M. Tech. (Computer Science) Dissertation Series

Analysis and Vectorization of Line Drawings

a dissertation submitted in partial fulfillment of the requirements for the M. Tech. (computer Science) degree of the Indian Statistical Institute

By

Rajeev
(MTC9625)

Under the supervision of

Dr. B. Chanda

Certificate of Approval

This is to certify that the dissertation work entitled **Analysis and Vectorization of Line Drawing** submitted by Rajeev . in partial fulfillment of the requirements for M.Tech in *Computer Science* degree of the *Indian Statistical Institute* , is an acceptable work for the award of the degree.

Date :

29th July 1998

[illegible]

(Supervisor)

P. K. Nandi 29/7/98

(External Examiner)

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. B Chanda for supervising this project and posing this interesting problem. I am thankful to him for providing me with valuable suggestions during the course of the dissertation work. This work would not have been realised without the help of Mr. K. Rajesh Babu , Mr. Dinesh Pande , Mr. Dipankar Saha and my other batchmates.

[illegible]
29/7/98

CONTENTS

INTRODUCTION

In any Industrial design and production activity, proper technical documentation is one of the major components. In this context engineering drawing play a very vital role. Engineering drawing being graphical in nature, represents complex information in a very concise manner. Any typical project of reasonable magnitude could consist of well over 10000 drawings as part of the documentation with design lives ranging from about 20 to 30 years.

Most of the companies file a large number of such drawing in their cabinet ,and roughly 20% of these are active each year. A large number of man hours are required in creating, updating and maintaining these drawings using conventional drafting techniques. In the present time where time is money, no company would like to expend so many man hours just for this purpose. Rather the company would prefer expensive computerised design or drafting systems, which provides an efficient means for creating, storing and updating engineering drawings.

Mechanical engineering companies that implement computer aided design (CAD) and manufacturing (CAM) systems must convert their archives of paper drawings to a format suitable for CAD system. Digitisation of a drawing creates an image of several million black and white pixel that represent the original drawing with varying accuracy, depending on the resolution of the scanning device and the quality of the original drawing. However this raster image information is not directly suitable for CAD systems, which operate with basic structures such as lines and curves.

Manual entry of drawing into a computer database is slow, expensive and tedious process. Hence one feels the need for an automated system that would convert raster images to vector representations suitable for vector editing. This would provide a convenient bridge between the manual and the computerised world.

This work presents the implementation of a system for processing scan-digitised engineering drawings wherein the lines structures are fitted with CAD primitives such as straight lines segments, circles and circular arcs.

Automated recognition and understanding of engineering drawing is by far a more complex task requiring the capacity for visual perception and intelligent interpretation. Some of the recent efforts aimed at understanding engineering drawings, include works by Haralick and Queeny [1], Sato[2] and others [3].

PROBLEM DESCRIPTION

The problem :

Line drawings need to be converted to a format suitable for computer aided system. Digitisation of a drawing creates an image consisting of a few million white and black pixels which represents the original drawing. The accuracy will be governed by the resolution of the scanning device and the quality of original drawings. Generally, the digitised image suffers from the minor deficiencies of scanning due to one or more of the following factors:

1. Unequal illumination of the document.
2. Noise due to illumination fluctuation.
3. Spatial bandwidth limitations of the scanning system.
4. Poor quality of the original drawing.

However this raster image information is not directly suitable for CAD systems which operates with basic structure such as lines and curves. So the problem is to vectorize the digitised line drawings into lines and curves. In other words, from the raster image (black and white pixels) we have to first identify whether the pixel point belongs to the background or is a part of straight line segment or is a part of curve. Next we try to store the end points of the straight lines and three points of the arc. This gives us two fold advantages of

1. compressing the large raster image into a simple data file, and
2. making it suitable for further processing by computer (CAD system).

Assumptions:

Developing such a system for any kind of line drawing is difficult because of the diversity of contextual knowledge. We would prefer to design this system for a subset of line drawings with the following basic assumptions:

1. Digitised image contains pure graphics part, i.e, it does not contain any text and dimension lines.
2. It has
   (a) straight lines (horizontal and vertical).
   (b) circular arcs.
3. The digitised image is free from noise.
4. Illumination is proper and uniform.
5. Drawing is of good quality.

Experimental data:

1. Synthetic data:

A synthetic digitised line drawing was generated using Paintbrush application program. This was done in order to be sure that the digitised image confirms to the above assumptions. Besides it is free from any of the deficiencies that are encountered during scanning the real data.

2. Real data:

In order to check the validity of this system, we run the algorithm on several real image data. These real data line drawings are being taken from the engineering drawing book by N.D. Bhatt. These line drawings were scanned at 200 dpi. Then the text portion as well as dimension lines are erased from the digitised image maximally. So whatever is left is pure graphics.

DESCRIPTION OF ALGORITHMS

In this section we go through the basic steps and algorithms used for the realisation of this system. We provide mainly the description of each step with logic , the expected output and the figure depicting the validity

Following are the basic steps involved for the realisation of this system:

1. *Skew Correction* :

The document page from which we scan the line drawings might not be properly aligned with the margins of the scanner. So, lines which were horizontal or vertical in the original image might not be perfectly horizontal or vertical in the digitised image. This is a human error, but the computer does not know about it. The angle by which the image has tilted is called *skew angle*. Before going for further processing we will have to correct this skew. The idea is to find the skew angle and the sense (clockwise or anticlockwise). Once the skew angle and the sense is known we can give the digitised image a rotation (equal to the skew angle) in the reverse sense. In order to find the skew angle and the sense we use HOUGH TRANSFORM.

1.1 *Hough Transform:*

Consider a point (x_i, y_i) and the general equation of a straight line in the slope intercept form, $y_i = a x_i + b$. Infinitely many lines pass through (x_i, y_i), but they all satisfy the equation $y_i = a x_i + b$ for varying values of a and b. However, writing the equation as $b = -a x_i + y_i$ and considering the a b- plane (also called parameter space) yields the equation of a single line for a fixed point (x_i, y_i). Furthermore, a second point (x_j, y_j) also has a line in parameter space associated with it, and this line intersects the line associated with (x_i, y_i) at (a', b'), where a' is the slope and b' the intercept of the line containing both (x_i, y_i) and (x_j, y_j) on the *xy*-plane. In fact, all points lying on this line correspond to lines in parameter space that intersect at (a', b'). Figure 1 illustrates these concepts.

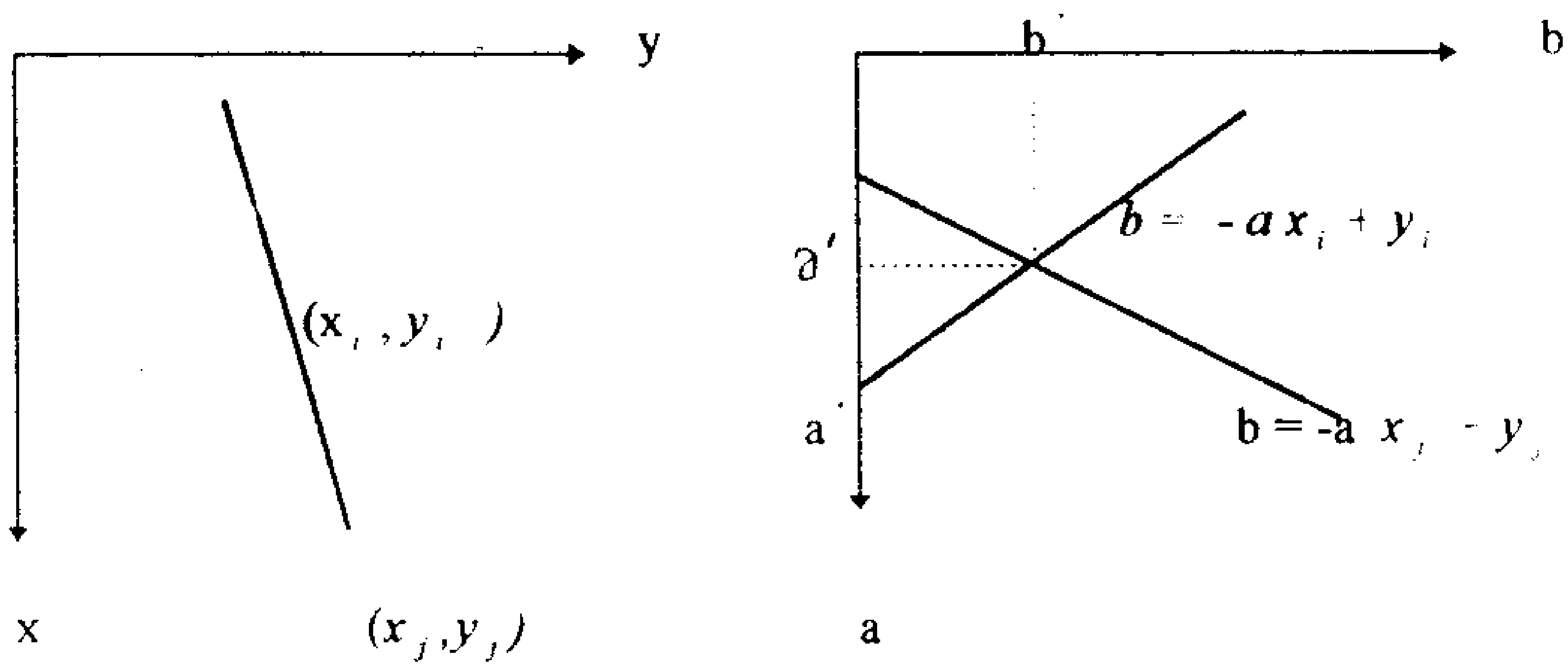


Figure 1

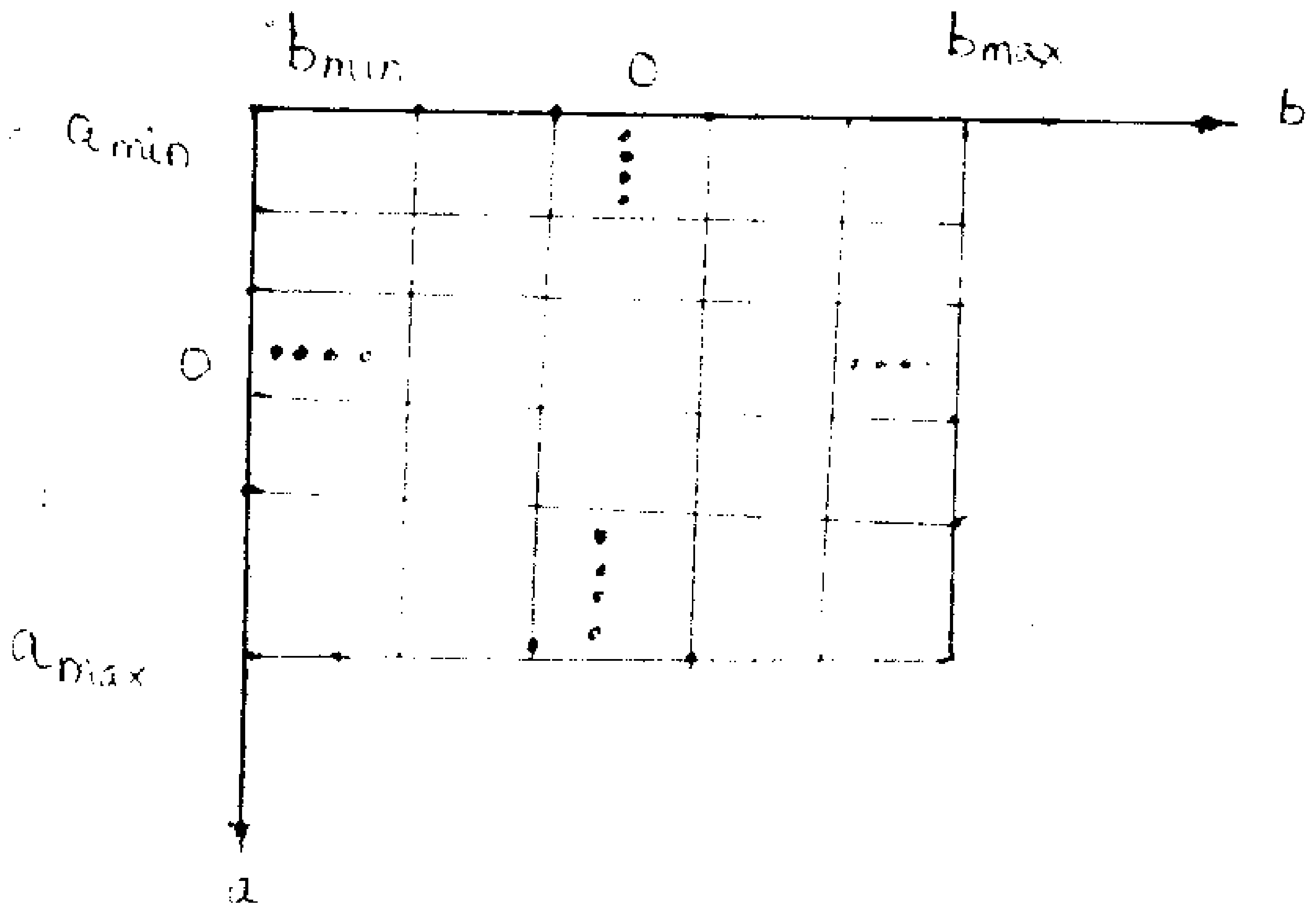


Figure 2

ıutational attractiveness of the Hough transform arises
ɔf the parameter space into so called *accumulator cells,*
ɛure 2.

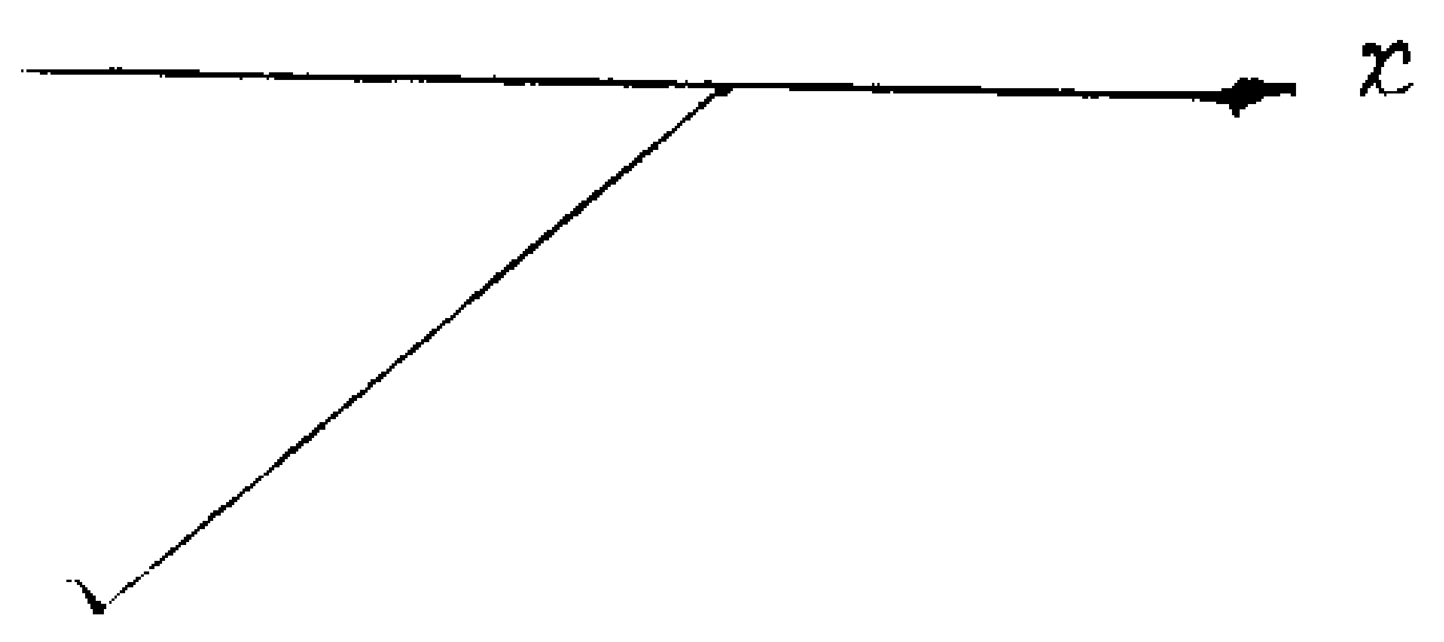


Figure 3

ax ,a min) and (b max ,b min) are the expected ranges
ɛept values. The cell at coordinates *(i,j)* with accumulator
esponds to the square associated with parameter space
b_i). Initially these cells are set to zero. Then for every
the image plane, we consider all possible values of a and
the equation $b = - a x_k + y_k$. The resulting bs are then
: nearest integer. At the end of this procedure, a value of
sponds to M points in the xy-plane lying on the line y =
accuracy of the collinearity of these points is determined
subdivisions in the ab-plane. Note that subdividing the a-
ıs gives, for every point (x_k ,y_k), K values of b
the K possible values of a. With n image points, this

method requires nK computations. Thus the procedure just discussed is linear in n, the number of points.

But a problem arises when we use the equation $y = a x + b$ to represent a line. The slope approaches infinity as the line approaches to be vertical. So we use the parametric equation of the line, i.e.,

$$x \cos\theta + y \sin\theta = p$$

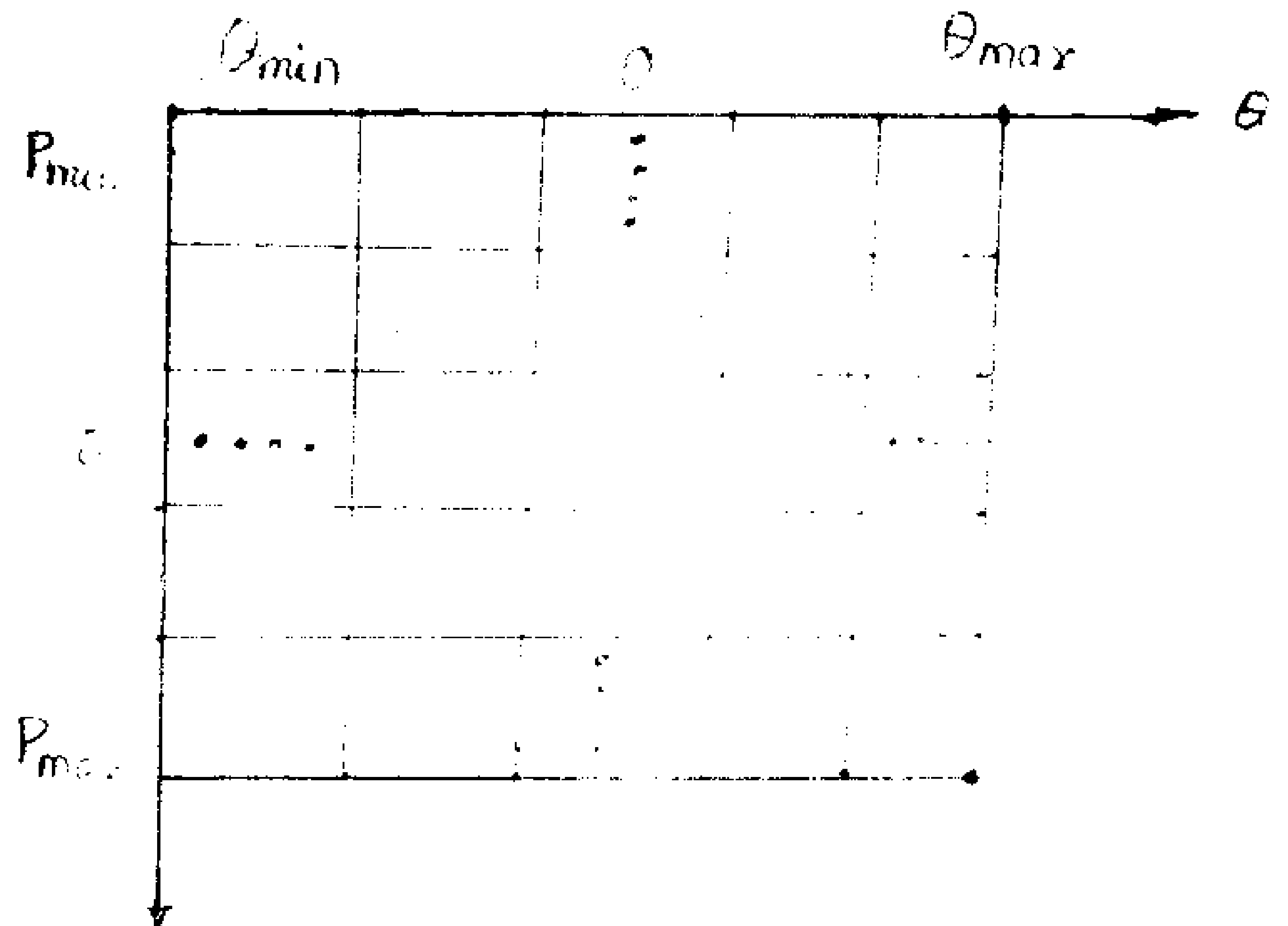
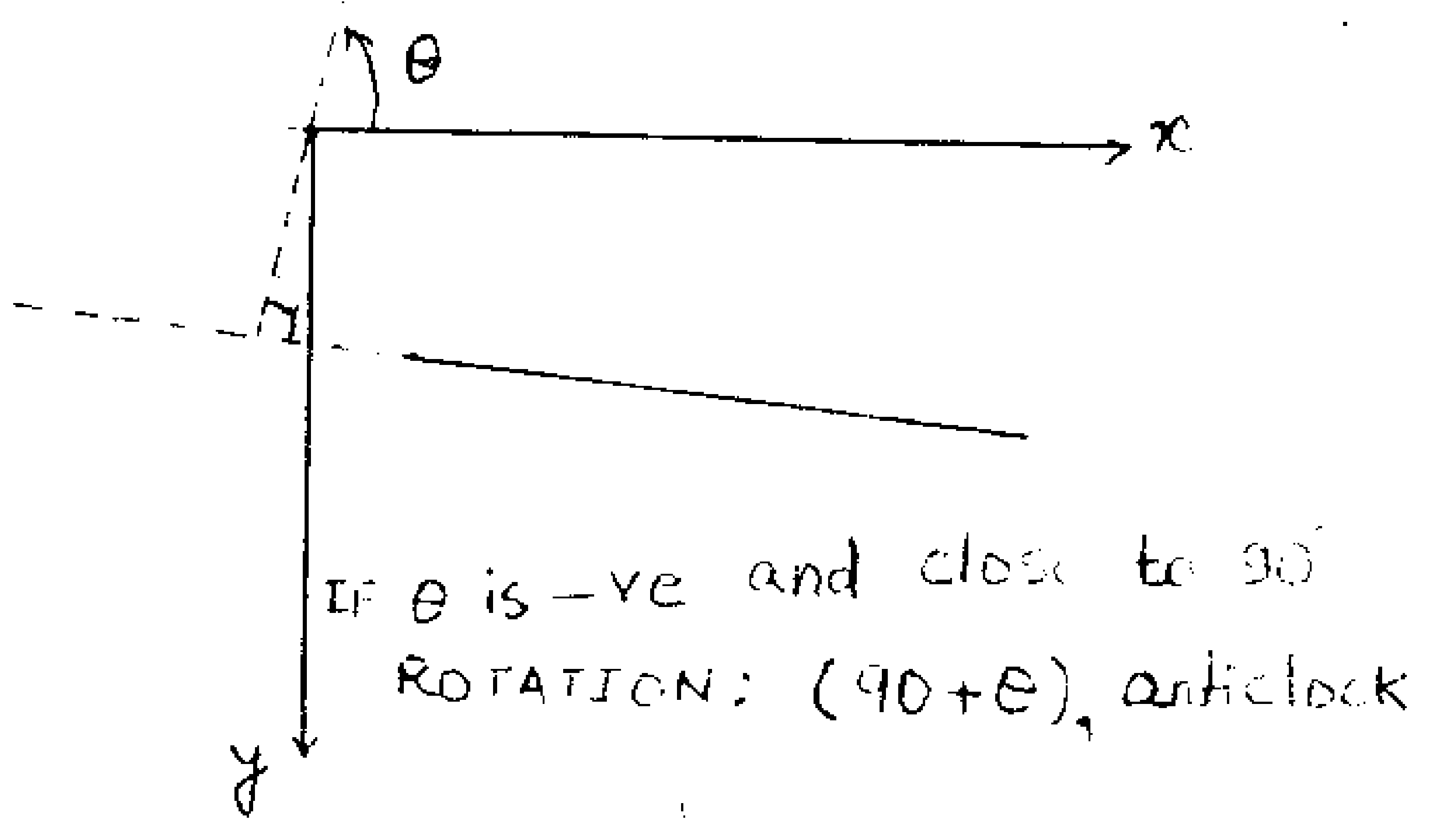
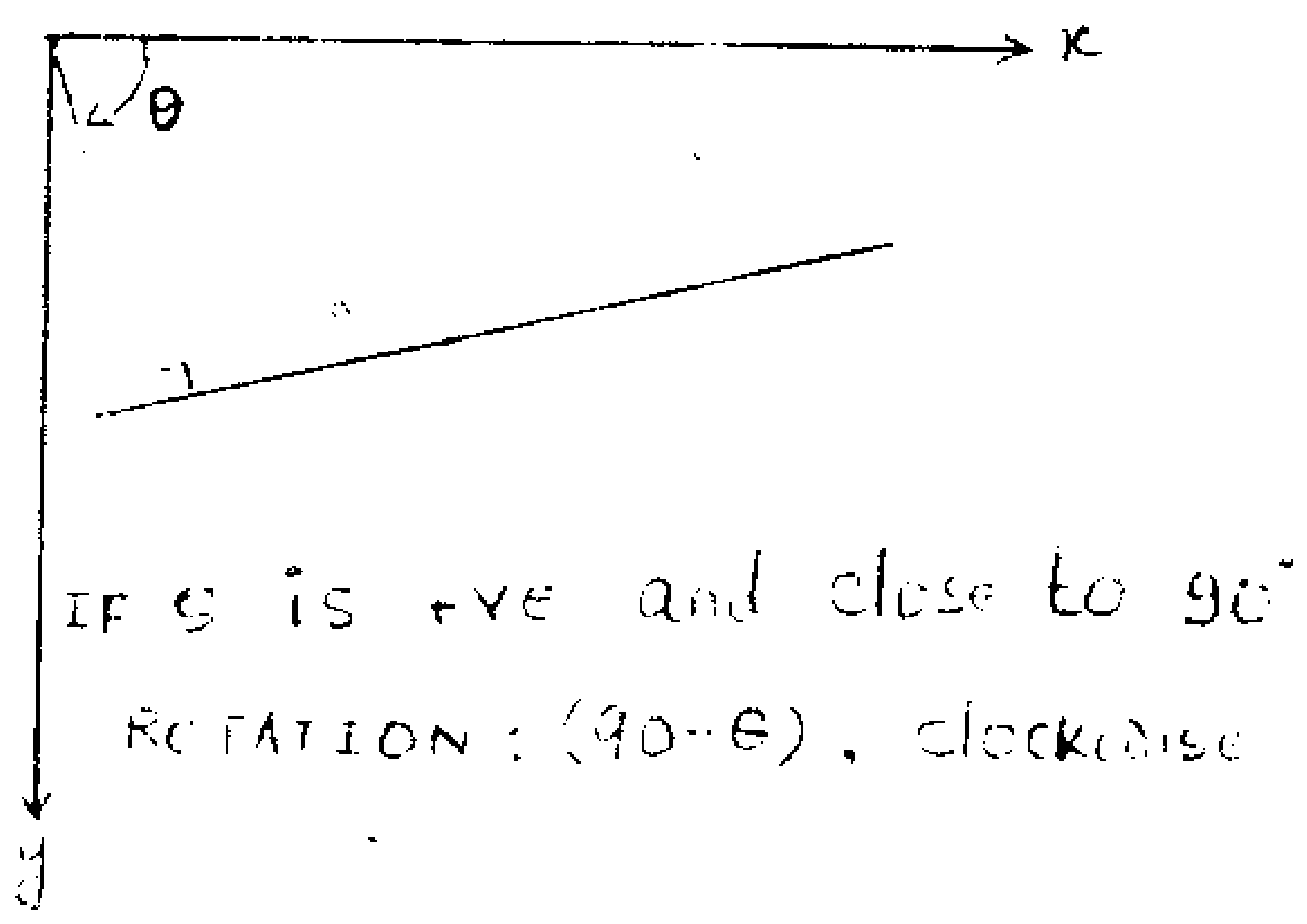


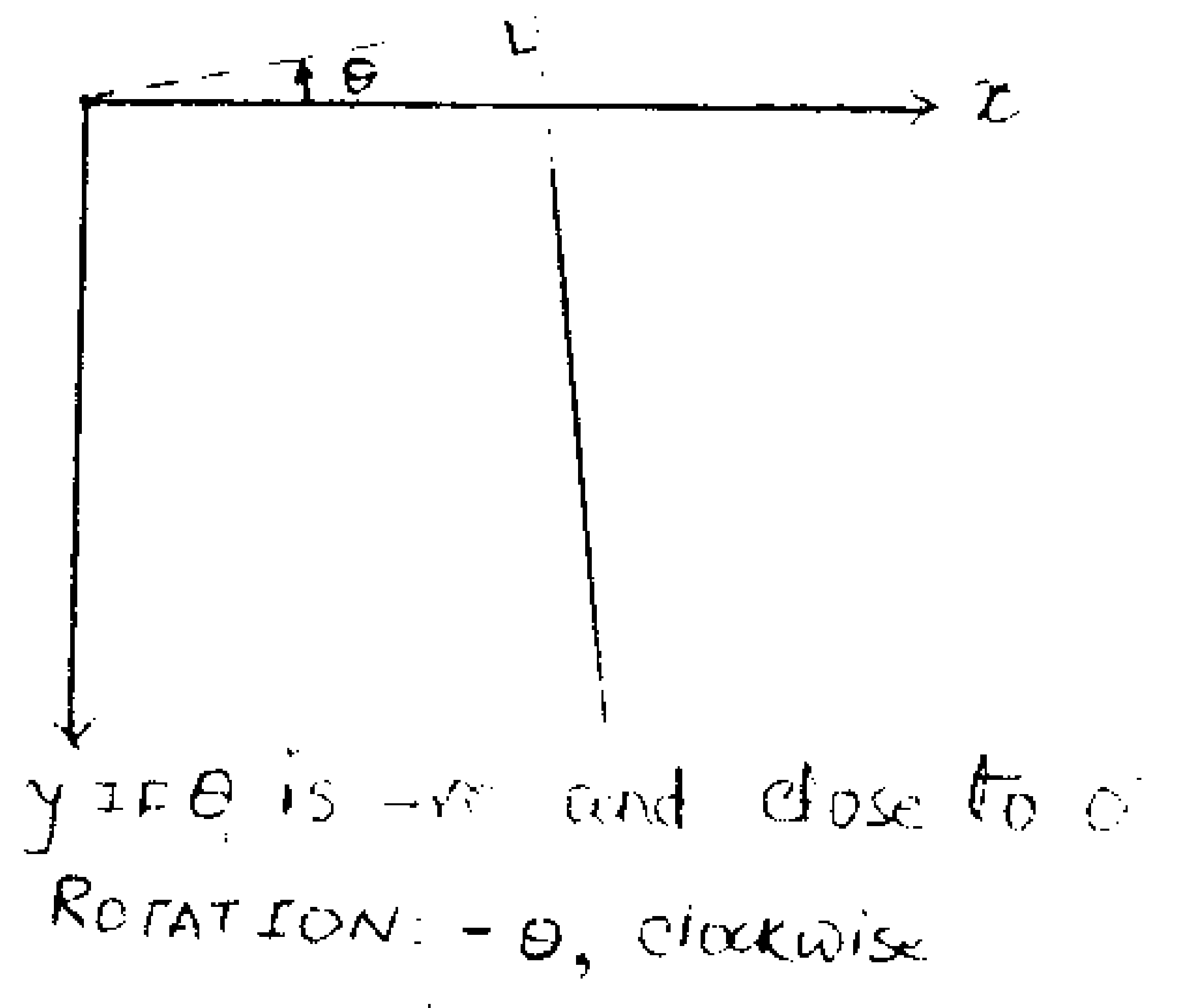
Figure 4

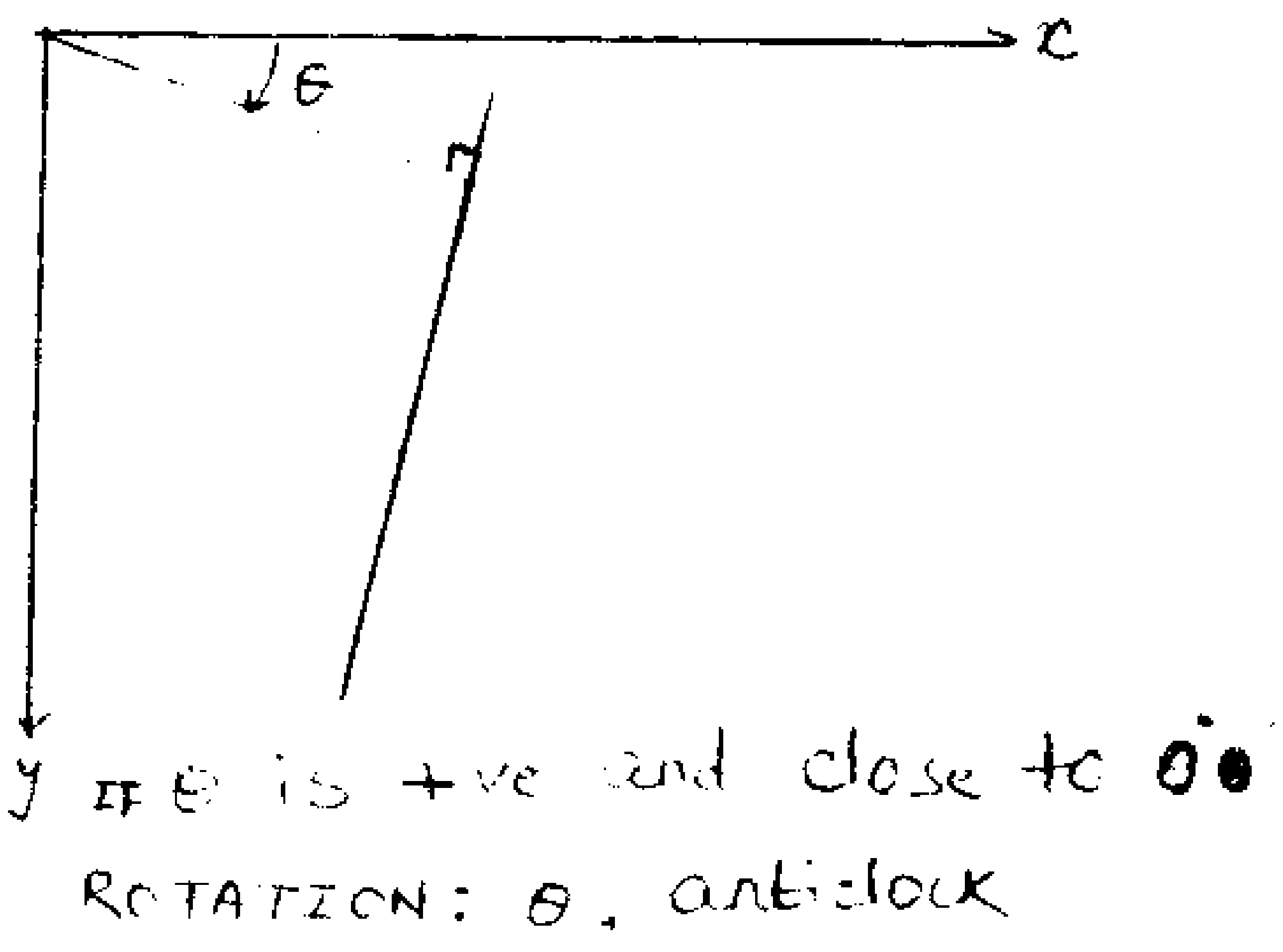
Figure 3 shows the meaning of the parameters used in this equation. Here each point (x, y) in the xy-plane corresponds to a sinusoidal curve in the $p\theta$ plane. As before, M collinear points lying on a line $x \cos\theta_j + y \sin\theta_j = p_i$ yields M sinusoidal curves that intersect at (p_i, θ_j) in the parameter space. Incrementing θ and solving for p gives M entries in accumulator A(*i,j*) associated with the cell determined by (p_i, θ_j). Figure 4 illustrates the subdivision of the parameter space.

Now the line drawing generally contains parallel lines. All these parallel lines will have same value of θ but different value of p. The next step is to find the global maximum from this two dimensional p θ-array. This maximum value will correspond to horizontal lines if number of pixels on horizontal lines are more than that on vertical lines or it corresponds to vertical lines if number of pixels on vertical lines are more than that on horizontal lines. So depending on this fact and the value of θ and its sign we can find the value of skew angle and its sense. This has been illustrated in the following figures.









1.2 *Rotation of the Image* :

Let θ is the skew angle. If sense of this skew is clockwise then we give the image an anticlockwise rotation. If sense of this skew is anticlockwise, then we give it a clockwise rotation. Let R denotes the rotation matrix that will be used to give rotation to the original image to get skew free image.

$$R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Problem in directly using R :

If we take coordinate of each black pixel of the original image and multiply it by R we get the coordinates in the skew free image. Corresponding to this coordinate, in the skew free image we can put the pixel as black. But we miss certain information in this manner. Now, if we try to get the original image from skew free image it may not be possible. So we should use some method which would be compatible in both ways. That is, we get back the original image from the skew free image. The solution to this problem is very simple. Instead of using R we will use inverse of R.

$$R^{-1} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

Now we consider each coordinate of the skew free image and multiply it by R^{-1} to get coordinate point in the original image. If that coordinate point in the original image correponds to black pixel we put black pixel at the coordinate of the skew free image. The resulting image is skew free. It should be noted that the accuracy depends on the subdivisions of the parameter space.

At the end of this step we get skew free image. In this skew free image nearly horizontal and vertical lines will become horizontal and vertical respectively. Validity of this step can be checked from Figure that was obtained after this stage.

2. *Isolating Horizontal and Vertical lines*:

From the first step we get skew free image. In this image we have perfectly horizontal and vertical lines along with the arcs. Now we try to isolate the solid horizontal and vertical lines (leaving the dashed lines and arcs). First horizontal lines are considered. Result of this step will be stored in a two dimensional array. Let us name this array as h_array, lr_array and rl_array. For this we do morphological opening using a long horizontal structuring element (a simple one dimensional array) having all the bits set to 1 (black pixel). Length of this mask will depend on the stroke length of smallest solid line (excluding the dashed and the arc portion). The synthetic data that is used has the stroke length of 40. Now start scanning the image in row major fashion. We open the image twice with the same structuring element except the origin of the structuring element is assumed to be at two different ends. Finally, two opened images are ORed bitwise to obtain image with horizontal lines only.

Now we try to isolate the vertical lines from the skew free image. For this the basic method remains same as for horizontal lines. The basic difference is the structuring element which is a vertical line in this case. So at the end of this step we successfully isolate horizontal and vertical lines from the image. They are stored in h_array and v_array respectively. Figure 5 shows the validity of this step.

3. *Thinning* :

This is one of the most important step in implementing this system. This is required after almost every stage. Before discussing thinning algorithm let us clear the concepts about Medial Axis.

Suppose that a fire line propagates with constant speed from the contour of a connected object towards its inside. Then all those points lying in positions where at least two wave fronts of the fire line meet during the propagation (quench points) constitutes a skeleton called *medial axis* of the object.

Thinning algorithms transform an object to a set of simple digital arcs, which lie roughly along their medial axes. The structure obtained should not be influenced by small contour inflections that may be present on the initial contour. The basic approach is to delete from the object **X** simple border simple border points that have more than one neighbour in **X** and whose deletion does not locally disconnect **X**. Here a connected region is defined as one in which any two points in region can be connected by a curve that lies entirely in the region. In this way, end point of thin arc are not deleted. A simple algorithm that yields connected arcs while being insensitive to contour noise is as follows:

p3	p2	p9
p4	p1	p8
p5	p6	p7

labelling point p1 and its neighbours

Referring to the above figure, let Z0(p1) be the number of zero to nonzero transitions in the ordered set p2,p3,p4,p5,p6,p7,p8,p9,p2. Let NZ(p1) be the number of nonzero neighbours of p1. Then p1 is deleted if

$$2 \leq NZ(p1) \leq 6$$

and $Z0(p1) = 1$

and $p2.p4.p8 = 0$ or $Z0(p2) \neq 1$

and $p2.p4.p6 = 0$ or $Z0(p4) \neq 1$

This procedure is repeated until no further changes occur in the image. Note that at each location such as p1 we end up examining pixels from a 5 x 5 neighbourhood. However first row (column) and last row(column) pixel might create problem. So before thinning any n x n array we must increase its dimension to (n+2) x (n+2). Validity of this algorithm can be checked from Figure 6.

4. *Reconstruction:*

Before this step the h_array and v_array are thinned using the above thinning algorithm. Let us store the thinned horizontal and vertical image in two dimensional array hthin_array and vthin_array respectively. This gives us thinned horizontal and vertical lines. In this step we reconstruct the image with just the thin horizontal and vertical lines. We perform OR operation on hthin_array and vthin_array and store the result in another two dimensional array recon_array At the end of this step we will have image with only the horizontal and vertical lines. The validity of this step can checked from Figure 7.

5. *Dilation:*

In this step we dilate the reconstructed image by a 3 x 3 structuring element and store the result in another two dimensional array dilate_array. So dilate_array contains horizontal and vertical lines (at least 3 pixel thick) which are also present in the skew free image. So at the end of this step we are able to separate horizontal and vertical lines from the skew free image. Here we have assumed that in line drawings the lines are also less than equal to 3 pixel thick. Figure 8 depicts the validity of this step.

6. *Subtraction* :

Once we get horizontal and vertical lines, we try to separate it from the dashed lines and the arc portion. This is needed for further processing of dashed lines and arcs. Once again, the algorithm employed is very simple. Subtract the dilated image from the skew free image. That is put 0 in another array subtract_array if a pixel is 1 in the dilate_array. At the end of this step subtract_array contains dashed lines and arcs. Next step will separate out dashed lines from the arc portion.

7. *Projection:*

Now we have come to a point where we have an image that contains only dashed arcs. In this step we separate dashed lines from the arc portion. First we isolate vertical dashed lines from subtract_array and store it in another array vdash_array. Next we isolate horizontal dashed lines from subtract_array and store it in hdash_array. Then we eliminate dashed lines from subtract_array to get the arc portion.

7.1 *Vertical projection:*

In this step we find the column sum for each column one by one. If this column sum is greater than certain threshold value (for the synthetic data threshold value of 50 was taken), it implies that the 1s in that column belongs to the vertical dashed lines and not the arcs. So we copy that column of subtract_array into vdash_array. So at the end of this step vdash_array will contain the vertical dashed lines only.

7.2 *Horizontal projection:*

In this step we find the row sum for each row one by one. If this row sum is greater than certain threshold value (for the synthetic data threshold value of 50 was taken), it implies that the 1's in that row belongs to horizontal dashed lines and not the arcs. So we will simply copy that row of subtract_array into hdash_array. So at the end of this step hdash_array contains the horizontal dashed lines only.

Next step is to subtract vdash_array and hdash_array from subtract_array to get the arc portions only. For this 1 is put in a new array arc_array if there is 1 in subtract_array and 0 in vdot_array as well as hdot_array and a 0 is put in arc_array if it 0 in subtract array or 1 in subtract_array as well as 1 in vdot_array or hdot_array. After this subtraction arc_array contains points that belong to arc only.

8. *Vectorization* :

8.1 *Vectorizing horizontal and vertical lines* :

In this step we represent the horizontal and vertical lines (solid as well as dashed) by their end points. This process of representing a straight line by its end point is called vectorization. For implementing this step first we vectorize horizontal lines. For this, solid horizontal lines and dashed horizontal lines are thinned (note that they were stored earlier in h_array and hdash_array) separately. Then the thinned solid horizontal lines and thinned dashed horizontal lines are combined in a single array. So all the lines are one pixel thick. Now we use this array for vectorization of horizontal lines. This array is now scanned in row major fashion. When a 1 is encountered for the first time, it means that it is the starting point of the horizontal line. Go on scanning in that row until a 0 is encountered to get the other end point of this line. Repeat the above step whenever next 1 is encountered. Repeat the above step for all the rows.

Next, we vectorize the vertical lines. For this, solid vertical lines and dashed vertical lines are thinned (note that they were stored earlier in v_array and vdash_array) separately. Then the thinned solid vertical lines and thinned dashed vertical lines are combined in a single array. So all the lines are one pixel thick. Now we use this array for the vectorization of vertical lines. This array is now scanned in a column major fashion. When a 1 is encountered for the first time, it means that it is the starting point of the vertical line. Go on scanning in that column until a 0 is encountered to get the other end point of this line. Repeat the above step whenever next 1 is encountered. Repeat this step for all the columns.

In this way we find the end points of the horizontal and vertical lines and store it in a file . This file can be used later on for regenerating the horizontal and vertical lines of the original line drawing.

8.2 *Vectorization of arcs*:

An arc is uniquely defined by three points. If two end points and a third point (other than end points) of an arc are known, then we can always construct a unique arc between the two end points. Now the problem is to find whether two points are on the same arc. The solution is given by component labelling. We label the connected components. This gives us the number of components which should be equal to number of arcs. The points that lie on the same component are given the same label by this method.

Labelling of connected components :

Connectivity between pixels is an important concept used in establishing boundaries of objects components of regions in an image. To establish whether two pixels are connected, it must be determined if they are adjacent in some sense (say, if they are 4-neighbours) and if their gray level satisfy a specified criterion of similarity (say, if they are equal). For instance, in a binary image with values 0 and 1, two pixels may be 4-neighbours, but they are not said to be connected unless they have the same value.

Imagine scanning an image pixel by pixel, from left to right and from top to bottom. We are interested in 8-connected component. Let p denote the pixel at any step in the scanning process and let r and t denote the upper and left hand neighbours of p, respectively. Let q and s denote the two upper diagonal neighbours of p. The nature of scanning sequence ensures that these neighbours have already been processed by the time the procedure gets to p. We maintain a list of labels. Whenever a new label is used we add that label to this list. We also maintain an equivalent table. This is a simple array of list. The array index gives the component name and list gives the list of labels which are equivalent to this label. Whenever we find that two labels are equivalent we put the two labels in each others' list. This table helps in resolving conflicts and minimising the number of components. Steps are as follows:

1. If p is 0 , move on to the next scanning position. If p is 1 and all four neighbours are 0, assign a new label to p. Add this new label to the list of labels. If only one of the neighbours is 1, assign its label to p. If two or more neighbours are 1, assign one of the labels to p and make a note of the appropriate equivalences. For simplicity we assign the minimum label value to p.
2. After completing the scan of the image, we sort (in increasing order) the list of equivalent labels. The equivalent labels are now assigned a unique label (minimum value among the equivalent labels).
3. We do a second scan through the image, replacing each label by the label assigned to its equivalence class. Now we find the number of distinct components from the final list of labels

We store the points that lie on the same component. This is done for each component. Now, we know the points that lie on the same component. The bounding rectangle is obtained by simple sorting of these points along two coordinate axes. At the end of this operation we know the number of connected components, the points that lie on the same connected component and the bounding rectangle. Bounding rectangle is specified by the coo[illegible]dinates of the top-left corner of the rectangle, its height and width. The next step is to find the two end points and a third distinct point on the arc.

In order to vectorize the arc we need two end points and a distinct third point on the arc. For each component we know the bounding rectangle and image array. In order to find the end points of the arc, we make certain assumptions. For most cases this assumptions are valid. While isolating horizontal and vertical lines from the original image, certain points (at tangents to the circles parallel to the two axes) of the arcs are also isolated as part of horizontal and vertical lines. So all the arcs is of length less than a quarter of a circle. Under this assumption, one end point must lie on the first row and the other on the last row of the bounding rectangle. Searching for a 1 in these two rows give the two end points. For the third point, we find the middle row of the bounding rectangle and look for a 1 in this row. This gives the third point of the arc. We store these points in a data file. This file can later be used for regeneration of the original image. However, this assumption may not be valid for every component. We will go through the special cases at end of this section.

9. *Reconstruction* :

The first and foremost objective of developing this system is to vectorize the given line drawing and store the points representing the vectors in a simple data file. But the efficiency of vectorization depends on the accuracy with which the line drawing can be reconstructed from the data file. First we consider the reconstruction of horizontal and vertical lines. This is quite easy in comparison to reconstruction of arcs. Since end points of the horizontal and vertical lines are known (from data file), we make a horizontal or vertical line by putting 1 in the positions starting from one end point and ending at the other end point. In this way we can reconstruct the horizontal and vertical lines. The next step is to reconstruct arcs from three points on the arc. For reconstruction of arcs we use

Lagrange interpolation. Since three points of the arc are known, we use Lagrange polynomial of second degree.

Here we use row (say x) as the independent variable and column (say y) as the dependent variable. We vary x from one end point to the other end point and find out the value of y at the intermediate points. Let the three known points be (x_1, y_1), (x_2, y_2) and (x_3, y_3). Then Lagrange polynomial of second degree is :

$$P(x) = l_1(x)\ y_1 + l_2(x)\ y_2 + l_3(x)\ y_3$$

where,

$$l_1(x) = (x - x_2)(x - x_3) / (x_1 - x_2)(x_1 - x_3)$$

$$l_2(x) = (x - x_1)(x - x_3) / (x_2 - x_1)(x_2 - x_3)$$

$$l_3(x) = (x - x_1)(x - x_2) / (x_3 - x_1)(x_3 - x_2)$$

EXPERIMENTAL RESULTS

The algorithm presented in the preceding sections is incorporated into a system for vectorizing line drawings, currently running on LINUX operating system. A tool was developed for X-windows environment for displaying the image at the end of each operation. This would also aid in the analysis of the image. The input to this system consists of line drawings digitised at a resolution of 200 dpi using a scanner. Figure shows a binary image obtained from the scanner. The text and dimension lines are erased from the original diagram using xpaint application program. Figure 9 shows the result of the skew correction operation described in Section 1, which rectifies the possible skew error during scanning the original image. Figure 10 shows the result of thinning operations on the image. This image is obtained by combining two images, one obtained by thinning horizontal lines and the other obtained by thinning vertical lines of Figure [illegible]. Figure 11 shows the result after vectorization, wherein all the horizontal and vertical line segments are represented by their end points and the arcs by their end points and a point in between the two end points.

One of the major concerns in vectorizing line drawings is the accuracy of the

resulting representation. The system was designed not only to get a compact representation of the image of a line drawing, but an accurate one as well. Using this system we successfully vectorized A4 size line drawings. Figures [illegible] and 13 show results after vectorization. From these results one can see that the representation of the line drawing after vectorization is in close agreement with the original. The system does however fail in some situations where circles and arcs are intersected by other line structures. In such cases entities are fitted by more than one arc

Pixel based thinning algorithms introduce defects in the arcs. Also the portions of arcs, which are small and do not appear to exhibit significant curvature, are approximated with straight lines.

CONCLUSIONS

This work discusses an algorithm required as a first step in developing an intelligent system for interpreting line drawings. Our implementation provides the following features:

- Correction of possible skew error during scanning.
- Extracting thinned horizontal and vertical lines from the image.
- Extracting thinned horizontal dashed lines and vertical dashed lines from the image.
- Extracting thinned arcs from the image.
- Vectorizing line structures obtained from the image. The vectorization procedure not only identifies straight line segments but recognizes arcs as well. This achieves a compact coding of information present in the line drawing, since all the straight line segments are represented by two points, while arcs are represented by three points only. This also provides a better and more accurate representation of the line drawings.
- Reconstruction of horizontal and vertical lines as well as arcs from the vector points.

Scope for Improvement:

This vectorization process may be extended in the case of inclined lines. Possible modifications may be necessary during extraction of the connected compone s to identify and fit conic sections present in the original line drawing.

REFERENCES

1. Haralick R.M. and Queeney D., Understanding engineering drawings, *Comput. Graphics Image Process.* **20**, 1982, 244-258.

2. Sato T. and Tojo A., Recognition and understanding of hand drawn diagrams, *Proceedings, Sixth International Conference on Pattern Recognition, IEEE, 1982*, 674-677

3. Nagasamy V., Langrana N. and Peskin R., Progress Toward the Development of an Intelligent Engineering Drawing Interpretation System, *Technical Report CAIP-TR-037*, Rutgers University, May26, 1987.

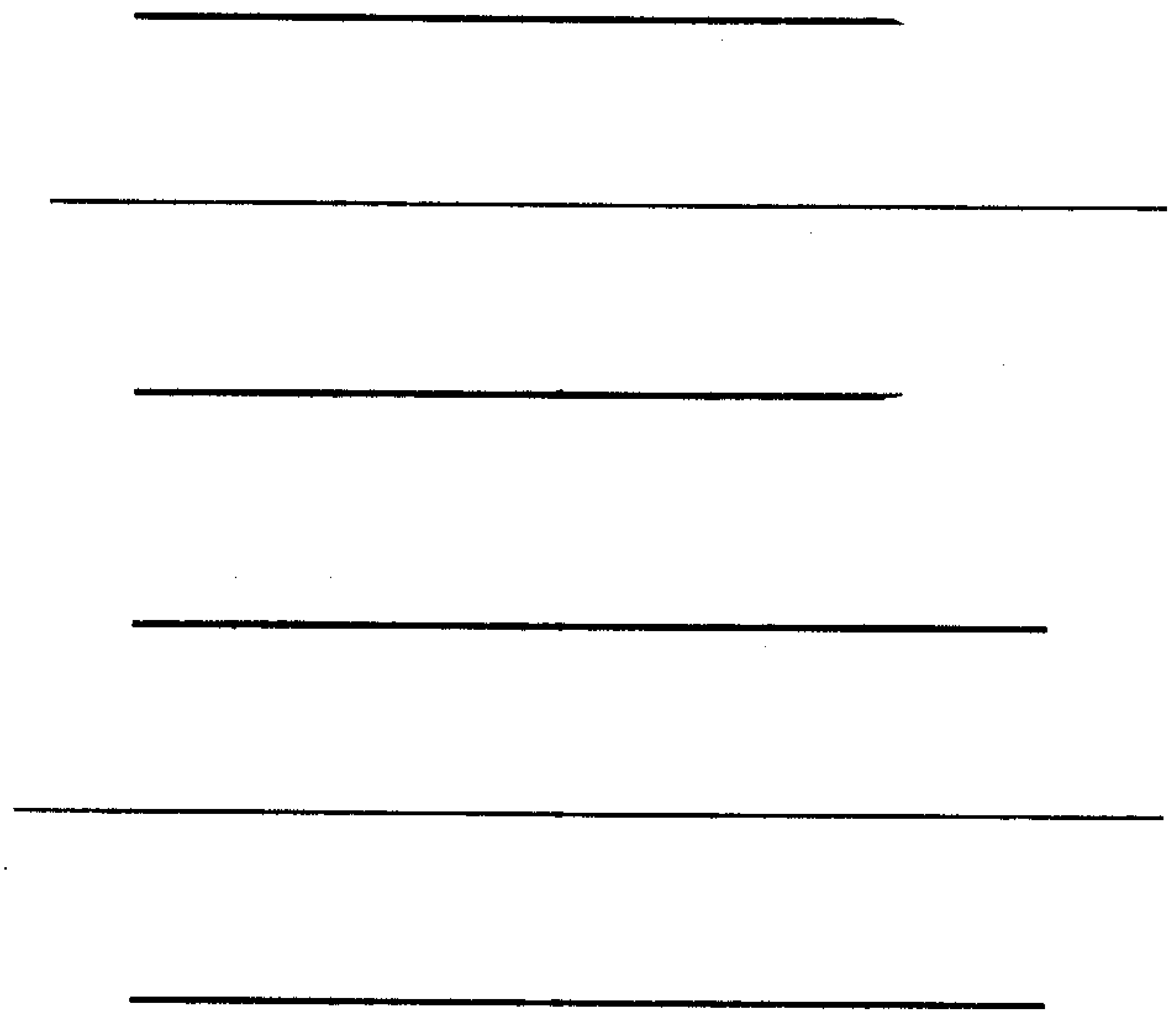


FIG - 5

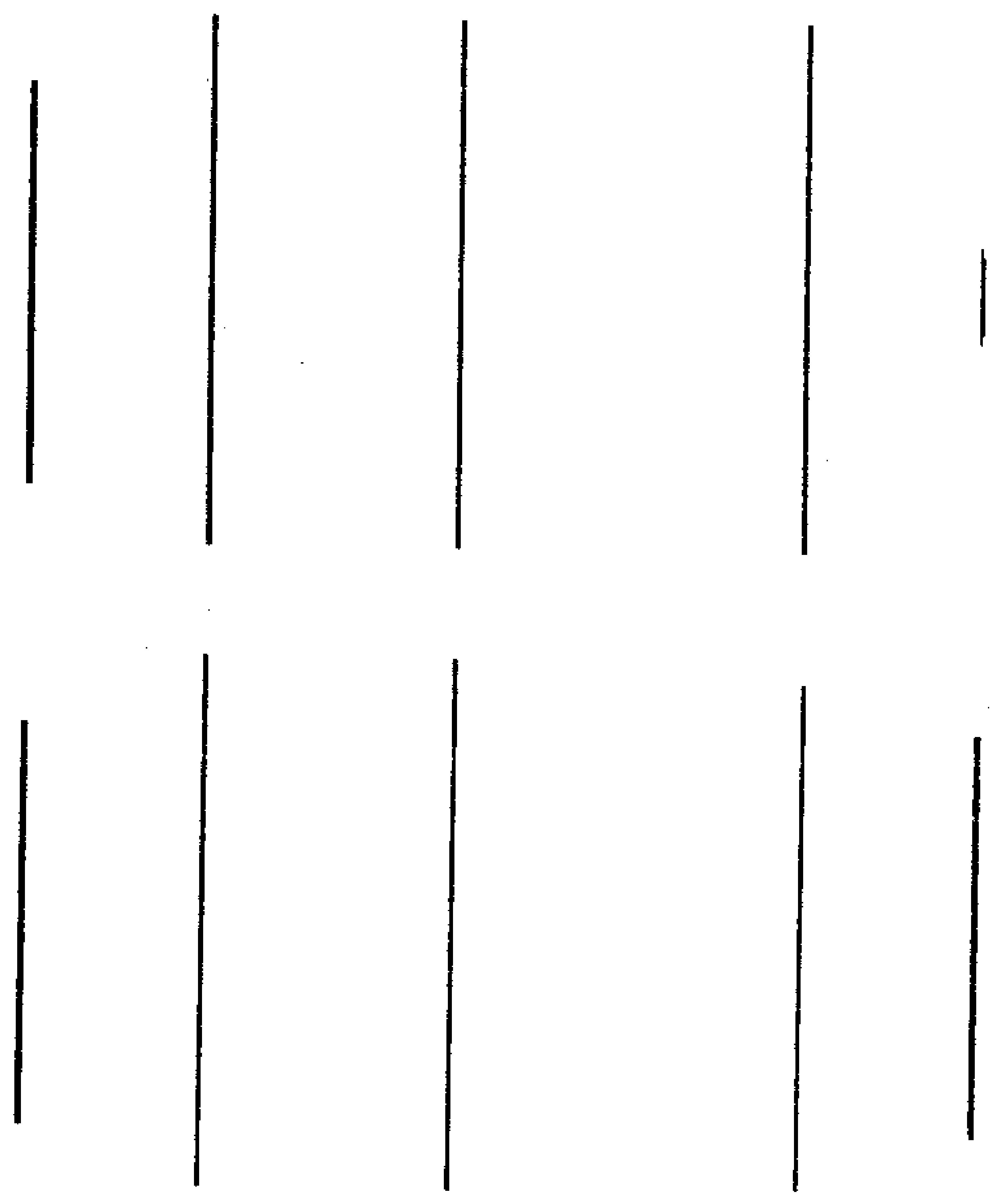


FIG - 5

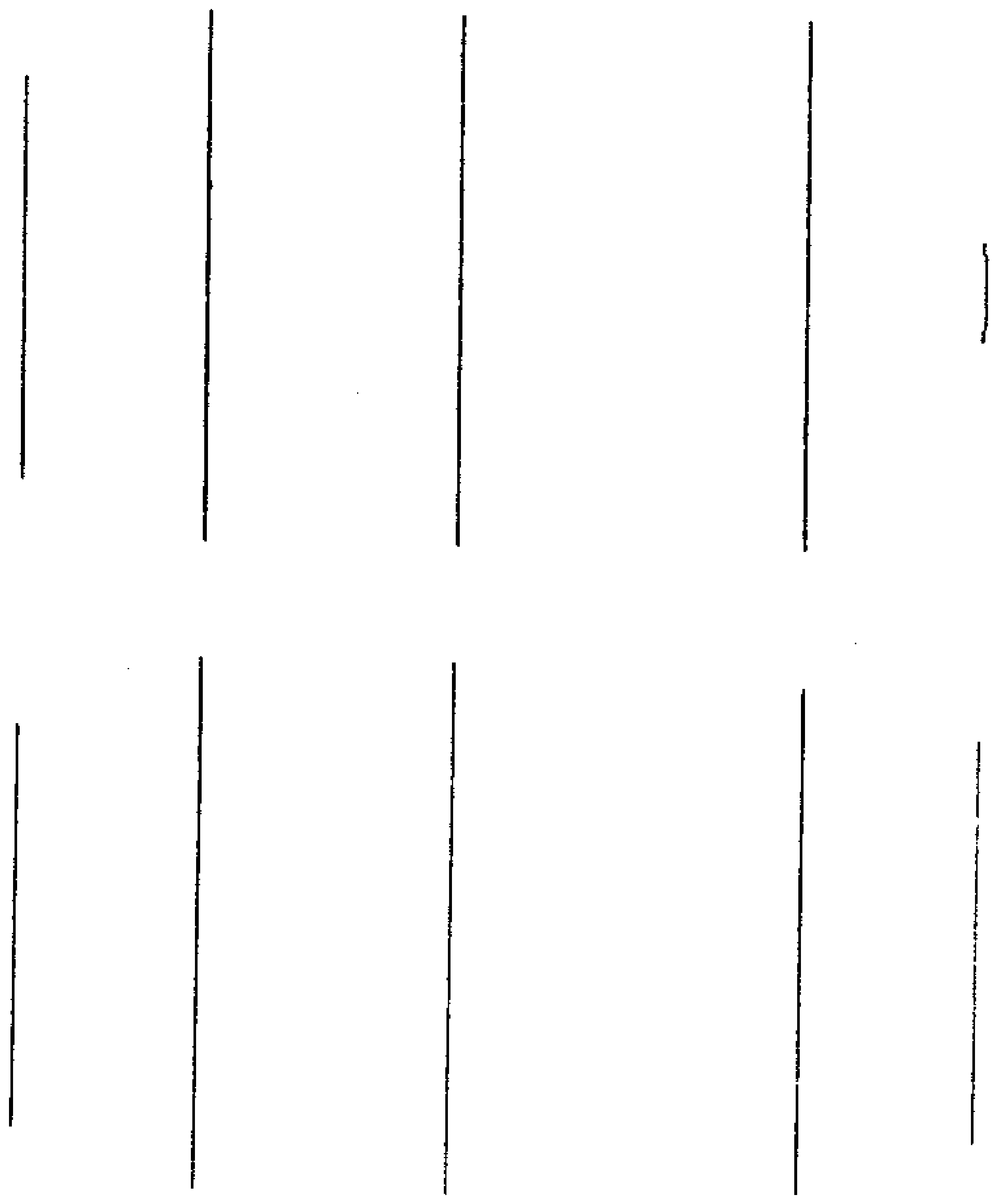


FIG - 6

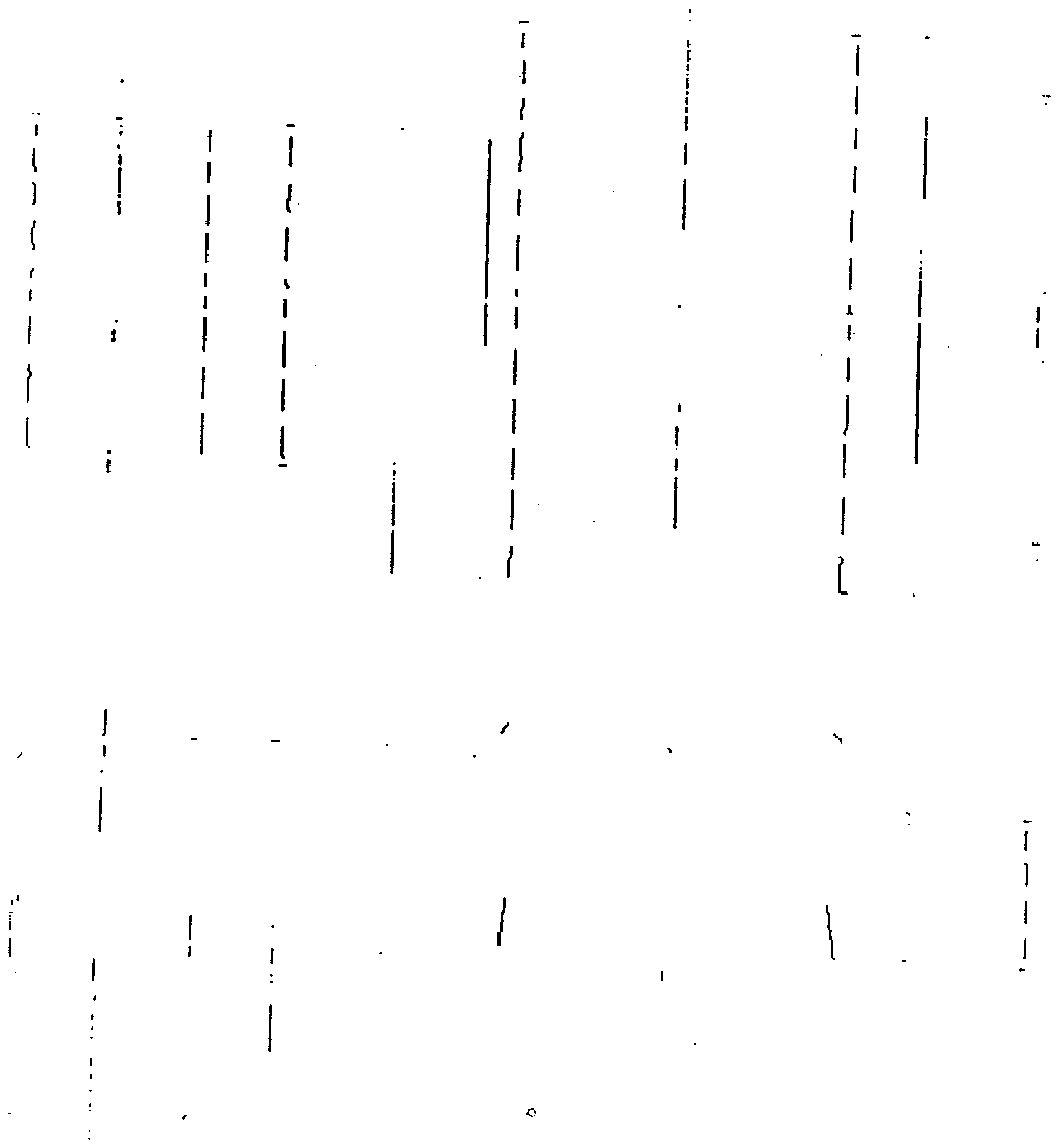


FIG - 6

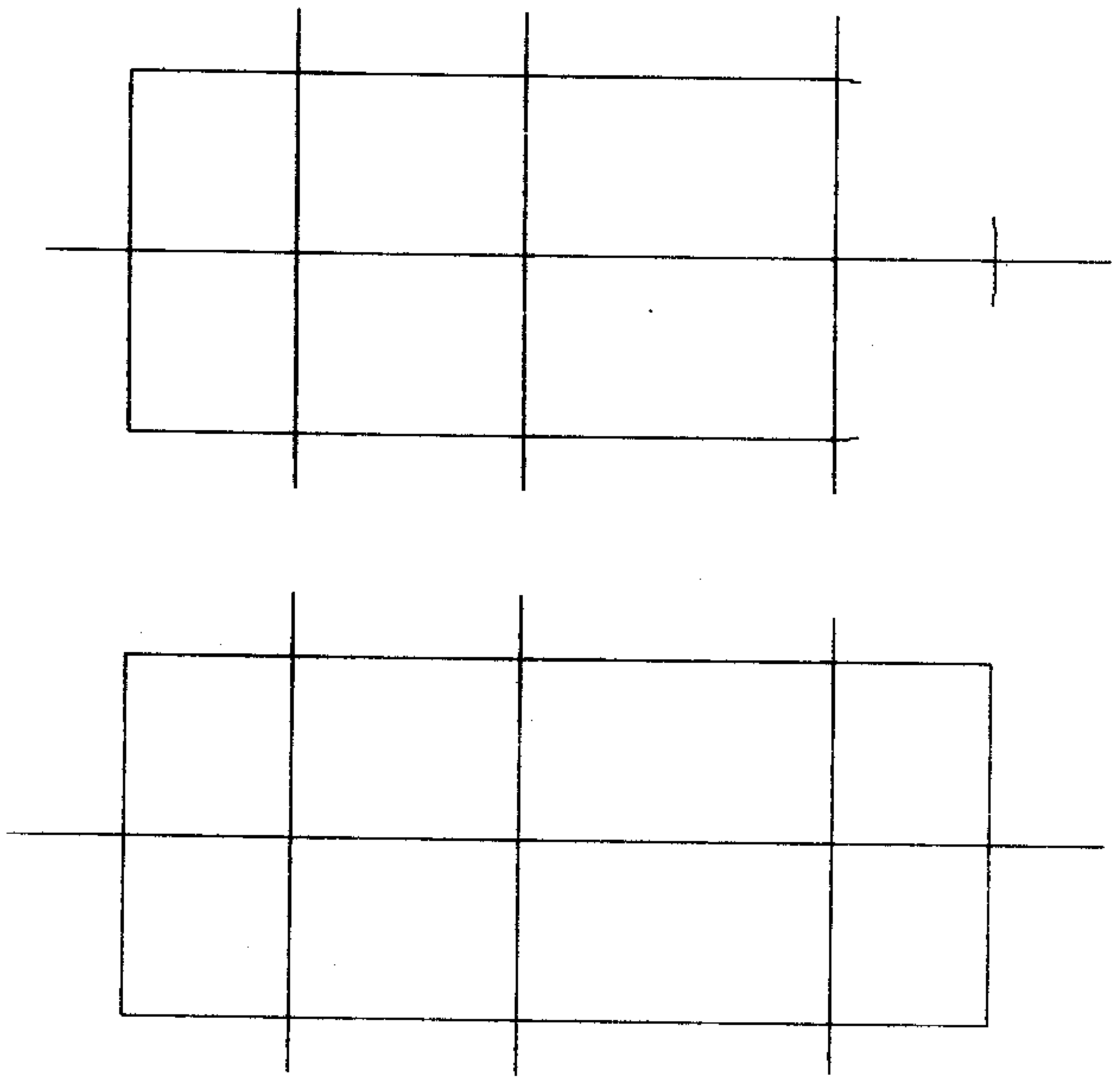


FIG - 7

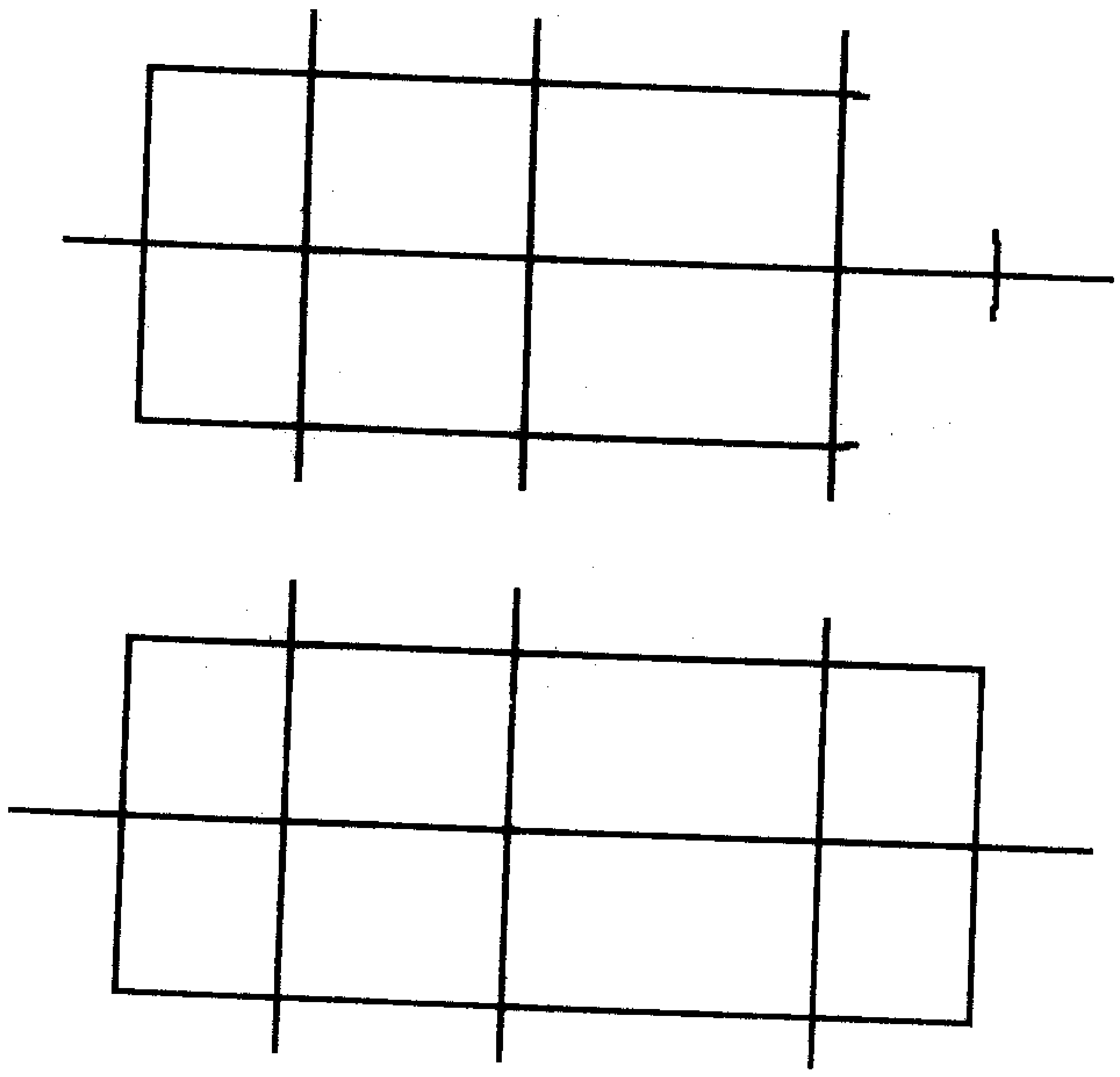


FIG - 8

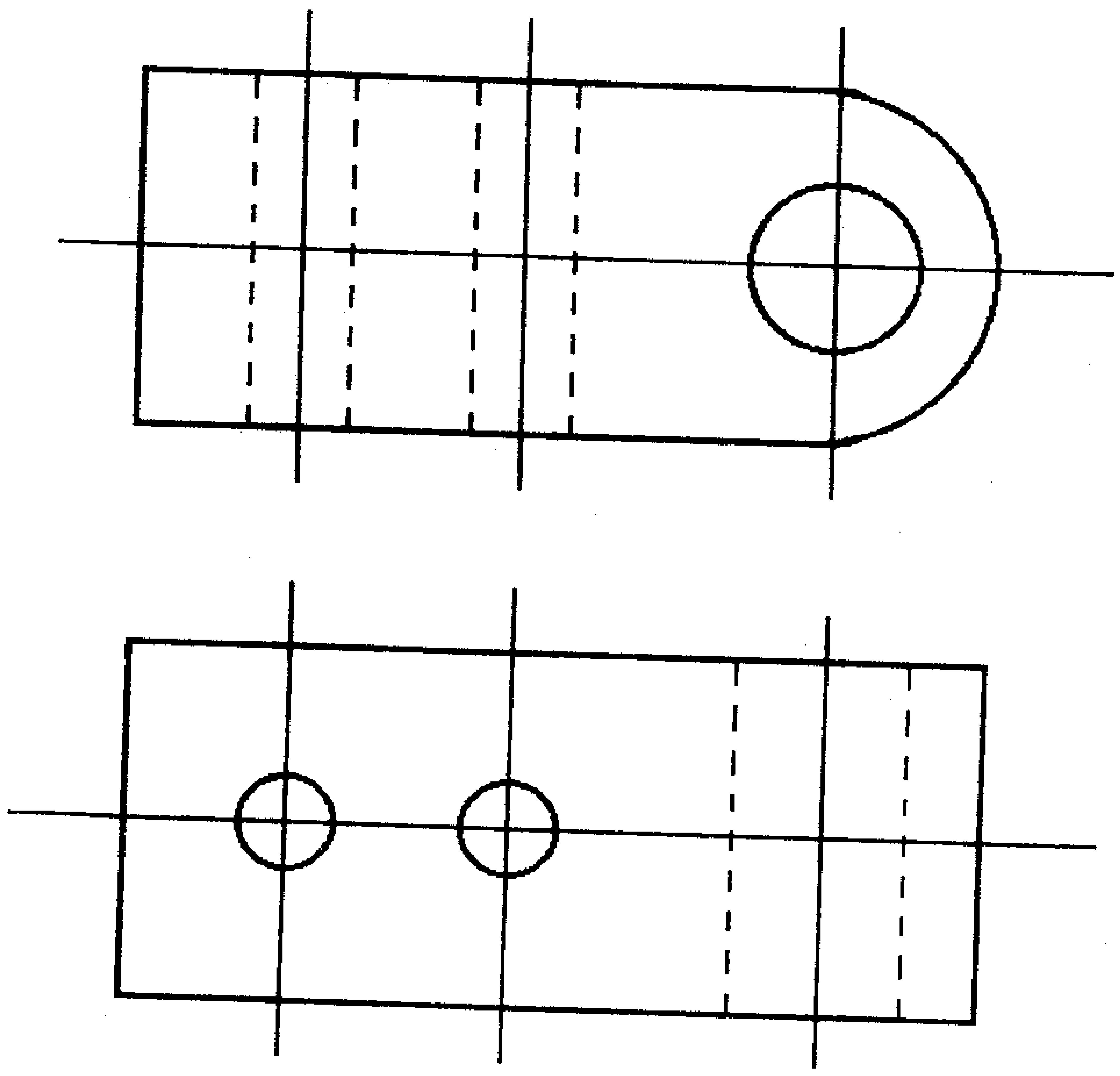


Fig - 9

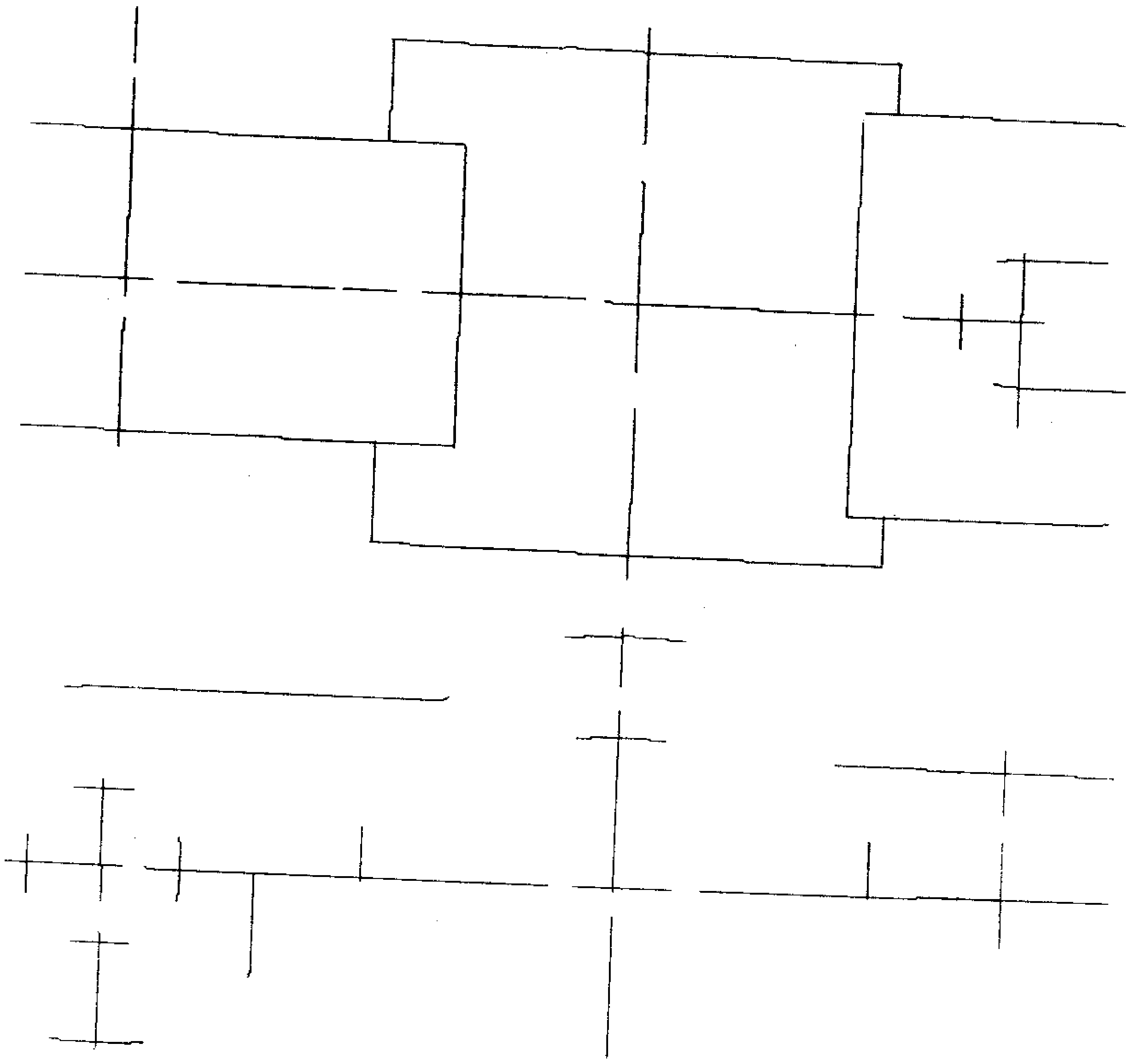


FIG - 10

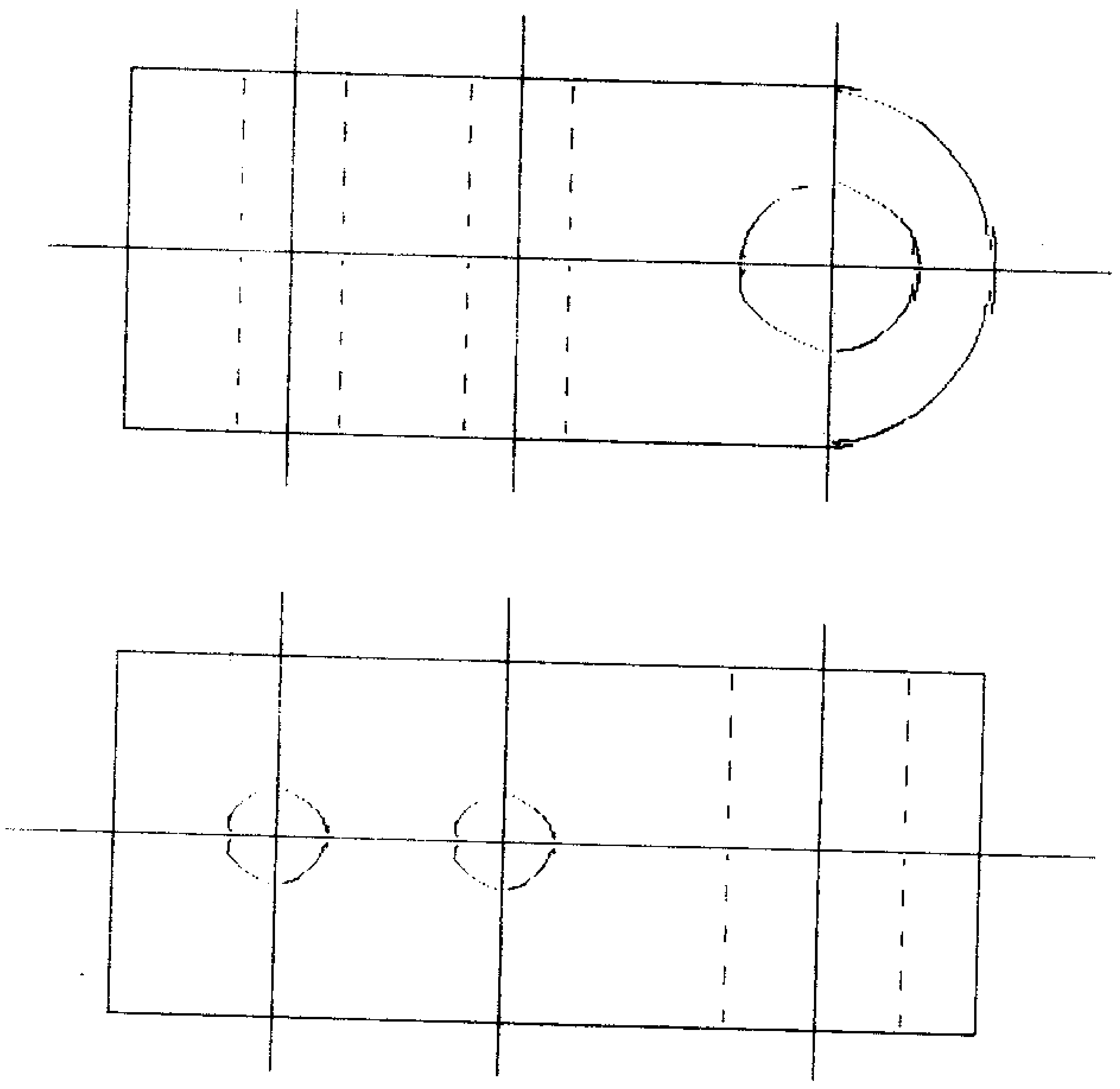


FIG - 11

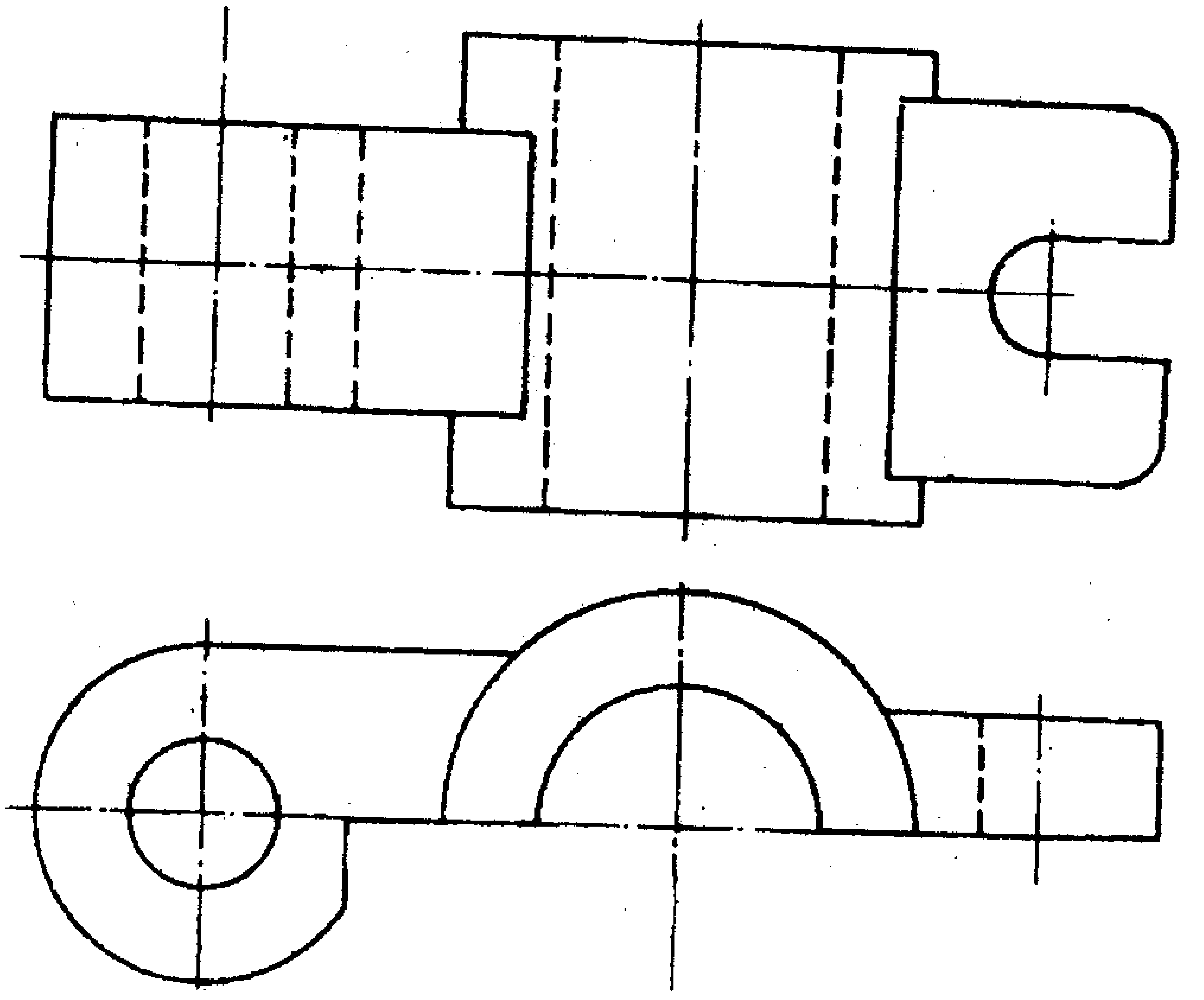


FIG - 12

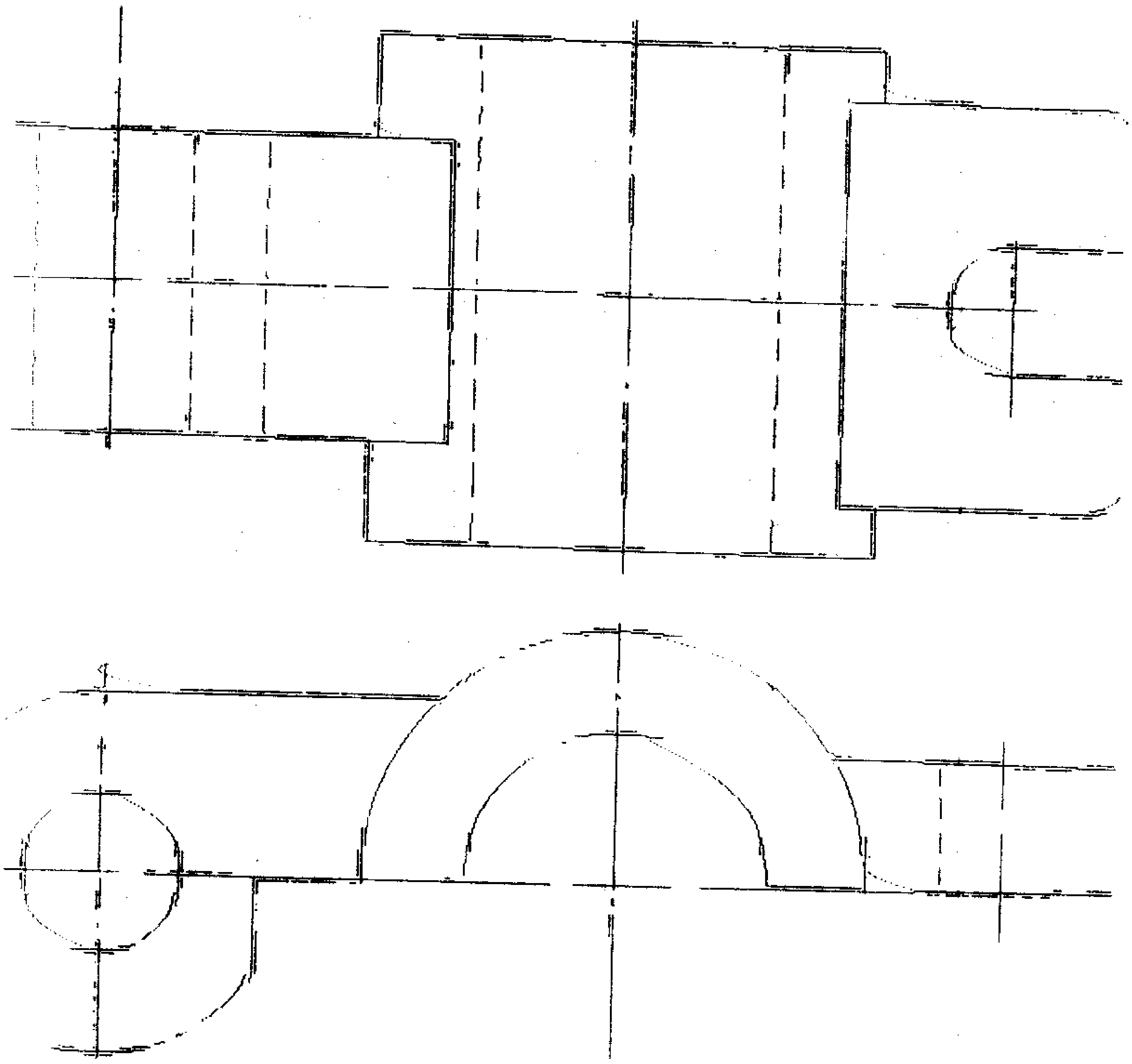


FIG-13