Détection et reconnaissance dynamique de panneaux routiers

Problématique : Comment concevoir un algorithme de détection des panneaux routiers satisfaisant des exigences simples de fiabilité et de rapidité ?

N°candidat : 02240

Thomas MAGNIN

Plan

Introduction/Contextualisation

I) Utilisation des symétries
II) Détection par composante connexe
III) Détection probabiliste
IV) Utilisation dynamique

Conclusion

Introduction - Contexte

Signalisation verticale

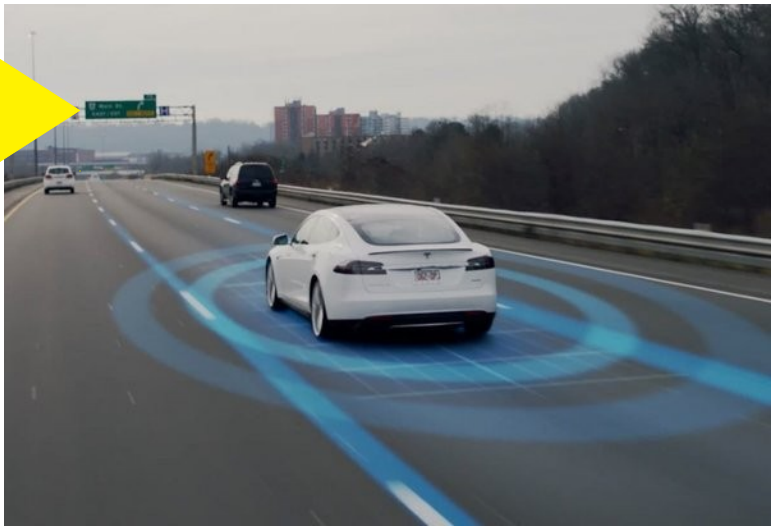
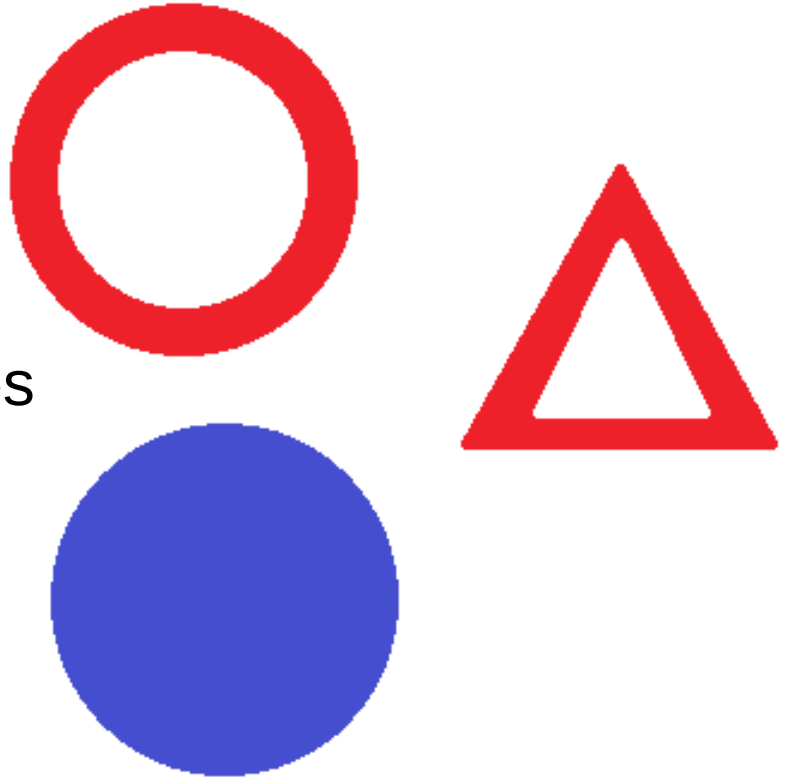


Image © Tesla Motors

<u>Restriction de l’étude :</u>

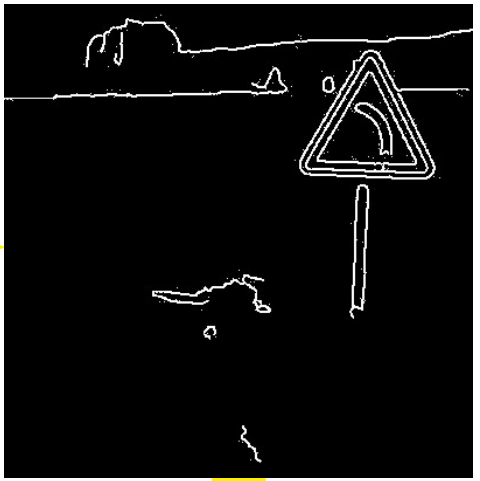


- Panneaux rouges
- Circulaires/Rectangulaires/Triangulaires
- Conditions de luminosité acceptables

Données utilisées

Images fournies par une caméra embarquée

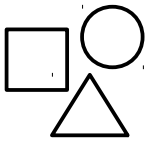
Détection de contour



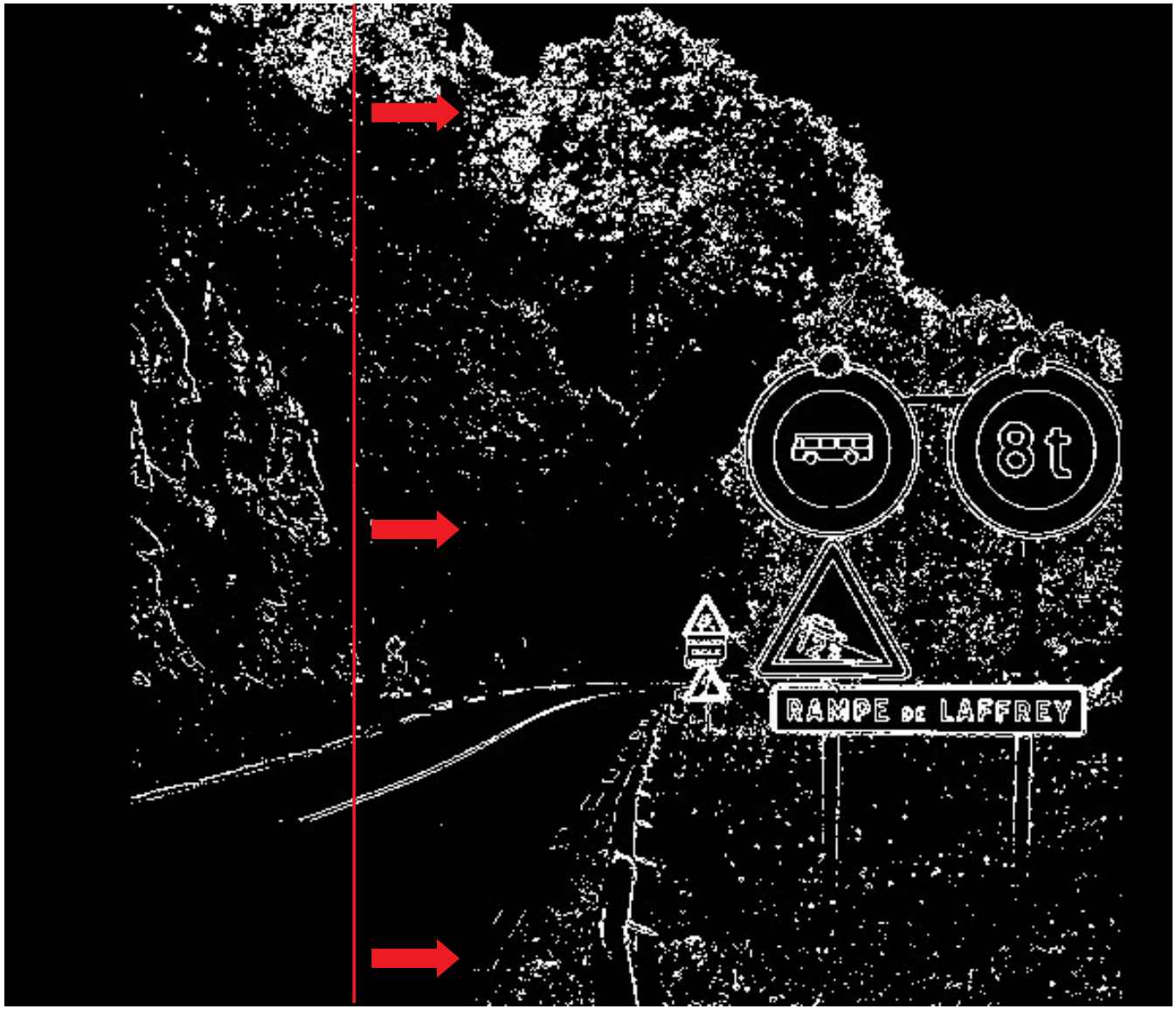
Filtre de couleur

Données colorimétriques

Données géométriques

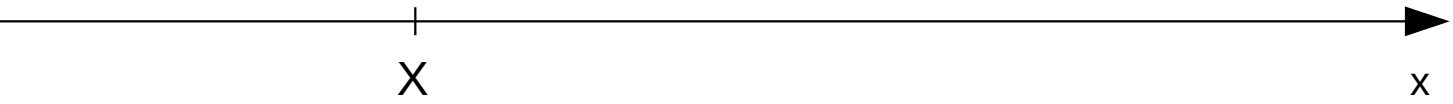


I) Utilisation des symétries

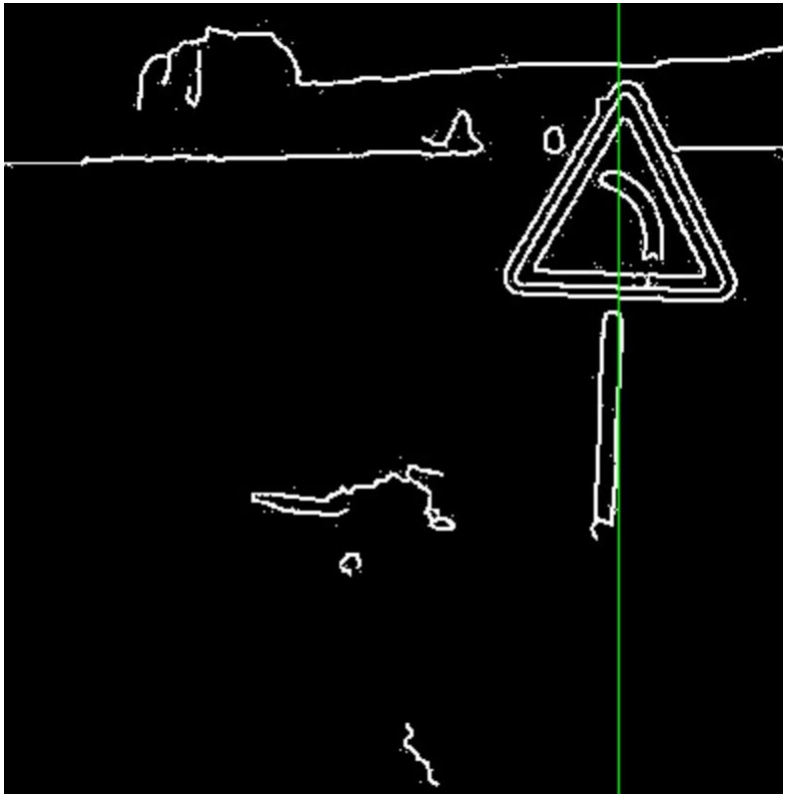



```
Pour X allant de 0 à L :
    lim = min(L-X,X)
    Pour x allant de X+Δ à lim :
        Pour y allant de 0 à H :
            Si (x,y) et (2X-x,y) sont blancs :
                ajouterVote(X)
Xmax = max(votes)
DessinerAxe(Xmax)
```

X	Votes
1	1
2	10
⋮	⋮
159	208
⋮	⋮



Premiers résultats

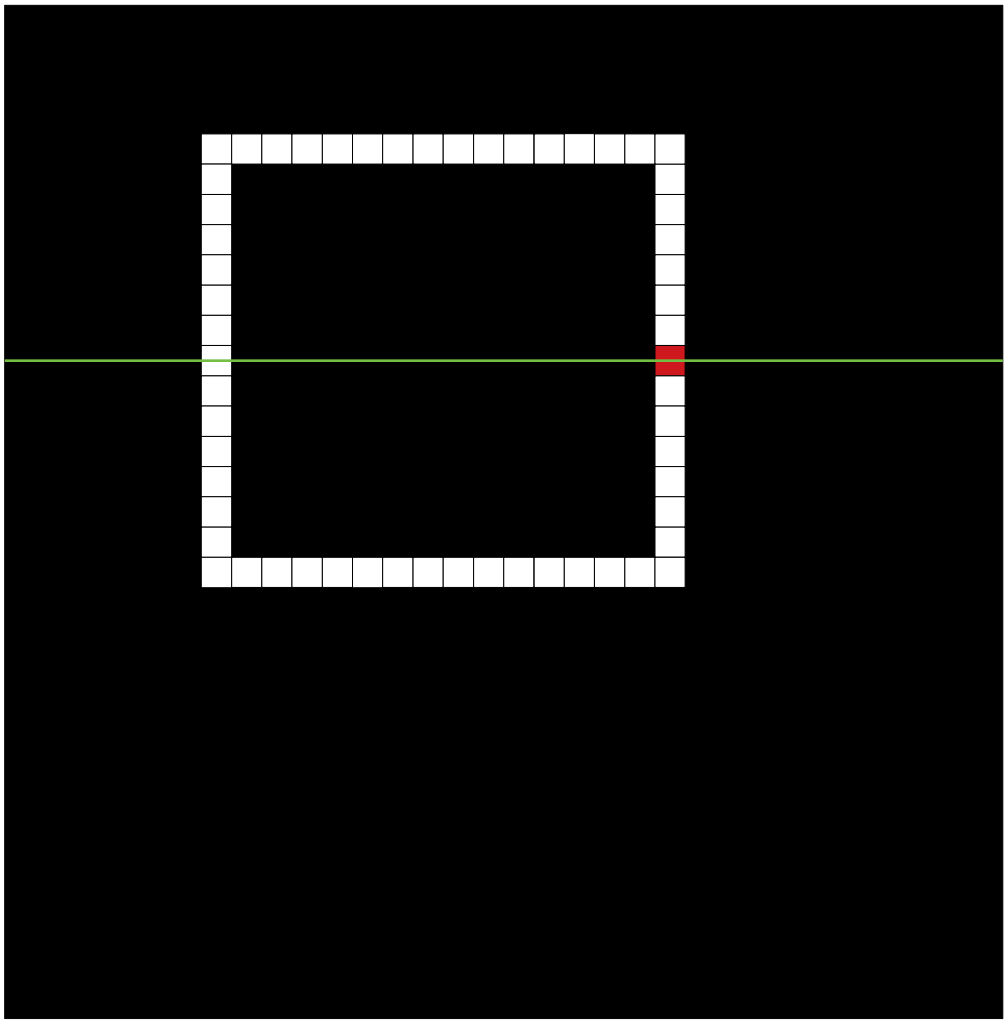


<u>Principaux problèmes :</u>

- Temps d’exécution ≈ 20 secondes
- Détection selon un seul axe
- Imprécision lorsqu’il y a plusieurs panneaux

Améliorations

Recherche de symétrie pixel par pixel



Pour chaque pixel blanc :
- recherche d'un symétrique
- vote pour l'axe médian

Axe de symétrie donné par le maximum des votes

<u>Avantage :</u>

- On ne parcourt qu'une seul fois chaque pixel

<u>Problèmes :</u>

- Pas de différenciation entre les panneaux
- Encore lent ≈ 2 s

II) Détection par composante connexe

1ère étape :

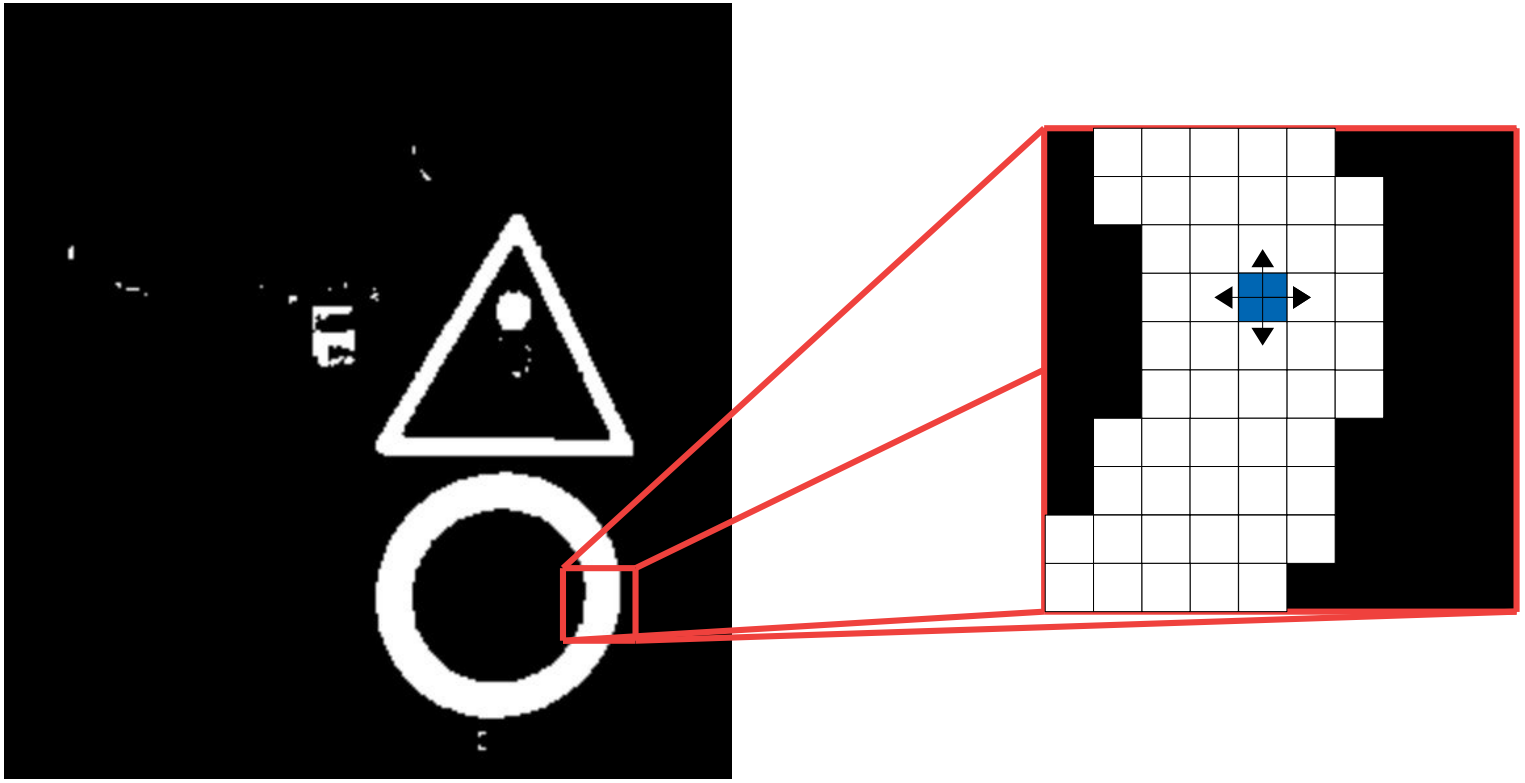
Application d'un filtre colorimétrique :

Sélection des pixels blancs
pxBlancs = [[5,1], ...]

Composante connexe

2ème étape :

Détermination de la composante connexe :



```
25def composanteConnexe(point, pixels,L,H):
26    composante = [point]
27    pile = [point]
28    while len(pile) > 0 :
29        px = pile.pop()
30        v = voisins(px,L,H)
31        for el in v :
32            if el not in composante and pixels[el[0],el[1]] == (255,255,255) :
33                composante.append(el)
34                pile.append(el)
35    return composante
```

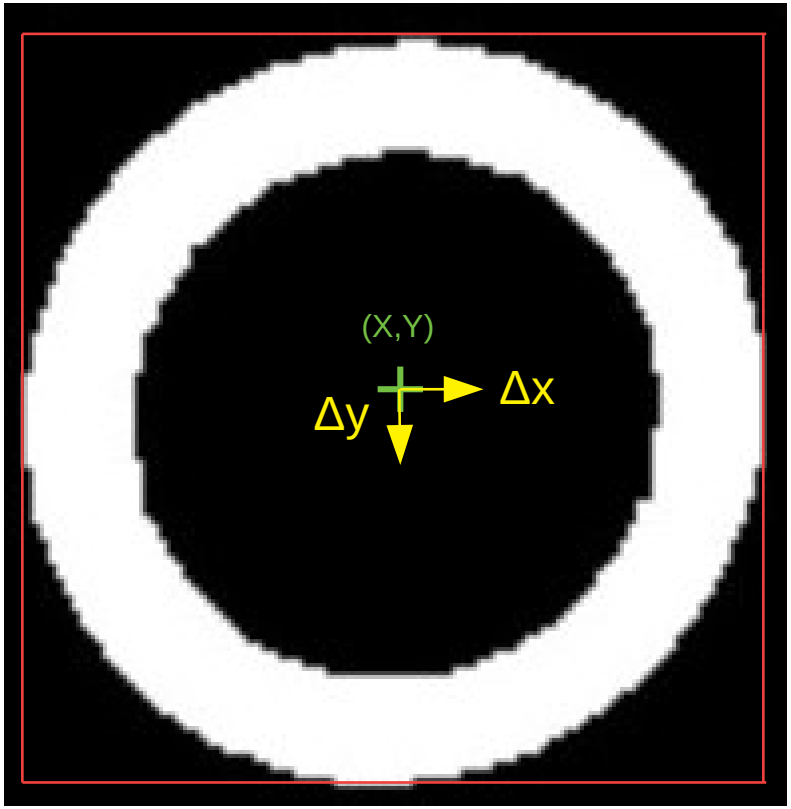
Utilisation des composantes connexes

Recherche de panneaux circulaires :

Application de la transformée de Hough à chaque composante connexe

$$(x-a)^2+(y-b)^2=R^2$$

(0,0)



(L,H)

$$(a,b) \in [X-\Delta x, X+\Delta x] \times [Y-\Delta Y, Y+\Delta Y]$$

$$R=\sqrt{(x-a)^2+(y-b)^2}$$

Accumulateur
(dictionnaire python)

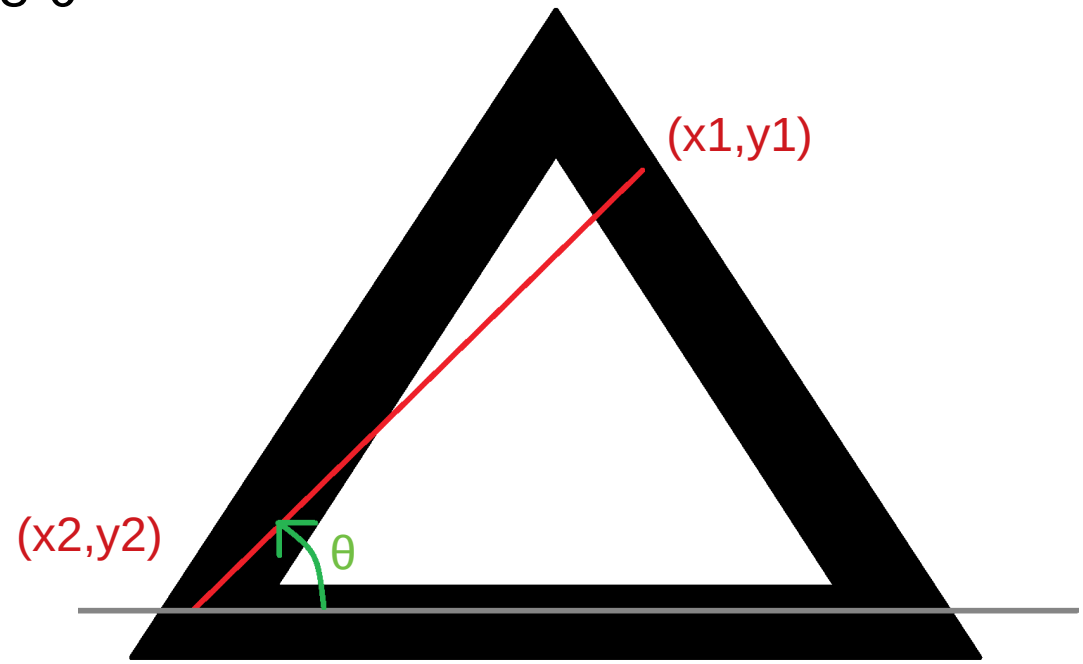
Cercle	Vote
(140,130,70)	5
(40,10,20)	100
⋮	⋮

Résultats

III) Détection probabiliste

Expérience aléatoire :

- Tirer un couple de pixels au hasard dans la composante connexe
- Calculer l'angle θ



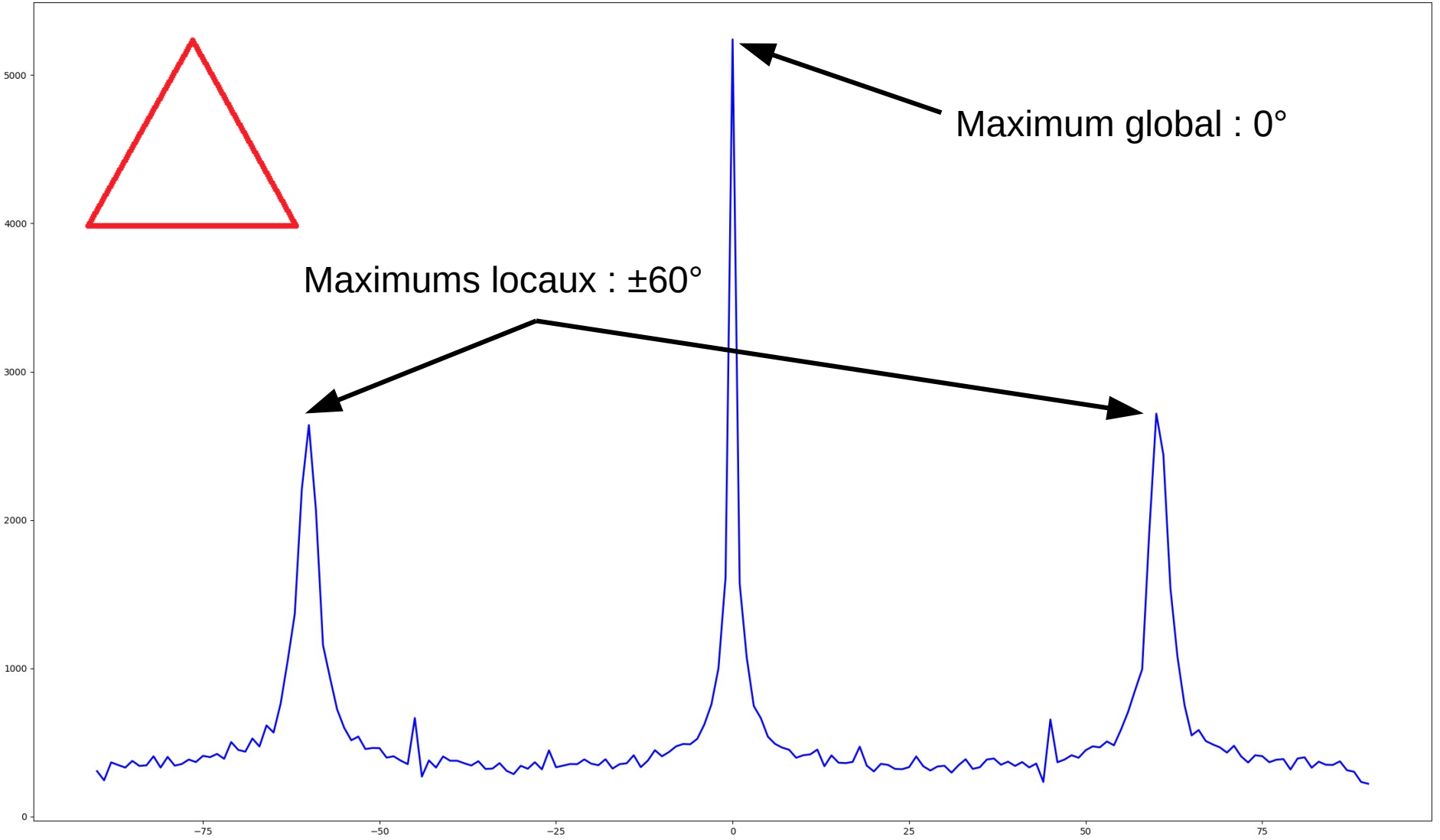
- Ajouter un vote pour θ

On réalise l'expérience n fois

Exploitation des résultats

Cas du triangle

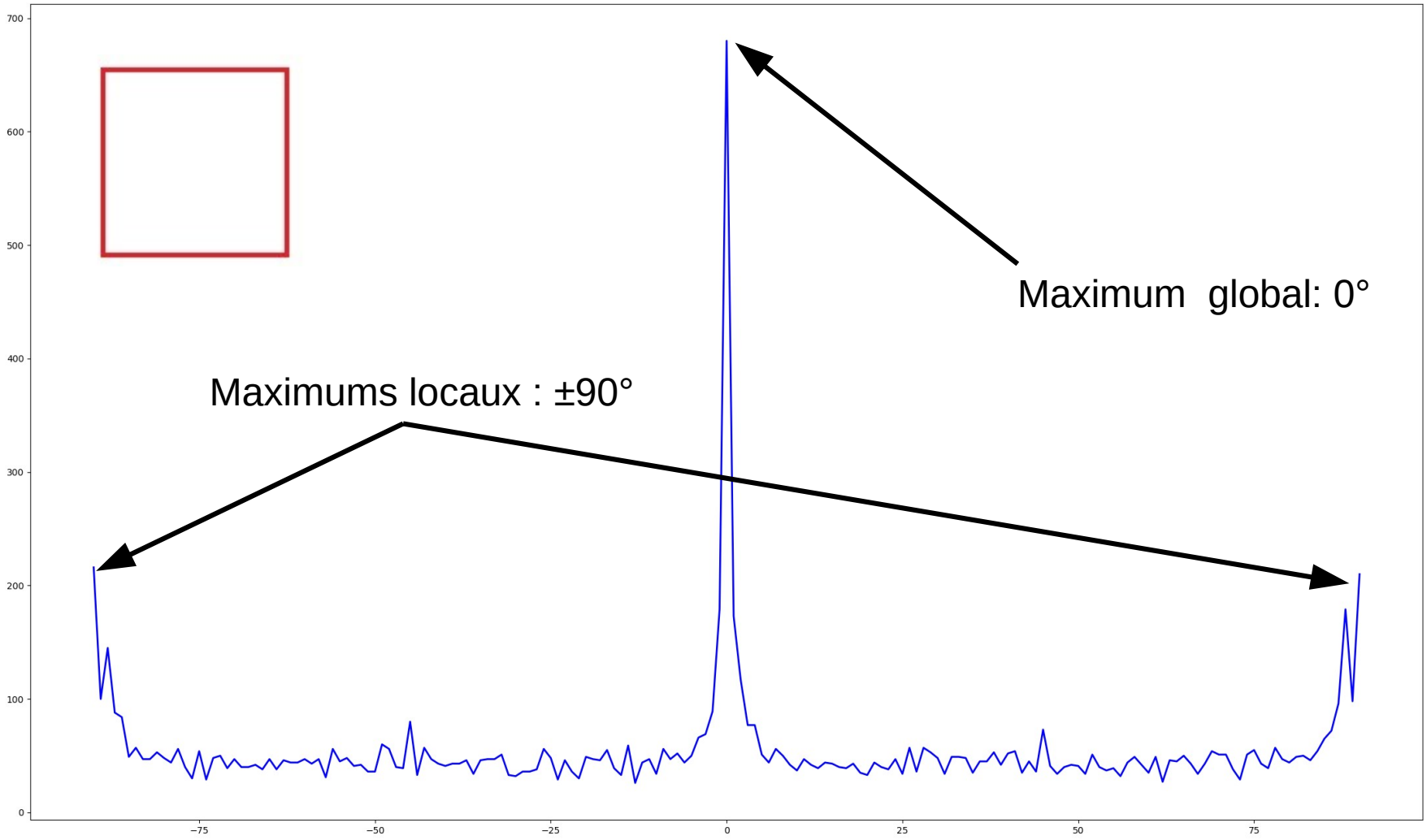
Résultats : n = 10 000
Temps exécution : ≈ 2s



Exploitation des résultats

Cas du carré

Résultats : n = 10 000
Temps exécution : ≈ 2s



Résultats pratiques


```
t triangulaires)
te connexe (panneaux circulaires)


cle[1][2], cercle[1][1]-cercle[1][2],\
cle[1][2], cercle[1][1]+cercle[1][2]),\

b("+str(r)+","+str(g)+","+str(b)+")")




position avec
```

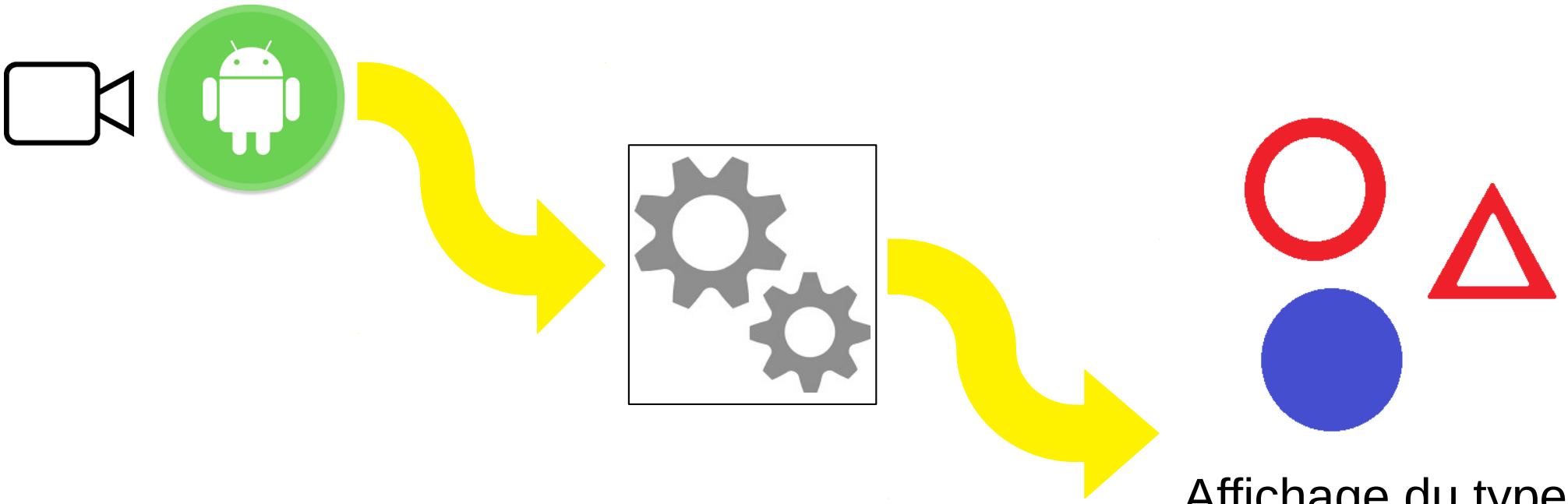
```
497
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0

In [19]: runfile('C:/Users/tthoo/
Desktop/TIPE/programmeFinal.py',
wdir='C:/Users/tthoo/Desktop/
TIPE')
Danger
Danger
Interdiction/restriction
Danger
Interdiction/restriction
```

IV) Fonctionnement dynamique

Principe : utiliser l'algorithme précédent en continu, sur un flux vidéo

Acquisition d'un flux vidéo par la caméra d'un smartphone Android

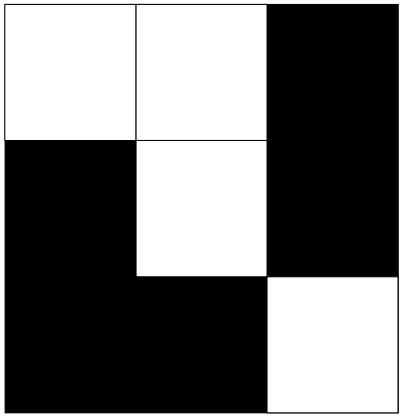


Traitement d'une image
en 0,04s maximum

Affichage du type
de panneau détecté

Optimisation de la fonction *composanteConnexe(point)* :

Représentation de l'image binaire par un tableau à 2 dimensions de booléens :



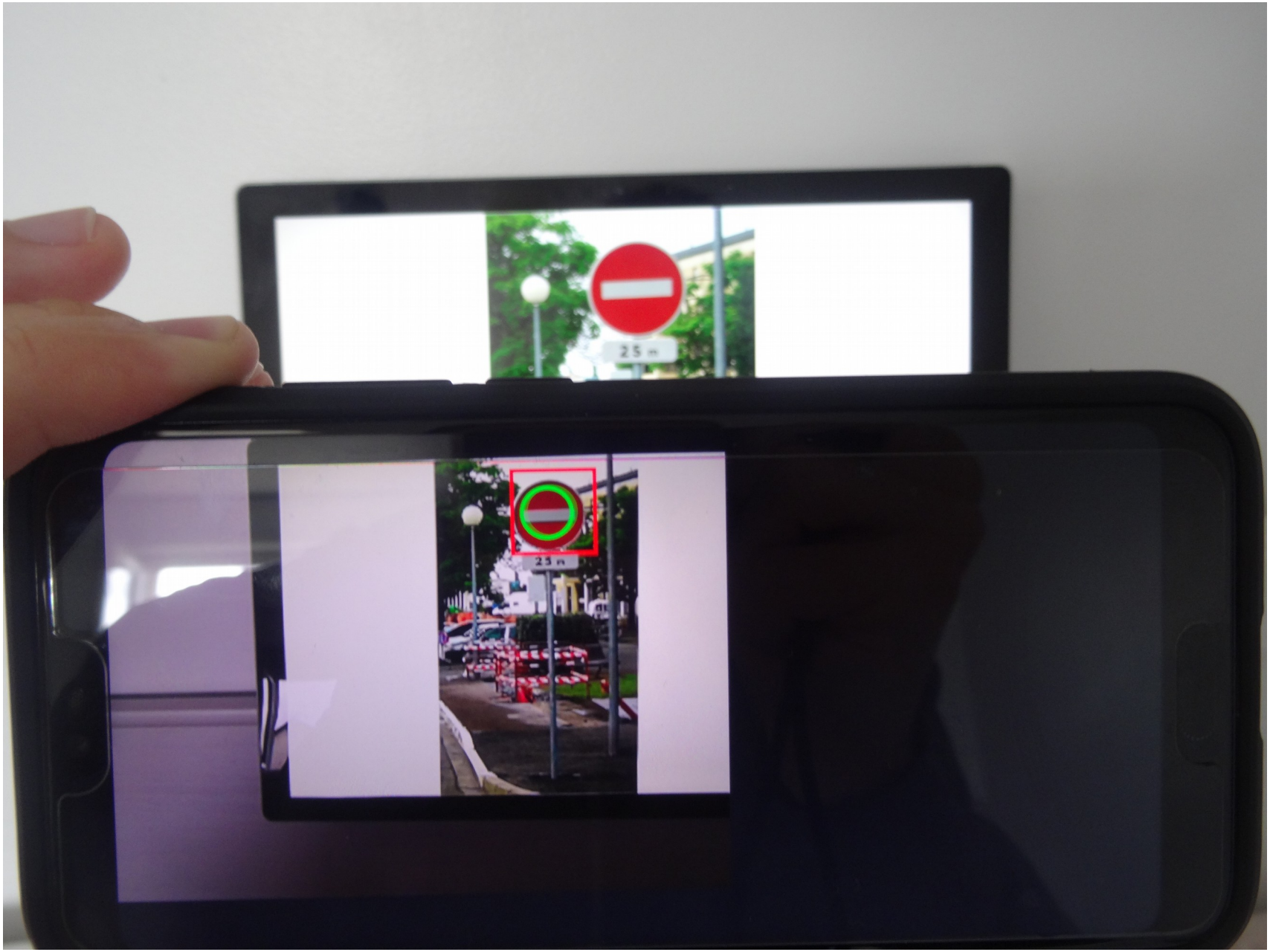
Passage à *False* des pixels visités

True	True	False
False	True	False
False	False	True

→ Plus besoin de parcourir composanteConnexe[]

```
protected ArrayList<Point> composanteConnexe(boolean[][] matrice, Point point){
    ArrayList<Point> composante = new ArrayList<>();
    ArrayList<Point> pile = new ArrayList<>();
    composante.add(point);
    pile.add(point);
    while(pile.size() > 0){
        Point px = pile.remove( index: pile.size()-1);
        ArrayList<Point> voisins = voisins(px,L,H);
        for(int i =0;i<voisins.size();i++){
            Point el = voisins.get(i);
            if(matrice[el.x][el.y]){
                composante.add(el);
                pile.add(el);
                matrice[el.x][el.y] = false;
            }
        }
    }
    return composante;
}
```

Résultats dynamiques



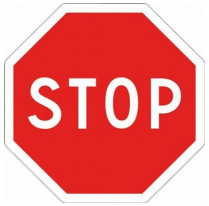
Conclusion

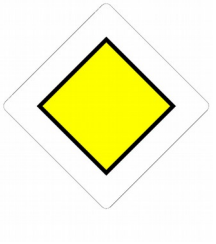
- Possibilité de créer un algorithme de détection efficace

Améliorations :

- Reconnaissance de la signification précise du panneau

- Détection d’autres types de panneaux





Annexes : Algorithme symétrie

```
from PIL import Image
import time

debut = time.time()

img = Image.open('contour.jpg')
pixels = img.load()

#Dimensions de l'image

L = img.size[0]
H = img.size[1]

#Abscisse de l'axe de symétrie variable
X = 0
pas = 1

#Création d'un accumulateur pour les votes
votes = [0]*L

# l'axe de symétrie parcourt l'image de gauche à droite (en restant vertical)
for X in range(0,L,pas) :
    Limite = min(L-X,X)
    for x in range(X+10,X+Limite):
        for y in range(0,H) :
            r,g,b = img.getpixel((x,y))
            if r == 255 and g == 255 and b == 255 :
                #On parcourt tous les pixels BLANCS à droite de l'axe de symétrie
                r,g,b =img.getpixel((2*X-x,y))
                if r==255 and g == 255 and b == 255 :
                    #Si le symétrique du pixel p/r à l'axe est aussi blanc
                    #on ajoute un vote
                    votes[X]+=1

Xmax = max(votes)
print(Xmax)
for X in range(len(votes)) :
    if votes[X] == Xmax :
        for y in range(0,H):
            pixels[X,y] = (0,255,0)


img.show()
```

Annexes : Algorithme symétrie 2

```
from PIL import Image
import operator

img = Image.open('img.jpg')
pixels = img.load()

#Dimensions de l'image

L = img.size[0]
H = img.size[1]

pxBlancs = []
vote = dict()


for x in range(L):
    for y in range(H):
        r,g,b = img.getpixel((x,y))
        if r == 255 and g == 255 and b == 255 :
            pxBlancs.append((x,y))


for el in pxBlancs :
    for x in range(0,100) :
        try :
            r,g,b = img.getpixel((el[0]+x-50,el[1]))
            if r == 255 and g == 255 and b == 255 :
                try :
                    X = el[0]+(x-50)/2 #Abscisse du pixel central
                    Y = el[1]
                    vote[X][0]+=1 #On ajoute un vote
                    if  Y > vote[X][2] :
                        vote[X][2] = Y
                    elif Y < vote[X][1] :
                        vote[X][1] = Y
```

Annexes : Algorithme symétrie 2

```
                #S'il n'y avait aucun vote pour l'axe X on créé une paire
                #clé/(vote, min ord, max ord) dans le dictionnaire égale
                #à (1 vote, y,y) où y est l'abscisse du pixel central
                except KeyError :
                    vote[el[0]+(x-50)/2] = [1,el[1], el[1]]
        #Si le pixel testé est en dehors de l'image
        except IndexError :
            continue


#On cherche l'axe de symétrie qui a le maximum de votes dans le dictionnaire
Xmax = max(vote.items(), key=operator.itemgetter(1))[0]
for y in range(0,H):
    pixels[Xmax,y] = (0,255,0)

img.show()
```

Algorithme composante connexe 1/8

```
from PIL import Image, ImageDraw
import math
import random
import collections


def appartient(point, L,H):
    """Retourne True si le point appartient au
    rectangle délimité par les points (0,0) et (L,H)"""
    if point[0]>=0 and point[0]<L and point[1]>=0 and point[1]<H:
        return True
    else :
        return False

def voisins(point, L, H):
    """Renvoi les pixels voisins de (x,y) situés dans l'image de taille LxH"""
    x, y = point[0], point[1]
    voisins = []
    for i in range(0,3) :
        for j in range(0,3) :
            if abs(j-1) == abs(i-1) :
                pass
            else :
                v = [x+i-1,y+j-1]
                if appartient(v,L,H) and v not in voisins:
                    voisins.append(v)
    return voisins
```

Application python 2/8

```
def composanteConnexe(point, pixels,L,H):
    """Renvoie la composante connexe associée à point"""
    composante = [point]
    pile = [point]
    while len(pile) > 0 :
        px = pile.pop()
        v = voisins(px,L,H)
        for el in v :
            if el not in composante and pixels[el[0],el[1]] == (255,255,255) :
                composante.append(el)
                pile.append(el)
    return composante

def boiteEnglobante(composanteConnexe) :
    """Renvoie la boite englobante de composanteConnexe"""
    x1,x2,y1,y2 = L,0,H,0
    for px in composanteConnexe :
        if px[0] < x1 :
            x1 = px[0]
        if px[0] > x2 :
            x2 = px[0]
        if  px[1] < y1 :
            y1 = px[1]
        if px[1] > y2 :
            y2 = px[1]
    return [x1,y1,x2,y2] #coordonées du pixels en haut à gauche et en bas à droite
```

Application python 3/8

```
def TSV(r,v,b):
    """Convertit une couleur R,V,B en T,S,V"""
    M = max(r,v,b)
    m = min(r,v,b)
    if M == m :
        t = 0
    elif M == r:
        t = (60*(v-b)/(M-m)+360)%360
    elif M == v:
        t= (60*(b-r)/(M-m)+120)
    elif M == b :
        t = (60*(r-v)/(M-m) +240)
    if M == 0:
        s = 0
    else:
        s = 1-m/M
    return (t,s,M)

def maxAccumulateur(accumulateur) :
    """Recherche le maximum dans l'accumulateur (transformée de hough)"""
    M = 0
    cle = (0,0,0)
    for i in accumulateur.keys():
        if accumulateur[i] > M :
            M = accumulateur[i]
            cle = i
    return M,cle
```

Application python 4/8

```
def Hough(pixels, boite) :
    """Effectue la transformée de Hough pour les cercles dans les
    limite de la boite englobante"""
    accumulateur = dict()
    centre = (int((boite[0]+boite[2])/2),int((boite[1]+boite[3])/2))
    deltaX = int(5/100 * (boite[2]-boite[0]))
    deltaY = int(5/100 * (boite[3]-boite[1]))

    for x in range(boite[0], boite [2]) :
        for y in range(boite[1],boite[3]):
            if pixels[x,y] != (255,255,255) :
                continue
            for a in range(centre[0]-deltaX, centre[0]+deltaX) :
                for b in range(centre[1]-deltaY, centre[1]+deltaY):
                    R = int(math.sqrt((x-a)**2 + (y-b)**2))
                    try :
                        accumulateur[(a,b,R)]+=1
                    except KeyError :
                        accumulateur[(a,b,R)] = 1
    cercle = maxAccumulateur(accumulateur)
    return cercle
```

Application python 5/8

```
def Probabilite(composante) :
    """Effecture une detection probabiliste sur la composante
    connexe 'composante' """
    nbrDeTirage = 10000
    angles = dict()

    def ajouterVote(angle):
        try :
            angles[angle]+=1
        except Exception :
            angles[angle] = 1
    for i in range(nbrDeTirage) :
        pixel1 = composante[random.randint(0,len(composante)-1)]
        pixel2 = composante[random.randint(0,len(composante)-1)]
        x1,y1,x2,y2 = pixel1[0], pixel1[1],pixel2[0],pixel2[1]
        if x1!=x2 :
            ajouterVote(int(math.atan((y2-y1)/(x2-x1))*180/math.pi))
        else :
            if y1-y2 > 0 :
                ajouterVote(90)
            else :
                ajouterVote(-90)
    x = []
    y = []
```

Application python 6/8

```
angles = collections.OrderedDict(sorted(angles.items(),key=lambda t: t[1],\
                                        reverse = True))
q = 0
M1 = 0
M2 = 0
M3 = 0
for k,v in angles.items() :
    if q==0:
        M1= k
    elif q == 1:
        M2 = k
    elif q == 2 :
        M3 = k
    x.append(k)
    y.append(v)
    q+=1
if abs(M1)<=3 and abs(abs(M2)-60) <= 3 and abs(abs(M3)-60)<=3 and M2*M3<0 :
    print("Danger")
elif abs(M1) <= 3 and abs(abs(M2)-90) <=3 and abs(abs(M3)-90) <=3 and M2*M3 <0 :
    print("Indication")
```

Application python 7/8

```
img = Image.open('img.jpg')
contour = img.copy()
imageFinale = img.copy()
pixels = contour.load()

dessin = ImageDraw.Draw(contour)
dessin2 = ImageDraw.Draw(imageFinale)

L = contour.size[0]
H = contour.size[1]

pxBlancs = []

#Détection des pixels rouges dans l'image (contour)
for x in range(L):
    for y in range(H):
        r,g,b = contour.getpixel((x,y))
        couleur = TSV(r,g,b)
        if((couleur[0] < 30 or couleur[0] > 330 ) and couleur[1] >0.4 and couleur[2] > 0.4):
            pixels[x,y] = (255,255,255)
            pxBlancs.append([x,y])
        else :
            pixels[x,y] = (0,0,0)

#Détection des composantes connexes
composantesConnexes = []
for el in pxBlancs :
    c = composanteConnexe(el,pixels,L,H)
    r,g,b = 255,255,0
    boite = boiteEnglobante(c)
    #Détection probabiliste sur la composante connexe (panneaux carrés et triangulaires)
    Probabilite(c)
    #Transformée de hough sur la composante connexe (panneaux circulaires)
    cercle = Hough(pixels,boite)
```

Application python 8/8

```
    if(cercle[0] > 80):
        dessin2.ellipse((cercle[1][0]-cercle[1][2], cercle[1][1]-cercle[1][2],\
                         cercle[1][0]+cercle[1][2], cercle[1][1]+cercle[1][2]),\
        outline = "rgb(0,255,0)")
    dessin2.rectangle(boite, outline ="rgb("+str(r)+","+str(g)+","+str(b)+")")
    for px in c :
        if px in pxBlancs :
            pxBlancs.remove(px)
    composantesConnexes.append(c)

#Affichage des panneaux détectés en superposition avec
#l'image initiale
imageFinale.show()
```

Application Android (Java)

```java
public class MainActivity extends Activity {

    protected Camera camera = null;
    protected Affichage affichage;
    protected SurfaceView surface;
    protected SurfaceView dessin;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        surface = (SurfaceView)findViewById(R.id.surface);
        dessin = (SurfaceView)findViewById(R.id.dessin);
        dessin.setZOrderOnTop(true);
        SurfaceHolder holder = dessin.getHolder();
        holder.setFormat(PixelFormat.TRANSPARENT);
    }

    @Override
    protected void onResume() {
        super.onResume();
        ouvrirCamera();
        affichage = new Affichage(this, surface,dessin, camera);
    }

    private void ouvrirCamera() {
        fermerCamera();
        try {
            camera = Camera.open();
            Camera.Parameters parametres = camera.getParameters();
            parametres.setPreviewSize(Constantes.width,Constantes.height);
            parametres.setPreviewFpsRange(12000,15000);
            parametres.setFocusMode(Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE);
            List<Integer> format = parametres.getSupportedPreviewFormats();
            camera.setParameters(parametres);
        }catch(Exception e){
            Log.e("Caméra","Echec de l'ouverture de la camera");
        }
    }

    private void fermerCamera(){
        if(camera != null){
            affichage.stopAffichage();
            camera.release();
            camera = null;
        }
    }
}
```

```java
package com.example.tipe;

import android.graphics.Color;
import android.graphics.Point;
import android.graphics.Rect;
import java.util.ArrayList;
import java.util.Random;

public class Reconnaissance {

    protected int[] img;
    protected int L;
    protected int H;
    protected boolean [][] matrice;
    protected boolean [][] copie;
    protected ArrayList<Rect> boitesEnglobantes;

    public Reconnaissance(int[] _img){
        img = _img;
        L = Constantes.width;
        H = Constantes.height;
        matrice = new boolean[L][H];
        copie = new boolean[L][H];
        boitesEnglobantes = new ArrayList<>();
        for (int x = 0; x < L; x++) {
            for (int y = 0; y < H; y++) {
                int couleur = getPixel(x,y);
                int r = Color.red(couleur);
                int v = Color.green(couleur);
                int b = Color.blue(couleur);
                float[] TSV = new float[3];
                Color.RGBToHSV(r, v, b, TSV);
                if ((TSV[0] < 30 || TSV[0] > 330) && TSV[1] > 0.4 && TSV[2] > 0.4) {
                    matrice[x][y] = true;
                    copie[x][y] = true;
                }
            }
        }
    }
```

```
public ArrayList<Rect> detect() {

    for (int x=0; x < copie.length; x++) {
        for(int y = 0;y<copie[0].length;y++){
            if(copie[x][y]) {
                ArrayList<Point> c = composanteConnexe(copie,new Point(x, y));
                Rect b = boiteEnglobante(c);
                boitesEnglobantes.add(boiteEnglobante(c));
            }
        }
    }
    return boitesEnglobantes;
}

public int getPixel(int x, int y){
    int p = img[y*Constantes.width+x];
    int r = (p & 0xff0000) >> 16;
    int v = (p & 0xff00) >> 8;
    int b = p & 0xff;
    return Color.rgb(r,v,b);
}

protected boolean appartient(Point point, int L, int H){
    if (point.x >= 0 && point.x < L && point.y>=0 && point.y < H)
        return true;
    else
        return false;
}

protected ArrayList<Point> voisins(Point point, int L, int H){
    int x = point.x;
    int y = point.y;
    ArrayList<Point> voisins = new ArrayList<>();
    for(int i =-1;i<=1;i++){
        for(int j=-1;j<=1;j++){
            if(Math.abs(i) == Math.abs(j)){
                continue;
            }
            else{
                Point v = new Point(x+i,y+j);
                if(appartient(v,L,H)) {
                    voisins.add(v);
                }
            }
        }
    }
    return voisins;
}
```

```
protected ArrayList<Point> composanteConnexe(boolean[][] mat,Point point){
    ArrayList<Point> composante = new ArrayList<>();
    ArrayList<Point> pile = new ArrayList<>();
    composante.add(point);
    pile.add(point);
    while(pile.size() > 0){
        Point px = pile.remove(pile.size()-1);
        ArrayList<Point> voisins = voisins(px,L,H);
        for(int i =0;i<voisins.size();i++){
            Point el = voisins.get(i);
            if(mat[el.x][el.y]){
                composante.add(el);
                pile.add(el);
                mat[el.x][el.y] = false;
            }
        }
    }
    return composante;
}

protected Rect boiteEnglobante(ArrayList<Point> composanteConnexe){
    int x1=L,x2 =0,y1=H,y2 = 0;
    for(int i = 0;i<composanteConnexe.size();i++){
        Point px = composanteConnexe.get(i);
        if(px.x<x1)
            x1 = px.x;
        if(px.x>x2)
            x2 = px.x;
        if(px.y < y1)
            y1=px.y;
        if(px.y>y2)
            y2=px.y;
    }

    if(x1-5 >0)
        x1-=5;
    if(x2+5 < L)
        x2+=5;
    if(y1-5 > 0)
        y1-=5;
    if(y2+5 < H)
        y2+=5;
    return new Rect(x1,y1,x2,y2);
}
```

```
protected int max(int[] tab){
    int M = tab[0];
    for(int i=1;i<tab.length;i++){
        if(tab[i] > M)
            M = tab[i];
    }
    return M;
}

protected int min(int[] tab){
    int m = tab[0];
    for(int i=1;i<tab.length;i++){
        if(tab[i] < m)
            m = tab[i];
    }
    return m;
}

public int[] hough(Rect boite){
    Accumulateur accumulateur = new Accumulateur();
    Point centre = new Point(boite.centerX(),boite.centerY());
    int deltaX = (int)((float)10/100*(boite.right-boite.left));
    int deltaY = (int)((float)10/100*(boite.bottom- boite.top));
    Random random = new Random();
    for(int x=boite.left;x<boite.right;x++){
        for(int y=boite.top;y<boite.bottom;y++) {
            if(random.nextInt(10) == 1 && matrice[x][y]) {
                for (int a = centre.x - deltaX; a < centre.x + deltaX; a++) {
                    for (int b = centre.y - deltaY; b < centre.y + deltaY; b++) {
                        int R = (int) Math.sqrt((x - a) * (x - a) + (y - b) * (y - b));
                        accumulateur.ajouterVote(new int[]{a, b, R});
                    }
                }
            }
        }
    }
    return accumulateur.cercle();
}
```

```java
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.PorterDuff;
import android.graphics.Rect;
import android.hardware.Camera;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.ViewGroup;
import java.io.IOException;
import java.util.ArrayList;

public class Affichage extends ViewGroup implements SurfaceHolder.Callback, Camera.PreviewCallback {

    protected SurfaceView surface;
    protected SurfaceHolder dessin;
    protected SurfaceView surfaceDessin;
    protected SurfaceHolder surfaceHolderDessin;
    protected Camera camera;
    protected Context context;
    boolean affichageOn = false;

    public Affichage(Context _context, SurfaceView _surface, SurfaceView _surfaceDessin, Camera _camera){
        super(_context);
        context = _context;
        camera = _camera;
        surface = _surface;
        surfaceDessin = _surfaceDessin;
        surfaceHolderDessin = surfaceDessin.getHolder();
        dessin = surface.getHolder();
        dessin.addCallback(this);
        surfaceHolderDessin.addCallback(this);
    }

    public void stopAffichage(){
        if(camera != null && affichageOn){
            camera.stopPreview();
            affichageOn = false;
        }
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        if(holder == dessin) {
            if (camera != null && !affichageOn) {
                try {
                    camera.setPreviewDisplay(dessin);
                } catch (IOException e) {
                    e.printStackTrace();
                }
                camera.setPreviewCallback(this);
                camera.startPreview();
                affichageOn = true;
            }
        }
    }


    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
```

```
@Override
protected void onLayout(boolean changed, int l, int t, int r, int b) {

}

@Override
public void onPreviewFrame(byte[] data, Camera camera) {
    int[] rgb = convertYUV420_NV21toRGB8888(data, Constantes.width,Constantes.height);

    Reconnaissance reco = new Reconnaissance(rgb);
    ArrayList<Rect> boites = reco.detect();

    if(surfaceHolderDessin != null){
        Canvas canvas = surfaceHolderDessin.lockCanvas();
        canvas.drawColor(0, PorterDuff.Mode.CLEAR );
        Paint paint = new Paint();
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeWidth(8);

        for(int i = 0;i<boites.size();i++){
            Rect b = boites.get(i);
            if((b.right-b.left)*(b.bottom-b.top)>30*30) {
                int[] cercle = reco.hough(b);
                paint.setColor(Color.RED);
                canvas.drawRect(b.left * 4, b.top * 4, b.right * 4, b.bottom * 4, paint);
                paint.setColor(Color.GREEN);
                if (cercle != null)
                    canvas.drawCircle(cercle[0] * 4, cercle[1] * 4, cercle[2] * 4, paint);
            }
        }
        surfaceHolderDessin.unlockCanvasAndPost(canvas);
    }

}

public int getPixel(int[] img, int x, int y){
    int p = img[y*Constantes.width+x];
    int r = (p & 0xff0000) >> 16;
    int v = (p & 0xff00) >> 8;
    int b = p & 0xff;

    return Color.rgb(r,v,b);
}

public static int[] convertYUV420_NV21toRGB8888(byte [] data, int width, int height) {
    int size = width*height;
    int offset = size;
    int[] pixels = new int[size];
    int u, v, y1, y2, y3, y4;

    // i percorre os Y and the final pixels
    // k percorre os pixles U e V
    for(int i=0, k=0; i < size; i+=2, k+=2) {
        y1 = data[i  ]&0xff;
        y2 = data[i+1]&0xff;
        y3 = data[width+i  ]&0xff;
        y4 = data[width+i+1]&0xff;

        u = data[offset+k  ]&0xff;
        v = data[offset+k+1]&0xff;
```

```
            u = u-128;
            v = v-128;

            pixels[i] = convertYUVtoRGB(y1, u, v);
            pixels[i+1] = convertYUVtoRGB(y2, u, v);
            pixels[width+i  ] = convertYUVtoRGB(y3, u, v);
            pixels[width+i+1] = convertYUVtoRGB(y4, u, v);

            if (i!=0 && (i+2)%width==0)
                i+=width;
        }

        return pixels;
    }

    private static int convertYUVtoRGB(int y, int u, int v) {
        int r,g,b;

        r = y + (int)(1.402f*v);
        g = y - (int)(0.344f*u +0.714f*v);
        b = y + (int)(1.772f*u);
        r = r>255? 255 : r<0 ? 0 : r;
        g = g>255? 255 : g<0 ? 0 : g;
        b = b>255? 255 : b<0 ? 0 : b;
        return 0xff000000 | (b<<16) | (g<<8) | r;
    }


}
```

```java
import java.util.ArrayList;

public class Accumulateur {

    private ArrayList<int[]> cles = new ArrayList<>();
    private ArrayList<Integer> objets = new ArrayList<>();

    public void add(int[] cle, int objet) {
        cles.add(cle);
        objets.add(objet);
    }

    public int get(int[] cle) {
        for(int i = 0;i<cles.size();i++) {
            if(egaux(cles.get(i),cle)) {
                return objets.get(i);
            }
        }
        return -1;
    }

    public boolean ajouterVote(int[] cle) {
        for(int i=0;i<cles.size();i++) {
            if(egaux(cles.get(i),cle)) {
                objets.set(i, objets.get(i)+1);
                return true;
            }
        }
        cles.add(cle);
        objets.add(1);
        return false;
    }

    private boolean egaux(int[] a, int[] b) {
        if(a.length == b.length) {
            for(int i = 0 ;i<a.length;i++) {
                if(a[i] != b[i])
                    return false;
            }
            return true;
        }
        else {
            return false;
        }
    }

    public int[] cercle() {
        int max = 0;
        int[] cle = null;
        for(int i = 0;i<cles.size();i++) {
            if(objets.get(i) > max ) {
                max = objets.get(i);
                cle = cles.get(i);
            }
        }
        return cle;
    }


}
```