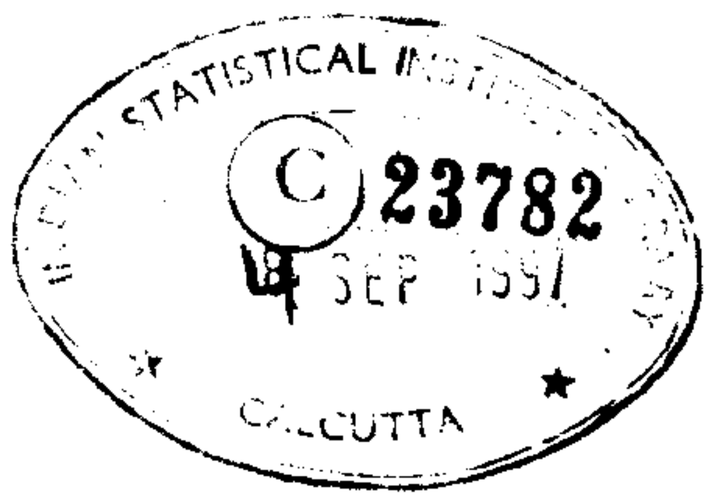
ON BACKPROJECTING SKEWED IMAGES

Nilanjan Ray
Indian Statistical Institute
203 B T Road
Calcutta 700035
India

Thesis submitted to Indian Statistical Institute
for partial fulfillment of the requirements for
the degree of Master of Technology (Computer Science)

July 1997



Acknowledgements

The author acknowledges Dr. Dipti Prasad Mukherjee for his encouragement, guidance and support. The author likes to thank Mr. Paramartha Dutta, Mr. Pinak Pani pal and Mr. Avik Mukherjee of E.C.S.U dept. ISI, without the help of whom it would have been impossible to complete the thesis. The author conveys his gratitude to Prof. J. Das, Head of E.C.S.U, for allowing him to use resources of the laboratory. The author also likes to thank the M.Tech(CS) 2nd year batch for their co-operative spirit.

Abstract

The general imaging situation is the perspective viewing condition, where we see objects from an oblique viewpoint. Under such situations it is very much essential in vision approach to get back the original shape of the objects from their perspective views. In this thesis we have proposed an algorithm to get back the object shape upto a similarity transformation using the object property *reflectional symmetry*.

Contents

List of Figures

Chapter 1

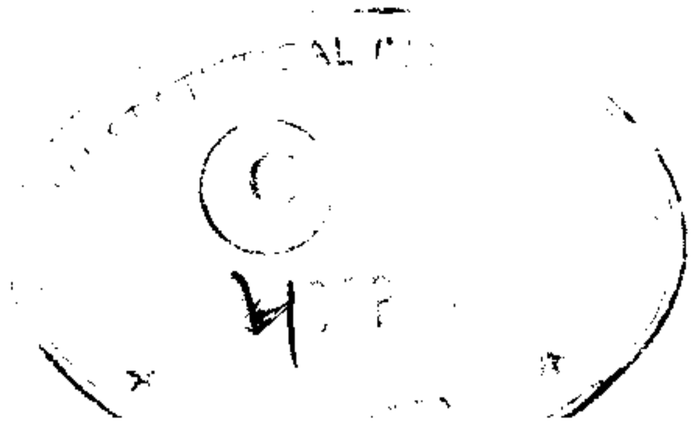
Introduction

The representation and recognition of shapes by computer has vast potential; however, in most cases the computerized approach has difficulty in understanding images of objects which otherwise are performed at ease by a human observer. Looking at an object from an oblique viewpoint gives rise to such a situation where the human observer can effortlessly recognize the object even though its shape appears to be distorted. Consider a circular object - it should appear elliptical when you are looking at it from a non-fronto-parallel[1] vantage point; a rectangular table top appears to be of trapezoidal shape. In addition, the degree of distortion depends on the distance between the viewpoint and the object plane.

In such "general" (that is with no restriction to the position of viewpoint) imaging situation, the degree of distortion to the original object shape could be modeled using geometric transformations like Euclidean to perspective transformation. Fortunately, there remains a set of parameters, commonly known as invariants which, as the term suggests, remain constant even after the object to image transformation has taken place.

To perform the fundamental objectives of Computer Vision such as object recognition, AGV navigation, shape description and in so many other situations it is necessary to recover the object shape under fronto-parallel (i.e. non-oblique orthographic) view point from the images under perspectivity. The objective of this thesis is to develop algorithms to recover such views. We define this to be a process of "backprojection" from the perspectivity to orthography.

[1]That is, camera or eye plane has slant and tilt angles with respect to the object plane.



Several techniques([3], [6], [5]) have already been developed to do so. However we utilise local symmetry of the object to achieve this backprojection. The essential idea of symmetry is a motion [2]: "suppose you have an object pick it up, move it around and set it down. If it is impossible to distinguish between the object in its original and final positions, we say that it has a symmetry." The line of thinking encapsulated by the quotation leads inexorably to modeling symmetry using the operations of group theory. However requiring transformed object to be "impossible to distinguish" from the original is far too restrictive both for computer vision and for human perception. Real objects such as faces, pears, wrenches, and the outline of fishes ([8]) are only approximately symmetric and more significantly, the shape only exhibit symmetries locally between segments of shape or pattern.

Within a plane symmetry can be of the following types :
Reflectional, rotational and translational and in the last case the pattern repeats itself in the plane thereby creating symmetries.

Our backprojection scheme exploits skew symmetry which is definitely a plane reflectional symmetry when looked at from non-fronto-parallel sense. The mathematical fact that the skew symmetry may be an accident of projection is evidently discounted. Indeed, Wagemans ([9]) has recently provided evidence that skewed symmetry is a non-accidental property of a shape that the human visual system exploits. When a robot "sees" objects all around, it can have the real shape of the object using the "removal" of distortion present in the object i.e. by backprojection and for which it can use local reflectional symmetry of objects as our effort suggests.

The entire analysis is done after detecting edges of the image of 2D objects, in reality we have used very thin objects. Canny's edge detection scheme is used successfully. However, from the implementation point of view Canny's method is highly computation and memory intensive. It is quite a challenge to implement such an algorithm under Windows programming environment in PC.

Given this, the contribution of the thesis can be summerised into following two major groups:

- implementation of preprocessing which includes implementation of Canny's algorithm and editing facilities. This is actually a kind of low-level processing as found in similar computer vision approaches.
- Backprojection algorithm.

Next we present the imaging transformations based on which the backprojection algorithm in chapter 3 is described. This is followed by the organization of the thesis.

1.1 Imaging transformation

Before we proceed further, we detail the fundamentals of imaging and surface geometry. Naturally, the basic issues have been discussed many times before in the vision literature. The plane to plane transformations describe world model to image mappings and vice versa. We begin their descriptions starting from the plane projective group.

The plane projective group A perspective transformation or *projectivity*, from one projective plane Π, to another, π, is a non-singular 3×3 matrix acting on homogeneous coordinates.

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} \tag{1.1}$$

or

$$\mathbf{x} = \mathbf{T}\mathbf{X} \tag{1.2}$$

The transformation matrix T has eight degrees of freedom because only the ratio of homogeneous coordinates is significant and there are 8 ratios among 9 elements of T.

Properties like concurrency, collinearity, order of contact (intersection, tangency, inflections), tangent discontinuities and cusps and cross-ratio are preserved under projective transformation [5].

The plane affine group In case of affine transformation the matrix T, as in equation (1.2), takes the form:

$$T = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ 0 & 0 & t_{33} \end{pmatrix} \tag{1.3}$$

Affine transformation has six degrees of freedom and is equivalent to the combined effects of translation, rotation, isotropic scaling and shear (non-uniform scaling in some direction).

Properties like parallelism, ratio of lengths of collinear or parallel segments (e.g. mid-points), ratio of areas, linear combination of vectors are invariant under affine transformation [5].

The plane similarity group This is a specialization of the affine transformation without shear and is equivalent to a Euclidean transformation composed with an isotropic scaling. This has four degrees of freedom and occurs when the world plane is parallel to the image plane i.e. under *fronto-parallel* viewing. Ratio of lengths, angles are preserved under plane similarity transform [5].

The plane Euclidean group The Euclidean transformation matrix is shown in equation (1.4). Here the top 2×2 sub-matrix of T is a rotation matrix and $\mathbf{t} = (t_1, t_2)^t$ is a translation vector. It has three degrees of freedom.

$$\mathrm{T} = \begin{pmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.4}$$

Lengths, angles, areas are preserved under the Euclidean transform [5].

Organization of the thesis

We first describe the implementation of preprocessing including Canny's approach in chapter 2. In chapter 3 we first give the mathematical framework for backprojection, then we have described the algorithm for backprojection, its implementation and results in support of the algorithmic process followed by a conclusion. This chapter is followed by chapter 4 containing an indication of future research issues in this direction.

Chapter 2

Preprocessing

In any computer vision approach some sort of preprocessing is always needed. In fulfilling our objective viz. "backprojection" we need to detect edge contours of images. These contours include many false edges as well as the required symmetric contours of the objects. Hence along with Canny's edge detection procedure we need to rub out false edges. After this, the actual backprojection process starts with the said symmetric contours.

2.1 Objective

We have already mentioned in the introduction that edge or boundary of the planar objects is to be detected first. This is actually a preprocessing or low-level processing before removal of projective distortion form planar objects. To be precise we require only the symmetric contours of the object before actual process of backprojection. Only detection of edge by Canny's approach is found insufficient for the said purpose, because, after detection of edge a number of false edges (e.g. those created by shadow of the object, those due to noise or the like.) appear. They are very difficult to manage by setting parameters of Canny's edge detector. So we have kept an editing facility over edge images,so that we can get exactly the said contours rubbing out others.

2.2 Implementation

In this section we give implementation of preprocessing in Windows3.x environment. This includes implementation of Canny's algorithm and a closer view into editing facility. Implementational details of source code files are given in the appendix.

2.2.1 Canny's algorithm implementation

Canny's edge detection approach([1]) combines several good features of edge detection schemes into an overall system which performs well on many images. In addition it is supported by a thorough analysis with well stated goals which are as follows:

- Good detection : There should be low probability of failing to mark real edge points, and low probability of falsely marking non-edge points. Since both these probabilities are monotonically decreasing function of the output SNR (signal-to-noise ratio), this criterion corresponds to maximizing SNR.

- *Good localization : The points marked as edge points by the operator should be as close as possible to the center of the true edge.*

It was shown by Canny that improving (1) and (2) corresponds to maximizing the quantity:

$$Q = \frac{\left| \int_{-w}^{+w} G(-x) f(x) dx \right| \left| \int_{-w}^{+w} G'(x) f'(x) dx \right|}{n_0 \sqrt{\int_{-w}^{+w} f^2(x) dx} \; n_0 \sqrt{\int_{-w}^{+w} f'^2(x) dx}} \qquad (2.1)$$

where, $f(x)$ = *impulse response of the filter*, $G(x)$ = edge *itself*, n_0 = *mean* noise impulse magnitude per unit length, [-w,w] is the bound for impulse response function.

Another additional constraint is needed which is as follows -

- Only one response to a single edge : This is implied in condition (1) but the above mathematical form does not contain it so we need an additional constraint on the above mathematical form. The constraint is $K_{max} = k \cdot w$, in words, the average maximum distance between two local maxima has to be some fraction of the spatial extent of the impulse response, and the assumption is that this spatial extent is finite.

The edge detector has several stages:

1. Gaussian smoothing $G * I$.
2. Directional derivative $\nabla(G * I)$ where G = Gaussian filter, I = image.
3. Non-maximal suppression.
4. Thresholding by hysteresis.

Gaussian filtering and computing directional derivatives are local operations.

Non-maximal suppression selects an edge point for which the gradient magnitude is the maximal in the direction of the gradient requiring only local operations viz. local interpolations.

Thresholding with hysteresis eliminates weak edges owing to noise and it also reduces "streaking" to a great extent. Streaking is the breaking up of an edge contour caused by the operator output fluctuating above and below a threshold along the length of the contour. Hysteresis was originally contributed by Canny. In this process if any part of a contour is above a high threshold those points are immediately output, as is the entire connected segment of the contour which contains the points and which lies above a low threshold. The probability of streaking is greatly reduced because for a contour to be broken it must now fluctuate above the high threshold and below the low threshold.

The program has been implemented in 'C' in the Windows 3.x environment in order to:

1. be able to work at ease with ".BMP" images which are taken as snapshots from video camera attached to the PC.
2. get a good amount of memory from Windows environment through its global memory resource management scheme. so that we can work comfortably with big sized gray-level images. All these implementation details are mentioned in the appendix.

Figure 2.1: Hex spanner and Lever

2.2.2 Editing facility

We allow user to rub out edges from the edge image obtained in subsection 2. We also allow user to undo the rubbing operation done latest before saving the rubbed edge image. We have also kept the option for saving the edge image in a binary ".BMP" file so that one can process this ".BMP" file in Windows PaintBrush program or the like.

2.3 Results

In this section we give original images and their edge images obtained by Canny's approach and necessary editing facility.

Figures 2.1, 2.2, 2.3 show three gray level images. The edge contours are to be detected from these images.
Running the program implementing Canny'c algorithm with different parameters produce results shown in the figures 2.4, 2.5. Figure 2.6 shows the symmetric contours obtained after rubbing out false edges from figure 2.5.

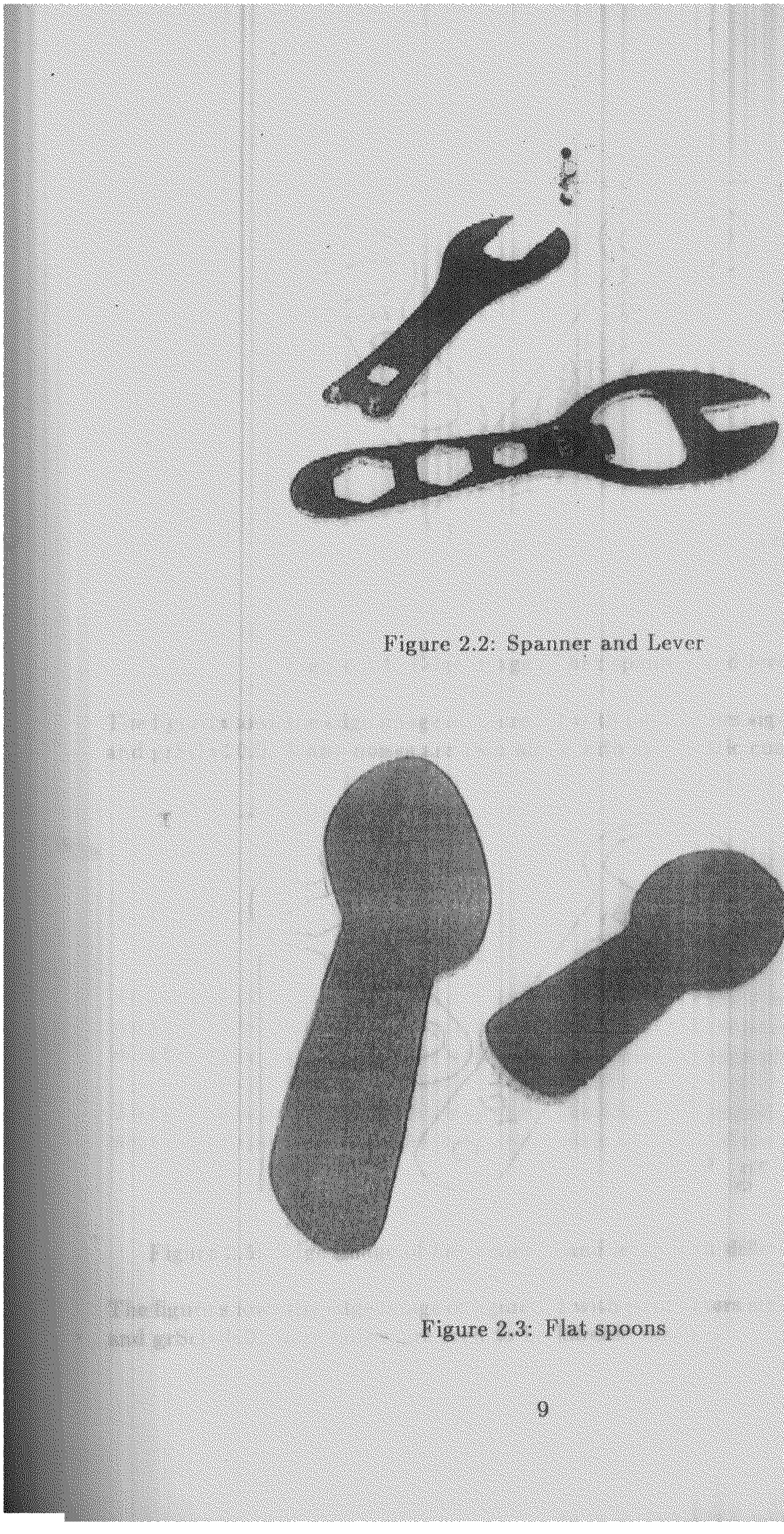


Figure 2.2: Spanner and Lever

Figure 2.3: Flat spoons

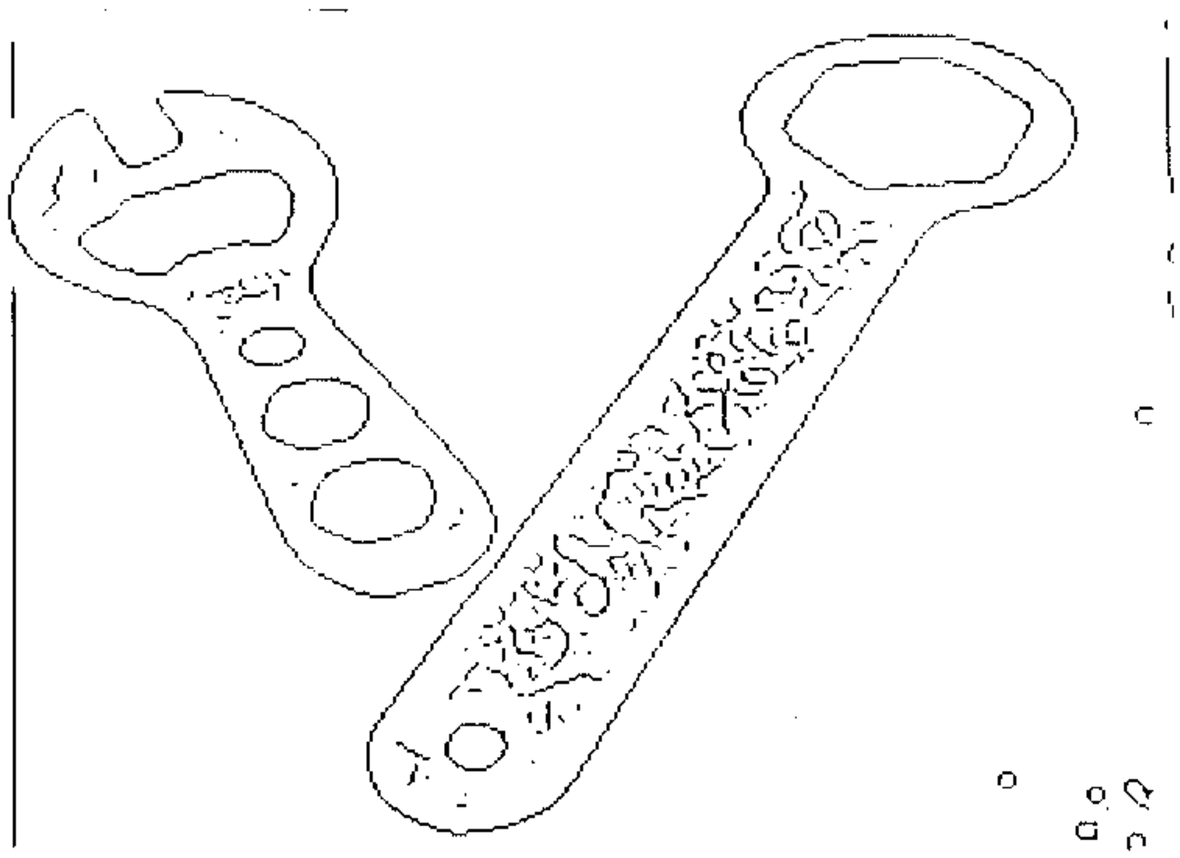


Figure 2.4: Edge image of hex spanner and lever

The figure shows the edge image of figure 2.1 with parameters sig=1.0,lt=2.0,ut=6.0 and grScl=20.0. Many noises are seen along with symmetric contours.

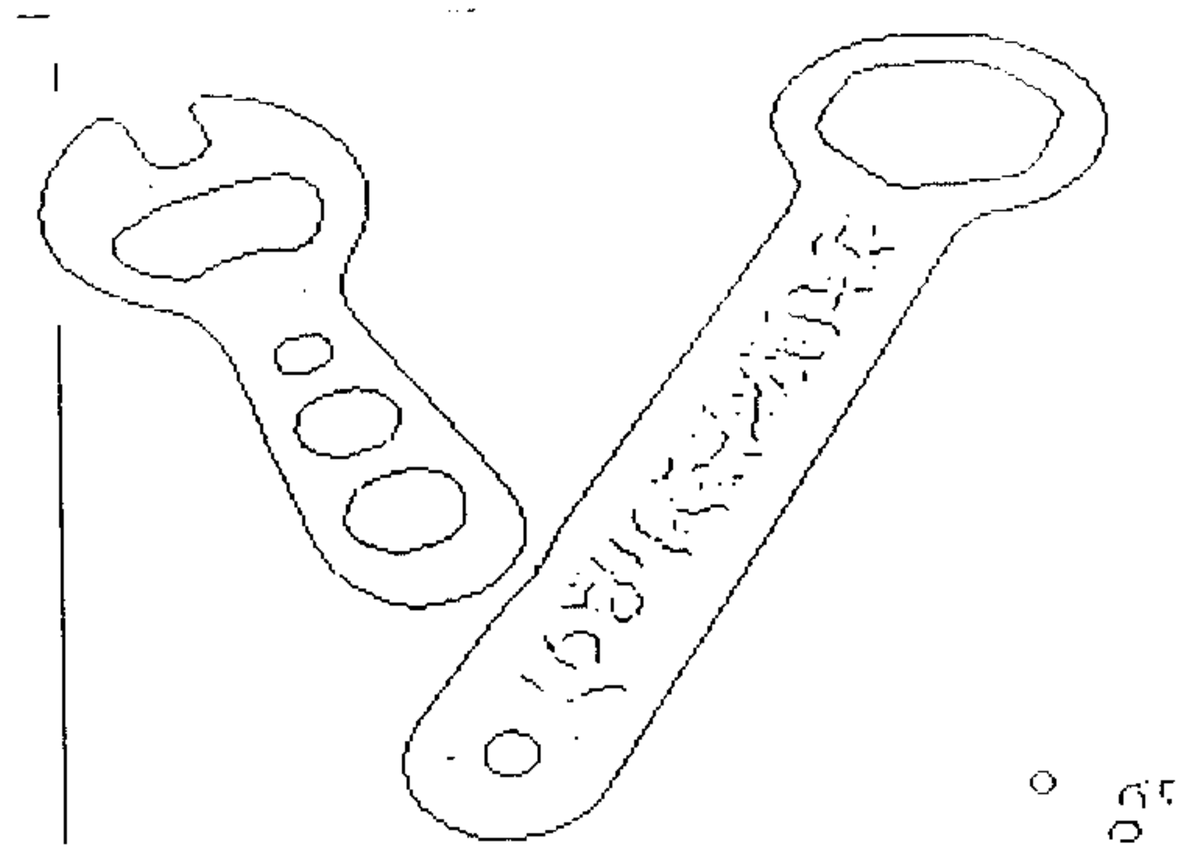


Figure 2.5: Edge image of hex spanner and lever with different parameters

The figure shows the edge image of figure 2.1 with parameters sig=3.0,lt=2.1,ut=2.4 and grScl=5.0. Here noises are seen to be lowered down.

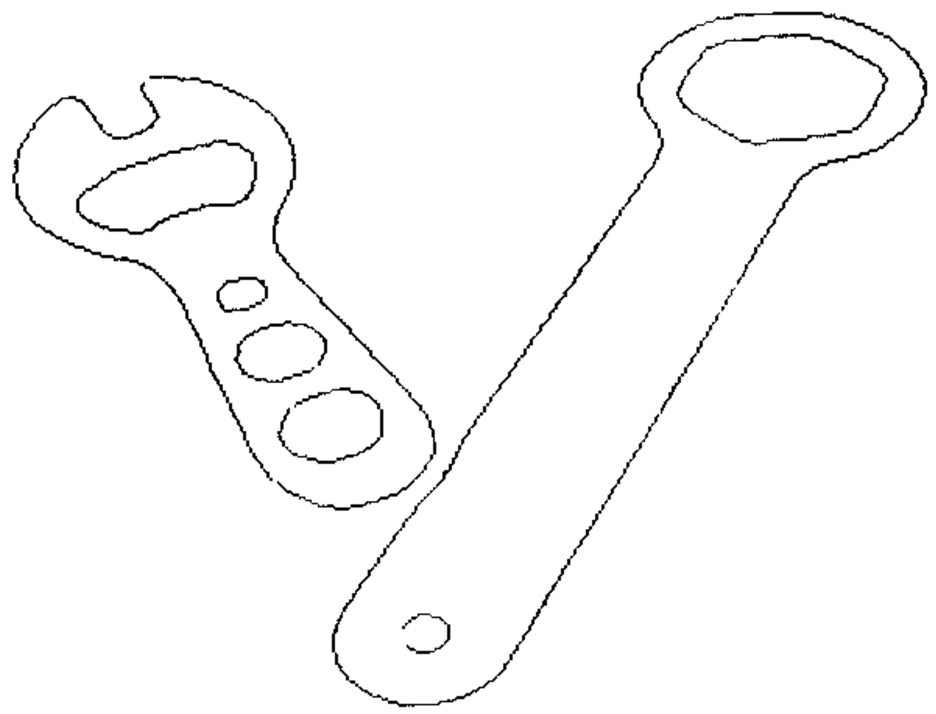


Figure 2.6: Edge image of hex spanner and lever after editing

The figure shows the edge image of figure 2.1 with parameters same parameters as that in figure 2.5 after editing.

The results illustrate that editing facility is an efficient tool along with edge detection tool for finding out symmetric contours.

2.4 Conclusion

At the end of this we get an edge image detected by Canny's approach and subsequently rubbing out the false edges by edit facility. This edge image is in the form of a file containing the x and y coordinate values of the edge pixels.

This edge image is required in the next chapter where we describe the backprojection algorithm exploiting the reflectional symmetric contours of the objects.

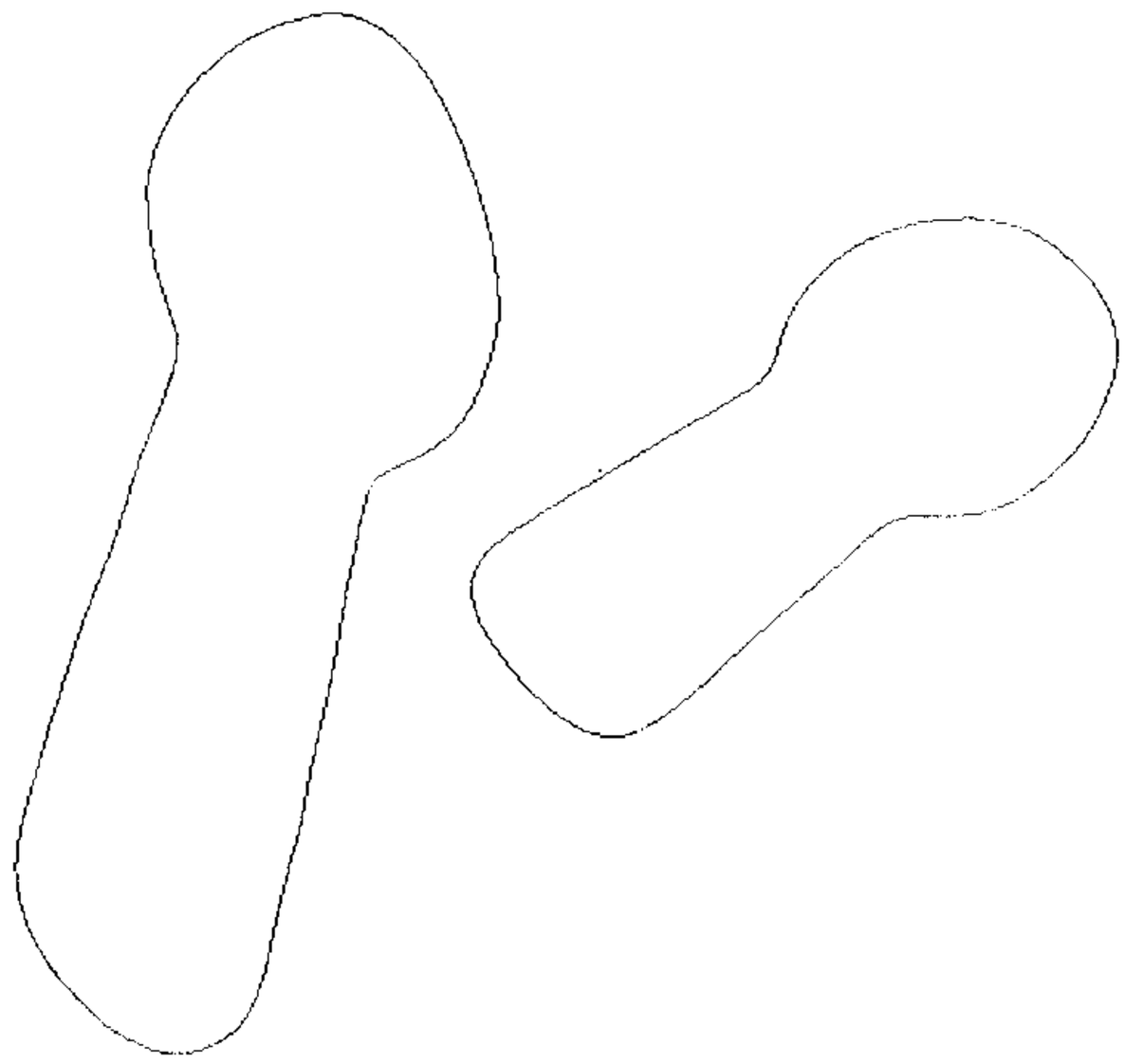


Figure 2.7: Edge image of flat spoons without editing

The figure shows the edge image of figure 2.3 with parameters sig=5.0,lt=2.0,ut=2.1 and grScl=0.001. Here noises are seen to be completely absent.

Chapter 3

Backprojection

We have already defined the term backprojection and its significance. The key issue of this chapter is to detail mathematical framework and implementation aspect of backprojection. In section 2 we give the mathematical background including backprojection theorem followed by its proof. In section 3 implementation aspect of backprojection followed by results we have obtained are provided. In the next section we give conclusion on this chapter.

3.1 Objective

Objective of this chapter is to illustrate our algorithm for backprojection which is based on a firm mathematical background and the results we have obtained following our algorithm. In this process one should also notice that the method not only works nicely for images containing the kind of objects we have assumed but the method can also be extended/modified (based on the same theory) to images containing other kinds of reflectional symmetric objects.

3.2 Mathematical Framework

In general case there is a projective transformation between object and image planes. Reflectional symmetry can be shown to constrain the transformation. This is illustrated in the following theorem -

Theorem 1 *Suppose two curves γ and γ', as in figure 3.1, are the images of two corresponding sides of a planar object with bilateral symmetry. Suppose further that image projection can be represented by a projective transformation. Then the transformation between γ and γ' has the following properties:*

1. *γ and γ' are related by a projective transformation. That is, if $\mathbf{x}$ is a point on γ then there is a point $\mathbf{x}'$ on γ' such that:*

$$\mathbf{x}' = \mathtt{T}\mathbf{x} \tag{3.1}$$

   where $\mathtt{T}$ is a non-singular 3×3 matrix, and $\mathbf{x}$ and $\mathbf{x}'$ are homogeneous three-vectors.

2. *The projective transformation $\mathtt{T}$ satisfies the following constraints:*

   (a) *$\mathtt{T}^2 = k\mathtt{I}$, where k is a scaler.*

   (b) *The fixed points of $\mathtt{T}$ are: a line of fixed points; and, a fixed point (not on the line) through which there is a pencil of fixed lines.*

   A projection with these properties is a collineation of period two, also known as a 2 cyclic homography, and a planar harmonic homology [7].

3. *The matrix $\mathtt{T}$ has eigenvectors $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$. Two of the eigenvalues, corresponding to $\mathbf{e}_2$ and $\mathbf{e}_3$ say, are equal. The third, corresponding to $\mathbf{e}_1$ is distinct and non-zero. The symmetry axis is given by the line $(\mathbf{e}_2 \times \mathbf{e}_3)$. Corresponding points, $\mathbf{b}'$ and $\mathbf{b}$, are collinear with $\mathbf{e}_1$ as shown in figure 3.1. The line $\mathbf{b}'\mathbf{b}$ intersects the symmetry axis in a point $\mathbf{b}_I$ say, and the four collinear points $\mathbf{b}$, $\mathbf{b}_I$, $\mathbf{b}'$ and $\mathbf{e}_1$ have a harmonic cross-ratio.*

4. *The transformation has four degrees of freedom. It can be determined from two correspondences.*

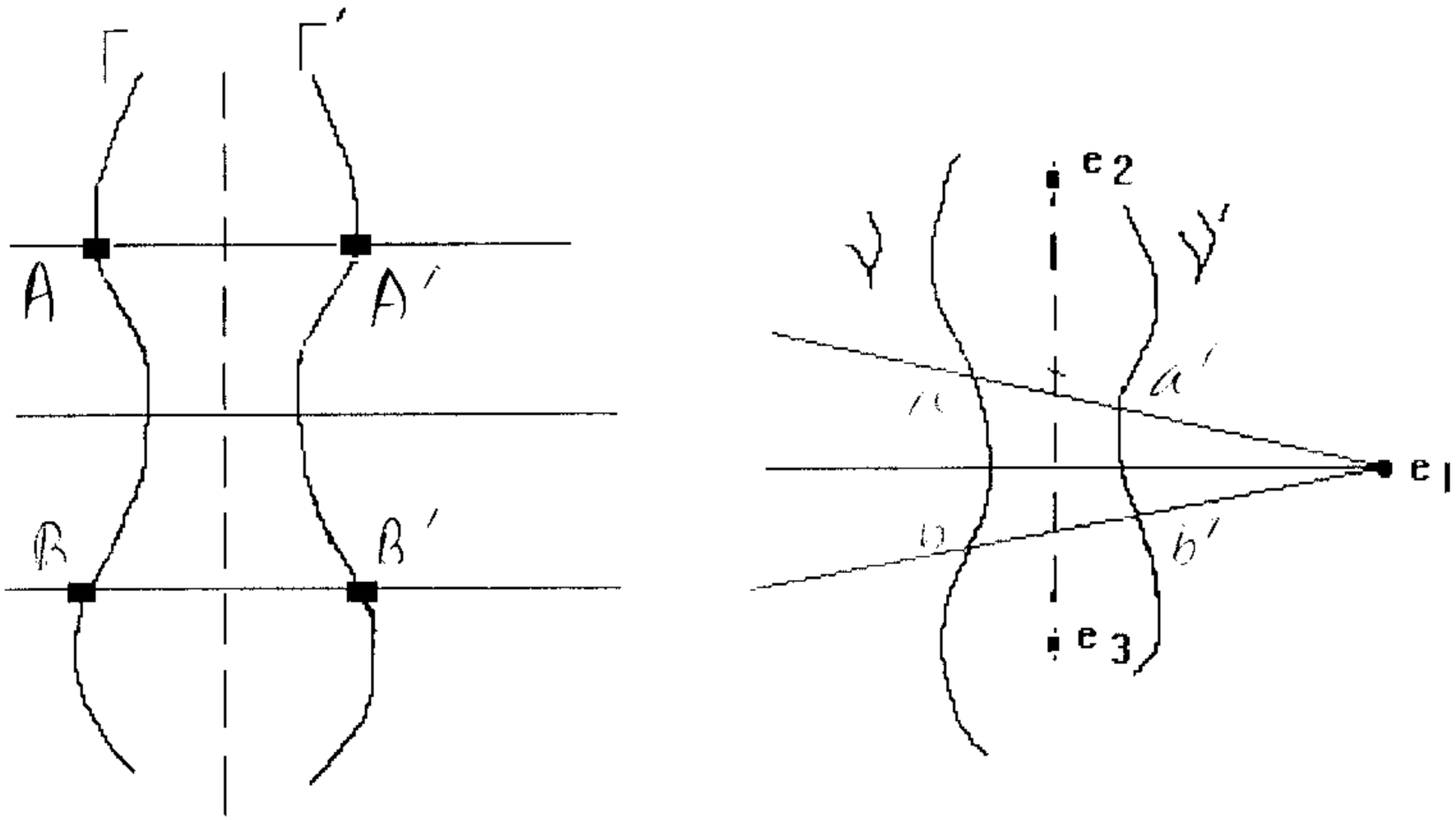


Figure 3.1: γ and γ' are the projective images of two corresponding sides of a planar object, Γ and Γ' respectively, with bilateral symmetry. Since projective transformation does not preserve parallelism and in particular mid-points, the imaged symmetry line may not pass through the mid-points of aa' and bb'. Following theorem 1, the symmetry axis is given by the line $(\mathbf{e}_2 \times \mathbf{e}_3)$. The eigenvector $\mathbf{e}_1$ is collinear to lines joining corresponding points of the imaged bilaterally symmetric object.

Interested reader will find the proof of this theorem in [4].

Theorem 2 *Suppose we have an (uncalibrated) image of two co-planar symmetric objects then the image can be backprojected uniquely, modulo a similarity, provided that the two symmetry axes are neither parallel nor orthogonal in the object plane.*

Proof

First some notation for backprojections. Suppose the linear backprojection relating the object and image planes is given by:

$$\mathbf{X} = \mathtt{U}\mathbf{x} \tag{3.2}$$

where, $\mathbf{x}$ is the 2D image vector represented in the 3D homogeneous coordinate frame, $\mathbf{X}$ is the corresponding three-vector in the backprojected planar scene, and $\mathtt{U}$ is the 3×3 linear transformation matrix with $det\ \mathtt{U} \geq 0$ responsible for backprojection. Note that $\mathtt{U}$ defines a projective transformation with eight degrees of freedom.

Referring to theorem 1, the constraints for backprojection are:

1. *The intersection of the correspondence lines* $(\mathbf{e}_1)$ *must be at infinity for parallelism in the backprojection. This is expressed projectively by requiring that* $\mathbf{e}_1$ *transform to a point* (q) *on the line at infinity,* $\mathbf{l}_\infty$.

2. *The correspondence direction* $(\mathbf{e}_1)$ *and the symmetry line* $(\mathbf{e}_2 \times \mathbf{e}_3)$ *must be perpendicular in the backprojection. This is expressed projectively by requiring that the intersection of the four points, namely, intersection of symmetry line and* $\mathbf{l}_\infty$, q, *and the two circular points* $((1, i, 0)^t$ *and* $(1, -i, 0)^t)$ *have a harmonic cross-ratio.*

The backprojection is achieved in two stages:

Stage 1. Removing the perspective distortion, leaving only affine.

Stage 2. Removing the affine distortion.

Each stage generates a 3×3 transformation matrix, and the backprojection matrix is the product of these two.

Stage 1. Removing the perspective distortion

Suppose there are two symmetries (a and b). Referring to theorem 1, the directions collinear to the corresponding points of the bilateral symmetries **a** and **b**, are given by $\mathbf{e}_1^a$ and $\mathbf{e}_1^b$ respectively. The line through these two points, say $\mathbf{l}'$, is given by,

$$\mathbf{l}' = \mathbf{e}_1^a \times \mathbf{e}_1^b = (l_1, l_2, l_3)^t \tag{3.3}$$

In removing the perspective distortion, we find the 3×3 transformation matrix, T_p, which takes $\mathbf{l}'$ to $\mathbf{l}_\infty$. A point $\mathbf{p}_\infty$ on $\mathbf{l}_\infty$ must be of the form $\mathbf{p}_\infty = (p_\infty^x, p_\infty^y, 0)$. Note that if point transforms as $\mathbf{p}' = \mathrm{T}\mathbf{p}$, then line transforms as $\mathbf{l}' = \mathrm{T}^{-t}\mathbf{l}$. Hence, T_p is given by,

$$\mathrm{T}_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix} \tag{3.4}$$

Stage 2. Removing the affine distortion

The affine distortion is removed following backprojection algorithm presented in [4]. First, the transformation T_p is applied to $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$ for symmetries **a** and **b**. Suppose, the transformed symmetry lines are given by $(\mathbf{e}_2^{a'} \times \mathbf{e}_3^{a'})$ and $(\mathbf{e}_2^{b'} \times \mathbf{e}_3^{b'})$ and the vectors representing symmetry correspondences are given by $\mathbf{e}_1^{a'}$ and $\mathbf{e}_1^{b'}$ for symmetries **a** and **b** respectively. By construction, $\mathbf{e}_1^{a'}$ (and, also $\mathbf{e}_1^{b'}$) is of the form $(p, q, 0)$ which is a line parallel to $y = (q/p)x$.

If, we express the symmetry line ($(\mathbf{e}_2^{a'} \times \mathbf{e}_3^{a'})$ or $(\mathbf{e}_2^{b'} \times \mathbf{e}_3^{b'})$) and the correspondence direction ($\mathbf{e}_1^{a'}$ or $\mathbf{e}_1^{b'}$) as 2D guiding vectors, say, **m** and **n** respectively, for any bilateral symmetry, these backproject to vectors respectively parallel to and orthogonal to the symmetry axis. In the object plane (modulo similarity), therefore, the scaler product $(\mathrm{T}_a.\mathbf{m}).(\mathrm{T}_a.\mathbf{n}) = 0$, where, T_a is the 2×2 transformation matrix responsible for affine distortion. Consequently [4],

$$\mathbf{m}^t \mathrm{T}_a^t \mathrm{T}_a \mathbf{n} = 0. \tag{3.5}$$

The matrix $\mathrm{V} = \mathrm{T}_a^t \mathrm{T}_a$ is clearly symmetric and it is positive definite. Based on the methodologies developed in [4], let the components of V be given by:

$$\mathtt{V} = \begin{bmatrix} \alpha & \beta \\ \beta & \gamma \end{bmatrix}, \tag{3.6}$$

then we have from equation (3.5)

$$\begin{pmatrix} m_x n_x & m_x n_y + m_y n_x & m_y n_y \end{pmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = 0 \tag{3.7}$$

This is a linear constraint on α, β, γ. Two such constraints determine the *ratio* $\alpha : \beta : \gamma$. Referring to [4], the sign is fixed by the requirement that V is positive definite, so that trace $\mathtt{V} = \alpha + \gamma \geq 0$.

Two symmetries generate two constraint equations (3.7):

$$\mathtt{M} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = 0$$

where

$$\mathtt{M} = \begin{bmatrix} m_x^1 n_x^1 & m_x^1 n_y^1 + m_y^1 n_x^1 & m_y^1 n_y^1 \\ m_x^2 n_x^2 & m_x^2 n_y^2 + m_y^2 n_x^2 & m_y^2 n_y^2 \end{bmatrix} \tag{3.8}$$

Provided the matrix M is of rank 2 this uniquely determines the ratio $\{\alpha : \beta : \gamma\}$. It can be shown that M drops rank if any of the vectors $\{\mathbf{m}^1, \mathbf{n}^1, \mathbf{m}^2, \mathbf{n}^2\}$ are parallel, hence the clause in the theorem. Note that parallel lines in the object remain parallel in the image after affine transformation. □

Having determined $\mathtt{T}_p$ and $\mathtt{T}_a$, the composite backprojection matrix U is given by,

$$\mathtt{U} = \mathtt{T}_a . \mathtt{T}_p \tag{3.9}$$

where, $\mathtt{T}_a$ is a 3×3 matrix of the form

$$\mathtt{T}_a = \begin{bmatrix} u_{11} & u_{12} & 0 \\ u_{21} & u_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

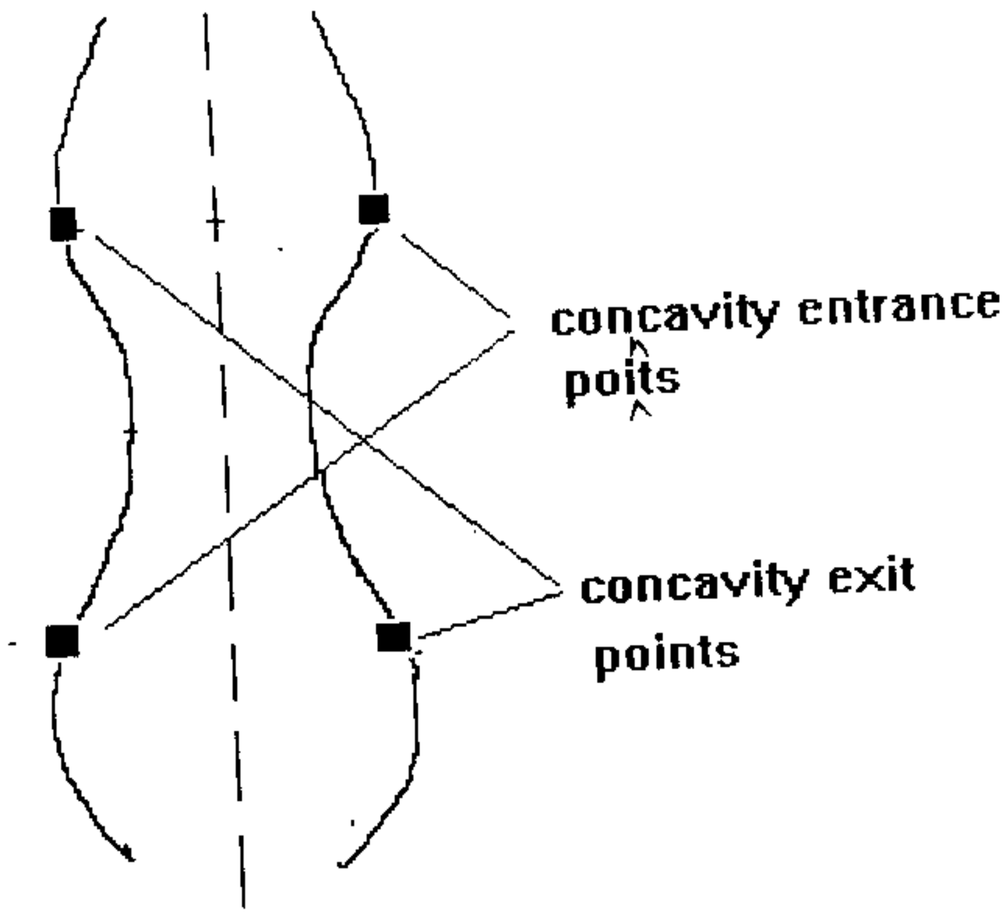


Figure 3.1: Concavity entrance and exit points

3.3 Backprojection Algorithm

Let us first restate assumptions more clearly:

- Objects are locally symmetric if not globally,
- there are at least two local symmetric concavities present in the image,
- two symmetric axes are neither parallel nor orthogonal.

At this moment we have the edge image with us and we run the following algorithmic steps on it.

Step 1: For each object in the edge image find convex hull of it and thus find four concavity entrance and exit points for each object, as shown in the figure 3.1.

Step 2: Determine T_i (3×3 matrix, see theorem 1) for each object $i(1, 2)$ from the following 8 equations:
$\mathbf{A}'_i = T_i \cdot \mathbf{A}_i$, $\mathbf{A}_i = T_i \cdot \mathbf{A}'_i$, $\mathbf{B}'_i = T_i \cdot \mathbf{B}_i$, $\mathbf{B}_i = T_i \cdot \mathbf{B}'_i$

These yields actually 8 equations for homogeneous coordinates of A_i and we set overall scale factor of T_i, by setting $T_i[3,3] = 1.0$.

Step 3: Do the following substeps:

- Find out eigenvectors of T_i say two eigenvectors $\mathbf{e}_{2i}$ and $\mathbf{e}_{3i}$ are having an equal eigenvalues and $\mathbf{e}_{1i}$ is having a different eigenvalue.
- Find symmetry lines SL_i as $\mathbf{e}_{2i} \times \mathbf{e}_{1i}$.
- Have correspondence point C_i as $\mathbf{e}_{1i}$.

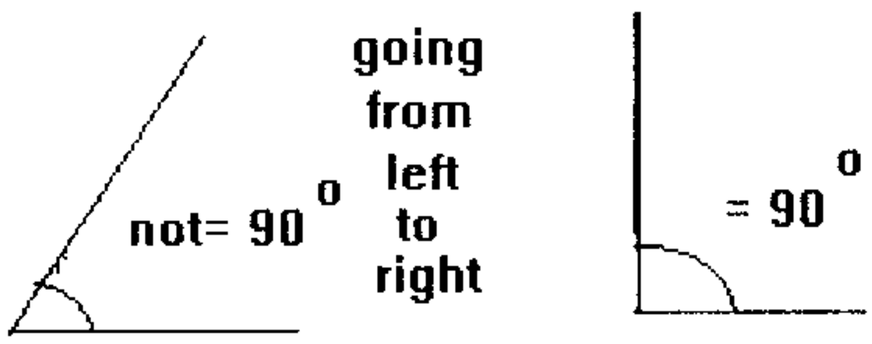
Step 4: Find T_p (as described in theorem 2)

$$T_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

where, $(l_1, l_2, l_3) = \mathbf{e}_{11} \times \mathbf{e}_{12}$.

Step 5: Find out SL_i and C_i in affine space, by the following formulae,
$\mathbf{e}^a_{ji} = T_p \cdot \mathbf{e}_{ji}$, $i = 1, 2$. $j = 1, 2, 3$. and
$SL^a_i : \mathbf{e}^a_{2i} \times \mathbf{e}^a_{3i}$.
$C^a_i : \mathbf{e}^a_{1i}$.

At this moment we are in affine space we can go from affine to similarity space through the following steps:



Step 6: Find out slope of correspondence axes for $i = 1, 2$ by the formula
$m_{c_i} = \mathbf{e}^a_{1i}[2]/\mathbf{e}^a_{1i}[1]$. for $i = 1, 2$.

Step 7: Have 4 points $\mathbf{A}'_i$,$\mathbf{B}'_i$ $(i = 1, 2)$ in affine space by
$\mathbf{A}'^a_i = T_p \cdot \mathbf{A}'_i$, $\mathbf{B}'^a_i = T_p \cdot \mathbf{B}'_i$.

Step 8: Find out equation of the line of correspondence through $\mathbf{B}'^a_i$ and $\mathbf{A}'^a_i$ having slope m_{c_i}. Let the two lines be CL^a_{Ai} and CL^a_{Bi} respectively.

Step 9: Find out intersections of SL_i^a with CL_{Ai}^a and CL_{Bi}^a respectively ($i = 1, 2$). Call these points as $\mathbf{p}_{1i}$, $\mathbf{p}_{2i}$. Assign $\mathbf{p}_{3i} = \mathbf{B}_i'^a$.

Step 10: Find out U,2 × 2 matrix and $\mathbf{b}_t$, 2 × 1 column vector as follows :

$$(0,1)_t = \mathsf{U} \cdot p_{11} + \mathbf{b}_t.$$
$$(0,0)_t = \mathsf{U} \cdot p_{21} + \mathbf{b}_t.$$
$$(\alpha,0)_t = \mathsf{U} \cdot p_{31} + \mathbf{b}_t.$$

These are six equations and for U and $\mathbf{b}_t$ we have six unknowns so we determine U and $\mathbf{b}_t$ in terms of α.

Step 11: Determine α from the 2nd object as follows Let $\mathbf{p}_{j2}^s = \mathsf{U} \cdot \mathbf{p}_{j2} + \mathbf{b}_t$, $j = 1, 2, 3$. Now alpha is solved from the equation,

$$(\mathbf{p}_{12}^s - \mathbf{p}_{22}^s) \cdot (\mathbf{p}_{32}^s - \mathbf{p}_{22}^s) = 0.$$

Step 12: Final unskewing matrix is obtained by,

$$\mathsf{T}_{uns} = \mathsf{T}_a \cdot \mathsf{T}_p$$

where, matrix T_a is given by

$$\mathsf{T}_a = \begin{bmatrix} \mathbf{U} & \mathbf{b}_t \\ \mathbf{0} & 1 \end{bmatrix}$$

These steps have been implemented in Mathematica.

3.4 Results

This section contains :

1. Figures with concavity entrance and exit points.
2. Backprojected images.

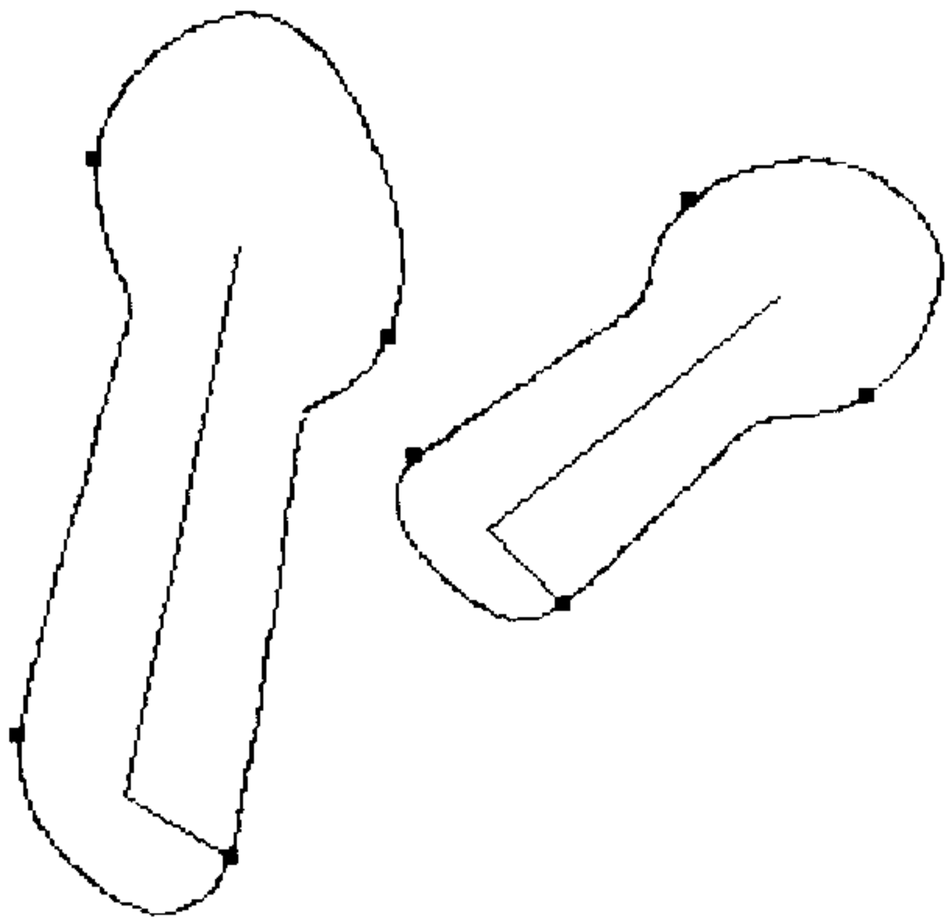


Figure 3.2: Edge image of flat spoons

The figure shows symmetry axis and concavity entrance and exit points detected by convex hull detection method.

Table 3.1: Table for spoons.

Objects	Small spoon	Big spoon
Initial angle (in degrees)	84.4	107.3
Final angle (in degrees)	90.5	90.0

Table 3.2: Table for spanner and lever.

Objects	Spanner	Lever
Initial angle(in degrees)	62.7	96.8
Final angle(in degrees)	89.6	90.0

Figure 3.2 shows a skewed symmetric object and figure 3.3 shows the result we have obtained using backprojection algorithm. Similarly figures 3.4 and 3.5 are one such set. We worked on another edge image (refer to figure 3.6) and obtained its unskewed edge image in figure 3.7.

In table 3.1 we have shown the angles between symmetry and correspondence axes before and after backprojection. two other such tables 3.2 and 3.3 are given for other two images.

3.5 Conclusion

In this chapter we have successfully implemented our backprojection scheme from projective to similarity space, i.e. after removal of projective distortion the aspect ratios of the objects remain same. The size of the objects get changed by a constant scale factor in both x and y directions.

This backprojection scheme has great importance and scope. We have tried to enumerate them in the next chapter.

Table 3.3: Table for hex spanner and lever.

Objects	Hex spanner	Lever
Initial angle(in degrees)	78.1	108.5
Final angle(in degrees)	90.8	90.0

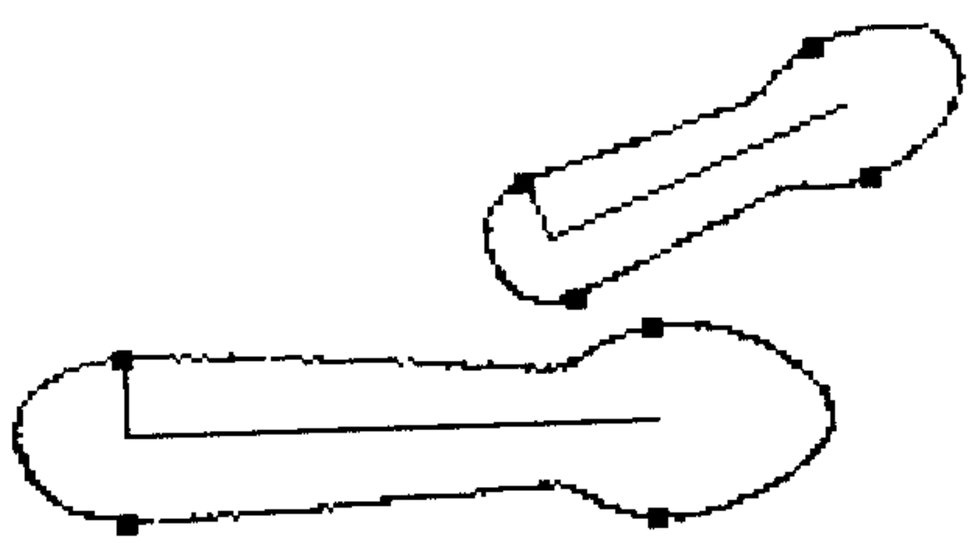


Figure 3.3: Edge image of flat spoons after backprojection

The figure shows symmetry axis, correspondence axis and concavity entrance and exit points. The two axes are seen to be approximately perpendicular for both the objects.

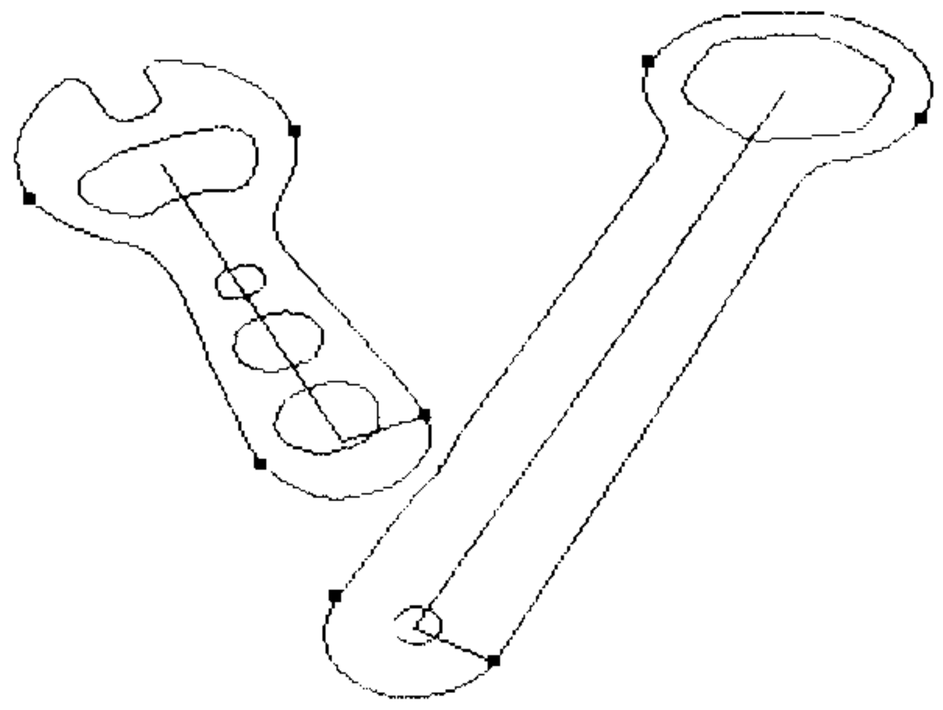


Figure 3.4: Edge image of Hex spanner and Lever

The figure shows symmetry axis and concavity entrance and exit points detected by convex hull detection method.

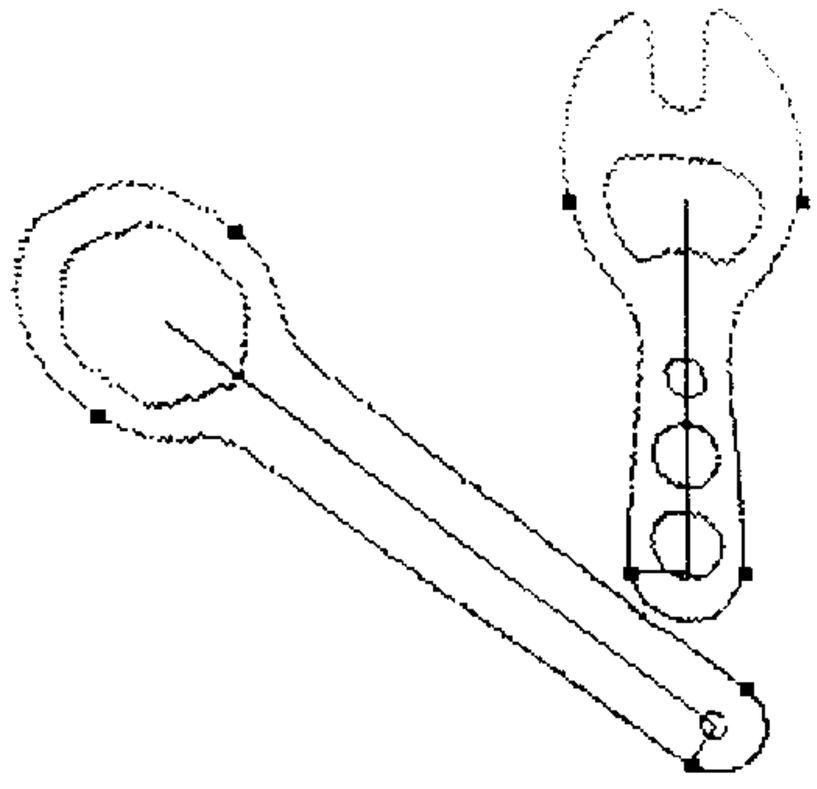


Figure 3.5: Hex spanner and lever after backprojection

The figure shows symmetry axis, correspondence axis and concavity entrance and exit points. The axes are seen to be mutually perpendicular.

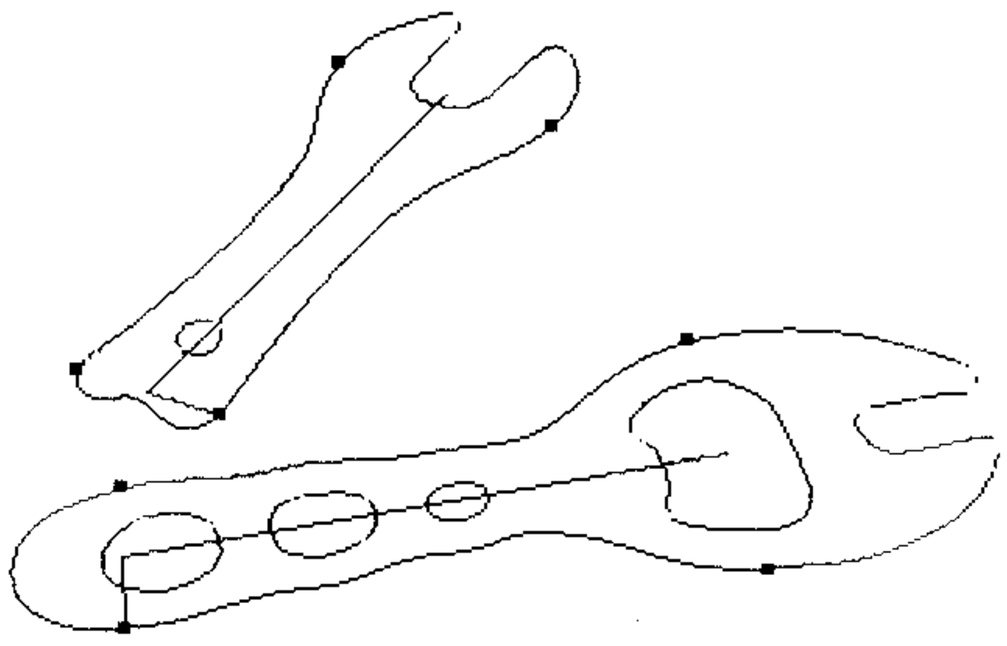


Figure 3.6: Edge image of spanner and Lever

The figure shows symmetry axis and concavity entrance and exit points detected by convex hull detection method.

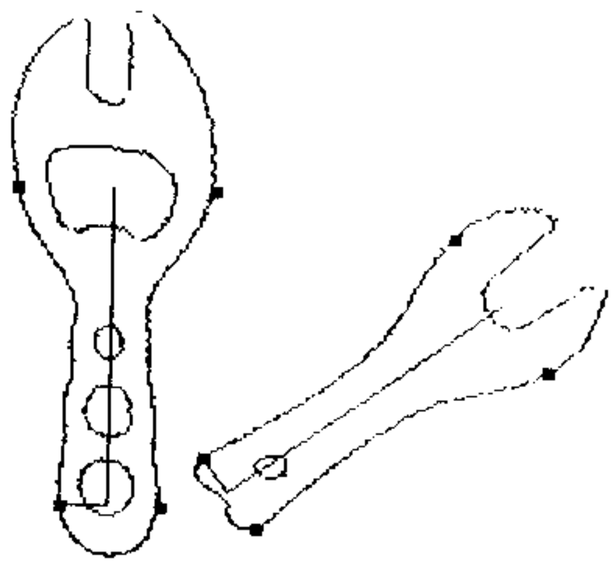


Figure 3.7: Spanner and lever after backprojection

The figure shows symmetry axis, correspondence axis and concavity entrance and exit points. The two axes in both the objects are seen to be perpendicular.

Chapter 4

Future Direction

We have successfully backprojected scenes consisting of two objects under perspective viewing condition. The method we have proposed and followed can easily be extended using least square technique to the case where the scene contains more than two planar objects. The backprojection matrices T_p and T_a could be obtained from linear least square solutions of equations (3.3) and (3.7). A similar treatment for affine images is found in [4].

Referring to theorem 1 of chapter 3, we have obtained the matrix T from 4-point correspondence. Ideally the matrix T should reflect the symmetric segment A to its counter part B, but in reality ~~T produces~~ A' is produced as seen in the figure 4.1 Elements of the matrix T can be improved if we try to minimize the disparity between segment B and A'. And this could be achieved by ***least square*** technique. As for example Levenberg-Marquardt method of least square can be applied here.

Our backprojection scheme exploits bilateral symmetry which restricts the projective transformation between corresponding image contours to a subset of general projective transformation group. Similar constraints can be obtained for other object relations (e.g. rotational symmetry).

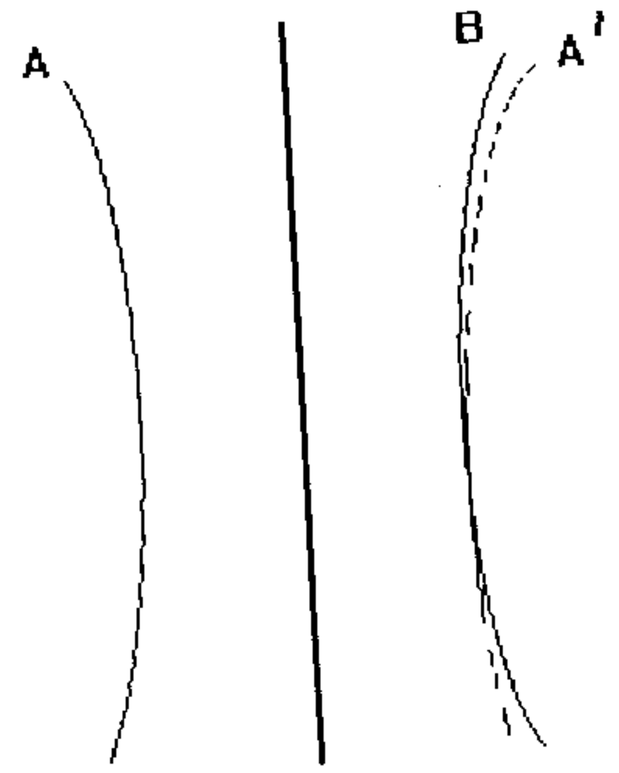


Figure 4.1: Symmetric contours

The figure shows that Symmetric contours do not overlap after being transformed by T.

Bibliography

[1] J. F. Canny. Finding edges and lines in images. *PAMI*, 8(6):679–698, 1986.

[2] M. Field and M. Golubitsky. *Symmetry in Chaos: A Search for Pattern in Mathematics, Art and Nature.* Oxford University Press, Oxford, UK, 1992.

[3] T. Kanade. Recovery of three dimensional shape of an object from a single view. *Artificial Intelligence*, 17:409–460, 1981.

[4] D. P. Mukherjee, A. Zisserman, and J. M. Brady. Shape from symmetry - detecting and exploiting symmetry in affine images. 1993. Accepted in Proc. Royal Society, Series A.

[5] J. L. Mundy and A. Zisserman(Ed.). *Geometric Invariance in Computer Vision.* MIT Press, Boston, USA, 1992.

[6] C. A. Rothwell, A. Zisserman, D. A. Forsyth, and J. L. Mundy. Canonical frames for planar object recognition. In *Proceedings of ECCV2*, pages 757 772, 1992.

[7] C. E. Springer. *Geometry and Analysis of Projective Spaces.* Freeman, New York, USA, 1964.

[8] N. J. Strachan. Recognition of fish species by colour and shape. *Image and Vision Computing*, 11(1):2–10, 1993.

[9] J. Wagemans. Skewed symmetry: a nonaccidental property used to perceive visual forms. *Journal of Expert Psychology: Human Perception and Performance*, 19(2):1–17, 1993.

Appendix

4.1 Implementational details

We have developed a GUI based edge-detection scheme which can work on ".BMP" /".DIB" images. Basic options given to the are:

1. Loading a .BMP/.DIB image in client window and scrolling if needed.
2. Detecting edge by choosing a menu, which also creates a dialogbox allowing the user to enter parameter values needed in the edge-detection process.
3. After detecting the edge one can save the co-ordinates of edge-pixels in a file whose name is to be entered by him/her.
4. After detection of edge one also can save the displayed edges (black in white background) in a ".BMP" file whose name is to be entered by the user.
5. One also can "thin" the contents of a displayed .BMP/.DIB file at any moment.
6. Editing and undoing facilities are also given to the user. The user can rub the edge image or can undo rubbing action latest done before saving his action.

The main file where all processes are actuated is "showdib.c". Other files are "canny.c","hyster.c","showedg.c","dibfunc.c"and "thin.c".There is also a header file named "showdib.h". There is one more important file viz. "showdib.rc"- a resource file.All these are coupled through BC45's project option in "Large" memory model.

Description of source code files are as follows:

1. "Showdib.c" :
   This file contains the following functions:

   (a) int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
   This is the main function which contains the *message loop* - an essential part of any Windows program. This function registers window class as

follows:
client window background is white, mouse appears as an arrow, no icon is given, and window style is "CS_HREDRAW and CS_VREDRAW"i.e., it supports horizontal and vertical scrolling.

(b) long FAR PASCAL _export WndProc(HWND hwnd, UINT message, UINT wParam, LONG lParam)
This is the *call-back* function which is actually called by Windows operating system and this is the greatest difference between dos and Windows. Windows itself calls any application through the callback "window procedure", dos has no such capability of call-back. This procedure in our case contains a message switch. On various options chosen by the user, control goes to different choices of the switch. For example when one clicks the Horizontal scroll-bar by mouse control goes to WM_HSCROLL option and necessary action scrolling is activated. In order to redraw the client area properly we have kept a memory device context which is nothing but another conceptual client window where we have kept everything drawn on our original application window. In the switch "WM_PAINT" we are actually redrawing the invalidated rectangle of client window from the conceptual window we have just mentioned. For example, say a .BMP file has been displayed in the application window. Actually the image has also been kept in the memory device context. Now when a pop-up dialogbox comes over this application client window area, the covered area is invalidated and when this dialogbox goes off exposing again the client area of our application window a WM_PAINT message is generated by Windows OS for our application and Windows OS calls our function with WM_PAINT as the value of the parameter "message" in "WndProc" and hence control goes to the switch of WM_PAINT message in our procedure WndProc and that same rectangular zone is copied from memory device context is pasted to the same rectangular zone of client area of application window. There are other message switches which are all self explanatory in the function WndProc.

(c) BOOL FAR _export EditDlgProc(HWND hDlg, UINT message, UINT wParam, UINT lParam)
This is actually a pseudo call-back function for the edit dialogbox we are creating for the user to enter values of different parameters needed in the process of edge detection. They are explained shortly after this. We have called this as a *pseudo* call-back function because Windows operating system maintains the main call-back function within itself and this procedure is only a layer over that actual call back function.

2. "dibfunc.c" :
   This file contains a number of small procedures as follows :

   (a) BYTE huge *ReadDib(char *szFileName) :
   Here the input parameter szFileName is the name of the ".BMP" file user wants to open. This function opens the .BMP file and allocates space for a huge pointer to unsigned char i.e. BYTE, loads the contents of the .BMP file in this allocated space and returns the pointer to its caller. On success return value is 1, otherwise it is 0.

   (b) BYTE huge *GetDibBitsAddr(BYTE huge *lpDib) :
   Input parameter is lpDib which points to the loaded .BMP image. Output is the address of the actual bitmap which resides after the header and color pallete table.

   (c) WORD GetDibHeight(BYTE huge *lpDib) and WORD GetDibWidth(BYTE huge *lpDib) :
   Both these functions return the height and width of the .BMP image loaded in the memory pointed to by the input parameter lpDib.

   (d) DWORD GetInfoHeaderSize(BYTE huge *lpDib) :
   This function returns the information header size of a .BMP file loaded in memory space pointed to by the input parameter lpDib.

   (e) void DrawBitmap(HDC hdc, HBITMAP hBitmap, short xStart, short yStart, short Width, short Height) :
   This function actually draws a Bitmap (given as input bitmap handle hBitmap) (a *bitmap*, is nothing but an array of bits containing picture pattern i.e., pixels of an image), on the input device context hdc and other short type parameters specify the start and end positions of the bitmaps and hdc.These functions use mighty "BitBlt" and "StretchBlt" functions.

   (f) void EraseClientWindow(HDC hdc, WORD Width, WORD height) :
   This function simply erases the client window. Handle to the device context of the window is passed as an input parameter along with width and height of client window.

   (g) int SaveAsBmp(LPSTR szOutFile,DWORD width, DWORD height, LPSTR TmpFile)
   Here I/P parameters are:

      i. szOutFile : output .BMP file name.

ii. width and height : width and height of the image whose edges have been detected.

iii. TmpFile : temporary file name where co-ordinates of edge pixels.

Output value is 1 on successfully saving edgels as a .BMP file,otherwise 0. This function simply creates a .BMP file with header size 62 (14 for File Header structure,40 for InfoHeader structure,8 for a color pallete having only black(0) and white(255). And then from the input "TmpFile" it gets edge pixel co-ordinates and loads in array either 0 or 1 (index of color pallete. and finally writes the contents of the array in szOutFile.

(h) int LoadImage(DWORD huge*huge*image, BYTE huge *lpDib):
I/P parameter is:

- lpDib : .BMP image pointer to a memory space where .BMP file has already been loaded.

Output parameter is:

- image : 2D image array which is to be loaded by this procedure from the input lpDib.

return value : 1-on success, 0-on failure. This function explores the lpDib bitmap and loads image with RGB value as indicated by the color pallete of lpDib. This "image" is needed by the edge detection routines.

3. "canny.c" :

This file contains several functions which collectively implements Canny's edge pixel detection method. The functions are given below:

(a) int input(BYTE huge *lpDib):
The input parameter "lpDib" contains the DIB(device independent bitmap) of the image on which we want to run Canny's edge detection method. This function allocates memory for several static global variables used within the file. Memory allocation is done through Global memory allocation library functions. They are called *global* because they allocate memory from global heap. This function also calls "LoadImage" function in order to load image from the DIB lpDib.
Return value is 1 on success, 0 on failure.

(b) void FreeMem():
This function frees up memory allocated in this file.

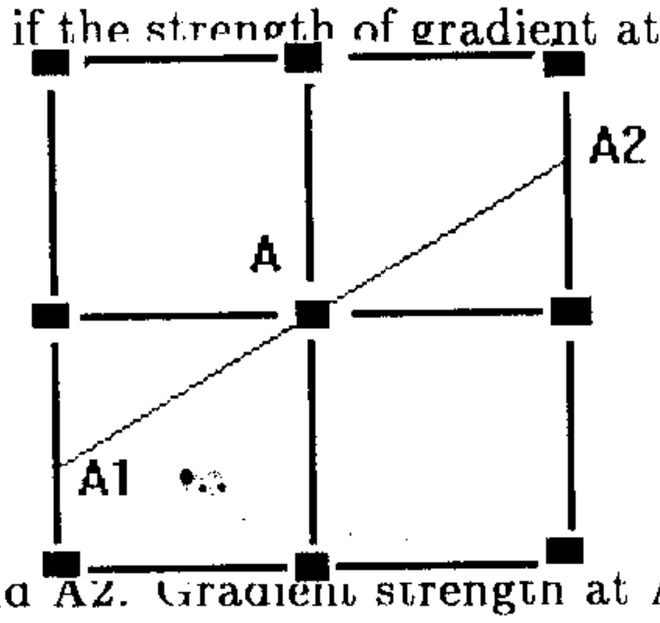
(c) static float gprof(float x, float s2):
This function returns the Gaussian $G(x)$, with s2= twice the square of standard deviation.

(d) DWORD getprof(float sig):
The input parameter contains the value of standard deviation provided by the user. This function repeatedly calls gprof and assigns the value returned by this to an array "profile" which contains Gauss profile in 1D. Return value is the size of Gauss profile.

(e) int gauss(DWORD psize):
Input parameter "psize" is the Gauss profile size returned by getprof. This function computes the convolution of profile with image, and stores the result i.e. $G * I$ in an array named "fimage1".

(f) void grad():
This function computes gradient of fimage1 ie. $G * I$ and stores x-component of the gradient in fimage2, y-component in fimage3 and overwrites fimage1 by keeping the strength or magnitude of the gradient.

(g) int nonmaxsup(char * szOutFile):
szOutFile is the output file name on top of which number of output edgels or edge pixels is written by this function and next x-coordinates,y-coordinates, magnitude and strength of the gradient are written for all pixels by this routine. it actually does the following job:
'A' is output if the strength of gradient at A is found to be greater than



that at A1 and A2. Gradient strength at A1 and a2 are found by linear interpolation. In fact this function shifts the output pixel centre by a fraction.

This function actually creates a linked list of edgels. Each node of the list contains the four information for edgels we have already mentioned. For maintaining the linked list we have two more functions.

(h) edgel_list huge* Add_Edgel(float data[4], edgel_list huge *list):
This function creates a node containing data and adds this node in front of the node pointed to by "list."

We actually have a pool of memory space allocated in our program from which we have allocated the memory space for each node of the linked list. Our programming experience says that in Windows environment allocating a big chunk and working with it is much easier than allocating a number of small chunks and working with it. This is why we have kept a memory pool for creating the linked list.

(i) void huge *AllocFromPool(DWORD dwSize):
This function allocates memory from the pool we have just mentioned.Input parameter "dwSize" contains the size in bytes of the memory to allocated from the pool.

(j) int Canny(BYTE huge *lpDib, float s, LPSTR szOutFile):
This is the main file in the file "canny.c." This function integrate all procedures for edge detection in this file.
Return value is 1 on success and 0 on failure.

4. "hyster.c":
This file contains many edge detection supporting functions which we describe first:

   (a) int initializeStack(void):
   This function initializes stack pointer to zero and allocates memory for stack array.

   (b) void DeleteStack(void):
   This function deallocates memory and initializes stack pointer to zero.

   (c) int pop(void huge* data, UINT dataSize):
   It pops up "dataSize" number of bytes from stack and assigns the address of the first byte to "data."
   Return value is 1 on success and 0 on failure.

   (d) int push(void huge* data, UINT dataSize):
   Parameter description is the same as that of pop. This function pushes "data" on stack.

We need this stack functions because we have simulated a recursive function "follow" by stack and making it non-recursive in nature. This is done because in PC environment depth of recursion cannot exceed a large value.

Now comes the actual functions by which the process of hysteresis has been implemented.

(a) int inedgels(int hInFile):
Input parameter, "hInFile" is a handle to a input file, which is output by canny.c file. This function allocates memory for all internal data structures and initializes them.
Return value is 1 on success and 0 on failure.

(b) int linkedgels(void):
This function tries to keep sufficient information for making edge contour. It first tries to connect current pixel to the others by 4-connectivity. If 4-connectivity is not found the 8-connectivity is considered.
Return value is 1 on success and 0 on failure.

(c) int addlink(DWORD edgel, DWORD to):
This function links "edgel" to "to" and vice versa by creating nodes for the linked list. This function is repeatedly called by the function linkedgels.
We now have sufficient information for realizing an edge contour. So we run the actual process of hysteresis by means of the following functions:

(d) int DoHysterresis(void):
and

(e) int follow(UINT to, UINT from):

(f) int hyster(float low, float hi, float grScl, LPSTR szOutFile):
This the main function in the file "hyster.c." This integrates all other functions we have described.
Input parameters:

- low: Contains the lower threshold value.
- high: Contains the higher threshold value.
- grScl: Contains another threshold value that is used by a function outedge after the process of hysteresis is completed.
- szOutFile: Name of output file which contains the output edgel description by means of four parameters we have already described.