Lucrarea 5 SRF: Extragerea trăsăturilor (features).
Momentele unei forme.
Calculul distanței în spațiul Features.
Recunoașterea optică a caracterelor.

Scopul lucrării

Lucrarea aceasta prezintă o aplicație de recunoaștere a caracterelor folosind un algoritm specializat bazat pe librăria OpenCV. Ca urmare a executării acestei lucrări, studenții vor realiza importanța etapei de extragere a trăsăturilor unei forme (feature extraction) pentru recunoașterea acelei forme. Trăsăturile pe care studenții le vor extrage la acest laborator sunt momentele unei forme. In partea a doua a lucrării, studenții vor putea să testeze diferiți algoritmi de decizie, pe baza calculării distanței dintre forme în spațiul trăsăturilor (Features).

Pe baza acestui exemplu studenții vor putea înțelege practic ce înseamnă invariabilitatea momentelor la translație, rotație și scalare, caracteristici necesare pentru construirea unui spațiu al trăsăturilor (feature space) cât mai performant. Vor putea de asemenea să construiască și să optimizeze algoritmi de decizie folosindu-se de distanța dintre obiecte în spațiul Features.

1. Breviar teoretic:

Trăsăturile (features) sunt valori matematice pe baza cărora putem departaja între ele diferitele forme sau obiecte. Trăsăturile unei forme trebuie să fie invariabile la TRANSLAȚIE, ROTAȚIE și SCALARE pentru a fi eficiente. De asemenea, ele trebuie să aibă valori asemănătoare pentru formele ce aparțin unei categorii de forme, dar diferite pentru formele aparținînd altor categorii.

Algoritmii de recunoaștere a formelor pot fi departajați în funcție de tipul de transformare matematică pe care se bazează în felul următor (Sven Loncaric, A survey of shape analysis techniques, Pattern Recognition, Volume 31, Issue 8, 1998):

A. Transformări Contur -> Scalar:

- Bending Energy
- Arc-Height Method

B. Transformări Contur -> Spatiu:

- Hough Transform (aplicată pe contur)
- Fourier Transform (aplicată pe contur sau masca binara)
- Chain Code
- Boundary Approximations
- Boundary Decomposition

C. Transformări Spatiu -> Scalar:

- Maxima count (aplicată transformatei Hough)

- Moments (aplicata asupra măștii binare)

- Metoda Momentelor Mecanice: bazată pe calculul momentului cartezian 2-dimensional m(p,q) de ordin p+q a imaginii matriceale f(x,y)

- Shape Matrices

- Metode morfologice

D. Spatiu -> Spatiu

- Shape Decomposition

Pentru această lucrare de laborator s-a ales ca și algoritm de calcul al trăsăturilor Metoda Momentelor Mecanice. Descrierea formelor pe baza de momente nu duce la pierderi de informatie despre formele analizate.

- Momente de ordin zero:

m(0,0) - atunci cand f(x,y) este normata la 1, momentul de ordin zero reprezinta chiar suprafata (in pixeli) a obiectului analizat

- Momente de ordin unu:

m(0,1) si m(1,0) - pe baza lor putem afla coordonatele centrului de greutate (xc, yc) ale obiectului analizat

xc=m(1,0)/m(0,0)

yc=m(0,1)/m(0,0)

- Momente de ordin doi: se numesc momente de inertie si pot fi folosite pentru a determina asa-zisele axe principale ale obiectului, care sunt acele axe pentru care aceste momente de inertie ating valoarea lor maxima/minima: m(0,2), m(2,0), m(1,1)

TOATE ACESTE MOMENTE NU SUNT INVARIANTE LA TRANSLATIE, ROTATIE, SCALARE.

Pentru a crea momente mecanice care sa fie invariabile la translatie, alegem ca punct de referinta pentru calcularea lor centrul de greutate al obiectului => introducem Momentele Centrale.

Aceste momente centrale se pot folosi si pentru a crea momente care sa fie invariante si la rotatie si scalare (pe langa translatie). Hu (1962) a descris un numar de 7 astfel de momente (mi1, mi2, mi3, mi4, mi5, mi6, mi7), dar pentru identificarea corecta a formei literelor alfabetului de exemplu sunt suficiente intre 2 si 4 astfel de momente.

2. Desfășurarea lucrării

Lucrarea de față își propune, pe baza exemplificării unui algoritm, să urmărească un algoritm de detecție a literelor, pe baza folosirii momentelor mecanice. Pentru o mai ușoară exemplificare, fontul folosit va fi **Times New Roman.**

Programul din acest laborator are următoarele funcții: **setup()**, **curata()**, **draw(), miu()**, **m(), distanta()** *(funcție cu patru parametri - x1,y1,x2,y2 dar și cu șase parametri – x1,x2,y1,y2,z1,z2)***, Hough(), DTF3(), DTF2(), FFT()** și **mousePressed()**

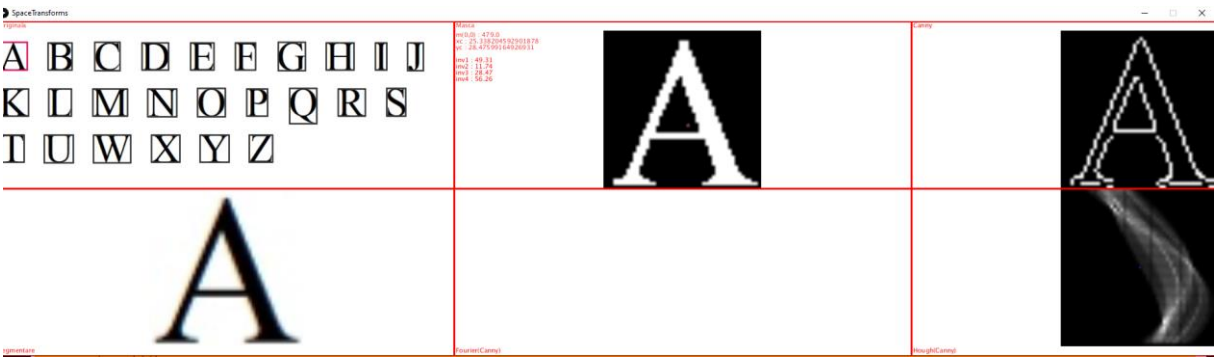
În primă instanță, vom deschide un editor de imagine (GIMP, Photoshop, etc), vom scrie un text demonstrativ de două sau trei litere (exemplu: A B C sau D E F) **în așa fel încât să fie suficient spațiu între litere** (pentru o mai bună încadrare a literelor în blob-uri diferite).

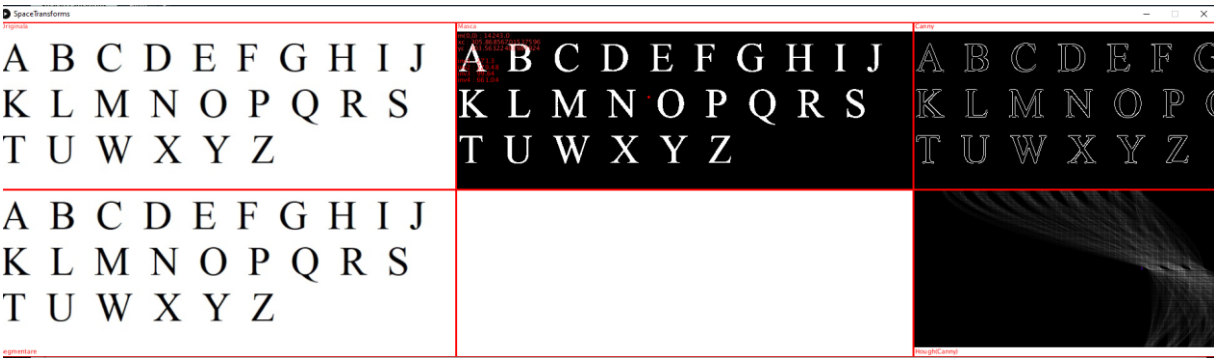
Vom face un tabel care să aibă pe fiecare rând câte o literă din alfabet si patru coloane cu cele 4 momente mecanice: **inv1, inv2**, **inv3** si **inv4** precum și **dist2, dist 4**.

Așadar, în acest algoritm, avem sase zone, prima este reprezentată de poza care este dată spre analiză, masca ce reprezintă detecția obiectului în sine, apoi celelalte sunt reprezentate de **segmentare** (*separarea pixelilor obiectului efectiv la care facem analiza)***,** apoi putem aplica diferite transformate (**Fourier** pe masca binară, **Fourier pe contur** și **Hough pe contur**).

Atentie: în cadrul acestui program, se va folosi de către toți studenții același format de font pentru a nu avea rezultate cu o diferență notabilă.

Observăm, în poza de mai jos, ce se întâmplă dacă efectuăm analiza amănunțită pe baza unei imagini care are linii. Avem masca, iar în cadrul zonei Transformatei lui Hough (dreapta jos), nu contează faptul că există întreruperi, ci faptul că fragmentele fac parte din aceeași linie și se acumulează.





Codul programului folosit la acest laborator este următorul:

```
import gab.opencv.*;

OpenCV opencv;
ArrayList<Line> lines;
PImage src, src1, img1, img2;
int xt = 3, yt = 15;
boolean ready = false, afisaj = false;
ArrayList<Blob> blobs = new ArrayList<Blob>();
color bgColor = color(255);
int threshold = 20;

void settings(){
    src = loadImage("imag7.jpg");
    src.resize(0, 250);
    size(src.width * 3, src.height * 2 + 30);
    src1 = src.copy();
}

void setup(){
    curata();
}

void curata(){

    PImage alb = new PImage(src.width * 3, src.height * 2 + 30);
    for (int i = 0; i < alb.width; i++){
        for (int j = 0; j < alb.height; j++)
            alb.pixels[j * alb.width + i] = color(255);
        }
    }

    image(alb, 0, 0);

    img1 = src1.copy();
    img1.filter(GRAY);
    img1.filter(INVERT);
    img1.filter(THRESHOLD, 0.3);

    opencv = new OpenCV(this, img1);
    opencv.findCannyEdges(20, 60);
    img2 = opencv.getOutput();

    fill(250, 50, 50);
```

```
    textSize(10);

    image(src, 0, yt);
    text("Originala", xt, 10);

    PImage img1a = img1.copy();
    img1a.resize(0, src.height);
    image(img1a, src.width + (src.width - img1a.width) / 2, yt);
    text("Masca", src.width + xt, 10);
    double m00 = m(img1a, 0, 0);
    double xc = m(img1a, 1, 0) / m00;
    double yc = m(img1a, 0, 1) / m00;
    text("m(0,0) : " + m00, src.width + xt, yt + 10);
    text("xc : " + xc, src.width + xt, yt + 20);
    text("yc : " + yc, src.width + xt, yt + 30);
    //circle(src.width+(src.width-
img1a.width)/2+round(xc),yt+round(yc),5);

    text("inv1 : " + mi1(img1a), src.width + xt, yt + 50);
    text("inv2 : " + mi2(img1a), src.width + xt, yt + 60);

    PImage src1a;
    // DFT(imagine, luminozitate)
    if (!afisaj) src1a = src.copy();
    else src1a = src1.copy();
    //  image(DFT2(src1,10), 0, yt+src.height);

    src1a.resize(0, src.height);
    image(src1a, (src.width - src1a.width) / 2, yt + src.height);

    text("Segmentare", xt, yt + 2 * src.height + 10);

    if (afisaj)
        //PImage imagine=DFT2(img2,10);
        //imagine.resize(0, src.height);
        //image(imagine, src.width+(src.width-imagine.width)/2,
yt+src.height);
        text("Fourier(Canny)", src.width + xt, yt + 2 * src.height + 10);
    }

    if (true){
        PImage img2a = img2.copy();
        img2a.resize(0, src.height);
        image(img2a, 2 * src.width + (src.width - img2a.width) / 2, yt);
        text("Canny", 2 * src.width + xt, 10);
    }

    if (true){
        PImage img2a = Hough(img2);
        img2a.resize(0, src.height);
        image(img2a, 2 * src.width + (src.width - img2a.width) / 2, yt +
src.height);
        text("Hough(Canny)", 2 * src.width + xt, yt + 2 * src.height +
10);
    }

    strokeWeight(1);
    stroke(0, 0, 255, 150);
    //line(2*src.width,yt+1.5*src.height,3*src.width,yt+1.5*src.height);
    //line(2.5*src.width,yt+1*src.height,2.5*src.width,yt+2*src.height);
    circle(2.5 * src.width, yt + 1.5 * src.height, 2);
    ready = true;

    // Begin loop to walk through every pixel
    for (int x = 0; x < src.width; x++){
```

```
        for (int y = 0; y < src.height; y++){
            int loc = x + y * src.width;
            // What is current color
            color currentColor = src.pixels[loc];

            float r1 = red(currentColor);
            float g1 = green(currentColor);
            float b1 = blue(currentColor);
            float r2 = red(bgColor);
            float g2 = green(bgColor);
            float b2 = blue(bgColor);

            float d = distanta(r1, g1, b1, r2, g2, b2);

            if (d > threshold){
                boolean found = false;
                for (Blob b: blobs) {
    if (b.isNear(x, y)){
        b.add(x, y);
        found = true;
        break;
    }
}

if (!found){
    Blob b = new Blob(x, y);
    blobs.add(b);
}
            }
        }
    }
    for (int j = blobs.size() - 1; j >= 0; j--){
    Blob b1 = blobs.get(j);
    for (int i = blobs.size() - 1; i > j; i--){
        Blob b2 = blobs.get(i);
        if (b1.minx <= b2.minx && b1.maxx >= b2.maxx && b1.miny <=
b2.miny && b1.maxy >= b2.maxy){
            blobs.remove(i);
        }
    }
}
if (afisaj){
    for (Blob b: blobs){
        b.show();
        if (b.minx < mouseX && b.maxx > mouseX && b.miny < mouseY - yt &&
b.maxy > mouseY - yt){
            b.highlight();
        }
    }
}

strokeWeight(3);
stroke(255, 0, 0);
line(0, 0, 3 * src.width, 0);
line(0, yt + src.height, 3 * src.width, yt + src.height);
line(0, 2 * (yt + src.height), 3 * src.width, 2 * (yt + src.height));
line(src.width, 0, src.width, 2 * src.height + 2 * yt);
line(0, 0, 0, 2 * src.height + 2 * yt);
line(2 * src.width, 0, 2 * src.width, 2 * src.height + 2 * yt);
line(3 * src.width, 0, 3 * src.width, 2 * src.height + 2 * yt);
}

void draw(){

}
```

```
void mousePressed(){

    if (mouseX < src.width && mouseY < yt + src.height){
        for (Blob b: blobs) {
    b.show();
    if (b.minx < mouseX && b.maxx > mouseX && b.miny < mouseY - yt &&
b.maxy > mouseY - yt)    {
        b.highlight();
        src1 = new PImage(b.maxx - b.minx + 4, b.maxy - b.miny + 4);
        for (int i = 0; i < b.maxx - b.minx + 4; i++){
            for (int j = 0; j < b.maxy - b.miny + 4; j++){
                src1.pixels[j * (b.maxx - b.minx + 4) + i] =
src.pixels[(j + b.miny - 2) * src.width + i + b.minx - 2];
            }
        }

        //image(src1a, (src.width-src1.width)/2, yt+src.height+2);
        afisaj = true;
        curata();
    }
}

    }

    PImage src1a = src1.copy();
src1a.resize(0, src.height);
int M = src1a.width;
int N = src1a.height;
if (mouseX > 2 * src.width && mouseX < 3 * src.width && mouseY > yt +
src.height && mouseY < yt + 2 * src.height){
    int x = mouseX - 2 * src.width - (src.width - src1a.width) / 2;
    int y = mouseY - src.height - yt;
    float r = 2 * sqrt(M * M + N * N) * (x - (M - 1) / 2) / (M - 1);
    float theta = PI * (y - (N - 1) / 2) / (N - 1);
    strokeWeight(2);
    stroke(0, 0, 0);
    int intersectieY1 = round(r / cos(theta));
    int intersectieX1 = round(r / sin(theta));
    int intersectieX2 = 0;
    int intersectieY2 = 0;

    println("B:(" + intersectieX2 + "," + intersectieY1 + ")-(" +
intersectieX1 + "," + intersectieY2 + ") - theta: " + theta);
    if (intersectieY1 > N - 1)    {
        intersectieX2 = round((intersectieY1 - N + 1) / tan(theta));
        intersectieY1 = N - 1;
    }
    if (intersectieX1 > M - 1)   {
        intersectieY2 = round((intersectieX1 - M + 1) * tan(theta));
        intersectieX1 = M - 1;
    }
    if (intersectieY1 < 0 && intersectieX1 > 0){
        intersectieY1 = intersectieY2 + (intersectieY2 - intersectieY1);
        intersectieX2 = intersectieX1 + (intersectieX1 - intersectieX2);
    }
    if (intersectieX1 < 0 && intersectieY1 > 0)
        intersectieX1 = intersectieX2 + (intersectieX2 - intersectieX1);
        intersectieY2 = intersectieY1 + (intersectieY1 - intersectieY2);
    }
    println("A:(" + intersectieX2 + "," + intersectieY1 + ")-(" +
intersectieX1 + "," + intersectieY2 + ")");

    if (abs(theta) < 0.000001){
        intersectieY2 = intersectieY1;
```

```
    }

    /*   if(intersectieY1>N-1) {
         intersectieX2=round((intersectieY1-N+1)/tan(theta));
         intersectieY1=N-1;
       }
       if(intersectieX1>M-1) {
         intersectieY2=round((intersectieX1-M+1)*tan(theta));
         intersectieX1=M-1;
       }*/

    //if(intersectieX2<M && intersectieY2<N && intersectieX2>=0 &&
intersectieY2>=0 && intersectieX1>=0 && intersectieY1>=0) {
    stroke(0);
    line(intersectieX2 + (src.width - src1a.width) / 2, intersectieY1 +
src.height + yt, intersectieX1 + (src.width - src1a.width) / 2,
intersectieY2 + src.height + yt);
    //}
    //println("r : "+r);
    //println("theta : "+theta*180/PI);
    //println();
    stroke(255, 0, 0);
    strokeWeight(1);
    noFill();
    circle(mouseX, mouseY, 5);
}

}

void keyPressed(){
    if (key == 'c') curata();

}

PImage FFT(PImage intrare, int luminozitate){
    intrare.filter(GRAY);
    PImage iesire = intrare.copy();
    int M = intrare.width;
    int N = intrare.height;
    int maxim = 0;
    int[] t = new int[M * N];
    float[] f = new float[M * N];
    for (int y = 0; y < N; y++)
    {
        for (int x = 0; x < M; x++)
        {
            f[y * M + x] = red(intrare.pixels[y * M + x]);
        }
    }
    return iesire;
}

PImage DFT2(PImage intrare, int luminozitate){
    intrare.filter(GRAY);
    PImage iesire = intrare.copy();
    int M = intrare.width;
    int N = intrare.height;
    int maxim = 0;
    int[] t = new int[M * N];
    float[] f = new float[M * N];
    float[] Freal = new float[M * N];
    float[] Fimag = new float[M * N];
    int media = 0;
    for (int y = 0; y < N; y++)     {
        for (int x = 0; x < M; x++)
```

```
        {
            f[y * M + x] = red(intrare.pixels[y * M + x]);
        }
    }

    for (int v = 0; v < N; v++){
        for (int x = 0; x < M; x++){
            Freal[v * M + x] = 0;
            Fimag[v * M + x] = 0;
            for (int y = 0; y < N; y++)
            {
                Freal[v * M + x] += f[y * M + x] * cos(2 * PI * v * y /
(N + 0.0));
                Fimag[v * M + x] += f[y * M + x] * sin(2 * PI * v * y /
(N + 0.0));
            }
        }
    }

    for (int v = 0; v < N; v++){
        for (int u = 0; u < M; u++){
            float real = 0;
            float imag = 0;
            for (int x = 0; x < M; x++){
                real += Freal[v * M + x] * cos(2 * PI * u * x / (M +
0.0)) - Fimag[v * M + x] * sin(2 * PI * u * x / (M + 0.0));
                imag += Fimag[v * M + x] * cos(2 * PI * u * x / (M +
0.0)) + Freal[v * M + x] * sin(2 * PI * u * x / (M + 0.0));
            }
            t[v * M + u] = round((sqrt((real * real + imag * imag) / (M *
N * 1.0))));

            if (t[v * M + u] > maxim) maxim = t[v * M + u];
            media += t[v * M + u];
        }

    }
    media = media / (M * N);

    for (int v = 0; v < N; v++){
        for (int u = 0; u < M; u++)
            //t[v*M+u]=t[v*M+u]*255/maxim;
            t[v * M + u] = luminozitate * t[v * M + u] / media;
            if (t[v * M + u] > 255) t[v * M + u] = 255;
            int vv = v + N / 2;
            int uu = u + M / 2;
            if (vv > N - 1) vv = vv - N;
            if (uu > M - 1) uu = uu - M;
            iesire.pixels[vv * M + uu] = color(t[v * M + u]);
        }
    }

    return iesire;
}

PImage DFT3(PImage intrare, int luminozitate){
    intrare.filter(GRAY);
    PImage iesire = intrare.copy();
    int M = intrare.width;
    int N = intrare.height;
    int maxim = 0;
    int[] t = new int[M * N];
    int media = 0;
    for (int v = 0; v < N; v++){
        for (int u = 0; u < M; u++){
```

```
            float real = 0.0;
            float imag = 0.0;
            for (int y = 0; y < N; y++){
                for (int x = 0; x < M; x++)
                    float f = red(intrare.pixels[y * M + x]);
                    real += f * cos(2 * PI * (u * x / (M + 0.0) + v * y /
(N + 0.0)));
                    imag += f * sin(2 * PI * (u * x / (M + 0.0) + v * y /
(N + 0.0)));
                }
            }
            t[v * M + u] = round(sqrt((real * real + imag * imag) / (M *
N * 1.0)));

            if (t[v * M + u] > maxim) maxim = t[v * M + u];
            media += t[v * M + u];
        }

    }
    media = media / (M * N);

    for (int v = 0; v < N; v++){
        for (int u = 0; u < M; u++){
            //t[v*M+u]/=maxim;
            t[v * M + u] = luminozitate * t[v * M + u] / media;
            if (t[v * M + u] > 255) t[v * M + u] = 255;
            int vv = v + N / 2;
            int uu = u + M / 2;
            if (vv > N - 1) vv = vv - N;
            if (uu > M - 1) uu = uu - M;
            iesire.pixels[vv * M + uu] = color(t[v * M + u]);
        }
    }

    return iesire;
}

PImage Hough(PImage intrare){
    intrare.filter(GRAY);
    PImage iesire = intrare.copy();
    int M = intrare.width;
    int N = intrare.height;
    int[] t = new int[M * N];
    int maxim = 0;
    for (int y = 0; y < N; y++)
    {
        for (int x = 0; x < M; x++)
        {
            t[y * M + x] = 0;
        }
    }
    for (int y = 0; y < N; y++){
        for (int x = 0; x < M; x++){
            if (red(intrare.pixels[y * M + x]) > 100) {
                for (float theta = -90; theta <= 90; theta = theta + 0.5)
                {
                    int r = round(x * sin(radians(theta)) + y *
cos(radians(theta)));
                    int xx = round((M - 1) / 2 + r * (M - 1) / (2 *
sqrt(M * M + N * N)));
                    int yy = round((N - 1) / 2 + theta * (N - 1) / 180);

                    t[yy * M + xx]++;
                    if (t[yy * M + xx] > maxim) maxim = t[yy * M + xx];
                }
```

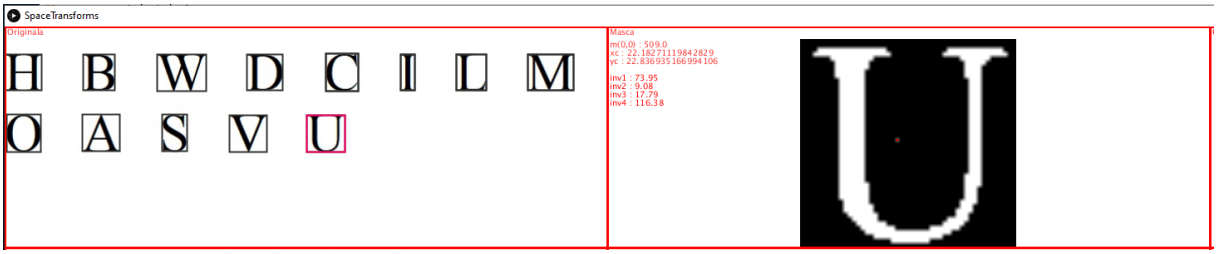
```
            }
        }
    }

    for (int y = 0; y < N; y++){
        for (int x = 0; x < M; x++){
            iesire.pixels[y * M + x] = color(round(t[y * M + x] * 255 /
maxim));
        }
    }

    return iesire;
}

float distanta(float x1, float y1, float x2, float y2){
    float d = sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
    return d;
}

float distanta(float x1, float y1, float z1, float x2, float y2, float
z2){
    float d = sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1) + (z2 -
z1) * (z2 - z1));
    return d;
}

double m(PImage intrare, int p, int q){
    double moment = 0.0;
    intrare.filter(GRAY);
    int M = intrare.width;
    int N = intrare.height;
    for (int y = 0; y < N; y++){
        double yy = 1.0;
        for (int i = 0; i < q; i++){
            yy = yy * y;
        }

        for (int x = 0; x < M; x++){
            double f = red(intrare.pixels[M * y + x]) / 255.0;
            double xx = 1.0;
            for (int i = 0; i < p; i++){
                xx = xx * x;
            }

            moment = moment + xx * yy * f;
        }
    }

    return moment;
}

double miu(PImage intrare, int p, int q){
    double m00 = m(intrare, 0, 0);
    double xc = m(intrare, 1, 0) / m00;
    double yc = m(intrare, 0, 1) / m00;

    println("xc: " + xc + " ; yc: " + yc);

    double moment = 0.0;
    intrare.filter(GRAY);
    int M = intrare.width;
    int N = intrare.height;

    double minim = 0;
    double xmin = 0, ymin = 0;
```

```
    for (int y = 0; y < N; y++){
        double yy = 1.0;
        for (int i = 0; i < q; i++){
            yy = yy * (y - yc);
        }

        for (int x = 0; x < M; x++){
            double f = red(intrare.pixels[M * y + x]) / 255.0;
            double xx = 1.0;
            for (int i = 0; i < p; i++){
                xx = xx * (x - xc);
            }

            moment = moment + xx * yy * f;
            if (moment < minim){
                minim = moment;
                xmin = x;
                ymin = y;
            }
        }
    }

    println("M: " + M + " , N: " + N);
    println("minim: " + minim + " , x : " + xmin + " , y : " + ymin);
    return moment;

}

double mi1(PImage intrare)
    double m00 = m(intrare, 0, 0);
    double moment = miu(intrare, 0, 2) + miu(intrare, 2, 0);
    //double moment=miu(intrare,0,1);

    return moment * 100.0 / m00 / m00;
}

double mi2(PImage intrare)
{
    double m00 = m(intrare, 0, 0);
    double moment1 = miu(intrare, 2, 0) - miu(intrare, 0, 2);
    double moment2 = miu(intrare, 1, 1);
    double moment_ = moment1 * moment1 + 4 * moment2 * moment2;
    double moment = sqrt((float)moment_);
    return moment * 100.0 / m00 / m00;
}
```

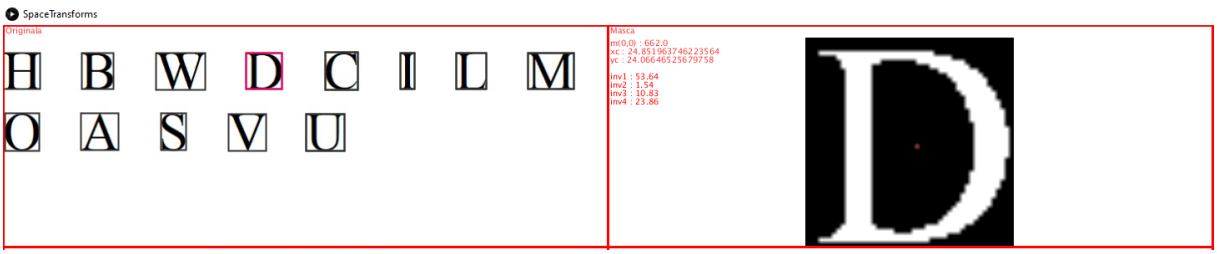
Se va crea o imagine in care să apară toate literele fontului **Times New Roman** cu dimensiunea **48**. Prin rularea programului, se vor calcula și nota pentru fiecare literă in parte cele 4 momente mecanice: **inv1,inv2,inv3,inv4** si se vor adăuga in tabelul de mai jos:

	A	B	C	D	E	F	G	H	I
1			inv1	inv2	inv3	inv4	dist2	dist4	
2	Times New Roman Marime font 48	A	49.86	11.85	28.98	56.06	0.50	1.27	A
3		B	39.43	6.91	4.12	7.13	0.36	0.69	B
4		C	75.24	21.92	26.33	226.25	1.06	6.81	C
5		D	54.62	1.35	11.18	21.96	0.02	0.07	D
6		E	60.02	23.84	3.99	70.96	1.09	1.91	E
7		F	50.91	29.77	15.89	34.76	1.37	1.42	F
8		G	62.28	4.43	9.16	68.43	0.21	1.49	G
9		H	49.69	2.31	0.44	11.89	0.08	0.49	H
10		I	35.98	51.98	0.29	1.29	2.47	2.59	I
11		J	66.18	57.82	15.97	57.27	2.74	2.96	J
12		K	45.62	14.94	8.76	52.11	0.67	1.15	K
13		L	63.88	41.42	30.59	99.06	1.94	3.20	L
14		M	46.77	10.06	3.62	12.23	0.43	0.61	M
15		N	50.31	5.69	4.86	18.91	0.21	0.31	N
16		O	55.75	5.22	0.28	2.33	0.18	0.79	O
17		P	41.88	19.84	17.33	47.12	0.91	1.21	P
18		Q	57.98	9.05	12.21	4.78	0.37	0.74	Q
19		R	41.08	13.68	5.21	19.17	0.63	0.66	R
20		S	48.92	22.28	2.65	10.73	1.01	1.12	S
21		T	65.57	38.43	40.91	89.26	1.80	2.93	T
22		U	69.56	6.5	15.87	126.5	0.37	3.43	U
23		V	52.43	11.78	30.26	72.02	0.50	1.75	V
24		W	42.93	5.65	12.58	38.9	0.28	0.57	W
25		X	51.36	18.35	2.17	11.4	0.82	0.94	X
26		Y	55.99	20.62	38.83	30.13	0.93	1.19	Y
27		Z	64.47	35.03	6.87	26.32	1.64	1.64	Z

Se va crea apoi o nouă imagine în care să apară una dintre litere dar rotită și scalată. Să numim noul caracter creat litera necunoscută. Vom calcula cele 4 momente pentru caracterul necunoscut și vom compara cele 4 valori obținute cu valorile momentelor pentru toate literele calculate la pasul anterior. Vom implementa manual un algoritm de calcul al distanțelor dintre 2 caractere în spațiul fazelor. Vom numi funcția care implementează acest algoritm dist2 dacă folosim doar 2 momente, și dist4 dacă folosim toate cele 4 momente. Vom compara eficiența detecției în cele 2 cazuri.



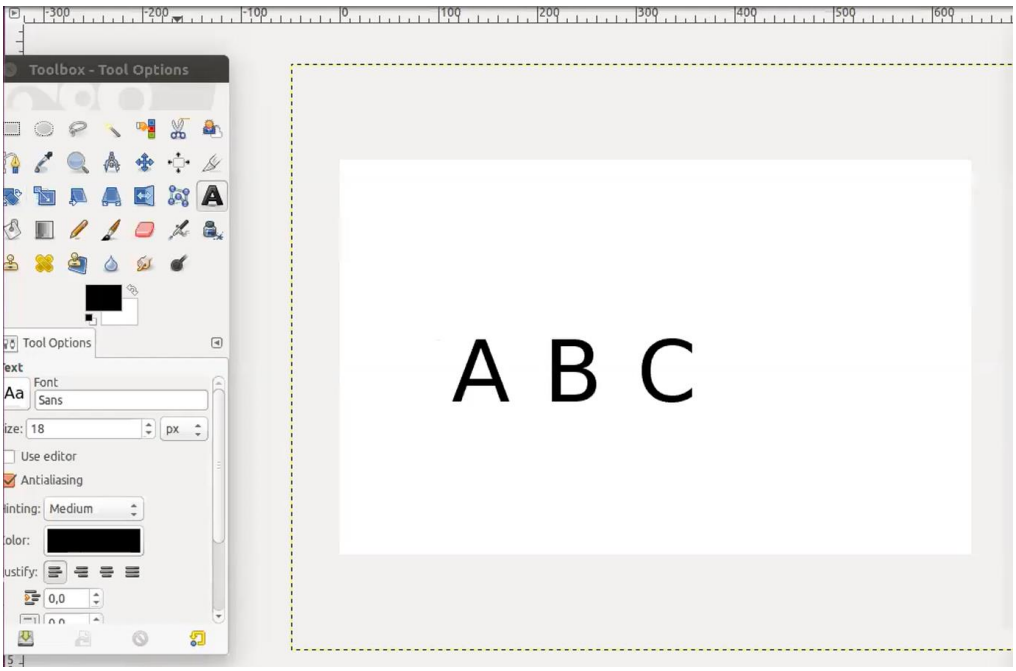
	A	B	C	D	E	F	G	H	I
1			inv1	inv2	inv3	inv4	dist2	dist4	
2	Times New Roman Marime font 48	A	49.86	11.85	28.98	56.06	0.45	2.07	A
3		B	39.43	6.91	4.12	7.13	0.63	3.70	B
4		C	75.24	21.92	26.33	226.25	0.62	3.71	C
5		D	54.62	1.35	11.18	21.96	0.51	3.18	D
6		E	60.02	23.84	3.99	70.96	0.76	1.72	E
7		F	50.91	29.77	15.89	34.76	1.08	2.92	F
8		G	62.28	4.43	9.16	68.43	0.31	1.64	G
9		H	49.69	2.31	0.44	11.89	0.54	3.54	H
10		I	35.98	51.98	0.29	1.29	2.19	4.43	I
11		J	66.18	57.82	15.97	57.27	2.37	3.08	J
12		K	45.62	14.94	8.76	52.11	0.58	2.22	K
13		L	63.88	41.42	30.59	99.06	1.58	1.71	L
14		M	46.77	10.06	3.62	12.23	0.49	3.51	M
15		N	50.31	5.69	4.86	18.91	0.45	3.28	N
16		O	55.75	5.22	0.28	2.33	0.38	3.83	O
17		P	41.88	19.84	17.33	47.12	0.77	2.43	P
18		Q	57.98	9.05	12.21	4.78	0.29	3.72	Q
19		R	41.08	13.68	5.21	19.17	0.63	3.30	R
20		S	48.92	22.28	2.65	10.73	0.78	3.61	S
21		T	65.57	38.43	40.91	89.26	1.43	1.79	T
22		U	69.56	6.5	15.87	126.5	0.15	0.37	U
23		V	52.43	11.78	30.26	72.02	0.41	1.56	V
24		W	42.93	5.65	12.58	38.9	0.58	2.64	W
25		X	51.36	18.35	2.17	11.4	0.60	3.56	X
26		Y	55.99	20.62	38.83	30.13	0.65	2.98	Y
27		Z	64.47	35.03	6.87	26.32	1.27	3.26	Z
28									
29		STD-DEV	10.00	15.32	12.13	49.16			
30									
31		Litera misteroa	73.95	9.08	17.79	116.38	0.15	0.37	minim



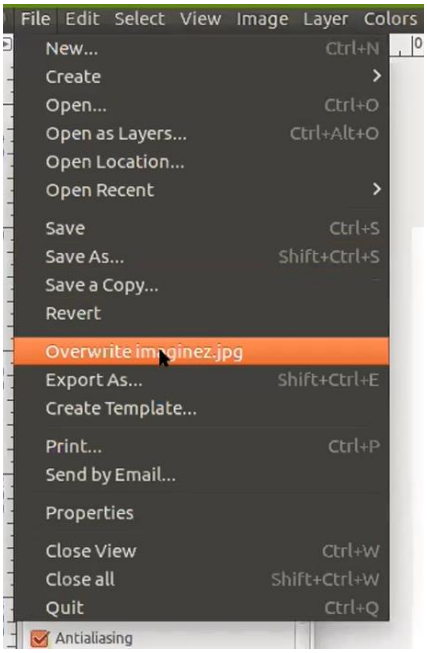
	A	B	C	D	E	F	G	H	I
1			inv1	inv2	inv3	inv4	dist2	dist4	
2	Times New Roman Marime font 48	A	49.86	11.85	28.98	56.06	0.50	1.27	A
3		B	39.43	6.91	4.12	7.13	0.36	0.69	B
4		C	75.24	21.92	26.33	226.25	1.06	6.81	C
5		D	54.62	1.35	11.18	21.96	0.02	0.07	D
6		E	60.02	23.84	3.99	70.96	1.09	1.91	E
7		F	50.91	29.77	15.89	34.76	1.37	1.42	F
8		G	62.28	4.43	9.16	68.43	0.21	1.49	G
9		H	49.69	2.31	0.44	11.89	0.08	0.49	H
10		I	35.98	51.98	0.29	1.29	2.47	2.59	I
11		J	66.18	57.82	15.97	57.27	2.74	2.96	J
12		K	45.62	14.94	8.76	52.11	0.67	1.15	K
13		L	63.88	41.42	30.59	99.06	1.94	3.20	L
14		M	46.77	10.06	3.62	12.23	0.43	0.61	M
15		N	50.31	5.69	4.86	18.91	0.21	0.31	N
16		O	55.75	5.22	0.28	2.33	0.18	0.79	O
17		P	41.88	19.84	17.33	47.12	0.91	1.21	P
18		Q	57.98	9.05	12.21	4.78	0.37	0.74	Q
19		R	41.08	13.68	5.21	19.17	0.63	0.66	R
20		S	48.92	22.28	2.65	10.73	1.01	1.12	S
21		T	65.57	38.43	40.91	89.26	1.80	2.93	T
22		U	69.56	6.5	15.87	126.5	0.37	3.43	U
23		V	52.43	11.78	30.26	72.02	0.50	1.75	V
24		W	42.93	5.65	12.58	38.9	0.28	0.57	W
25		X	51.36	18.35	2.17	11.4	0.82	0.94	X
26		Y	55.99	20.62	38.83	30.13	0.93	1.19	Y
27		Z	64.47	35.03	6.87	26.32	1.64	1.64	Z
28									
29		STD-DEV	10.00	15.32	12.13	49.16			
30									
31		Litera misteroa	53.64	1.54	10.83	23.86	0.02	0.07	minim

Pas 1:

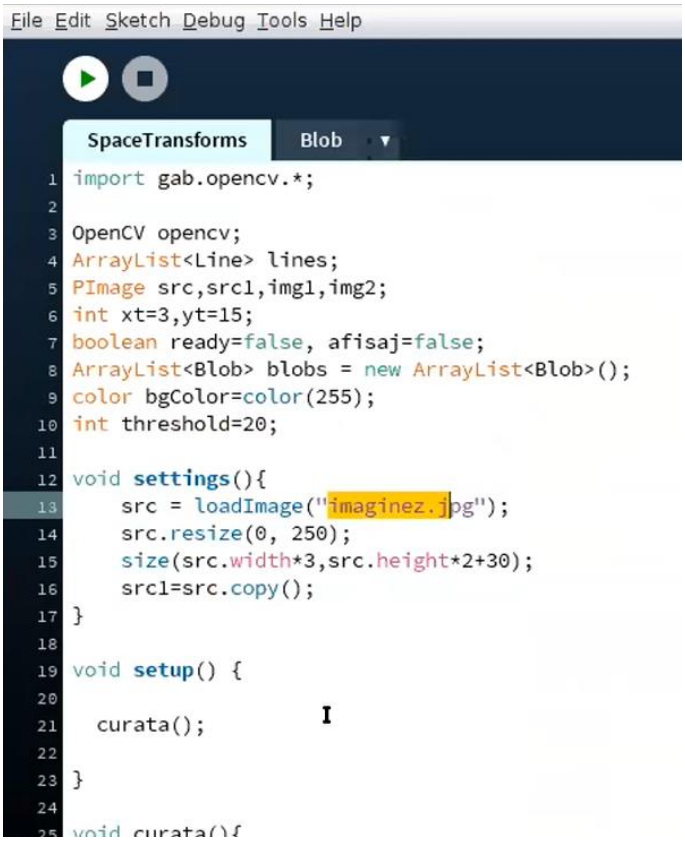
Scrieti literele corespunzatoare cu spatiu intre ele ca sa le poata separa ca bloburi diferite pentru ca in programul de detectie a gramezilor de pixeli a bloburilor exista un parametru care arata ca daca 2 obiecte sunt foarte aproape o sa le ia ca un singur obiect.



Pas 2:Exportati sub o anumită denumire (ex. imaginez.jpg)



Pas3: Puneți in cod denumirea anterioară.

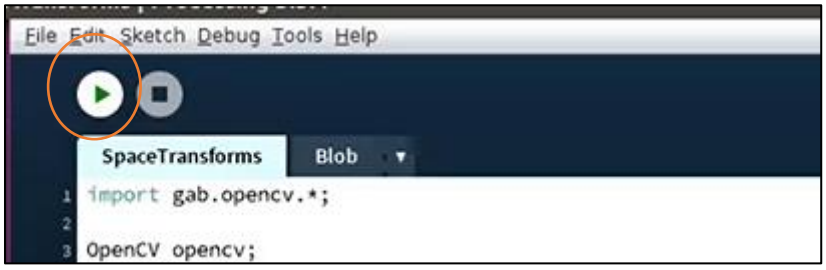


Fișierul imagine (imaginez.jpg) trebuie sa fie in folderul in care aveți si programul.

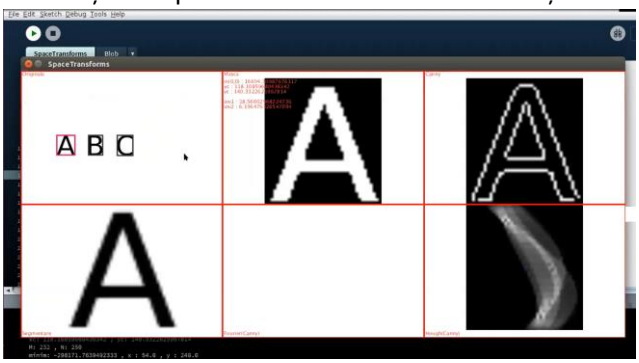
Imaginea sa aibă o dimensiune recomandată de 600x400px.

Pas 4:

Apăsați Run:



Pas 5:
Selectați rând pe rând fiecare caracter si notați fiecare moment invariabil(inv1 ,inv2).



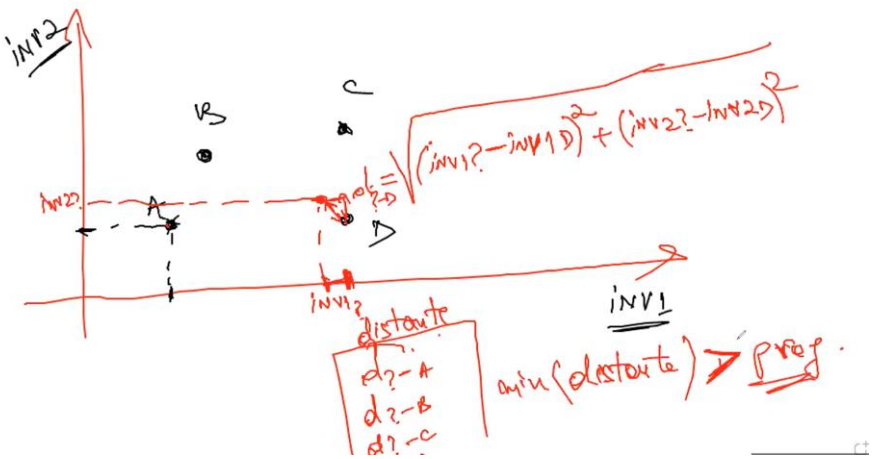
Calculați distanta intre litera necunoscută si una pe care o aveți deja folosind formula prezentata mai jos:

Dist=$\sqrt{(inv1n - inv1D)^2 + (inv2n - inv2D)^2}$ unde inv1n=inv1 pentru litera misterioasa

inv2n=inv2 pentru litera misterioasa

inv1D=inv1 pentru litera D

inv2D=inv2 pentru litera D

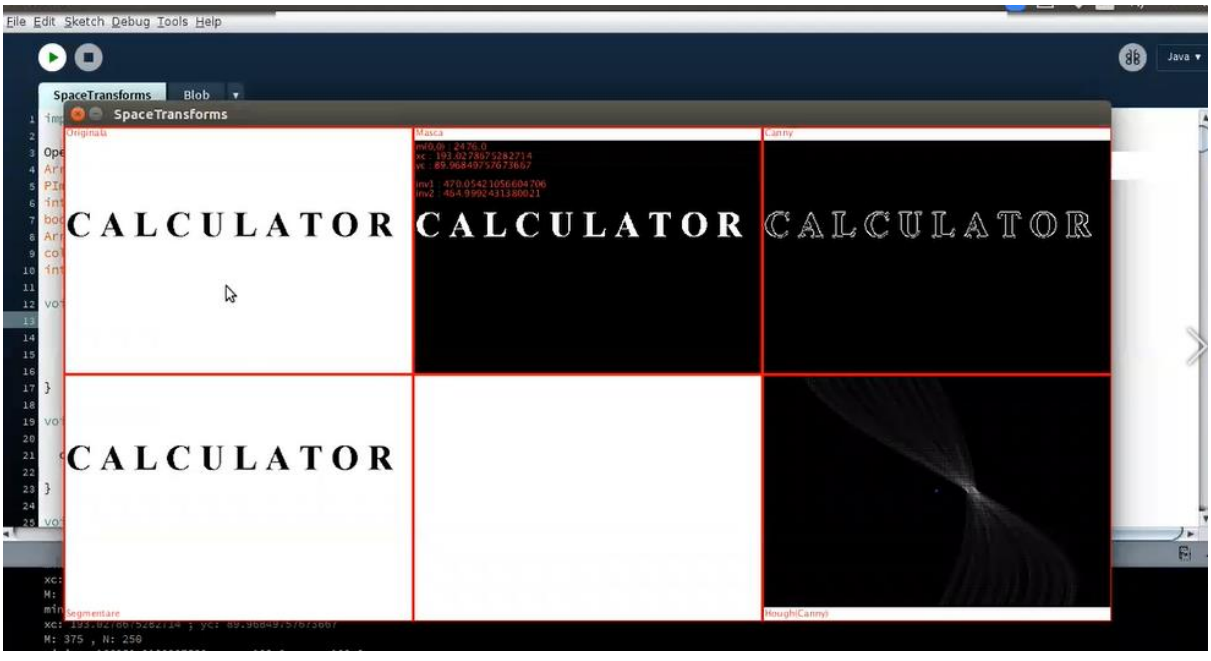


		INV1	INV2	DIST
Times New Roman	A	49.39	10.45	40.10
	B	40.73	7.95	46.91
	C	25.99	20.86	49.41
	D	51.67	2.83	46.02
	E	59.25	23.47	24.05
	F	55.78	34.13	17.31
	G	46.15	4.51	4.87
	H	33.11	0.31	57.62
	I	60.4	55.44	12.96
	J	60.36	53.04	11.29
	K	44.99	15.14	38.61
	L	68.28	45.04	0.72
	M	34.75	7.52	51.04
	N	38.89	7.35	48.51
	O	45.35	3.68	47.98
	P	37.7	20.44	39.97
	Q	56.41	7.15	40.34
	R	39.65	13.3	43.46
	S	49.5	21.3	30.98
	T	59.55	35.48	13.67
	U	76.26	8.99	37.30
	V	53.6	10.84	37.88

Litera misterioasa	68.8	45.54	L

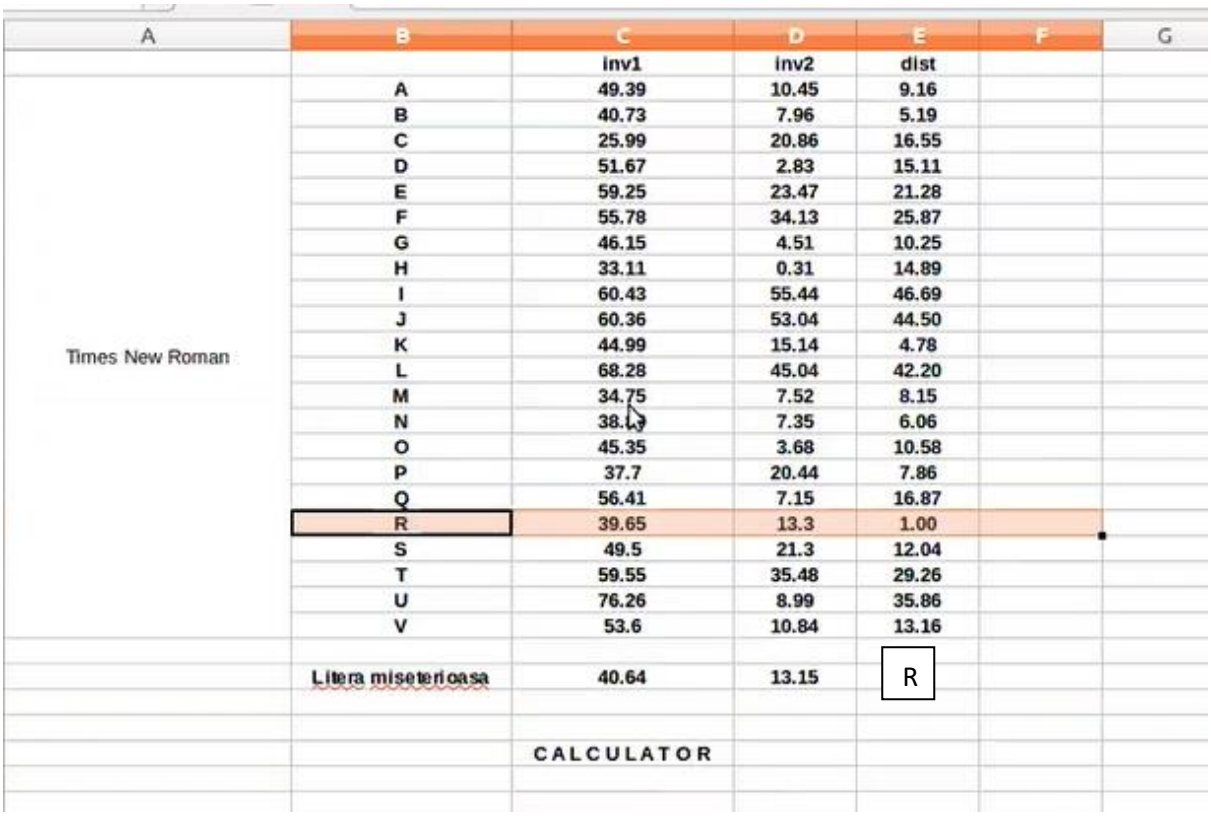
Implementați singuri codul pentru calculul distanței dintre 2 caractere în spațiul trăsăturilor folosindu-vă de formula de mai sus.

Analizați cuvântul calculator:



De RETINUT!!!

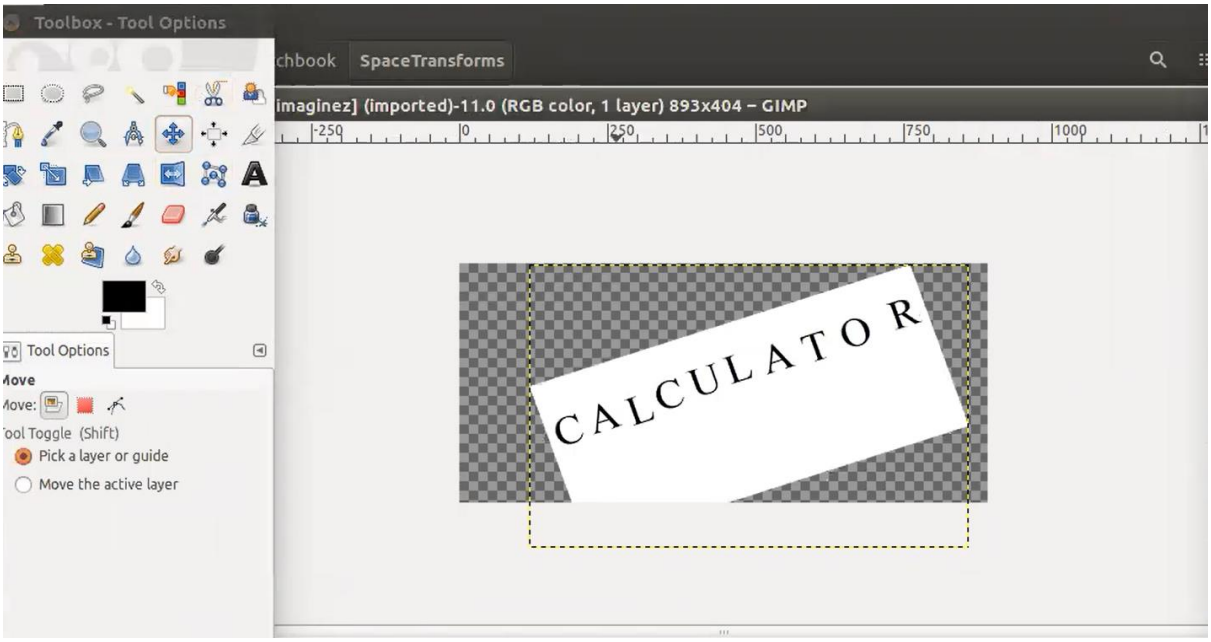
Aceste momente invariabile diferă pentru ca respectivele fonturi sunt puțin diferite in funcție de dimensiunea aleasa pentru scrierea lor .

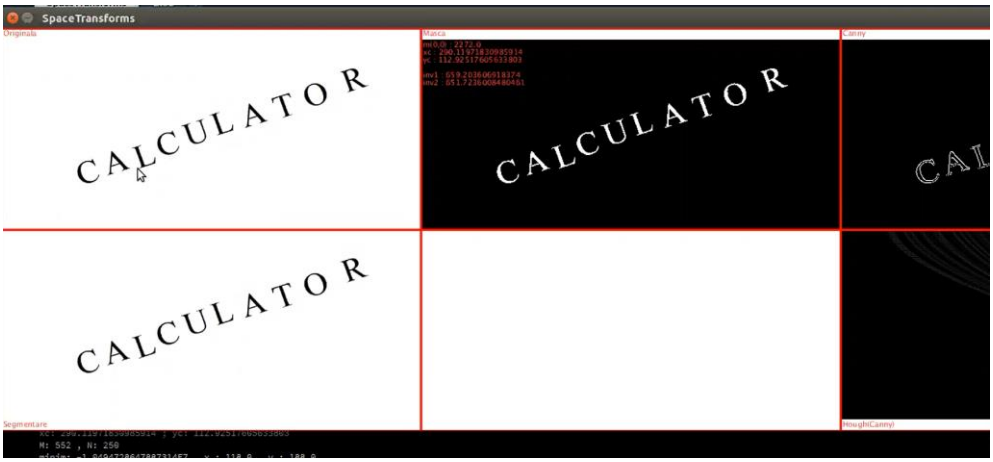


		inv1	inv2	dist	
	A	49.39	10.45	9.16	
	B	40.73	7.96	5.19	
	C	25.99	20.86	16.55	
	D	51.67	2.83	15.11	
	E	59.25	23.47	21.28	
	F	55.78	34.13	25.87	
	G	46.15	4.51	10.25	
	H	33.11	0.31	14.89	
	I	60.43	55.44	46.69	
	J	60.36	53.04	44.50	
Times New Roman	K	44.99	15.14	4.78	
	L	68.28	45.04	42.20	
	M	34.75	7.52	8.15	
	N	38.[illegible]	7.35	6.06	
	O	45.35	3.68	10.58	
	P	37.7	20.44	7.86	
	Q	56.41	7.15	16.87	
	R	39.65	13.3	1.00	
	S	49.5	21.3	12.04	
	T	59.55	35.48	29.26	
	U	76.26	8.99	35.86	
	V	53.6	10.84	13.16	
	Litera misetericoasa	40.64	13.15	R	
		CALCULATOR			

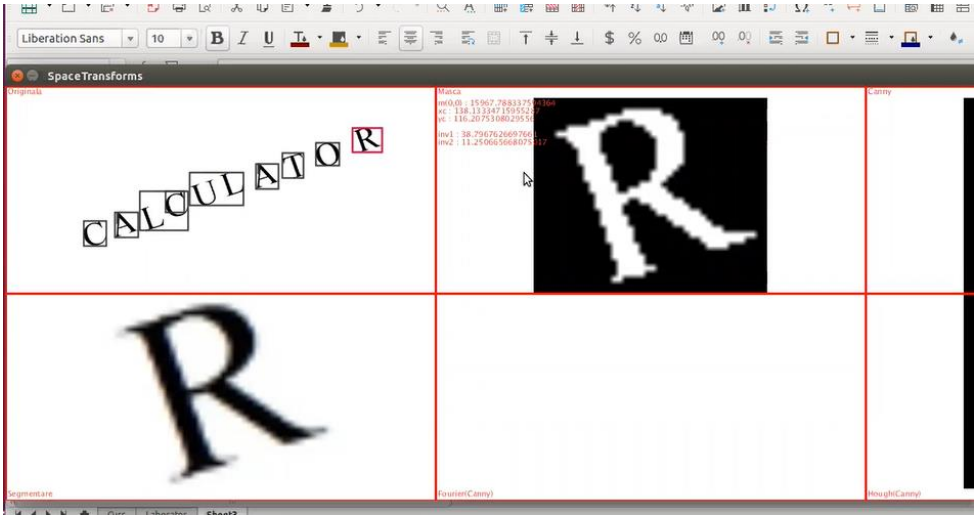
Alt exemplu:

Redimensionam imaginea si o inclinam.





Pentru caracterul R:



=SQRT((C19-C25)*(C19-C25)+(D19-D25)*(D19-D25))

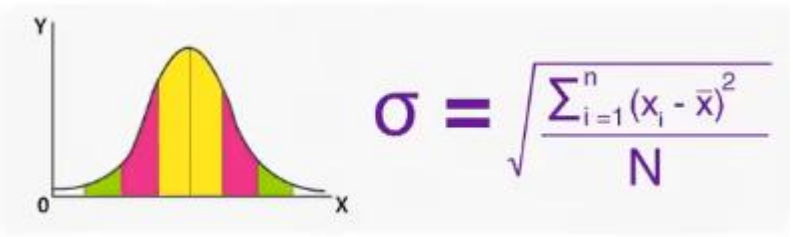
	A	B	C	D	E	F
			inv1	inv2	dist	
	Times New Roman	A	49.39	10.45	10.63	
		B	40.73	7.96	3.82	
		C	25.99	20.86	16.01	
		D	51.67	2.83	15.39	
		E	59.25	23.47	23.83	
		F	55.78	34.13	28.50	
		G	46.15	4.51	9.98	
		H	33.11	0.31	12.33	
		I	60.43	55.44	49.20	
		J	60.36	53.04	47.03	
		K	44.99	15.14	7.32	
		L	68.28	45.04	44.85	
		M	34.75	7.52	5.50	
		N	38.89	7.35	3.90	
		O	45.35	3.68	10.02	
		P	37.7	20.44	9.25	
		Q	56.41	7.15	18.09	
		R	39.65	13.3	2.22	
		S	49.5	21.3	14.69	
		T	59.55	35.48	31.91	
		U	76.26	8.99	37.54	
		V	53.6	10.84	14.82	
		Litera misetericoasa	38.79	11.25		
			CALCULATOR			

Exercitiu:

Faceti aceleasi caractere ca la exercitiul anterior cu fontul Times New Roman si dimensiune 48:

A	B	C	D	E	F
		inv1	inv2	dist	
Times New Roman, size 48	A	49.39	10.45	33.44	
	B	40.73	7.96	39.84	
	C	25.99	20.86	42.14	
	D	51.67	2.83	39.82	
	E	58.59	23.13	18.41	
	F	56.19	35.12	9.00	
	G	59.47	3.5	37.67	
	H	47.6	2.78	41.21	
	I	66.19	61.58	20.83	
	J	59.1	51.54	11.24	
	K	45.44	14.59	31.72	
	L	70.65	45.49	8.87	
	M	40.15	9.44	38.96	
	N	46.2	4.73	39.97	
	O	56.2	4.91	36.73	
	P	45.41	21.98	25.91	
	Q	57.39	7.54	33.93	
	R	38.68	11.32	38.37	
	S	36.64	11.51	39.55	
	T	39.79	21.25	30.48	
	U	76.26	8.99	34.65	
	V	53.6	10.84	31.59	
	Litera misetericoasa	63	41		

Se va îmbunătăți algoritmul de decizie pe baza distanței folosindu-ne de normarea distanțelor cu ajutorul abaterilor pătratice ale momentelor.



Formula: $\theta = \sqrt{\frac{\sum_{i=0}^{N-1}(xi - x\prime)^2}{N-1}}$

Va apărea în felul următor în codul nostru:

float Abatere (float [] minV){

int N = minv.length;

float medie = 0;

for (int i=0; i<N;i++)

medie +=minV[i];

```
float abatere
float abaterePartiala = 0;
for ( int i=0; i<N;i++)
abaterePartiala += (minV[i]-medie)*(minV[i]-medie);
abatere =sqrt(abaterePartiala/(N-1));
return abatere;
}
```