

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI  
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE  
DOMENIUL: Calculatoare și tehnologia informației  
SPECIALIZAREA: Tehnologia informației

## **LUCRARE DE DIPLOMĂ**

Coordonator științific  
Ş.l.dr.ing. Paul Herghelegiu

Absolvent  
Georgiana Cristina Petrea

Iași, 2017

## DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII LUCRĂRII DE DIPLOMĂ

Subsemnatul(a) \_\_\_\_\_, legitimat(ă) cu \_\_\_\_\_ seria \_\_\_\_\_ nr. \_\_\_\_\_, CNP \_\_\_\_\_ autorul lucrării \_\_\_\_\_

elaborată în vederea susținerii examenului de finalizare a studiilor de licență organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași, sesiunea \_\_\_\_\_ a anului universitar \_\_\_\_\_, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 – Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data

Semnătura

# Cuprins

Introducere.....	1
Capitolul 1. Noțiuni teoretice.....	4
1.1. Domeniul și contextul temei.....	4
1.2. Referințe la teme și subiecte similare:.....	5
1.3. Resurse software utilizate:.....	6
1.4. Tipuri de imagini utilizate.....	8
1.4.1. Imagini grayscale.....	8
1.4.2. Modelul de culoare RGB.....	8
1.5. Algoritmi și metode alese:.....	9
1.6. Caracteristici așteptate de la aplicație:.....	10
Capitolul 2. Proiectarea aplicației.....	11
2.1. Extragerea muchiilor din imagini. Noțiuni teoretice.....	11
2.2. Operatorul Sobel.....	12
2.3. Transformata Hough.....	13
2.4. Reprezentarea liniilor în Spațiul Hough.....	14
2.5. Maparea punctelor în Spațiul Hough.....	15
2.6. Coordonate polare.....	16
2.7. Coordonate carteziene.....	17
2.8. Conversia din coordonate polare în coordonate carteziene și invers.....	17
2.9. Ecuăția unei drepte.....	18
2.10. Panta unei drepte.....	18
2.11. Distanța de la un punct la o <i>dreaptă</i> .....	19
2.12. Distanța euclidiană.....	19
Capitolul 3. Implementarea aplicației.....	20
3.1. Filtre liniare. Filtrul gaussian.....	20
3.2. Filtre neliniare. Filtrul median.....	20
3.3. Implementarea Operatorului Sobel:.....	21
3.4. Transformata Hough.....	24
3.4.1. Transformata Hough Standard.....	24
3.4.2. Transformata Hough Standard Probabilistică.....	25
Capitolul 4. Testarea aplicației.....	27
4.1. Elemente de configurare.....	27
4.2. Rezultatele experimentale:.....	27
Capitolul 5. Concluzii.....	31
Bibliografie.....	32
Anexă.....	34

# Detectia trecerii de pietoni in imagini

Georgiana Cristina Petrea

## Rezumat

Scopul acestei lucrări este de a verifica și detecta existența unei treceri de pietoni într-o imagine sau într-un clip video. O astfel de aplicație poate fi una foarte utilă pentru a dezvolta echipamente ce pot ajuta persoanele care au probleme cu vederea, ce necesită îndrumare în special pe stradă, locul destul de aglomerat în zilele noastre. Acest gen de aplicație se adresează, în special acestor persoane.

Trecerea în mod regulamentar a unei străzi, presupune găsirea unei treceri de pietoni semaforizată sau nu. În acest proiect ne vom axa pe detectarea trecerilor de pietoni nesemaforizate. Algoritmul își propune detectarea liniilor de pe carosabil și în funcție de anumiți parametri precum: lungime, numărul de linii, culoare, utilizatorul va primi un mesaj în legătură cu existența unei treceri de pietoni.

Ca și resurse software am utilizat Visual Studio cu limbajul C++ și OpenCV.

Obiectivul principal al programului este detectarea liniilor paralele și verificarea lungimii segmentelor detectate, verificarea numărului de linii și în funcție de anumiți parametri prestabiliti se va putea decide dacă există sau nu o trecere de pietoni în imaginea urmărită.

Lucrarea este structurată pe mai multe capituloare, introducerea prezintă contextul abordării temei, motivația alegerii temei, obiectivele și metodologia utilizată. Capitolul de fundamente teoretice descrie conceptele și formulele matematice utilizate în implementarea algoritmului. În proiectarea aplicației sunt prezentate noțiuni teoretice despre filtrul Sobel, Gaussian, median, despre Transformata Hough, componentă importantă în realizarea proiectului, iar în implementarea aplicației sunt prezentate detaliat anumite componente din program, modalitatea de determinare a segmentelor și filtrele implementate de mine.

În final se pot observa imaginile de test și rezultatele obținute urmând apoi concluziile lucrării.

Această aplicație poate reprezenta un prim pas în realizarea unei aplicații complexe de detectie a trecerii de pietoni, în timp real, aplicație Android sau iOS, astfel încât să fie la îndemâna oricărei persoane.

## Introducere

Principalul obiectiv al acestei lucrări este de a detecta trecerea de pietoni nesemaforizată în imagini. Acest tip de aplicație are ca și interes principal dezvoltarea de echipamente ce pot ajuta persoanele ce au nevoie de ajutor pe stradă, în locuri aglomerate și nu numai.

Motivul pentru care am ales ca și temă pentru lucrarea de licență detecția trecerii de pietoni, este faptul că în zilele noastre traficul în orașele mari și nu numai este din ce în ce mai aglomerat, pe străzi sunt foarte multe mașini, stilul nostru de viață este unul destul de haotic și din păcate mai devreme sau mai târziu cu toții avem de suferit din această cauză.

Dacă persoanele ce sunt în stare bună de sănătate se descurcă în diferite situații în oraș, trebuie să ne gândim și la persoanele mai puțin norocoase, care au diferite tipuri de probleme, precum deficiențe de vedere. Potrivit statisticilor realizate de Organizația Mondială a Sănătății aproximativ 40 de milioane de oameni sunt orbi în toată lumea. O modalitate de a traversa în siguranță strada este extrem de necesară pentru a îmbunătăți mobilitatea și independența acestor persoane. În primul rând, pietonii au nevoie să cunoască localizarea trecerii de pietoni, dacă este în fața sa, sau nu, apoi cunoașterea unor detalii legate de lungimea sa, sau starea traficului la momentul respectiv ar fi informații de mare folos.

O astfel de aplicație, de recunoaștere a trecerii de pietoni, vine în ajutorul oamenilor, făcându-le viața cotidiană un pic mai sigură întrucât cunoașterea mediului înconjurător, un mod independent de a circula pe stradă sunt lucruri de o importanță vitală în viața oricărei persoane. Pentru a îmbunătăți abilitățile de înțelegere și explorare a mediului pentru oamenii cu deficiențe de vedere, au fost dezvoltate diferite dispozitive sonore cu scopul de a ajuta în detectarea obstacolelor, a găsi anumite căi, a face viața mai ușoară.

Siguranța pietonilor a devenit din ce în ce mai îngrijorătoare datorită faptului că tot mai mulți oameni aleg a fi pietoni, bicicliști având în vedere traficul din orașe și poluarea din zilele noastre. În anul 2003 NHTSA<sup>1</sup> a raportat o estimare privind faptul că 70 000 de pietoni au fost răniți și 4749 de pietoni au fost omorâți în accidente de mașini. Mai precis, fatalitățile produse pe șosele, în care au fost implicați și pietoni ajung la 90% din totalul accidentelor, indiferent de tipul de autovehicul. În anul 2007, 24.3% din toate accidentele cu pietoni în Statele Unite ale Americii s-au petrecut în intersecții. Și în privința transportului profesionist sunt din păcate creșteri în ceea ce privește accidentarea pietonilor. Unele state, printre care și Minnesota au început să adopte planuri pentru siguranța pietonilor, a biciclistilor și a motociclistilor.

În consecință, inginerii au început să implementeze tipuri noi de străzi și design nou pentru intersecții cu scopul de a reduce riscul tamponărilor și a accidentelor. Obiectivul acestui studiu este de a testa un set de unelte pentru a extrage și clasifica evenimentele în care sunt implicați pietonii, în diferite intersecții, ajutându-i astfel pe inginerii care își doresc să monitorizeze și să caracterizeze evenimentele din trafic. Sensurile giratorii sunt în atenția inginerilor, deoarece numărul lor a crescut într-un mod semnificativ după încercarea de modernizare a străzilor. Design-ul nou al acestor tipuri de intersecții este menit pentru a canaliza și a încetini viteza șoferilor, lucru ce s-a și observat, iar natura izbiturilor tinde să fie mai puțin severă. Un studiu în SUA a raportat faptul că schimbarea intersecțiilor cu semafoare sau semne de circulație în sensuri giratorii, a redus accidentele soldate cu răniți cu 80% și toate tipurile de accidente cu 40%. Tot rezultate satisfăcătoare au fost observate și în Europa și Australia.[1]

1 NHTSA=National Highway Trafic Safety Administration(Administrația Siguranței Traficului pe Șosele)

Îmi propun în această lucrare să realizez detectarea de treceri de pietoni în imagini RGB. Algoritmul de detecție cuprinde o serie de pași ce sunt punctați în figura următoare.

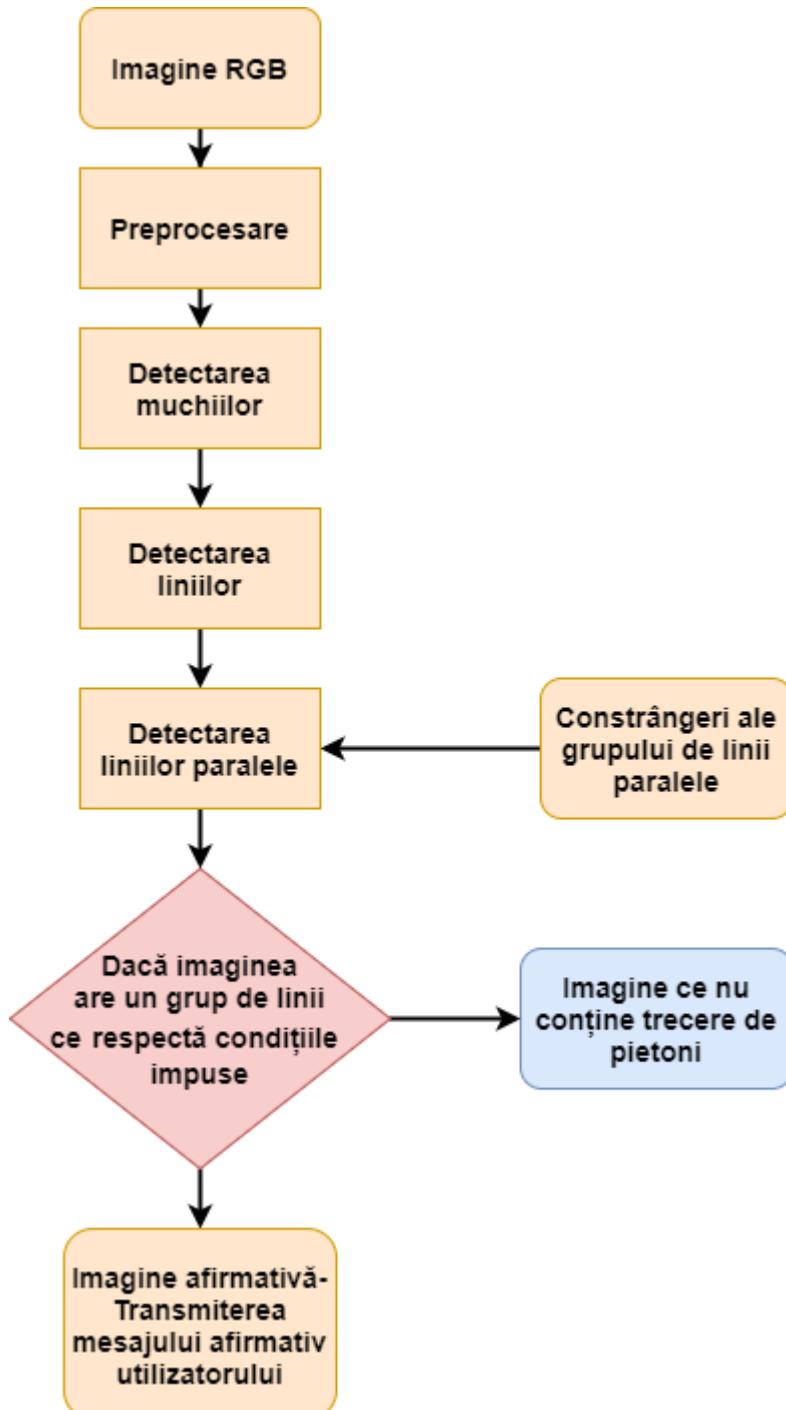


Figura 1: Schema algoritmului de detecție a trecerii de pietoni

Fiecare bloc din această figură reprezintă o operație ce se aplică imaginii. În faza de preprocesare, se transformă imaginea dintr-o imagine color într-o imagine alb-negru, se filtrează imaginea dacă este cazul, aplicându-se un filtru median pentru eliminarea zgomotelor. Apoi urmează detectarea muchiilor, iar eu am ales să implementez operatorul Sobel, aplicându-se în cele ce urmează Transformata Hough pentru extragerea liniilor din imagine. În funcție de parametrii ce reies după aplicarea acestei funcții pot calcula ecuațiile dreptelor de interes, distanța de la celelalte puncte din imagine la dreaptă, verific culoarea pixelilor, lungimea segmentelor detectate și le desenez pe imaginea inițială. Pentru verificare, utilizez și Transformata Hough Probabilistică. Dacă imaginea conține un grup de linii paralele atunci utilizatorul va primi un mesaj afirmativ.

Această lucrare este structurată pe cinci capitole, după cum urmează:

- În capitolul întâi sunt prezentate noțiunile teoretice despre acest proiect, informații despre domeniul temei, referințe la teme similare și voi descrie câteva lucruri despre ce s-a făcut până acum despre această temă, voi prezenta resursele software utilizate și foarte important algoritmul ales pentru implementare dar și caracteristicile așteptate de la această aplicație.
- Capitolul doi reprezintă proiectarea aplicației și aici voi detalia teoria din spatele fiecărui pas din procedeul realizat. Extragerea muchiilor din imagini, câteva lucruri importante despre Operatorul Sobel, Transformata Hough, reprezentarea liniilor în spațiul Hough și noțiuni de geometrie: reprezentarea în coordonate polare, în coordonate carteziene și conversia dintre cele două. De asemenea voi scrie despre ecuația unei drepte, panta, distanța de la un punct la o dreaptă și despre distanța euclidiană.
- Al treilea capitol conține implementarea aplicației, voi scrie exact ce filtre am implementat cu referire la codul din program. Se vor regăsi informații despre implementarea filtrelor gaussian, median, sobel, imagini pentru exemplificare, detalii și explicații ale codului scris. Apoi urmează Transformata Hough, Transformata Hough Probabilistică și detalii despre modul de funcționare.
- Capitolul patru prezintă testarea aplicației cu elementele specifice de configurare și rezultatele experimentale ale programului, regăsindu-se diferite imagini cu rezultate.
- În ultimul capitol, se vor prezenta cele mai importante concluzii din lucrare pornind de la obiectivele propuse.

## Capitolul 1. Noțiuni teoretice

### 1.1. Domeniul și contextul temei

Cunoașterea mediului înconjurător în viața de zi cu zi are o importanță vitală pentru fiecare persoană, iar tehnologia vine în ajutorul celor ce au nevoie. Printre domeniile ce sunt necesare în realizarea și dezvoltarea unui dispozitiv, se numără și prelucrarea de imagini, indispensabil din punctul meu de vedere atunci când se dorește perceperea lucrurilor din jur. În zilele noastre se dezvoltă din ce în ce mai multe aplicații ce trebuie să perceapă cât mai multe detalii din mediul înconjurător, detectie de fețe, de mașini.

Procesarea imaginilor(eng. Image Processing) presupune studiul proprietăților imaginilor și transformarea acestora. Majoritatea algoritmilor de vizuire artificială necesită folosirea unor algoritmi de procesare a imaginilor. Exemple de metode:

- îmbunătățirea calității imaginilor- prin transformarea imaginilor, punerea în evidență a detaliilor ascunse/obscure, a trasăturilor de interes.
- compresia- reprezentarea compactă a imaginilor/secvențelor pentru transmisie și stocare.
- restaurarea- eliminarea elementelor de degradare cunoscute.
- extragerea de trasăuri- localizarea anumitor șabloni: muchii, colțuri.[2]

Procesarea informațiilor conținute într-o imagine este strâns legată de procesarea de semnale în general. Imaginele se găsesc cel mai adesea sub forma unor fișiere în format binar. În cel mai simplu caz, când nu se aplică algoritmi de compresie, imaginile conțin câte o valoare sau un set de valori numerice pentru fiecare pixel constituit. Aceste valori sunt utilizate de programele de prelucrare și afișare a imaginilor pentru a identifica și compune culorile pixelilor corespunzători.

Trecerea de pietoni(zebra) reprezintă un set uniform de linii albe, paralele pe o suprafață de culoare închisă(negru sau nuanțe de gri). Fiecare componentă este în formă unui dreptunghi, sau în cazul trecerii în diagonală poate fi în formă unui paralelogram. În Statele Unite ale Americii s-au format anumite standarde, precum: lățimea trecerii să fie de cel puțin 180 cm, iar grosimea benzilor albe cuprinsă între 15 cm și 60 cm. În ceea ce privește tipul marcărilor ce sunt utilizate pentru a defini o trecere, acestea sunt multiple. În SUA, cel puțin două tipuri de marcaje sunt valabile. Zona de traversat este constituită și din două linii albe , perpendicularare pe direcția șoselei cu lățimea cuprinsă între 15 cm și 60 cm, iar distanța dintre aceste două linii este de cel puțin 180 cm. Trecerea de pietoni nesemafonizată,cunoscută și sub numele de „continental crossing ” în SUA, poate fi vizual detectată de la mare distanță, în comparație cu alte marcaje în același luminozitate. Datorită importanței sale, aceasta este des întâlnită în apropierea școlilor și spitalelor, dându-le șoferilor un plus de atenție, fiind obligați să reducă viteza autovehiculului. [3]

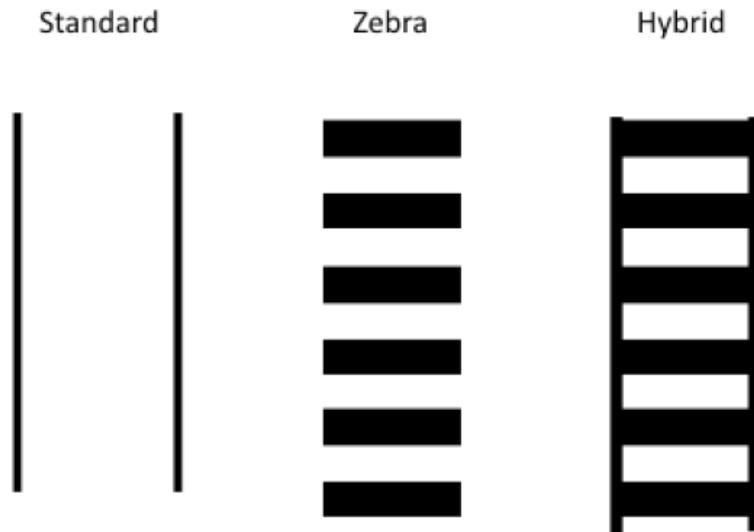


Figura 1.1: Câteva tipuri de treceri de pietoni[0]

### **1.2. Referințe la teme și subiecte similare:**

În ultimii ani au fost realizate numeroase studii în ceea ce privește pietonii, semnele de circulație de pe stradă, mașinile, intersecțiile, iar câteva exemple de astfel de proiecte ar fi:

- Recunoașterea automată a plăcuței de înmatriculare a unui autovehicul
- Aplicații de recunoaștere a semnelor de circulație
- Aplicații ce ajută la verificarea stării traficului la un moment dat.

Predispoziția spre acest gen de aplicații și proiecte este de la sine înțeleasă deoarece în zilele noastre numărul mașinilor crește de la an la an și din păcate crește și numărul de accidente, ce poate, pot fi prevenite cu ajutorul unor bune aplicații. În ceea ce privește detectarea trecerii de pietoni studiile făcute pe acest subiect se împart în mai multe categorii în funcție de tipul imaginii și a algoritmului folosit, iar câteva dintre acestea sunt:

- Coughlan a dezvoltat o metodă de a găsi trecerea de pietoni bazându-se pe segmentarea imaginii(eng. figure-ground segmentation), în care se construiește într-un model grafic pentru a grupa caracteristicile geometrice într-o structură coerentă.[4]
- Ivanchenko a extins algoritmul pentru a detecta localizarea și orientarea trecerii de pietoni pentru persoanele nevăzătoare utilizând o cameră a unui telefon mobil. Prototipul sistemului poate lucra în timp real pe Nokia N95. Telefonul preia în mod automat câteva imagini pe secundă, analizând fiecare imagine într-o fracțiune de secundă și răspunde cu un semnal sonor dacă detectează trecerea de pietoni.[5]
- Advanyi a utilizat ochelarii bionici pentru a furniza nevăzătorilor și indivizilor cu deficiențe de vedere, orientarea, bazându-se pe procesarea culorilor și îmbunătățirea lor. Apoi detectia trecerii de pietoni a fost realizată cu succes prin algoritmul Cellular Nanoscale Networks.[6]
- Se, de asemenea, a propus o metodă de detecție a trecerii de pietoni. Mai întâi detecta liniile trecerii, găsind un grup de linii paralele. Muchiile erau apoi extrase utilizând diferențele de variații.[7]
- Omachi a propus o detectare a semnelor de circulație din imagini folosind forma și culoarea informațiilor recepționate. Metoda a fost îmbunătățită introducând Transformata Hough.[8]

- Un alt proiect bazat pe identificarea și localizarea trecerii din satelit este dezvoltat de către Dragan Ahmetovic și Roberto Manduchi utilizând un algoritm de computer vision și Google Maps, cu o precizie foarte bună.[9]

### 1.3. Resurse software utilizate:

Ca și resurse software principale în realizarea programului sunt Visual Studio 2015 și OpenCV(Open Source Computer Vision).

Utilizez Microsoft Visual Studio fiindcă include un set complet de instrumente de dezvoltare pentru generarea de aplicații desktop și mobile în diferite limbi, cel utilizat fiind C++. Aceasta este un limbaj multi-paradigmă, cu verificarea statică a tipului variabilelor ce suportă programarea procedurală, abstractizare a datelor, programare orientată pe obiecte, devenind unul dintre cele mai populare limbi de programare.

OpenCV este o bibliotecă de funcții open source<sup>2</sup> scrisă în C și C++. Biblioteca a fost proiectată astfel încât să ofere eficiență computațională pentru aplicații de computer vision<sup>3</sup> în timp real. Din proiectul inițial făceau parte aspecte legate de urmărirea razelor de lumină și desenarea 3D a pereților. Principalii contribuitori la dezvoltarea proiectului au fost programatorii de la Intel și o parte din experții în procesare și analiza imaginilor de la Intel Rusia. Aceasta este gratuit de utilizat sub o licență BSD<sup>4</sup>. Biblioteca OpenCV conține peste 500 de funcții care acoperă domenii precum imagistică medicală, securitatea, calibrarea camerelor fotografice, vedere stereo și robotică. Suplimentar OpenCV conține un set complet de funcții de machine learning<sup>5</sup>(MLL). OpenCV are mai mult de 47000 de utilizatori în comunitatea proprie și se estimează un număr de 14 milioane de descărcări.

Structura bibliotecii OpenCV este prezentată în următoarea figură:

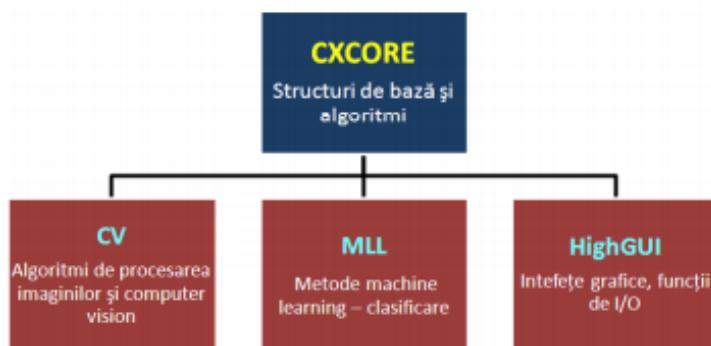


Figura 1.2: Structura bibliotecii OpenCV[12]

Acum OpenCV este disponibil în mod gratuit de la adresa de Internet: <http://opencvlibrary.SourceForge.net/> OpenCV poate rula pe Linux, Windows și Mac OS X, având interfețe precum Python, Matlab, Ruby și altele.[10]

Scopul bibliotecii este de a pune la dispoziție potențialilor utilizatori o infrastructură de procesare a imaginilor, ușor de folosit ce poate fi utilizată în dezvoltarea rapidă a unor aplicații

- 2 Open source=(ro. Sursă deschisă) descrie practica de a produce anumite produse finite, permitând accesul utilizatorilor să acționeze liber asupra procesului de producție sau dezvoltare.
- 3 Computer vision= este un domeniu care studiază dezvoltarea de metode și sisteme computaționale capabile să percepă lumea din imagini și înregistrări într-un mod intelligent.
- 4 BSD=(eng. Berkeley Software Distribution) reprezintă o familie permisivă de licențe software.
- 5 Machine learning= reprezintă tipul de inteligență artificială care redă PC-urilor și dispozitivelor abilitatea de a învăța în mod automat.

complexe. Funcțiile definite sunt numeroase, diversificate și foarte utile în procesarea de imagini, iar câteva exemple sunt:

- Filtrul Gaussian: *GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX, double sigmaY=0);*
- Filtrul Median: *median Blur(InputArray src, OutputArray dst, int ksize);*
- Calcularea Laplacian-ului imaginii: *Laplacian(InputArray src, Output Array dst, int ddepth, int ksize=1, double scale=1, double delta=0).[11]*

Ca și modalitate de stocare a obiectelor am utilizat clasa Mat. Această clasă este o clasă pentru manipulare a matricelor generice bidimensionale, sau a matricelor cu mai multe dimensiuni. Câmpurile importante ale clasei Mat sunt:

1. rows- numărul de rânduri ale matricei= înălțimea imaginii
2. col- numărul de coloane ale matricei= lățimea imaginii
3. data- un pointer la locația din memorie a pixelilor imaginii

Citirea unei imagini din fișier și stocarea acestuia într-un obiect de tip Mat se face folosind funcția *imread*: *Mat image=imread(„path\_to\_image”, flag);* Primul parametru este calea relativă sau absolută spre fișierul imagine iar al doilea parametru poate fi:

1. CV\_LOAD\_IMAGE\_UNCHANGED (-1) – deschide imaginea în același format în care este salvată pe disc;
2. CV\_LOAD\_IMAGE\_GRAYSCALE (0) – deschide imaginea ca imagine monocromă (grayscale); încărcarea face conversia la 8UC1 (1 canal, unsigned char);
3. CV\_LOAD\_IMAGE\_COLOR (1) – se încarcă imaginea și se convertește la 8UC3 (3 canale unsigned char);

#### Accesarea datelor din imagine:

Elementele din matrice sunt indexate conform convenției standard din matematică. Acest lucru înseamnă că originea va fi în colțul stânga sus al imaginii, primul parametru va indica rândul (crescător în jos), iar al doilea parametru va indica coloana (crescător spre dreapta). Următoarea figură ilustrează schema de indexare:



Figura 1.3: Indexarea elementelor în OpenCV

Pentru a accesa pixelul de la coordonatele (i, j) dintr-o imagine monocromă img, se va folosi metoda *at*: *unsigned char pixel = img.at<unsigned char>(i,j);* Trebuie specificat tipul de date care sunt stocate în matrice(*unsigned char*).

### Vizualizarea unei imagini

Pentru a vizualiza o imagine se folosește funcția *imshow*, urmată de un apel al funcției *waitKey*:

```
imshow("image", img);
waitKey(0);
```

Acste linii de cod vor afișa imaginea într-o fereastră nouă, și apoi programul va aștepta ca utilizatorul să apese o tastă. Funcția *waitKey* are un singur parametru: cât de mult să aștepte după acțiunea utilizatorului (în milisecunde). Valoarea zero înseamnă că sistemul va aștepta un timp infinit.[12]

## 1.4. Tipuri de imagini utilizate

### 1.4.1. Imagini grayscale

Cel mai simplu caz este cel al imaginilor binare, care au doar două valori posibile pentru fiecare pixel. Aceste valori corespund nuanțelor de alb și negru. Imaginile alb-negru au asociată câte o singură valoare pentru fiecare pixel(adesea aceste valori sunt pe 8 biți fiecare, adică aparțin domeniului 0-255). Valoarea unui pixel corespunde intensității imaginii în punctul asociat aceluui pixel. Așadar, nu se poate vorbi de culori propriu-zise în acest caz, fiind vorba de puncte sau regiuni cu diverse nivele ale intensității. Intensitatea maximă corespunde nuanței de alb, cea minimă nuanței de negru, în timp ce intensitățile intermedie iau diverse nuanțe de gri(de aici și termenul de imagine grayscale.)

### 1.4.2. Modelul de culoare RGB

Acest model de culoare îl voi folosi pentru imaginile de intrare. În acest caz, culoarea unui pixel are la bază trei componente: roșie, verde și albastră(Red, Green, Blue). Culoarea efectivă se obține prin combinarea celor trei componente, care pot fiecare avea diferite intensități. Așadar, unui pixel dintr-o imagine RGB îi sunt asignate 3 valori, cel mai adesea întregi fără semn pe 8 biți. Numărul de biți alocați pentru valorile culorii unui pixel reprezintă adâncimea de culoare(color depth). Caracteristicile dimensionale ale unei imagini sunt constituite atât din numărul de pixeli de pe axele orizontală și verticală(lățime și înălțime), cât și de o a treia dimensiune, non-geometrică, reprezentată de această adâncime de culoare.

O imagine RGB este constituită atât din caracteristicile ei geometrice sau non-geometrice, cât și din datele efective, reprezentate printr-o matrice cu aceleași dimensiuni ca și numărul de pixeli de pe fiecare axă, unde pentru fiecare element există trei valori care corespund celor trei componente de culoare. Astfel, prelucrarea unei imagini RGB presupune modificarea valorilor componentelor de culoare ale tuturor pixelilor sau ale anumitor pixeli, pentru a se obține efectul dorit.[13] Modelul de culoare RGB este prezentat în imaginea următoare.

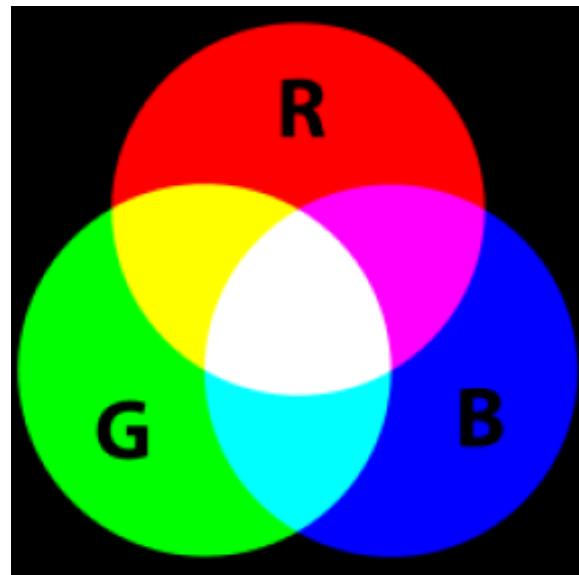


Figura 1.4: Modelul de culoare RGB[13]

### ***1.5. Algoritmi și metode alese:***

Obiectivul general al lucrării este de a detecta trecerea de pietoni nesemaforizată în imagini RGB și secvențe video. Algoritmul de detecție cuprinde operatorul Sobel ce are rolul de a detecta muchiile obiectelor, apoi va fi aplicată Transformata Hough pentru a extrage liniile. Constraințele grupului de drepte paralele ne vor indica dacă există în imagine o trecere de pietoni. Algoritmul lui Hough este cel mai important algoritm ce îl utilizăm deoarece interesul nostru este să identificăm liniile în imagini.

Pașii parcursi în realizarea algoritmului sunt următorii:

- Pas1: Detectarea muchiilor din imagine;
- Pas2: Evaluarea transformatei Hough a imaginii RGB pentru a obține  $r$  și  $\theta$ , unde:  
 $r$ =distanța dintre punctul de start și punctul de sfârșit a liniei(lungimea liniei)  
 $\theta$ =unghiul dintre marginea liniei și direcția orizontală
- Pas3:Computarea ecuațiilor dreptelor detectate;
- Pas4:Găsirea pixelilor ce formează dreptele de interes;
- Pas4:Extragerea liniilor din imaginea RGB;
- Pas5:Detectarea unui grup de lini î paralele bazate pe anumite constrângeri de lungime, lățime, numărul de lini detectate.

Toate acestea vor contribui la decizia finală și anume: există, sau nu o trecere de pietoni în imagine.

### **1.6. Caracteristici așteptate de la aplicație:**

Rezultatul dorit al aplicației este identificarea trecerii de pietoni în cel mai bun mod și în cel mai scurt timp. Inițial nu vor fi luăți în calcul factori exteriori precum lumina foarte slabă, ceața, dar în funcție de rezultatele obținute se vor testa diferite posibilități. Detectarea se va realiza inițial în imagini, apoi în urma rezultatelor se va încerca în video.

Ca și pași intermediari așteptați de la aplicație sunt:

- identificarea în mod corect și cât mai precis a conturului obiectelor prin implementarea filtrului Sobel,
- identificarea liniilor utilizând Transformata Hough,
- calcularea ecuațiilor dreptelor detectate,
- calcularea distanțelor de la fiecare punct din imagine la dreapta identificată
- verificarea culorii pixelilor pe liniile detectate
- calculul lungimii segmentelor

Și în funcție de anumiți parametri se va putea stabili dacă există sau nu o trecere de pietoni în imagine.

## Capitolul 2. Proiectarea aplicației

### 2.1. Extragerea muchiilor din imagini. Noțiuni teoretice

Detectia frontierelor este o problema fundamentală în prelucrarea imaginilor, deoarece permite extragerea informației utile dintr-o imagine, necesară în analiza imaginilor.

Frontierele sunt zone din imagine caracterizate prin variații brusă ale intensității. Ele pot reprezenta obiecte subțiri(linii) sau contururi ale unor zone din imagine cu caracteristici de intensitate similară.

Majoritatea metodelor de detectie a frontierelor presupun efectuarea următoarelor etape:

1. Detectia pixelilor de front: Un pixel de front(pe scurt, front) este un pixel în care intensitatea se schimbă brusc. Din păcate, nu orice pixel de front este pixel de frontieră. El poate fi un punct de zgromot. Din acest motiv, unele detectoare de pixeli de front operează asupra unei imagini netezite. Dar, filtrele de netezire diminuează nu numai intensitatea punctelor de zgromot ci și pe aceea a pixelilor de frontieră îngreunând detectia punctelor de front.
2. Se elimină dintre pixelii de front aceia care, pe baza unor criterii, nu sunt pixeli de frontieră.
3. Se conectează pixelii de frontieră pentru a forma contururi(frontiere).

Pentru conectarea pixelilor de frontieră se folosesc două tipuri de metode:

- Metode locale, care folosesc relațiile(bazate pe intensitate) ale fiecărui pixel din imagine cu pixelii vecini. Frontieră este construită iterativ, într-un proces de „urmărire” a punctelor de pe frontieră.
- Metode globale, care utilizează informații globale, cum ar fi cunoașterea formei geometrice a frontierelor sau a ecuației matematice.(ex. Transformata Hough)

Majoritatea detectoarelor de frontiere utilizează derivată întâi și derivată a doua a funcției imagine, știind că într-un punct de front:

- Derivata întâi are un maxim sau un minim local
- Derivata a doua trece prin zero.

În cazul 2D, pentru o funcție  $f(x,y)$ , prima derivată corespunde Gradientului, iar a doua derivată corespunde Laplacianului. Detectia de fronturi în imagini se bazează pe operatori care aproximează în planul discret Gradientul sau Laplacianul imaginii. Detectoarele bazate pe Gradient produc matricea amplitudinii gradientului în fiecare punct al imaginii de intrare. De asemenea, ele pot produce și matricea direcției gradientului în fiecare punct al imaginii de intrare. Detectoarele bazate pe Laplacian sunt *izotropice*- rezultatul lor este același indiferent de direcțiile frontului din imagine. Un detector care folosește Laplacianul produce o matrice în care punctele de front sunt puncte de tranziție prin valoarea zero.

Vectorul gradient reprezintă direcția și mărimea variației maxime de intensitate într-un punct al unei imagini. Aceasta este perpendicular pe direcția frontierei. În figura ce urmează este ilustrat vectorul gradient perpendicular pe direcția frontierei.

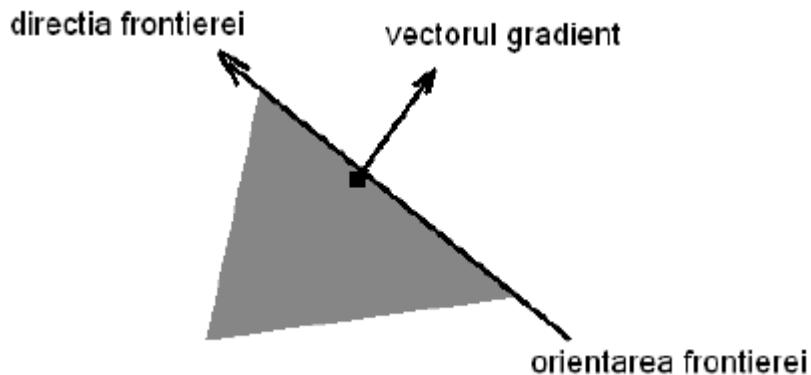


Figura 2.1: Direcția vectorului gradient[14]

Punctele de front dintr-o imagine sunt detectate calculând gradientul în fiecare pixel și identificând acei pixeli pentru care amplitudinea gradientului este mai mare decât un prag dat. Cei mai utilizați detectori de fronturi bazați pe gradient sunt: Roberts numit și operatorul cruce, Sobel și Prewitt. Ei sunt definiți pentru imagini în mai multe nivele de gri, dar utilizarea lor poate fi extinsă pentru imagini color. [14] În continuare vom trata în detaliu operatorul Sobel.

## 2.2. Operatorul Sobel

Ca și prim pas în aplicație este necesară detectarea muchiilor cu ajutorul unui operator ce realizează acest lucru, în cazul meu fiind operatorul Sobel. Aceasta, numit și filtrul Sobel este utilizat în prelucrarea de imagini și computer vision, în special în algoritmi de detecție a muchiilor, unde se creează o imagine ce subliniază muchiile obiectelor. Este denumit după Irwin Sobel and Gary Feldman, colegi la Stanford Artificial Intelligence Laboratory(SAIL). Acest operator este bazat pe conoluția imaginii cu un filtru pe direcția verticală și orizontală, prin urmare relativ ușor din punct de vedere al calculelor. Pe de altă parte, gradientul obținut este relativ brut pentru imaginile ce conțin variații mari de frecvențe.

Operatorul utilizează două matrice de dimensiune  $3 \times 3$ , iar prin operația de conoluție cu imaginea inițială se calculează gradientul aproximativ. Unul pentru schimbările pe orizontală și unul pentru schimbările pe verticală. Cele două măști de dimensiune  $3 \times 3$  sunt următoarele:

$$G_x = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \quad G_y = \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}$$

Măștile sunt proiectate pentru răspuns maxim la marginile cu direcțiile verticale și orizontale. Ele pot fi aplicate separat imaginii de intrare, fiind apoi folosite pentru calculul amplitudinii și al direcției marginii în fiecare pixel al imaginii de intrare. Amplitudinea este aproximată de formula următoare:  $|G| = \sqrt{G_x^2 + G_y^2}$ . Unghiul orientării marginii (relativ la pixelii gridului) este dat de:

- $\theta = \text{arctg}(G_y/G_x)$ , pentru  $G_x \neq 0$  (direcția).
- $+/-90^\circ$ , pentru  $G_x=0$ ,  $G_y \neq 0$
- $0^\circ$ , pentru  $G_y=0$ .

Unghiul zero înseamnă că direcția contrastului maxim de la negru la alb este de la stânga la dreapta în imagine, celealte unghiuri fiind măsurate în sens trigonometric (invers acelor de ceas) față de unghiul zero. Atunci când ieșirea detectorului de fronturi este numai matricea

amplitudinilor, componentele  $G_x$  și  $G_y$  pot fi calculate și adunate într-un singur pas, folosind operatorul de pseudo-convoluție:

$$G(x,y) = |(P_3+2*P_6+P_9)-(P_1+2*P_4+P_7)+(P_1+2*P_2+P_3)-(P_7+2*P_8+P_9)| \quad (1)$$

unde  $P_1, \dots, P_9$  sunt pixelii acoperiți de masca de convoluție, precum:

$$\begin{matrix} & P_1 & P_2 & P_3 \\ P = & P_4 & P_5 & P_6 \\ & P_7 & P_8 & P_9 \end{matrix}$$

Operatorul Sobel[14] solicită mai multe calcule decât operatorul Roberts<sup>6</sup>, dar masca sa de convoluție fiind mai mare, netezește mai mult imaginea și de aceea este mai puțin sensibil la zgomot. Operatorul Sobel produce valori de amplitudine mai mari decât cele produse de operatorul Roberts, pentru aceleași margini. Matricea amplitudinilor produsă de acest operator poate fi vizualizată ca o imagine în nivele de gri. Datorită efectului de netezire pe care îl are, marginile apar adesea în această imagine ca linii a căror lățime este de mai mulți pixeli. Pentru subțierea muchiei este necesară o postprocesare, cum ar fi aceea de “histerezis”, din algoritmul Canny<sup>7</sup>.

### 2.3. Transformata Hough

Transformata Hough este o metodă ce rezolvă o problemă clasică din viziunea artificială: găsirea dreptelor într-o imagine ce conține o mulțime de puncte de interes. Metoda clasică de a calcula drepte din fiecare pereche de puncte are o complexitate computațională ridicată,  $O(n^2)$  și nu este aplicabilă pentru un număr mare de puncte. Transformata Hough a fost propusă și patentată de Peter Hough și în varianta inițială a fost o metodă de timp real pentru a număra câte puncte sunt plasate pe fiecare posibilă dreaptă dintr-o imagine. Această metodă se baza pe reprezentarea dreptei în forma pantă-termen liber,  $(y=ax+b)$  și pe construirea unui spațiu de parametrii, numit și accumulator Hough. Pentru fiecare punct de interes din imagine, se calculează toate posibilele drepte ce trec prin el și se incrementează elementele din spațiul parametric. Dreptele relevante sunt localizate în maximele locale ale spațiului parametric.[15]

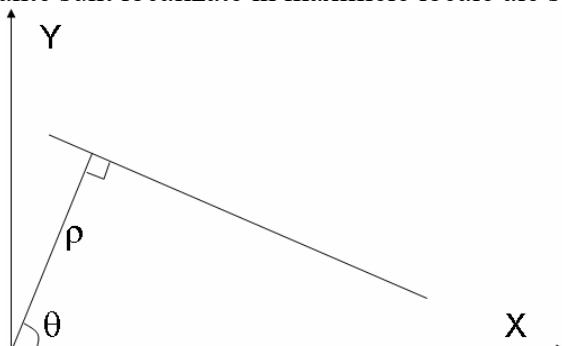


Figura 2.2: Vectorul perpendicular pe dreptă, ce trece prin origine, reprezintă dreapta prin parametrii  $\rho$  și  $\theta$ [15]

<sup>6</sup> Operatorul Roberts(operatorul cruce) este utilizat în procesarea de imagini pentru detectarea marginilor aplicându-se filtre verticale și orizontale în ordine.

<sup>7</sup> Algoritmul lui Canny este un algoritm de detectare a muchiilor ce își propune să îmbunătățească detectoarele de margini existente din anumite puncte de vedere precum: punctele de pe margină să fie bine localizate, detectorul să furnizeze un singur răspuns la un singur punct de pe margină.

Varianta inițială a fost orientată pe detecția dreptelor în imagini video, pe baza reprezentării dreptelor ca pantă și termen liber. Această reprezentare este sub-optimă, deoarece nu este mărginită: pentru a reprezenta toate posibilele drepte din imagine, panta și termenul liber trebuie să varieze în domeniul  $-\infty$  și  $+\infty$ . Rezultatele lui Duda și Hart[16] au făcut transformata Hough populară în domeniul viziunii artificiale. Principala problemă, parametrii nemărginiți, a fost rezolvată prin parametrizarea normală. Parametrizarea normală a unei drepte consistă în reprezentarea dreptei prin vectorul ce trece prin origine și este perpendicular pe dreaptă. Reprezentarea normală se mai numește și reprezentarea  $\rho\text{-}\theta$ . (Figura 2.2)

$$\rho = x \cdot \cos \theta + y \cdot \sin \theta \quad (2)$$

Pe lângă faptul că parametrii sunt limitați, cuantizarea parametrilor joacă un rol important în descreșterea complexității computaționale. Cuantizarea are legătură cu dimensiunea accumulatorului Hough. Pentru fiecare din cei doi parametri ai dreptei se stabilește un nivel de cuantizare, ce depinde de acuratețea cerută (ex. Acuratețea lui  $\rho$  poate fi 10, 1, 0.5 pixeli, etc, iar acuratețea lui  $\theta$  poate fi 10 grade, 1 grad, 0.5 grade etc). Parametrii  $\rho$  și  $\theta$  au un interval de variație limitat deoarece imaginea are o dimensiune finită. Valoarea maximă pentru  $\rho$  este diagonala imaginii,  $\rho_{\max}$ . În funcție de intervalul ales pentru  $\theta$ , există mai multe configurații echivalente pentru domeniul parametrilor (prima este cea propusă în articolul original):

1.  $\theta \in [-90^\circ, 90^\circ]$  sau  $\theta \in [0, 180^\circ]$ ,  $\rho \in [-\rho_{\max}, +\rho_{\max}]$
2.  $\theta \in [0, 360^\circ]$ ,  $\rho \in [0, \rho_{\max}]$

Deși transformata Hough se folosește cel mai des pentru detecția dreptelor, poate fi folosită și pentru detecția curbelor mai complexe, atâtă timp cât o parametrizare adecvată este disponibilă.

Duda și Hart[16] au propus detecția cercurilor, folosind un spațiu tridimensional și transformând fiecare punct într-un con circular în spațiul parametric (toate cercurile posibile ce conțin respectivul punct). Mai târziu Ballard a generalizat transformata Hough pentru a detecta orice formă non-analitică[17].

#### **2.4. Reprezentarea liniilor în Spațiul Hough**

Transformata Hough utilizează o reprezentare parametrică a liniilor pentru reducerea complexității calculului căutării lor în imagini :

$$y = a \cdot x + b \quad (3)$$

unde  $a$  = pantă dreptei iar  $b$  este termenul liber și se construiește un spațiu parametric numit și accumulator Hough.

Pentru fiecare punct de interes din imagine, se calculează toate posibilele drepte ce trec prin el, și se incrementează elementele din spațiul parametric. Dreptele relevante sunt localizate în maximele locale ale spațiului parametric. Pentru a reprezenta toate posibilele drepte din imagine, panta și termenul liber trebuie să varieze în domeniul  $-\infty$  și  $+\infty$ .

Pentru detecția liniilor drepte se poate utiliza această formă a ecuației dar o reprezentare mult mai bună este reprezentarea polară:  $\rho = x \cdot \cos \theta + y \cdot \sin \theta$ , ecuație ce este echivalentă cu [18]

$$y = \frac{\cos \theta}{\sin \theta} x + \frac{\rho}{\sin \theta} \quad (4)$$

unde:

- $\rho$  = distanța de la origine la cel mai apropiat punct de linia dreaptă,
- $\theta(\text{theta})$  = unghiul dintre axa Ox și linia ce se intersectează cu originea în cel mai apropiat punct.

Liniile paralele, ce sunt și în interesul nostru, au unghiul  $\theta$  similar.

Toate liniile pot fi reprezentate sub această formă când  $\theta$  aparține intervalului  $[0, 180^{\circ}]$  și  $\rho$  este un număr real. [18]

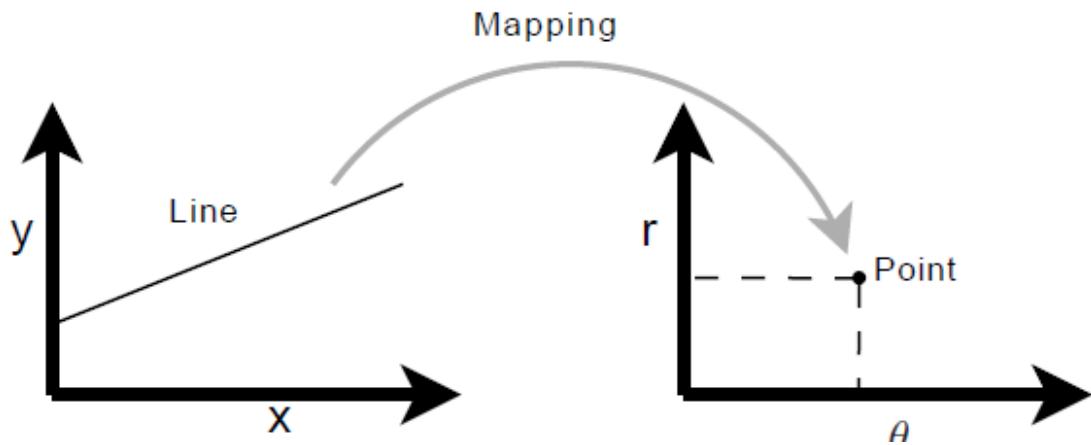


Figura 2.3: Maparea unei liniilor în Spațiul Hough[18]

Un concept important pentru Transformata Hough este maparea unui singur punct. Ideea este că un punct este mapat mai multor liniilor, care trec prin punctul respectiv. Principiul este ilustrat în imaginea următoare pentru punctul  $P_0(40,30)$ .

### 2.5. Maparea punctelor în Spațiul Hough

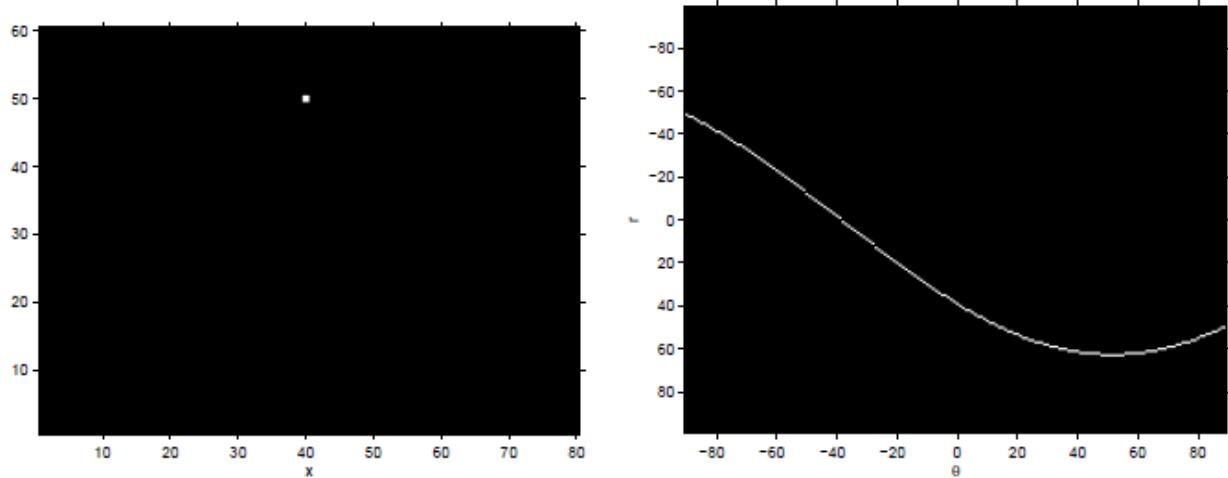


Figura 2.4: Transformarea unui punct ( $P_0$ ) într-o linie în Spațiul Hough. Liniile Spațiului Hough reprezintă toate liniile posibile ce trec prin punctul  $P_0$ .[18]

Deoarece pentru fiecare punct  $(x,y)$  există o infinitate de perechi  $(\rho,\theta)$  în spațiul parametric, trebuie găsită o modalitate de reducere a dimensionalității problemei pentru aplicarea practică. Pentru aceasta se discretizează problema prin discretizarea valorilor parametrilor.[18] Astfel, dacă imaginea inițială, în formă discretă, avea  $N \times M$  pixeli, putem discretiza spațiul parametrilor alegând, de exemplu  $d\theta=1^\circ$  și  $d\rho=1$  pixel. Discretizarea se face în funcție de problemă.

## 2.6. Coordonate polare

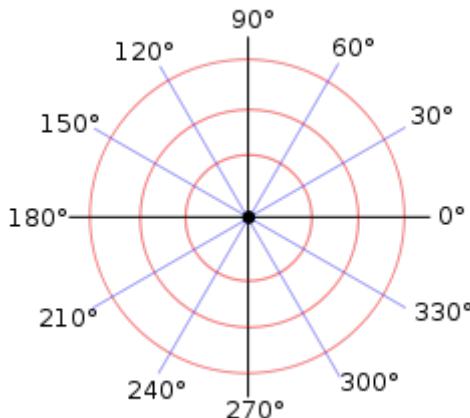


Figura 2.5: Un sistem polar cu unghiuri exprimate în grade[19]

În matematică, sistemul de coordonate polare este un sistem de coordonate bidimensional în care fiecărui punct din plan îi se asociază un unghi și o distanță. Sistemul coordonatelor polare este util mai ales în situații în care relația dintre două puncte este mai ușor de exprimat în termeni de distanțe și direcții (unghiuri); în sistemul cartezian sau ortogonal, o astfel de relație poate fi găsită doar cu ajutorul formulelor trigonometrice.[19]

### Trasarea punctelor în coordonate polare

Fiecare punct din sistemul de coordonate polare poate fi descris folosind două coordonate polare, numite ușual  $r$  (coordonata radială) și  $\theta$  (coordonata unghiulară, unghiul polar, sau azimutul, uneori reprezentat ca  $\phi$ ). Coordonata  $r$  reprezintă distanța radială de pol, și coordonata  $\theta$  reprezintă unghiul în sens trigonometric (invers acelor de ceasornic) de la direcția de  $0^\circ$  (numită uneori axă polară). De exemplu, coordonatele polare  $(3, 60^\circ)$  ar fi reprezentate în plan ca un punct aflat la 3 unități depărtare de pol pe direcția de  $60^\circ$ . Coordonatele  $(-3, 240^\circ)$  ar fi reprezentate prin același punct deoarece o distanță radială negativă este măsurată ca o distanță pozitivă pe aceeași direcție în sens opus (direcția reflectată față de origine, care diferă de direcția originală cu  $180^\circ$ ).

Aceasta ilustrează un aspect important al sistemului de coordonate polare, aspect care lipsește la cel cartezian: un singur punct poate fi exprimat printr-o infinitate de coordonate diferite. În general, punctul  $(r, \theta)$  poate fi reprezentat ca  $(r, \theta \pm n \times 360^\circ)$ . Coordonatele arbitrar  $(0, \theta)$  sunt utilizate prin convenție pentru reprezentarea polului, pentru că indiferent de coordonata  $\theta$ , un punct de rază 0 va fi mereu în pol. Pentru a obține o reprezentare unică a unui punct, este ușual să limităm  $r$  la numere nenegative  $r \geq 0$  și pe  $\theta$  la intervalul  $[0, 360^\circ]$  sau  $(-180^\circ, 180^\circ]$  (sau, în radiani,  $[0, 2\pi]$  sau  $(-\pi, \pi]$ ).

Unghiurile în notație polară sunt în general exprimate fie în grade, fie în radiani, utilizând

conversia  $2\pi$  rad =  $360^\circ$ . Alegerea depinde de context. Aplicațiile nautice folosesc gradele, în timp ce unele aplicații din fizică (mai ales mecanica rotației) și aproape toată literatura matematică legată de analiza matematică folosesc radiani.[19]

## 2.7. Coordonate carteziene

În matematică, sistemul de coordonate carteziene este folosit pentru a determina în mod unic un punct în plan prin două numere, numite de regulă abscisa și ordonata punctului. Pentru a defini coordonatele, se specifică două drepte perpendiculare și unitatea de lungime, care este marcată pe cele două axe. Coordonatele carteziene sunt folosite și în spațiu (unde se folosesc trei coordonate) și în mai multe dimensiuni. Pe lângă sistemul cartezian mai există și alte sisteme de specificare a poziției unui punct în plan, de ex. sistemul de coordonate polare.

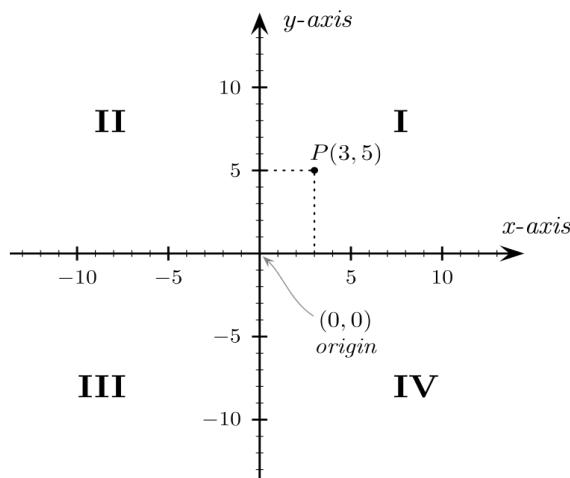


Figura 2.6: Cele 4 cadrane ale unui sistem de coordonate carteziene. [20]

Un sistem de coordonate cartezian în două dimensiuni este definit de obicei de două axe în unghi drept una cu celalaltă, formând un plan. Axa orizontală este în mod normal etichetată  $x$ , și axa verticală este notată cu  $y$ . Într-un sistem de coordonate tridimensional se adaugă o altă axă, de regulă notată cu  $z$ , furnizând a treia dimensiune de măsurare a spațiului. Axele sunt de regulă definite ca fiind perpendiculare una pe celalaltă. (Primele sisteme permitteau și axe oblice, adică axe care nu se intersectau în unghi drept, astfel de sisteme fiind folosite și astăzi, dar mai ales ca exercițiu teoretic.) Toate punctele dintr-un sistem de coordonate cartezian luate împreună formează un așa-numit plan cartezian. Ecuatiile care folosesc sistemul de coordonate cartezian sunt numite ecuații carteziene.[20]

## 2.8. Conversia din coordonate polare în coordonate carteziene și invers

Cele două coordonate polare  $r$  și  $\theta$  pot fi convertite în coordonate carteziene  $x$  și  $y$  prin utilizarea funcțiilor trigonometrice sinus și cosinus:

$$x = r \cdot \cos \theta$$

$$y = r \cdot \sin \theta$$

În timp ce două coordonate carteziene  $x$  și  $y$  pot fi transformate în coordonata polară  $r$  prin:

$$r = \sqrt{x^2 + y^2} \quad (5)$$

(Aplicând teorema lui Pitagora pe următoarea figură).

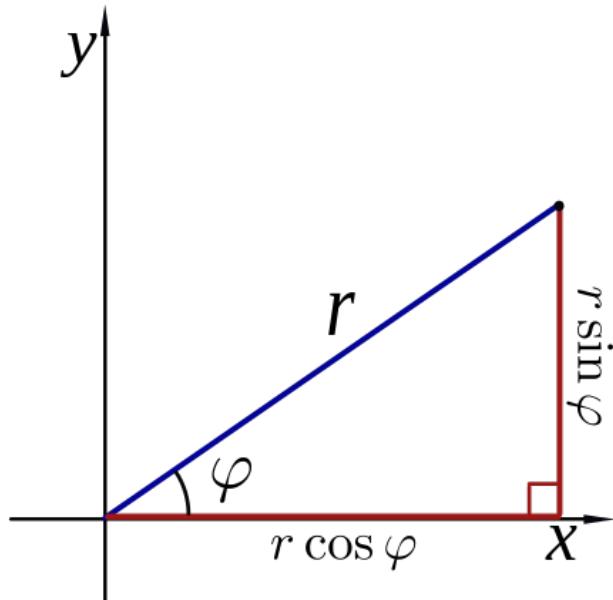


Figura 2.7: Diagramă care ilustrează formulele de conversie[19]

Pentru a obține coordonata polară  $\theta$ , trebuie să fie luate în considerare următoarele două idei:

- Pentru  $r=0$ ,  $\theta$  poate fi orice număr real.
- Pentru  $r \neq 0$ , pentru a obține o unică reprezentare a lui  $\theta$ , aceasta trebuie limitată la un interval de lungime  $2\pi$ . Alegeri convenționale pentru acest interval sunt  $[0, 2\pi)$  și  $(-\pi, \pi]$ .

Pentru a obține  $\theta$  în intervalul  $[0, 2\pi)$ , se poate folosi funcția  $\arctan$  ce reprezintă inversa funcției tangente.[19]

## 2.9. Ecuația unei drepte

Pentru a detecta ce pixeli din imagine fac parte din liniile trecerii de pietoni a trebuit să calculez ecuația dreptelor. Pentru a realiza acest lucru sunt mai multe modalități ce vor fi detaliate în cele ce urmează:

- Ecuația unei drepte când se cunosc 2 puncte  $A(x_a, y_a), B(x_b, y_b)$  prin care trece dreapta este dată de următoarea formulă:

$$d : \frac{x - x_A}{x_B - x_A} = \frac{y - y_A}{y_B - y_A} \quad (6)$$

- Ecuația unei drepte dacă cunoaștem un punct de pe dreaptă  $A(x_A, y_A)$  și panta  $m$  atunci: [21]
- 

$$d: y - y_A = m(x - x_A) \quad (7)$$

## 2.10. Panta unei drepte

În matematică, panta unei drepte este o valoare numerică ce descrie direcția și înclinația unei drepte. Din punct de vedere geometric, panta reprezintă unghiul făcut de dreaptă cu orizontală (într-un sistem de coordonate cartezian, axa  $Ox$ ).

Determinarea pantei unei drepte:

- Dacă se cunoasc 2 puncte de pe dreapta A(x<sub>a</sub>,y<sub>a</sub>) și B(x<sub>b</sub>,y<sub>b</sub>) atunci panta dreaptei va fi:

$$m_{AB} = \frac{y_B - y_A}{x_B - x_A} \quad (8)$$

- Dacă cunoaștem ecuația dreptei, de exemplu:  
d: ax+by+c=0 atunci panta va fi:

$$m_d = \frac{-b}{a} \quad (9)$$

- Cel mai adesea extragem panta din relații de paralelism și perpendicularitate între drepte:
- Dacă d<sub>1</sub> || d<sub>2</sub>, atunci m<sub>d1</sub> = m<sub>d2</sub>
- Dacă d<sub>1</sub> ⊥ d<sub>2</sub>, atunci m<sub>d1</sub> \* m<sub>d2</sub> = -1 [21]

### **2.11. Distanța de la un punct la o dreaptă**

Fie dreapta (d): y=mx+n, neparalelă cu axele de coordonate și P(x<sub>0</sub>,y<sub>0</sub>) un punct dat pentru care vrem să determinăm distanța d de la P la dreapta (d).

Distanța de la punctul P la dreapta (d) se exprimă prin:

$$d = \frac{|y_0 - mx_0 - n|}{\sqrt{1+m^2}} \quad (10)$$

Fie dreapta (d): ax+by+c=0 și P(x<sub>0</sub>,y<sub>0</sub>).

Distanța se obține înlocuind în ecuația ei, coordonatele curente prin acelea ale punctului dat, împărțind această cantitate prin rădăcina pătrată a sumei pătratelor coeficienților variabilelor și luând valoarea absolută a rezultatului astfel obținut.[22]

$$d = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}} \quad (11)$$

### **2.12. Distanța euclidiană**

În matematică, distanța euclidiană sau metrica euclidiană este distanța obișnuită între două puncte, dată în coordonate carteziene de formula lui Pitagora. Utilizând această formulă ca distanță într-un spațiu euclidian, acest spațiu (ca și orice alt spațiu cu produs scalar) devine spațiu metric. Norma asociată acestui spațiu metric se numește normă euclidiană.

În planul euclidian, dacă avem punctele P(p<sub>1</sub>,p<sub>2</sub>) și Q(q<sub>1</sub>,q<sub>2</sub>) atunci distanța este dată de următoarea formulă.[23]

$$d = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2} \quad (12)$$

## Capitolul 3. Implementarea aplicației

În această lucrare mă voi focaliza pe treceri de pietoni uniforme, generale, cu benzi alternante albe ce se află la o distanță relativ apropiată de camera de fotografiat. Trecerea de pietoni poate fi caracterizată ca fiind un şablon de dungi albe și negre ce alternează.

Pentru a extrage aceste caracteristici, încep prin a detecta muchiile din imaginea RGB și apoi este aplicată Transformata Hough pentru a extrage liniile. Așadar, un grup de linii paralele vor reprezenta structura unei treceri de pietoni, iar pentru a decide dacă în imagine se află o trecere de pietoni, adăugăm constrângeri legate de numărul de linii găsite, lungimea acestora, culoare.

### 3.1. Filtre liniare. Filtrul gaussian

Filtrarea imaginilor este necesară pentru a micșora efectele nedorite cauzate de un sistem de achiziție, de rezoluția prea scăzută sau de un sistem de transmisie de imagini de bandă mică.

Vom considera o filtrare liniară cu un filtru(mască) de dimensiune  $3 \times 3$  cu ponderile  $w(s,t)$  și imaginea dată prin  $f(x,y)$ . Imaginea filtrată o vom nota cu  $g(x,y)$ .

Coefficienții filtrului:

$$\begin{matrix} f(x-1, y-1) & f(x, y-1) & f(x+1, y-1) \\ f(x-1, y) & f(x, y) & f(x+1, y) \\ f(x-1, y+1) & f(x, y+1) & f(x+1, y+1) \end{matrix}$$

Pixelii din imagine acoperiți de mască:

$$g(x,y) = w(-1,-1)f(x-1,y-1) + w(0,-1)f(x,y-1) + w(1,-1)f(x+1,y-1) + w(-1,0)f(x-1,y) + w(0,0)f(x,y) + w(0,1)f(x,y+1) + w(-1,1)f(x-1,y+1) + w(0,1)f(x,y+1) + w(1,1)f(x+1,y+1) \quad (13)$$

Filtrul implementat de mine este filtrul gaussian, fiind cel mai potrivit pentru eliminarea zgomotului gaussian. Ponderea se reduce odată cu creșterea distanței față de pixelul central. Filtrarea este cu atât mai puternică cu cât valoarea deviației standard  $\sigma$  și dimensiunea măștii este mai mică. Pentru dimensiunea  $3 \times 3$ , aplicarea acestui filtru se face cu masca următoare:[24]

$$\begin{matrix} 1 & 2 & 1 \\ 1/16 & 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

### 3.2. Filtre neliniare. Filtrul median

Filtrul median este un filtru de ordine a cărui ieșire este statistica de ordin central, adică elementul ce se află pe poziția de mijloc a șirului ordonat de valori selectate de fereastra de filtrare.

Filtrul median este potrivit pentru a elimina zgomotul de tip „sare și piper”. După ordonarea valorilor pixelilor, valorile zgomotului(adică 0 sau 255) se vor situa pe primele, respectiv ultimele poziții în multime și deci, la ieșirea filtrului, vom avea o valoare diferită de valorile zgomotului. Totuși există și situații în care, după filtrare, mai există pixeli afectați de zgomot. În acest caz, spunem că filtrul a fost „străpuns” de zgomot. Acest lucru este posibil atunci când mai mult de jumătate din pixelii selectați de fereastra de filtrare, sunt afectați în același mod(sare sau piper,255 sau 0) de zgomot.

De exemplu, considerând o fereastră 3x3, valorile pixelilor din fereastra respectivă sunt:

$$\begin{bmatrix} 10 & 20 & 20 \\ 15 & 20 & 20 \\ 20 & 25 & 100 \end{bmatrix}$$

Prin sortarea acestor valori se obține sirul(10,15,20,20,**20**,20,20,25,100), pentru care valoarea mediană este 20. Deci aplicarea filtrului va furniza în această fereastră pentru pixelul central valoarea 20. În acest fel se forțează ca fiecare pixel din imagine să fie cât mai asemănător cu vecinii săi, dar în același timp contururile sunt conservate. Filtrul median este neliniar, fiind de fapt o operație morfologică. Astfel, la operația de eroziune, fiecare pixel este înlocuit cu valoarea cea mai mică din fereastră, la operația de dilatare, fiecare pixel este înlocuit cu valoarea cea mai mare din fereastră, în timp ce la filtrarea mediană fiecare pixel este înlocuit cu valoarea mediană din fereastră.[25] În imaginile următoare, se poate observa efectul filtrului median.



Figura 3.2: Imagine originală afectată de zgomot "sare și piper"



Figura 3.1: Imagine filtrată cu ajutorul filtrului median

### 3.3. Implementarea Operatorului Sobel:

Pentru implementarea filtrului Sobel am utilizat cele două măști de dimensiune 3x3:

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

În programul meu, măștile sunt scrise în forma următoare:

```
Mat maskGy = ((Mat<char>(3, 3) << -1, 0, 1, -2, 0, 2, -1, 0, 1));
Mat maskGx = ((Mat<char>(3, 3) << -1, -2, -1, 0, 0, 0, 1, 2, 1));
```

Una pentru schimbările pe orizontală și una pentru schimbările pe verticală. Pentru a verifica rezultatul obținut după implementarea proprie, am comparat cu ce există deja în OpenCV iar acest lucru se poate observa în următoarele imagini.



Figura 3.3: Filtrul Sobel implementat

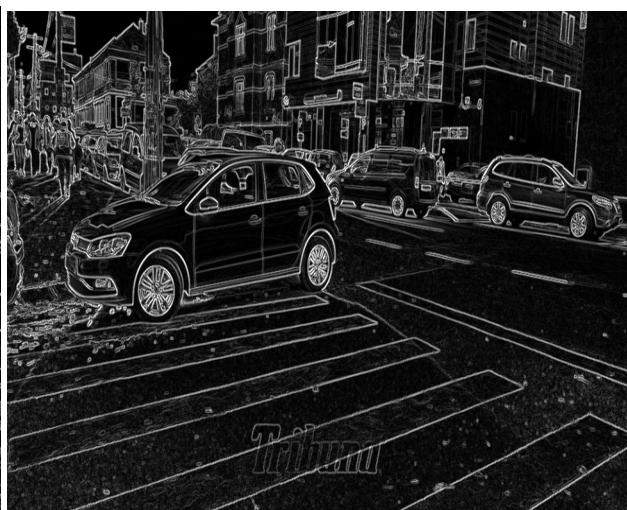


Figura 3.4: Filtrul Sobel din OpenCV

Atât pentru Gx cât și pentru Gy am parcurs imaginea cu ajutorul a două for-uri calculând astfel două sume. Pentru gradientul imaginii utilizez următoarea formulă:  $|G| = \sqrt{G_x^2 + G_y^2}$

Exemplificarea calculului componentei Gx:

```
Gx = maskGx.at< char>(0, 0)*(img.at< char>(row - 1, col - 1)) +
      maskGx.at< char>(1, 0)*(img.at< char>(row, col - 1)) +
      maskGx.at< char>(2, 0)*(img.at< char>(row + 1, col - 1)) +
      maskGx.at< char>(0, 1)*(img.at< char>(row - 1, col)) +
      maskGx.at< char>(1, 1)*(img.at< char>(row, col)) +
      maskGx.at< char>(2, 1)*(img.at< char>(row + 1, col)) +
      maskGx.at< char>(0, 2)*(img.at< char>(row - 1, col + 1)) +
      maskGx.at< char>(1, 2)*(img.at< char>(row, col + 1)) +
      maskGx.at< char>(2, 2)*(img.at< char>(row + 1, col + 1));
```

După aplicarea acestui filtru a trebuit aplicat și un threshold<sup>8</sup>, care are rolul să subțieze liniile pentru a avea rezultate cât mai satisfăcătoare. Operația de threshold preia o imagine și un thresh(prag) ca și intrare și produc o imagine de ieșire, comparând fiecare pixel din imaginea de intrare cu acel prag. Dacă valoarea este mai mare decât pragul impus atunci imaginii de ieșire îi este atribuită o valoare, în caz contrar altă valoare. În OpenCV există mai multe tipuri de threshold, dar cel pe care l-am utilizat eu, este cel binar(Binary Thresholding).

```
threshold(src, dst, thresh, maxValue, THRESH_BINARY);
```

Parametrii, având următoarele semnificații:

- src- imaginea de intrare
- dst- imaginea de ieșire
- thresh- valoarea pragului
- maxValue- valoarea maximă de utilizat cu acest tip de threshold.
- THRESH\_BINARY- tipul de threshold utilizat

Prin verificări repetate am observat că un prag de 120 este suficient pentru a evidenția pixelii de care am nevoie, adică dacă valoarea pixelilor este mai mare decât 120 atunci le voi schimba valoarea în 255.

<sup>8</sup> Threshold= prag

Funcția ce implementează Sobel în OpenCV este următoarea:

`void Sobel(InputArray src, OutputArray dst, int ddepth, int dx, int dy, int ksize=3, double scale=1, double delta=0, int borderType=BORDER_DEFAULT)` iar parametrii ei au următoarea semnificație:

- src: imaginea de intrare
- dst: imaginea de ieșire
- ddepth: adâncimea imaginii
- ksize: mărimea nucleului Sobel. Trebuie să fie 1,3, 5 sau 7.
- scale: mărime optională pentru scalare
- delta: mărime optională ce este adăugată în rezultat
- borderType: metoda extrapolării pixelilor

Ca și exemplu din program voi exemplifica prin calculul componentei Gx în openCV.

```
Sobel(src_gray, grad_x, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT);
convertScaleAbs(grad_x, abs_grad_x)
```

A fost necesar și calculul valorii absolute și conversia rezultatului la reprezentare pe 8 biți prin apelarea funcției `convertScaleAbs`.



Figura 3.5: Imaginea inițială



Figura 3.6: Imaginea după aplicarea filtrului Sobel

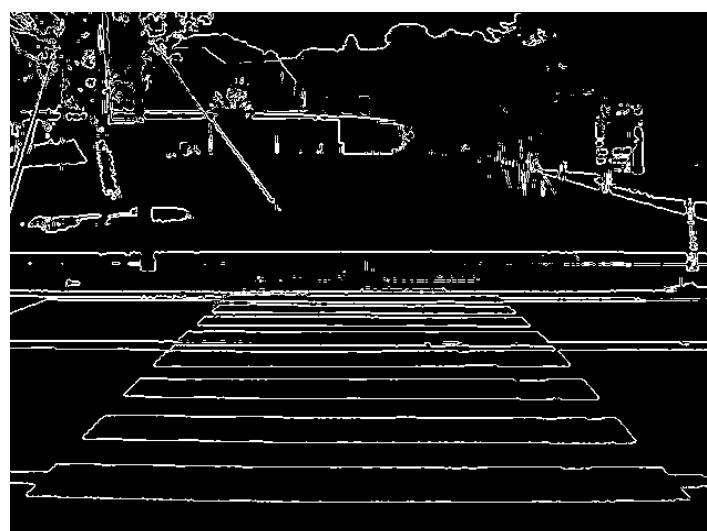


Figura 3.7: Imaginea după aplicarea pragului

Rezultatul obținut a fost unul satisfăcător ce mi-a permis să merg spre pasul următor și anume, extragerea liniilor din imagini.

### 3.4. Transformata Hough

În algoritmul urmărit, următoarea etapă este identificarea liniilor paralele iar acest lucru îl voi realiza cu ajutorul Transformatei Hough.

În OpenCV există implementate două tipuri de transformate Hough și anume:

- Transformata Hough Standard
- Transformata Hough Probabilistică

#### 3.4.1. Transformata Hough Standard

Are ca și rezultat un vector de perechi  $(r, \theta)$  și este implementată prin funcția *HoughLines*.  
*HoughLines(InputArray image, OutputArray lines, double rho, double theta, int threshold, double srn=0, double stn=0 )*

Aceasta are următoarele argumente:

- dst: reprezintă ieșirea detectorului de muchii. Ar trebui să fie o imagine alb-negru.
- lines: un vector ce memorează parametrii  $(r, \theta)$  a liniilor detectate.
- rho: rezoluția parametrului  $r$  exprimată în pixeli
- theta: rezoluția parametrului  $\theta$  în radiani.
- threshold: numărul minim de intersecții pentru a detecta o linie.
- srn și stn: sunt parametri setati în mod implicit pe zero.

Apelarea functiei HoughLines:

```
HoughLines(cdst, lines, 1, CV_PI / 90, 200, 0, 0);
```

În urma a diferite încercări, am considerat că un threshold de 200 este suficient.

După cum știm o linie poate fi reprezentată în două forme. De exemplu:

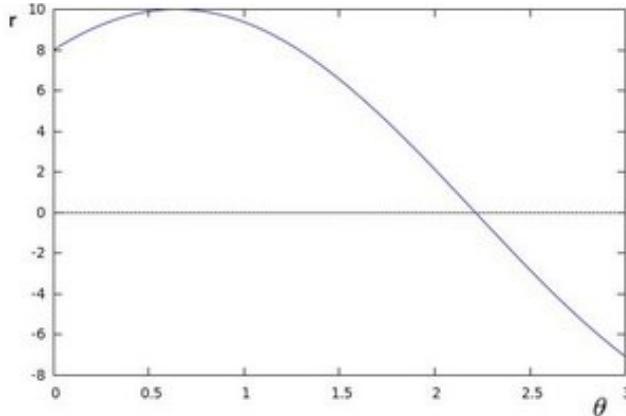
- a) în sistemul de coordonate carteziene prin parametri  $P(m, b)$ .
- b) în sistemul de coordonate polare prin parametrii  $P(r, \theta)$ .

În Transformata Hough vom exprima liniile în coordonate polare sub forma următoarei ecuații:

$$r_0 = x_0 \cdot \cos \theta + y_0 \cdot \sin \theta \quad (14)$$

În general pentru fiecare punct  $(x_0, y_0)$  putem defini familia de linii ce trece prin punctul respectiv prin ecuația de mai sus, însemnând că fiecare pereche  $(r_0, \theta)$  reprezintă fiecare linie ce trece prin  $(x_0, y_0)$ .

Dacă pentru un punct  $(x_0, y_0)$  reprezentăm grafic familia de linii ce trec prin acel punct, obținem un sinus. De exemplu, pentru  $x_0=8$  și  $y_0=6$  obținem următoarea reprezentare în planul  $(\theta, r)$ :

Figura 3.8: Reprezentare în planul  $\theta$ - $r$ [26]

Considerăm doar punctele care au  $r>0$  și  $0<\theta<2\pi$ .

În general, o linie poate fi detectată găsind numărul de intersecții între curbe. Cu cât avem mai multe curbe intersectate, cu atât intersecția este reprezentată de mai multe puncte. De obicei, se definește un prag(eng. Threshold) a numărului minim de intersecții, necesare pentru a defini o linie. În acest mod funcționează și Transformata Hough [26]

### 3.4.2. Transformata Hough Standard Probabilistica

Este o implementare mai eficientă a transformatei Hough. Ca și ieșire avem extretele liniilor detectate( $x_0, y_0, x_1, y_1$ ) și este implementată prin funcția *HoughLinesP*.

*HoughLinesP*(*InputArray dst*, *OutputArray lines*, *double rho*, *double theta*, *int threshold*, *double minLineLength*=0, *double maxLineGap*=0 )

Aceasta are următoarele argumente:

- dst: reprezintă ieșirea detectorului de muchii. Ar trebui să fie o imagine alb-negru.
- lines: vectorul ce va memora parametrii ( $x_{start}, y_{start}, x_{end}, y_{end}$ ) a liniilor detectate.
- rho: rezoluția parametrului  $r$  exprimată în pixeli. Se folosește de obicei 1 pixel.
- theta: rezoluția parametrului  $\theta$  în radiani. Se folosește 1 grad (CV\_PI/180).
- threshold: numărul minim de intersecții pentru a „detectata” o linie.
- minLineLength: numărul minim de puncte ce pot forma o linie. Liniile cu mai puține puncte decât acest număr nu sunt luate în considerare.
- maxLineGap: numărul maxim de goluri dintre două puncte de pe aceeași linie.

În programul meu am utilizat un threshold de 200, lungimea minima a unei linii 50, iar maximul de spații vide l-am setat la 10.

```
HoughLinesP(dst, lines, 1, CV_PI / 180, 200, 50, 10);
```

Pentru a desena segmentele date de această funcție și a calcula lungimea acestora, am parcurs vectorul lines ce reține extremitățile liniilor, calculând distanța dintre două puncte cu formula prezentată la capitolul anterior.

```
for (size_t i = 0; i < lines.size(); i++)
{
    Vec4i l = lines[i];
    line(inputImage2, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 255, 0), 1, CV_AA);
    distancePQ = sqrt(((l[2] - l[0])*(l[2] - l[0])) + ((l[3] - l[1])*(l[3] - l[1])));
    cout << "distanța HoughLinesP " << distancePQ << endl;}
```

În programul meu, am utilizat ambele funcții, dar cu anumite considerente. Mai întai am aplicat HoughLines, interesându-mă în principal vectorul lines, cu rho și theta. Din noțiunile prezentate în capitolul anterior, am utilizat transformările din coordonate polare în coordonate carteziene și invers știind că:

$$\begin{aligned}x &= \rho \cos(\theta) \\y &= \rho \sin(\theta)\end{aligned}\tag{15}$$

Deducând astfel un punct de pe linie având coordonatele x,y. Dar pentru a trasa o dreaptă sunt necesare două puncte, motiv pentru care creez încă două puncte  $Pt_1$  și  $Pt_2$  la o distanță de 1000 de primul punct descoperit. Un aspect foarte important al liniilor paralele îl reprezintă unghiul acestora. Pentru toate este același și îl știm ca fiind  $\theta(\theta)$ .

Următorul pas este identificarea pixelilor ce se află pe linia descoperită a trecerii de pietoni. Iar pentru acest lucru calculez distanța de la toate punctele din imagine la dreapta mea, folosind formula distanței de la un punct la o dreaptă. În cazul în care distanța este 0, verific dacă culoarea pixelului este albă(are valoarea 255). În caz afirmativ voi evidenția acești pixeli cu o culoare(roșie) pe imaginea color și voi contoriza numărul acestora.

Unul dintre cele mai importante lucruri de aflat este lungimea segmentelor detectate și pentru acest lucru avem nevoie de punctul de start și punctul de stop al segmentului și astfel putem aplica formula distanței dintre două puncte. Ca și verificare se poate compara lungimea găsită în acest pas cu lungimea segmentelor dată de Transformata Hough Probabilistică.

Pentru simplitate, pentru fiecare linie am contorizat numărul de pixeli albi, iar dacă acest număr este mai mare decât un număr impus, atunci există trecere în imagine.

În experimentul nostru, vom seta lungimea unei linii la 60 de pixeli și numărul minim al liniilor paralele 5, bazându-ne și pe performanțele bune ale camerei de fotografiat.

Un lucru important atunci când o astfel de aplicație se pune în practică este timpul necesar rulării. Timpul necesar fiecărui pas din program și anume timpul pentru încărcarea imaginii, timpul pentru procesare, pentru afișarea rezultatului. În experimentul meu am calculat timpul pentru mai multe tipuri de imagini, astfel:

```
clock_t beginRead = clock();
//codul necesar...
clock_t endRead = clock();
double time_forRead = (double)(endRead - beginRead) / CLOCKS_PER_SEC;
```

Timpul necesar citirii este cuprins între 75 și 35 milisecunde, iar timpul pentru procesare este cuprins între 1.755 și 1.334 secunde la prima rulare. Apoi scade ajungând la 19 milisecunde, respectiv 1.262 secunde.

## Capitolul 4. Testarea aplicației

### 4.1. Elemente de configurare

Proiectul este realizat ca fiind Qt Project în Visual Studio, și utilizez limbajul C++. Ca și prim pas, este necesară descărcarea bibliotecii și realizarea setărilor pentru a recunoaște bibliotecile necesare. Foarte important este OpenCV ce trebuie și acesta descărcat de pe site-ul următor: <http://opencv.org/>. Pentru a putea utiliza funcții qt sau opencv trebuie să includem bibliotecile următoare:

```
#include <QtWidgets/QApplication>
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
```

Setările necesare pot fi găsite la următoarea adresă web: [https://wiki.qt.io/How\\_to\\_setup\\_Qt\\_and\\_OpenCV\\_on\\_Windows](https://wiki.qt.io/How_to_setup_Qt_and_OpenCV_on_Windows). Precizez că sistemul de operare pe care am lucrat eu este Windows.

### 4.2. Rezultatele experimentale:

În urma implementării, am realizat diferite teste pe mai multe imagini ce le voi prezenta în cele ce urmează. Sunt imagini în care se compară rezultatul obținut de mine prin Transformata Hough și rezultatele obținute prin Transformata Hough Probabilistică.

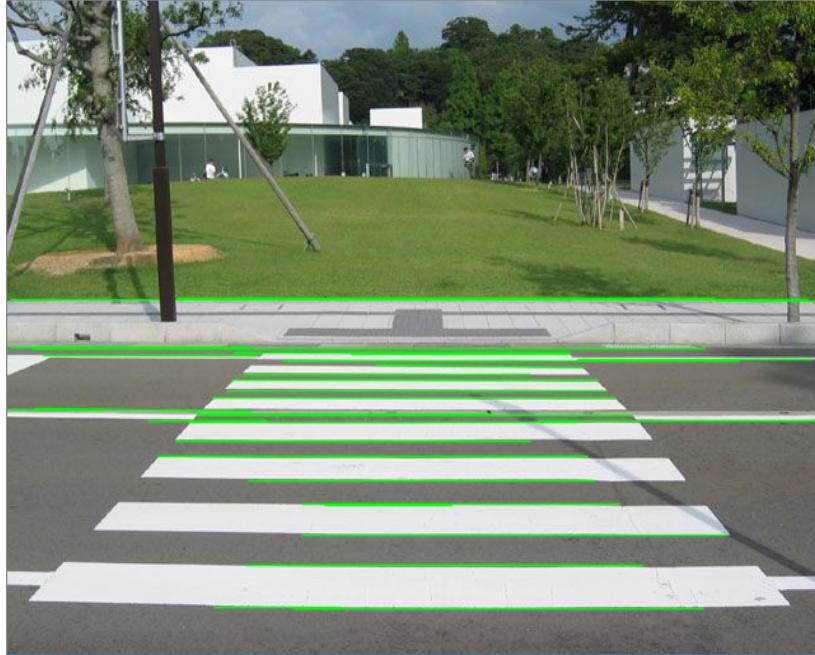


Figura 4.1: Detectarea liniilor utilizând Transformata Hough Probabilistică

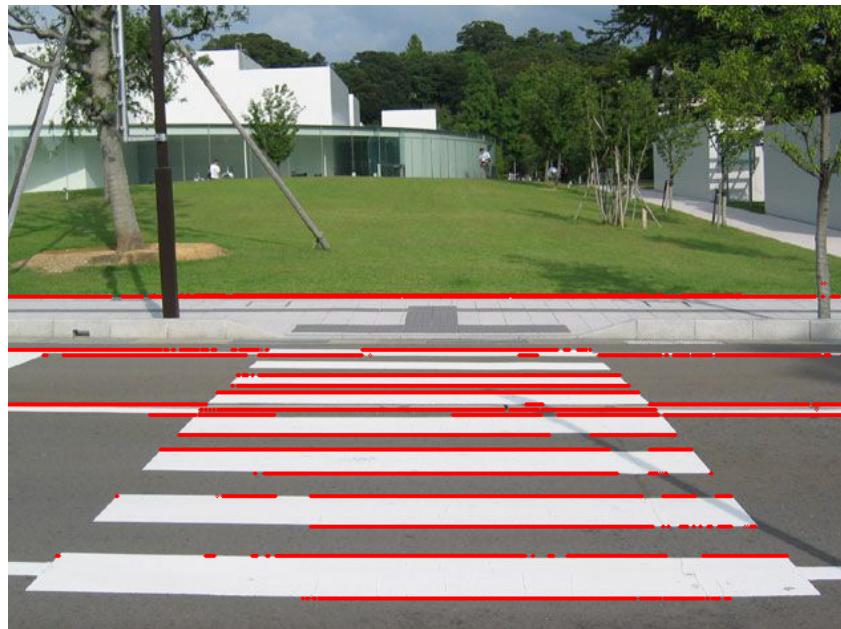


Figura 4.2: Detectarea liniilor paralele utilizând Transformata Hough



Figura 4.3: Detectarea liniilor utilizând Transformata Hough Probabilistică

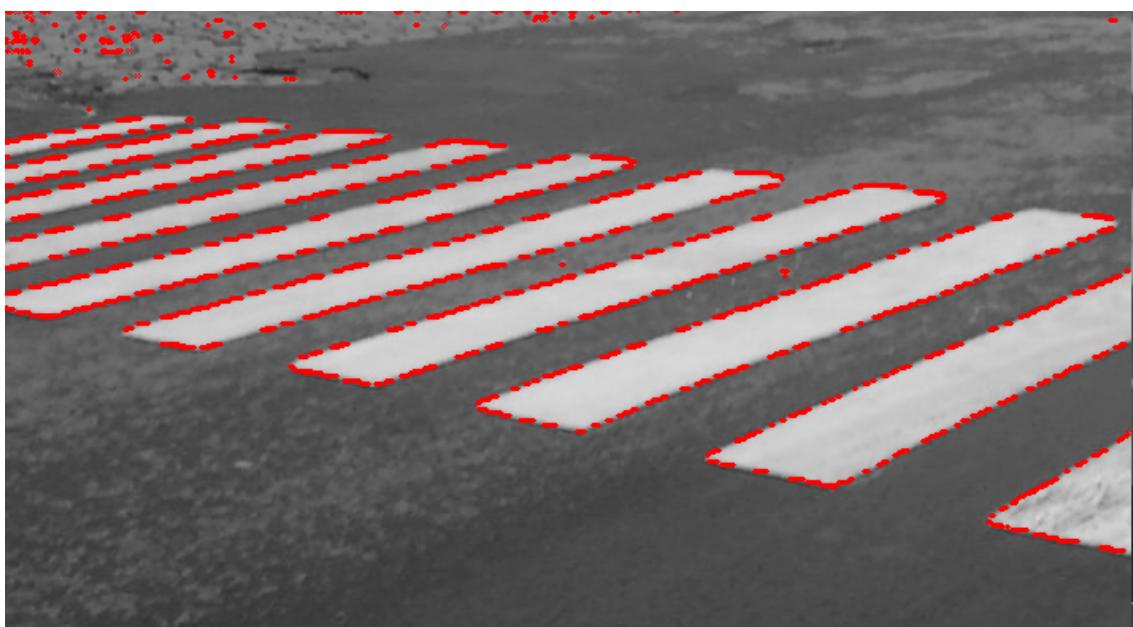


Figura 4.4: Detectarea liniilor paralele utilizând Transformata Hough

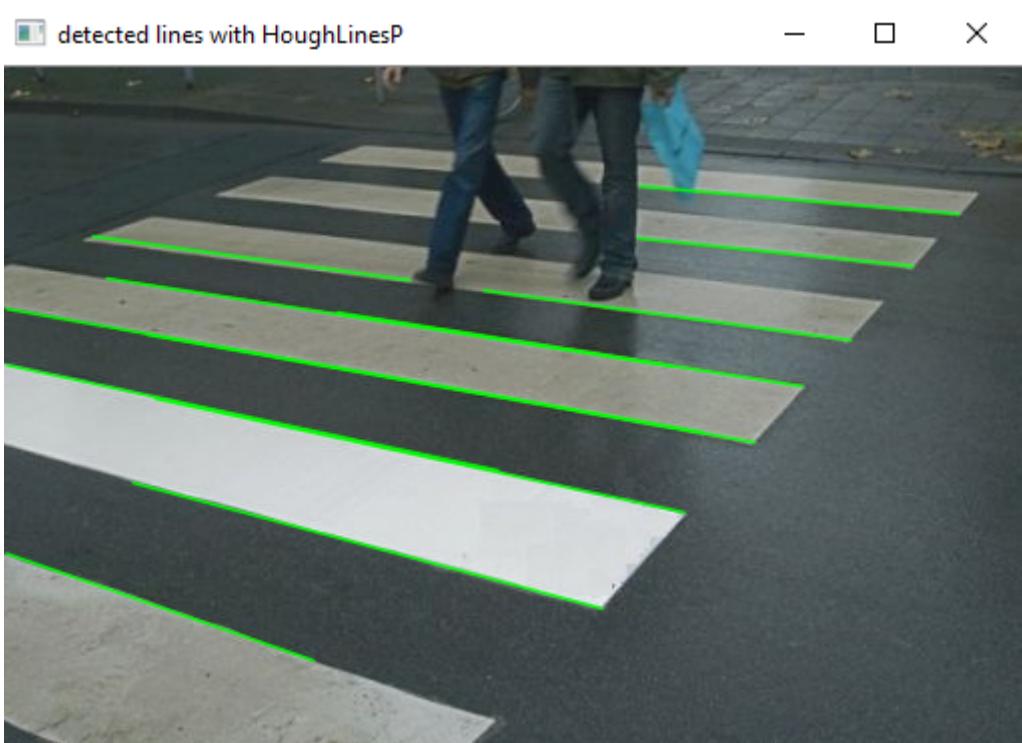


Figura 4.5: Detectarea liniilor utilizând Transformata Hough Probabilistică



Figura 4.6: Detectarea liniilor paralele utilizând Transformata Hough

## Capitolul 5. Concluzii

Prin urmare o aplicație ce detectează trecerea de pietoni este una folositoare în special pentru a dezvolta echipamente ce pot ajuta persoanele ce necesită ajutor pentru deplasare.

În această lucrare am realizat detectarea trecerii de pietoni în imagini. Modalitatea de implementare a algoritmului cuprinde atât funcții din OpenCV cât și filtre implementate propriu. Inițial, în caz de zgomote în imagine, se poate aplica un filtru median implementat pentru a elimina zgomotele din imagine. Eu am testat atât filtrul median cât și filtrul gaussian, în cazul meu, mai bun a fost filtrul median deoarece păstrează muchiile. Următorul pas este detectarea muchiilor, iar pentru acest lucru am implementat un filtru Sobel, iar pentru comparație și verificare am apelat și funcțiile din OpenCV aferente celor două filtre. Pentru rezultate satisfăcătoare în pașii ce urmează, a fost necesar un threshold după aplicarea operatorului Sobel pentru a avea muchiile mai subțiri.

Pentru a descoperi liniile în imaginea de intrare am utilizat Transformata Hough în ambele forme, atât cea standard cât și cea probabilistică. Cea din urmă m-a ajutat pentru a verifica exact lungimea segmentelor detectate, utilizând distanța dintre două puncte (distanță euclidiană) și pentru a compara segmentele desenate pe imagine cu cele realizate de mine prin Transformata Hough.

Vectorul de ieșire al Transformantei Hough conține parametrii  $\rho(\text{rho})$  și  $\theta(\text{theta})$ . Pentru a ajunge la coodonate carteziene am utilizat ecuațiile:

$$x = \rho * \cos(\theta) \quad (16)$$

$$y = \rho * \sin(\theta) \quad (17)$$

Acest fapt m-a ajutat să descopăr coordonatele unui punct de pe linia detectată, apoi să îmi creez eu câte puncte îmi sunt necesare pentru trasarea liniilor. Un aspect deloc de neglijat este faptul că unghiul liniilor paralele este același, în cazul nostru theta, fapt care mi-a dat voie să verific ce liniile din imagine sunt paralele. Pentru a ști exact pixelul ce aparține liniei de interes am calculat distanța de la toate punctele din imagine la drepte, utilizând formule matematice pentru distanța de la un punct la o dreaptă. Pixelii care se aflau la distanță zero, erau pe dreptele trecerii de pietoni. Culoarea pixelilor am verificat să aibă valoarea 255(alb). Pentru a verifica lungimea segmentelor detectate am contorizat numărul de pixeli de culoarea albă de pe fiecare linie, iar liniile au fost numărate și ele. Pentru a considera că există o trecere am setat ca lungimea să fie minimă de 60 pixeli, iar numărul minim de liniile paralele din grup să fie 5.

În cele din urmă aş putea afirma faptul că o primă îmbunătățire a acestui program este detecția trecerii în secvențe video, o astfel de aplicație fiind foarte utilă în opinia mea. Dezvoltată ca și o aplicație pe telefonul mobil, aplicație Android sau iOS ar putea surprinde imagini în timp real din jur, dându-i utilizatorului un răspuns audio în cazul existenței unei treceri de pietoni în imagine.

## Bibliografie

- [1] Ted Morris, Xinyan Li, Vassilios Morellas, Nikos Papanikolopoulos, „Video Detection and Classification of Pedestrian Events at Roundabouts and Crosswalks”, pp. , 2013.
- [2] Prof. dr. ing. fiz. Vasile Manta, „Procesarea imaginilor”, Universitatea Tehnică „Gheorghe Asachi”, Iași.
- [3] Deepak Sharma, „Review Paper on Zebra Crossing and Traffic Signals using Mobile Phone Camera for Blind Persons”, , , pp. , .
- [0] Jason Banich, „Zebra Crosswalk detection assisted by neural networks”, , , pp. , 2016.
- [4] James M. Coughlan și Huiying Shen , „A Fast Algorithm for Finding Crosswalks using Figure-Ground Segmentation”, , , pp. , .
- [5] V. Ivanchenko, J. Coughlan, H. Shen, „Detecting and Locating Crosswalks using a Camera Phone”, 2008.
- [6] R. Advanyi, B. Varga, K. Karacs, „Advanced crosswalk detection for the Bionic Eyeglass”,pp, 2010.
- [7] S. Se, „Zebra-crossing Detection for the Partially Sighted”, 2000.
- [8] M. Omachi, „S. Traffic Light Detection with Color and Edge Information”, 2009.
- [9] Dragan Ahmetovic, Roberto Manduchi, James Coughlan, „Automatic Population of Spatial Databases for Increased Safety of Blind Travelers”.
- [12] Introducere în utilizarea bibliotecii OpenCV [Online], Disponibil la adresa: <http://users.utcluj.ro/~robert/ip/PI-L1r.pdf>, Accesat: .
- [10] Introducere în OpenCV [Online], Disponibil la adresa: [http://imag.pub.ro/common/staff/cfloreau/papers/SSPI\\_OpenCV.pdf](http://imag.pub.ro/common/staff/cfloreau/papers/SSPI_OpenCV.pdf), Accesat: .
- [11] Basic Structures-OpenCV [Online], Disponibil la adresa: [http://docs.opencv.org/2.4.13/modules/core/doc/basic\\_structures.html](http://docs.opencv.org/2.4.13/modules/core/doc/basic_structures.html), Accesat: .
- [13] Prof.dr.ing.fiz.Vasile Manta, „Introducere în prelucrarea imaginilor. Imagini grayscale. Modelul de culoare RGB. Formatul imaginilor bitmap”, Universitatea Tehnică „Gheorghe Asachi”, Iași .
- [14] Prof.univ.dr.ing. Florica Moldoveanu, Detectia frontierelor din imagini [Online], Disponibil la adresa: <http://andrei.clubcisco.ro/cursuri/4spg/9.detectia.frontierelor.in.imagini.PDF>, Accesat: .
- [15] Detecția dreptelor prin transformata Hough [Online], Disponibil la adresa: [http://users.utcluj.ro/~rdanescu/srf/lab\\_11r.pdf](http://users.utcluj.ro/~rdanescu/srf/lab_11r.pdf), Accesat: .
- [16] R.O. Duda si P.E.Hart, „Use of the Hough Transformation to Detect Lines and Curves in Pictures”, Comm.ACM, 1972.
- [17] D.H.Ballard, „Generalizing the Hough Transform to Detect Arbitrary Shapes”, Pattern Recognition, 1981.
- [18] „Line Detection by Hough transformation”,[Online], Disponibil la adresa: [http://web.ipac.caltech.edu/staff/fmasci/home/astro\\_refs/HoughTrans\\_lines\\_09.pdf](http://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/HoughTrans_lines_09.pdf) .
- [19] Coordonate Polare [Online], Disponibil la adresa: <https://www.scribd.com/doc/9546570/COORDONATE-POLARE-teorie>, Accesat: .
- [20] Reprezentarea punctelor în plan cu ajutorul sistemelor de axe de coordonate [Online], Disponibil la adresa: <http://matepedia.ro/elemente-de-organizare-datelor/>, Accesat: .
- [21] Ecuatia unei drepte [Online], Disponibil la adresa: <http://deacoffee.com/ecuatia-unei-drepte/>, Accesat.
- [22] Distanța de la un punct la o dreaptă [Online], Disponibil la adresa: [http://www.meditationonline.ro/44100-15-39-0-0-Formule\\_Matematica\\_Determinanti\\_Distanta\\_de\\_la\\_un\\_punct\\_la\\_o\\_dreapta.html](http://www.meditationonline.ro/44100-15-39-0-0-Formule_Matematica_Determinanti_Distanta_de_la_un_punct_la_o_dreapta.html), Accesat: .

- [23] Distanta dintre doua puncte in plan, cand se cunosc coordonatele punctelor [Online], Disponibil la adresa: <https://sites.google.com/site/videomeditatii/clase-liceale-9-12/clasa-a-10-a/programa-scolara-pentru-matematica-clasa-a-x-a/distanta-dintre-doua-puncte-in-plan-cand-se-cunosc-coordonatele-punctelor>, Accesat: .
- [24] Prof. dr. ing. fiz. Vasile Manta, „Prelucrarea imaginilor in domeniul spațial”, Universitatea Tehnică „Gheorghe Asachi”, Iași.
- [25] Filtrarea imaginilor [Online], Disponibil la adresa: <http://www.miv.ro/ro/documentatie/pi/PIIlab07.pdf>, Accesat: .
- [26] Hough Line Transform [Online], Disponibil la adresa: [http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough\\_lines/hough\\_lines.html](http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html), Accesat: .

## Anexă

### Main.cpp

```
#include "hello.h"
#include <QtWidgets/QApplication>
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>
#include <string>
#include<time.h>
using namespace cv;
using namespace std;
int main(int argc, char** argv)
{
    clock_t beginRead = clock();
    cv::CommandLineParser parser(argc, argv,
        "{help h||}{@image}../data/Crosswalk_02.jpg|}"
    );
    if (parser.has("help"))
    {
        return 0;
    }
    string filename = parser.get<string>("@image");
    if (filename.empty())
    {
        cout << "no image_name provided" << endl;
        return -1;
    }

    Mat inputImage = imread(filename); //incarcarea unei imagini
    clock_t endRead = clock();
    clock_t startProcessing = clock();
    Mat inputImage2 = inputImage.clone();
    Mat src = inputImage.clone();
    int s[360];
    cvtColor(inputImage, src, CV_RGB2GRAY); //convertesc imaginea de la o imagine RGB la una grayscale
    if (src.empty()) //verific daca imaginea este nevida
    {
        help();
        cout << "can not open " << filename << endl;
        return -1;
    }
    Mat img3(src.size(), CV_8U);
    //myGaussianFilter(src,img3); //aplicarea filtrului gaussian
    //imshow("My Gaussian filter", img3); //afisarea imaginei dupa filtrul gaussian
    myMedianFilter(src,img3);
    imshow("My Median Filter", img3);
    mySobelFilter(src, img3); //aplicarea filtrului Sobel
    imshow("My sobel filter", img3);
    double thresh = 120;
    double maxValue = 255;
    Mat dst, cdst;
```

```

int countLines = 0;
int countWhitePixel = 0;
double slope; int maxCountWhitePixels = 0;
int width = src.size().width;
int height = src.size().height;
img3.convertTo(cdst, CV_8UC1);
threshold(cdst, cdst, thresh, maxValue, THRESH_BINARY);
imshow("After threshold", cdst);
vector<Vec2f> lines;
vector<Vec4i> liness;
double distance, distancePQ;
Point fromMatrix1, fromMatrix2;
Point pt1, pt2, pt0, pt, ptt, imagePoint;
vector<Point>locations;
double eps;
HoughLines(cdst, lines, 1, CV_PI / 90, 220, 0, 0); // aplicarea transformatiei Hough
HoughLinesP(cdst, liness, 1, CV_PI / 180, 200, 50, 10); // aplicarea transformatiei Hough
Probabilistica
for (size_t i = 0; i < liness.size(); i++) // desenarea liniilor dupa aplicarea HoughLinesP si
afisarea distantei segmentelor detectate
{
    Vec4i l = liness[i];
    line(inputImage2, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(255, 0, 0), 1, CV_AA);
    distancePQ = sqrt(((l[2] - l[0])*(l[2] - l[0])) + ((l[3] - l[1])*(l[3] - l[1])));
    cout << "distanta HoughLinesP " << distancePQ << endl; // afisarea distantei
}
imshow("detected lines with HoughLinesP", inputImage2);
cout << "width: " << width << " height: " << height << endl; // afisez latimea si inaltimea
imaginei utilizeaza
cout << endl << "lines.size()=" << lines.size() << endl;
for (size_t i = 0; i < lines.size() - 1; i++) // parcurgerea vectorului iesire de la
transformata Hough, vector ce contine perechea rho si theta
{
    eps = lines[i + 1][1] - lines[i][1];
    if (abs(eps) <= 0.3)
    {
        countLines++;
        float rho = lines[i][0], theta = lines[i][1];
        double a = cos(theta), b = sin(theta); // construirea ecuatiei dreptei detectate
        cout << "a=" << a << " b=" << b;
        double x0 = a*rho, y0 = b*rho; // x=r*cos(theta) y=r*sin(theta) => pt0(x0,y0) // y =
        b - ecuatie dreptei mele: y=b, x=x0;
        pt0.x = x0;
        pt0.y = y0; // pt0-punctul meu
        pt1.x = cvRound(x0 + 1000 * (-b)); // creearea celor doua puncte pt1 si pt2 in
        functie de pt0
        pt1.y = cvRound(y0 + 1000 * (a));
        pt2.x = cvRound(x0 - 1000 * (-b));
        pt2.y = cvRound(y0 - 1000 * (a));
        pt.y = pt1.y; // ecuatie liniei mele orizontale y=b
        pt.x = 0; // ecuatie liniei mele verticale
        cout << " pt1 " << pt1;
        cout << " pt2 " << pt2 << endl;
        if (pt1.x > 0 || pt2.x > 0)
        {
    }
}

```

```

        slope = (pt2.y - pt1.y) / (pt2.x - pt1.x); //calculul ecuatiei dreptei
        cout << "slope: " << slope;
    }
    for (int row = 1; row < height - 1; row++)
    {
        s[row] = 0; //vector pentru contorizarea pixelilor albi
        for (int col = 1; col < width - 1; col++)
        {
            distance = abs(row - pt1.y); //distanta dintre pixelii de pe
imagine si dreptele mele
            if (distance == 0) //daca distanta este 0(adica ma aflu pe linia
dorita)
            {
                imagePoint.x = row;
                imagePoint.y = col;
                if (cdst.at<uchar>(row, col) == 255) //verific daca
culoarea pixelului este alb
                {
                    cv::circle(inputImage, cv::Point(col, row), 1,
cv::Scalar(0, 0, 255.0)); //desenez un cerc de raza 1
                    s[row]++; //contorizez nr de pizeli albi gasiti pe o
linie
                }
            }
        }
        if (s[row] > 60)
            cout << "pe linia " << row << " nr de pixeli albi este " << s[row]
<< endl;
    }
}
if (inputImage.type() == CV_8UC1)
{
    cvtColor(inputImage, inputImage, CV_GRAY2RGB);
}
cout << endl << "countLines->" << countLines << endl;
if (countLines >= 5)
{
    cout << "trecere";
}
cout << endl;
// Filtrul Sobel din OpenCV
Mat src_gray;
Mat grad;
char* window_name = "Sobel Demo - Simple Edge Detector";
int scale = 1;
int delta = 0;
int ddepth = CV_16S;
int c;
GaussianBlur(src, src, Size(3, 3), 0, 0, BORDER_DEFAULT);
// Conversia imaginii la o imagine alb-negru
cvtColor(src, src_gray, COLOR_GRAY2BGR);
namedWindow(window_name, CV_WINDOW_AUTOSIZE);
Mat grad_x, grad_y;
Mat abs_grad_x, abs_grad_y;

```

```

// Gradient X
Sobel(src_gray, grad_x, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT);
convertScaleAbs(grad_x, abs_grad_x);
// Gradient Y
Sobel(src_gray, grad_y, ddepth, 0, 1, 3, scale, delta, BORDER_DEFAULT);
convertScaleAbs(grad_y, abs_grad_y);
// Gradientul Total (aproximativ)
addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, grad);
clock_t endProcessing = clock();
clock_t end = clock();
double time_spentProcessing = (double)(endProcessing - startProcessing) / CLOCKS_PER_SEC;
double time_forRead = (double)(endRead-beginRead) / CLOCKS_PER_SEC;
cout << "timpul de procesare este " << time_spentProcessing << endl;
cout << "timpul pentru citirea imaginii este " << time_forRead << endl;
imshow(window_name, grad);
imshow("source", src);
imshow("detected lines", inputImage);
waitKey();
return 0;
}
source.cpp
#include "hello.h"
void exit()
{
    QApplication::exit();
}
void myGaussianFilter(Mat img, Mat &img3)
{
    Mat mask = ((Mat<unsigned char>(3, 3) << 1, 2, 1, 2, 4, 2, 1, 2, 1));
    Mat img2 = img.clone();
    int cols = img.cols;
    int rows = img.rows;
    int i, j;
    cout << mask << endl << endl;
    for (i = 1; i < rows - 1; i++)
    {
        for (j = 1; j < cols - 1; j++)
        {
            img3.at<unsigned char>(i, j) = (mask.at<unsigned char>(0, 0)*(img2.at<unsigned char>(i - 1, j - 1)) +
                mask.at<unsigned char>(1, 0)*(img2.at<unsigned char>(i, j - 1)) +
                mask.at<unsigned char>(2, 0)*(img2.at<unsigned char>(i + 1, j - 1)) +
                mask.at<unsigned char>(0, 1)*(img2.at<unsigned char>(i - 1, j)) +
                mask.at<unsigned char>(1, 1)*(img2.at<unsigned char>(i, j)) +
                mask.at<unsigned char>(2, 1)*(img2.at<unsigned char>(i, j + 1)) +
                mask.at<unsigned char>(0, 2)*(img2.at<unsigned char>(i - 1, j + 1)) +
                mask.at<unsigned char>(1, 2)*(img2.at<unsigned char>(i, j + 1)) +
                mask.at<unsigned char>(2, 2)*(img2.at<unsigned char>(i + 1, j + 1))) / 16.0;
        }
    }
    void help()
    {
        cout << "\nHelp please!!!!This program demonstrates line finding with the Hough transform.\n"
        << endl;
    }
}

```

```

}

void mySobelFilter(Mat img, Mat &img3)
{
    Mat maskGy = ((Mat<char>(3, 3) << -1, 0, 1, -2, 0, 2, -1, 0, 1));
    Mat maskGx = ((Mat<char>(3, 3) << -1, -2, -1, 0, 0, 0, 1, 2, 1));
    int cols, rows;
    int width = img.size().width;
    int height = img.size().height;
    int Gx = 0, Gy = 0;
    double sum = 0, max = 0;
    cout << "Gy " << maskGy << endl << endl;
    cout << "Gx " << maskGx << endl << endl;
    for (int row = 1; row<height - 1; row++)
    {
        for (int col = 1; col<width - 1; col++)
        {
            Gy = maskGy.at< char>(0, 0)*(img.at< char>(row - 1, col - 1)) +
                maskGy.at< char>(1, 0)*(img.at< char>(row, col - 1)) +
                maskGy.at< char>(2, 0)*(img.at< char>(row + 1, col - 1)) +
                maskGy.at< char>(0, 1)*(img.at< char>(row - 1, col)) +
                maskGy.at< char>(1, 1)*(img.at< char>(row, col)) +
                maskGy.at< char>(2, 1)*(img.at< char>(row + 1, col)) +
                maskGy.at< char>(0, 2)*(img.at< char>(row - 1, col + 1)) +
                maskGy.at< char>(1, 2)*(img.at< char>(row, col + 1)) +
                maskGy.at< char>(2, 2)*(img.at< char>(row + 1, col + 1));

            Gx = maskGx.at< char>(0, 0)*(img.at< char>(row - 1, col - 1)) +
                maskGx.at< char>(1, 0)*(img.at< char>(row, col - 1)) +
                maskGx.at< char>(2, 0)*(img.at< char>(row + 1, col - 1)) +
                maskGx.at< char>(0, 1)*(img.at< char>(row - 1, col)) +
                maskGx.at< char>(1, 1)*(img.at< char>(row, col)) +
                maskGx.at< char>(2, 1)*(img.at< char>(row + 1, col)) +
                maskGx.at< char>(0, 2)*(img.at< char>(row - 1, col + 1)) +
                maskGx.at< char>(1, 2)*(img.at< char>(row, col + 1)) +
                maskGx.at< char>(2, 2)*(img.at< char>(row + 1, col + 1));

            sum = sqrt((Gx*Gx) + (Gy*Gy));
            if (sum > max)
            {
                max = sum;
            }
        }
    }
    for (int row = 1; row < height - 1; row++)
    {
        for (int col = 1; col < width - 1; col++)
        {
            Gy = maskGy.at< char>(0, 0)*(img.at< char>(row - 1, col - 1)) +
                maskGy.at< char>(1, 0)*(img.at< char>(row, col - 1)) +
                maskGy.at< char>(2, 0)*(img.at< char>(row + 1, col - 1)) +
                maskGy.at< char>(0, 1)*(img.at< char>(row - 1, col)) +
                maskGy.at< char>(1, 1)*(img.at< char>(row, col)) +
                maskGy.at< char>(2, 1)*(img.at< char>(row + 1, col)) +
                maskGy.at< char>(0, 2)*(img.at< char>(row - 1, col + 1)) +
                maskGy.at< char>(1, 2)*(img.at< char>(row, col + 1)) +

```

```

        maskGy.at< char>(2, 2)*(img.at< char>(row + 1, col + 1));

Gx = maskGx.at< char>(0, 0)*(img.at< char>(row - 1, col - 1)) +
      maskGx.at< char>(1, 0)*(img.at< char>(row, col - 1)) +
      maskGx.at< char>(2, 0)*(img.at< char>(row + 1, col - 1)) +
      maskGx.at< char>(0, 1)*(img.at< char>(row - 1, col)) +
      maskGx.at< char>(1, 1)*(img.at< char>(row, col)) +
      maskGx.at< char>(2, 1)*(img.at< char>(row + 1, col)) +
      maskGx.at< char>(0, 2)*(img.at< char>(row - 1, col + 1)) +
      maskGx.at< char>(1, 2)*(img.at< char>(row, col + 1)) +
      maskGx.at< char>(2, 2)*(img.at< char>(row + 1, col + 1));
sum = sqrt((Gx*Gx) + (Gy*Gy));
sum = sum / max * 255;
img3.at<unsigned char>(row, col) = sum;
    }
}
//sortarea ferestrei utilizand insertion Sort
void insertionSort(int window[])
{
    int temp, i, j;
    for (i = 0; i < 9; i++) {
        temp = window[i];
        for (j = i - 1; j >= 0 && temp < window[j]; j--) {
            window[j + 1] = window[j];
        }
        window[j + 1] = temp;
    }
}
void myMedianFilter(Mat src, Mat &dst)
{
    //creez fereastra de dimensiune 9
    int window[9];
    dst = src.clone();
    for (int y = 0; y < src.rows; y++)
        for (int x = 0; x < src.cols; x++)
            dst.at<uchar>(y, x) = 0.0;
    for (int y = 1; y < src.rows - 1; y++) {
        for (int x = 1; x < src.cols - 1; x++) {
            //inserare de elemente din imagine in fereastra
            window[0] = src.at<uchar>(y - 1, x - 1);
            window[1] = src.at<uchar>(y, x - 1);
            window[2] = src.at<uchar>(y + 1, x - 1);
            window[3] = src.at<uchar>(y - 1, x);
            window[4] = src.at<uchar>(y, x);
            window[5] = src.at<uchar>(y + 1, x);
            window[6] = src.at<uchar>(y - 1, x + 1);
            window[7] = src.at<uchar>(y, x + 1);
            window[8] = src.at<uchar>(y + 1, x + 1);
            // sortarea ferestrei pentru a gasi medianul
            insertionSort(window);
            dst.at<uchar>(y, x) = window[4];//asignarea elementului median
        }
    }
}

```