In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), San Francisco, CA, 2010.

Fast Polygonal Integration and Its Application in Extending Haar-like Features to Improve Object Detection

Minh-Tri Pham Yang Gao
Surrey Space Centre
University of Surrey, England
{t.pham, yang.gao@}@surrey.ac.uk

Viet-Dung D. Hoang Tat-Jen Cham
School of Computer Engineering
Nanyang Technological University, Singapore
hoangducvietdung@gmail.com astjcham@ntu.edu.sg

Abstract

The integral image is typically used for fast integrating a function over a rectangular region in an image. We propose a method that extends the integral image to do fast integration over the interior of any polygon that is not necessarily rectilinear. The integration time of the method is fast, independent of the image resolution, and only linear to the polygon's number of vertices. We apply the method to Viola and Jones' object detection framework, in which we propose to improve classical Haar-like features with polygonal Haar-like features. We show that the extended feature set improves object detection's performance. The experiments are conducted in three domains: frontal face detection, fixed-pose hand detection, and rock detection for Mars' surface terrain assessment.

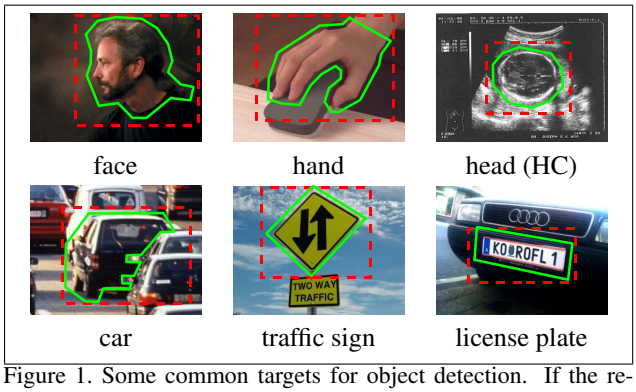


Figure 1. Some common targets for object detection. If the region of interest is known, it may be better to use a polygon than a rectangle to approximate the domain of integration, as more unwanted area can be avoided. With this work, fast integration over a polygonal region is possible, in time independent of the image resolution.

1. Introduction

The integral image [4] is a linear transform of an image [18], that allows integration of a function over the interior of any axis-aligned[1] rectangle in the image in constant time. It has been widely used in many computer vision applications, such as (in chronological order): template matching [11], Haar-like features [24], integral histograms [20], spatial pyramid matching [10], region covariances [23], histograms of oriented gradients (HOGs) [28], scale-invariant features (SIFTs) [7], speeded up robust features (SURFs) [2], bilateral filtering [21], and medical anatomy detection [3].

Many applications in computer vision involve one or both of the following tasks: 1) extracting information from a specified region, and 2) comparing two regions of the same shape. However, the shape is restricted to rectilinear, because that is what an integral image can only address. Conversely, objects in real life do not have rectilinear shapes. Integration over a rectangle's interior may incur some noise from unwanted regions (*e.g.*, background or occlusion). Better results can be achieved if a polygonal shape is used for integration. Figure 1 shows some common targets often used in object detection.

Some extensions of the integral image in which edges are rotated were proposed in [1, 3, 5, 13, 16]. In these methods, the image is physically or virtually rotated until the edges of the region of integration are aligned vertically or horizontally, and an integral image is pre-computed for each rotated image. In another direction [25], which is a straightforward extension of the integral image, the domain of integration is generalized from a rectangle to an axis-aligned polygon.

The core idea of the integral image is to decompose a rectangular integral or a rectilinear polygonal integral into a finite weighted sum of pre-computed rectangular integrals. However, this decomposition imposes a strong restriction to the polygonal integral of interest: *the set of edge slopes of the polygon must not exceed 2*. In other words, it is impossible for the integral image and its variants [1, 3, 5, 13, 25]

[1] *i.e.*, parallel to the axes of the image coordinate system

to integrate over the interior of a polygon that has at least 3 different edge slopes, a *triangle* for example.

In this paper, we present an integration method that eliminates the above-mentioned restriction. The set of edge slopes can be specified by the user. The speed of our polygonal integration method is as fast as that of the integral image, in time *independent of the image resolution* and *only linear to the polygon's number of vertices*. To achieve this speed, we replace the core idea of the integral image with a different idea. We show that a polygonal integral can be decomposed into a finite weighted sum of pre-computed *right-triangular integrals*. We also show how these right-triangular integrals can be pre-computed fast, using dynamic programming.

An extensive set of applications directly benefit from fast polygonal integration[10, 11, 20, 21, 23, 28]. By replacing a rectangular region with a polygonal region, one may obtain the same kind of information, but with better accuracy. We restrict ourselves to one of the most popular applications of the integral image: Haar-like features. Using the object detection method of Viola and Jones [24] as the base framework, we show that by extending Haar-like features from rectangular features to polygonal features, an object detector's performance is improved.

The rest of the paper is organized as follows. Section 2 reviews work related to the integral image. Section 3 describes our fast polygonal integration method. Section 4 gives experimental results in object detection with Haar-like features extended to polygonal Haar-like features. Section 5 summarizes the work and gives conclusions.

2. Related Work

The history of integral image techniques is arguably related to that of Haar-like features. The integral image was first introduced in computer graphics in 1984 [4], but was not widely used in the computer vision community until its prominent use in Viola and Jones' real-time object detection framework twenty years later [24]. Initially, the integral image was used to evaluate, in constant time, the sum of pixels in a rectangle of a Haar-like feature.

Given an image $\mathbf{I}$ of size $M \times N$, let $\mathbf{I}[x, y]$ denote the pixel intensity at location (x, y), where $0 \leq x < M$ and $0 \leq y < N$. Here, a pixel at location (x, y) is a unit square with homogeneous intensity, and with its bottom-left corner[2] located at coordinate (x, y) of the image coordinate system. An integral image is an image $\mathbf{J}$ such that for every pixel location $(x, y) \in \mathbb{T} = \{0, 1, \ldots, M\} \times \{0, 1, \ldots, N\}$, $\mathbf{J}[x, y] = \sum_{x' < x} \sum_{y' < y} \mathbf{I}[x', y']$. When $\mathbf{J}$ has been computed, integration over the interior of any rectangle with bottom-left corner at (x, y), width w, and height h, can be computed with only 4 memory references:

$$\sum_{x \leq x' < x+w} \sum_{y \leq y' < y+w} \mathbf{I}[x', y'] = \mathbf{J}[x + w, y + h] + \mathbf{J}[x, y] - \mathbf{J}[x + w, y] - \mathbf{J}[x, y + h]. \quad (1)$$

[2]We assume location $(0, 0)$ is at the bottom-left corner of an image.

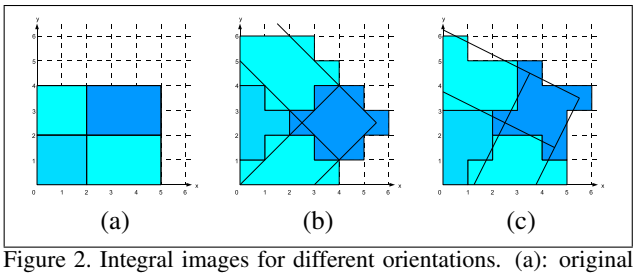


Figure 2. Integral images for different orientations. (a): original axis-aligned integral image. (b): 45°tilted integral image [13]. (c): 26.57°rotated integral image [1, 5].

Naturally, (1) is generalizable to address axis-aligned polygons [25]. A simplification of the integral image also exists [8]. In [13], Lienhart and Maydt proposed a 45°tilted integral image (figure 2b) to deal with 45°rotated rectangles. This extended the set of Haar-like features, and improved the performance of Viola and Jones' rapid object detection framework. Continuing along this line, Barczak *et al.* [1] and Du *et al.* [5] published similar techniques to address Haar-like features rotated by 26.57°(figure 2c). Messom and Barczak [16] also generalized the set of angles to what they defined as integer-rotated $1 : n$ and $n : 1$ angles. Instead of virtually rotating the image, Carneiro *et al.* [3] physically rotated the image, and computed an integral image for the rotated image.

Although these techniques build integral images from virtually or physically rotated images, they use the core idea of the integral image in decomposing a rectangular integral. The decomposition is based on pre-computed axis-aligned rectangular integrals, thus limiting the domain of integration to be rectilinear, and with at most 2 edge slopes. In addition, these methods ignore pixels along an edge which may not reside fully inside or outside a rectangle due to rotation. Thus, some approximation to the true value of an integral is introduced.

Our integration method removes the rectilinear restriction and increases the number of edge slopes to as many as the user wishes. Also, it does not ignore partial pixels as in existing methods, as explained in what follows.

3. Fast Polygonal Integration

3.1. Problem Formulation

Let I be a 2D intensity function defined as $I(x, y) = \mathbf{I}[\lfloor x \rfloor, \lfloor y \rfloor]$, where $\lfloor x \rfloor$ and $\lfloor y \rfloor$ are the largest integers not greater than real numbers x and y, respectively. If either $x \notin [0, M)$ or $y \notin [0, N)$, we assume $I(x, y) = 0$.

Algorithm 1 Fast Polygonal Integration

Require:
Pre-computed integrals $G = \{g(x, y, d)\}$ {see (3)}
Polygon P with n vertices
Ensure: $f(P)$
$S = 0$;
for $i = 1$ to n **do**
  if $x_i < x_{i+1}$ **then**
    $S+ = |(g(x_{i+1}, y_{i+1}, d_i) - g(x_i, y_i, d_i)|$;
  else if $x_i > x_{i+1}$ **then**
    $S- = |(g(x_{i+1}, y_{i+1}, d_i) - g(x_i, y_i, d_i)|$;
  end if
end for
return S;

We represent a polygon by a list of vertex coordinates in clockwise order. For example, in figure 3a, $P = [(1, 1), (2, 6), (6, 5), (3, 3), (5, 2)]$. We assume polygon P, the domain of integration, is non-self-intersecting and has integer-coordinate vertices. In addition, we assume that the slope of every edge of P belongs to a pre-defined set $\mathcal{D}$. Let $R(P)$ be the interior region of P. Our goal is to evaluate an integral of function I over $R(P)$, defined as:

$$f(P) = \iint_{R(P)} I(x, y)dxdy. \tag{2}$$

3.2. Fast Polygonal Integration

As mentioned in section 1, the integral image decomposes a rectangular integral into a linear combination of pre-computed rectangular integrals. This restricts the number of edge slopes to exactly 2. To eliminate this restriction, consider the following right-triangular integral:

$$g(x, y, d) = \begin{cases} f([(x, y), (x, 0), (-\infty, 0)]) & \text{if } d = 0 \\ 0 & \text{if } d = \pm\infty \\ f([(x, y), (x, 0), (x - \frac{y}{d}, 0)]) & \text{otherwise} \end{cases} \tag{3}$$

where $x \in \{0, \ldots, M\}$, $y \in \{0, \ldots, N\}$, and $d \in \mathcal{D}$. The integration domain corresponds to a right triangle with one corner at point (x, y), one cathetus on the x-axis, and the hypotenuse with slope d. An example of such a triangle is illustrated in figure 3b. Note that, when $d = 0$, the triangle is an axis-aligned rectangle cornered at $(0, 0)$ and (x, y), as the image is assumed to be zero for negative coordinates. When $d = \pm\infty$, the triangle is a vertical line segment.

The decomposition of a polygonal integral into right-triangular integrals is done in two steps. Given a polygon P with n vertices: $A_1(x_1, y_1)$, $A_2(x_2, y_2)$, ..., $A_n(x_n, y_n)$, let $B_1, B_2, \ldots, B_n$ be the projections of $A_1, A_2, \ldots, A_n$ onto the x-axis respectively (figure 3a). In the first step,

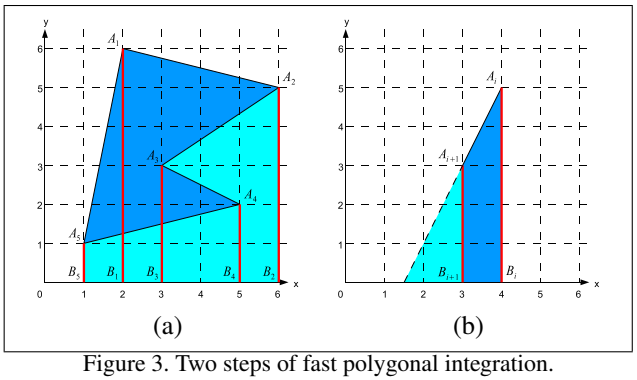


Figure 3. Two steps of fast polygonal integration.

using the trapezoidal rule, $f(P)$ is decomposed into:

$$f(P) = \sum_{i=1}^{n} c_i f(Q_i), \tag{4}$$

where (assuming $A_{n+1} = A_1$):

$$(c_i, Q_i) = \begin{cases} (1, [A_i, A_{i+1}, B_{i+1}, B_i]) & \text{if } x_i < x_{i+1} \\ (0, []) & \text{if } x_i = x_{i+1} \\ (-1, [A_{i+1}, A_i, B_i, B_{i+1}]) & \text{if } x_i > x_{i+1} \end{cases} \tag{5}$$

Here, Q_i represents a trapezoid and c_i represents the sign of the integral over Q_i. When $x_i = x_{i+1}$, $f(Q_i)$ becomes zero regardless of the value of c_i. Thus, we set c_i to 0.

In the second step, for each non-empty trapezoid Q_i, let d_i be the slope of edge A_iA_{i+1}. We further rewrite $f(Q_i)$ as the absolute difference between two right-triangular integrals, as depicted in figure 3b:

$$f(Q_i) = |(g(x_{i+1}, y_{i+1}, d_i) - g(x_i, y_i, d_i)|. \tag{6}$$

Therefore, a polygonal integral with n vertices is decomposed into a linear combination of at most $2n$ right-triangular integrals. In the next section, we show how to quickly pre-compute the set $G = \{g(x, y, d)\}$ of right-triangular integrals for all possible values of x, y, and $d \in \mathcal{D}$. With these integrals pre-computed, any polygonal integration with edge slopes in $\mathcal{D}$ can be done in not more than $2n$ memory references.

The pseudocode of the algorithm is summarized in algorithm 1. Also note that when $\mathcal{D} = \{0, \pm\infty\}$, this technique is equivalent to the axis-aligned integral image technique.

3.3. Pre-processing

The key issue in the pre-processing step is that different slopes have different difficulties in integration. Integration along horizontal lines and vertical lines are easy, as they are parallel to the axes. A line with an *irrational* slope, however, is hard to integrate, because at each pixel intersected by the line, the partition pattern is always different. Unless some approximation to the pattern is introduced, whichever

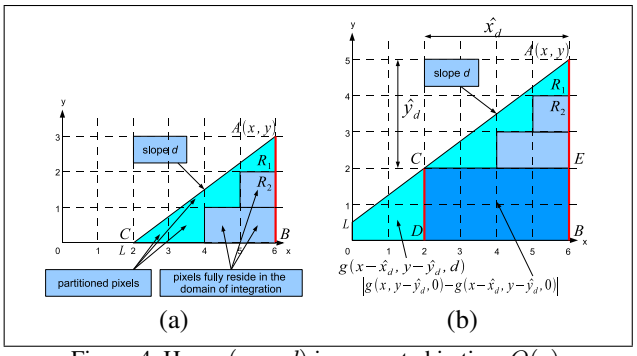


Figure 4. How $g(x, y, d)$ is computed in time $O(r)$.

algorithm that integrates along this line must visit every intersected pixel. Hence, the time complexity to integrate is at least linear to the image's width or height.

On the other hand, the slope of a polygon's edge with integer vertices is always rational. More importantly, a line with a *rational* slope produces a *finite* number of pixel partition patterns. Consider, for example, a slope of positive integer value n. When traveling along a line with this slope, the pixel partition pattern repeats itself at every n intersected pixels. In fact, the repetition in pixel partition pattern is the key factor to our solution for fast computing the integrals in G. Generalizing the above example, we notice that for any rational slope $d = n/m$ ($n, m \in \mathbb{Z}$ and $m \geq 0$), the integer vector $\mathbf{v}_d = (m, n)/\gcd(|n|, |m|)$ represents an interval in 2D such that the partition patterns is periodic. In other words, the partition pattern at any pixel location (x, y) is the same that at the shifted location $(x, y) + \mathbf{v}_d$.

We now describe how an integral $g(x, y, d)$ for given values of x, y, and d can be computed quickly. By the definition in (3), when $d = \pm\infty$, $g(x, y, d) = 0$. The case that $d = 0$ is trivial, we simply compute an axis-aligned integral image. The problem is when $d \neq 0$. In this case, let L be the line with slope d crossing the 2D integer point (x, y). We choose $(\hat{x}_d, \hat{y}_d) = \text{sign}(d)\mathbf{v}_d$ so that $\hat{y}_d$ is positive. Our focus is at the line segment from the 2D integer point $A = (x, y)$ to the 2D integer point $C = (x - \hat{x}_d, y - \hat{y}_d)$, as illustrated in figure 4. Let r_d be the number of pixels being partitioned by line segment AC. It is provable that $r_d = |\hat{x}_d| + |\hat{y}_d| - 1$.

When C lies outside the image, *i.e.*, $(x - \hat{x}_d, y - \hat{y}_d) \notin \mathbb{T}$, both catheti of the right triangle domain of $g(x, y, d)$ have lengths not greater than r_d (figure 4a). We partition this triangle into two regions R_1 and R_2, where R_1 contains all the pixels partially partitioned by L, and R_2 contains all the pixels that are fully inside the triangle. Since the number of pixels in R_1 is r_d, integration over R_1 is *equivalent to computing a dot product* in a r_d-dimensional space, between the intensities of the partial pixels and their corresponding proportional areas covered by the triangle. One can *rasterize* R_2 (either vertically or horizontally) into no more than r_d unit-wide rectangles, and use an axis-aligned integral image (*e.g.*, $g(\cdot, \cdot, 0)$) to integrate over each of them. This summation step can also be implemented as a dot product in a space of at most r_d dimensions. Overall, the integral $g(x, y, d)$ is computed in time $O(r_d)$.

Algorithm 2 Pre-processing

Require: Image $\mathbf{I}$
Ensure: Integrals $G = \{g(x, y, d)\}$ {see (3)}

```
Allocate 3D array G, where G[x, y, d] is intended to hold the value of g(x, y, d);
Assign G[·, ·, 0] = J, the integral image of I;
for all d ≠ 0 do
    for y = 0 to N do
        for x = 0 to M do
            if (x − x̂_d, y − ŷ_d) ∉ T then
                G[x, y, d] = 0;
                Let (x_C, y_C) be the point of intersection between line L and the x-axis;
            else
                (x_C, y_C) = (x − x̂_d, y − ŷ_d);
                G[x, y, d] = |G[x, y_C, 0] − G[x_C, y_C, 0]| + G[x_C, y_C, d];
            end if
            Partition the intersected region between triangle [(x, y), (x, y_C), (x_C, y_C)]'s interior and [0, M) × [0, N) into R1 and R2;
            Integrate I over R1 and R2 in time O(r_d), and add the sum to G[x, y, d];
        end for
    end for
end for
return G;
```

When $(x - \hat{x}_d, y - \hat{y}_d) \in \mathbb{T}$, we can define $g(x, y, d)$ ($d \neq 0$) in a recursive manner:

$$\begin{aligned} g(x, y, d) \quad = \quad & f([(x - \hat{x}_d, y - \hat{y}_d), (x, y), (x, y - \hat{y}_d)]) \\ & + |g(x, y - \hat{y}_d, 0) - g(x - \hat{x}_d, y - \hat{y}_d, 0)| \\ & + g(x - \hat{x}_d, y - \hat{y}_d, d). \end{aligned} \tag{7}$$

With reference to figure 4b, the first term corresponds to the integral over triangle CAE, the absolute difference of the middle term corresponds the integral over rectangle $CEBD$, and the last term corresponds to the triangular integral cornered at C and with slope d. Integration over triangle CAE is similar to the previous case. If $g(\cdot, \cdot, 0)$ and $g(x - \hat{x}_d, y - \hat{y}_d, d)$ are previously computed, the total integration time is also $O(r_d)$.

Finally, all the integrals in G are computed using *dynamic programming* over x, y, and d, exploiting the recursion in (7). Integrals $g(\cdot, \cdot, 0)$ are computed first, so that they can be used for computing subsequent integrals.

Factor y iterates in ascending order, so that at any time, $g(x - \hat{x}_d, y - \hat{y}_d, d)$ is computed before $g(x, y, d)$ (*i.e.*, $\hat{y}_d > 0$). This pre-processing step's pseudocode is shown in algorithm 2.

The pre-processing step has space complexity $O(MN|\mathcal{D}|)$, and time complexity $O(MN|\mathcal{D}|r_{\mathcal{D}})$, where $r_{\mathcal{D}} = \max_{d\in\mathcal{D}} r_d$. In practice, the number of slopes required by an application is small (typically $|\mathcal{D}| < 10$) and most of the commonly used slopes have small r_d (typically $r_{\mathcal{D}} < 5$). Methods that use the axis-aligned integral image only require 2 edge slopes: $\mathcal{D} = \{0, \pm\infty\}$ and $r_{\mathcal{D}} = 1$. In the 45°titled integral image [13], $\mathcal{D} = \{\pm 1\}$ and $r_{\mathcal{D}} = 1$. In the 26°rotated integral images [1, 5], $\mathcal{D} = \{-0.5, 2\}$ or $\mathcal{D} = \{0.5, -2\}$ and $r_{\mathcal{D}} = 2$. With only 4 slopes $\{0, \pm 1, \infty\}$ (and still $r_{\mathcal{D}} = 1$), one can integrate any polygon whose edge orientations are multiples of 45°, which is arguably sufficient for methods that extract 2D rotation information like the SIFT descriptor [14].

In addition, it is provable that the space complexity $O(MN|\mathcal{D}|)$ is *optimal*[3], which means we cannot reduce the size of the array G in algorithm 2 further. In this regard, the time complexity can be considered as nearly optimal, since factor $r_{\mathcal{D}}$ is a relatively small number.

An implementation of our method will be available at the open-source PyOpenCV[4] project.

4. Experiments on Polygonal Haar-like Features

We have shown a method that can integrate over a polygonal region fast. As a generalization of the integral image, this work can be readily applied to a number of methods in computer vision [2, 7, 10, 11, 20, 21, 23, 24, 28]. However, experimenting on all of them is outside the paper's scope. In the remaining of the paper, we show how the work improves object detection, when rectangular Haar-like features are extended to polygonal Haar-like features. We focus on three domains: frontal face detection, fixed-pose hand detection, and rock detection for Mars' surface terrain asssessment. Due to space limitation, we mainly report experiments in face detection. All of our experiments were conducted on an Intel(R) Core(TM)2 @ 2.67GHz Linux platform with 2GB of RAM, using a single thread.

4.1. Face Detection

In face detection, Haar-like features are the most widely used features to date. Currently, there are two trends that extend Haar-like features to improve face detection. One trend is by rotating Haar-like features, as described in section 2. The other is by adding more rectilinear shapes [12, 18] or by combining a few rectangular features into one [17]. Using the proposed method, we extended the feature pool to a new set of parameterized polygonal Haar-like features, as shown in figure 5. The parameters allowed the features to shift and scale. Flipping and rotations by multiples of 90°of the features were included. The new set extends classical Haar-like features in both shape and in-plane orientation. Note that all the edge orientations of the polygonal features are multiples of 45°, which means $\mathcal{D} = \{0, \pm 1, \pm\infty\}$.

We conducted two experiments in face detection. A col-

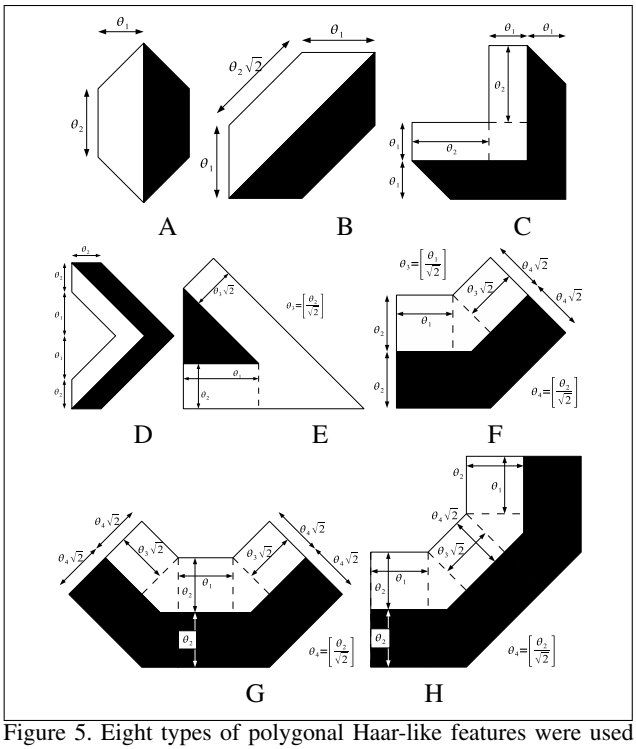


Figure 5. Eight types of polygonal Haar-like features were used in addition to Viola-Jones' four types of Haar-like features [24]. They were parameterized by $(x, y, \theta_1, \theta_2, [\theta_3])$. Scaling and translation were naturally incorporated in these parameters. Features were generated from these types, rotated by multiples of 90°, and mirrored. After all duplicate features were removed, a total of 210,366 features formed a polygonal feature pool.

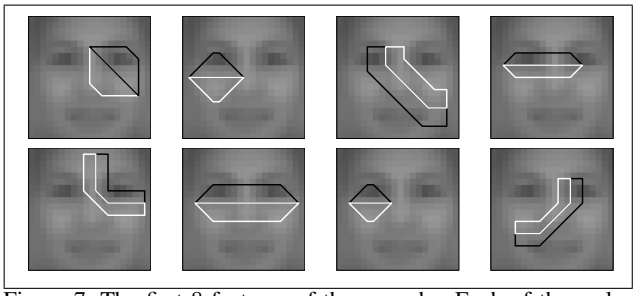
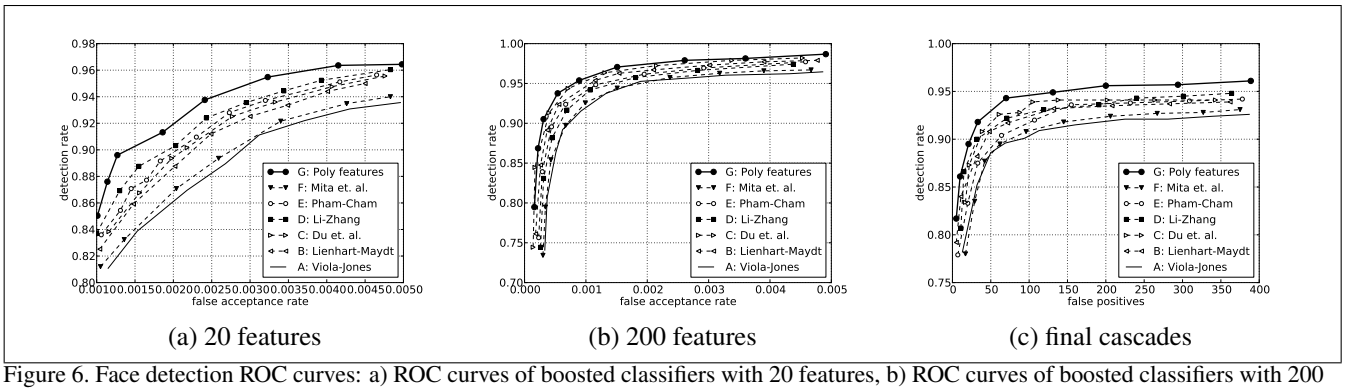


Figure 7. The first 8 features of the cascade. Each of them also acted as a non-face rejector. For illustration, the background is the mean of all the face patches in the training set.

[3] A proof for this claim is available in the supplementary material. We omit it here since it is not too relevant to the discussion.

[4] http://code.google.com/p/pyopencv/



(a) 20 features (b) 200 features (c) final cascades

Figure 6. Face detection ROC curves: a) ROC curves of boosted classifiers with 20 features, b) ROC curves of boosted classifiers with 200 features, and c) ROC curves of the final cascades.

Method/pool	A	B	C	D	E	F	G
No of types	4	14	15	3	19	4	12
No of features (in thousands)	50	86	113	427	296	50	210

Table 1. Some statistics of the feature pools studied in section 4.1.

lection of 8,000 faces was gathered, containing 1,521 face images from the BioID Face Database[5], and about 6,500 randomly selected frontal face images from Xiao *et al.* [27]'s face dataset. We built a generator that selected known face locations randomly, and resized them down to a base patch resolution of 20×20 pixels, with some pertubation bias [26]. We used the same collection of a few thousands large images containing no faces of Wu *et al.* [26]. For testing, we used the MIT+CMU standard test set, consisting of 130 grayscale images with 507 frontal faces [22]. Our main interest was to study the discriminant power of feature extraction when Haar-like features were extended.

The first experiment was done as follows. Using Viola-Jones' framework, we trained a cascade of 6 boosted classifiers that rejected 97% non-face patches at a cost of losing 1% face patches. We collected 5,000 true positives and 5,000 false positives of the cascade to form a training set that respresented a *hard* classification problem. Using the training set and the asymmetric boosting method of Pham *et al.* [19], we trained multiple boosted classifiers with the same number of features. We varied the λ parameter of the method (*i.e.*, a trade-off between FAR and FRR) to obtain different ROC operating points[6]. The idea of training a cascade was to avoid a trivial case in which the resultant boosted classifiers would contain so few features that we could not tell their differences [17, 19]. We considered 7 different methods for learning a weak classifier, whose differences were mainly at their feature pools: **A**) Viola and Jones' method [24] using a pool of 50,000 features (our patch size was slightly bigger than theirs), **B**) [24]'s method with the pool added with 45°tilted features [13], **C**) [24]'s method with the pool added with 26°rotated features [5], **D**) [24]'s method with Li and Zhang [12]'s feature pool of steerable features, **E**) Pham and Cham's fast weak-learning method [18] with a feature pool of 300,000 rectilinear features, **F**) Mita *et al.*'s co-occurent features learning method [17] using [24]'s pool, and **G**) [24]'s method using our polygonal feature pool. To be as fair as possible, we used Discrete AdaBoost instead of RealAdaBoost to combine the weak classifiers into stronger ones.

Figures 6a and 6b show the ROC curves we obtained by learning multiple boosted classifiers with 20 features and 200 features, respectively. In the former case, all methods that extend the feature pool outperformed Viola and Jones' method. Mita *et al.*'s learning method (method F) only increased the performance slightly. Our feature pool gave the best performance, resulting in a marginable gain over the other methods. In the latter case, the differences became less obvious but our pool still gave the best performance. This experiment shows that it is beneficial to use polygonal Haar-like features at the front layers of the cascade. By using polygonal Haar-like features, more negatives can be rejected early, allowing the cascade to evaluate faster at the same level of accuracy.

In the second experiment, we learned a face detector for each of the 7 above-mentioned weak-classifier learning methods. We used Pham *et al.*'s multi-exit asymmetric boosting method [19] to learn a cascade. Every exit node was designed to have $FAR < 0.7$ and $FRR < 0.001$. There were 32 exit nodes per cascade. Figure 6c depicts the resultant ROC curves, obtained from testing the cascades with the MIT+CMU test set. The most accurate cascade curve belongs to the cascade that was trained using our polygonal feature pool. Once again, the experiment confirmed the usefulness of polygonal features. Figure 7 presents the first 8 weak classifiers in the cascade, which also happened to be the cascade's exit nodes.

[5] http://www.bioid.com/downloads/facedb/index.php

[6] Compared to the traditional way of generating an ROC curve by training a single boosted classifier and varying the classifier's threshold, the method is more time consuming but more accurate [19].

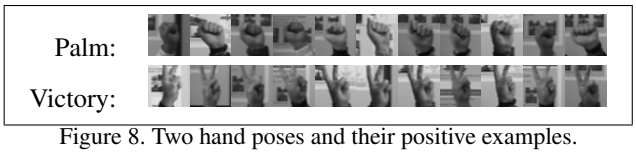
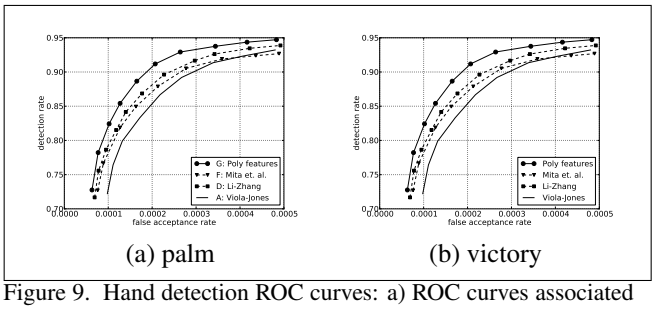


Figure 8. Two hand poses and their positive examples.



(a) palm (b) victory

Figure 9. Hand detection ROC curves: a) ROC curves associated with the palm pose, b) ROC curves associated with the victory pose.

4.2. Hand Detection

Haar-like features were also used in fixed-pose hand detection[9, 17]. We conducted an experiment analogous to section 4.1. Our setup was similar to that of [9, 17]. We trained detectors for detecting two different hand poses: palm and victory (figure 8). For data collection, we took video sequences at resolution 320×240 in which people made two different hand poses in front of different backgrounds. For each pose, we collected about 1,800 images, of which 600 randomly selected images formed a test set and the remaining images formed a training set for the pose. A generator was built that selected known hand locations randomly, and resized them down to a base patch resolution of 20×20 pixels, with some pertubation bias. Negative patches were generated from a sufficiently large set of images containing no hand. Training the cascades was the same as in section 4.1, except that each cascade had 28 exit nodes, and only the weak classifier learning methods A, D, F, and G were considered. These are the methods where hand detection has been investigated with[9, 17].

Figure 9 depicts the ROC curves obtained from the experiment. In both poses, the cascade trained with the polygonal Haar-like features outperformed other cascades. The gain was more obvious in the case of the victory pose.

4.3. Rock Detection for Terrain Assessment

In space research, terrain assessment is an important problem for autonomous navigation of planetary exploration rovers. Knowledge of the physical properties of terrain surrounding a planetary rover allows the rover to fully exploit its mobility capabilities [6, 15]. One critical challenge is how to detect rocks from other terrain properties, as it allows the rover to avoid collision while navigating.

We conducted an experiment similar to sections 4.1 and 4.2, to train two detectors that could detect rocks from images taken on Mars' surface. The first detector used Pham-Cham's Haar-like features [18], the second detector used the polygonal Haar-like features. We used publicly available images taken from NASA's Mars Exploration Rover missions for training and testing the detectors. Rock regions in 30 images of resolution 1024×1024 were manually annotated, each of which consisted of 10 to 50 rocks. We developed a generator to generate rock and non-rock patches of resultion 20×20 from the annotated images. Some positive examples are illustrated in figure 10.

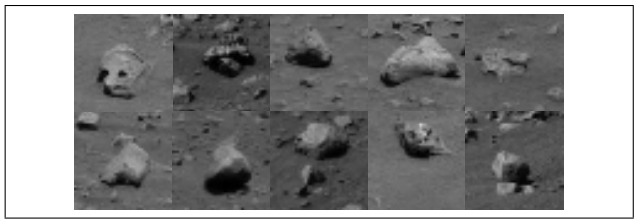
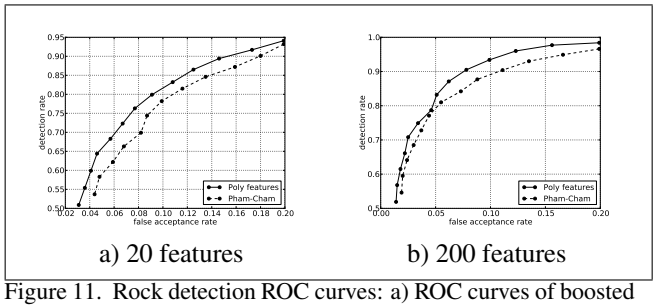


Figure 10. Positive examples of a rock patch.



a) 20 features b) 200 features

Figure 11. Rock detection ROC curves: a) ROC curves of boosted classifiers with 20 features, and b) ROC curves of boosted classifiers with 200 features.

Unlike the previous two domains, the rock detection problem turned out to be rather tough for Haar-like features. When training a cascade using each of the two feature pools, we obtained first boosted classifiers with 67 and 49 weak classifiers respectively, at $FAR \leq 0.05$ and $FRR \leq 0.01$. In other words, *a cascade would be even slower than a single boosted classifier*. Therefore, we abandoned the cascade idea and only trained each detector with a single boosted classifier. The ROC curves are reported figure 11. The results confirmed that using polygonal features would improve the accuracy of the detector.

4.4. Speed

Throughout the experiments, the average evaluation time was 0.30μs per feature polygonal Haar-like feature. The integration time in average was about 0.023μs per vertex, which implied, for instance, 15 million triangular integrations per second. The pre-processing time for a 320x240 image was 15.9ms. The average detection speeds of the trained cascades are reported in table 2 and table 3. The cascades trained using the polygonal feature pool ran the

Method	A	B	C	D	E	F	G
Face	7.8	8.7	8.5	9.2	9.6	7.6	**10.5**

Table 2. Average detection speeds (in fps) of the 7 face detectors when applied to the MIT+CMU test set.

Method	A	D	F	G
Palm	15.6	17.7	16.3	**18.4**
Victory	12.8	13.7	13.4	**14.5**

Table 3. Average detection speeds (in fps) of the 4 hand detectors when applied to testing video sequences at resolution 320×240.

fastest compared to other cascades in their domains.

5. Conclusions

We proposed a method for fast integrating a function over the interior of an arbitrary polygon in an image. The method integrated in time independent of the image resolution, and only linear to the number of vertices of the polygon. We applied the method on improving Haar-like features for object detection, by adding the feature pool with polygonal Haar-like features. We showed empirically that the extended feature pool led to improvement in detection performance, both in accuracy and in speed. The experiments were conducted in three domains: frontal face detection, fixed-pose hand detection, and rock detection for Mars' surface terrain asssessment.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 218814 "PRoVisG".

References

[1] A. L. C. Barczak, M. J. Johnson, and C. H. Messom. Real-time computation of haar-like features at generic angles for detection algorithms. In *Research Letters in the Information and Mathematical Sciences*, 2005. 1, 2, 5

[2] H. Bay, T. Tuytelaars, and L. J. V. Gool. SURF: Speeded up robust features. In *ECCV*, 2006. 1, 5

[3] G. Carneiro, B. Georgescu, S. Good, and D. Comaniciu. Detection and measurement of fetal anatomies from ultrasound images using a constrained probabilistic boosting tree. *TMI*, 27(9):1342–1355, Sep 2008. 1, 2

[4] F. C. Crow. Summed-area tables for texture mapping. In *SIGGRAPH*, 1984. 1, 2

[5] S. Du, N. Zheng, Q. You, Y. Wu, M. Yuan, and J. Wu. Rotated haar-like features for face detection with in-plane rotation. In *VSMM*, 2006. 1, 2, 5, 6

[6] Y. Gao. Optical flow based techniques for exomars rover autonomous navigation. In *Proc. Int. Symp. Artificial Intelligence, Robotics and Automation in Space*, 2008. 7

[7] M. Grabner, H. Grabner, and H. Bischof. Fast approximated SIFT. In *ACCV*, volume I, pages 918–927, 2006. 1, 5

[8] C. Huang, H. Ai, Y. Li, and S. Lao. Learning sparse features in granular space for multi-view face detection. In *FGR*, pages 401–407, Southampton, UK, 2006. 2

[9] M. Kölsch and M. Turk. Robust hand detection. In *AFGR*, pages 614–619, 2004. 7

[10] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 1, 2, 5

[11] J. P. Lewis. Fast template matching. In *Vision Interface*, pages 120–123, 1995. 1, 2, 5

[12] S. Li and Z. Zhang. Floatboost learning and statistical face detection. *TPAMI*, 26(9):1112–1123, Sept 2004. 5, 6

[13] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *ICIP*, 2002. 1, 2, 5, 6

[14] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004. 5

[15] M. Makhlouta, Y. Gao, and K. Shala. A vision and behaviour based approach for short-range autonomous navigation of planetary rovers. In *Proc. ESA Workshop on Advanced Space Technologies for Robotics and Automation*, 2008. 7

[16] C. Messom and A. Barczak. Fast and efficient rotated haar-like features using rotated integral images. In *Australasian Conference on Robotics and Automation*, 2006. 1, 2

[17] T. Mita, T. Kaneko, B. Stenger, and O. Hori. Discriminative feature co-occurrence selection for object detection. *TPAMI*, 30(7):1257–1269, 2008. 5, 6, 7

[18] M.-T. Pham and T.-J. Cham. Fast training and selection of haar features using statistics. In *ICCV*, 2007. 1, 5, 6, 7

[19] M.-T. Pham, V.-D. D. Hoang, and T.-J. Cham. Detection with multi-exit asymmetric boosting. In *CVPR*, 2008. 6

[20] F. M. Porikli. Integral histogram: A fast way to extract histograms in cartesian spaces. In *CVPR*, 2005. 1, 2, 5

[21] F. M. Porikli. Constant time O(1) bilateral filtering. In *CVPR*, 2008. 1, 2, 5

[22] H. Rowley, S. Baluja, and T. Kanade. Rotation invariant neural network-based face detection. In *CVPR*, pages 38–44, Santa Barbara, CA, USA, 1998. 6

[23] O. Tuzel, F. M. Porikli, and P. Meer. Region covariance: A fast descriptor for detection and classification. In *ECCV*, volume II, pages 589–600, 2006. 1, 2, 5

[24] P. A. Viola and M. J. Jones. Robust real-time face detection. *IJCV*, 57(2):137–154, 2004. 1, 2, 5, 6

[25] X. G. Wang, G. Doretto, T. Sebastian, J. Rittscher, and P. Tu. Shape and appearance context modeling. In *ICCV*, 2007. 1, 2

[26] J. Wu, S. C. Brubaker, M. D. Mullin, and J. M. Rehg. Fast asymmetric learning for cascade face detection. *TPAMI*, 20(3):369–382, 2008. 6

[27] R. Xiao, H. Zhu, H. Sun, and X. Tang. Dynamic cascades for face detection. In *ICCV*, 2007. 6

[28] Q. A. Zhu, M. C. Yeh, K. T. Cheng, and S. Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *CVPR*, 2006. 1, 2, 5