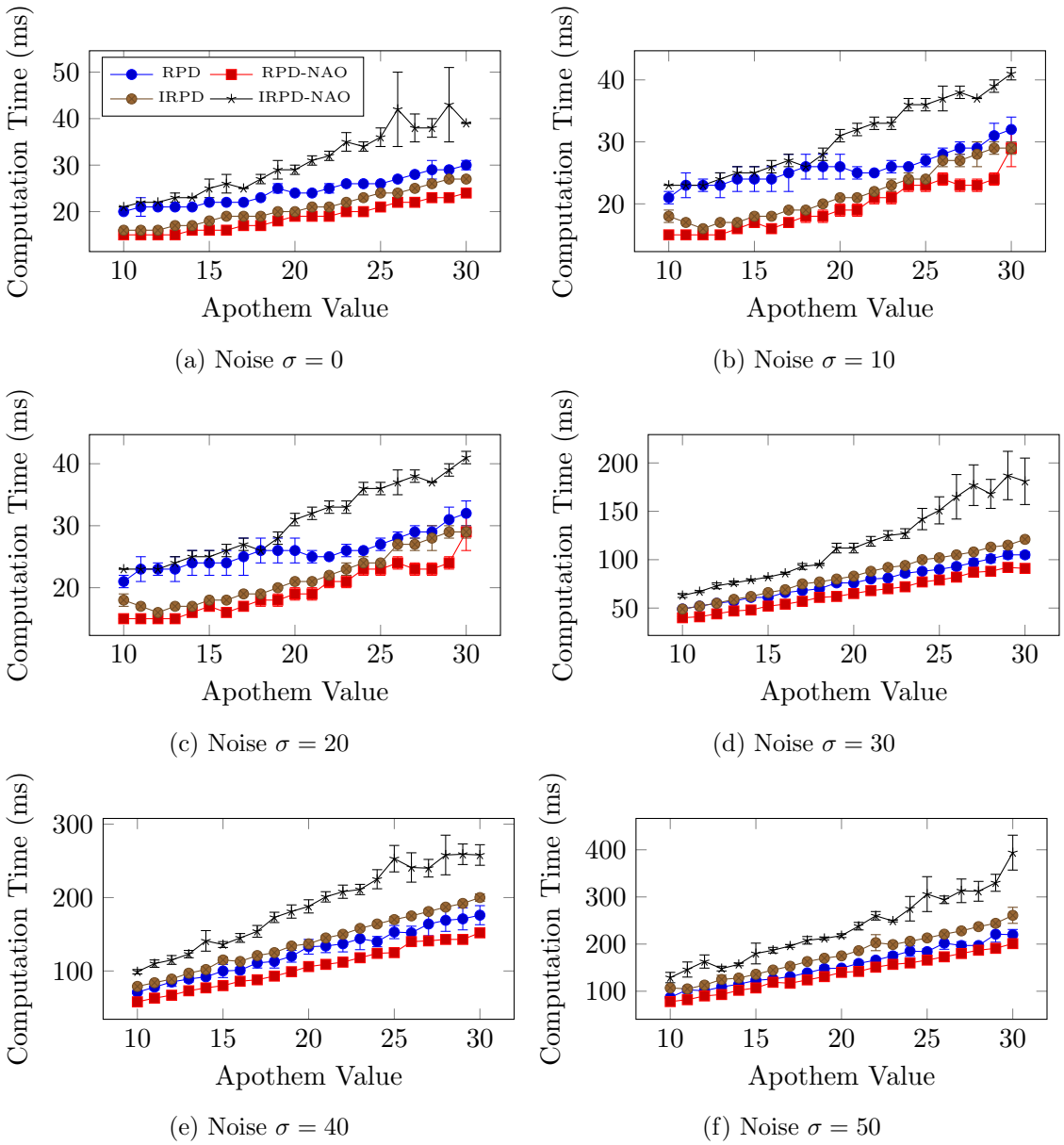


(e) Noise $\sigma = 40$



(f) Noise $\sigma = 50$

Figure 36: Square Detector False Positive Rates with Noise from $\sigma = 0$ to $\sigma = 50$



(a) Noise $\sigma = 0$

(b) Noise $\sigma = 10$

(c) Noise $\sigma = 20$

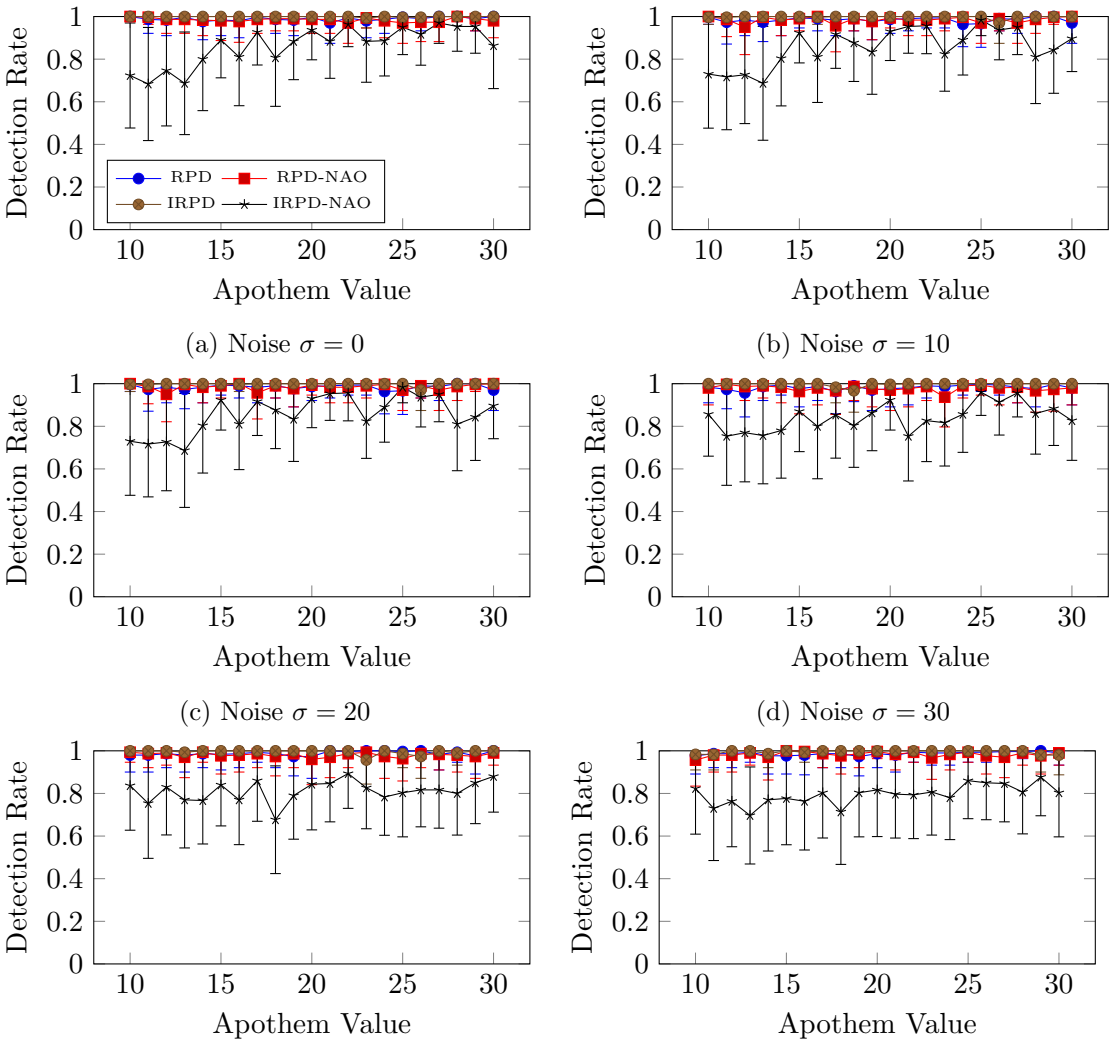
(d) Noise $\sigma = 30$

(e) Noise $\sigma = 40$

(f) Noise $\sigma = 50$

Figure 37: Square Detector Computation Time (ms) with Noise from $\sigma = 0$ to $\sigma = 50$

A.1.4 Octagon Detector



(a) Noise $\sigma = 0$

(b) Noise $\sigma = 10$

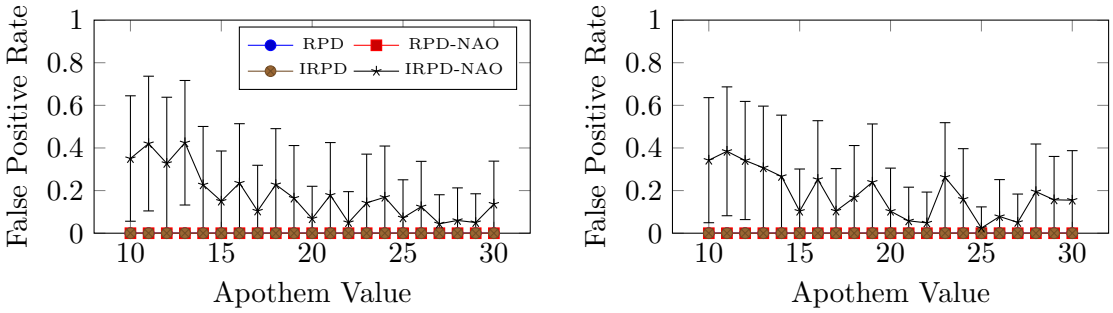
(c) Noise $\sigma = 20$

(d) Noise $\sigma = 30$

(e) Noise $\sigma = 40$

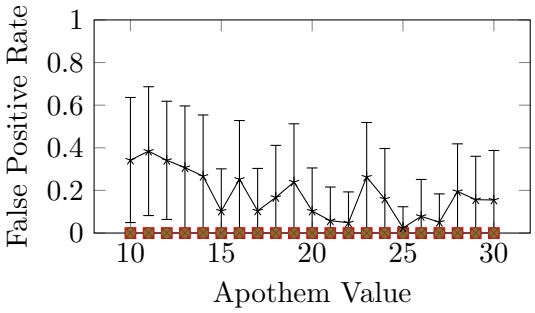
(f) Noise $\sigma = 50$

Figure 38: Octagon Detector Detection Rates with Noise from $\sigma = 0$ to $\sigma = 50$

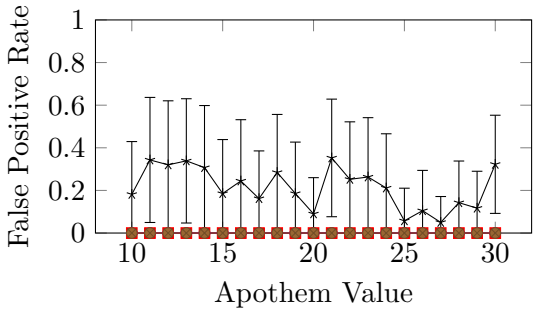


(a) Noise $\sigma = 0$

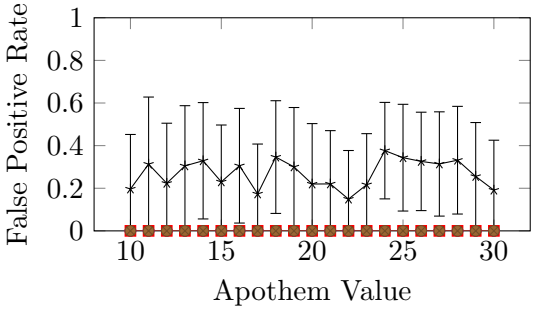
(b) Noise $\sigma = 10$



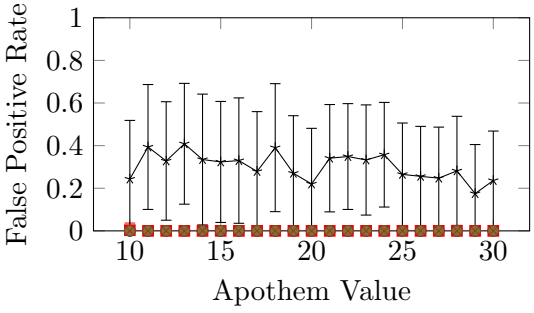
(c) Noise $\sigma = 20$



(d) Noise $\sigma = 30$

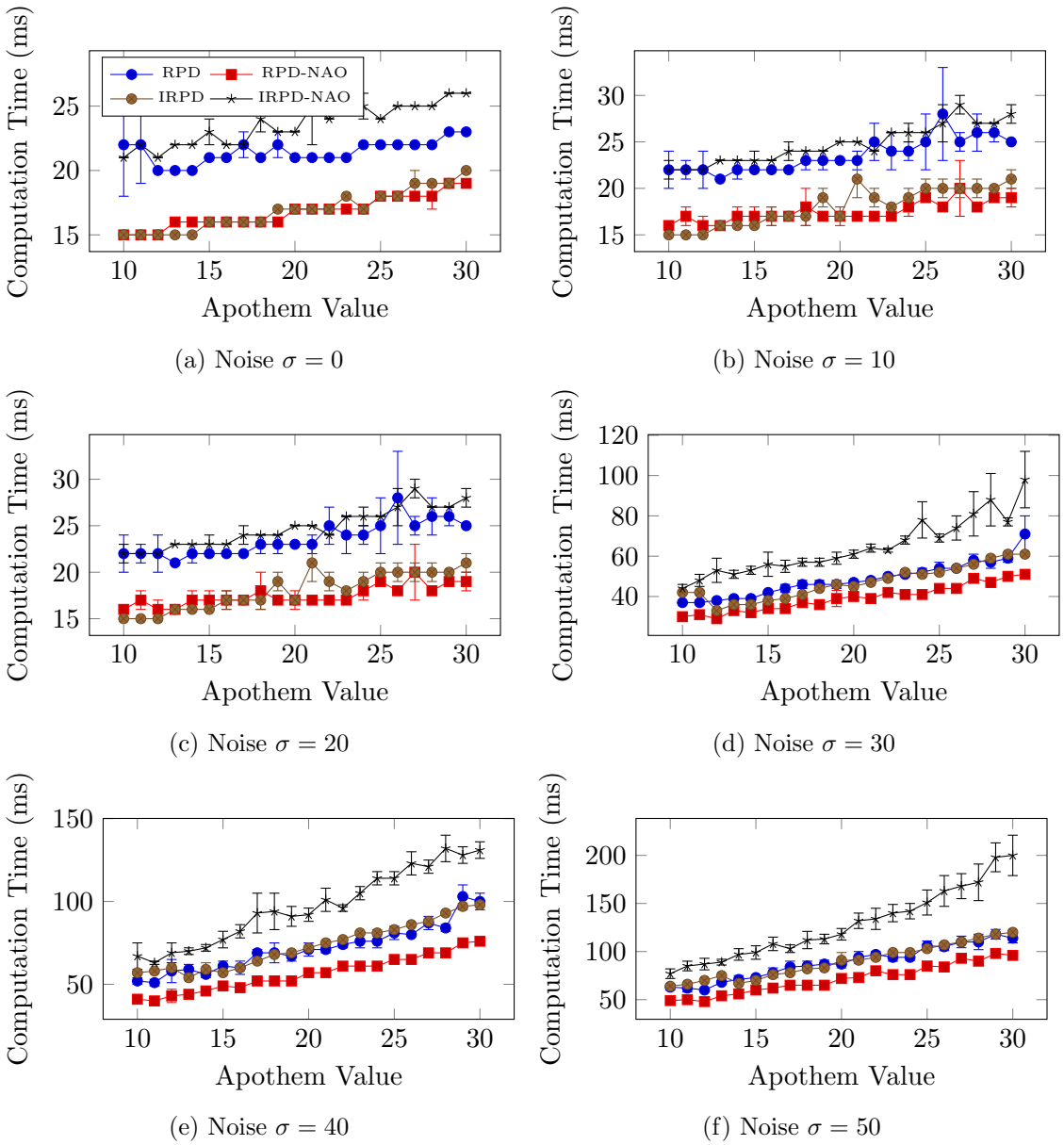


(e) Noise $\sigma = 40$



(f) Noise $\sigma = 50$

Figure 39: Octagon Detector False Positive Rates with Noise from $\sigma = 0$ to $\sigma = 50$



(a) Noise $\sigma = 0$

(b) Noise $\sigma = 10$

(c) Noise $\sigma = 20$

(d) Noise $\sigma = 30$

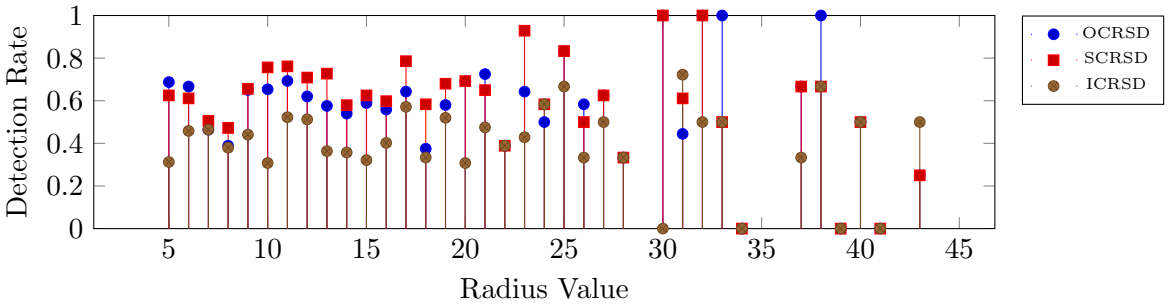
(e) Noise $\sigma = 40$

(f) Noise $\sigma = 50$

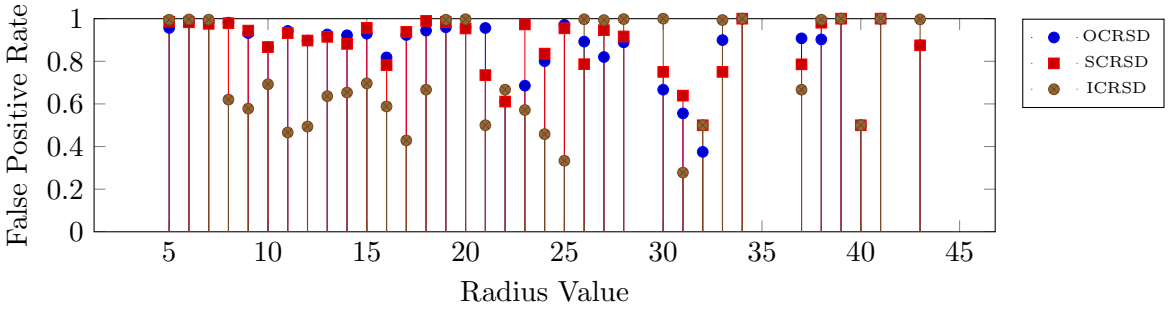
Figure 40: Octagon Detector Computation Time (ms) with Noise from $\sigma = 0$ to $\sigma = 50$

A.2 Traffic Sign Dataset Evaluation

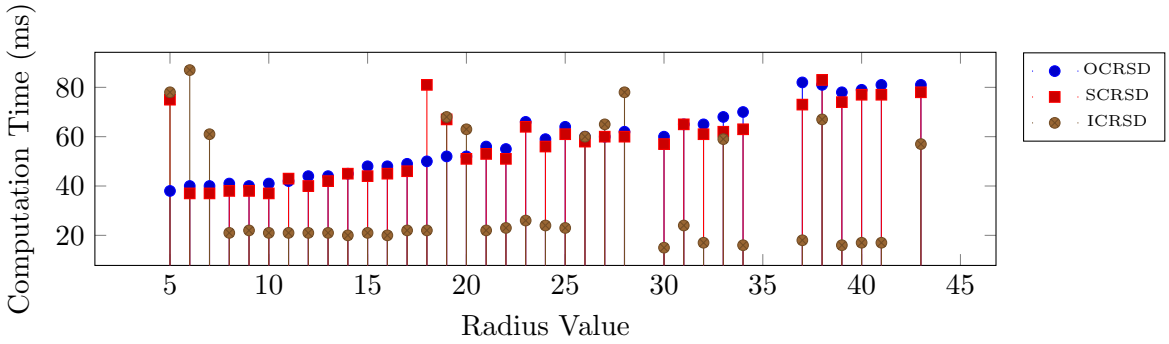
A.2.1 Circle Detector



(a) Detection Rate



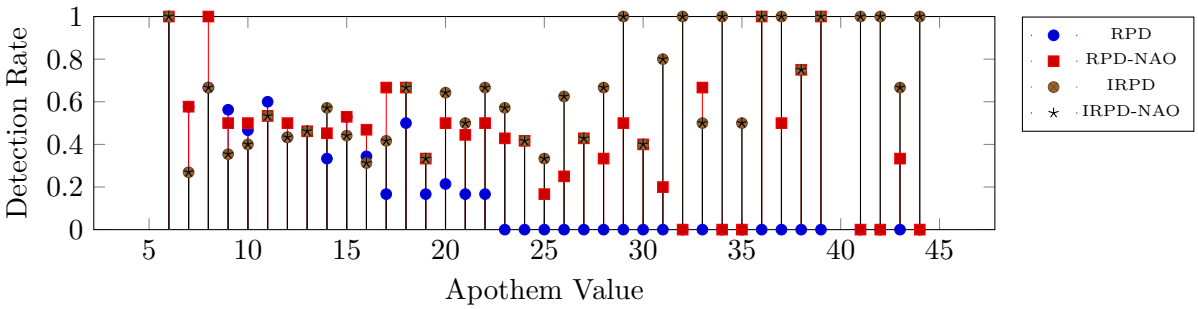
(b) False Positive Rate



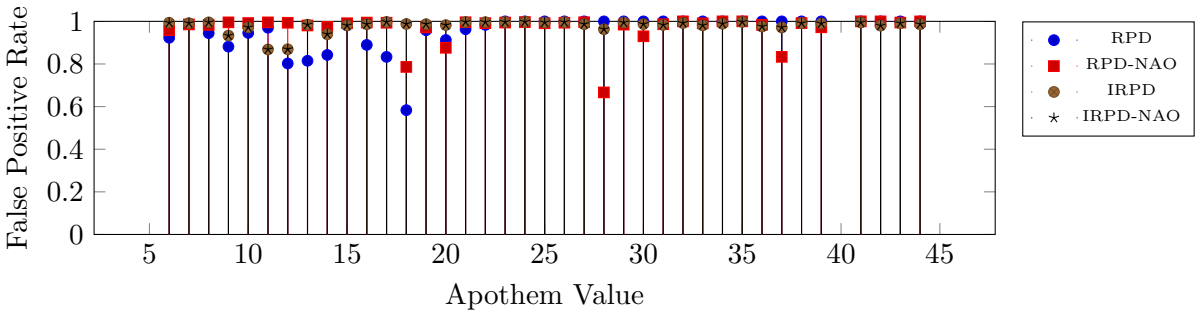
(c) Computation Times (ms)

Figure 41: Circle Detector Performance under the GTSDB by Circle radius

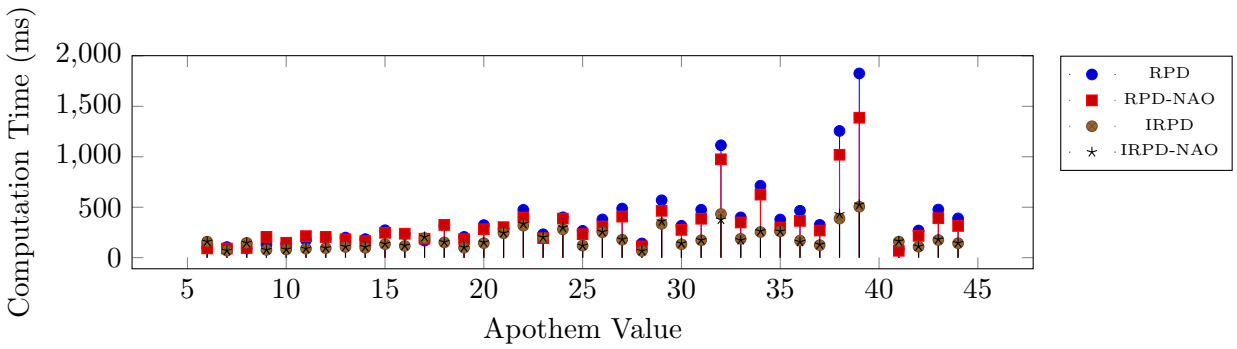
A.2.2 Triangle Detector



(a) Detection Rate



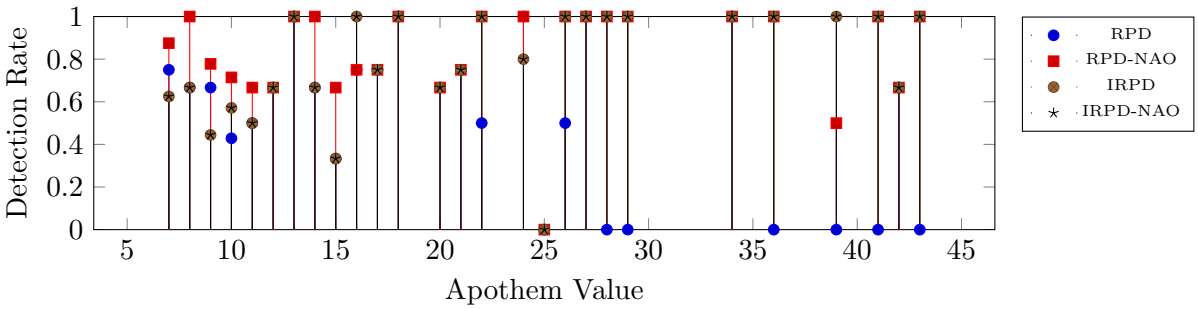
(b) False Positive Rate



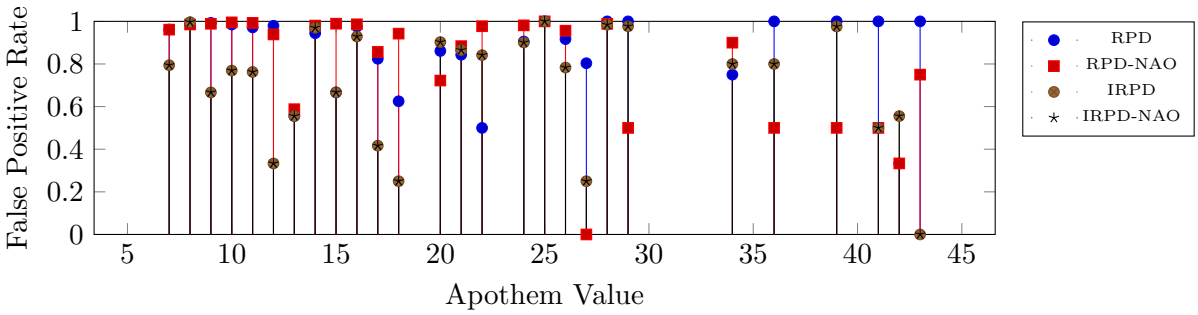
(c) Computation Times (ms)

Figure 42: Triangle Detector Performace under the GTSDB by Apothem.

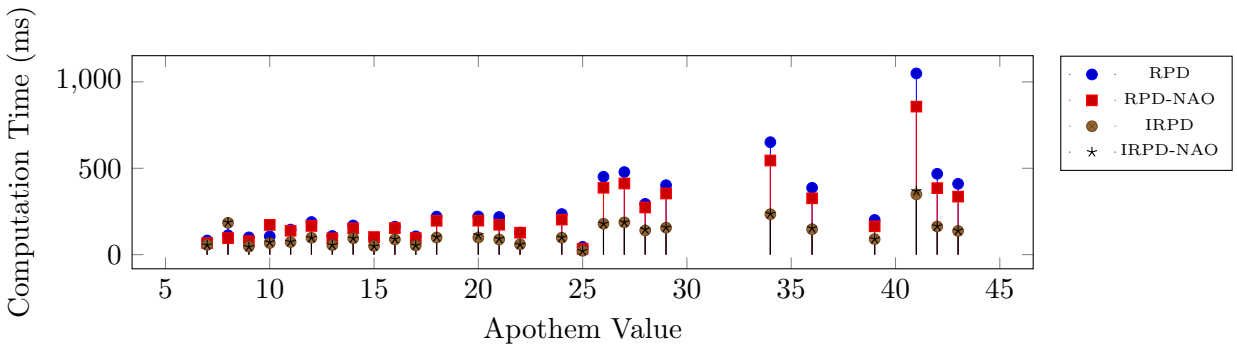
A.2.3 Square Detector



(a) Detection Rate



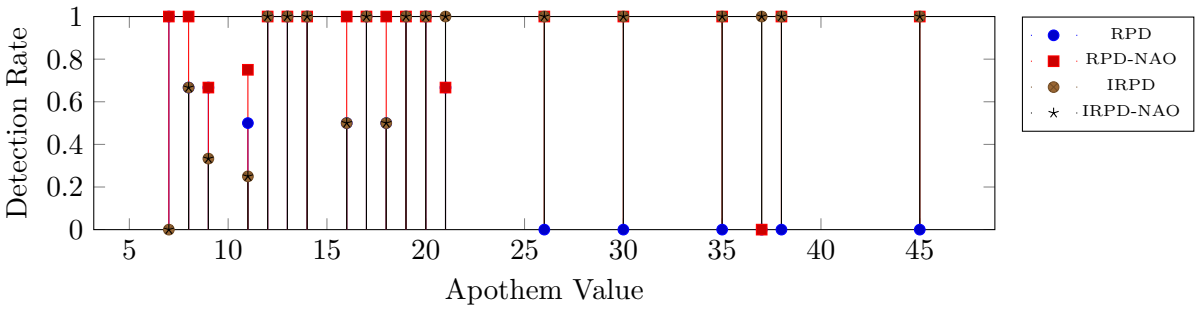
(b) False Positive Rate



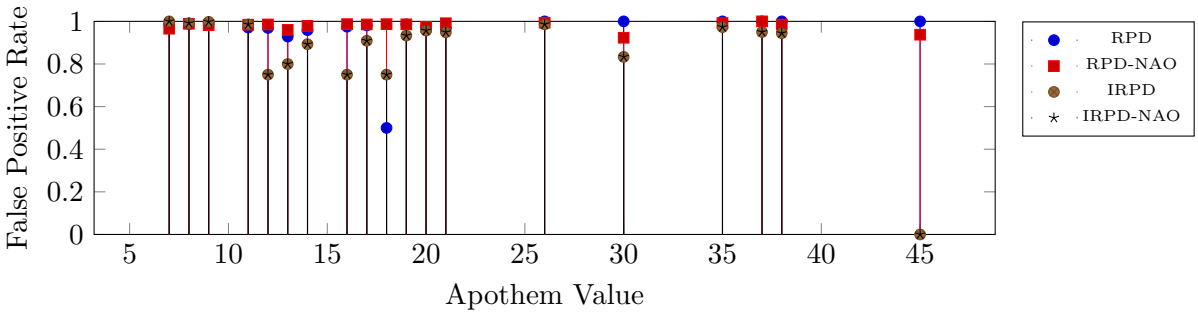
(c) Computation Times (ms)

Figure 43: Square Detector Performance under the GTSDB by Apothem.

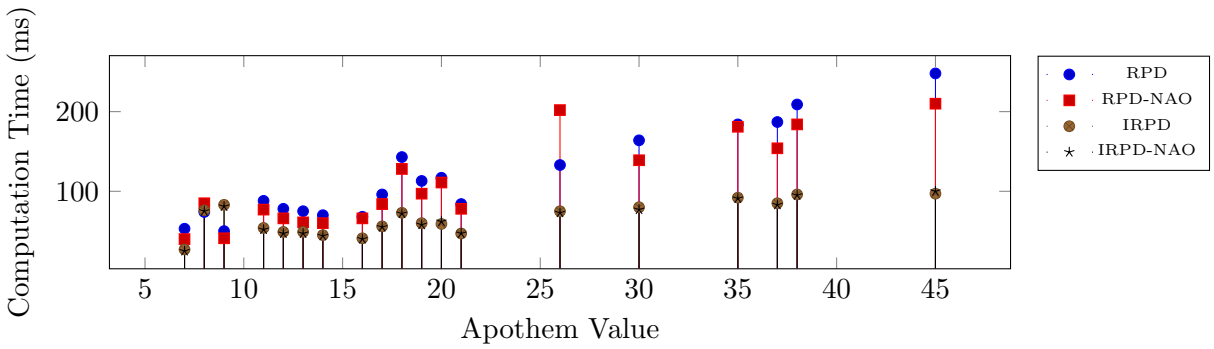
A.2.4 Octagon Detector



(a) Detection Rate



(b) False Positive Rate

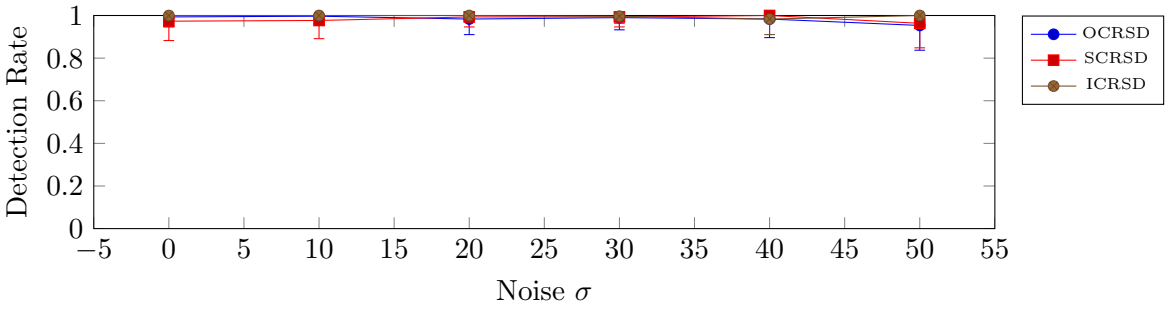


(c) Computation Times (ms)

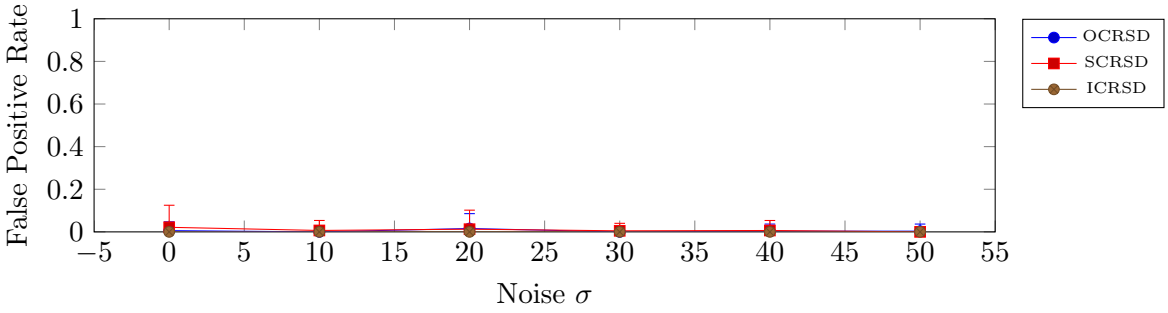
Figure 44: Octagon Detector Performance under the GTSDB by Apothem.

A.3 Multiple Radius Evaluation

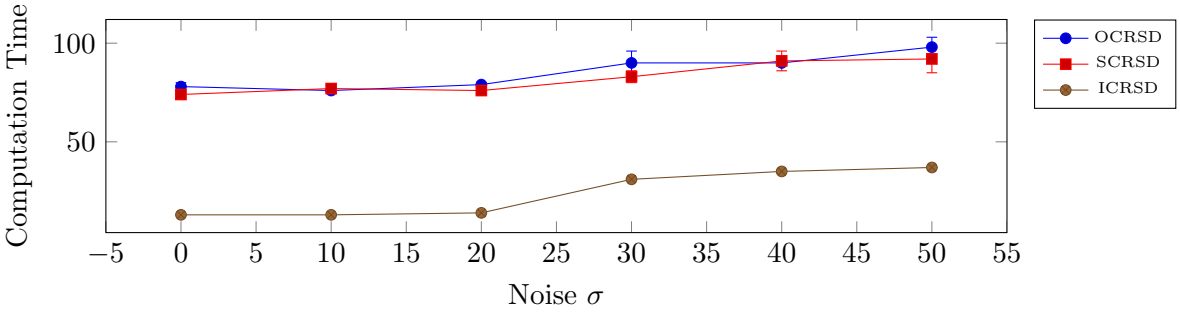
A.3.1 Circle Detector



(a) Detection Rates



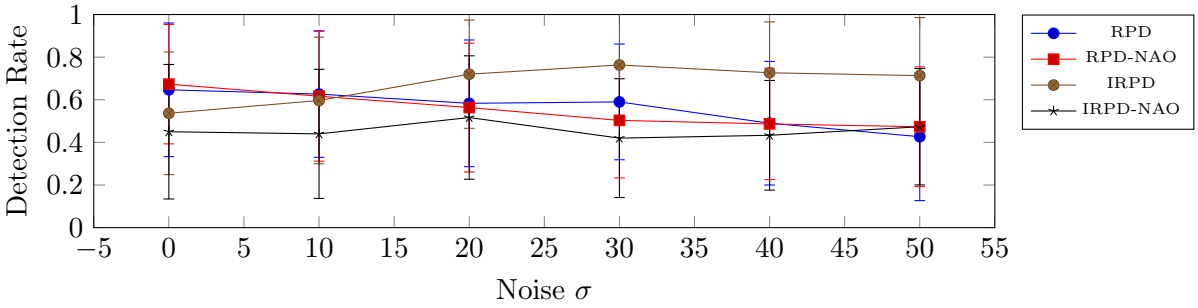
(b) False Positive Rates



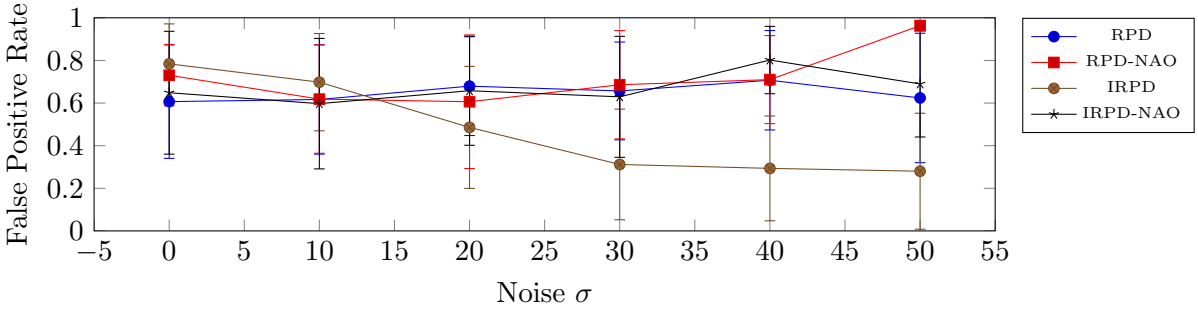
(c) Computation Times

Figure 45: Multiple Radius Evaluation for the Circle Detector

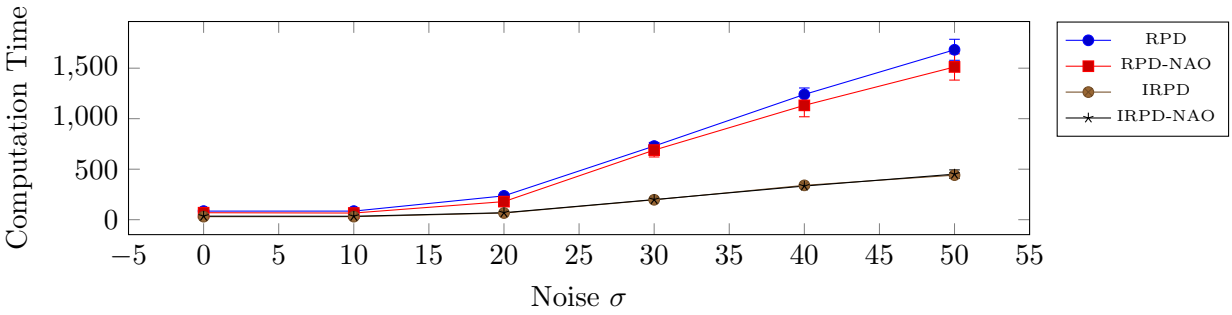
A.3.2 Triangle Detector



(a) Detection Rates



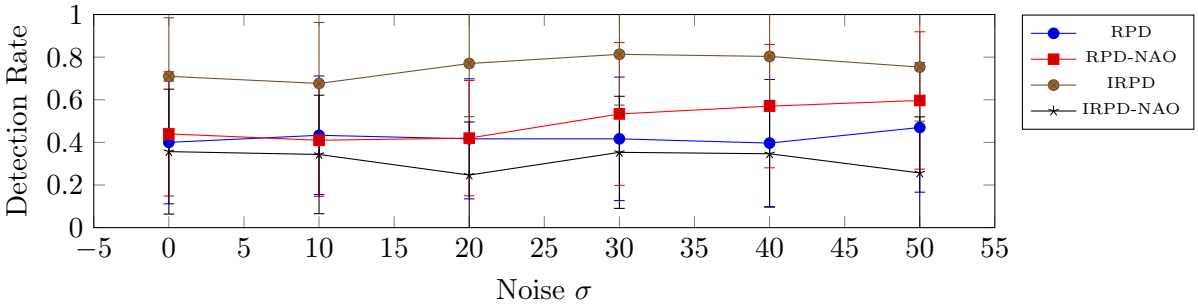
(b) False Positive Rates



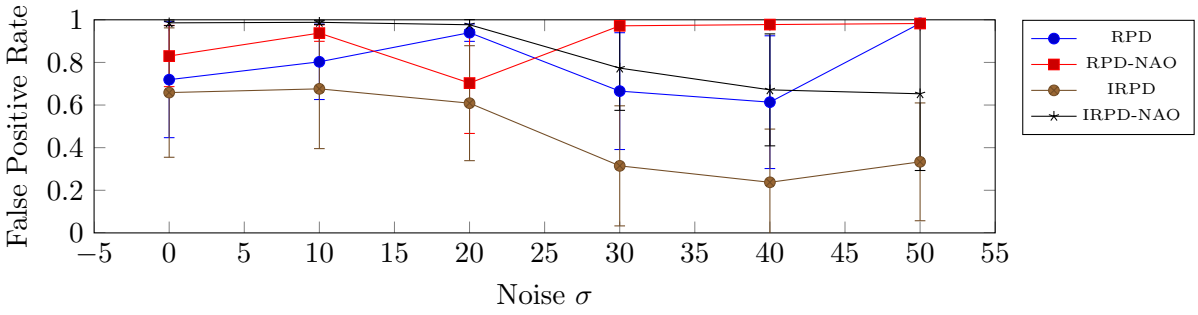
(c) Computation Times

Figure 46: Multiple Radius Evaluation for the Triangle Detector

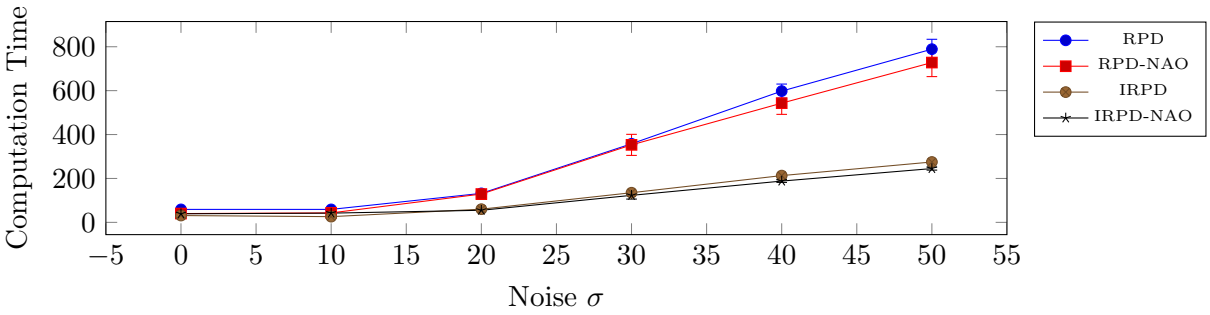
A.3.3 Square Detector



(a) Detection Rates



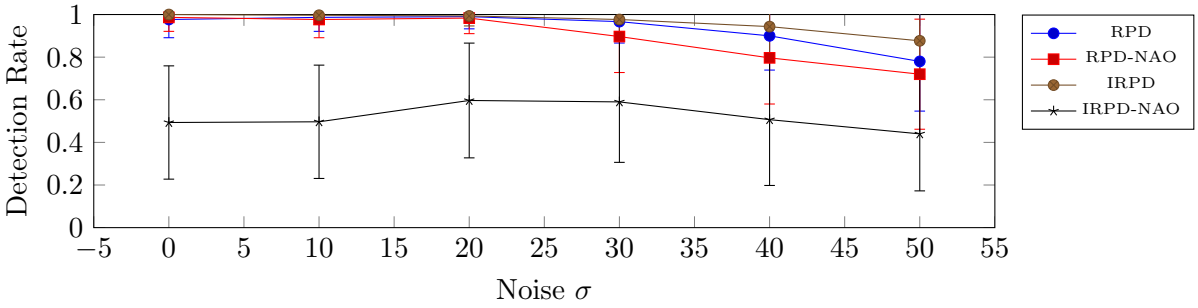
(b) False Positive Rates



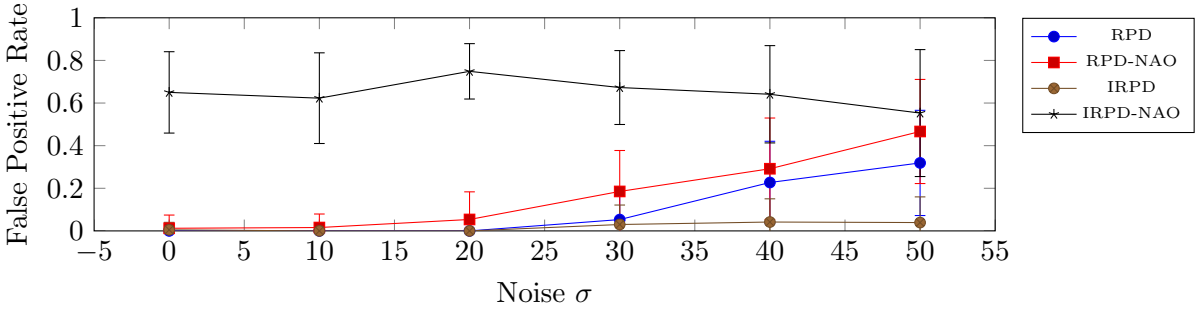
(c) Computation Times

Figure 47: Multiple Radius Evaluation for the Square Detector

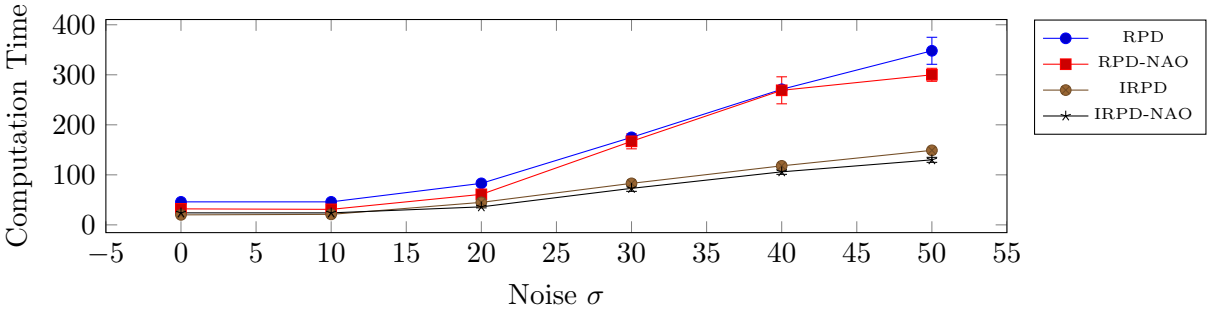
A.3.4 Octagon Detector



(a) Detection Rates



(b) False Positive Rates



(c) Computation Times

Figure 48: Multiple Radius Evaluation for the Octagon Detector