

# Detectia semnelor de circulatie din scene de trafic

Vonica Ioan-Paul

## 1 Scopul proiectului

Proiectul are ca scop dezvoltarea unei aplicatii software care detecteaza si marcheaza semnele de circulatie din imagini statice de trafic. Tipurile de semne vizate sunt:

- **STOP**: Un semn octogonal rosu cu textul „STOP”, care impune oprirea completa a vehiculului.
- **YIELD** (Cedeaza trecerea): Un triunghi echilateral cu varful in jos, de obicei rosu si alb, indicand obligatia de a acorda prioritate altor participantii la trafic.
- **NO ENTRY** (Acces interzis): Un cerc rosu cu o banda orizontala alba, ce interzice accesul vehiculelor in directia respectiva.
- **SPEED** (Limitare viteza): Un cerc alb cu bordura rosie si un numar in centru, indicand viteza maxima permisa.

Solutia foloseste metode clasice de procesare a imaginilor, cum ar fi segmentarea bazata pe culoare, detectarea contururilor si clasificarea geometrica, pentru a identifica si eticheta aceste semne. Acest tip de sistem poate fi integrat in sisteme avansate de asistent la conducere (ADAS) sau folosit pentru inventarierea si verificarea semnelor rutiere in cadastru.

## 2 Principiile algoritmice si fundamentarea teoretica

### 2.1 Spatiul de culoare HSV

Algoritmul utilizeaza spatiul de culoare HSV (Hue-Saturation-Value) in loc de RGB pentru segmentarea culorii, deoarece HSV separa informația de culoare (nuantă/hue) de intensitate (value) și saturatie. Acest lucru ofera următoarele avantaje:

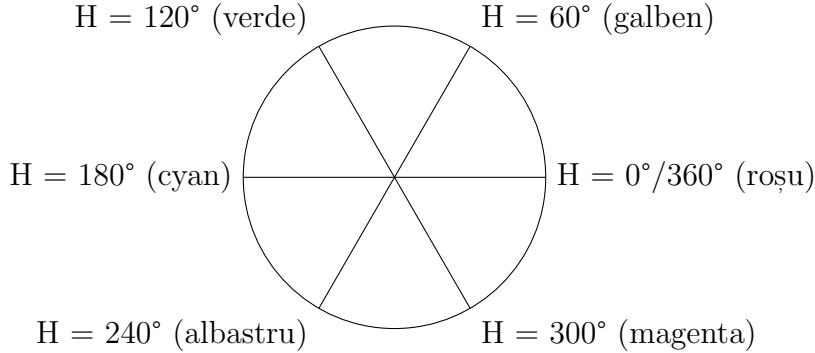


Figura 1: Reprezentarea circulară a componentei H (Hue) din spațiul HSV

- **Rezistență la variații de iluminare:** În HSV, componenta V (Value) captează informația despre luminozitate, permitând segmentarea bazată doar pe H și S.
- **Intuitivitate:** Nuanța (H) este reprezentată circular (0-360°), unde roșul apare atât la început (0-10°) cât și la sfârșit (160-180°). Această proprietate facilitează detectarea eficientă a culorii roșii.
- **Segmentare precisă:** Prin definirea unui interval pentru H și a unor praguri minime pentru S și V, putem izola eficient regiunile roșii, chiar și în condiții variabile de iluminare.

În spațiul HSV implementat în OpenCV, componenta H este scalată de la 0° la 180° (nu 360°), iar intervalele pentru culoarea roșie sunt aproximativ 0-10 și 160-180.

## 2.2 Filtrarea Gaussiană

Filtrul Gaussian este aplicat ca etapă de pre-procesare pentru:

- **Reducerea zgomotului:** Elimină variațiile de intensitate de înaltă frecvență care pot afecta segmentarea.
- **Estomparea detaliilor fine:** Uniformizează texturi și modele care ar putea fragmenta segmentarea.
- **Îmbunătățirea coeranței regiunilor:** Contribuie la formarea unor regiuni mai omogene în urma segmentării.

Filtrul Gaussian utilizează o distribuție normală bi-dimensională pentru a calcula ponderile kernel-ului:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (1)$$

unde  $\sigma$  reprezintă deviația standard a distribuției, care în implementarea noastră are valoarea 2.

## 2.3 Operații morfologice

După segmentarea culorii roșii, aplicăm două operații morfologice esențiale:

1. **Operația de închidere (MORPH\_CLOSE)**: Este o dilatare urmată de o eroziune, care:
  - Elimină găurile mici din interiorul regiunilor segmentate
  - Conectează regiuni fragmentate care sunt apropiate
  - Netezează conturul fără modificarea semnificativă a dimensiunii regiunii
2. **Operația de deschidere (MORPH\_OPEN)**: Este o eroziune urmată de o dilatare, care:
  - Elimină punctele izolate și proeminentele subțiri
  - Separa regiuni conectate printr-o punte îngustă
  - Reduce zgomotul păstrând forma și dimensiunea elementelor semnificative

Utilizăm elemente structurante eliptice de dimensiuni diferite ( $5 \times 5$  pentru închidere și  $3 \times 3$  pentru deschidere) pentru a adapta efectul operațiilor la scala imaginii și la dimensiunea așteptată a semnelor.

## 2.4 Aproximarea poligonală a contururilor

Algoritmul **approxPolyDP** simplifică contururile complexe în poligoane cu mai puține vârfuri, păstrând în același timp forma esențială. Metoda implementează algoritmul Douglas-Peucker, care funcționează astfel:

1. Unește primele și ultimele puncte ale conturului cu o linie dreaptă
2. Identifică punctul cel mai îndepărtat de această linie
3. Dacă distanța punctului depășește un prag specificat (în cazul nostru, 2% din perimetrul conturului), păstrează acel punct
4. Procesul se repetă recursiv pentru segmentele de linie rezultate

Această aproximare poligonală este crucială pentru a determina numărul de vârfuri ai semnelor (3 pentru triunghi, 8 pentru octogon) și pentru a simplifica analiza geometrică ulterioară.

## 2.5 Caracteristici geometrice pentru clasificare

Algoritmul extrage și utilizează mai multe caracteristici geometrice pentru a diferenția tipurile de semne:

### 2.5.1 Circularitatea

Circularitatea măsoară cât de aproape este forma unui contur de un cerc perfect și este calculată conform formulei:

$$\text{Circularitate} = \frac{4\pi \cdot \text{Arie}}{\text{Perimetru}^2} \quad (2)$$

Un cerc perfect are o circularitate de 1.0, iar orice altă formă are o valoare mai mică. În implementarea noastră, utilizăm un prag de 0.80 pentru a identifica semnele circulare (SPEED și NO ENTRY).

### 2.5.2 Raportul de umplere cu roșu

Această caracteristică reprezintă proporția de pixeli roșii din interiorul conturului și este esențială pentru diferențierea între semne cu forme similare dar distribuții diferite ale culorii:

$$\text{RedFillRatio} = \frac{\text{Număr pixeli roșii}}{\text{Număr total pixeli în contur}} \quad (3)$$

Valorile caracteristice pentru fiecare tip de semn sunt:

- STOP:  $> 0.85$  (predominant roșu)
- NO ENTRY: valoare moderată (0.35-0.85)
- SPEED:  $< 0.35$  (predominant alb cu bordură roșie)

### 2.5.3 Centroidul și orientarea

Pentru semnul YIELD (triunghi), determinăm orientarea prin compararea poziției vârfului superior cu cea a centroidului. Un triunghi cu vârful în jos va avea vârful superior deasupra centroidului.

Centroidul ( $C_x, C_y$ ) este calculat folosind momentele geometrice:

$$C_x = \frac{M_{10}}{M_{00}}, \quad C_y = \frac{M_{01}}{M_{00}} \quad (4)$$

unde  $M_{pq}$  reprezintă momentul de ordinul  $(p + q)$ :

$$M_{pq} = \sum_{x,y \in \text{Region}} x^p y^q \quad (5)$$

## 3 Descrierea detaliată a soluției

### 3.1 Pre-procesarea imaginii

Etapa de pre-procesare pregătește imaginea pentru segmentarea robustă a culorii roșii:

1. **Încărcarea imaginii:** Imaginea sursă este citită din fișier și verificată pentru validitate.

- Aplicarea filtrului Gaussian:** Un kernel Gaussian de dimensiune  $5 \times 5$  cu  $\sigma = 2$  este aplicat pentru reducerea zgomotului și estomparea detaliilor fine.
- Conversia în spațiul HSV:** Imaginea este transformată din RGB în HSV pentru a facilita segmentarea bazată pe nuanță.

### 3.2 Segmentarea culorii roșii

Segmentarea identifică regiunile roșii care pot conține semne de circulație:

- Definirea intervalelor de culoare:** Sunt create două măști binare pentru a capta întregul spectru de roșu:
  - Prima mască: H: 0-10, S: 60-255, V: 80-255 (roșu portocaliu)
  - A doua mască: H: 160-180, S: 60-255, V: 80-255 (roșu purpuriu)
- Combinarea măștilor:** Cele două măști sunt combinate printr-o operație logică OR pentru a obține masca roșie completă.
- Îmbunătățirea măștii:** Se aplică operațiile morfologice pentru curățarea și consolidarea regiunilor:
  - Operația de închidere cu un element structurant eliptic  $5 \times 5$  umple goulurile mici
  - Operația de deschidere cu un element structurant eliptic  $3 \times 3$  elimină zgomotul și artefactele mici

### 3.3 Detectia și procesarea contururilor

Această etapă identifică formele potențiale ale semnelor din masca binară:

- Extragerea contururilor:** Funcția `findContours` identifică toate contururile exterioare din masca binară, folosind metoda `CHAIN_APPROX_SIMPLE` pentru stocarea eficientă.
- Filtrarea pe baza ariei:** Contururile cu arie mai mică de 500 de pixeli sunt eliminate pentru a reduce false pozitive.
- Calculul perimetrelui:** Pentru fiecare contur valid, se determină perimetrul folosind `arcLength`.
- Aproximarea poligonală:** Conturul este simplificat în formă poligonală utilizând algoritmul Douglas-Peucker, cu un prag de 2% din perimetru.
- Calculul momentelor și centroidului:** Momentele geometrice sunt calculate pentru a determina centrul de masă al fiecărei forme detectate.
- Extragerea caracteristicilor:** Pentru fiecare contur, se calculează:
  - Circularitatea:  $\frac{4\pi \cdot \text{Arie}}{\text{Perimetru}^2}$
  - Raportul de umplere cu roșu: proporția de pixeli roșii din interiorul conturului
  - Numărul de vârfuri ale poligonului: `approx.size()`
  - Convexitatea: verificată cu `isContourConvex`

### 3.4 Algoritmul de clasificare a semnelor

Clasificarea utilizează un sistem de reguli bazat pe caracteristicile extrase:

---

**Algorithm 1** Clasificarea semnelor de circulație

---

```
1: for fiecare contur  $c$  din lista de contururi do
2:   Calculează aria  $\leftarrow \text{contourArea}(c)$ 
3:   if aria < 500 then
4:     continuă cu următorul contur
5:   end if
6:   Calculează perimetru  $\leftarrow \text{arcLength}(c, \text{true})$ 
7:   Obține poligonul aproximativ  $\text{approx}$  folosind  $\text{approxPolyDP}$ 
8:   Calculează centroidul  $\text{cent}$  folosind  $\text{moments}$ 
9:   Calculează circularitatea  $\leftarrow 4\pi \cdot \text{area}/\text{peri}^2$ 
10:  Calculează redRatio  $\leftarrow \text{redFillRatio}(\text{approx}, \text{redMask}, \text{size})$ 
11:  if  $\text{approx}$  are 3 vârfuri  $\text{isTrianglePointingDown}(\text{approx})$  then
12:    Clasifică ca "YIELD"
13:  else if  $\text{approx}$  are 8 vârfuri  $\text{isContourConvex}(\text{approx})$   $\text{redRatio} > 0.85$  then
14:    Clasifică ca "STOP"
15:  else if circularitatea > 0.80 then
16:    if redRatio < 0.35 then
17:      Clasifică ca "SPEED"
18:    else
19:      Clasifică ca "NO ENTRY"
20:    end if
21:  end if
22:  if clasificare reușită then
23:    Desenează conturul și adaugă eticheta centrată pe centroid
24:  end if
25: end for
```

---

Regulile de clasificare sunt optimizate pentru a diferenția cele patru tipuri de semne vizate:

1. **YIELD (Cedează trecerea):**

- Număr de vârfuri: exact 3 (triunghi)
- Orientare: vârful cel mai de sus este deasupra centroidului (triunghi cu vârful în jos)

2. **STOP:**

- Număr de vârfuri: exact 8 (octogon)
- Convexitate: contur convex
- Raport de umplere cu roșu:  $> 0.85$  (predominant roșu)

3. **Semne circulare (circularitate > 0.80):**

- **SPEED (Limită de viteză):** raport de umplere cu roșu  $< 0.35$  (predominant alb)

- **NO ENTRY (Acces interzis):** raport de umplere cu roșu  $\geq 0.35$  (cantitate semnificativă de roșu)

### 3.5 Funcția isTrianglePointingDown

Această funcție determină orientarea unui triunghi prin compararea poziției vârfului său superior cu cea a centroidului:

- Calculul momentelor și centroidului:** Se determină centrul de masă al triunghiului.
- Identificarea vârfului superior:** Se parcurg cele trei puncte pentru a găsi punctul cu coordonata  $y$  minimă (cel mai sus pe imagine).
- Compararea pozițiilor:** Se verifică dacă vârful superior are coordonata  $y$  mai mică decât coordonata  $y$  a centroidului.
- Interpretarea rezultatului:** Dacă vârful superior este deasupra centroidului, triunghiul are vârful în jos (semnul YIELD).

Această verificare este esențială pentru a evita confuzia cu alte semne triunghiulare care au orientări diferite.

### 3.6 Funcția redFillRatio

Această funcție determină proporția de pixeli roșii din interiorul unui contur, esențială pentru diferențierea între semnele cu forme similare:

- Crearea măștii poligonului:** Se creează o mască binară în care poligonul este umplut cu valoarea 255 (alb).
- Intersecția cu masca roșie:** Se realizează operația AND logic între masca poligonului și masca roșie.
- Numărarea pixelilor:** Se numără pixelii nenuli atât în intersecție cât și în masca poligonului.
- Calculul raportului:** Se împarte numărul de pixeli roșii din interior la numărul total de pixeli din poligon.

Aceste rapoarte de umplere cu roșu permit diferențierea precisă între semnele de limitare de viteză (predominant albe) și cele de acces interzis (cu o proporție mai mare de roșu).

### 3.7 Vizualizarea rezultatelor

În etapa finală, semnele detectate și clasificate sunt marcate pe imaginea originală:

- Desenarea contururilor:** Contururile poligonale ale semnelor sunt trasate cu culori specifice (verde pentru STOP și YIELD, roșu pentru SPEED, albastru pentru NO ENTRY).

2. **Adăugarea etichetelor:** Etichetele text sunt adăugate centrat pe centroidul fiecărui semn detectat.
3. **Afișarea rezultatului:** Imaginea finală cu semnele detectate este afișată într-o fereastră.

## 4 Analiza soluției și perspective de îmbunătățire

### 4.1 Avantaje ale metodei implementate

- **Eficiență computațională:** Utilizarea segmentării bazate pe culoare și a regulilor geometrice simple permit procesarea rapidă, potențial aplicabilă în sisteme în timp real.
- **Robustete la variații de scală:** Caracteristicile geometrice folosite (circularitate, raport de umplere, număr de vârfuri) sunt invariante la scală, permitând detectarea semnelor de dimensiuni diferite.
- **Simplitate conceptuală:** Abordarea bazată pe reguli este ușor de înțeles și de modificat, fără a necesita seturi de date de antrenare extinse.
- **Extensibilitate:** Sistemul poate fi extins pentru a include alte semne prin adăugarea de reguli și caracteristici suplimentare.

### 4.2 Limitări și provocări

- **Sensibilitate la condiții de iluminare:** Segmentarea bazată pe culoare poate fi afectată de umbră, reflexii sau condiții de iluminare extremă.
- **Dependență de culoare:** Implementarea actuală se bazează puternic pe detecția culorii roșii, limitând aplicabilitatea la semne care conțin roșu.
- **Vulnerabilitate la ocluzie:** Semnele parțial acoperite pot avea contururi modificate, afectând extragerea caracteristicilor și clasificarea.
- **Geometrie distorsionată:** Unghiurile de vizualizare oblice pot distorsiona forma percepției a semnelor, afectând numărul de vârfuri și circularitatea.
- **Absența recunoașterii textului:** Sistemul nu poate diferenția între diverse semne de limită de viteză (30, 50, 70 km/h etc.), tratându-le pe toate ca "SPEED".

### 4.3 Direcții de îmbunătățire

- **Implementarea detecției multi-culoare:** Extinderea segmentării pentru a include alte culori (albastru, galben, alb) ar permite detectarea unei game mai largi de semne.
- **Integrarea OCR (Optical Character Recognition):** Adăugarea recunoașterii textului pentru a identifica limitele specifice de viteză și alte informații textuale.
- **Corectarea perspectivei:** Implementarea transformărilor de perspectivă pentru a normaliza formele semnelor văzute din unghiuri oblice.

- **Utilizarea metodelor bazate pe învățare automată:** Integrarea clasificatoarelor machine learning (SVM, CNN) ar putea îmbunătăți precizia clasificării și robustețea la variații.
- **Filtrare temporală:** Pentru aplicații video, implementarea filtrării temporale pentru a reduce detecțiile false și a îmbunătăți consistența clasificării între cadre.
- **Estimarea distanței:** Adăugarea estimării distanței până la semnele detectate, folosind informații despre dimensiunea aparentă a semnelor.

## 5 Exemple de imagini pentru semnele detectate

Secțiunea prezintă exemple vizuale pentru fiecare tip de semn, utilizate pentru testarea și validarea algoritmului.



Figura 2: STOP - semn octogonal roșu cu text alb



Figura 3: YIELD - triunghi roșu cu bordură albă, vârf în jos

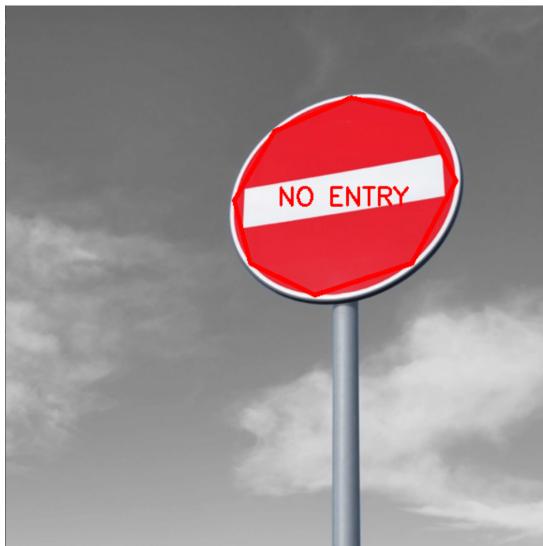


Figura 4: NO ENTRY - cerc roșu cu bandă orizontală albă



Figura 5: SPEED - cerc alb cu bordură roșie și număr în centru

## **6 Concluzii**

Sistemul implementat demonstrează eficacitatea metodelor clasice de procesare a imaginilor în detecția și clasificarea semnelor de circulație. Prin combinarea segmentării bazate pe culoare, a analizei geometrice și a regulilor de clasificare, algoritmul poate identifica cu succes patru tipuri comune de semne rutiere: STOP, YIELD, NO ENTRY și SPEED.

Soluția oferă un echilibru bun între complexitate și performanță, fiind potrivită pentru aplicații educaționale și pentru sisteme simple de asistență la conducere. Cu îmbunătățirile sugerate, în special integrarea tehniciilor de învățare automată și adăugarea recunoașterii textului, sistemul ar putea fi extins pentru a acoperi o gamă mai largă de semne și pentru a funcționa în condiții de mediu mai variate.