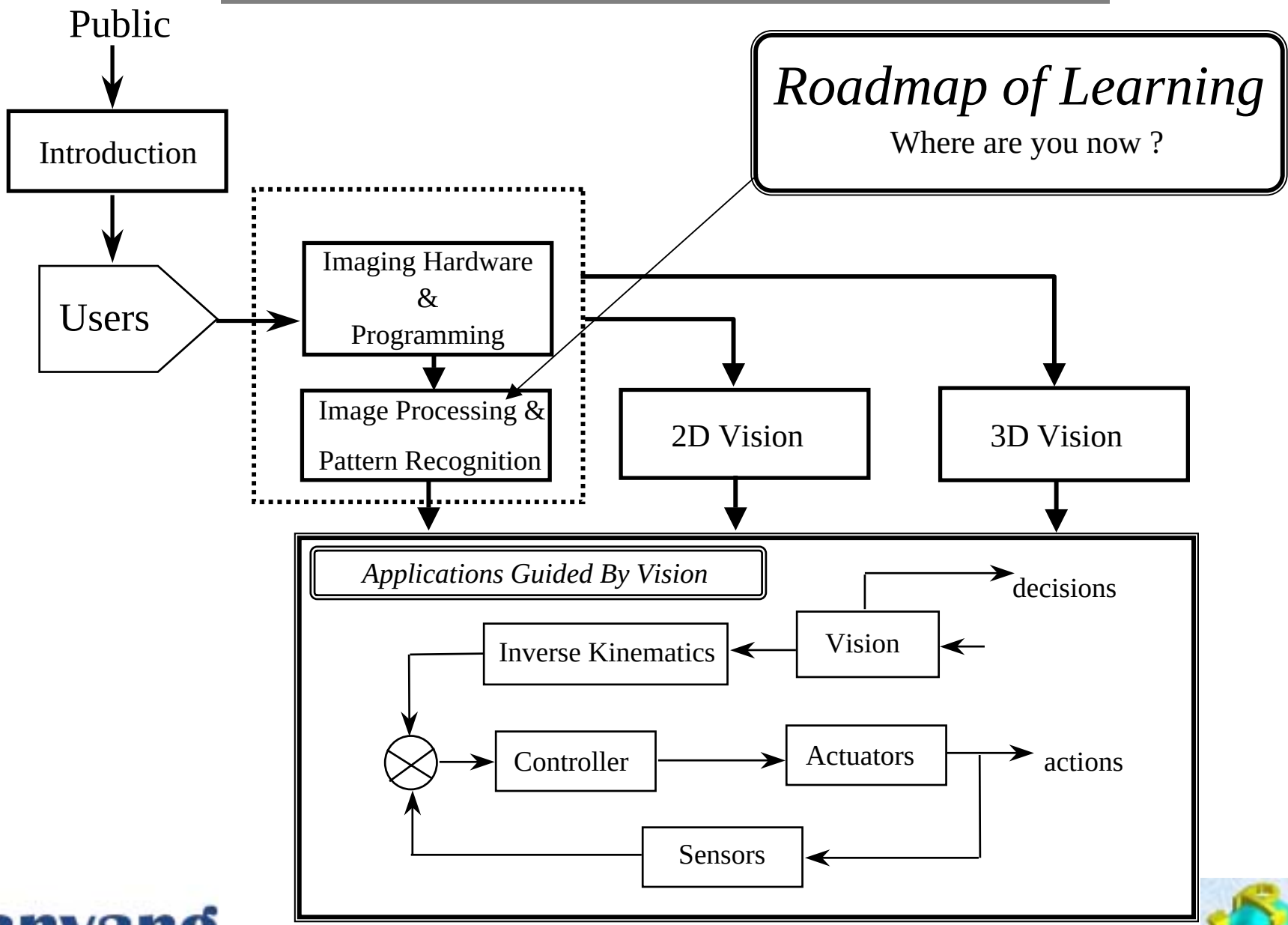


CONTENT

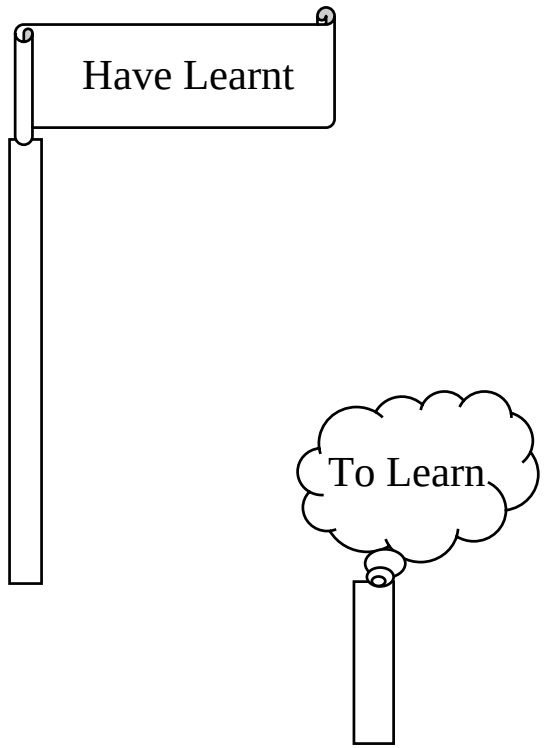
Chapter 4: Fundamentals of Image Processing

4.1 Point-Based Processing Techniques

4.2 Frame-Based Processing Techniques

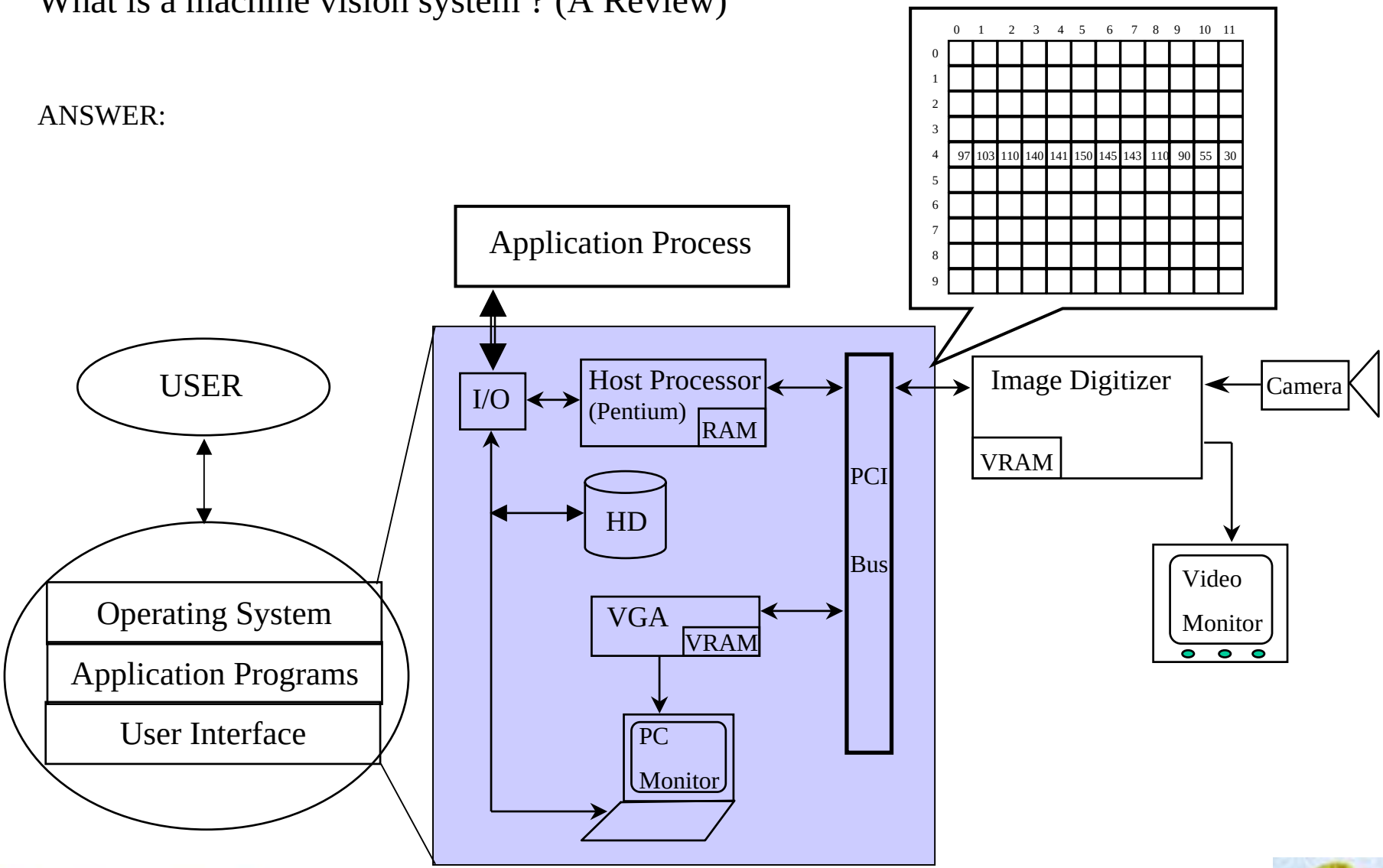
4.3 Area-Based Processing Techniques

4.4 Image Primitive Extraction



What is a machine vision system ? (A Review)

ANSWER:

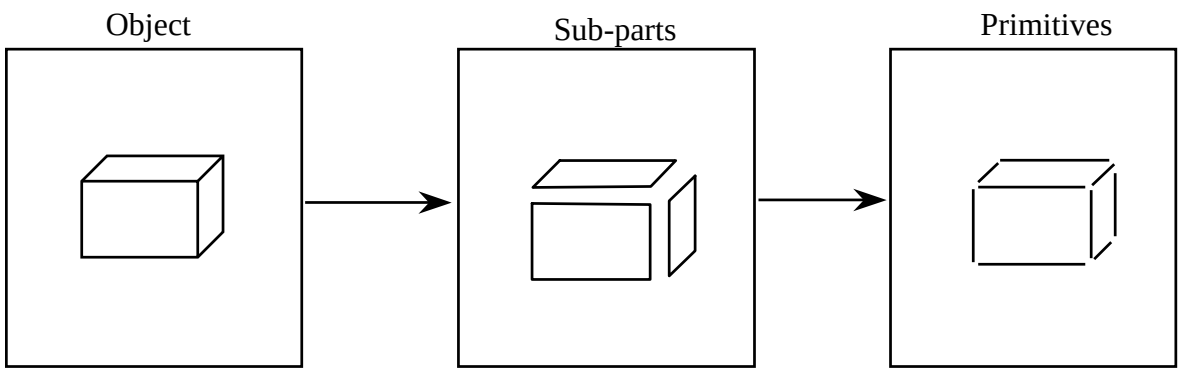


How to compute image primitives in order to detect objects ?

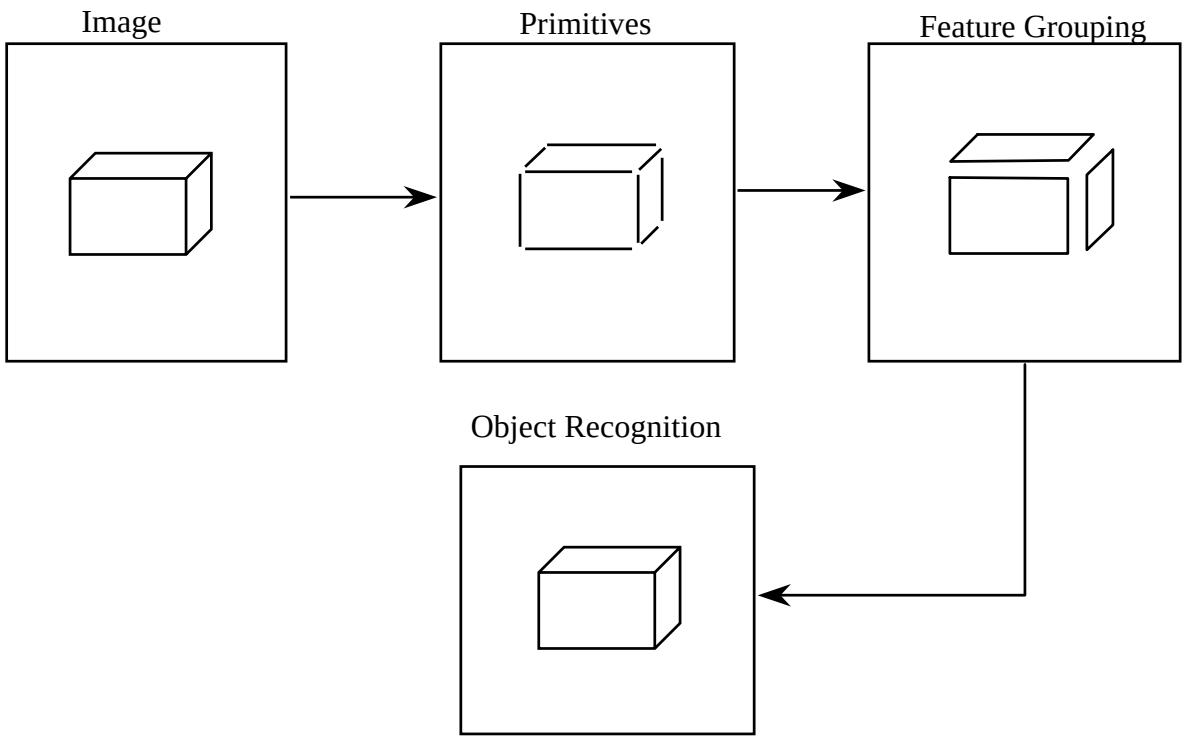
What is the general procedure of detecting objects from images ?

ANSWER:

1. Procedure of decomposing an object:



2. Procedure of object detection/recognition:



Primitive extraction is very important in the process of doing object detection and recognition.

How to do primitive extraction (in simple cases) ?

ANSWER:

Some simple and useful techniques are:

1. Image segmentation by thresholding.
2. Template matching.
3. Edge detection.

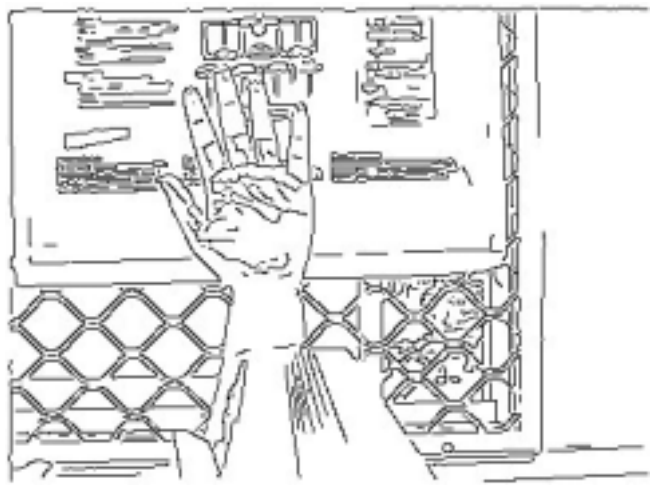


Image Segmentation By Thresholding

Objective:

To separate objects from its background by using the criterion of "pixel value difference".

Principle:

Input image : $I^{in} = \{g_{i,j}^{in} \mid 0 \le i < n,\ 0 \le j < m\}$

Output image : $I^{out} = \{g_{i,j}^{out} \mid 0 \le i < n,\ 0 \le j < m\}$

Operation :

$$g_{i,j}^{out} = \begin{cases} v_1 & \text{if } g_{i,j}^{in} < s_0 \\ v_2 & \text{if } g_{i,j}^{in} \ge s_0 \end{cases}$$

Illustration

Input Image

	0	1	2	3	4	5	6	7	8	9	10	11
0	30	30	30	30	30	30	30	30	30	30	30	30
1	30	30	30	30	30	30	30	30	30	30	30	30
2	30	30	66	66	66	66	66	66	66	66	30	30
3	30	30	66	66	66	66	66	66	66	66	30	30
4	30	30	66	66	66	66	66	66	66	66	30	30
5	30	30	30	30	99	99	99	99	30	30	30	30
6	30	30	30	30	99	99	99	99	30	30	30	30
7	30	30	30	30	99	99	99	99	30	30	30	30
8	30	30	30	30	99	99	99	99	30	30	30	30
9	30	30	30	30	30	30	30	30	30	30	30	30

s0 = 50;

v1 = 0 ;

v2 = 200

Output Image

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	200	200	200	200	200	200	200	200	0	0
3	0	0	200	200	200	200	200	200	200	200	0	0
4	0	0	200	200	200	200	200	200	200	200	0	0
5	0	0	0	0	200	200	200	200	0	0	0	0
6	0	0	0	0	200	200	200	200	0	0	0	0
7	0	0	0	0	200	200	200	200	0	0	0	0
8	0	0	0	0	200	200	200	200	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0

Problems:

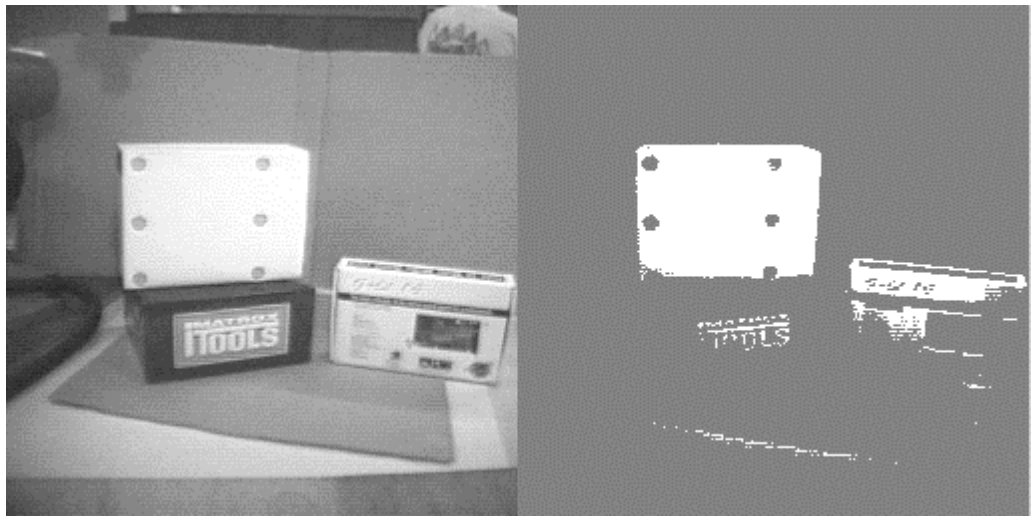
1. The pixel values of object must be different than the pixel value of background
2. Scene must be simple.

Sample Program

```
#include <stdio.h>

unsigned char   image[512*512] ;

main(int argc, char **argv)
{
        int  r, c, s0, v1, v2 ;

       s0 = 100 ; v1 = 0 ; v2 = 1 ;

       for (r = 0 ; r < 512 ; r++)
       {
              for (c = 0 ; c < 512 ; c++)
                     if (image[r*512+c] < s0)
                            image[r*512+c] = v1 ;
                     else
                            image[r*512+c] = v2 ;
       }
}
```


Result

Input Output



Template Matching

Objective:

To detect the location of a sub-image (region) that looks like the sample image of an object.

Principle:

Input images : $I^{in} = \{g^{in}_{i,j} \mid 0 \le i < n,\ 0 \le j < m\}$

$I^{s} = \{g^{s}_{i,j} \mid 0 \le i < n_1,\ 0 \le j < m_1\}$

Condition : $n_1 < n$ and $m_1 < m$.

Output : $(i_0, j_0) \mid [sub(I^{in}, i_0, j_0) - I^{S}] = \min, \forall (i_0, j_0)$

Operation :

$$\begin{cases} Sub(I^{in}, i, j) = \{(g^{sub}_{s,t} = g^{in}_{i-n_1/2-s,\, j-m_1/2-t}) \mid 0 \le s < n_1, 0 \le t < m_1\} \\ \min\limits_{i,j}\{\sum\limits_{s=0}^{n_1}\sum\limits_{t=0}^{m_1} | g^{sub}_{s,t} - g^{s}_{s,t} |\}. \end{cases}$$

Input Image

	0	1	2	3	4	5	6	7	8	9	10	11
0	30	30	30	30	30	30	30	30	30	30	30	30
1	30	30	30	30	30	30	30	30	30	30	30	30
2	30	30	66	66	66	66	66	66	66	66	30	30
3	30	30	66	66	66	66	66	66	66	66	30	30
4	30	30	66	66	66	66	66	66	66	66	30	30
5	30	30	30	30	99	99	99	99	30	30	30	30
6	30	30	30	30	99	99	99	99	30	30	30	30
7	30	30	30	30	99	99	99	99	30	30	30	30
8	30	30	30	30	99	99	99	99	30	30	30	30
9	30	30	30	30	30	30	30	30	30	30	30	30

Illustration

Sample Image

30	30	30	30	30
30	30	66	66	66
30	30	66	66	66

Output is the location at (2, 2)

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

Problem:

The pixel values of sample image must be similar to the pixel values of object image (sub-image occupied by object)

Sample Program

```
#include <stdio.h>

void   DoTemplateMatching(unsigned char *image_in, int rx, int  ry, unsigned char *sample, int  sx, int  sy, int  *i0, int *j0)
{
       int   r, c, s, t, min_value, difference ;

       min_value = 1000000 ;

      for (r = ry/2 ; r < ry - sy/2; r++)                  /* do not process border area */
      {
            for (c = rx/2 ; c < rx - sx/2 ; c++)         /* do not process border area */
            {
                  difference = 0 ;
                  /* at the location (r, c), compute the image difference */
                  for (s = 0 ; s < sy ; s++)
                  {
                        for (t = 0 ; t < sx ; t++)
                        {
                              difference = difference + abs(image_in[(r-sx/2-s)*rx+(c-sy/2-t)] - sample[s*sx+t]) ;
                        }
                  }

                  if (difference < min_value)
                  {
                        min_value = difference ;    *i0 = r ;    *j0 = c ;
                  }
            }
       }
}
```


Result

Left Image Right Image

Template matching

Edge Detection

Objective:

To detect the important transition of pixel values because this kind of transition reflects:

1. Contours of objects.
2. Apparent boundaries between objects.
3. Contours created by shadow, etc.

Principle:

1. Input Image :

$$I^{in} = \{g^{in}_{i,j} \mid 0 \le i < n,\ 0 \le j < m\}$$

2. Ideal Vertical/Horizontal Edges:

$$I^{v} = \begin{array}{|c|c|c|}\hline -1 & 0 & +1 \\ \hline -1 & 0 & +1 \\ \hline -1 & 0 & +1 \\ \hline \end{array} \qquad I^{h} = \begin{array}{|c|c|c|}\hline -1 & -1 & -1 \\ \hline 0 & 0 & 0 \\ \hline +1 & +1 & +1 \\ \hline \end{array}$$

3. Output Image :

$$I^{edge} = \{g^{edge}_{i,j} \mid 0 \le i < n,\ 0 \le j < m\}$$

4. Operation : $I^{edge} = \underline{I^{in} \otimes I^{v}} + \underline{I^{in} \otimes I^{h}}$.

Vertical edge

Horizontal edge

Illustration

Input Image

	0	1	2	3	4	5	6	7	8	9	10	11
0	30	30	30	30	30	30	30	30	30	30	30	30
1	30	30	30	30	30	30	30	30	30	30	30	30
2	30	30	66	66	66	66	66	66	66	66	30	30
3	30	30	66	66	66	66	66	66	66	66	30	30
4	30	30	66	66	66	66	66	66	66	66	30	30
5	30	30	30	30	99	99	99	99	30	30	30	30
6	30	30	30	30	99	99	99	99	30	30	30	30
7	30	30	30	30	99	99	99	99	30	30	30	30
8	30	30	30	30	99	99	99	99	30	30	30	30
9	30	30	30	30	30	30	30	30	30	30	30	30

An Ideal vertical edge

-1	0	+1
-1	0	+1
-1	0	+1

Output:

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	36	36	0	0	0	0	0	0	36	36	0
2	0	72	72	0	0	0	0	0	0	72	72	0
3	0	108	108	0	0	0	0	0	0	108	108	0
4	0	72	72	69	69	0	0	69	69	72	72	0
5	0	36	36	138	138	0	0	138	138	36	36	0
6	0	0	0	207	207	0	0	207	207	0	0	0
7	0	0	0	207	207	0	0	207	207	0	0	0
8	0	0	0	138	138	0	0	138	138	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0

Vertical edges

Input Image

	0	1	2	3	4	5	6	7	8	9	10	11
0	30	30	30	30	30	30	30	30	30	30	30	30
1	30	30	30	30	30	30	30	30	30	30	30	30
2	30	30	66	66	66	66	66	66	66	66	30	30
3	30	30	66	66	66	66	66	66	66	66	30	30
4	30	30	66	66	66	66	66	66	66	66	30	30
5	30	30	30	30	99	99	99	99	30	30	30	30
6	30	30	30	30	99	99	99	99	30	30	30	30
7	30	30	30	30	99	99	99	99	30	30	30	30
8	30	30	30	30	99	99	99	99	30	30	30	30
9	30	30	30	30	30	30	30	30	30	30	30	30

-1	-1	-1
0	0	0
+1	+1	+1

An Ideal horizontal edge

Output:

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	36	72	108	108	108	108	108	108	72	36	0
2	0	36	72	108	108	108	108	108	108	72	36	0
3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	36	72	105	102	99	99	102	105	72	36	0
5	0	36	72	105	102	99	99	102	105	72	36	0
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	69	138	138	138	138	69	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0

Horizontal edges

Problems:

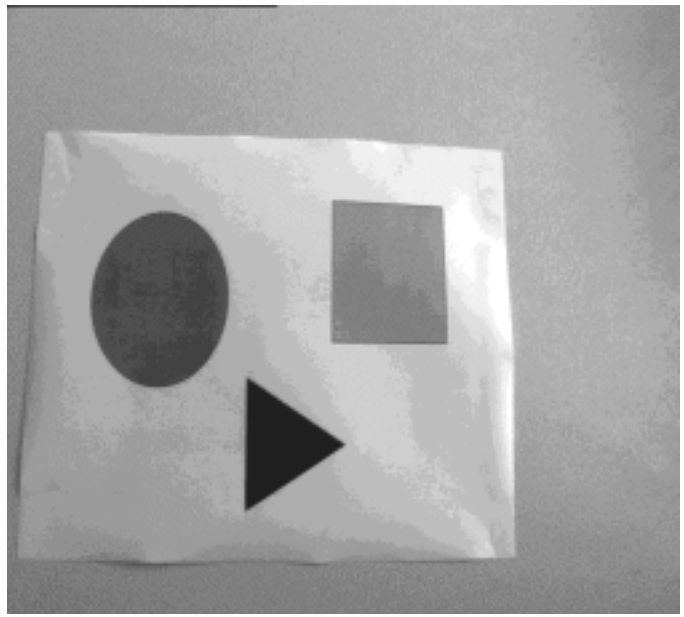
1. The need of doing “edge thinning”.
2. The need of doing “edge linking”.

Sample Program

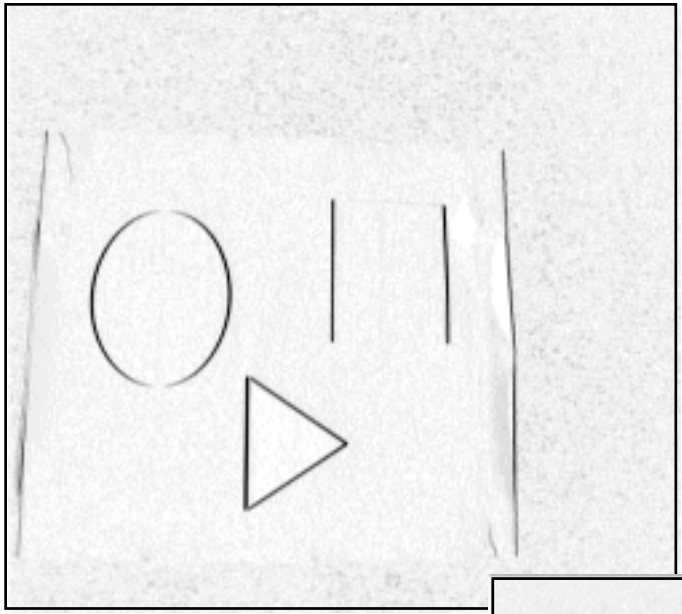
```
void EdgeDetection(void)
{
        int  i, j, v ;

        // 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] + gpBuffer0[(i-1)*512+j] + gpBuffer0[(i-1)*512+j+1] -
                    gpBuffer0[(i+1)*512+j-1] - gpBuffer0[(i+1)*512+j] - gpBuffer0[(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) ;
            }
        }

        // 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) ;
            }
        }

        // Display Horizontal/Vertical Edges
        memcpy(gpBuffer0, gpBuffer1, 512*512) ;   DisplayBuffer() ;
        MessageBox(hWndCtl, "Horizontal Edges", "Message", NULL) ;

        memcpy(gpBuffer0, gpBuffer2, 512*512) ;   DisplayBuffer() ;
        MessageBox(hWndCtl, "Vertical Edges", "Message", NULL) ;

        // Combined Edgemap
        for (i = 0; i < 512*512 ; i++)  gpBuffer0[i] = (gpBuffer1[i] + gpBuffer2[i])/2 ;
        DisplayBuffer() ;   MessageBox(hWndCtl, "Final Edgemap", "Message", NULL) ;
}
```


Result

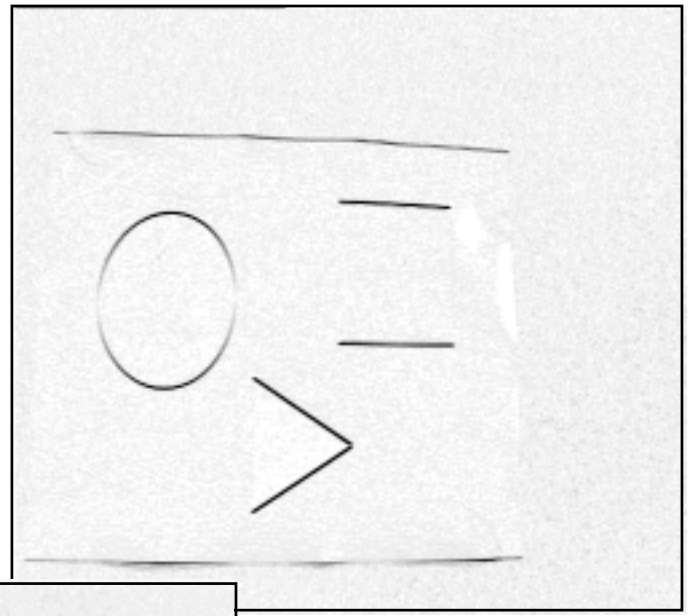
Input Image



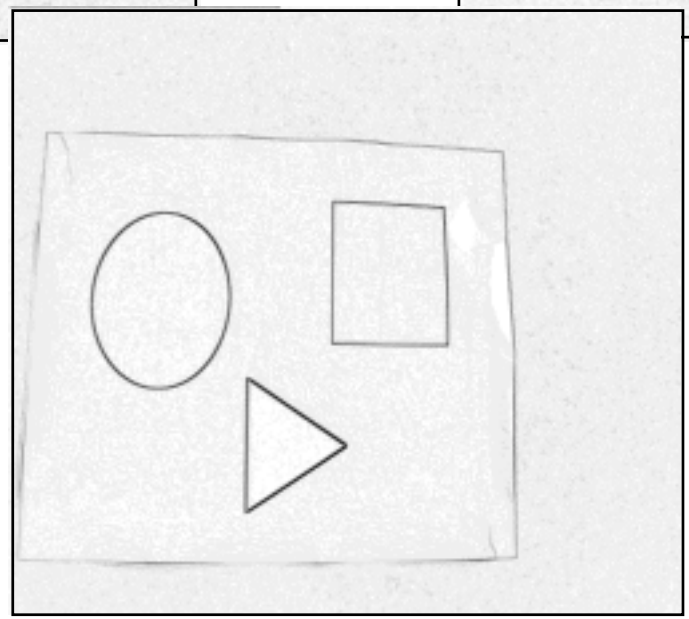
Vertical Edges



Horizontal Edges

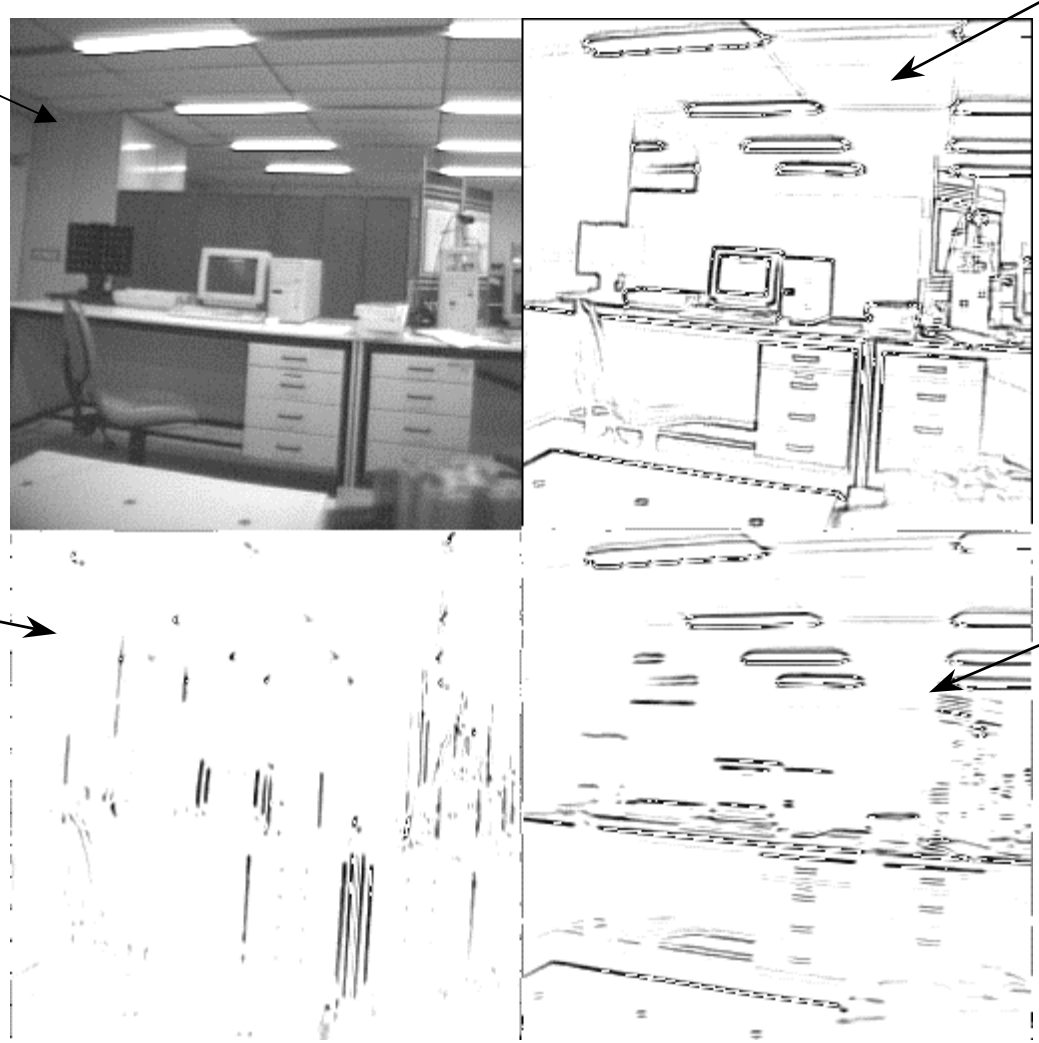


Combined Edges



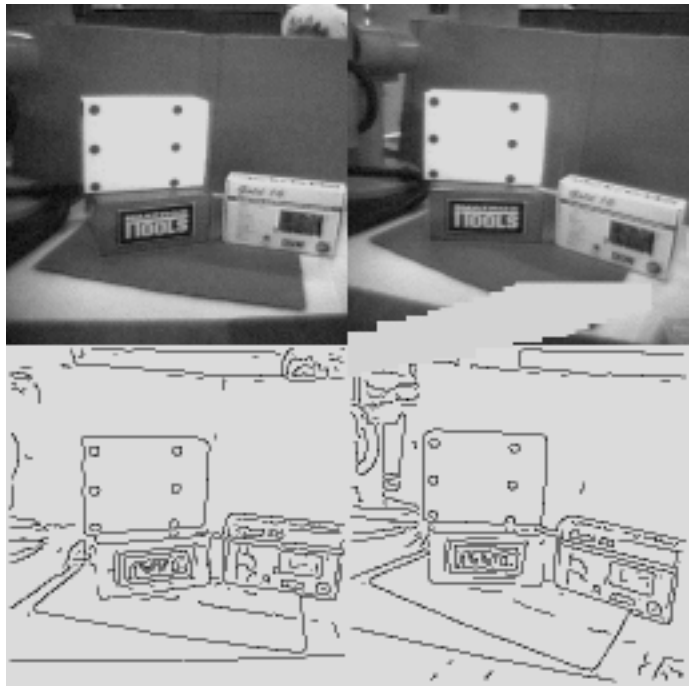
Combined edges

Input Image

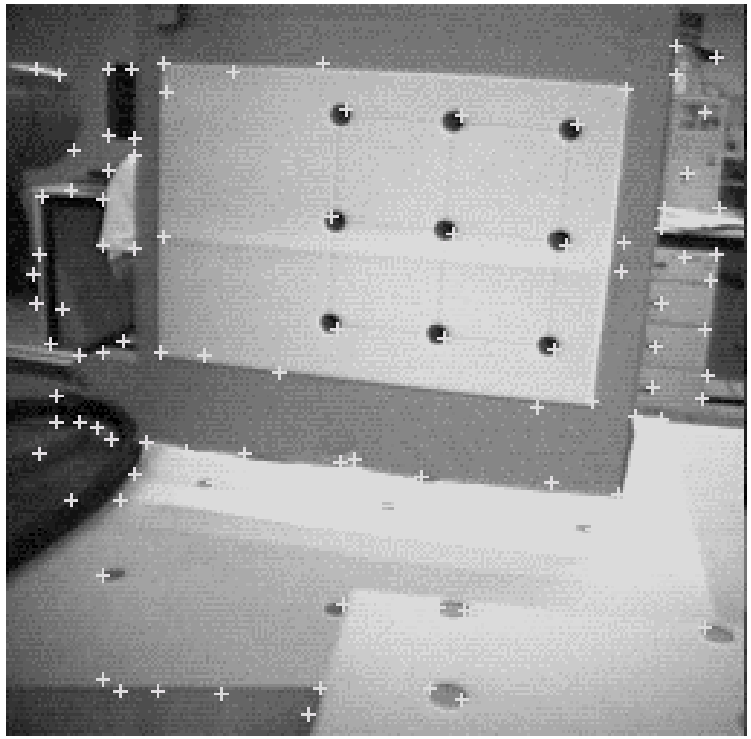


Vertical edges

Horizontal edges



Canny-Deriche Edge Detector



Detector for “Point of Interest”

SUMMARY

1. Object can be decomposed into sub-parts. A sub-part can be decomposed into primitives.

2. Object detection/recognition starts by doing “primitive extraction”.

3. Some useful techniques for “primitive extraction” are:

* Image Segmentation By thresholding.
  (Problem: the image must be simple)

* Template Matching.
  (Problem: the pixel values must be similar)

* Edge Detection.
  (Problem: some post-processing are needed)