International Journal of System Signal Control and Engineering Application 1 (1): 1-14, 2008
ISSN: 1997-5422

Automatic Target Recognition and Classification in Aerial Photography

[1]Kayvan Najarian, [2]Robert DeMott II and [2]Jeremy Cooper
[1]Department of Computer Science,
[2]Electrical and Computer Engineering, University of Virginia Commonwealth,
601 W Main St, Richmond,VA, United States, America

Abstract: The use of multiple shape recognition techniques for target recognition and classification are explored and compared. An overview into the methods used to isolate the targets and remove background clutter is also included. The primary method used for shape detection was a version of geometric hashing, used in combination with a specialized corner detection method and a circle detection method. The corner detection method described obtained better results at finding corners in the black and white outlined images than several other compared methods. The geometric hashing algorithm implemented with a weighted-neighbourhood, performed better than the alternative methods compared.

Key words: Shape detection, aerial imagery, edge detection, morphology

INTRODUCTION

The primary objectives of the project were to identify and locate the position of well-defined targets on an airfield. The targets were contrasted with the surrounding area and consisted of simple geometric shapes of a solid color with an alphanumeric character inside of a different color. Both the objectives of this project and the images used during testing were obtained during the 2007 AUVSI competition (http://uav.navair.navy.mil/seafarers/default.htm for a newer, but similar set of rules). The table of parameters is shown in Table 1. The data obtained during this competition consists of 94 high-resolution images and a data stream describing the altitude and orientation of the camera.

The following section contains a review of some previous research done on shape processing. After that, the overall method is described. The first step of the overall method consisted of loading all the images and synchronizing the data stream information with the camera data. During this step, the camera orientation and position were determined during the exact moment each image was taken. Clutter removal is performed to remove regions that clearly do not match the target shape parameters. Once most clutter is removed, perspective correction is performed on the remaining regions.

Table 1: Abbreviated list of target parameters from the 2007 AUVSI competition

Shape	Size (ft)	BG color	Alpha color
Square	2×2	Red	Red
Triangle	2×4	Orange	Orange
Rectangle	2×8	Yellow	Yellow
Circle	4×4	Green	Green
Cross	4×8	Blue	Blue
Hexagon	8×8	Black	Black
Octagon		White	White
		Purple	Purple

REVIEW OF PREVIOUS SHAPE DETECTION WORK

Over the last few decades, multiple algorithms for shape detection have been developed. Among them, 2-dimensional shape matching techniques were of particular interest. Several categories exist including global image transforms, global object methods, voting schemes, computational geometry, etc. (Veltkamp and Hagedoorn, 1999). Global image transforms, such as wavelet-based methods, are not typically good at representing shape information; only the image as a whole is measured. Global object methods, such as curvature scale space, work on objects as a whole. Voting schemes, such as the geometric hashing method or pose clustering, operate on points of interest of the shape (Olson, 1997; Rigoutsos and Wolfson, 1997). Computational geometry deals with the geometric information of the shapes, such as their points, lines, or polygons (Veltkamp and Hagedoorn, 1999). Here a few techniques are briefly mentioned before moving on to the methods that were implemented.

One method explored was the generalized Hough transform or pose clustering. The generalized Hough

Corresponding Author: Kayvan Najarian, Department of Computer Science, University of Virginia Commonwealth, 601 W Main St, Richmond,VA, United States, America

transform can detect arbitrary shapes that are 2 dimensional and have undergone translation (Ballard, 1981). Hough transforms that account for rotation and scaling also exists. Pose clustering works similarly to this, but also works well with different rotations and scaling. The pose clustering method determines which, transformation parameters align groups of model features with groups of image features. A large number of these transformations should appear near the pose of the object in pose space. The clustering is performed by moving a box around the pose space. If a large number of points are found inside the box, a potential match exists (Olson, 1997).

Shape context are another method that was investigated. Shape context is a method to measure the similarity between shapes. A shape context is attached to each point. This shape context determines the distribution of all remaining points relative to it. Similar, shapes will have similar shape contexts, so comparing shape contexts of 2 shapes together enables the ability to perform shape detection (Belongie *et al*., 2000). Finally, the geometric hashing technique was explored and decided on for the primary method.

Geometric hashing essentially uses a 2D hash table to store different geometric transformations of a set of points from an image. This table is referenced using a set of query points from an image that can be a scaled, rotated, or translated version of the model used to generate the points stored in the table. For each set of points in the table, a set of points from the query shape are transformed and mapped into the table. For each unique point set in the table, a vote is cast based on the number of matches with that set and the query point set (Rigoutsos and Wolfson, 1997). Many other shape detection methods exist. For more information on these and other methods, (Veltkamp and Hagedoorn, 1999).

INITIALIZATION AND FILTERING

The first few steps of the algorithm read in an image and the data stream. The image is then resized and stored in memory temporarily for future access. The time of the image capture is taken from the camera and the pitch, roll, yaw, altitude, latitude and longitude of the camera is extracted from the data stream at the time of image capture. Once this step is done, the images are converted to YCbCr color space with MATLAB's rgb2ycbrc command. The individual channels are then separated to obtain grayscale images of the scene. The primary method is intended to be used on grayscale images only, so color information is ignored as much as possible until target details are extracted after a potential target has been identified. For this project, each channel was processed separately to locate the target shapes. For each channel, canny edge detection was performed with the edge MATLAB command. With this, a black and white edge image can be obtained such as that shown to the right (Canny, 1983) for information on canny edge detection). These images show the target shape, usually fully-enclosed and several unwanted regions of high contrast from the airfield. A large majority of these are filtered in subsequent steps (Fig. 1).

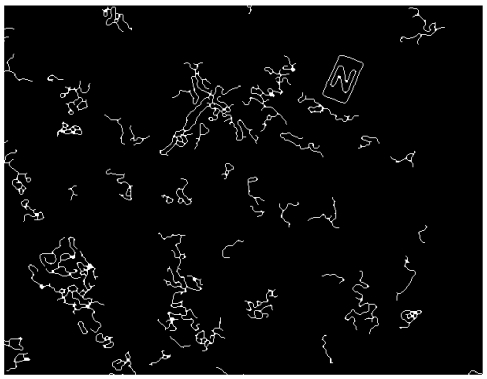

Fig. 1: Example image showing results of canny edge detection

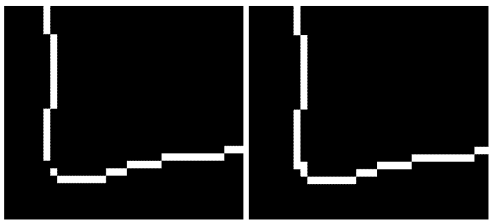

Fig. 2: Example images showing before and after of applying lookup table to a target shape's corner

The pre-filtering process consists of several morphological steps before the blob analysis is performed in the following study. The first stage of filtering involved morphologically opening the image using MATLAB's bwareaopen. This step removed regions that were too small to be a shape. The next step involved morphologically closing the images. First, MATLAB's imclose was used with a 3×3 square structuring element, then a custom lookup table was used to connect corners that would not have been connected with the imclose operation. A figure showing the effect of the lookup table operation is shown Fig. 2. After this initial filtering, the remaining regions are segmented using MATLAB's bwlabel and blob analysis is performed on the resulting regions using regionprops.

CLUTTER REMOVAL

The clutter removal stage is conceptually similar to the method described in Sun *et al.* (2005). Because the canny edge detector will detect edge segments that do not correspond to a target, it is necessary to filter the output of the segmentation stage. Each region detected must be classified as either "clutter" or a potential target. This is accomplished with a series of checks that take into account both image space information and real-world coordinates and dimensions for a given region. Prior knowledge of the target shapes is used to calculate valid thresholds for each of the checks. Some of these checks require the use of perspective correction.

Absolute pixel dimensions check: The first check will discard any region that contains insufficient information to accurately reconstruct any of the target shapes. This is accomplished be checking the width and height of the region's bounding box. If either the width or height is too small, that region is classified as clutter.

Filled area vs. unfilled area check: The second check will discard any region that does not contain a fully enclosed shape (i.e., the detected edge is incomplete). This is accomplished by counting the number of pixels in the filled version of the region and comparing it with the number of pixels in the unfilled region (outline). If the ratio is too low, the region is classified as clutter (Fig. 3).

Invalid location check: The third check will discard any region that is partially or completely above the horizon. This is accomplished by calculating the perspective corrected coordinates for each corner of the region's bounding box. If any of these points are invalid, the region is classified as clutter.

Physical area check: The fourth check will discard any region having a physical size outside the range of valid target sizes. The physical area of the filled region is computed using perspective correction. If the region size is smaller than 0.9 times the minimum target size or larger than 1.5 times the maximum target size, the region is classified as clutter. These values were chosen to compensate for potential noise and distortion in the image.

MajorAxisLength vs. MinorAxisLength check: The 5th check will discard any region having an eccentricity that is significantly outside the range calculated for the target shapes. For all target shapes, the ratio between its MajorAxisLength and its MinorAxisLength is calculated.

Fig. 3: Example of a region's outline area vs. filled area

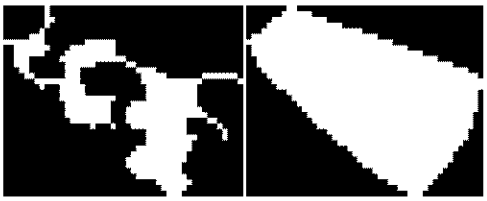

Fig. 4: Example of a region's filled area vs. convex area

An upper threshold ratio, $E_{MaxThresh}$, is calculated using the following formula, where, E_{Max} is the largest ratio calculated for all target shapes:

$$E_{MaxThresh} = (E_{Max})^{1.2} \quad (1)$$

Likewise, a lower threshold ratio $E_{MinThresh}$, is calculated using the following formula, where, E_{Min} is the smallest ratio calculated for all target shapes:

$$E_{MinThresh} = (E_{Min})^{0.8} \quad (2)$$

If the ratio calculated for a given region is outside of this range, it is classified as clutter.

Filled area vs. convex area check: The 6th and final check will discard any regions that are overly concave. For all target shapes, the ratio between each shape's filled area and its convex area is calculated. A minimum threshold, $C_{MinThresh}$, is calculated where, C_{Min} is the minimum ratio calculated for all target shapes (Fig. 4).

$$C_{Minthresh} = (C_{Min})^{1.5} \quad (3)$$

If any region has a ratio lower than $C_{MinThresh}$ it is classified as clutter.

Future work for clutter removal: Target shapes could potentially be segregated into various bins based on their physical size, eccentricity and convexity. The clutter

removal stage could then be used to classify a given region based on the same 3 parameters. It would then only need to be compared with target shapes from the corresponding bins. This could be used as both a performance enhancement method and a false identification reduction method.

PERSPECTIVE CORRECTION

In order to accurately detect a particular shape in a given region, that region must undergo perspective correction. In addition to pixel coordinates from the image, this process requires the angle of view of the camera lens, the camera's position and the camera's orientation at the time the image was captured (Fig. 5).

The first step is to calculate the diagonal angle of view. This requires the focal length of the lens and the physical size of the camera's image sensor. The focal length varies with the zoom level and is typically recorded in the image's EXIF data at the time of capture. The image sensor dimensions can be obtained from the camera's manufacturing specifications. Let d be diagonal size of the image sensor and f be the focal length of the lens. The diagonal angle of view, a, is then given by the following formula:

$$a = 2\arctan\left(\frac{d}{2f}\right) \tag{4}$$

Note: The following steps must be repeated for every point of interest in a given region.

The next step is to calculate a vector relative to the camera's forward-pointing lens vector. Let H be the horizontal size of the image in pixels, V be the vertical size of the image in pixels and a be the diagonal angle of view. For a given point in the image, let X_{Offset} be the horizontal offset from the center of the image (positive right) and Y_{Offset} be the vertical offset from the center of the image (positive down). Then two angles, Ψ_{Lens} and θ_{Lens} can be calculated as follows:

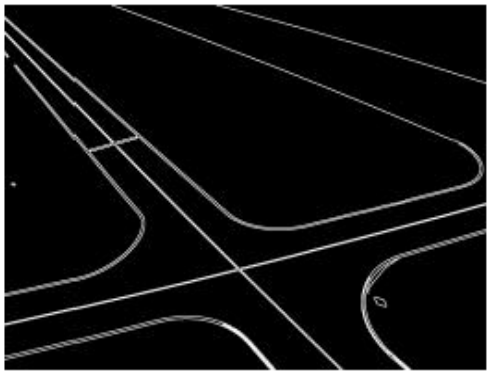

Fig. 5: Example image before prespective correction

$$\Psi_{Lens} = \arctan 2\left(X_{Offset}, -Y_{Offset}\right) \tag{5}$$

$$\theta_{Lens} = a\frac{\sqrt{X_{Offset}^{\ 2} + Y_{Offset}^{\ 2}}}{\sqrt{H^2 + V^2}} \tag{6}$$

The lens-relative unit vector, $\hat{V}_{Lens}$, can then be calculated as follows:

$$\begin{aligned}
\hat{V}_{Lens,X} &= \cos\left(\theta_{Lens}\right)\\
\hat{V}_{Lens,Y} &= \sin\left(\theta_{Lens}\right)\sin\left(\psi_{Lens}\right)\\
\hat{V}_{Lens,Z} &= -\sin\left(\theta_{Lens}\right)\cos\left(\psi_{Lens}\right)
\end{aligned} \tag{7}$$

After obtaining the lens-relative vector, it must then be converted to a camera-relative vector in world space. This is done using quaternion rotation with the provided camera orientation information. Let Q_{Camera} be a unit quaternion describing the orientation of the camera in world space.

First, a pure quaternion, Q_{Lens}, is created from $\hat{V}_{Lens}$:

$$\begin{aligned}
Q_{Lens,X} &= \hat{V}_{Lens,X}\\
Q_{Lens,Y} &= \hat{V}_{Lens,Y}\\
Q_{Lens,Z} &= \hat{V}_{Lens,Z}\\
Q_{Lens,W} &= 0
\end{aligned} \tag{8}$$

An intermediate quaternion, Q_{Temp} is then calculated as follows:

$$Q_{Temp} = Q_{Camera} \cdot Q_{Lens} \cdot Q_{Camera}^{\ -1} \tag{9}$$

The world space unit vector, $\hat{v}_{world}$, can then be extracted from the pure component of Q_{Temp}.

$$\begin{aligned}
\hat{V}_{World,X} &= Q_{Temp,X}\\
\hat{V}_{World,Y} &= Q_{Temp,Y}\\
\hat{V}_{World,Z} &= Q_{Temp,Z}
\end{aligned} \tag{10}$$

Before continuing, it is important to note that if the Z component of $\hat{v}_{world}$ is zero or negative, then the given point of interest is above the horizon and this process will fail.

Finally, relative north and east distances can be calculated using $\hat{v}_{world}$ and the altitude of the camera, A.

$$D_{North} = A\frac{\hat{V}_{World,X}}{\hat{V}_{World,Z}} \tag{11}$$

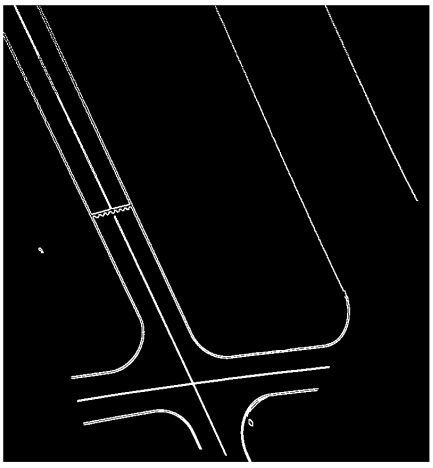

Fig. 6: Example image after perspective correction

$$D_{East} = A\frac{\hat{V}_{World,Y}}{\hat{V}_{World,Z}} \quad (12)$$

These distances can then be re-mapped to pixel coordinates on a new image. This process can be used to redraw an image or selected region from a vertical bird's eye view. This allows for accurate shape detection to be performed on ground level objects without requiring a strictly vertical view point.

The method described here approximates the ground as an infinitely flat plane. It also assumes that the camera is significantly higher than the objects being observed. If the terrain is significantly bumpy, a terrain height map should be employed and the equations modified accordingly. Also, the curvature of the earth can be factored in to provide increased accuracy at extreme altitudes (Fig. 6).

CORNER POINT DETECTION

Two different binary-image based corner detection methods were implemented initially. The first one used the Hough transform to find straight lines along the outline of the image. The second method is a specialized method that looks at the distance to the centroid of the object for each point along the perimeter to find the corners. Two alternative methods were attempted that operate on grayscale images as well, rather than the binary images that have been pre-filtered. Another binary-image based method was tried, but there were issues with the target image resolution that prevented the method from working as well as it could have. The centroid-based method was shown to work the best out of the methods tried for the test data, but may not work as well for more complex shapes. In addition to the corner point detection methods, a corner point minimization technique is also described.

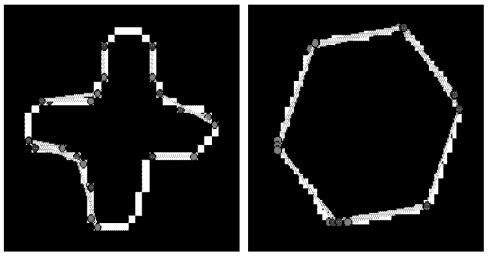

Fig. 7: Example result for Hough line method

Hough lines: The first method to detect the lines around the outline of the image was a version of the Hough transform for detecting lines (Duda and Hart, 1972). The basic method works by the method described here. First, points in the (x,y) plane are translated into the (ρ,θ) plane, where:

$$\rho = x_i \cos\theta + y_i \sin\theta \quad (13)$$

Every point in a parameter plane corresponds to a straight line in the picture plane and curves through, a common point in the parameter plane correspond to a straight line in the picture plane. Therefore, a potential line exists where, there are a sufficient number of intersections of curves in the parameter plane. The location of the line can be found in the picture plane since the points lying on the same curve in the parameter plane correspond to lines through the same point in the picture plane.

To use this method, the lines around the outline were broken down into line segments using a combination of MATLAB's houghpeaks and houghlines functions. The results of these functions are a list of points where, each line segment ends. For shapes other than circles, these line segments should end on or near corners. This technique worked relatively well on certain shapes, but the edges for certain shapes weren't perfect enough to obtain good results (Fig. 7).

Centroid-distance corner detector: The centroid-distance corner detector is another method that can be used with binary images. Its basic operation is described as follows:

Given the coordinates of a shape's centroid and an ordered list of points composing its perimeter, a discrete wave can be generated based on the Euclidian distance of each perimeter point to the centroid. Let P be the list of perimeter points, i be the index into P and C be the centroid point. Then the distance signal, D, is calculated as follows:

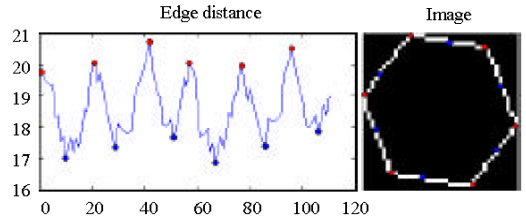

Fig. 8: Centroid-Distance signal for a test image. Red points are local maxima, Blue points are local minima

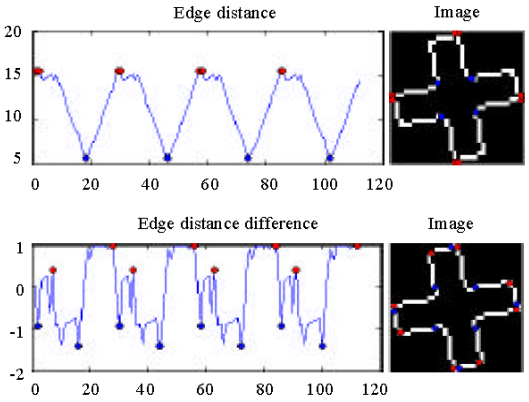

Fig. 9: Centroid-Distance signal and its derivative for cross template. Red points are local maxima, Blue points are local minima

$$D(i) = \sqrt{\left(C_x - P_x(i)\right)^2 + \left(C_y - P_y(i)\right)^2} \qquad (14)$$

Potential points of interest lie at the local maxima and local minima of the distance signal. It is important to note that the distance signal calculated will be cyclical in nature and this must be taken into account when determining the local maxima and minima locations (Fig. 8).

For convex shapes, corner points can be determined by looking at only the local maxima. For concave shapes, the local minima must also be examined. This method cannot always locate all corners if a shape is concave. To improve the accuracy with concave test shapes, the first derivative of the distance signal should also be examined for local maxima and minima (Fig. 9).

In order to account for all types of shapes, both the distance signal and its derivative are always analyzed. To reduce the number of extraneous points detected in convex shapes, a point minimization technique is employed (Fig. 10).

Additional corner detection methods: In addition to Hough lines and the centroid-based method, other corner detection techniques were tested. An implementation

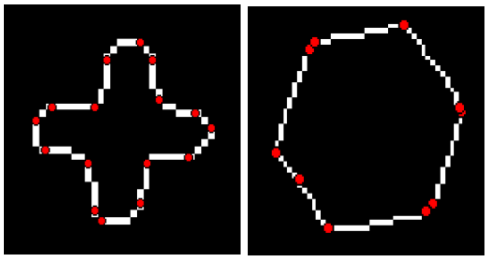

Fig. 10: Example result for centroid-based method

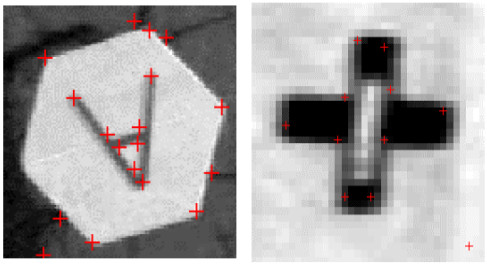

Fig. 11: Example results for Harris

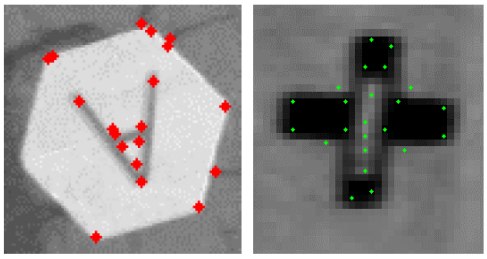

Fig. 12: Example results for FAST

of the Harris corner detection algorithm (Harris and Stephens, 1988; http://www.csse.uwa.edu.au/~pk/research/matlabfns/) and another algorithm called the FAST corner detector (Roston and Drummond, 2005, 2006) were implemented. Both the Harris and FAST corner detection algorithms rely on grayscale images and do not work on filled or outline binary images. For comparison purposes, they are shown below. One major problem with operating on the grayscale images is that they are not the filtered-versions and are susceptible to background noise around the target. If the outline of the object is operated on directly, as with the centroid-based method described earlier, the corner points are chosen based entirely on the outline of the targets and not the ground around them (Fig. 11 and 12).

Another binary method attempted was a pixel-based curvature approximation technique. Essentially, this technique traces over pixel-boundaries (sub-segments of a certain size of the shape outline) and determines

potential corners by calculating the maximum bending ratio in a moving window. Non-maximal suppression is applied to remove points close to each other, so only the strongest corners are kept (Kiranyaz *et al*., 2007). It works very well on higher resolution images, but most of the target shapes were too small to reliably detect using this technique.

Corner point minimization: The corner point minimization method described here can be used to remove redundant (non-corner) points from the output of any of the previously described binary corner point detectors. This method works by calculating the angle formed by a group of 3 consecutive "corner" points. If the calculated angle is close to forming a straight line (180°), the middle point can be removed. This method is a similar, but opposite approach as that taken in (Kiranyaz *et al*., 2007).

This method requires an ordered list of potential corner points. If the output of a binary corner detector is not ordered, then the ordered list of shape perimeter points can be used to sort the corner point list.

For every point, P, in the cyclical corner point list, let P_{Prev} be the previous point and P_{Next} be the next point. The following calculations can then be made:

$$\begin{aligned}
Diff_{Prev,x} &= P_x - P_{Prev,x}\\
Diff_{Prev,y} &= P_y - P_{Prev,y}\\
Diff_{Next,x} &= P_{Next,x} - P_x\\
Diff_{Next,y} &= P_{Next,y} - P_y\\
Angle_{Prev} &= atan2\left(Diff_{Prev,y}, Diff_{Prev,x}\right)\\
Angle_{Next} &= atan2\left(Diff_{Next,y}, Diff_{Next,x}\right)
\end{aligned} \tag{15}$$

The angular difference, $Angle_{Diff}$, between the 2 absolute angles calculated above is then given by:

$$Angle_{Diff} = mod\left(Angle_{Prev} - Angle_{Next} + \pi, 2\pi\right) - \pi \tag{16}$$

A simple minimum angle threshold, $Angle_{MinAbs}$ can be specified. If the absolute value of $Angle_{Diff}$ is less than or equal to this threshold, then P is not a corner and can be discarded.

To help detect redundant points in low resolution images, a modification can be made. Instead of relying on a simple angular threshold, a dynamic threshold can be calculated based on the distances between the three points. This dynamic minimum angle, $Angle_{MinDyn}$ can be calculated as follows:

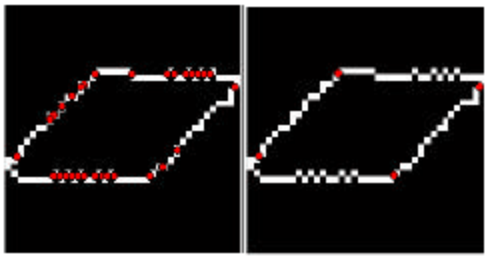

Fig. 13: Example of corner point minimization on a noisy test shape

$$Diff_{Prev,max} = max\left(\left|Diff_{Prev,x}\right|, \left|Diff_{Prev,y}\right|\right) \tag{17}$$

$$Diff_{Next,max} = max\left(\left|Diff_{Next,x}\right|, \left|Diff_{Next,y}\right|\right) \tag{18}$$

$$Angle_{MinDyn} = atan2\left(1, min\left(Diff_{Prev,max}, Diff_{Next,max}\right)\right) \tag{19}$$

This helps compensate for the lower granularity inherent in digitized images.

Both thresholds can be used in combination for best results (Fig. 13).

$$Angle_{Min} = max\left(Angle_{MinAbs}, Angle_{MinDyn}\right) \tag{20}$$

SHAPE DETECTION

A few shape detection techniques are discussed and compared. First, the primary method, geometric hashing, is discussed. Next, bitmap template matching is discussed. As a third method to compare, the shape contexts technique was implemented and is discussed. This implementation of geometric hashing relies on corner points, so a method to detect circles to go along with the geometric hashing technique was necessary. The circle detection used for that is briefly discussed as well. Finally, the different methods of shape detection implemented are compared.

Geometric hashing: The primary method implemented to perform shape matching was geometric hashing. It was briefly described earlier, but will be described here in more detail. Note that the technique implemented here only uses two basis points to detect 2D objects in different translations, rotations and scales. Using more than two basis points, discussed elsewhere (Rigoutsos, 1992; Rigoutsos and Wolfson, 1997; Veltkamp and Hagedoorn,

1999), is a method that can be used to detect 3D objects as well as 2D objects, but it requires more computational complexity. The method implemented consists of 2 stages: preprocessing and recognition. The preprocessing phase consists of the following steps for each model:

- Determine the points of interest.
- For each ordered pair or basis, compute the coordinates of the rest of the points in the model based on those points.
- Use those points as indices to index into the 2D hash table. The model name and set of basis points are stored in the hash table bin.

Once the preprocessing phase is done, the next step is to recognize the object given a query image. The following steps comprise the recognition phase for each image:

- Determine the points of interest.
- Choose an ordered pair or basis and compute the coordinates of the rest of the interest points in the model based on those points.
- Using those points as indices into the hash table, determine all entries found in each accessed hash table bin.
- Histogram all hash table entries and compute a score for each potential match taking into consideration both the input image features and the model features.
- If the score isn't high enough, go back to step 2 and repeat with a different image basis pair (Rigoutsos and Wolfson, 1997).

In order to determine the points of interest, the corner point detection algorithms were used. The next step, determining the basis points and computing table coordinates is done in the following method. First, for the table can only be a finite size so there needs to be a limit on how close 2 basis points can be apart from each other. The table can then be scaled by this size according to the method described here.

Assume 2 points in an image can never be closer than 1/k times the longest side of the bounding box of an object. Let d_{unit} be the length 1/k. If the 2 points closest together on the side of an object are chosen (spaced exactly d_{unit} apart), then the farthest point can be at most k distances away from either of the points. When populating the hash table, points far away from the basis points must fit inside the table, so the table must be sufficiently large to accommodate k d_{unit} distances away from either basis point location. Also assume that the basis points are to be spaced b_{dist} apart in the table. The smallest size of the hash table's rows and columns are then:

$$\begin{aligned} \text{Columns} &= 2kd_{unit}b_{dist} + b_{dist} \\ \text{Rows} &= 2kd_{unit}b_{dist} \end{aligned} \tag{21}$$

From this, the locations to store the two basis points are simply:

$$\begin{aligned} \text{Basis}_{col} &= kd_{unit}b_{dist} \pm \frac{b_{dist}}{2} \\ \text{Basis}_{row} &= kd_{unit}b_{dist} \end{aligned} \tag{22}$$

Once the table has been sized and the row and column of each basis point are known, the template points are inserted into the table. Each set of basis points are chosen from the templates one pair at a time to be placed in the table. That means there is a maximum of n (n-1) basis point pairs for each model stored assuming each model contains n points. For each basis point pair, it is necessary to find the locations of all other points assuming the first basis point should to be located at (0,0) and the second point to be at (1,0). First, translate all points so that the first point moves to (0,0). So, for all M(i), where, i is between 1 and n and some b_1, where, b_1 and b_2 are the first and second basis points, respectively:

$$M(i) = M(i) - M(b_1) \tag{23}$$

The next step is to rotate all points about the first basis point (b_1) so that the second basis point (b_2) lies on the x-axis. For this, one way is to compute the sine and cosine of the angle required to rotate b_2 about the origin (b_1) and then rotate all objects the same angle. This was done using the following method:

$$\begin{aligned} \text{cosine} &= \frac{M(b_2)_x}{\sqrt{M(b_2)_x^2 + M(b_2)_y^2}} \\ \text{sine} &= \frac{-M(b_2)_y}{\sqrt{M(b_2)_x^2 + M(b_2)_y^2}} \\ M(i)_x &= \text{cosine}M(i)_x - \text{sine}M(i)_y \\ M(i)_y &= \text{sine}M(i)_x + \text{cosine}M(i)_y \end{aligned} \tag{24}$$

Finally, scale all points so that the second point lies at (1,0) on the x-axis. For this, a scale factor (s) must be determined and all points must be multiplied by that value. The scale factor is found by determining how much b_2 must be scaled along the x-axis for it to lie on (1,0). Since, it already lies on the x-axis from the previous

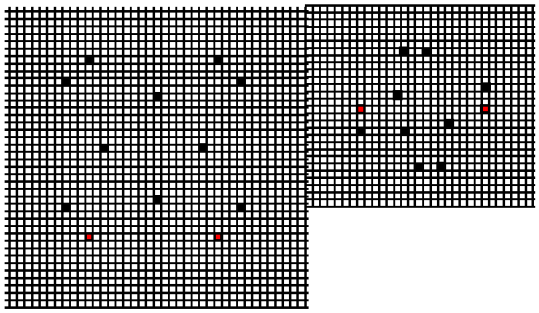

Fig. 14: Example of 2 hash table mappings for different basis pairs

transformations, only the following steps need to be performed:

$$\begin{aligned} s &= 1/M(b_2)_x \\ M(i)_x &= sM(i)_x \\ M(i)_y &= sM(i)_y \end{aligned} \tag{25}$$

To determine where, the points map into the table, it is necessary to translate and scale all points again so that the basis points map to the middle of the table. For example, if a table is 100×100, the basis points should map to [50, 45] and [50, 55] (assuming [row , column] table format and 10 cell separation between basis points). At this point, the first and second basis points are located at (0,0) and (1,0), respectively. Therefore, all points only need to be scaled by b_{dist} (set s = b_{dist} in Eq. 25) and translated by ($Basis_{col,1}$, $Basis_{row}$) so that b_1 lies on [$Basis_{row}$, $Basis_{col,1}$] and b_2 lies on [$Basis_{row}$, $Basis_{col,2}$] (from Eq. 22). Once these new points are calculated, they may simply be rounded to obtain their locations in the hash table (Fig. 14).

Now that the locations to map the basis points are known, a neighborhood should be determined. It has been mentioned in previous research (Rigoutsos, 1992; Rigoutsos and Wolfson, 1997) that noise can cause positional errors (a point maps into a hash bin near, but not exactly on the desired bin) that can potentially completely mismatch the shape. For this reason, a weighted neighborhood was implemented such that the weights of surrounding bins are inversely proportional to the distance to the primary hash bin. The primary hash bin is given a weight of 1, but the rest are scaled according to the following function:

$$W = 1 - \frac{d}{r} \tag{26}$$

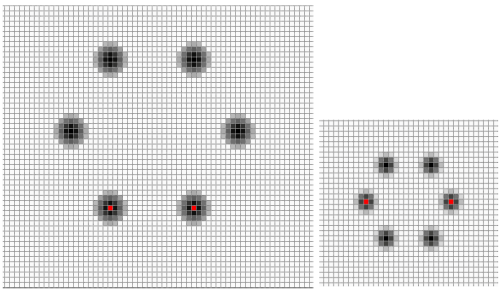

Fig. 15: Example of scaled neighbourhoods for 2 different basis pairs

where, d is the distance away from the primary (center) bin and r indicates the maximum radius from the primary bin to propagate the neighbourhood. The value of r is determined by the following formula:

$$r = \left| \frac{d_{min}}{\sqrt{(b_{1,x} - b_{2,x})^2 + (b_{1,y} - b_{2,y})^2}} r_{max} \right| + 1 \tag{27}$$

where, d_{min} is the smallest distance between any 2 basis points in the model and r_{max} is the maximum radius allowed. This method gives us an approximately circular weighting scheme that gives direct hits the largest scores, but can deal with noise as well. The Fig. 15 shows a graphical representation of the neighbourhood weighting scheme. Darker cells indicate a larger weight; while, brighter cells indicate smaller weight (white has no weight, while, black has maximum weight).

The above procedure is used to place all models into the hash table for each unique set of basis points. Overall, this takes approximately O (n^2 m) time to insert the m models of n points each into the table. Once this is done, a query image can be tested in as little as O(n) time complexity (Rigoutsos, 1992).

The query images are tested with the same initial steps as loading the model information into the hash table. First, the interest points are found based on the corner detection methods. The points from the query image undergo the same transformations to determine their location in the hash table as well. There is no need to compute a neighbourhood for the query points, however, since the neighbourhood is computed for the model points only. Once a set of points has been mapped into hash table locations, the score for each unique model transformation (chosen pair of basis points for a model) is calculated. The score is calculated in the following manner:

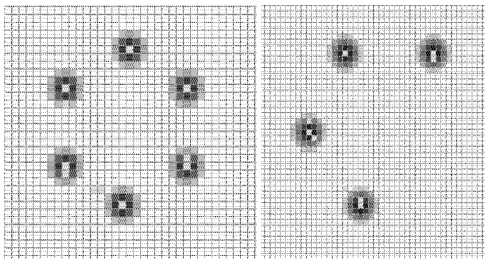

Fig. 16: Example results for geometric hashing

$$
\begin{aligned}
s &= \frac{\sum W(M_k(b_i, b_j))}{n} \\
t &= \frac{C(M_k(b_i, b_j))}{n} \\
u &= \frac{U(M_k(b_i, b_j)) - 2}{M_{k,n} - 2} \\
\text{Score} &= (s)^{W_s} (t)^{W_t} \left(\frac{1}{1+e^u}\right)^{W_u}
\end{aligned}
\tag{28}
$$

Here, the value s is the sum of the weights (W) for all the hits on the hash table of a specific model k with basis points b_i and b_j divided by the number of points in the query image n. Similarly, t is the total count (C) of the number of hits on the hash table for a specific model k and basis points b_i and b_j also divided by the number of points in the query image. The variable u keeps track of the unique hits (U) on the hash table for a specific model k with basis points b_i and b_j divided by the number of points in the model ($M_{k,n}$). Note that u is calculated to ignore the basis points as hits, while, the other scores include the basis points. It is also calculated with a sigmoid function to enable a better non-linear cut-off for which shapes match the templates correctly. The overall score is computed with weights for each sub-score. If W_s is a large number, for example, more weight is given towards the sum of weights sub-score. After the weights are computed for each model and each model's unique basis point set, the maximum of all Score values are found. The best score is then examined to ensure that it is above some minimum score ($Score_{min}$) and if it is, then a match occurred. Otherwise, there is not a close enough similarity to the points of the shapes in the hash table to be called a match (Fig. 16).

Bitmap template matching: Bitmap template matching is a simple and straight-forward method that can be used to calculate the similarity between a binary test bitmap and all stored bitmap templates. The concept is similar to that described in Sun *et al*. (2005).

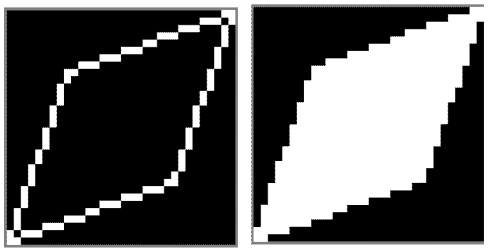

Fig. 17: Example 32×32 bitmap template for a diamond

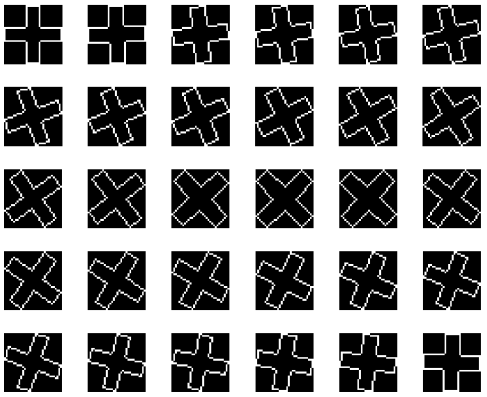

Fig. 18: Unique outline rotations for cross template using a 3 degree interval

A bitmap template consists of a filled version and an outline version of a given shape, scaled to a fixed resolution (X,Y). The shape is always centered and scaled to fill as much area on the bitmap template as possible. Depending on the chosen resolution and the degree of outline precision required; the outline image may optionally undergo morphological dilation. This will increase the outline match score for non-perfect test shapes.

Bitmap templates will need to be created for every target shape included in the database. One of two main methods can be used to allow detection of rotated and flipped shapes (Fig. 17).

Method 1: For every template shape, pre-compute bitmap templates for every rotation (using an N-degree interval) and the corresponding flipped view. Store all unique templates in the database. The test shape can then be compared directly with all stored templates.

Method 2: Only store one view of every template shape. For each test shape, compute all rotations (using an N-degree interval) and flipped views. Compare each view with all stored templates.

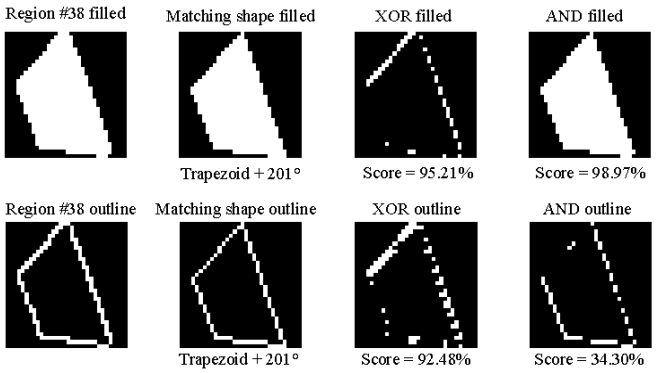

Fig. 19: Example bitmap template matching results

There is a storage size and speed tradeoff for the two methods. If storage size is not a concern, the first method is generally faster (Fig. 18).

Bitmap template scoring: Since, the bitmap templates consist of simple binary images, calculating a similarity score between 2 templates can be done using logical operators. Implementations of this method are often quite fast relative to more complex algorithms.

For a given binary image A, let X be the number of horizontal pixels and Y be the number of vertical pixels. The following function is then defined:

$$\text{NumTrue}(A) = \sum_{i=1}^{X}\sum_{j=1}^{Y} A_{i,j} \tag{29}$$

This function is used simply to count the number of "on" or "true" pixels in a binary image.

In addition to this function, two main operations are performed: binary XOR and binary AND. These two operations are performed on both the outline bitmaps and the filled bitmaps of both templates.

Let A and B be two bitmap templates. The scoring function is then broken into four sub-scores (S_{FA}, S_{OA}, S_{FX} and S_{OX}) as follows:

$$F_{And} = A_{Filled} \wedge B_{Filled}$$
$$S_{FA} = \frac{\left(\text{NumTrue}(F_{And})\right)^2}{\text{NumTrue}(A_{Filled})\,\text{NumTrue}(B_{Filled})} \tag{30}$$

$$O_{And} = A_{Outline} \wedge B_{Outline}$$
$$S_{OA} = \frac{\left(\text{NumTrue}(O_{And})\right)^2}{\text{NumTrue}(A_{Outline})\,\text{NumTrue}(B_{Outline})} \tag{31}$$

$$F_{Xor} = A_{Filled} \oplus B_{Filled}$$
$$S_{FX} = 1 - \frac{\text{NumTrue}(F_{Xor})}{X \cdot Y} \tag{32}$$

$$O_{Xor} = A_{Outline} \oplus B_{Outline}$$
$$S_{OX} = 1 - \frac{\text{NumTrue}(O_{Xor})}{X \cdot Y} \tag{33}$$

The final similarity score, S, is then computed as follows:

$$S = (S_{FA})^{W_{FA}} (S_{OA})^{W_{OA}} (S_{FX})^{W_{FX}} (S_{OX})^{W_{OX}} \tag{34}$$

For each sub-score, $S_?$, a corresponding weight, $W_?$, is applied. Weights can vary from 0 to 8 with higher weights giving more importance to a particular sub-score. A weight of 0 will cause that sub-score to be ignored (Fig. 19).

Shape contexts: Shape context was another method tested. It was briefly mentioned earlier in the study, but will be discussed here in some further detail. The main steps involved (Belongie, 2000) are as follows for each shape:

- Convert edge elements of the shape into a set of N feature points.
- For each point P, compute a course histogram of the relative coordinates of the remaining points (called the shape context).
- Store these histograms into a log polar coordinate system with n equally spaced angle bins and m equally spaced log-radius bins. Let the total number of bins, K be m×n.

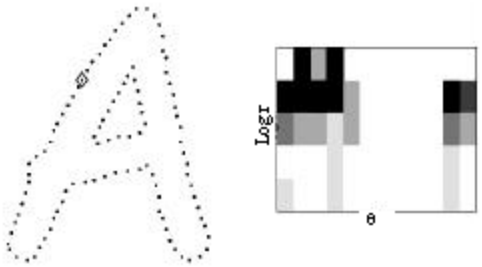

Fig. 20: Example edge points and selected histogram (Belongie *et al*., 2000)

- Normalize all radial distances by the median distance (λ) between all N^2 point pairs in the shape.

Once these steps are complete, shape contexts from different shapes can be matched with the following method. Given an interest point i for one shape and an interest point j on a second shape, a cost for matching these points ($C_{i,j}$) can be determined. Assume the histogram at i is g(k) and the histogram at j is h(k). The cost is then compute using the X^2 statistic:

$$C_{i,j} = \frac{1}{2}\sum_{k=1}^{K}\frac{[g(k)-h(k)]^2}{g(k)+h(k)} \quad (35)$$

These costs are then computed for all points i and j in the first and second shape, respectively. As mentioned by (Belongie, 2000), the Hungarian method can then used to compute the minimal cost function for all $C_{i,j}$ values, which can be computed in $O(N^3)$ time. A cost function lower than a particular threshold value indicates the two shapes are similar. An example image is shown in Fig. 20.

Circle detection: Two methods were used in conjunction to perform circle detection. The first method was Hough circles (Kimmie *et al*., 1975). The Hough circle method relies on a 3 dimensional array of accumulators that track of the number of points that lie on each circle of a number of different radiuses at each point in an image. If the number of elements found lying on a circle of a particular radius centered at a particular point is greater than a certain threshold, a potential circle of that radius exists centered at that point.

The implementation of this algorithm (http://www.mathworks.com/) took in a parameter for the threshold value, the desired radius of the circle to find and the black and white outline image and it returned the center points of any potential circles found. Using the bounding box of the region of interest, a radius for the circle can be calculated in a straight-forward manner and the threshold should be some value proportional to the perimeter of the circle. For nearly perfect circles, this works quite well. This technique also picks up approximately circular shapes like hexagons as well with a lower threshold. Some of the circles in the test data required thresholds that were low enough to detect hexagons almost every time as circles. Due to this issue, another method was implemented to remove false positives from the circle detection.

A simple, but effective roundness metric was implemented by examining the perimeter and area of the shape detected (http://www.mathworks.com/). The perimeter and area of a circle are measured as:

$$\begin{aligned} p &= 2\pi r \\ a &= \pi r^2 \end{aligned} \quad (36)$$

where, r is the radius of the circle. From this, the following roundness metric can be computed such that the metric will be 1 for a perfect circle and some number less than 1 otherwise:

$$\text{metric} = \frac{4\pi a}{p^2} \quad (37)$$

If the measured perimeter from the target region is close to the circle perimeter and the area is close to the circle area from the radius, then the object is detected as a circle. Combining the two methods worked for all cases of circle detection in the test data and managed to only detect the target circles as well. Note, again, that this circle detection was only performed for the geometric hashing and not the comparison methods. The comparison methods were able to detect circles with reasonable accuracy on their own, while, the geometric hashing method's dependence on corner points prevented this.

Shape matching comparison: A comparison between Geometric Hashing (with Circle Detection), Shape Context Matching and Bitmap Template Matching was performed. The centroid-distance corner detection method was used to detect corner points for Geometric Hashing.

The input data for each algorithm consisted of all the perspective corrected regions that passed the clutter removal stage (Table 2). Templates were created for all the simple geometric shapes listed already.

The following parameters were used for Geometric Hashing:

Table 2: Summary of shape matching results ignoring shapes removed with the clutter removal stage

Method	Accuracy (%)
Geometric hashing	93.75
Shape context matching	87.50
Bitmap template matching	88.75

$$b_{dist} = 12$$
$$r_{max} = 3$$
$$W_s = 1$$
$$W_t = 1$$
$$W_u = 1$$

The following parameters were used for Shape Context Matching:

$$N_{Points} = 50$$
$$\theta_{Bins} = 12$$
$$r_{Bins} = 5$$
$$r_{Low} = 0.125$$
$$r_{High} = 2$$

The following parameters were used for Bitmap Template Matching:

$$X = 32$$
$$Y = 32$$
$$N = 3$$
$$W_{FX} = 1$$
$$W_{OX} = 2$$
$$W_{FA} = 1$$
$$W_{OA} = 0.5$$

COLOR CLASSIFICATION

Since, the targets were classified both by shape and color, a simple color detection technique was performed to determine the color of the target. First, the pixels around the outline of the target in the original color image were converted to the HSV color space. It becomes simple to classify colors in this color space with basic thresholds. Since, the target parameters only specified targets as red, orange, yellow, green, blue, purple, black, or white, the following basic method is used:

- Target is classified as black if value is low.
- Target is classified as white if saturation is low.

Table 3: Final comparison of shape matching methods

Method	Accuracy (%)
Geometric hashing and circle detection	85.32
Shape context matching	80.73
Bitmap template matching	81.31

- Target is classified as one of the other colors based on the hue.

In HSV color space, the hue is a degree from 0 to 360. Based on that degree, the individual colors are segmented using thresholds. For example, red could be segmented as (red < 10 or 340 < red), while, green could be (80 < green < 150). The remaining colors were segmented similarly. The results from this method with the test data were 100% accurate so another method for this dataset was not considered.

Once a method for determining the color was implemented, another segmentation step was implemented by splitting the image into color regions labelled with an index corresponding to the color. This was tested as a simple, but effective way to extract the letter out of the image. For future work, the letter stored inside the target would be determined, as well as the orientation of the target based on the top of the letter.

RESULTS

The overall final results of the methods described taking into account the clutter removal stage are shown in Table 3. The primary method worked better than the comparison methods. Accuracies are slightly lower here than in the shape comparison section because the clutter removal stage erroneously classified some targets as clutter.

Geometric hashing, in combination with the circle detection worked better than all other methods compared. The corner point algorithms developed in conjunction with the geometric hashing neighbourhood scheme were demonstrated to work effectively for the targets and images being tested. Further research can be done to improve the geometric hashing and clutter removal. Implementing the character recognition would also be a good next step for the improvement of the overall project. In addition, there are other shape detection techniques that can be compared with those currently implemented.

ACKNOWLEDGEMENT

We would like to acknowledge Dr. Kayvan Najarian of Virginia Commonwealth University for his invaluable assistance and guidance in this project.

REFERENCES

Ballard, D.H., 1981. Generalizing the Hough transform to detect arbitrary shapes. Pattern Recognition, 13 (2).

Belongie, S., J. Malik and J. Puzicha, 2000. Shape context: A new descriptor for shape matching and object recognition. In NIPS.

Canny, J.F., 1983. Finding Edges and Lines in Images. MIT AI Lab Tech Report TR-720.

Duda, R. and P.E. Hart, 1972. Use of the Hough transformation to detect lines and curves in pictures. Commun. ACM., 15: 11-15.

Harris, C. and M. Stephens, 1988. A combined corner and edge detector. In ALVEY Vision Conference, pp: 147-151.

Kimmie, C., D. Ballard and J. Sklansky, 1975. Finding Circles by an Array of Accumulators. Commun. ACM., 18: 120-122.

Kiranyaz, S., H. Liu, M. Ferriera and M. Gabbouj, 2007. An efficient approach for boundary based corner detection by maximizing bending ratio and curvature. In: Proceeding of the International Conference on Information Sciences, Signal Processing and their Applications, ISSPA, Dubai, UAE.

Kovesi, P.D., 2007. MATLAB and Octave Functions for Computer Vision and Image Processing. School of Computer Science and Software Engineering, The University of Western Australia. http://www.csse.uwa.edu.au/~pk/research/matlabfns/.

Olson, C.F., 1997. Efficient pose clustering using a randomized algorithm. Int. J. Comput. Vision, 23 (2): 131-147.

Rigoutsos, 1992. Massively parallel Bayesian object recognition. Ph.D. Thesis, Courant Institute of Mathematical Sciences, New York.

Rigoutsos and H. Wolfson, 1997. Geometric Hashing. IEEE. Computational Sci. Eng., 1070-9924.

Roston, E. and T. Drummond, 2005. Fusing points and lines for high performance tracking. IEEE. Int. Conf. Comput. Vision, 2: 1508-1511.

Roston, E. and T. Drummond, 2006. Machine learning for high-speed corner detection. Eur. Conf. Comput. Vision, 1: 430-443.

Sarafraz, 2007. Circle Detection via Standard Hough Transform. http://www.mathworks.com/.

Sun, S.G., D.M. Kwak, W.B. Jang and D.J. Kim, 2005. Small target detection using center-surround difference with locally adaptive threshold. In: Proc. 4th Int. Symp. Image and Signal Proc. Anal., pp: 402-407.

Veltkamp, R.C. and M. Hagedoorn, 1999. State-of-the-art in shape matching. Technical Report UU-CS-1999-27, Utrecht University. The Netherlands.