Laborator 4 SRF: Detecția liniilor. Transformata Hough

Scopul lucrării
Scopul acestei lucrări este de familiarizare a studentului cu diversele spații de transformate (Fourier, Hough). După parcurgerea acestui laborator studentul va putea, cu ajutorul transformatei Hough, să realizeze detecția liniilor dintr-o imagine. Studentul va continua de asemenea să aprofundeze folosirea librăriei OpenCV.

1. Breviar teoretic:

În spațiul transformatei Hough o linie apare ca un punct. Practic, acel punct ne dă informații atât despre orientarea liniei cât și despre distanța față de un punct de referință (colțul stânga sus al ecranului).
Trasaturile matematice ale formelor (features) se calculeaza atat in spatiul imaginii, cat si in spatiile transformatelor pe imagine.

Transformata Fourier, descoperită încă din anul 1822, este unul dintre cei mai folosiți algoritmi în domeniul procesării semnalelor, de orice natură ar fi acestea. Cel mai util algoritm care furnizează spațiul transformatei Fourier este algoritmul FFT (Fast Fourier Transform). Transformata Fourier Spatiala este reprezentarea unei imagini (cadru dintr-un video, etc) in spațiul frecventelor spațiale.
Transformata Fourier a unei forme geometrice este invariabila la translația formei în spațiul de bază, dar nu este invariabilă la rotația formei.
În matematica transformata Fourier (numită astfel după matematicianul și fizicianul Joseph Fourier) este o operație care se aplică unei funcții complexe și produce o altă funcție complexă care conține aceeași informație ca funcția originală, dar reorganizată dupa frecventele componente.
De exemplu, dacă funcția inițială este un semnal dependent de timp, transformata sa Fourier descompune semnalul după frecvență și produce un spectru al acestuia. Același efect se obține dacă funcția inițială are ca argument poziția într-un spatiu uni- sau multidimensional, caz în care transformata Fourier relevă spectrul uni- sau multidimensional al frecvențelor spațiale care alcătuiesc funcția de intrare.

Transformata Hough , creată în 1962, era inițial destinată doar detecției si recunoașterii tuturor dreptelor care apăreau intr-o imagine. Ulterior, a fost îmbunătățita (Tr. Hough generalizata, 1972) pentru a putea detecta si forme circulare. Spațiul transformatei Hough are ca si axe de coordonate parametrii dreptelor (fie a si b, fie r->distanta față de coltul din stânga sus si Θ ->unghiul de inclinare al dreptei fata de axa orizontala). **Fiecare dreaptă din spațiul imaginii originale este reprezentata de un singur punct (maxim local al luminozității) in spațiul transformatei Hough.**

Transformata Hough este o metodă ce rezolvă o problemă clasică din viziunea artificială: găsirea dreptelor într-o imagine ce conţine o mulţime de puncte de interes. Metoda directă de a calcula drepte din fiecare pereche de puncte are o complexitate computaţională ridicată, O(n2), şi nu este aplicabilă pentru un număr mare de puncte. Varianta iniţială a fost orientată pe detecţia dreptelor în imagini video, pe baza reprezentării dreptelor ca pantă şi termen liber. Această reprezentare este sub-optimă, deoarece nu este mărginită: pentru a reprezenta toate posibilele drepte din imagine, panta şi termenul liber trebuie să varieze în domeniul -∞ şi +∞.

2. Desfășurarea lucrării

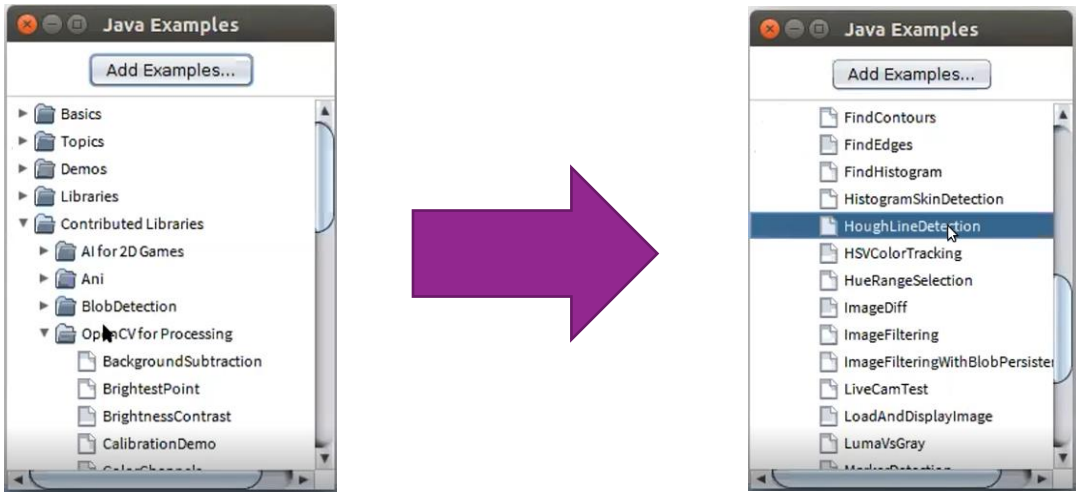
Acest laborator îl vom structura în două parți:

1. În prima parte vom testa un algoritm existent în librăria OpenCV pentru detecția liniilor.
2. În partea a doua vom testa diferite imagini aplicând mai multe tipuri de transformate, prin intermediul unui program original (**SpaceTransforms**).

Partea I

Începem prima parte a laboratorului deschizând programul **Processing** după care se urmează pașii :

- Alegem din colțul stănga sus opțiunea FILE
- Selectăm EXAMPLES..
- Se deschide o fereastră în care sunt expuse mai multe exemple pentru Processing și mergem pe **Contributed Libraries** alegem **OpenCV for Processing**, ulterior selectăm **HoughLineDetection**



Putem recunoaște liniile atât în video/live(utilizând clasele capture/movie) cât si în imagini.

Testul se va face pe o imagine statică (încărcata într-o variabilă de tip PImage).

Exemplul încărcat va trebui modificat astfel încât, la final să arate în felul următor (Fig. 1). Se poate vedea cum unghiul, măsurat față de verticală și marcat în colțul din stânga sus, determină care linii din imagine să fie scoase în evidență.

Figure 1

Programul va trebui să fie capabil să detecteze linii de diferite înclinații. În figura 1 sunt expuse cu verde doar liniile care fac unghiul de 0 grade cu verticala. Prin intermediul tastelor A, respectiv Z putem crește sau scădea unghiul, putând astfel să selectăm linii care fac diferite unghiuri cu verticala.

Figure 2

În figura 2 liniile sunt date de înclinația mâinii.

Figure 3

În figura 3 putem observa că unghiul este setat la valoarea de 90 grade, de unde rezultă și evidențierea cu verde a liniile orizontale.

Mai intai, trebuie sa punem in paralel cu imaginea originală, imaginea pe bază de contururi în partea dreaptă, și bineînțeles, în cadrul imaginii pe bază de contururi trebuie sa marcăm acele linii detectate si recunoscute ca linii.

Indici pentru rezolvarea cerinței :

- Nu putem pune size-ul flexibil în setup, deci va trebui să mai realizăm o functie settings pentru a putea încărca 2 poze în paralel.
- Referitor la funcția **rezise(x,y)**, dacă punem x = 0 și y = orice valoare, atunci x va fi forțat automat astfel încat să se păstreze factorul de aspect al imaginii inițiale.
- Funcționalitatea cu unghiul se implementează prin declararea unei variabile unghi:
  - int unghi=90.
- Memorăm valoarea unghiului în jurul căreia să se facă detecția .
- La afișaj atunci când algoritmul scanează fiecare linie :
  - If(line.angle < radians(90-(unghi-5)) && line.angle > radians (90-(unghi+5))).
  - -5, +5 reprezintă o toleranță.
- Variabila unghi trebuie să fie incrementată/ decrementată în functia keyPressed().
- Pentru a evidenția liniile din imaginea care conține contururile(cea din dreapta) va trebui să adăugăm funcția src.width+

Această detecție a liniilor nu înseamnă că liniile apar pe figură doar ca contur, ci înseamnă că aplicația le recunoaște ca fiind linii, are memorat undeva într-o listă de obiecte atât linia cât și unghiul specific acesteia.

Partea a IIa

In partea a doua vom observa imaginile de test prin prisma transformatelor Fourier si Hough. Putem să facem click pe oricare punct de pe transformate Hough și vom putea observa cărei drepte din imaginea originală îi aparține. Codul sursă specific programului utilizat în cea de-a doua parte a laboratorului este următorul :

```
import gab.opencv.*;


OpenCV opencv;

ArrayList<Line> lines;

PImage src,img1,img2;


void settings(){

  src = loadImage("imagine.jpg");

  src.resize(0, 250);

  size(src.width*3,src.height*2+30);


}


void setup() {


 img1=src.copy();

 img1.filter(GRAY);

 img1.filter(INVERT);

 img1.filter(THRESHOLD, 0.3);



 opencv = new OpenCV(this, img1);

 opencv.findCannyEdges(20, 60);

 img2=opencv.getOutput();







 // Find lines with Hough line detection

 // Arguments are: threshold, minLengthLength, maxLineGap

 lines = opencv.findLines(100, 30, 20);

  noLoop();

}
```

```
void draw() {
  fill(250,50,50);
  textSize(10);
  int xt=3,yt=15;

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

  image(img1,src.width,yt);
  text("Masca",src.width+xt,10);

  // DFT(imagine, luminozitate)
  //image(DFT(img1,5), 0, yt+src.height);
  text("Fourier(masca)", xt, yt+2*src.height+10);

  //image(DFT(img2,10), src.width, yt+src.height);
  text("Fourier(Canny)", src.width+xt, yt+2*src.height+10);

  image(img2,2*src.width,yt);
  text("Canny", 2*src.width+xt,10);

  image(Hough(img2), 2*src.width, yt+src.height);
  text("Hough(Canny)", 2*src.width+xt,yt+2*src.height+10);

  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);

}

PImage DFT(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){
 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*cos(radians(theta))+y*sin(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;
}
```

În momentul în care privim un obiect din mai multe unghiuri, luăm practic mai multe date/informații despre obiect, deoarece îl privim din diferite perspective. Așa și calculatorul, dacă

“privește” o imagine din punctul de vedere al cât mai multor transformate și folosește cât mai mulți algoritmi de decizie pentru detecția și evaluarea obiectelor, cu atât respectiva detecție este mai eficientă și are mai puține rate ale erorilor.

Imaginea utilizată pentru testarea algoritmului este următoarea:

Figure 4

După rularea programului al cărui cod este scris mai sus, apare pe ecran pe rândul de sus imaginea originală, masca (o imagine «binară» segmentată în care obiectul și fundalul au culori diferite pentru a le diferenția) și imaginea contururilor obținută cu ajutorul algoritmului Canny.

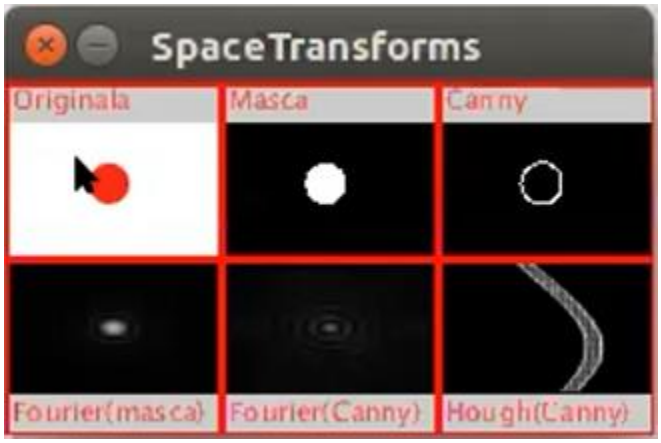


Figure 5

Algoritmul de mai sus nu va afișa imaginea prin TFT deoarece durează foarte mult execuția acestuia. Iar în cele din urmă avem imaginea prin transformata Hough(fiecare dreaptă va fi caracterizată prin distanța de la acel punct la acea dreaptă).

Întrebare : Cum se calculează distanța de la un punct la o dreaptă?

Răspuns : Trebuie să ne folosim de perpendiculara de la acel punct la acea dreaptă.

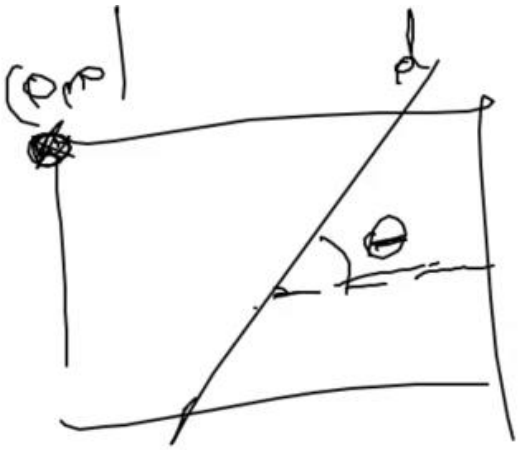


Figure 6

Schimbăm imaginile pentru a evidenția modificările apărute.

Exemplul 1

Figure 7

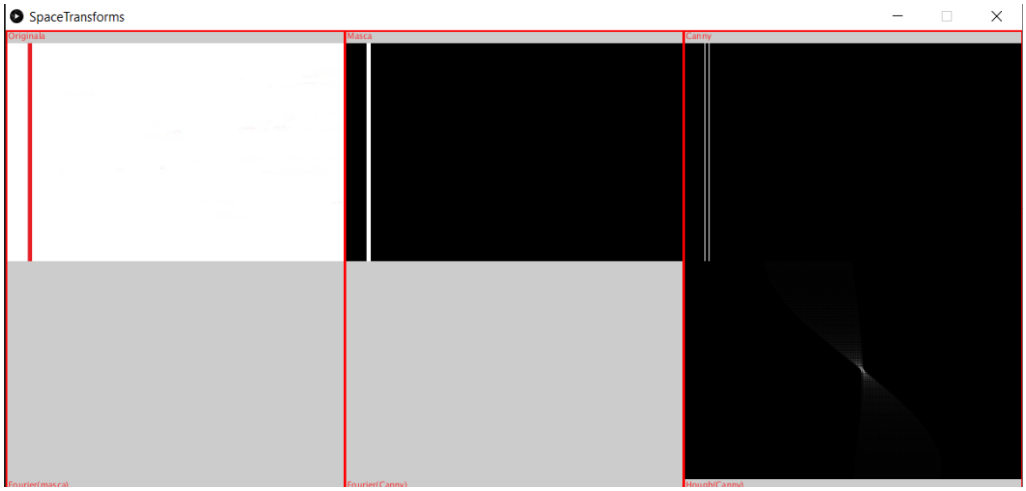


Figure 8

Intrebare: Ce diferă între cele 2 linii în transformata Hough? Distanța față de origine sau unghiul de înclinare?

Răspuns: Diferența constă în distanța față de origine(colțul stânga sus), deoarece cele două drepte sunt paralele și formează același unghi cu orizontala în imaginea inițială.

Exemplul 2

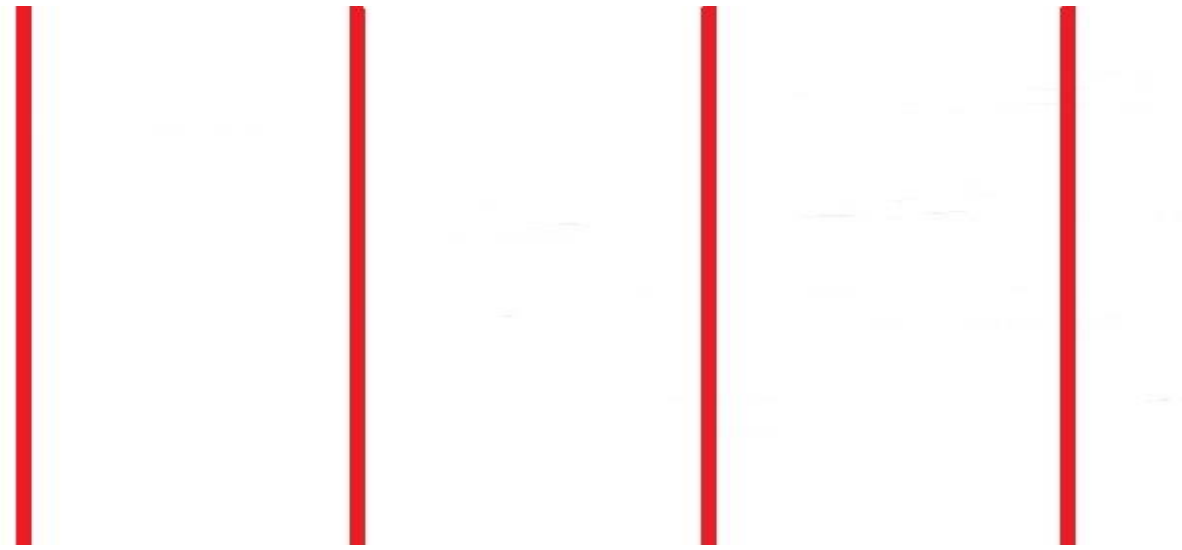


Figure 9

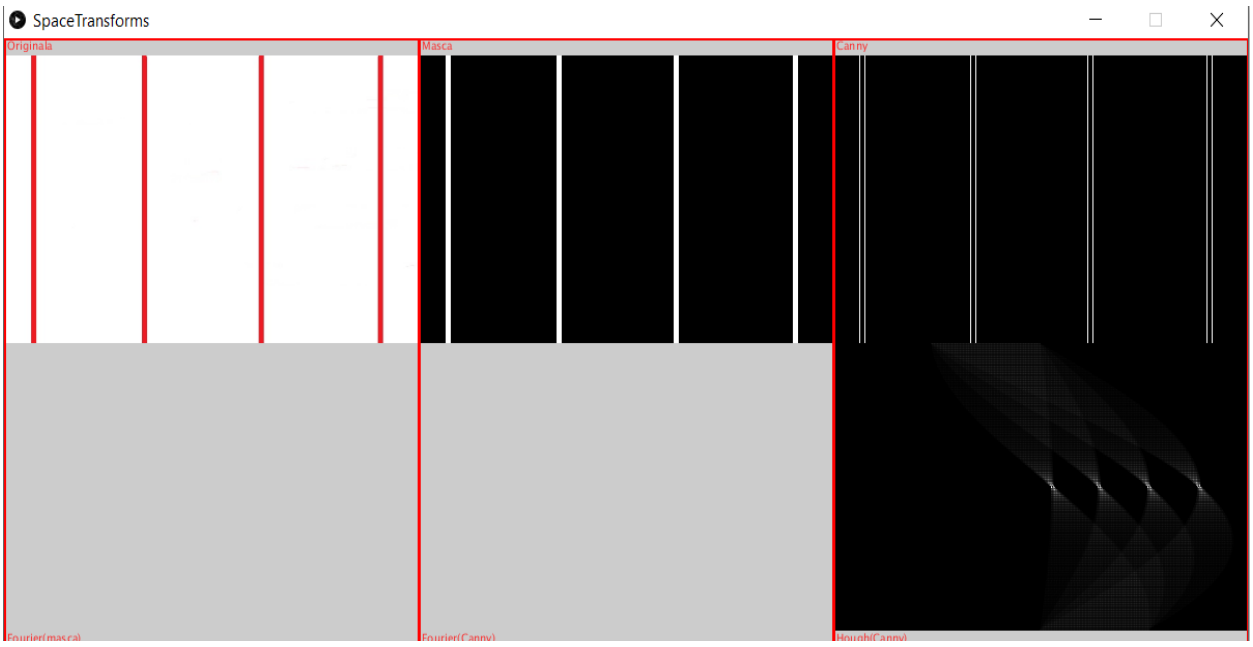


Figure 10

Fiecare pereche de linii (Canny) corespunde în Hough(Canny) la o pereche de puncte . Punctele de intensitate mare ne oferă atât poziția liniilor față de colțul stânga sus precum și înclinația.

Exemplul 3

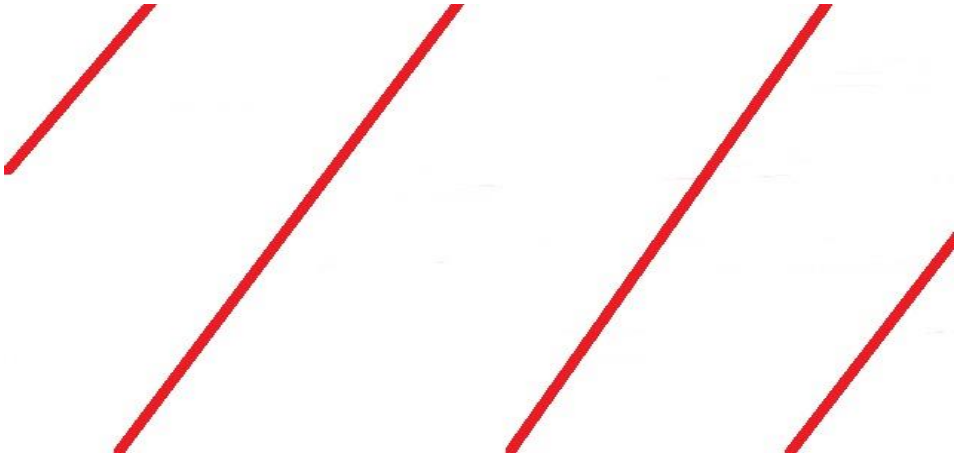


Figure 11

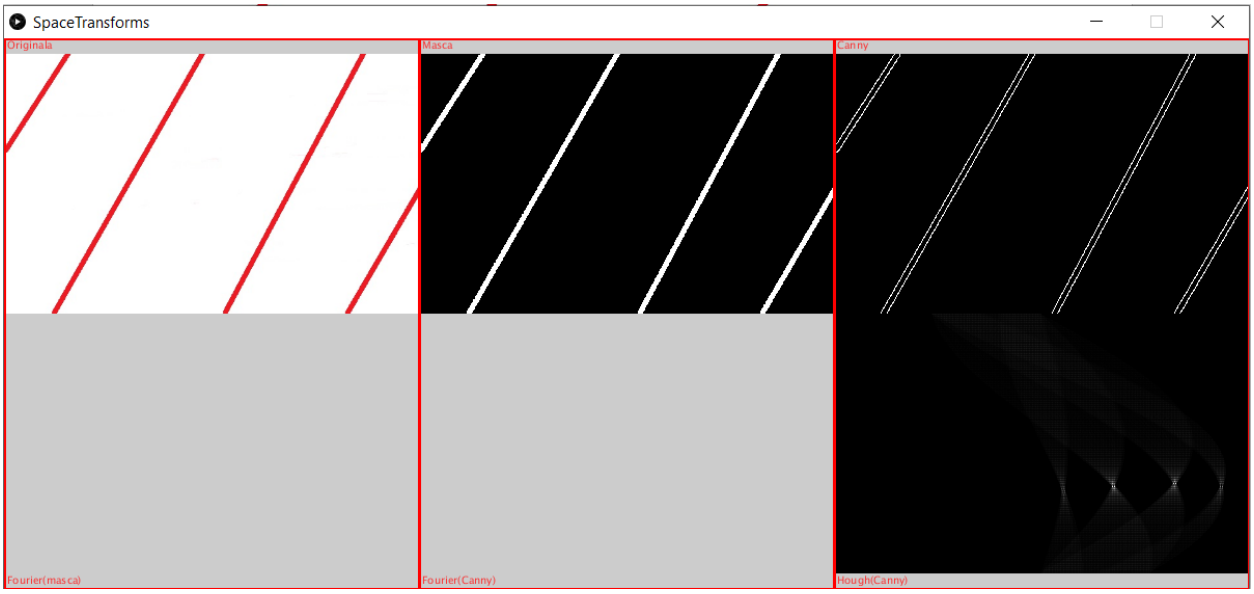


Figure 12

Modificarea a avut loc pe axa verticală(Hough{Canny}). Înclinând liniile, punctele se situeaza mai jos.

Exemplul 4

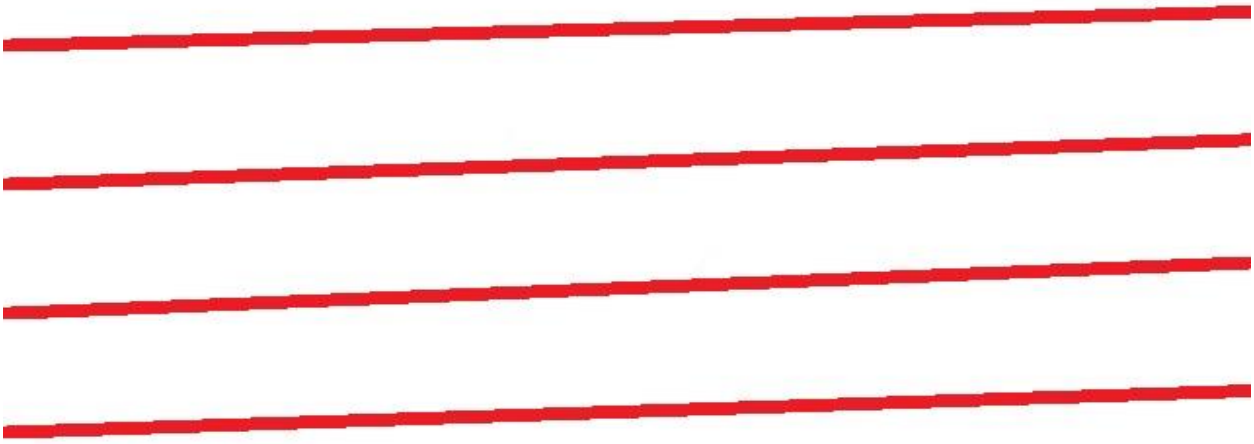


Figure 13

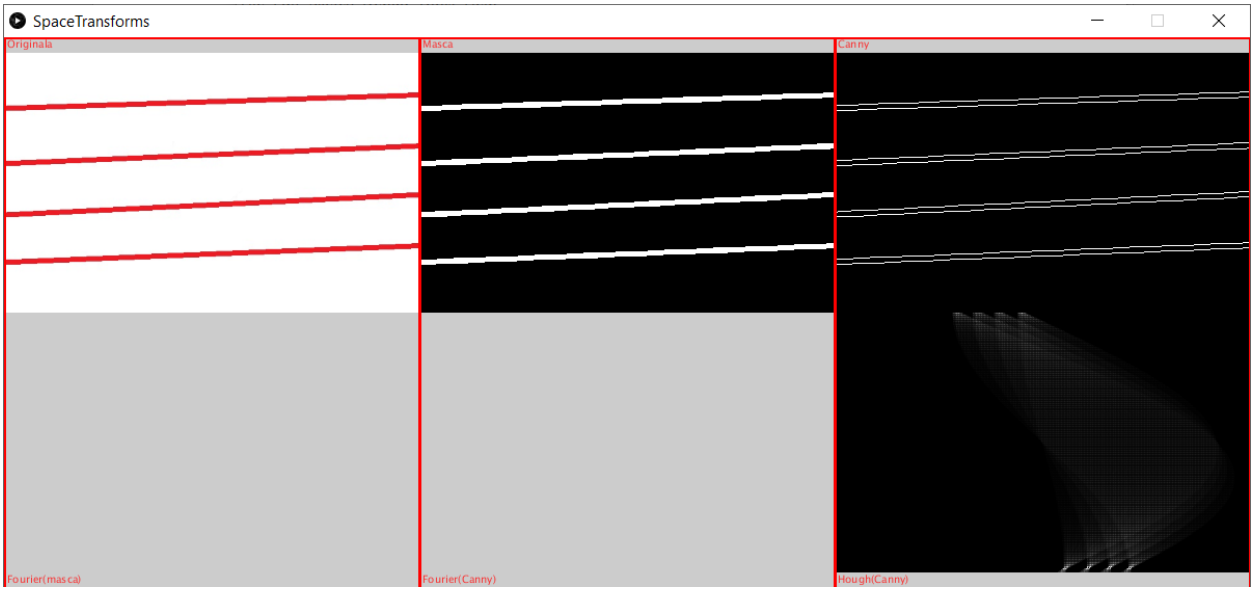


Figure 14

Punctele au migrat și mai mult spre axa de jos. Față de axa orizontală au rămas la fel.

Exemplul 5

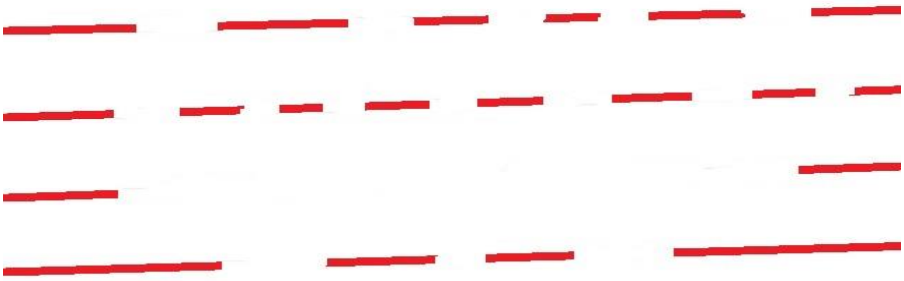


Figure 15

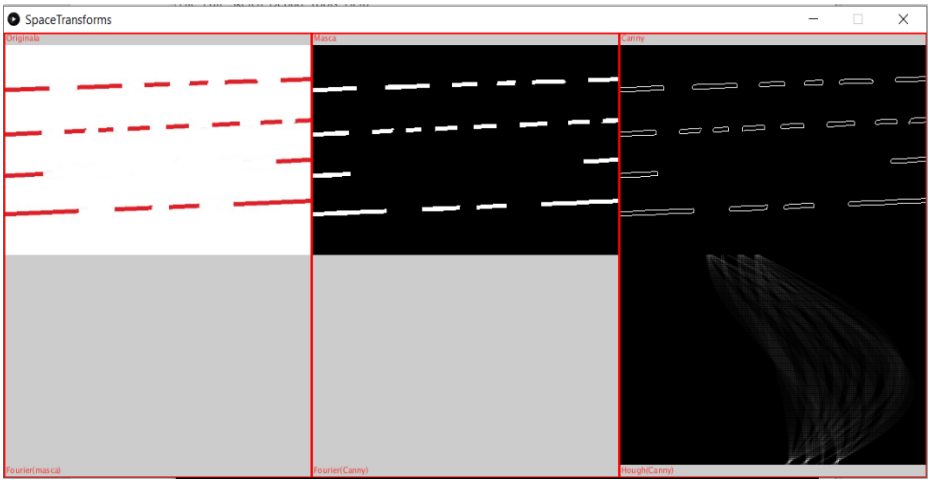


Figure 16

Pozițiile s-au păstrat deoarece distanța față de colțul stânga a rămas la fel, chiar dacă elementele sunt separate ele au aceeași distanță față de colțul stânga sus deoarece are aceeași lungime a perpendicularei. Cu cât lungimea liniei este mai mare cu atât intensitatea punctului este mai mare.

Următoarea imagine de test, trecută prin programul SpaceTransforms este afișată în figura de mai jos.

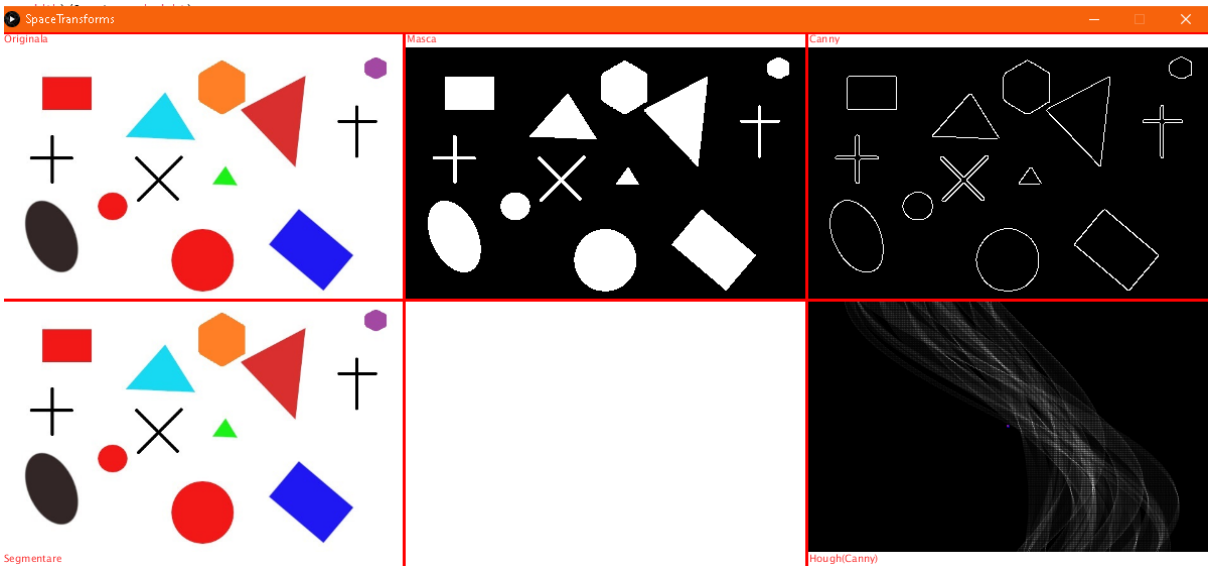


Figure 17

Când facem click asupra punctelor de pe diagrama Hough se vor afisa liniile corespunzatoare in imagina segmentata. Când apăsăm tasta **c** putem șterge liniile deja evidențiate. Se pot selecta si figurile separate din imaginea originala cu un click pe acestea. În momentul selecției se va calcula si transformata Fourier a figurii respective. Se pot selecta si figurile separate din imaginea originala cu un click pe acestea. În momentul selectiei se va calcula si transformata Fourier a figurii respective.

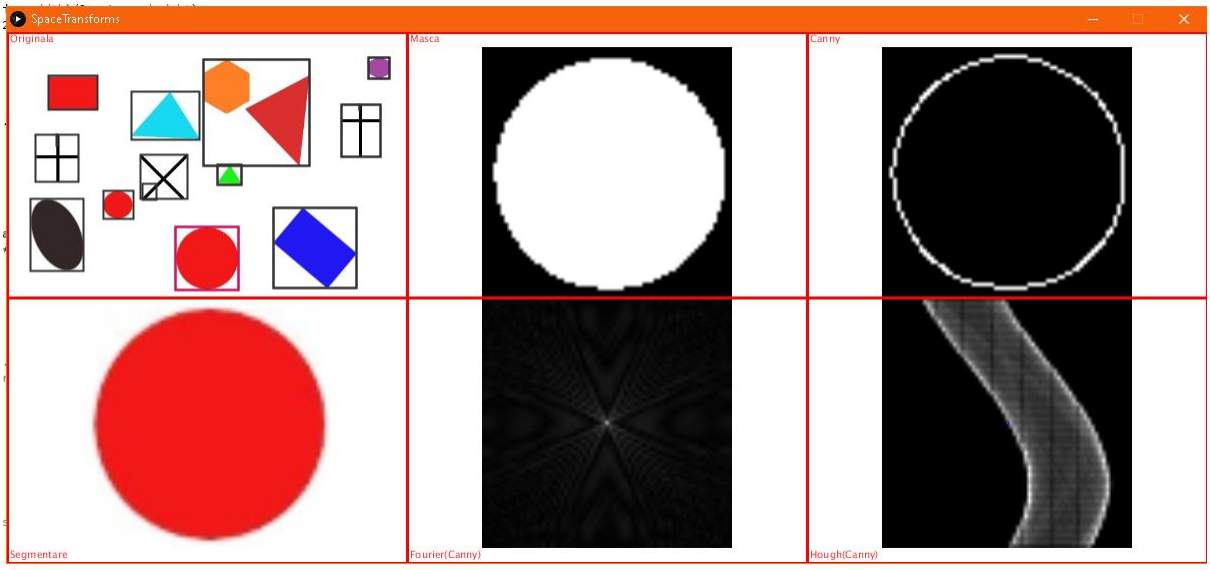


Figure 18