



# **SISTEM DE DETECȚIE A SEMNELOR DE CIRCULAȚIE**

Natalia Boncea, Vlad Bartolomei

Grupa 30239



# Cuprins

<b>Cuprins</b>	<b>2</b>
<b>1. Introducere</b>	<b>3</b>
<b>1.1. Contextul temei</b>	<b>3</b>
<b>1.2. Obiectivele propuse</b>	<b>3</b>
<b>1.3. Structura documentației</b>	<b>3</b>
<b>2. Fundamente teoretice</b>	<b>4</b>
2.1. Posibile soluții	4
2.2. Spații de culoare. RGB și HSV	4
2.3. Aria unui obiect	4
2.4. Etichetarea obiectelor. Algoritmul „două treceri”	5
2.5. Detecția unui contur	5
2.6. Închidere (Closing)	6
2.7. Detecția formelor geometrice	6
2.7.1. Elipse și cercuri	6
2.7.2. Octogoane. Triunghiuri și dreptunghiuri	6
<b>3. Proiectare și implementare</b>	<b>8</b>
3.1. Descrierea soluției alese	8
3.2. Schema bloc	10
3.3. Detalii de implementare. Funcționalitățile modulelor	11
3.3.1. Schimbarea spațiului de culoare din BGR în HSV	12
3.3.2. Filtrarea după culoarea roșie, respectiv după culoarea albastră	12
3.3.3. Etichetarea obiectelor	13
3.3.4. Eliminarea reziduurilor	13
3.3.5. Închidere	13
3.3.6. Găsirea contururilor	13
3.3.7. Detecția formelor geometrice	14
3.3.8. Desenarea conturului de încadrare (bounding box)	14
3.4. Manual de utilizare	15
<b>4. Rezultate experimentale</b>	<b>17</b>
<b>5. Concluzii</b>	<b>18</b>
5.1. Rezumat	18
5.2. Posibile dezvoltări ulterioare	18
<b>Bibliografie</b>	<b>20</b>



# 1. Introducere

## 1.1. Contextul temei

Sistemul creat de noi încearcă să rezolve problema identificării semnelor de circulație din imagini color preluate din trafic, asemenea celor făcute cu o cameră de bord. Mai mult decât atât, ne propunem nu doar ca sistemul nostru să identifice semnele de circulație de pe drum, ci să le și clasifice în funcție de formă și de culoare.

## 1.2. Obiectivele propuse

Obiectivele pe care ne propunem să le atingem în cadrul acestui proiect sunt:

- detecția corectă semnelor de circulație roșii și albastre dintr-o imagine color;
- marcarea semnelor detectate pe imaginea sursă, prin încadrarea lor într-un cadran însoțit de denumirea clasei din care fac parte;
- clasificarea lor în conformitate cu categoriile din care fac parte în codul rutier;
- clasificarea semnelor de circulație să se facă în funcție de culoarea dominantă și de forma semnului;
- programul să se poată aplica atât pe imagini, cât și pe înregistrări video.

## 1.3. Structura documentației

Documentația este structurată după cum urmează. Întâi vom prezenta noțiunile teoretice necesare pentru a înțelege procedeele folosite pentru prelucrarea imaginilor. Mai apoi vom aprofunda detaliile de proiectare și implementarea propriu-zisă, pentru ca la final să prezentăm rezultatele experimentale, concluziile noastre și posibilele îmbunătățiri.



## 2. Fundamente teoretice

### 2.1. Posibile soluții

La prima vedere, problema de față (o *clasificare* de obiecte) poate fi rezolvată în două moduri:

- fie prin utilizarea unor algoritmi de învățare de tip Machine Learning, prin aplicarea unei rețele neuronale convoluționale care să accepte un set de date constând în imagini cu semne gata clasificate,
- fie prin metode succesive de procesare a imaginilor, compunând astfel un proces end-to-end.

Se poate lucra cu ambele metode ca atare („full”-ML<sup>1</sup> sau „full”-PI<sup>2</sup>) sau se pot îmbina, formând hibrizi. În timp ce prima metodă poate prelucra setul de date de imagini pentru a evidenția culorile de interes ale semnelor de circulație, cea de-a doua metodă ar putea folosi un algoritm de clasificare al formelor geometrice în urma obținerii unui obiect care să conțină fix semnul dorit.

În cadrul acestui proiect am decis să mergem pe o abordare „full”-PI în care vom extrage semnele și vom folosi algoritmi de detecție al formelor geometrice fără intervenția inteligenței artificiale.

### 2.2. Spații de culoare. RGB și HSV

Un pixel color este obținut în general dintr-o combinație de componente, cunoscute în terminologia procesării de imagini drept *canale*.

Pentru spațiul de culoare RGB, culoarea fiecărui pixel se obține prin combinația a trei culori primare: roșu, verde și albastru (Red, Green și Blue). Astfel, fiecare pixel din imagine va fi caracterizat prin câte o valoare pentru fiecare din cele trei componente de culoare primare.

Spațiul de culoare HSV este un spațiu/model de culoare invariant (componentele H (culoare) și S (saturație) sunt separate și cvasi-independente de iluminarea scenei caracterizată de V (intensitate)). Se reprezintă sub forma unei piramide cu baza hexagonală sau a unui con.

### 2.3. Aria unui obiect

Aria este măsurată în pixeli și indică mărimea relativă a obiectului. Aceasta este egală cu suma intensităților tuturor pixelilor care compun obiectul

---

<sup>1</sup> Machine Learning.

<sup>2</sup> Procesare de Imagini.



## 2.4. Etichetarea obiectelor. Algoritm „două treceri”

Etichetarea poate fi realizată prin două parcurgeri liniare pe imagine, plus o procesare suplimentară pe un graf mult mai mic. Această abordare folosește mai puțină memorie. Deoarece în algoritm bazat pe BFS aveam nevoie de memorarea unei liste de puncte, în cazul în care o componentă conexă este foarte mare dimensiunea listei poate fi comparabilă cu dimensiunea imaginii.

Acest algoritm va face o primă parcurgere și va genera etichete inițiale pentru pixelii obiect. Pentru fiecare pixel vom lua în considerare pixelii deja vizitați și etichetați, deci vom folosi vecinătatea pixelilor anteriori definită mai sus, np. După inspectarea etichetelor pixelilor deja vizitați, avem următoarele cazuri:

- Dacă nu avem vecin anterior etichetat, vom crea o etichetă nouă
- Dacă avem vecini etichetați, vom selecta minimul acestor etichete (notat cu x).

După aceea, vom marca fiecare etichetă y din vecinătate, diferită de x, ca echivalentă cu x.

Vom asigna eticheta găsită în pasul anterior pentru poziția curentă. După prima trecere, există câte o etichetă pentru fiecare poziție din imagine. Totuși, vor exista etichete diferite pentru același obiect, care sunt etichete echivalente, deci va trebui să fie înlocuite cu o etichetă nouă, care reprezintă o clasă de echivalență.

Relațiile de echivalență definesc un graf neorientat, având ca noduri etichetele inițiale. Acest graf este de obicei mult mai mic decât graful inițial al pixelilor obiect, deoarece numărul de noduri este numărul de etichete generate la prima parcurgere. Muchiile grafului sunt relațiile de echivalență. Putem aplica algoritmul 1 pe acest graf mai mic, pentru a obține o nouă listă de etichete. Toate etichetele echivalente cu 1 se re-etichetează cu 1, următoarea componentă conexă ne-echivalentă cu 1 se re-etichetează cu 2, și aşa mai departe. O nouă trecere prin matricea de etichete inițiale va realiza re-etichetarea.

## 2.5. Detectia unui contur

Algoritmul de urmărire a conturului este folosit pentru extragerea conturului obiectelor dintr-o imagine. La aplicarea acestui algoritm presupunem că imaginea este binară sau că obiectele din imagine au fost etichetate în prealabil.

Pașii algoritmului:

1. Se scanăză imaginea din colțul stânga sus până când se găsește un pixel care aparține unei regiuni; acest pixel P0 reprezintă pixelul de start al conturului regiunii. Se definește o variabilă dir în care se reține direcția mutării anterioare de-a lungul conturului de la elementul anterior spre elementul curent. Se inițializează:

- (a) dir = 0 dacă conturul este detectat folosind vecinătate de 4
- (b) dir = 7 dacă conturul este detectat folosind vecinătate de 8

2. Se parcurge vecinătatea de  $3 \times 3$  a pixelului curent în sens invers acelor de ceasornic, începând cu pixelul corespunzător poziției:



(a)  $(\text{dir} + 3) \bmod 4$

(b)  $(\text{dir} + 7) \bmod 8$  dacă dir este par,  $(\text{dir} + 6) \bmod 8$  dacă dir este impar

Primul pixel găsit care are aceeași valoare ca și pixelul curent este noul element  $P_n$  al conturului. Se actualizează valoarea lui dir.

3. Dacă elementul curent  $P_n$  al conturului este egal cu al doilea element  $P_1$  din contur și dacă elementul anterior  $P_{n-1}$  este egal cu primul element  $P_0$ , atunci algoritmul se încheie. Altfel se repetă pasul (2).

4. Conturul detectat este reprezentat de pixelii  $P_0 \dots P_{n-2}$ .

## 2.6. Închidere (Closing)

Pentru a înțelege *închiderea* este necesar să parcurgem noțiunile de *dilatație* și *eroziune*. Dilatația (dilation) în procesarea imaginilor este o operațiune morfologică care mărește regiunile luminoase<sup>3</sup> ale unei imagini, adăugând pixeli la marginile acestora. Eroziunea (erosion) este operațiunea inversă, care micșorează regiunile luminoase<sup>4</sup>, îndepărând pixeli de la marginile acestora.

*Închiderea* constă într-o *dilatăre* urmată de o *eroziune* și poate fi folosită pentru umplerea de goluri sau mici discontinuități.

## 2.7. Detectia formelor geometrice

### 2.7.1. Elipse și cercuri

Un contur dat trebuie să aibă minim 5 puncte pentru a fi considerat elipsă. Aceasta este un criteriu necesar pentru a continua detectia cercurilor. Mai departe se calculează intersecția a două imagini binare, una conținând conturul obiectului detectat, iar a doua cea a elipsei detectate. Această intersecție va conține doar părțile care se suprapun din cele două contururi. În final se numără pixelii nenuli din imaginea intersecție rezultat și din imaginea contur dată inițial și se face un raport:

$$\text{raport} = \frac{\text{număr de pixeli nenuli din imaginea intersecție}}{\text{număr de pixeli nenuli din imaginea contur dată}} > \text{prag ales } [0, 1]$$

După cum se va observa în codul sursă, am considerat că un prag bine ales pentru aplicația noastră este 0.6, adică minim 60% potrivire.

O parte importantă a acestui algoritm este detectia elipselor. Aceasta se face prin Direct Least Square Fitting of Ellipses<sup>5</sup>, metodă ce implică rezolvarea unui sistem algebraic ce minimizează suma pătratelor distanțelor de la puncte către elipsă. Cert este că metoda este mult prea complexă pentru a fi expusă în interiorul acestei documentații.

<sup>3</sup> În cazul nostru este vorba despre pixelii obiect.

<sup>4</sup> Idem.

<sup>5</sup> A se vedea referințele.



## 2.7.2. Octogoane. Triunghiuri și dreptunghiuri

Având conturul închis al unui obiect dat, se poate folosi un algoritm de aproximare al acestuia, aşa încât să reducem problema la numărul de edge-uri pe care algoritmul respectiv îl returnează. Se vede imediat că un triunghi va avea 3 laturi, un dreptunghi va avea 4 laturi, iar un octogon 8 laturi.

În spatele funcției de aproximare folosite<sup>6</sup> de noi se află [algoritmul lui Ramer-Douglas-Peucker](#). Se dă o linie poligonală cu o mulțime de puncte intermediare, iar rezultatul va fi o linie poligonală cu mai puține puncte. La început se cunoaște și distanța  $\epsilon > 0$  dintre punctul inițial și final.

Algoritmul împarte recursiv linia. Automat, marchează primul și ultimul punct pentru a fi păstrate. Apoi găsește punctul care este cel mai îndepărtat de segmentul de linie având drept capete primul și ultimul punct; acest punct este evident cel mai îndepărtat de pe curbă față de segmentul de linie aproximativ între punctele finale. Dacă punctul este mai aproape decât  $\epsilon$  de segmentul de linie, atunci orice puncte care nu sunt marcate în prezent pentru a fi păstrate pot fi eliminate fără ca curba simplificată să fie mai rea decât  $\epsilon$ .

Dacă punctul cel mai îndepărtat de segmentul de linie este la o distanță mai mare decât  $\epsilon$  de aproximare, atunci acel punct trebuie păstrat. Algoritmul se apelează recursiv cu primul punct și punctul cel mai îndepărtat, apoi cu punctul cel mai îndepărtat și ultimul punct, ceea ce include marcarea punctului cel mai îndepărtat pentru a fi păstrat.

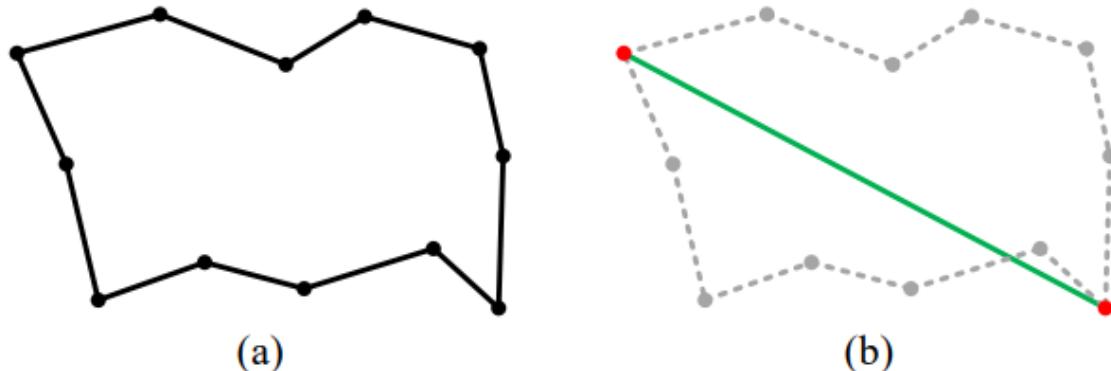
Când recursivitatea este completă, poate fi generată o nouă curbă de ieșire constând din toate și numai acele puncte care au fost marcate pentru a fi păstrate.

Problema cu care ne confruntăm este evidentă: algoritmul funcționează pe linii poligonale, iar nu pe poligoane, ceea ce înseamnă automat contururi deschise. În urma unei rulări a unui contur închis pe algoritmul prezentat mai sus observăm echivalența dintre primul și ultimul punct și, implicit, distanța  $\epsilon = 0$ . Acestea conduc către același poligon ca înainte, neavând loc în fapt nici o simplificare. Pentru a funcționa pe un contur închis, acest algoritm are nevoie de o ajustare.

Dându-se conturul inițial, se iau de pe acesta cele mai depărtate două puncte. Alegerea primului punct este irelevantă, având în vedere că al doilea punct este determinat de primul și de distanța  $\epsilon$ .

---

<sup>6</sup> cv::approxPolyDP



Practic problema este împărțită în două subprobleme de linii poligonale. La final, cele două soluții sunt combinate pentru a rezulta soluția dorită.

### 3. Proiectare și implementare

#### 3.1. Descrierea soluției alese

Pentru a putea extrage semnele de circulație dintr-o imagine, mai întâi aceasta va fi convertită în spațiul de culoare HSV<sup>7</sup>, oferindu-ne un control mai mare asupra *luminozității* și *saturației* culorilor. Mai apoi vom furniza conținutul imaginii pe două canale separate (roșu și albastru) în urma ajustării celor două componente anterior menționate. Imaginele vor trebui reparate printr-un proces de Closing<sup>8</sup>. O posibilă problemă este apariția imperfecțiunilor sau a micilor (sau *marilor*, dacă ne gândim și la autovehicule) obiecte de aceeași culoare ca cea a semnului dorit, pe care o vom adresa prin etichetarea obiectelor și filtrarea în funcție de arie<sup>9</sup>. Aceasta a fost prima parte a soluției, referitoare la obținerea obiectelor-candidat pentru a fi clasificate.

A doua parte a soluției o constituie determinarea formei obiectului candidat. Pentru fiecare obiect din imagine se determină conturul (închis), acesta urmând a fi aproximat din punct de vedere al numărului de muchii. Bazându-ne pe numărul de muchii obținut putem determina o primă clasificare a obiectelor în forme geometrice (cerc, triunghi, dreptunghi/pătrat, octogon). Urmând mai apoi reglementările codului rutier în vigoare și ținând cont de canalul (roșu sau albastru) de pe care au venit obiectele, precum și de forma obținută, putem clasifica semnele conform figurii 3.1.1.:

<sup>7</sup> Hue, Saturation, Value (Luminosity).

<sup>8</sup> Închidere, constând într-o dilatare a imaginii, urmată de o eroziune.

<sup>9</sup> Numărul de pixeli ce compun un obiect.

**TECHNICAL UNIVERSITY**

OF CLUJ-NAPOCA, ROMANIA

**COD RUTIER**

Chestionare Curs Video Codul rutier **Indicatoare rutiere** RCA Online Alerta ▾

### Indicatoare și marcaje rutiere

De avertizare

[Învață](#)

De interzicere sau restricție

[Învață](#)

De prioritate

[Învață](#)

De obligare

[Învață](#)

De informare

[Învață](#)

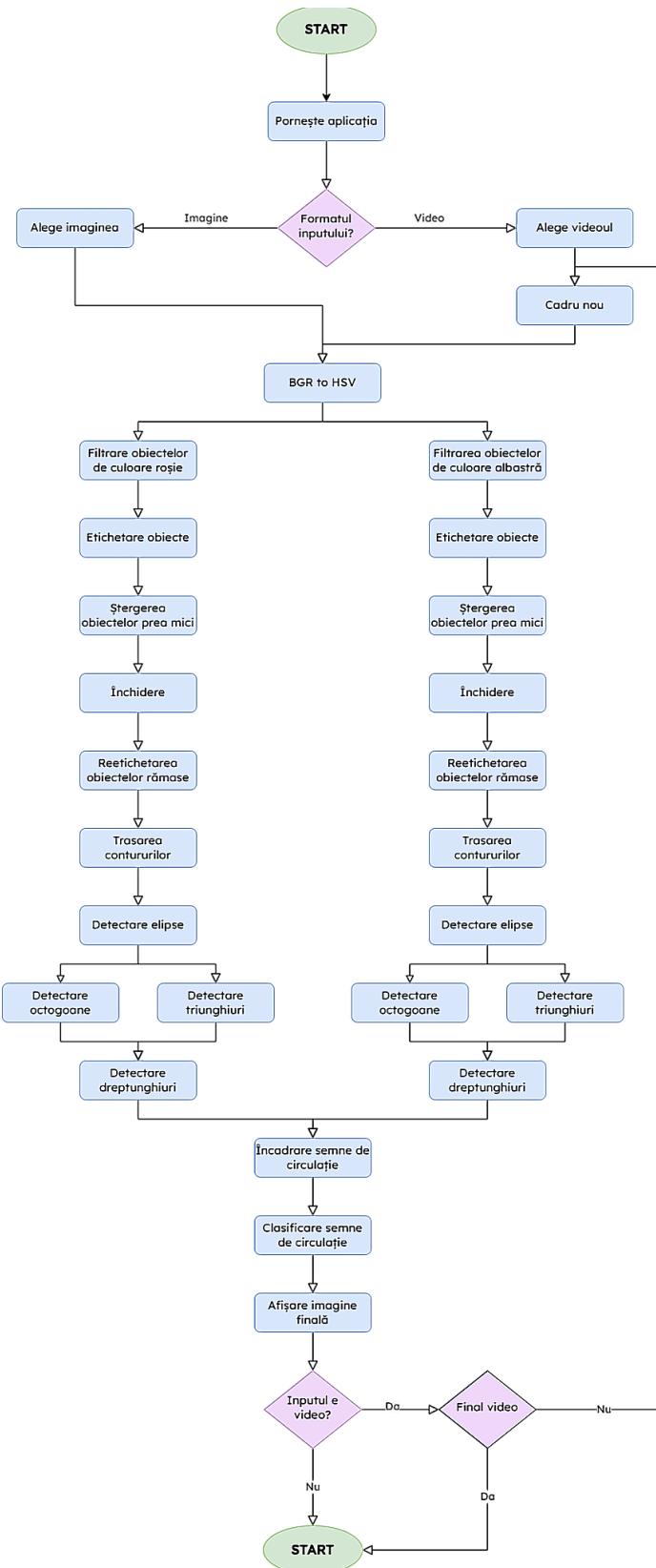
De orientare

[Învață](#)

**Figura 3.1.1. Clasificarea semnelor de circulație din România**



### 3.2. Schema bloc





### 3.3. Detalii de implementare. Funcționalitățile modulelor

Următorul tabel prezintă cine s-a ocupat de care funcționalități din cadrul proiectului. Ordinea menționării funcționalităților sau a dezvoltatorilor nu are nicio însemnătate specifică.

Topic	Dezvoltator(i)
Pregătirea repository-ului și a framework-ului de lucru	Vlad
Stabilirea strategiei de lucru aferente rezolvării problemei propuse	Vlad, Natalia
Conversia spațiului de culoare și filtrarea după cele 2 culori	Natalia
Eliminarea reziduurilor din imagini și corectarea imaginilor	Natalia
Etichetarea obiectelor	Natalia
Detectia formelor geometrice	Natalia, Vlad
Clasificarea conformă codului rutier în vigoare	Vlad
Asamblarea tuturor componentelor într-o versiune finală funcțională a proiectului	Vlad, Natalia
Suport pentru input foto și video	Vlad
Întocmirea documentației	Vlad, Natalia
Aprovizionarea cu imagini/videos de test	Vlad

Proiectul se bazează pe librăria OpenCV<sup>10</sup> și are următoarele componente:

- OpenCVApplication.cpp - conținând
  - o funcție `callPipeline` responsabilă cu implementarea [workflow-ului discutat](#)
  - o metodă `main()` în care se afișează interfața de întâmpinare din consolă și se apelează acel `callPipeline`

<sup>10</sup> Importată local în proiect.



- Pachetul Demo (Demo.cpp și Demo.h) - conținând funcțiile cu care framework-ul<sup>11</sup> a venit inițial
- Pachetul TSR<sup>12</sup> (TSR.cpp și TSR.h) - conține implementarea funcțiilor chemate de *callPipeline*. Aici se desfășoară magia întregului cod.

### 3.3.1. Schimbarea spațiului de culoare din BGR în HSV

O primă filtrare a semnelor de circulație pe care ne-am propus să o facem a fost filtrarea după culoare. Am încercat ca setul de imagini ales de noi să fie unul cât mai apropiat de realitate, astfel că imaginile noastre de test au prezentat multe diferențe ori imperfecțiuni date de condițiile de mediu sau de calitatea camerei. Alegerea ariei de nuanțe potrivite pentru semnele noastre ar fi fost destul de dificilă în mediul BGR, motiv pentru care am ales să facem trecerea imaginii noastre în spațiul de culoare HSV. Algoritmul poate fi regăsit sub această semnătură:

```
void BGR2HSV(Mat src, Mat H, Mat S, Mat V);
```

Această funcție primește ca parametrii imaginea sursă, alături de matricile celor 3 canale H, S, V, rezultate în urma transformării. Pentru început valorile din spațiul BGR sunt normalize, după care sunt calculate valorile HSV conform formulelor de conversie specifice. În final, scalăm cele 3 valori obținute în intervalul [0, 255].

### 3.3.2. Filtrarea după culoarea roșie, respectiv după culoarea albastră

După ce am aflat valorile pixelilor în spațiul de culoare HSV, a urmat filtrarea imaginilor în funcție de cele 2 culori alese pentru identificarea semnelor de circulație. În urma a mai multor experimente cu diverse poze și videouri, am reușit să reglez parametrii în aşa fel încât să îmi recunoască o mare parte din semnele de circulație, fără a avea prea multe zone nedorite (background). Trecerea în spațiul HSV mi-a dat o mai mare flexibilitate în alegerea nuanțelor potrivite, întrucât intensitatea culorii era reglată din parametrul **s**, din parametrul **v** am controlat cât de luminoasă să fie culoarea, iar din parametrul **h** am reglat cât de influențată să fie culoarea roșie sau albastră de celelalte culori de bază. Toată această filtrare are loc în funcția cu semnătura:

```
Mat filterbyRed(Mat H, Mat S, Mat V);  
Mat filterbyBlue(Mat H, Mat S, Mat V);
```

Logica din spatele celor 2 funcții este aproape identică. Se face parcurgerea celor 3 canale de culoare date ca parametru concomitent, timp în care se verifică dacă pixelul de pe o anumită poziție se încadreză în limitele impuse pe toate canalele de culoare. Dacă da, atunci pixelul respectiv este transpus în matricea destinație cu valoarea 255 (alb). Altfel, pe poziția respectivă este pusă valoarea 0 (negru). În final, se returnează matricea filtrată.

<sup>11</sup> În fapt acesta a fost furnizat la laboratorul de PI, având scopuri didactice.

<sup>12</sup> Traffic Sign Recognition



### 3.3.3. Etichetarea obiectelor

După cum am evidențiat și la [fundamente teoretice](#), algoritmul folosit pentru etichetare este cel „cu două treceri”, alias „cu două clase de echivalență”. Implementarea sa va fi regăsită în cod sub această semnătură:

```
Mat douaTreceri(Mat img, int* labelSize);
```

și funcționează după principiul expus la capitolul 2.

Etichetarea este realizată de două ori. Primul apel al funcției de etichetare contribuie la curățarea imaginilor filtrare de reziduuri și pregătirea pentru detecția formelor. Al doilea apel intervine în procesul de detecție a formelor, prin delimitarea clară a obiectelor pentru care urmează să se traseze conturul.

### 3.3.4. Eliminarea reziduurilor

În urma filtrării după culoare, ne-am confruntat cu problema obținerii unor imagini cu multe zone nedorite. Pentru a scăpa de o parte din ele, o primă operație a fost să scăpăm de zonele cu aria mai mică de 1000 de pixeli. Algoritmul care efectuează această operație se găsește în cod sub semnătura:

```
Mat deleteSmallObj(Mat labels, Mat src, int labelSize);
```

Această funcție numără pixelii în funcție de eticheta pe care o au, iar rezultatele sunt stocate într-un array. La următoarea parcurgere a matricii sursă se vor pune în destinație doar acei pixeli al căror etichetă a corespuns unei arii mai mari de 1000 de pixeli.

### 3.3.5. Închidere

Dat fiind diversitatea pozelor și diversele condiții în care acestea au fost făcute, rezultatul în urma filtrării pe baza culorii putea suferi anumite neregularități, ca de exemplu conturul semnelor nu era un poligon închis. Pentru a soluționa această problemă, am ales să aplicăm o închidere, utilizând algoritmul definit în secțiunea de [fundamente teoretice](#). Implementarea acestei tehnici de corecție a imaginilor se regăsește în cod sub semnătura:

```
Mat inchidere(Mat src, int dim);
```

### 3.3.6. Găsirea contururilor

Aici a apărut una dintre dilemele proiectului. S-ar fi putut folosi ușor cv::findContours care livra într-o listă de liste de puncte toate contururilor găsite, dar din cauza restricțiilor de evaluare impuse am ales să îl implementăm de la 0. Algoritmul poate fi regăsit sub această semnătură:

```
void findContoursLaMană(Mat src, vector<vector<Point>> &contours)
```

și este o funcție de bucătărie internă. Aceasta se bazează pe o etichetare a obiectelor depozitată într-o matrice nouă **labels**. Este nevoie de o singură traversare a acestui **labels** și un



contor **currentLabel**. Când un label nou este găsit, se apelează funcția de găsire a conturului închis pentru un obiect, se stochează în **contours** și se incrementează **currentLabel**. Dacă se ajunge în matrice din nou la un pixel găsit anterior, acesta nu mai este luat în seamă.

### 3.3.7. Detectia formelor geometrice

Formele geometrice pe care le-am avut de identificat sunt: elipsă, octogon, triunghi și dreptunghi. Pe baza contururilor trasate și salvate într-un vector de vectori de puncte, am identificat mai întâi dacă formele sunt elipse, conform algoritmul definit la secțiunea de fundamente teoretice. Coeficientul raportului de puncte situate pe conturul unui cerc ne împărtea formele în 3 categorii: elipse (pentru o valoare cât mai aproape de 1), triunghiuri și dreptunghiuri (pentru o valoare cât mai aproape de 0), respectiv octagoane și dreptunghiuri (pentru valori pe la mijlocul intervalului [0, 1]. Încadrarea dreptunghiului în 2 categorii se datorează imperfecțiunilor conturului în anumite situații.

Pentru celelalte 3 forme geometrice, s-a utilizat o funcție de aproximare a conturului poligonului, care ne returna numărul de vârfuri, criteriu suficient după care puteam face clasificarea.

Cele 4 funcții de detectie a formelor geometrice au următoarele semnături:

```
Mat detectShapes(Mat binaryImage);
void detectOctagon(Mat& resultImage, const vector<vector<Point>>& contours, int index);
void detectRectangle(Mat& resultImage, const vector<vector<Point>>& contours, int index);
void detectTriangles(Mat& resultImage, const vector<vector<Point>>& contours, int index);
```

### 3.3.8. Desenarea conturului de încadrare (bounding box)

Total pleacă de la această semnătură:

```
void drawBoundingBox(Mat src, vector<vector<Point>> contours,
                     boolean isRed, int index, int classification)
```

Metoda este apelată de mai sus (din *detectShapes*) la fiecare contur găsit, astfel că la RunTime/momentul apelului se cunosc:

- **Indexul** curent al conturului din **contours**;
- Dacă semnul găsit **isRed** sau nu, influențând clasificarea conformă codului rutier;
- **Clasificarea** care dictează propriu zis însemnatatea semnului în corelație cu forma sa geometrică.

Pentru conturul curent se determină punctele stânga-sus și dreapta-jos prin determinarea lui  $x_{\min}$ ,  $y_{\min}$ ,  $x_{\max}$  și  $y_{\max}$ , spre a servi desenării ulterioare a bounding box-ului<sup>13</sup>

<sup>13</sup> Pentru aceasta se apelează funcția cv::rectangle.



### 3.4. Manual de utilizare

În urma lansării în execuție a fișierului executabil, utilizatorul este întâmpinat de o consolă (command prompt) în care acesta poate alege să testeze clasificarea semnelor de circulație dintr-un material foto (*comanda 1*) sau video (*comanda 2*). Părăsirea aplicației se face prin *comanda 0*.

 C:\Users\Vlad\Documents\FACULTA\AN3\SEM2\1. PROIECT\_PI\TrafficSignDetection\TSD\x64\Release\OpenCVApplication.exe

Choose the desired mode:

- 1 - Photo
- 2 - Video

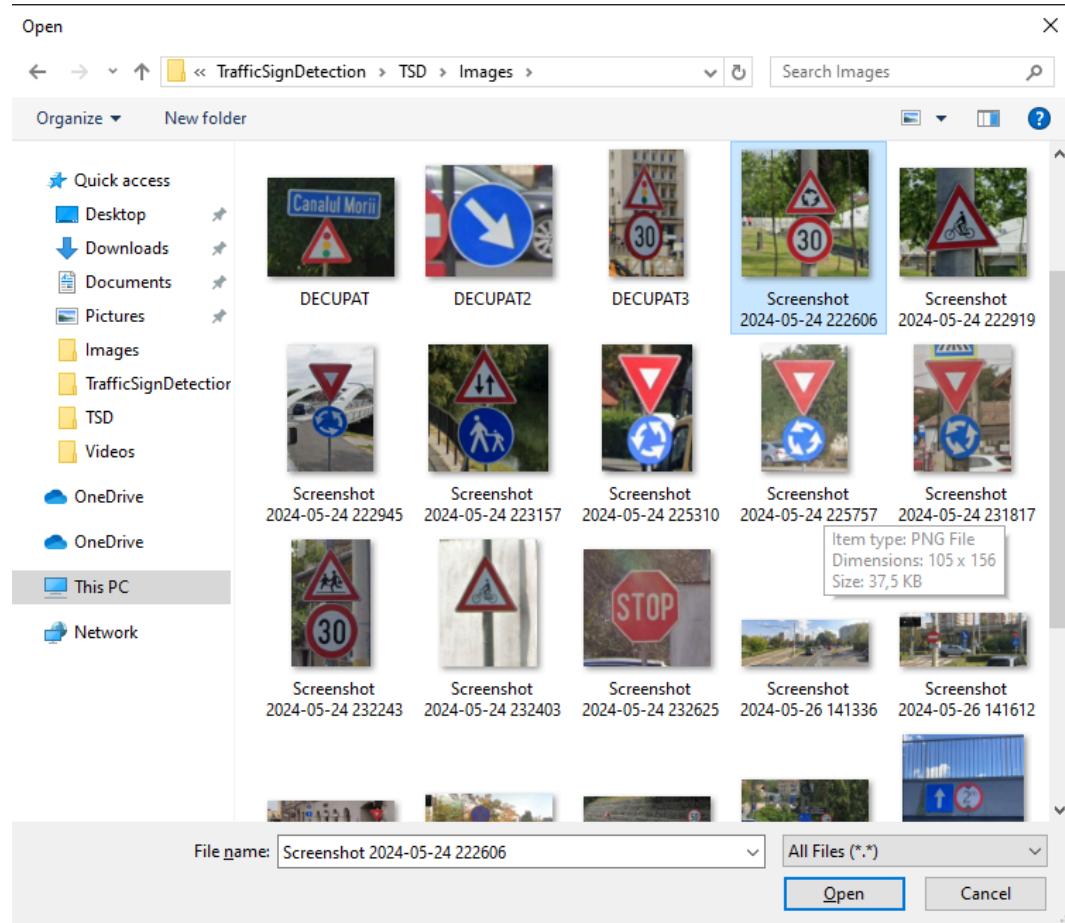
Press 0 to exit

Option: -

```
C:\Users\Vlad\Documents\FACULTA\AN3\SEM2\1. PROIECT_PI\TrafficSignDetection\TSD\x64\Release\OpenCVApplication.exe
Choose the desired mode:
  1 - Photo
  2 - Video
Press 0 to exit

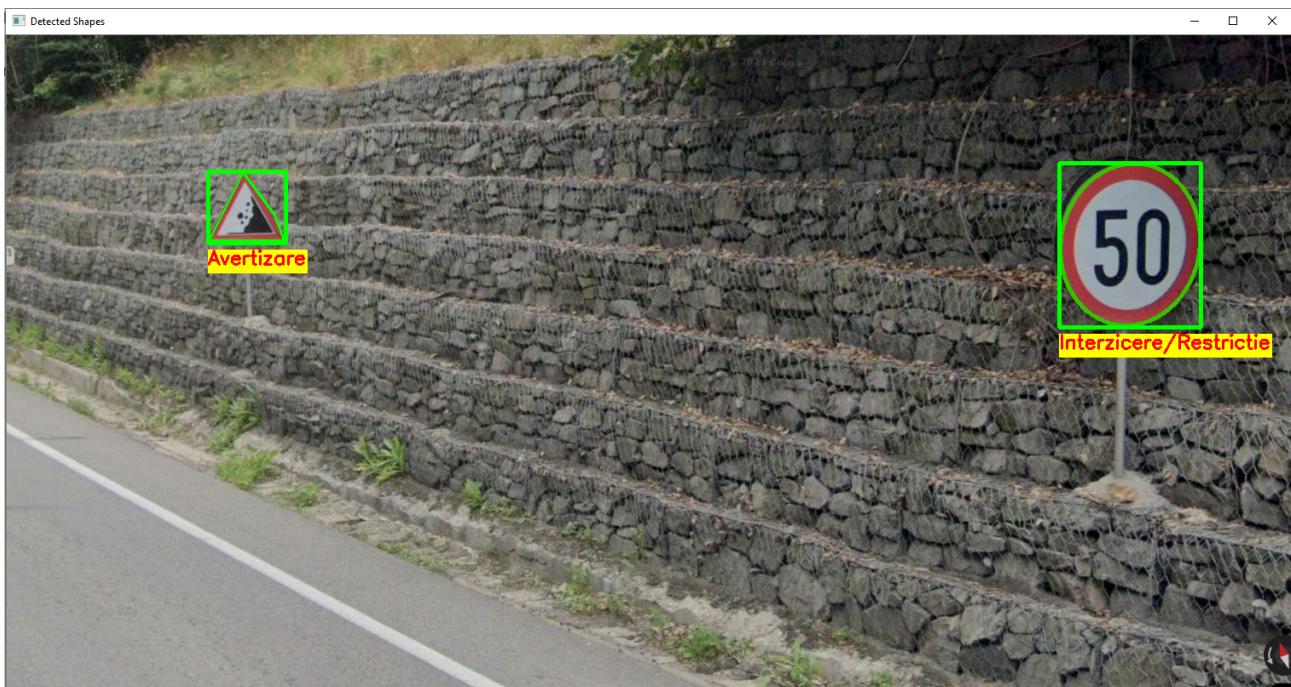
Option: -
```

Continuarea cu oricare mod de execuție va conduce automat la alegerea fișierului dorit din memoria locală.





După selectare, rezultatul final (format foto sau video) va fi afișat, iar prin apăsarea oricărei taste se revine la meniul anterior de selecție al fișierului de test.



Pentru revenire la meniul de selecție al modului de lucru, se va apăsa butonul *Cancel* sau tastă *Esc*.

De asemenea, în modul video, în urma afișării rezultatului, acesta va fi salvat local ca filmare în format .avi.

output\_video

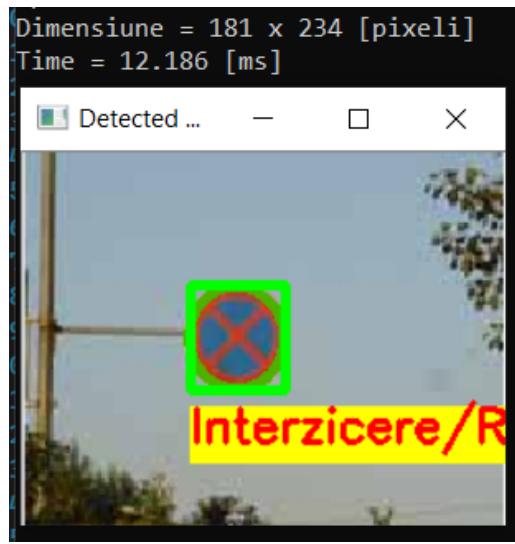
27.05.2024 14:16

AVI File



## 4. Rezultate experimentale

Existența prezentului proiect ar fi de prisos fără prezentarea unor rezultate provenite din fișiere de intrare cât mai variate. În continuare vor fi analizate diferite fotografii conținând semne de circulație și se vor prezenta și comenta rezultatele. În această secțiune a documentației vom include și timpii de rulare pentru diferite fișiere, specificând și dimensiunile de intrare (rezoluție, durată - pentru videos).



Dimensiune = 766 x 1794 [pixeli]  
Time = 175.657 [ms]

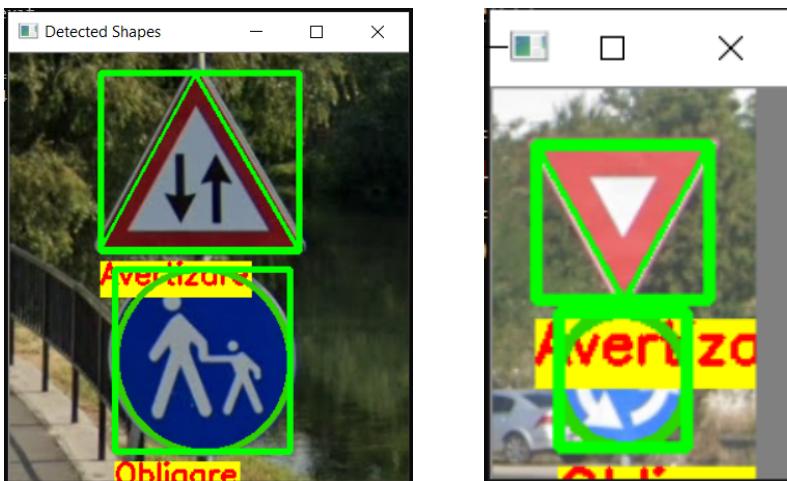


Din cele 2 imagini de mai sus se poate observa că sistemul nostru recunoaște semnele de circulație atât pentru imagini cu o rezoluție slabă, cât și pentru imaginile cu

**TECHNICAL UNIVERSITY**

OF CLUJ-NAPOCA, ROMANIA

rezoluție mare. O altă observație pe care o putem face din cele 2 exemple este legată de timpii de execuție. Timpul de execuție pare să varieze în funcție de dimensiunea imaginii. Chiar și în cazul imaginii de dimensiuni mari, timpul de execuție s-a păstrat sub o secundă, ceea ce consider că este un rezultat destul de bun, dat fiind tot parcursul de procesare al imaginii. Cu toate acestea, ca o viitoare dezvoltare a acestui proiect, timpul de execuție ar putea fi îmbunătățit.



Din cele 2 imagini de mai sus se poate observa gama largă de nuanțe și intensități de culoare pe care le surprinde programul nostru, ceea ce denotă flexibilitatea lui pentru diferite condiții de vreme sau pentru diferite apărăte de fotografiat.

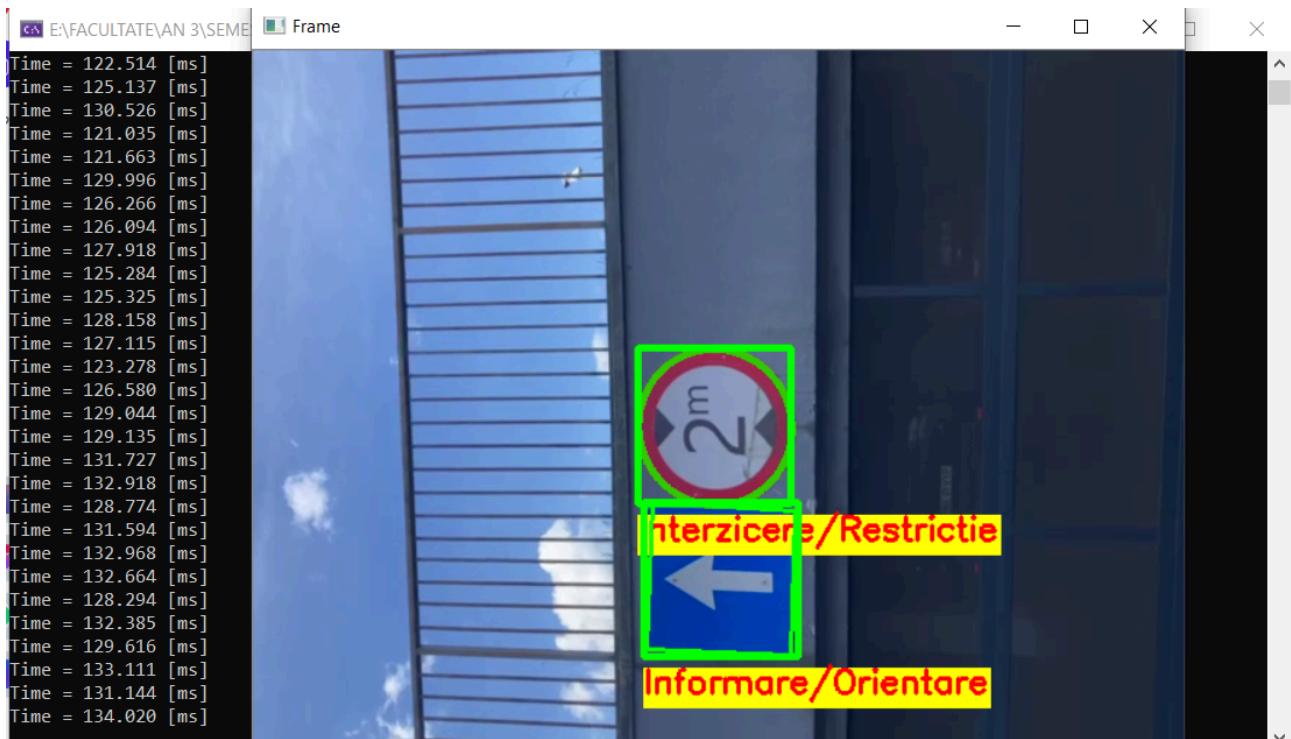
De asemenea, se poate observa cu ușurință faptul că sistemul nostru reușește să se descurce și în cazul prezenței a mai multor semne de circulație într-o singură imagine.





Aplicația pare să evite cu succes prezența a altor obiecte de aceeași culoare cu semnele căutate de noi, ca de exemplu farurile mașinilor sau clădirile albastre din spate.

În urma testării pe un video de 3 secunde, având rezoluție 4K și 60fps, programul reușește să recunoască destul de bine semnele de circulație pentru fiecare frame. Se observă totuși o ușoară încetinire a videoului, ceea ce nu ar fi prea convenabil într-o aplicație reală, însă aceasta rămâne ca o posibilă dezvoltare ulterioară.





## 5. Concluzii

### 5.1. Rezumat

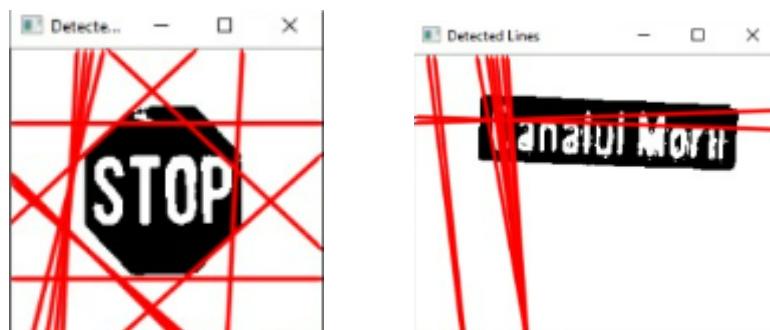
În prezentul document am explicitat concepțele și fundamentele teoretice necesare detecției de indicatoare rutiere. Am prezentat și comentat cele mai relevante părți ale implementării noastre. În final pus în evidență rezultatele muncii noastre prin prezentarea semnelor de circulație detectate pe fișierele foto și video de input.

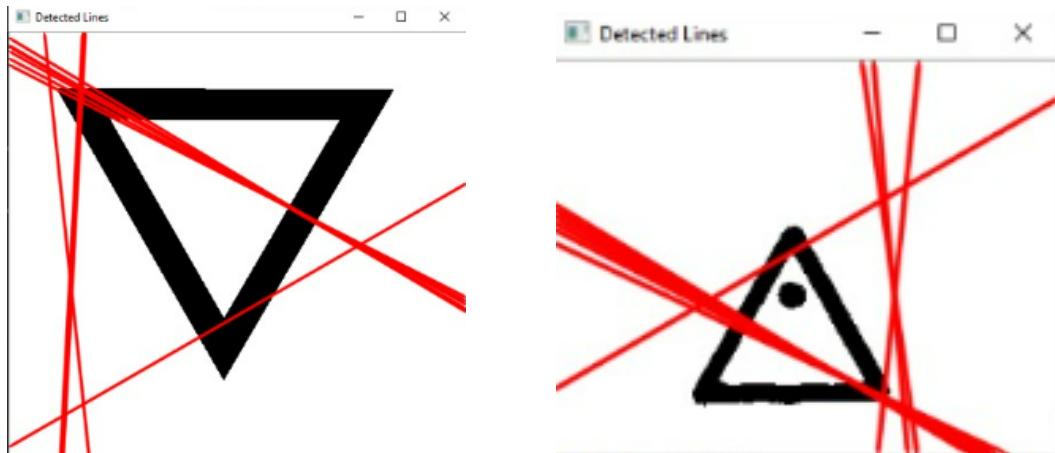
În urma acestor discuții, susținem că proiectul nostru poate servi ca o bază solidă pentru viitoare dezvoltări ale unor sisteme mai complexe de detecție a indicatoarelor rutiere.

### 5.2. Posibile dezvoltări ulterioare

Putem fi de acord cu faptul că aplicația noastră detectează niște clase mari de indicatoare rutiere (de avertizare, de orientare, de obligare și.a.m.d). Pentru a fi relevant în lumea automotivelor, proiectului i se pot aduce modificări pentru a diversifica gama de semne de circulație ce pot fi recunoscute. O primă metodă o poate constitui o însuruire de operații morfologice de a detecta caracteristicile anumitor semne, precum cel de „Semafor în 150m” sau „Limita maximă de viteză admisă este x km/h”, însă ar constitui o mare bătaie de cap și un cod foarte complex pentru foarte multe tipuri de semne. Să nu mai vorbim de faptul că pentru unele semne este foarte dificilă interpretarea matematică a anumitor desene (de exemplu „Atenție, căprioare”). De aceea, o metodă mai bună și de actualitate ar constitui-o construirea unei rețele neuronale convoluționale capabile să recunoască fiecare semn de circulație în parte.

Recunoașterea formelor geometrice ar putea fi reproiectată, schimbând bazarea pe approxPolyDP cu Transformata Hough, lucru care s-a și încercat de altfel în decursul acest proiect, însă fără rezultate notabile și comportament instabil, după cum se poate observa.





De asemenea, s-ar putea încerca aplicarea unui filtru gaussian în timpul preprocesării imaginilor, pentru a mai scăpa de imperfecțiuni. Mai mult decât atât, am putea trata și cazul zgomotelor de tip Salt & Pepper.

O altă idee demnă de încercat în timpul preprocesării imaginilor este o corecție gamma prin *decompresie* pentru a obține un spectru mai larg de culori.

O optimizare care poate fi adusă proiectului este la detecția conturului. La momentul curent obiectele din imagine sunt etichetate, traversate și aplicate pe fiecare în parte un algoritm de detectie a conturului bazat pe vecinătăți de pixeli. Însă un Canny edge detection ar rezolva mult mai rapid problema contururilor, rămânând necesară doar stocarea în memorie a contururilor închise.



# Bibliografie

1. Colour detection & Object Tracking (OpenCV Tutorials):  
<https://www.opencv-srf.com/2010/09/object-detection-using-color-seperation.html>
2. Shape detection & Tracking using Contours (OpenCV Tutorials):  
<https://www.opencv-srf.com/2011/09/object-detection-tracking-using-contours.html>
3. Cod rutier. Clasificarea semnelor de circulație:  
<https://www.codrutier.ro/semne-de-circulatie>
4. Hough transformation implementation c++ (for detecting edges):  
<https://www.keymolen.com/2013/05/hough-transformation-c-implementation.html>
5. Hough transform explanations (mathematically):  
<https://www.youtube.com/watch?v=t1GXMrK9m84&t=263s>
6. Hough transform explanations (computer vision - wise):  
<https://www.youtube.com/watch?v=6yVMpaIxIU>
7. Hough transform explained (german):  
<https://www.youtube.com/watch?v=8fyWszzPCQE&t=23s>
8. O abordare similară a proiectului: <https://durak-bolat.de/rsdc.html>
9. Detecția triunghiurilor:  
<https://berenoune.blogspot.com/2012/10/opencv-c-triangle-detection.html>
10. Detecția elipselor  
<https://stackoverflow.com/questions/35121045/find-cost-of-ellipse-in-opencv>
11. Detecția obiectelor prin separarea culorilor:  
<https://www.opencv-srf.com/2010/09/object-detection-using-color-seperation.html>
12. O implementare a lui Hough Transform:  
<https://www.keymolen.com/2013/05/hough-transformation-c-implementation.html>
13. Detecția contururilor:  
[https://docs.opencv.org/4.x/dd/d49/tutorial\\_py\\_contour\\_features.html](https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html)
14. Algoritmul de aproximare al poligoanelor folosit de cv::approxPolyDP:  
[https://en.wikipedia.org/wiki/Ramer-Douglas-Peucker\\_algorithm](https://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithm)
15. Laboratoarele de PI ale domnului Radu Dănescu
16. Douglas-Peucker algorithm adapatat pentru contururi închise  
[https://www.researchgate.net/publication/303793930\\_FOOTPRINT\\_MAP\\_PARTITIONING\\_USING\\_AIRBORNE\\_LASER\\_SCANNING\\_DATA](https://www.researchgate.net/publication/303793930_FOOTPRINT_MAP_PARTITIONING_USING_AIRBORNE_LASER_SCANNING_DATA)
17. Detecția elipselor prin [Direct Least Square Fitting of Ellipses](#)