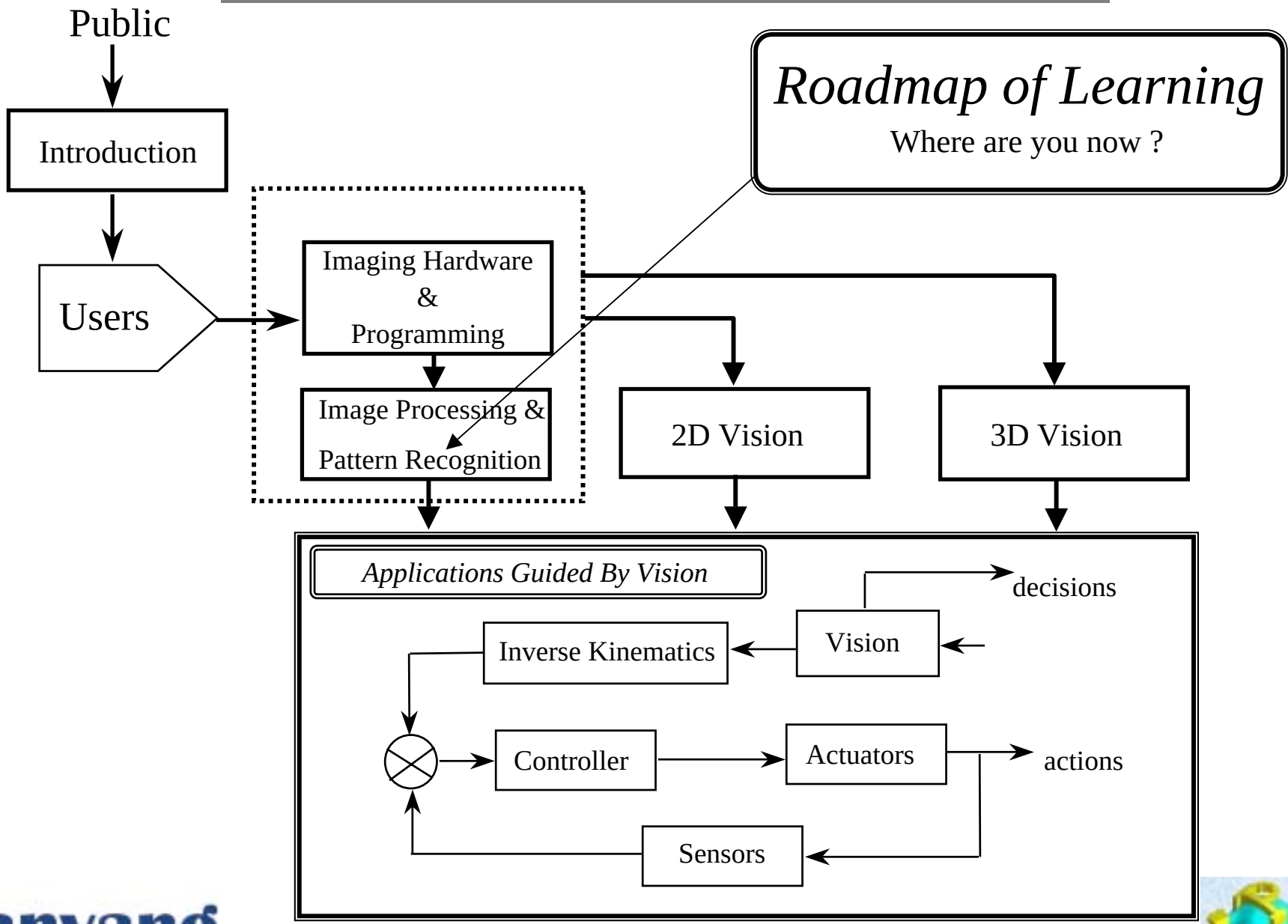
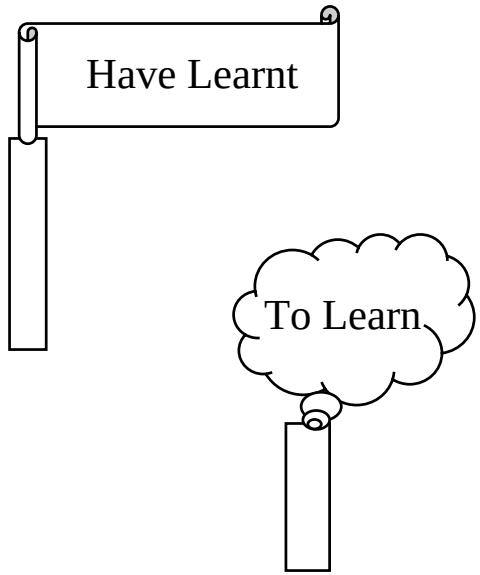


CONTENT

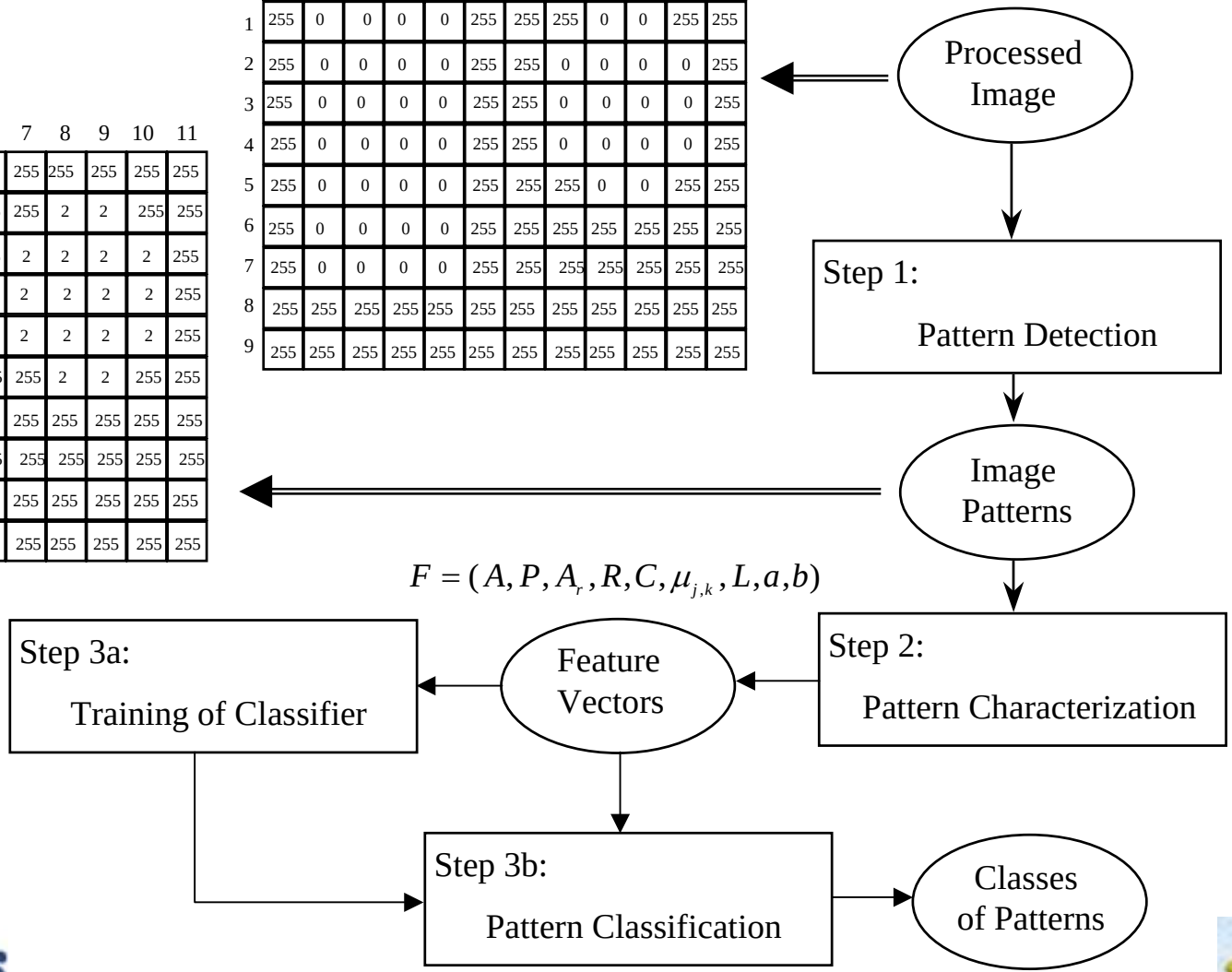


What are the general steps of doing pattern recognition ? (A review)

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

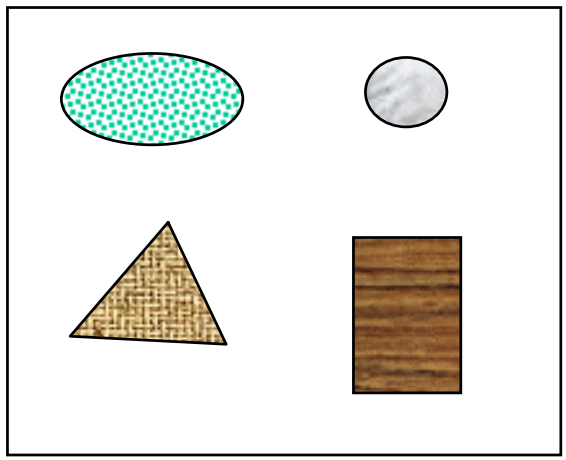
	0	1	2	3	4	5	6	7	8	9	10	11
0	255	255	255	255	255	255	255	255	255	255	255	255
1	255	0	0	0	0	255	255	255	0	0	255	255
2	255	0	0	0	0	255	255	0	0	0	0	255
3	255	0	0	0	0	255	255	0	0	0	0	255
4	255	0	0	0	0	255	255	0	0	0	0	255
5	255	0	0	0	0	255	255	255	0	0	255	255
6	255	0	0	0	0	255	255	255	255	255	255	255
7	255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



Is it necessary to do pattern characterization before implementing pattern classification ?

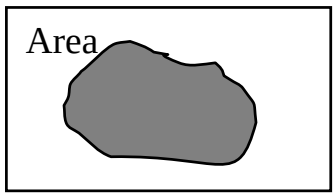
ANSWER:

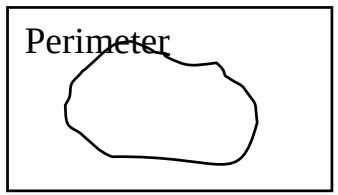
Yes, because pattern classification must rely on some quantitative (or qualitative) measurements. Pattern characterization will result in the output of a so-called "feature vector" for each image pattern.

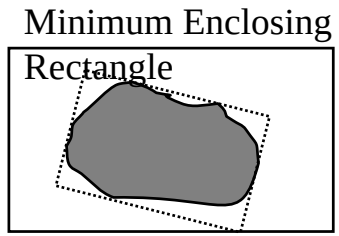


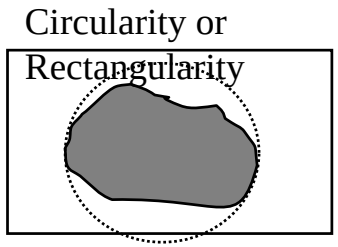
What are the measurable characteristics of an image pattern ?

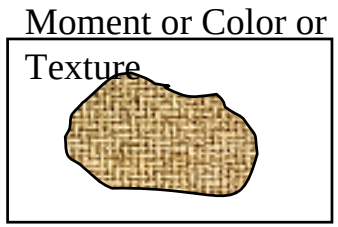
ANSWER:





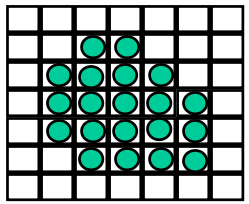






Area ("A")

CASE 1: If we know the primitive pixels that belong to the area of an image pattern:



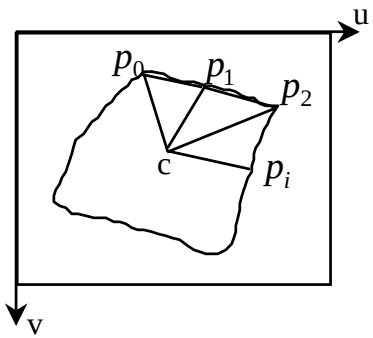
A = Number of The Primitive Pixels of The Pattern.

Sample Program:

```
typedef struct {
        int   u, v ;
}       Point ;

typedef struct {
        int         label ;
        Point       pixel[10000] ;
        int         nb_pixel ;
}       Pattern ;

unsigned char   image[512*512] ;
unsigned char   labelmap[512*512] ;
Pattern         pattern[100] ;
int             nb_pattern ;

static int CountTheNumberOfPixels(int label)
{
        int  r, c, nb ;
        nb = 0 ;
        for (r = 0 ; r < 512 ; r++)
                for (c = 0 ; c < 512 ; c++)
                        if (labelmap[r*512+c] == label)
                        {
                                pattern[label].pixel[nb].u = c ;
                                pattern[label].pixel[nb].v = r ;
                                nb++ ;
                        }
        return nb ;
}

main(int argc, char **argv)
{
        int      label ;
        for (label = 1 ; label <= nb_label; label++)
                pattern[label].nb_pixel = CountTheNumberOfPixels(label) ;
}
```


Area ("A")

CASE 2: If we know the primitive pixels that belong to the boundary of an image pattern:



$$c = (u_0, v_0)$$

$$p_i = (u_i, v_i) \quad i = 0,1,2,...,n-1 \quad \text{and} \quad p_n = p_0$$

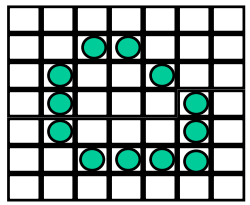
$$A = \frac{1}{2}\sum_{i=0}^{n} |(u_i - u_0)(v_{i+1} - v_0) - (u_{i+1} - u_0)(v_i - v_0)|$$

Sample Program:

```
typedef struct {
        int   u, v ;
}       Point ;

typedef struct {
        int         label ;
        Point       pixel[10000] ;
        int         nb_pixel ;
        int         cu, cv ;            /*  centre of the pattern */
        double      area ;              /*  area of the pattern   */
}       Pattern ;

unsigned char   image[512*512] ;
unsigned char   labelmap[512*512] ;
Pattern         pattern[100] ;
int             nb_pattern ;


static int ComputeTheAreaOfThePattern(int label)
{
        int    p ;
        pattern[label].area = 0.0 ;
        for (p = 0 ; p < pattern[label].nb_pixel ; p++)
        {
                pattern[label].area += 0.5*(
                        (pattern[label].pixel[p].u - pattern[label].cu)*
                        (pattern[label].pixel[p+1].v - pattern[label].cv) -
                        (pattern[label].pixel[p+1].u - pattern[label].cu)*
                        (pattern[label].pixel[p].v - pattern[label].cv)   ) ;
        }
}
```


Perimeter ("P")

CASE 1: If we know the primitive pixels that belong to the boundary of an image pattern:



$p_i = (u_i, v_i)$ i=0,1,2, ..., n-1 and $p_n = p_0$

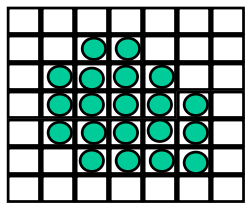
$$P = \sum_{i=0}^{n-1} \sqrt{(u_{i+1} - u_i)^2 + (v_{i+1} - v_i)^2}$$

Sample Program:

```
typedef struct {
        int   u, v ;
}       Point ;

typedef struct {
        int         label ;
        Point       pixel[10000] ;
        int         nb_pixel ;
        int         cu, cv ;            /*  centre of the pattern      */
        double      area ;              /*  area of the pattern        */
        double      perimeter ;         /*  perimeter of the pattern */
}       Pattern ;

unsigned char   image[512*512] ;
unsigned char   labelmap[512*512] ;
Pattern         pattern[100] ;
int             nb_pattern ;

static int ComputeThePerimeterOfThePattern(int label)
{
        int    p ;
        pattern[label].perimeter = 0.0 ;
        for (p = 0 ; p < pattern[label].nb_pixel ; p++)
        {
                pattern[label].perimeter += sqrt(
                        (pattern[label].pixel[p+1].u - pattern[label].pixel[p].u)*
                        (pattern[label].pixel[p+1].u - pattern[label].pixel[p].u) +
                        (pattern[label].pixel[p+1].v - pattern[label].pixel[p].v)*
                        (pattern[label].pixel[p+1].v - pattern[label].pixel[p].v)   ) ;
        }
}
```


Perimeter ("P")

CASE 2: If we know the primitive pixels that belong to the area of an image pattern:



Algorithm:

Step 1: Identify and link the boundary pixels

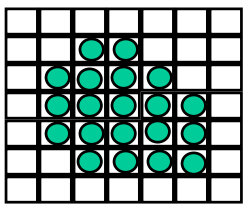
(Criterion:

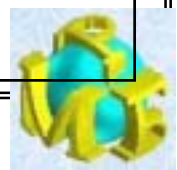
A primitive pixel is a boundary pixel if one of its (North, South, East,West) neighbors does not belong to the pattern).

Step 2: Compute the perimeter with the linked boundary pixels.

Sample Program:

```
#define    True     1
#define    False    0

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

static int IsItBoundaryPixel(int u, int v)
{
       if (labelmap[(v-1)*512+u] == 255)  return  True ;    if (labelmap[(v+1)*512+u] == 255) return True ;
       if (labelmap[v*512+u-1] == 255) return True ;        if (labelmap[v*512+u+1] == 255) return True ;
       return False ;
}

static void   MarkBoundaryPixels(int label)
{
      int    u, v ;

      for (v = 0 ; v < 512 ; v++)
            for (u = 0 ; u < 512 ; u++)
                  if (labelmap[v*512+u] == label)
                  {
                        if (IsItBoundaryPixel(u,v) == True)  labelmap[v*512+u] = 0 ;
                  }
}

static void  LinkMarkedPixels(int label)
{
      /* link the boundary pixels and fill them into the Pattern structure */
}

main()
{
      memset(labelmap, 255, 512*512) ;
      MarkBoundaryPixels(label);
      LinkMarkedPixels(label);
      ComputeThePerimeterOfThePattern(label)
}
```

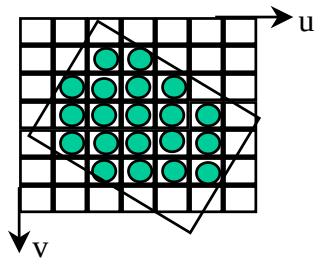




Minimum Enclosing Rectangle ("Ar")

Definition:

It refers to the smallest enclosing rectangle of an image pattern.



Algorithm:

Step 1: Determine the ranges (umin, umax, vmin, vmax) in both horizontal and vertical directions.

Step 2: Compute the size of the rectangle determined by (umin, umax, vmin, vmax).

Step 3: Rotate the pattern by an angle.

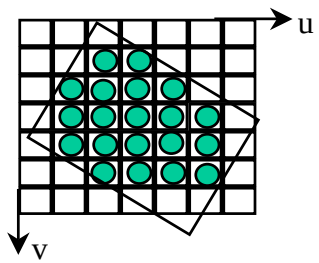
Step 4: Repeat Step 1, Step 2 and Step 3.

Step 5: Choose the smallest rectangle as the output.

Sample Program:

```
unsigned  char image[512*512] ;
unsigned char labelmap[512*512], rotated_image[512*512] ;

static  double   umin, umax, vmin, vmax ;
static  double   min_size ;

static void  UpdateTheEnclosingRectangle(double r, double c)
{
        if (r > vmax)   vmax = r ;          if (r < vmin)   vmin = r ;
        if (c > umax)  umax = c ;          if (c < umin)   umin = c ;
}

static void ComputeMinimumEnclosingRectangle(int label)
{
       double  angle, r1, c1 ;
       int        r, c, size ;

       min_size = 1000000.0 ;
       for (angle = 0.0 ; angle < 90.0 ; angle += 3.0)
       {
            umin = 1000.0 ;              umax = 0.0 ;
            vmin = 1000.0 ;              vmax = 0.0 ;

            for (r = 0 ; r < 512 ; r++)
                 for (c = 0 ; c < 512 ; c++)
                      if (labelmap[r*512+c] == label)
                      {
                            c1 = cos(3.14*angle/180)*c - sin(3.14*angle/180)*r ;
                            r1 =  sin(3.14*angle/180)*c + cos(3.14/angle/180)*r ;

                            UpdateTheEnclosingRectangle(r1, c1) ;
                       }
            this_size = (umax - umin)*(vmax-vmin) ;
            if (this_size < min_size)    min_size = this_size ;
       }
}
```

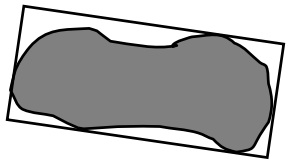
$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$



Rectangularity ("R")

Definition:

It refers to the ratio between the pattern's area and the pattern's MER (Minimum Enclosing Rectangle).

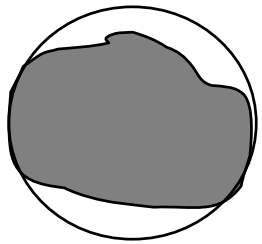


$$R = \frac{A}{A_r}. \qquad R \in (0,1)$$

Circularity ("C")

Definition:

It refers to the ratio between the pattern's squared perimeter and the pattern's area.



Perimeter of a circle: $2\pi r$

Area of a circle: $\frac{1}{2}\pi r^2$

$$C = 8\pi \bullet \frac{A}{P^2}. \qquad C \in (0,1)$$

Moments ("M")

Definition:

If f(x,y) is a bounded function, its moment of (j+k)th order is defined as:

$$m_{j,k} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^j y^k f(x,y)dxdy$$

And, its central moment of (j+k)th order is defined as:

$$\mu_{j,k} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x - \frac{m_{1,0}}{m_{0,0}})^j (y - \frac{m_{0,1}}{m_{0,0}})^k f(x,y)dxdy$$

Application to image pattern:

If we define a bounded discrete function f(u,v) as follows:

$$f(u,v) = \begin{cases} I(u,v) & \text{if } (u,v) \text{ belongs to the pattern;} \\ 0 & \text{otherwise;} \end{cases}$$

then, the moment and central moment of (j+k)th order will be:

$$\begin{cases} m_{j,k} = \sum_{v=0}^{n-1} \sum_{u=0}^{m-1} [(u^j v^k f(u,v)] \\ \mu_{j,k} = \sum_{v=0}^{n-1} \sum_{u=0}^{m-1} [(u - \frac{m_{1,0}}{m_{0,0}})^j (v - \frac{m_{0,1}}{m_{0,0}})^k f(u,v)] \end{cases}$$

(the image has “n” rows and “m” columns).

Sample Program:

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

static double ComputeMoment(int label, int j, int k)
{
        double  m  ;        int         r, c ;
        m = 0.0 ;
        for (r = 0 ; r < 512 ; r++)
              for (c = 0 ; c < 512 ; c++)
                   if (labelmap[r*512+c] == label)  m += (pow(c, j)*pow(r, k)*image[r*512+c]) ;
        return m ;
}

static double ComputeCentralMoment(int label, int j, int k)
{
        double u, m10, m01, m00 ;        int        r, c ;

        m00 = ComputeMoment(label, 0, 0) ;
        m10 = ComputeMoment(label, 1, 0)/m00 ;       m01 = ComputeMoment(label, 0, 1)/m00 ;
        u = 0.0 ;
        for (r = 0 ; r < 512 ; r++)
             for (c = 0 ; c < 512 ; c++)
                  if (labelmap[r*512+c] == label)   u += (pow(c-m10, j)*pow(r-m01, k)* image[r*512+c]) ;
         return u ;
}

main(int argc, char **argv)
{
      int         label, j, k  ;      double   u ;

      /* set j, k and label values */
      u = ComputeCentralMoment(label, j, k) ;
}
```

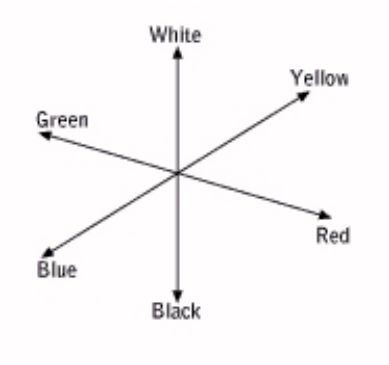

Color

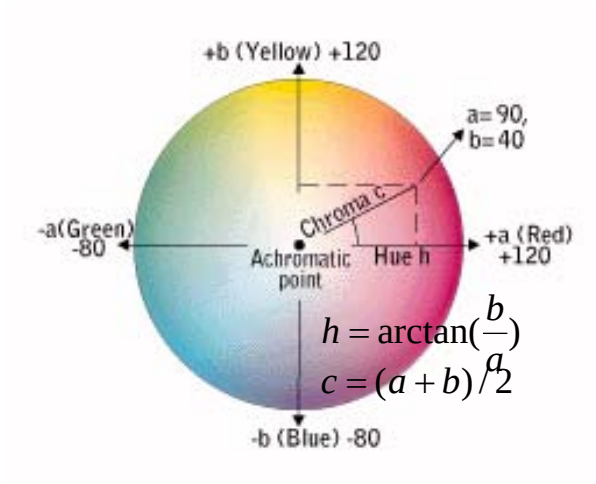
CIELAB Color Space

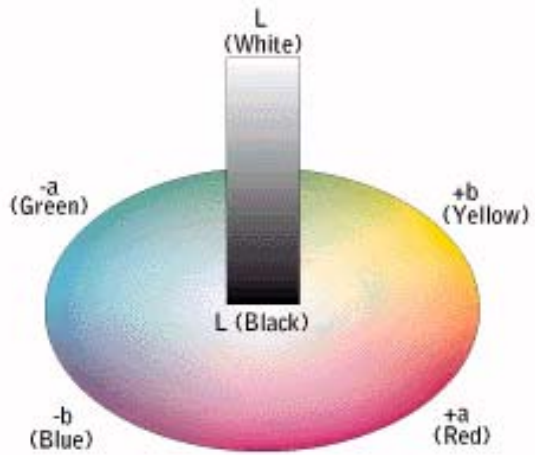
Color vision is complex. While the retina at first registers three color stimuli relating to red, green and blue light rays -- it is not until a further processing stage that three sensations are generated:

a red-green sensation
a yellow-blue sensation
a brightness sensation

These sensations are used to develop a system known as the complementary color system. It is based on the differences of three elementary color pairs: red-green, yellow-blue and black-white.



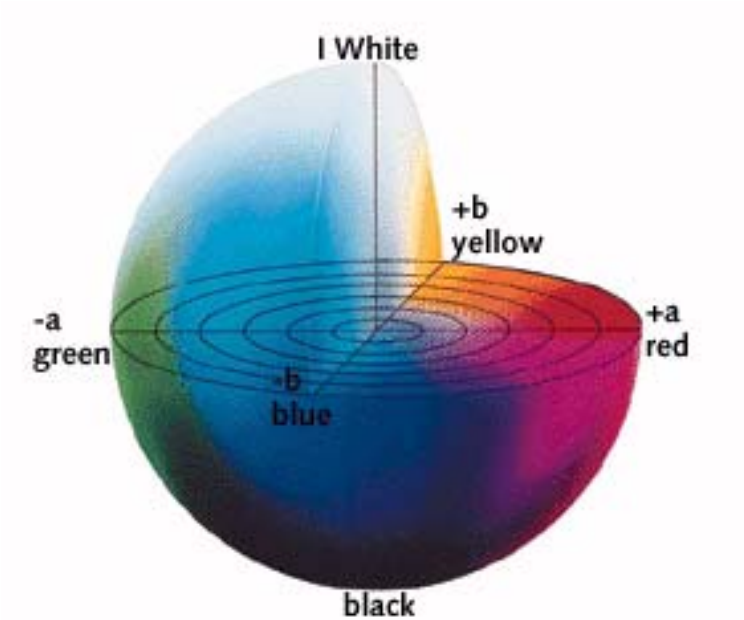




Hue and Chroma

Color Imaging System:

A color is a combination of three primary colors: Red, Green and Blue. A digital color imaging system usually outputs three pixel maps that correspond to the decomposed Red, Green and Blue images. By Red image (respectively, Green and Blue images), we mean that the pixel values of this image indicate the magnitude of the red color component.

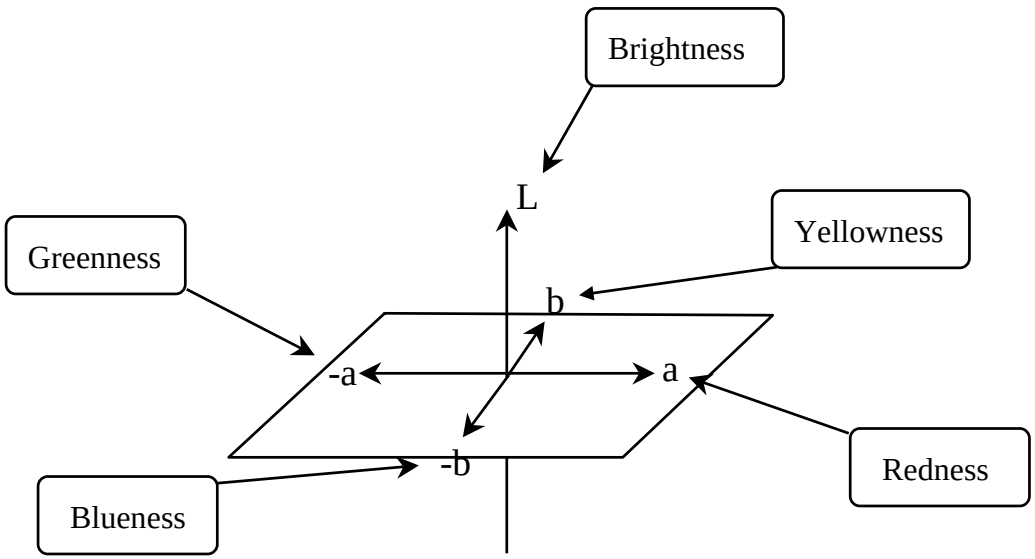


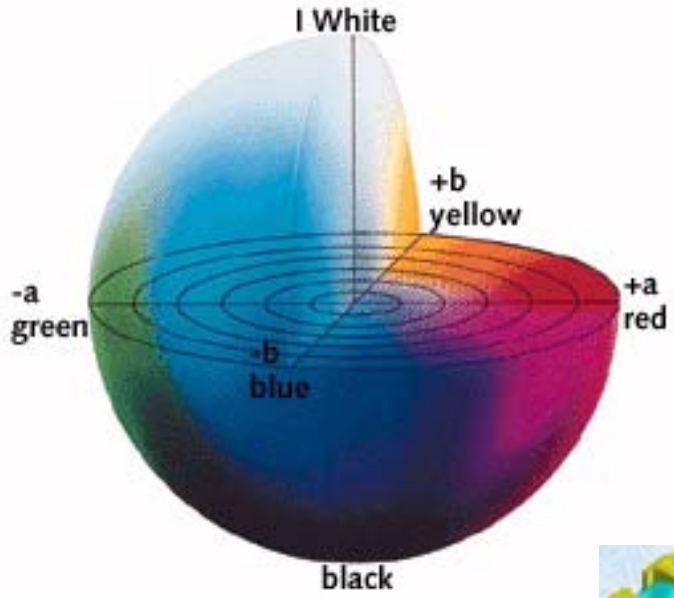
Color Coordinates:

A color can be represented by the (R, G, B) vector. But, this is not a good measurement for color comparison. The more useful color coordinates are (L, a, b) that are defined as follows:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.607 & 0.174 & 0.200 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.066 & 1.116 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{cases} L = 25.0 \bullet (100.0 \bullet Y / Y_{\max})^{1/3} - 16.0 \\ a = 500.0 \bullet [(X / X_{\max})^{1/3} - (Y / Y_{\max})^{1/3}] \\ b = 200.0 \bullet [(Y / Y_{\max})^{1/3} - (Z / Z_{\max})^{1/3}] \end{cases}$$





SUMMARY

1. In order to compare and classify image patterns, one needs to first establish some quantitative or qualitative measurements for the patterns.

2. In the case of image patterns, some measurable characteristics are:

   - Area
   - Perimeter
   - Minimum enclosing rectangle
   - Rectangularity
   - Circularity
   - Moments
   - Color
   (- Texture)

3. If one puts all the measured characteristics into a vector, we obtain a so-called feature vector:

$$F = (A, P, A_r, R, C, \mu_{j,k}, L, a, b)$$