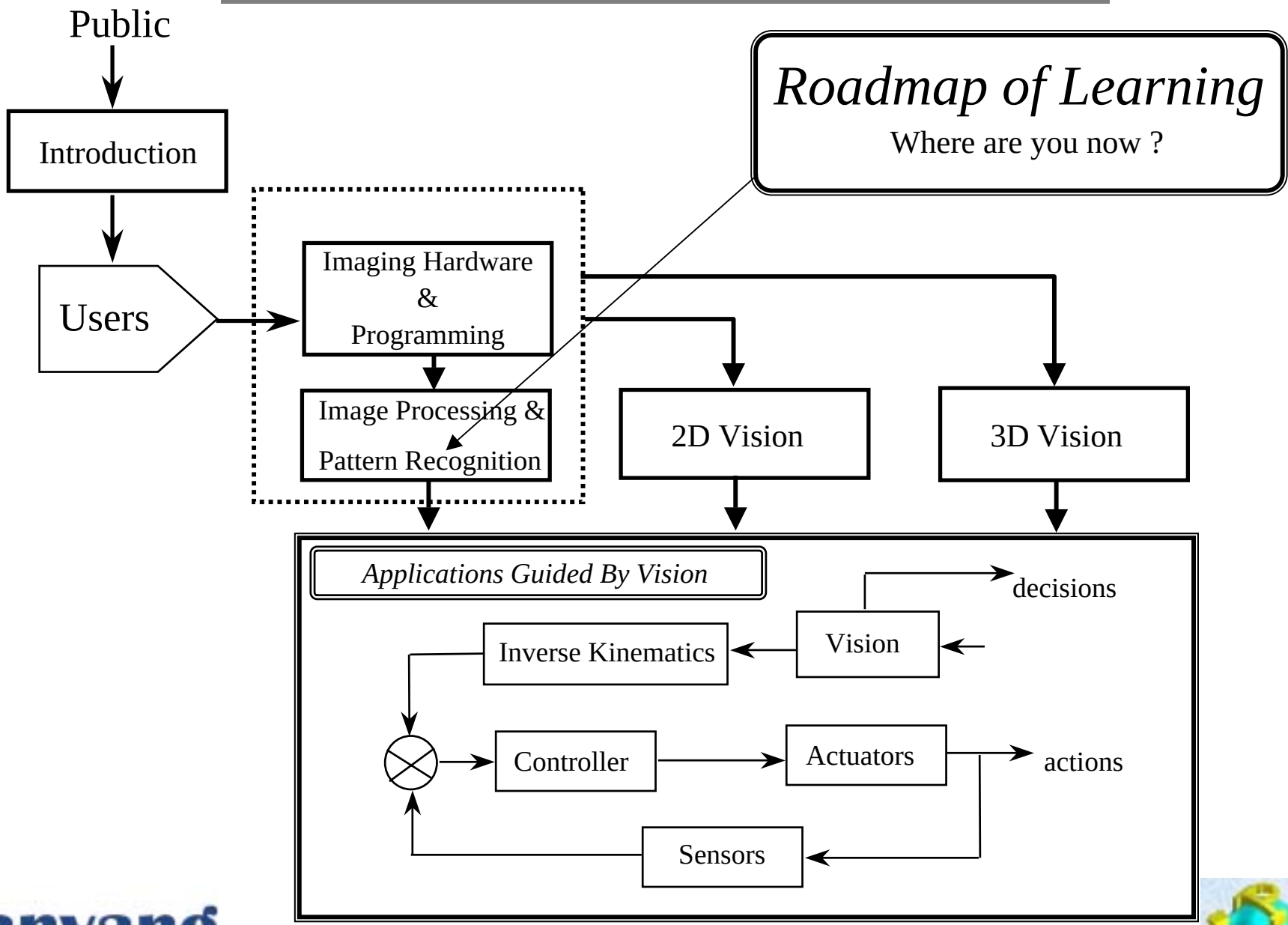
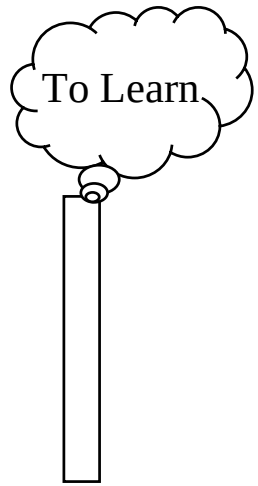


CONTENT



Chapter 5: Fundamentals of Pattern Recognition

5.1 Concept of Pattern Recognition

5.2 Pattern Detection

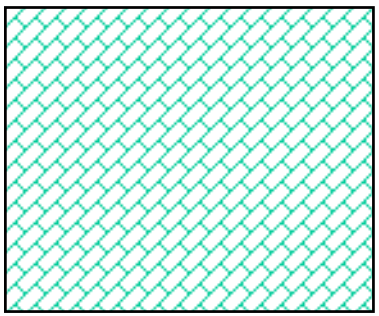
5.3 Pattern Characterisation

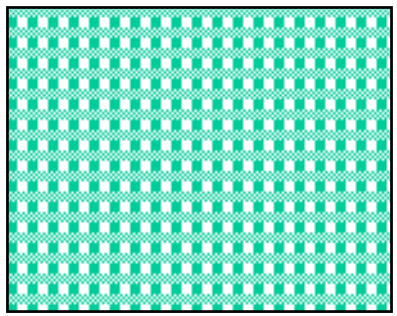
5.4 Pattern Classification

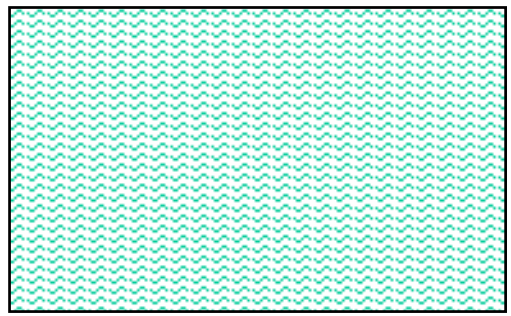
What is the literal meaning of the word “pattern” ?

ANSWER:

Pattern refers to “regularly repeated arrangement of shapes, colors, or lines on a surface”.



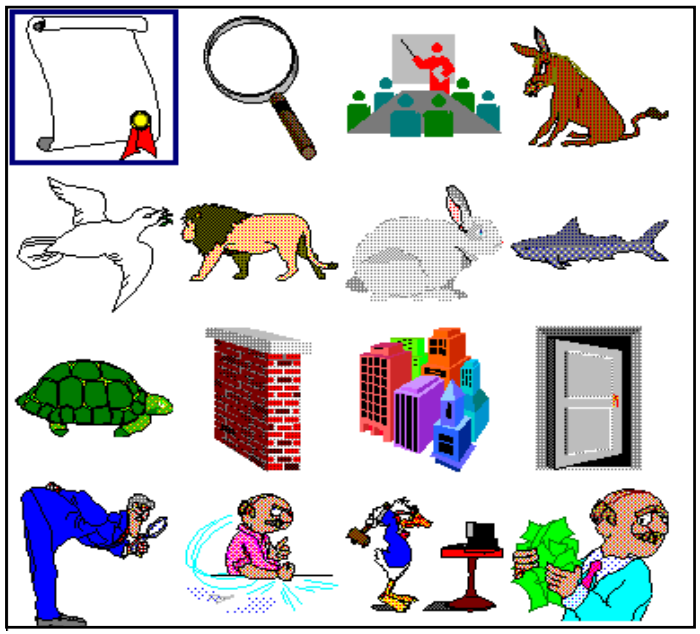




What do we mean by "(2D image) pattern" in machine vision ?

ANSWER:

Image pattern refers to "meaningful appearance of shapes, colors or primitive pixels on an image plane.



What do we mean by “pattern recognition” ?

ANSWER:

It refers to the process of detecting, characterizing and classifying image patterns. In this way, one can recognize an object through its image.

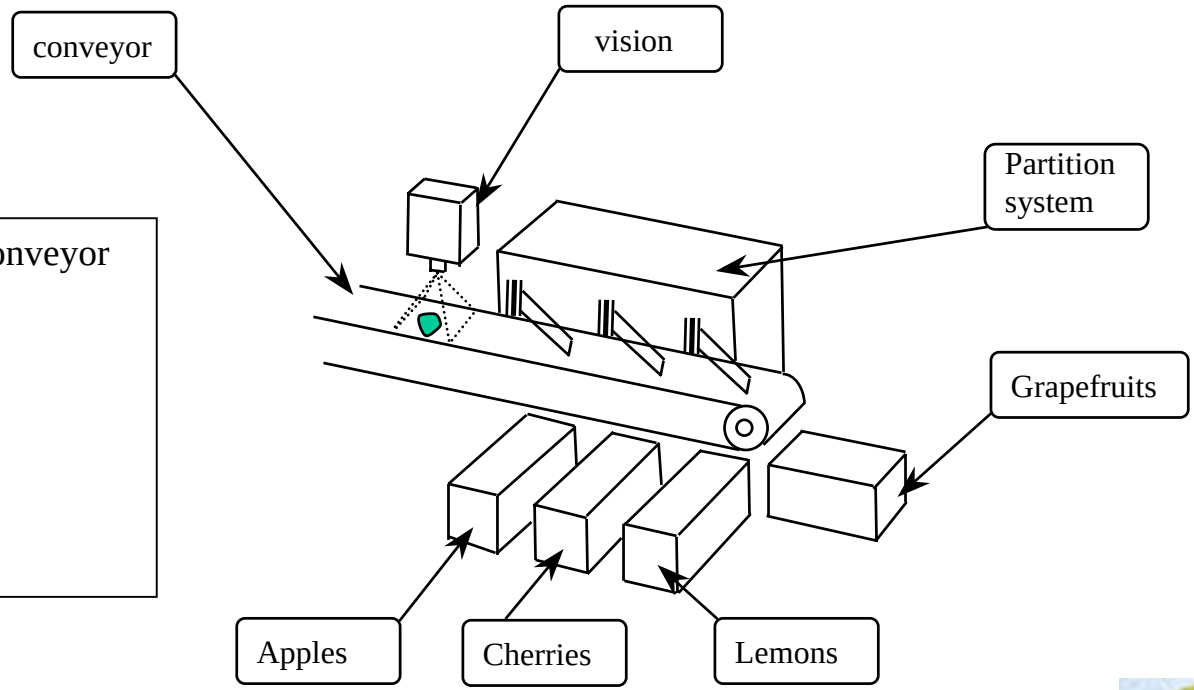
Why to do “pattern recognition” ?

ANSWER:

1. The process of doing Pattern Recognition may be required by an application:

To sort the fruits placed on a conveyor into four categories:

1. Apple
2. Cherry
3. Lemon
4. grapefruit



2. Image processing is necessary but may not be sufficient to meet the requirement of an application:

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	100	100	100	100	255	255	255	80	80	255	255
2	255	100	100	100	100	255	255	80	80	80	80	255
3	255	100	100	100	100	255	255	80	80	80	80	255
4	255	100	100	100	100	255	255	80	80	80	80	255
5	255	100	100	100	100	255	255	255	80	80	255	255
6	255	100	100	100	100	255	255	255	255	255	255	255
7	255	100	100	100	100	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

Thresholding

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	0	0	0	0	255	255	255	0	0	255	255
2	255	0	0	0	0	255	255	0	0	0	0	255
3	255	0	0	0	0	255	255	0	0	0	0	255
4	255	0	0	0	0	255	255	0	0	0	0	255
5	255	0	0	0	0	255	255	255	0	0	255	255
6	255	0	0	0	0	255	255	255	255	255	255	255
7	255	0	0	0	0	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

Edge Detection

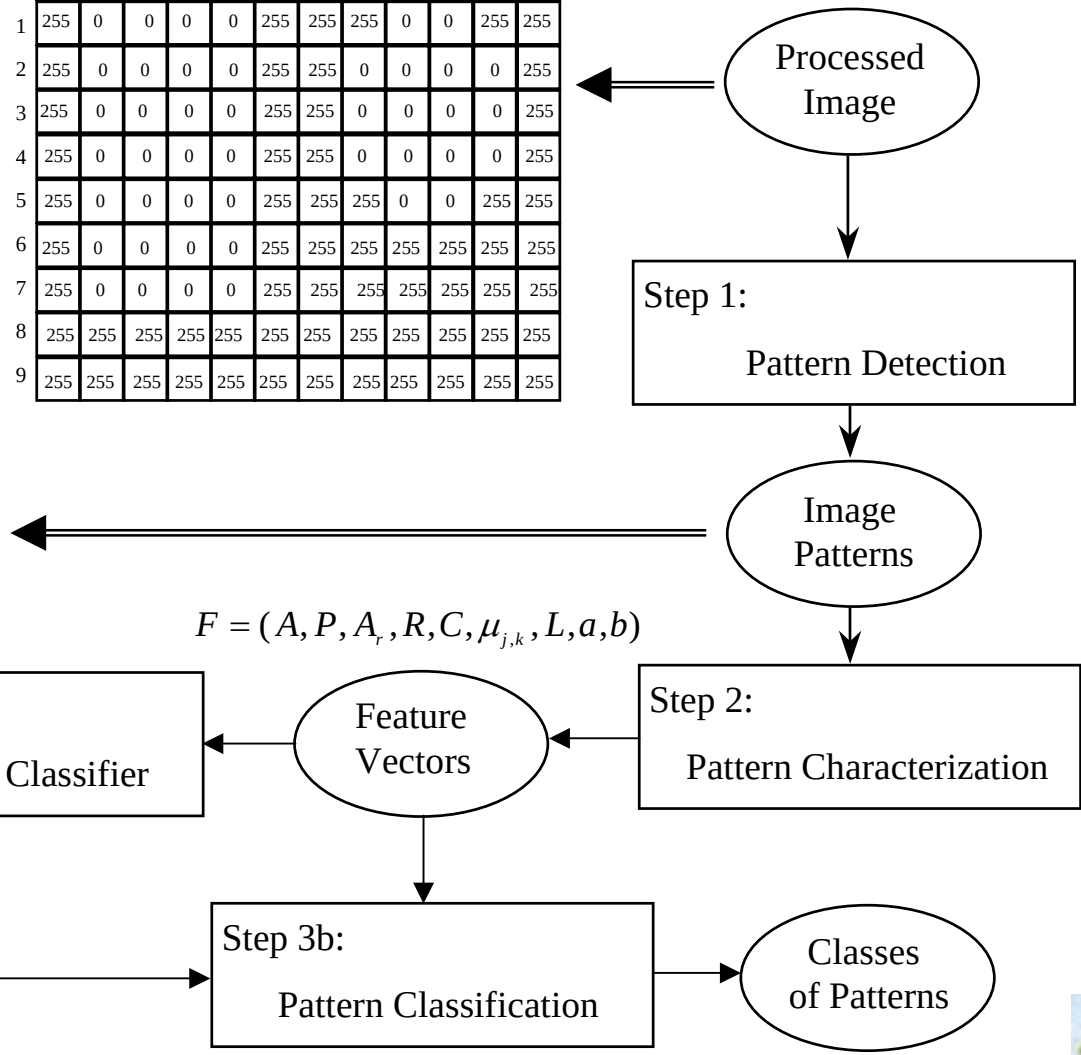
	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	0	0	0	0	255	255	255	0	0	255	255
2	255	0	255	255	0	255	255	0	255	255	0	255
3	255	0	255	255	0	255	255	0	255	255	0	255
4	255	0	255	255	0	255	255	0	255	255	0	255
5	255	0	255	255	0	255	255	255	0	0	255	255
6	255	0	255	255	0	255	255	255	255	255	255	255
7	255	0	0	0	0	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

What are the general steps of doing pattern recognition ?

ANSWER:

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	1	1	1	1	255	255	255	2	2	255	255
2	255	1	1	1	1	255	255	2	2	2	2	255
3	255	1	1	1	1	255	255	2	2	2	2	255
4	255	1	1	1	1	255	255	2	2	2	2	255
5	255	1	1	1	1	255	255	255	2	2	255	255
6	255	1	1	1	1	255	255	255	255	255	255	255
7	255	1	1	1	1	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	0	0	0	0	255	255	255	0	0	255	255
2	255	0	0	0	0	255	255	0	0	0	0	255
3	255	0	0	0	0	255	255	0	0	0	0	255
4	255	0	0	0	0	255	255	0	0	0	0	255
5	255	0	0	0	0	255	255	255	0	0	255	255
6	255	0	0	0	0	255	255	255	255	255	255	255
7	255	0	0	0	0	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255



How to detect the image pattern of an object of interest ?

ANSWER:

Thresholding

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	0	0	0	0	255	255	255	0	0	255	255
2	255	0	0	0	0	255	255	0	0	0	0	255
3	255	0	0	0	0	255	255	0	0	0	0	255
4	255	0	0	0	0	255	255	0	0	0	0	255
5	255	0	0	0	0	255	255	255	0	0	255	255
6	255	0	0	0	0	255	255	255	255	255	255	255
7	255	0	0	0	0	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

Technique 1:

Grouping by connectivity

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	100	100	100	100	255	255	255	80	80	255	255
2	255	100	100	100	100	255	255	80	80	80	80	255
3	255	100	100	100	100	255	255	80	80	80	80	255
4	255	100	100	100	100	255	255	80	80	80	80	255
5	255	100	100	100	100	255	255	255	80	80	255	255
6	255	100	100	100	100	255	255	255	255	255	255	255
7	255	100	100	100	100	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

Technique 2:

Edge Linking or Tracing

Edge Detection

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	0	0	0	0	255	255	255	0	0	255	255
2	255	0	255	255	0	255	255	0	255	255	0	255
3	255	0	255	255	0	255	255	0	255	255	0	255
4	255	0	255	255	0	255	255	0	255	255	0	255
5	255	0	255	255	0	255	255	255	0	0	255	255
6	255	0	255	255	0	255	255	255	255	255	255	255
7	255	0	0	0	0	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

Grouping by Connectivity

Principle:

If the primitive pixels at the locations (i,j) and (s,t) are neighbors, these two pixels belong to a same pattern.

Operation:

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	0	0	0	0	255	255	255	0	0	255	255
2	255	0	0	0	0	255	255	0	0	0	0	255
3	255	0	0	0	0	255	255	0	0	0	0	255
4	255	0	0	0	0	255	255	0	0	0	0	255
5	255	0	0	0	0	255	255	255	0	0	255	255
6	255	0	0	0	0	255	255	255	255	255	255	255
7	255	0	0	0	0	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

Given a location *(i,j)* where $Value(i, j) = 0$, repeatedly do:

$$Label(i, j) = n$$

$$Neighbors(i, j) = \{(i-1, j-1), (i-1, j), (i-1, j+1), (i, j-1), (i, j+1), (i+1, j-1), (i+1, j), (i+1, j+1)\}$$

$\forall (s,t) \in Neighbors(i, j)$, if $Value(s,t) = 0$, do:

$$\begin{cases} Label(s,t) = n \\ i = s \\ j = t \end{cases}$$ and repeat the main loop.

Algorithm:

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	0	0	0	0	255	255	255	0	0	255	255
2	255	0	0	0	0	255	255	0	0	0	0	255
3	255	0	0	0	0	255	255	0	0	0	0	255
4	255	0	0	0	0	255	255	0	0	0	0	255
5	255	0	0	0	0	255	255	255	0	0	255	255
6	255	0	0	0	0	255	255	255	255	255	255	255
7	255	0	0	0	0	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	255	255	255	255	255	255	255	255	255	255	255
2	255	255	255	255	255	255	255	255	255	255	255	255
3	255	255	255	255	255	255	255	255	255	255	255	255
4	255	255	255	255	255	255	255	255	255	255	255	255
5	255	255	255	255	255	255	255	255	255	255	255	255
6	255	255	255	255	255	255	255	255	255	255	255	255
7	255	255	255	255	255	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

Step 1: Initialize the label map and set the label variable to 1 (n=1)

Step 2: Scan the image row by row and column by column.

Step 3: If encounter a unlabeled primitive pixel, define it as the current pixel. NOTE: If the pixel value at the same location on the label map is 255, the primitive is said to be not yet labeled.

Step 4: Assign the label “n” to the pixel value at the same location on the label map. Now, this pixel is labeled.

Step 5: Find all the neighbors of the current pixel. If no neighbor, go to Step 7.

Step 6: For each neighbor that is a primitive pixel and is not labeled, define it as the current pixel.

Repeat Step 4, Step 5 and Step 6. (Dynamic Programming)

Step 7: Increment the label (n=n+1) and continue from Step 2.

Sample Program:

```
/* assume that the pixel values of all the primitive pixels are "0" */

unsigned char     image[512*512] ;
unsigned char     labelmap[512*512] ;

static void  AssignLabelToPixel(int r, int c, int n)
{
      if (labelmap[r*512+c] == 255) labelmap[r*512+c] = n ;     /* if a unlabeled pixel */
      else                return ;

      /* find its neighbors, and for each neighbor, assign the same label to it */
      if (image[(r-1)*512+c-1] == 0)   AssignLabelToPixel(r-1, c-1, n) ;
      if (image[(r-1)*512+c] == 0)     AssignLabelToPixel(r-1, c, n) ;
      if (image[(r-1)*512+c+1) == 0)  AssignLabelToPixel(r-1, c+1, n) ;

      if (image[r*512+c-1] == 0)   AssignLabelToPixel(r, c-1, n) ;
      if (image[r*512+c+1] == 0)   AssignLabelToPixel(r, c+1, n) ;

      if (image[(r+1)*512+c-1] == 0) AssignLabelToPixel(r+1, c-1, n) ;
      if (image[(r+1)*512+c] == 0)   AssignLabelToPixel(r+1, c, n) ;
      if (image[(r+1)*512+c+1] == 0)  AssignLabelToPixel(r+1, c+1, n) ;
}

main(int argc, char **argv)
{
      int   r, c, n ;

      n = 1 ;     memset(labelmap, 255, 512*512) ;   /* clear the content of labelmap */

      for (r = 1 ; r < 511 ; r++)
            for (c = 1 ; c < 511 ; c++)
                  if (image[r*512+c] == 0)
                  {
                        AssignLabelToPixel(r, c, n) ;     n = n + 1 ;
                  }
}
```


Result:

Processed Image As Input

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	0	0	0	0	255	255	255	0	0	255	255
2	255	0	0	0	0	255	255	0	0	0	0	255
3	255	0	0	0	0	255	255	0	0	0	0	255
4	255	0	0	0	0	255	255	0	0	0	0	255
5	255	0	0	0	0	255	255	255	0	0	255	255
6	255	0	0	0	0	255	255	255	255	255	255	255
7	255	0	0	0	0	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

Grouping by Connectivity

Label Map As Output

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	1	1	1	1	255	255	255	2	2	255	255
2	255	1	1	1	1	255	255	2	2	2	2	255
3	255	1	1	1	1	255	255	2	2	2	2	255
4	255	1	1	1	1	255	255	2	2	2	2	255
5	255	1	1	1	1	255	255	255	2	2	255	255
6	255	1	1	1	1	255	255	255	255	255	255	255
7	255	1	1	1	1	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

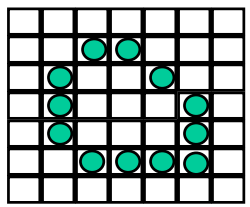
Edge Linking or Tracing

Requirement:

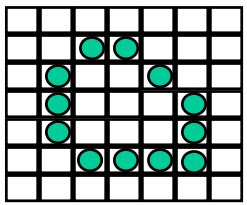
The edge must have the thickness of single pixel.

Principle:

If the primitive pixel at (i,j) is unlinked, label it and define it as the currently linked pixel. From the neighboring locations of (i,j), choose a primitive pixel according to a user-defined condition. If such a pixel exists, label it and define it as the currently linked pixel, and repeat the procedure; otherwise, one completely linked contour has been traced and labeled.



Operation:



Given a location (i,j) where $Value(i, j) = 0$, repeatedly do :

$$Label(i, j) = n$$

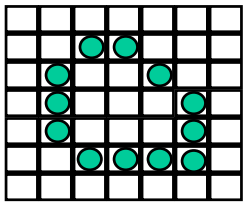
$$(cx, cy) = Centre(Subimage(i, j))$$

$$Neighbors(i, j) = \{(i-1, j-1), (i-1, j), (i-1, j+1), (i, j-1), (i, j+1), (i+1, j-1), (i+1, j), (i+1, j+1)\}$$

$\forall (s,t) \in Neighbors(i, j)$ and $Value(s,t) = 0$, choose the one nearest to (cx, cy).

Set : $i = s$ and $j = t$, and repeat the main loop.

Algorithm:



255	255	255	255	255	255	255
255	255	0	0	255	255	255
255	0	255	255	0	255	255
255	0	255	255	255	0	255
255	0	255	255	255	0	255
255	255	0	0	0	255	255
255	255	255	255	255	255	255

255	255	255	255	255	255	255
255	255	255	255	255	255	255
255	255	255	255	255	255	255
255	255	255	255	255	255	255
255	255	255	255	255	255	255
255	255	255	255	255	255	255
255	255	255	255	255	255	255

Step 1: Initialize the label map and set the label variable to 1 (n=1).

Step 2: Scan the image row by row and column by column.

Step 3: If encounter a unlabeled edge pixel, define it as the current pixel. And, compute the estimated image centre of a sub-image centered at the current pixel.

Step 4: Assign the label “n” to the current pixel. Now, this pixel is labeled.

Step 5: Find all the neighbors of the current pixel. If no neighbor, go to Step 7.

Step 6: Among the neighbors, choose the one nearest to the estimated image centre. Define it as the current pixel. Repeat Step 4, Step 5 and Step 6.

Step 7: Increment the label (n=n+1) and continue from Step 2.

Sample Program:

```
/* assuming that the pixel values of all the edge pixels are "0" */

unsigned char edgemap[512*512] ;
unsigned char labelmap[512*512] ;

static  int         cx, cy ;        /* estimated centre of pattern */
static  int         newx, newy ;  /* the next "current pixel"      */
static  double  min_distance ;
static  int         sx, sy ;

static  void  ComputeSubImageCentre(int x0, int y0)
{
         /* update the coordinates (pcx, pcy) */
}

static  void  CheckNearestNeighbor(int x, int y)
{
        /* identify the neighbor nearest to (pcx, pcy) */
}

static void  LinkEdge(int x, int y, int n)
{
       /* recursively link the edge points */
}

main(int argc, char **argv)
{
       int    x,  y, n ;

       n = 1 ;   sx = 10 ;      sy = 10 ;
       memset(labelmap, 255, 512*512) ;   /* clear the content of labelmap */

       for (y = 1 ; y < 511 ; y++)
             for (x = 1 ; x < 511 ; x++)
                   if (edgemap[y*512+x] == 0 && labelmap[y*512+x] == 255)
                   {
                             ComputeSubImageCentre(x, y);
                             LinkEdge(x, y, n) ;
                             n++ ;
                    }
}
```

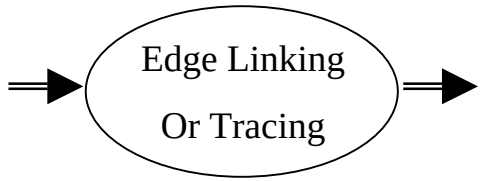

```
static  void  ComputeSubImageCentre(int x0, int y0)
{
        int  r, c, n ;

        cx = 0 ; cy = 0 ; n = 0 ;
        for (r = -sy/2 ; r < sy/2 ; r++)
                for (c = -sx/2; c < sx/2 ; c++)
                        if (edgemap[(y0+r)*512+x0+c] == 0 &&
                            labelmap[(y0+r)*512+x0+c] == 255)
                        {
                            cx = cx + x0 + c ;
                            cy = cy + y0 + r ;
                            n = n + 1 ;
                         }
        if (n != 0)
        {
              cx = cx/n ;
              cy = cy/n ;
         }
}
```

```
static void  LinkEdge(int x, int y, int n)
{
        labelmap[y*512+x] = n ;    /*  label the pixel */

        min_distance = 1000000.0 ;

        CheckNearestNeighbor(x-1, y-1) ;
        CheckNearestNeighbor(x, y-1) ;
        CheckNearestNeighbor(x+1,y-1);
        CheckNearestNeighbor(x-1,y) ;
        CheckNearestNeighbor(x+1,y) ;
        CheckNearestNeighbor(x-1, y+1) ;
        CheckNearestNeighbor(x-1, y+1) ;
        CheckNearestNeighbor(x, y+1) ;
        CheckNearestNeighbor(x+1, y+1) ;

         if (min_distance != 1000000.0)
               LinkEdge(newx, newy, n) ;
}
```

```
static  void  CheckNearestNeighbor(int x, int y)
{
        double  distance ;

        if (edgemap[y*512+x] != 0 || labelmap[y*512+x] != 255)
              return ;

        if ((distance = sqrt((cx - x)*(cx - x) + (cy - y)*(cy - y)))
                        < min_distance)
        {
                min_distance = distance;
                newx = x ; newy = y ;
        }
}
```


Result:

	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	0	0	0	0	255	255	255	0	0	255	255
2	255	0	255	255	0	255	255	0	255	255	0	255
3	255	0	255	255	0	255	255	0	255	255	0	255
4	255	0	255	255	0	255	255	0	255	255	0	255
5	255	0	255	255	0	255	255	255	0	0	255	255
6	255	0	255	255	0	255	255	255	255	255	255	255
7	255	0	0	0	0	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255



	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	1	1	1	255	255	255	255	2	2	255	255
2	255	1	255	255	1	255	255	2	255	255	2	255
3	255	1	255	255	1	255	255	2	255	255	2	255
4	255	1	255	255	1	255	255	2	255	255	2	255
5	255	1	255	255	1	255	255	255	2	2	255	255
6	255	1	255	255	1	255	255	255	255	255	255	255
7	255	255	1	1	255	255	255	255	255	255	255	255
8	255	255	255	255	255	255	255	255	255	255	255	255
9	255	255	255	255	255	255	255	255	255	255	255	255

Problem:

The edges must have the thickness of single pixel.

Solution:

If using Sobel operator to detect the edge-map, one can proceed with the following steps:

Step 1: Compute Horizontal Edges.

Step 2: Select the local maximum along the vertical axis as the edge pixels.

Step 3: Compute Vertical Edges.

Step 4: Select the local maximum along the horizontal axis as the edge pixels.

Step 5: Combine all the edge pixels to form the output.

Sample Program:

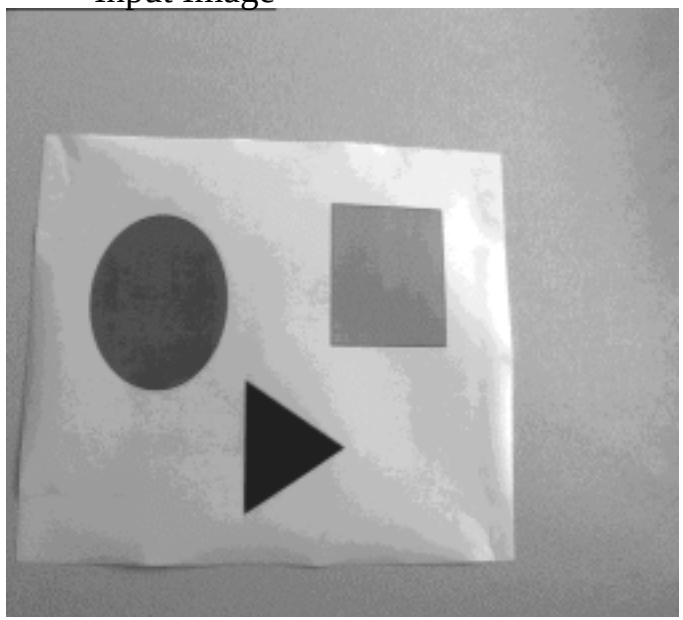
```
void EdgeDetection(void)
{
        int  i, j, v ;

        // Compute Horizontal Edges
        memset(gpBuffer1, 255, 512*512) ;
        for (i = 1 ; i < 511 ; i++)
        {
            for (j = 1 ; j < 511 ; j++)
            {
                v = abs(gpBuffer0[(i-1)*512+j-1] + pBuffer0[(i-1)*512+j] +  gpBuffer0[(i-1)*512+j+1] -
                        gpBuffer0[(i+1)*512+j-1] - gpBuffer0[(i+1)*512+j] - gBuffer0[(i+1)*512+j+1]) ;
                if (v >= 255)   gpBuffer1[i*512+j] = 0;
                if (v > 35 && v < 255) gpBuffer1[i*512+j] = (unsigned char) (255 - v) ;
            }
        }
        DoThinningAlongColumn() ;

        // Compute Vertical Edges
        memset(gpBuffer2, 255, 512*512) ;
        for (i = 1 ; i < 511 ; i++)
        {
            for (j = 1 ; j < 511 ; j++)
            {
                v = abs(gpBuffer0[(i-1)*512+j-1] + gpBuffer0[(i)*512+j-1] + gpBuffer0[(i+1)*512+j-1] -
                        gpBuffer0[(i-1)*512+j+1] - gpBuffer0[(i)*512+j+1] - gpBuffer0[(i+1)*512+j+1]) ;
                if (v >= 255)   gpBuffer2[i*512+j] = 0 ;
                if (v > 35 && v < 255) gpBuffer2[i*512+j] = (unsigned char) (255 - v) ;
            }
        }
        DoThinningAlongRow() ;

        // Display Horizontal Edges

        // Display Vertical Edges

        // Combined Edgemap
        memset(gpBuffer0, 255, 512*512) ;
        for (i = 0; i < 512*512 ; i++)
                if (gpBuffer1[i]  == 0 || gpBuffer2[i] == 0) gpBuffer0[i] = 0 ;

}
```


```
static DoThinningAlongColumn()
{
        int         i, j, v ;
        unsigned short temporaryBuffer[512*512] ;

        // step 1: smoothing
        memset(temporaryBuffer, 255*5, 512*512) ;
        for (i = 2 ; i < 512 - 2; i++)
        {
            for (j = 0 ; j < 512; j++)
            {
                temporaryBuffer[i*512+j] = 1*gpBuffer1[(i-2)*512+j] +
                                2*gpBuffer1[(i-1)*512+j] + 3*gpBuffer1[i*512+j] +
                                2*gpBuffer1[(i+1)*512+j] + 1*gpBuffer1[(i+2)*512+j] ;
            }
        }

        // step 2: select local minima
        memset(gpBuffer1, 255, 512*512) ;
        for (i = 2 ; i < 512 - 2; i++)
        {
            for (j = 0 ; j < 512; j++)
            {
                v = temporaryBuffer[i*512+j] ;
                if ( v < temporaryBuffer[(i-2)*512+j] && v < temporaryBuffer[(i-1)*512+j] &&
                        v < temporaryBuffer[(i+1)*512+j] &&
                        v < temporaryBuffer[(i+2)*512+j])
                {
                        gpBuffer1[i*512+j] = (unsigned char) 0 ;
                }
            }
        }
}
```

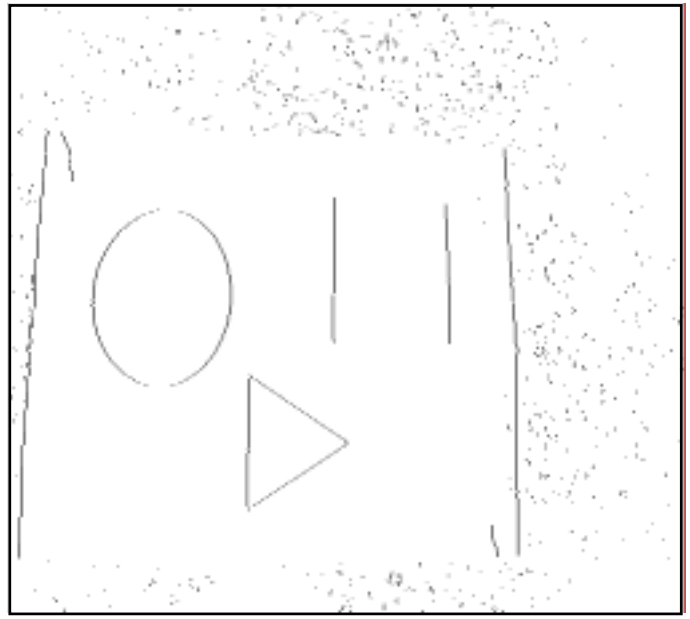

```
static DoThinningAlongRow()
{
        int         i, j, v ;
        unsigned short temporaryBuffer[512*512] ;

        // step 1: smoothing
        memset(temporaryBuffer, 255*5, 512*512) ;
        for (i = 0 ; i < 512; i++)
        {
            for (j = 2 ; j < 512 - 2 ; j++)
            {
                temporaryBuffer[i*512+j] = 1*gpBuffer2[i*512+j-2] + 2*gpBuffer2[i*512+j-1] +
                                3*gpBuffer2[i*512+j] + 2*gpBuffer2[i*512+j+1] + 1*gpBuffer2[i*512+j-2] ;
            }
        }

        // step 2: select local minima
        memset(gpBuffer2, 255, 512*512) ;
        for (i = 0 ; i < 512; i++)
        {
            for (j = 2 ; j < 512 - 2 ; j++)
            {
                v = temporaryBuffer[i*512+j] ;
                if ( v < temporaryBuffer[i*512+j-2] &&  v < temporaryBuffer[i*512+j-1] &&
                        v < temporaryBuffer[i*512+j+1] &&
                        v < temporaryBuffer[i*512+j-2])
                {
                        gpBuffer2[i*512+j] = (unsigned char) 0 ;
                }
            }
        }
}
```


Result

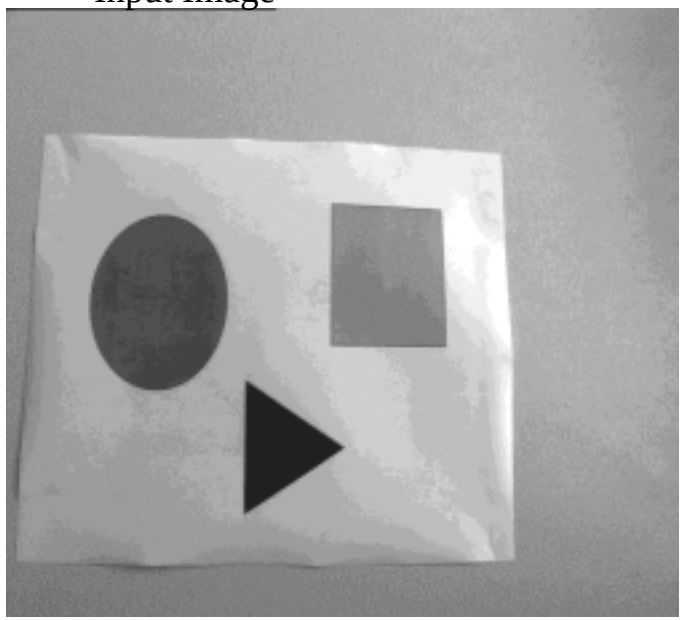
Input Image



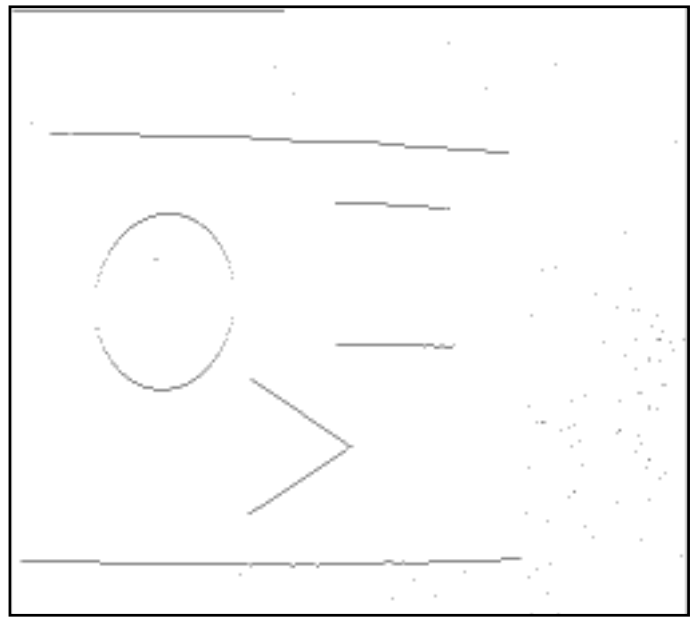
Thinned Vertical Edges



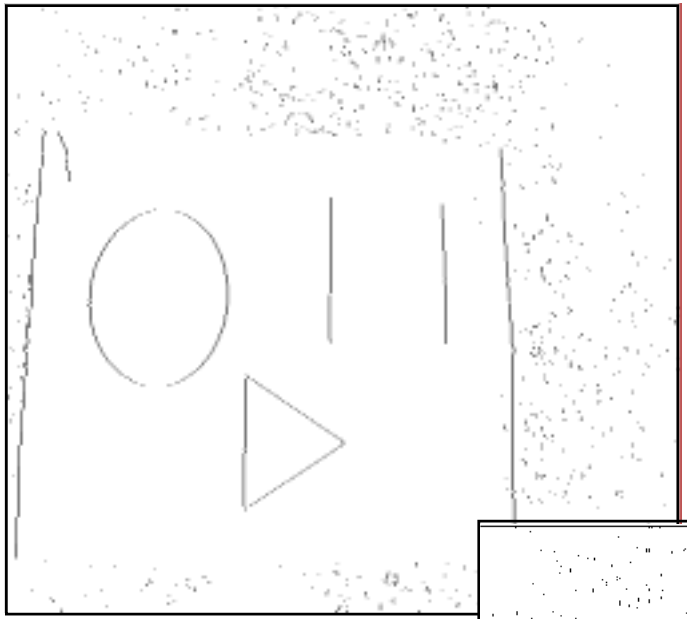
Input Image



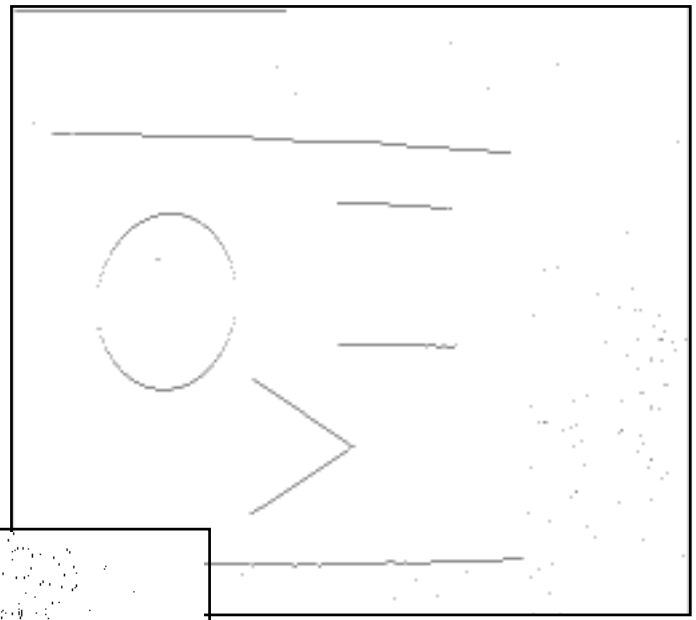
Thinned Horizontal Edges

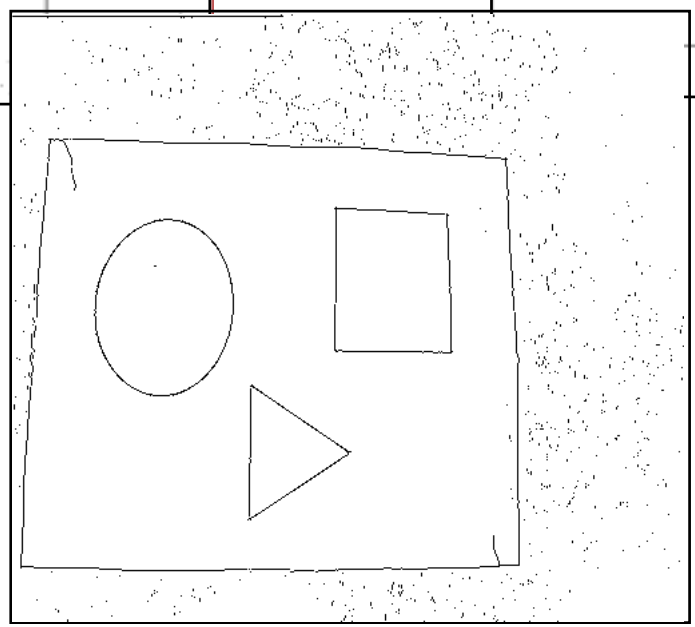


Thinned Vertical Edges



Thinned Horizontal Edges





Final Edgemap

SUMMARY

1. Pattern recognition refers to the process of identifying and classifying image patterns. The general steps of doing Pattern Recognition are:
    * Pattern detection
    * Pattern characterization
    * Pattern classification by classifier that needs to be trained before the use.

2. The general steps of detecting image patterns are:
    * Detect the primitive pixels that belong to either the boundary or area of image patterns
    * Group the primitive pixels into a set of patterns.

3. One can use either “grouping by connectivity” or “grouping by linking” techniques (or some combined approaches) to group the primitive pixels into patterns.