În realizarea aplicației de recunoaștere a semnelor de circulație, am urmat următorii pași pentru a analiza imaginea și a detecta semnele după culoare (roșu/albastru) și formă (triunghi, pătrat, cerc, octogon). Mai jos este detaliat fiecare pas pe care l-am urmat în implementare:

**1. Încărcarea imaginii și preprocesare**

Aplicația începe prin citirea unei imagini color în format BGR folosind funcția imread. Pentru a reduce zgomotul vizual care ar putea afecta analiza ulterioară, se aplică un filtru Gauss folosind funcția GaussianBlur. Această funcție estompează variațiile mici de culoare și netezește imaginea, prevenind apariția marginilor false în procesul de detecție.

Apoi, imaginea este convertită din spațiul de culoare BGR în HSV (Hue – Saturație – Valoare) cu ajutorul funcției bgrToHsv. Conversia este esențială deoarece în spațiul HSV, detecția culorii se face mai robust, fiind independentă de luminozitate. În HSV, nuanța este separată de saturație și luminozitate, facilitând aplicarea filtrelor de culoare.

**2. Crearea și rafinarea măștilor de culoare**

Se definesc intervale HSV pentru culorile de interes:

* Pentru roșu: Am definit două intervale HSV deoarece semnele de circulație roșii pot apărea în imagine atât sub forma unui roșu deschis, cât și ca o nuanță intensă, închisă. Observând că aceste două extreme se află la capetele scalei de tonuri în modelul HSV (aproape de 0 și de 180), am constatat că includerea întregului interval într-o singură limită ducea la apariția multor regiuni false în mască, cauzate de obiecte roșiatice nesemnificative din fundal. Prin urmare, am separat detectarea roșului în două intervale distincte și am combinat rezultatele
* Pentru albastru: un singur interval am constat ca este suficient.

Funcția inRange transformă imaginea HSV într-o mască binară: pixelii care se încadrează în intervalul specificat sunt setați la 255 (alb), restul la 0 (negru). Obținem astfel două măști binare ,una pentru roșu, alta pentru albastru.

Aceste măști sunt apoi prelucrate pentru a elimina zgomotul și pentru a închide eventualele găuri din interiorul formelor folosind următoarele operații morfologice:

* opening (eroziune urmată de dilatare) – elimină petele mici sau zgomotul.
* closing (dilatare urmată de eroziune) – închide găurile din interiorul obiectelor.
* dilation – extinde marginile formelor pentru a consolida structura.

**3. Umplerea găurilor din interiorul formelor**

După prelucrarea cu morfologie, unele semne (ex. cercuri) pot rămâne cu interiorul gol. Pentru a le umple:

* Se aplică floodFill de la un colț exterior (de regulă, colțul (0,0)), umplând zona de fundal cu alb.
* Se aplică bitwise\_not pentru a inversa imaginea umplută.
* Se combină cu masca originală prin operatorul | pentru a completa zonele neînchise.

**4. Detecția contururilor**

Cu ajutorul funcției findContours, se extrag contururile din măștile binare. Fiecare contur este o listă de puncte ce definesc o formă în imagine. Se filtrează:

* contururile cu arie prea mică (eliminate cu contourAreaNaive),
* contururile care se suprapun cu altă culoare (verificate prin overlapsWithOtherColor).

**5. Determinarea formei geometrice**

Pentru fiecare contur, se determină forma folosind funcția approxPolyDP, care aproximează curba conturului cu un poligon cu mai puține vârfuri, în funcție de precizia specificată.

Se aplică apoi funcțiile:

* esteTriunghi – verifică dacă forma are 3 vârfuri sau 4 cu aspect îngust din cauza ca nu este perfect.
* estePatrat – verifică dacă forma are 4 colțuri și aspectul (raportul lățime/înălțime) este aproape 1.
* esteCerc – verifică dacă forma are multe colțuri și circularitate mare. Circularitatea este calculată cu formula:Circularitate= 4 \* PI \* arie / (perimetru \* perimetru).
* esteOctogon – verifică dacă există între 7 și 9 colțuri și raportul lățime/înălțime este echilibrat.

**6. Verificarea culorii în interiorul formei**

Funcția areCuloare se asigură că interiorul formei are cu adevărat culoarea specificată. Ea face următoarele:

1. Creează o mască care acoperă doar zona conturului.
2. Aplică din nou inRange doar pe acea zonă.
3. Calculează raportul dintre pixelii ce corespund culorii dorite și numărul total de pixeli din mască.
4. Dacă proporția este suficient de mare (ex. peste 20%), culoarea este considerată validă.

**7. Afișarea rezultatelor**

Dacă toate condițiile (formă, culoare, dimensiune, neinterferență cu alte culori) sunt îndeplinite:

* se trasează un dreptunghi în jurul semnului cu rectangle,
* se adaugă un text descriptiv cu putText care precizează tipul semnului: „STOP”, „CERC ROSU”, „TRIUNGHI ROSU”, etc.

La final, se afișează:

* imaginea originală,
* imaginea cu semnele detectate și notate.

Codul se găsește pe GitHub la: <https://github.com/SiposBogdan/Detectare-semne-de-ciculatie> .