



Universitatea  
Transilvania  
din Brașov  
**FACULTATEA DE MATEMATICĂ  
ȘI INFORMATICĂ**

**Programul de studiu:**  
Informatică

## **Lucrare de licență**

**Sistem de gestionare al accesului într-o parcare auto  
folosind tehnici de procesare a imaginilor**

**Autor: George Daniel Patrașc  
Coordonator științific: Lect. Dr. Honorius Cezar Gâlmeanu**

**Brașov, 2024**

# Cuprins

<b>1 Introducere</b>	<b>3</b>
1.1 Scopul aplicației . . . . .	3
1.2 Motivarea aplicației . . . . .	4
<b>2 Noțiuni teoretice</b>	<b>5</b>
2.1 Ce este imaginea digitală și procesarea de imagine? . . . . .	5
2.2 Spații de culoare . . . . .	5
2.3 Histograma . . . . .	8
2.4 Decupare și redimensionare . . . . .	9
2.5 Padding . . . . .	10
2.6 Filtre de Binarizare . . . . .	11
2.7 Filtre trece-jos . . . . .	12
2.8 Filtre trece-sus . . . . .	13
2.9 Componete conexe . . . . .	15
2.10 Contururi . . . . .	16
2.11 Analiza componentelor conexe . . . . .	17
2.12 Operații logice . . . . .	18
2.13 Operații morfologice . . . . .	19
2.14 Transformări geometrice . . . . .	21
2.15 Transformata Hough . . . . .	22
2.16 Distanța de la un punct la o dreaptă . . . . .	25
2.17 Punctul de intersecție a două drepte . . . . .	25
2.18 Distanța dintre două drepte . . . . .	26
2.19 Determinarea unei drepte care trece printr-un punct . . . . .	27
<b>3 Tehnologii folosite</b>	<b>28</b>
3.1 Limbajul de programare C++ . . . . .	28
3.2 Programarea orientată pe obiect . . . . .	30
3.3 Dynamic-link library . . . . .	31

3.4	Testarea unitară . . . . .	32
3.5	Visual Studio . . . . .	33
3.6	CMake . . . . .	35
3.7	Dxygen . . . . .	36
3.8	GitLab . . . . .	37
3.9	OpenCV . . . . .	38
3.10	Tesseract OCR . . . . .	39
3.11	Qt . . . . .	41
<b>4</b>	<b>Prezentarea aplicației</b>	<b>43</b>
4.1	Arhitectura . . . . .	43
4.2	Segmentarea, rectificarea și recunoașterea textului . . . . .	45
4.3	Interfața grafică . . . . .	65
4.4	Generarea și configurarea proiectului . . . . .	69
<b>5</b>	<b>Concluzii și rezultate</b>	<b>74</b>
5.1	Direcții și dezvoltări viitoare . . . . .	74
5.2	Concluzii . . . . .	74
<b>6</b>	<b>Bibliografie</b>	<b>76</b>
<b>7</b>	<b>Anexă</b>	<b>77</b>

# Capitolul 1

## Introducere

### 1.1 Scopul aplicației

Aplicația are scopul de a facilita și automatiza gestionarea accesului în parcările auto, prin identificarea și înregistrarea automată a numerelor de înmatriculare ale autovehiculelor, aducând beneficii eficienței operaționale și securității.



- I. **Detectarea și identificarea automată a numărului de înmatriculare:** Aplicația utilizează camere video poziționate strategic pentru a capta imagini ale vehiculelor la momentul intrării sau ieșirii acestora din parcare. Aceste imagini sunt analizate în timp real pentru a se detecta și recunoaște numerele de înmatriculare.
- II. **Prelucrarea imaginilor:** Folosind algoritmi de procesare a imaginilor și tehnici de segmentare, aplicația extrage și rectifică numărul de înmatriculare din imaginea captată, asigurându-se că textul este clar în imagine.
- III. **Recunoașterea textului prin tehnologii de inteligență artificială:** Odată ce numărul de înmatriculare este izolat și optimizat, modelul AI Tesseract de recunoaștere optică a caracterelor (OCR) interpretează textul. Acest model este antrenat să recunoască în special fonturile și stilurile numerelor de înmatriculare.
- IV. **Înregistrarea și monitorizarea accesului:** Numerele de înmatriculare recunoscute sunt înregistrate într-o bază de date. Aceasta permite urmărirea vehiculelor

care intră și ies din parcare, oferind date importante pentru securitate și gestionarea spațiului de parcare.

- V. **Interfață grafică de utilizator (GUI):** Aplicația include o interfață grafică prietenoasă și intuitivă, care afișează în timp real informațiile despre accesul în parcare. Interfața permite utilizatorilor să vizualizeze imagini recente și să acceseze istoricul vehiculelor care au pătruns în parcare.

## 1.2 Motivarea aplicației

În cadrul proiectului, s-a urmărit dezvoltarea unei soluții eficiente prin valorificarea cunoștințelor existente din domeniul procesării imaginilor. Totodata, proiectul a implementat aplicarea și extinderea tehniciilor de analiză și procesare vizuală, pentru a identifica și extrage numerele de înmatriculare din imaginile captate de la intrările în parcările auto.

Ca limbaj de programare s-a optat pentru C++, care este larg recunoscut în domeniul procesării de imagini datorită eficienței sale remarcabile și a flexibilității pe care o oferă în gestionarea resurselor de sistem, contribuind la optimizarea timpului de execuție și permitând manipularea avansată și rapidă a datelor vizuale.

Pe parcursul dezvoltării soluției, s-a pus accent pe integrarea unei suite de tehnologii și instrumente de dezvoltare pentru a optimiza atât procesul de elaborare, cât și performanța finală a sistemului. În acest sens, s-a utilizat biblioteca OpenCV, care a fost esențială în lucrul cu imagini și utilizarea algoritmilor de procesare. De asemenea, s-a folosit modelul de inteligență artificială Tesseract OCR, recunoscut pentru eficiența sa în recunoașterea optică a caracterelor, pentru a extrage textul clar și precis din imagini.

Interfața grafică a fost creată cu ajutorul framework-ului Qt, oferind o manieră eficientă de interacțiune și o vizualizare intuitivă a datelor procesate. Pentru managementul proiectului și automatizarea construcției sistemului, s-au folosit CMake și Doxygen, acesta din urmă facilitând generarea documentației tehnice detaliate. De asemenea, pentru a asigura robustețea codului, s-au implementat unit teste, care au permis verificarea și validarea funcționalităților în diverse scenarii de utilizare.

Întregul proces de dezvoltare a fost suportat de mediul integrat de dezvoltare Visual Studio, și a fost gestionat folosind GitLab pentru versionarea codului.

# **Capitolul 2**

## **Notiuni teoretice**

### **2.1 Ce este imaginea digitală și procesarea de imagine?**

O imagine poate fi definită ca o funcție bidimensională,  $f(x, y)$ , unde  $x$  și  $y$  sunt coordinate spațiale, iar amplitudinea lui  $f$  la orice pereche de coordonate  $(x, y)$  este numită intensitatea sau nivelul de gri al imaginii în acel punct. Când  $x$ ,  $y$  și valorile intensității lui  $f$  sunt finite și discrete, imaginea se numește imagine digitală. Procesarea imaginii digitale se referă la procesarea imaginilor digitale cu ajutorul unui computer digital. O imagine digitală este compusă dintr-un număr finit de elemente, fiecare având o locație și o valoare specifică, numite pixeli.

Viziunea este cel mai avansat simț al nostru, astfel încât imaginile joacă un rol esențial în percepția umană. Spre deosebire de oameni, care sunt limitați la spectrul vizual al radiațiilor electromagnetice, mașinile de imagistică acoperă aproape întregul spectru electromagnetic, de la raze gamma la unde radio. Acestea pot opera pe imagini generate de surse neobișnuite pentru oameni, cum ar fi ultrasunetele, microscopia electronică și imagini generate de computer.

Nu există un acord general privind granițele dintre procesarea imaginii, analiza imaginii și viziunea computerizată. Uneori, procesarea imaginii este definită ca o disciplină în care atât intrarea, cât și ieșirea sunt imagini. Aceasta poate fi o definiție limitativă. Pe de altă parte, viziunea computerizată are ca scop imitarea vederii umane, inclusiv învățarea și luarea deciziilor pe baza inputurilor vizuale, fiind o ramură a inteligenței artificiale.

Un exemplu de procesare a imaginii digitale este analiza automată a textului, care include achiziția imaginii, preprocesarea, extragerea și recunoașterea caracterelor individuale. Înțelegerea conținutului paginii poate fi considerată analiză a imaginii sau chiar viziune computerizată, în funcție de complexitate.

### **2.2 Spații de culoare**

În procesarea imaginilor, spațiile de culoare sunt modalități de a reprezenta culorile în modele numerice care facilitează manipularea și analiza imaginilor. Cele mai comune spații de culoare utilizate sunt Grayscale, RGB și HSV.

Grayscale (tonuri de gri) este un spațiu de culoare în care fiecare pixel este reprezen-

tat de o singură valoare care indică intensitatea luminii. Valorile sunt de obicei cuprinse între 0 (negru) și 255 (alb) pentru imagini pe 8 biți. Conversia unei imagini color într-o imagine grayscale se face prin combinarea componentelor de culoare ale fiecărui pixel într-o singură valoare de luminanță.

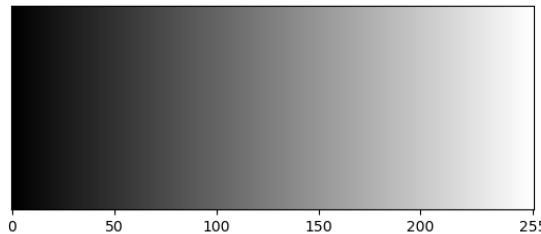


Figura 2.1: Gradientul Grayscale

RGB (Red, Green, Blue) este un spațiu de culoare aditiv în care orice culoare poate fi creată prin combinarea celor trei culori primare: roșu, verde și albastru. Fiecare componentă are o valoare între 0 și 255 într-o imagine pe 8 biți.

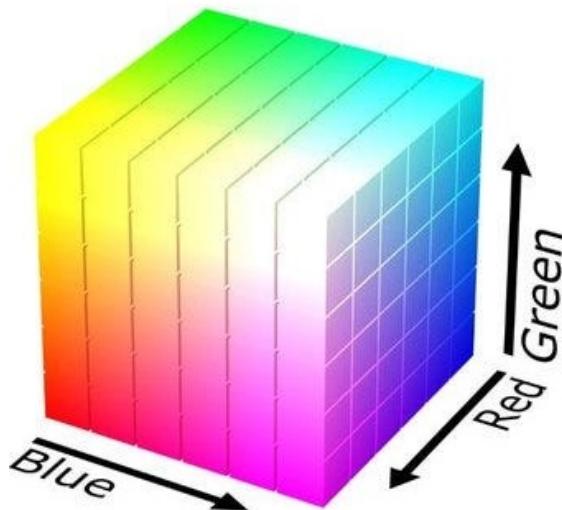


Figura 2.2: Cubul RGB

HSV (Hue, Saturation, Value) este un spațiu de culoare care reprezintă culoarea într-un mod mai intuitiv pentru perceptia umană. Cele trei componente sunt:

- **Hue (nuantă):** descrie tipul de culoare și este măsurat în grade de la 0 la 360.
- **Saturation (saturatie):** descrie intensitatea culorii și variază de la 0 (gri) la 1 (culoare pură).
- **Value (valoare):** descrie luminozitatea culorii și variază de la 0 (negru) la 1 (alb complet).

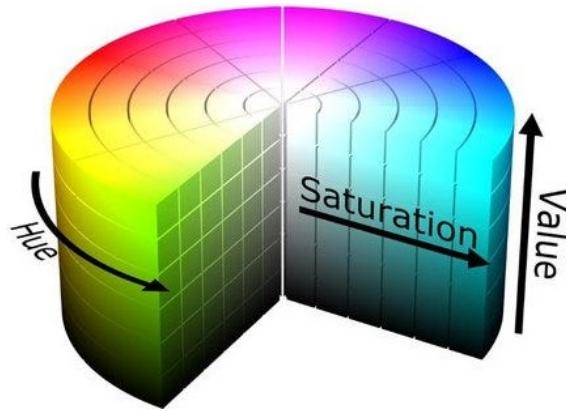


Figura 2.3: Cilindrul HSV

Conversia între diferite spații de culoare este adesea necesară în procesarea imaginilor pentru a facilita anumite operații. Aceste conversii permit adaptarea și prelucrarea imaginilor în funcție de cerințele specifice ale aplicațiilor, fie că este vorba de îmbunătățirea contrastului, identificarea obiectelor sau alte tehnici de analiză. Iată câteva formule de bază pentru conversie:

#### 1. Conversia RGB la Grayscale:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Această formulă folosește o combinație ponderată a componentelor roșu, verde și albastru pentru a calcula luminanța, rezultând o imagine în tonuri de gri care păstrează percepția de lumină a imaginii originale.

#### 2. Conversia RGB la HSV:

$$C_{\max} = \max(R, G, B)$$

$$C_{\min} = \min(R, G, B)$$

$$\Delta = C_{\max} - C_{\min}$$

$$H = \begin{cases} 0 & \text{dacă } \Delta = 0 \\ 60^\circ \times \frac{G-B}{\Delta} + 360^\circ \mod 360^\circ & \text{dacă } C_{\max} = R \\ 60^\circ \times \frac{B-R}{\Delta} + 120^\circ & \text{dacă } C_{\max} = G \\ 60^\circ \times \frac{R-G}{\Delta} + 240^\circ & \text{dacă } C_{\max} = B \end{cases}$$

$$S = \begin{cases} 0 & \text{dacă } C_{\max} = 0 \\ \frac{\Delta}{C_{\max}} & \text{altfel} \end{cases}$$

$$V = C_{\max}$$

Conversia RGB la HSV este utilă deoarece spațiul HSV este mai intuitiv pentru percepția umană a culorilor, facilitând operații cum ar fi ajustarea saturației sau a nuanței fără a afecta celelalte componente de culoare.

### 3. Conversia HSV la RGB:

$$C = V \times S$$

$$X = C \times \left(1 - \left|\left(\frac{H}{60^\circ} \bmod 2\right) - 1\right|\right)$$

$$m = V - C$$

$$(R, G, B) = \begin{cases} (C + m, X + m, m) & \text{dacă } 0^\circ \leq H < 60^\circ \\ (X + m, C + m, m) & \text{dacă } 60^\circ \leq H < 120^\circ \\ (m, C + m, X + m) & \text{dacă } 120^\circ \leq H < 180^\circ \\ (m, X + m, C + m) & \text{dacă } 180^\circ \leq H < 240^\circ \\ (X + m, m, C + m) & \text{dacă } 240^\circ \leq H < 300^\circ \\ (C + m, m, X + m) & \text{dacă } 300^\circ \leq H < 360^\circ \end{cases}$$

Această conversie permite transformarea din spațiul de culoare HSV în RGB, ceea ce este util pentru afișarea imaginilor după ce au fost manipulate în spațiul HSV. Conversiile între aceste spații de culoare permit efectuarea de prelucrări complexe pe imagini, cum ar fi filtrarea bazată pe nuanță sau saturatie, care sunt greu de realizat direct în spațiul RGB.

## 2.3 Histograma

Să notăm cu  $r_k$ , pentru  $k = 0, 1, 2, \dots, L - 1$ , intensitățile unei imagini digitale de nivel  $L$ ,  $f(x, y)$ . Histograma ne-normalizată a lui  $f$  este definită ca:

$$h(r_k) = n_k \quad \text{pentru } k = 0, 1, 2, \dots, L - 1$$

unde  $n_k$  este numărul de pixeli din  $f$  cu intensitatea  $r_k$ , iar subdiviziunile scalei de intensitate sunt numite *bin-uri de histogramă*. În mod similar, histograma normalizată a lui  $f$  este definită ca:

$$p(r_k) = \frac{h(r_k)}{MN} = \frac{n_k}{MN}$$

unde, ca de obicei,  $M$  și  $N$  sunt numărul de rânduri și coloane ale imaginii, respectiv. În general, lucrăm cu histograme normalizate, la care ne referim simplu ca histogramă sau histograme de imagine. Suma valorilor  $p(r_k)$  pentru toate valorile lui  $k$  este întotdeauna 1. Componentele  $p(r_k)$  sunt estimări ale probabilităților de apariție a nivelurilor de intensitate într-o imagine.

Forma histogramei este legată de apariția imaginii. De exemplu, presupunem imagini cu patru caracteristici de bază ale intensității: întunecată, luminoasă, contrast scăzut și contrast ridicat. Într-o imaginea întunecată, cele mai populate bin-uri ale histogramei sunt concentrate la capătul inferior (mai întunecat) al scalei de intensitate. Similar, majoritatea pixelilor luminoși sunt concentrati spre capătul superior al scalei de intensitate. O imagine cu contrast scăzut are o histogramă îngustă situată de obicei în jurul mijlocului scalei de intensitate. Pentru o imagine monocromă, aceasta implică un aspect gri, plăcăsitor.

Într-o imagine cu contrast ridicat, componentele histogramei acoperă un interval larg al scalei de intensitate, iar distribuția pixelilor nu este prea departe de uniformă,

doar câteva bin-uri fiind mult mai înalte decât altele. Intuitiv, este rezonabil să concluzionăm că o imagine ale cărei pixeli tind să ocupe întregul interval de niveluri de intensitate posibile și, în plus, tind să fie distribuți uniform, va avea un aspect de contrast ridicat și va prezenta o mare varietate de tonuri de gri. Efectul net va fi o imagine care arată o mare cantitate de detalii de nivel gri și are un interval dinamic ridicat.

## 2.4 Decupare și redimensionare

Decuparea (crop) este o operațiune esențială în prelucrarea imaginilor care presupune selecționarea unei subzone dintr-o imagine și eliminarea restului. Aceasta este utilă pentru focalizarea pe o regiune de interes specifică, eliminarea zonelor nedorite sau pregătirea imaginii pentru analize ulterioare. Procesul de decupare implică specificarea unui dreptunghi delimitat de coordonatele colțului din stânga sus ( $x, y$ ) și dimensiunile lățimii ( $w$ ) și înăltimii ( $h$ ).

Redimensionarea unei imagini implică schimbarea dimensiunilor acesteia pentru a se potrivi unor specificații date. Acest proces este esențial în diverse aplicații, cum ar fi pregătirea imaginilor pentru rețele neurale, ajustarea pentru afișare pe ecrane de diferite dimensiuni sau pregătirea pentru imprimare.

Pentru a redimensiona o imagine, trebuie să specificăm dimensiunea dorită a imaginii rezultate. Aceasta poate fi realizată în două moduri:

- **Redimensionare absolută:** Dimensiunea dorită a imaginii de ieșire este setată direct prin specificarea lățimii și înăltimii.
- **Redimensionare relativă:** Dimensiunea imaginii de ieșire este determinată prin factori de scalare aplicăți axelor  $x$  și  $y$ .

Un aspect important al redimensionării este metoda de interpolare utilizată pentru a calcula valorile pixelilor în imaginea redimensionată. Metodele comune de interpolare includ:

- **Interpolare cea mai apropiată:** Valoarea pixelului redimensionat este luată de la cel mai apropiat pixel din imaginea sursă.
- **Interpolare biliniară:** Valorile a patru pixeli din vecinătatea pixelului redimensionat sunt ponderate linear în funcție de distanța lor față de pixelul de destinație.
- **Interpolare bicubică:** Se utilizează o spline cubică pentru a interpola valorile dintr-o zonă  $4 \times 4$  din jurul pixelului redimensionat.
- **Interpolare Lanczos:** Utilizează o funcție sinc trunchiată pentru a interpola valoile dintr-o zonă  $8 \times 8$  din jurul pixelului redimensionat.

Metoda de interpolare influențează calitatea imaginii redimensionate, cu metode mai sofisticate oferind de obicei rezultate mai netede și mai precise, dar la un cost computațional mai mare.

## 2.5 Padding

Corelația produce un rezultat care este mai mic decât imaginea originală, ceea ce poate să nu fie de dorit în multe aplicații. Aceasta se datorează faptului că vecinătățile corelației tipice și operațiile de convoluție se extind dincolo de limitele imaginii în apropierea marginilor, astfel încât imaginile filtrate suferă de efecte de bordură (boundary effects).

Pentru a rezolva această problemă, au fost dezvoltate o serie de moduri de *padding* sau extindere pentru operațiile de vecinătate:

- **zero**: setează toți pixelii din afara imaginii sursă la 0 (o alegere bună pentru imagini cu margini alpha-matted);
- **constant (border color)**: setează toți pixelii din afara imaginii sursă la o valoare de bordură specificată;
- **clamp (replicate or clamp to edge)**: repetă pixelii de la margine la infinit;
- **cyclic (wrap, repeat, or tile)**: buclează în jurul imaginii într-o configurație toroidală;
- **mirror**: reflectă pixelii peste marginea imaginii;
- **extend**: extinde semnalul prin scăderea versiunii oglindite a semnalului de la valoarea pixelului de margine.

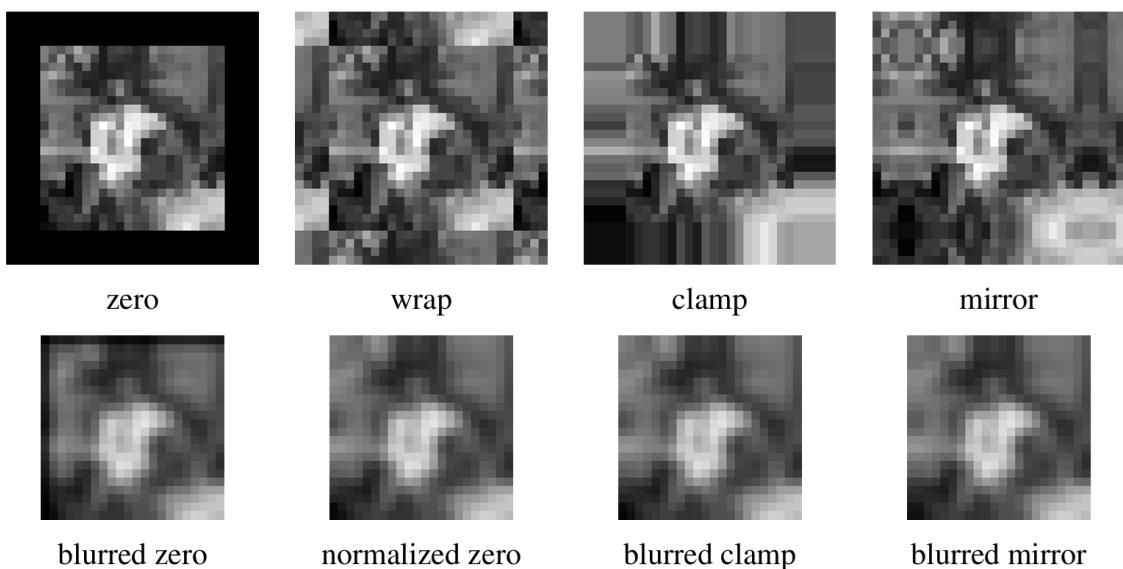


Figura 2.4: Diferite opțiuni de padding

Figura arată efectele *padding*-ului unei imagini cu fiecare dintre mecanismele de mai sus și apoi blurarea imaginii rezultante. După cum se poate vedea, *padding*-ul cu zero întunecă marginile, *padding*-ul cu *clamp* (replicare) propagă valorile pixelilor de la margini către interior, *padding*-ul cu *mirror* (reflecție) păstrează culorile aproape de margini. *Padding*-ul cu extindere (nu este prezentat) menține pixelii de la margini fixați (în timpul blur-ului).

O alternativă la *padding* este să înceșozi imaginea cu canale *zero-padded* și apoi să folosești valoarea alpha a imaginii pentru a elimina efectul de întunecare.

## 2.6 Filtre de Binarizare

În multe situații de prelucrare a imaginilor, este necesar să se ia o decizie finală asupra pixelilor dintr-o imagine sau să se eliminate categoric pixelii care se află sub sau peste o anumită valoare, păstrându-i pe ceilalți. Filtrul de binarizare îndeplinește aceste sarcini. Ideea de bază este că se ia un tablou de pixeli, împreună cu un prag, și apoi se aplică o operație fiecărui element al tabloului în funcție de faptul dacă valoarea sa este sub sau peste pragul respectiv. Dacă doriți, puteți considera binarizarea ca o operație de conoluție foarte simplă care utilizează un nucleu de dimensiuni  $1 \times 1$  și apoi efectuează una dintre mai multe operații neliniare pe acel pixel.



Figura 2.5: Imagine de Input

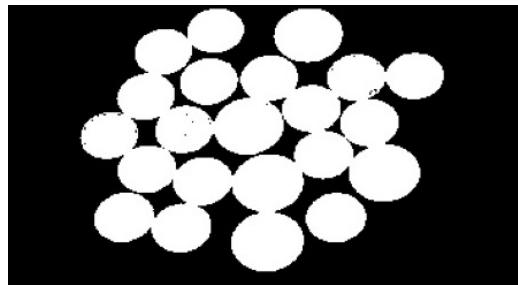


Figura 2.6: Filtrul Otsu

De asemenea, este posibil ca algoritmul de binarizare să determine valoarea optimă a pragului. Aceasta se poate realiza prin utilizarea unui algoritm precum Otsu. Pe scurt, algoritmul lui Otsu ia în considerare toate pragurile posibile și calculează varianța pentru fiecare dintre cele două clase de pixeli (adică clasa de sub prag și clasa de peste prag). Algoritmul lui Otsu minimizează următoarea expresie:

$$\sigma_w^2 \equiv w_1(t) \cdot \sigma_1^2 + w_2(t) \cdot \sigma_2^2$$

unde  $w_1(t)$  și  $w_2(t)$  sunt ponderile relative pentru cele două clase date de numărul de pixeli din fiecare clasă, iar  $\sigma_1^2$  și  $\sigma_2^2$  sunt varianțele din fiecare clasă. Se dovedește că minimizarea varianței celor două clase în acest mod este echivalentă cu maximizarea varianței între cele două clase. Deoarece este necesară o căutare exhaustivă a spațiului posibil de praguri, acest proces nu este unul foarte rapid, dar este foarte eficient în determinarea pragului optim pentru separarea pixelilor în două clase distințe.

Filtrul de binarizare prin metoda triunghi este o altă metodă eficientă pentru determinarea automată a pragului optim de binarizare a unei imagini. Această tehnică se bazează pe analiza histogramei imaginii și utilizarea unui algoritm geometric pentru a stabili punctul de prag. Procedura implică mai multe etape:

Prima etapă constă în calcularea histogramei imaginii. Fiecare intensitate este contorizată pentru a obține frecvența de apariție a fiecărei valori de pixel. După obținerea histogramei, se calculează histograma cumulativă. Aceasta ajută la identificarea distribuției totale a intensităților pixelilor și este folosită pentru a trasa linia de referință necesară în algoritmul de binarizare.

Pentru a determina linia de referință, se identifică punctele minim și maxim din histograma cumulativă. Linia de referință este trasată între aceste două puncte și servește ca bază pentru calcularea distanțelor. Pentru fiecare valoare din histograma cumulativă, se calculează distanța față de linia de referință. Această distanță este utilizată pentru a găsi punctul de pe histogramă care se află la cea mai mare distanță față de linia de referință.

Valoarea de intensitate care are cea mai mare distanță față de linia de referință este considerată pragul optim. Acest prag este folosit pentru a binariza imaginea inițială, separând pixelii în două clase: cei cu valori sub prag și cei cu valori peste prag.

Algoritmul de binarizare prin metoda triunghi oferă o metodă robustă și automată pentru binarizarea imaginilor, fiind util în diverse aplicații de prelucrare a imaginilor, unde separarea precisă a obiectelor de fundal este esențială.

## 2.7 Filtre trece-jos

Filtrele spațiale de netezire (denumite și de mediere) sunt utilizate pentru a reduce tranzițiile abrupte în intensitate. Deoarece zgomotul aleatoriu constă de obicei din tranziții abrupte în intensitate, o metodă evidentă a netezirii este reducerea zgomotului. Netezirea înainte de resampling-ul imaginii pentru a reduce aliasing-ul este, de asemenea, o practică comună.



Figura 2.7: Imagine de Input



Figura 2.8: Filtrul Gaussian

Netezirea este utilizată pentru a reduce detaliile nerelevante într-o imagine, unde „nerelevant” se referă la regiunile de pixeli care sunt mici în raport cu dimensiunea kernelului filtrului. O altă aplicație este netezirea contururilor false care rezultă din utilizarea unui număr insuficient de niveluri de intensitate într-o imagine. Filtrele de netezire sunt utilizate în combinație cu alte tehnici pentru îmbunătățirea imaginii, cum ar fi tehnici de procesare a histogramelor și mascare unsharp.

Cel mai simplu filtru trece-jos separabil este filtrul de tip cutie, ale căruia coeficienți au aceeași valoare (de obicei 1). Numele „kernel cutie” provine dintr-un kernel de dimensiuni  $m \times n$  cu toți coeficienții egali. Normalizarea kernelului se face prin împărțirea fiecărui coeficient la suma valorilor coeficienților (adică  $1/mn$  când toți coeficienții sunt 1).

Datorită simplității lor,filtrele de tip cutie sunt potrivite pentru experimentare rapidă și oferă de obicei rezultate de netezire acceptabile vizual. Cu toate acestea, filtrele de tip cutie au limitări care le fac alegeri mai puțin ideale în multe aplicații. Kernelurile Gaussian sunt utilizate frecvent datorită proprietăților lor utile și capacitatea de a netezi imaginile în mod eficient. Kernelurile Gaussian sunt simetrice circular (isotropic), ceea ce înseamnă că răspunsul lor este independent de orientare.

Kernelul Gaussian este dat de formula:

$$w(s, t) = G(s, t) = K e^{-\frac{s^2+t^2}{2\sigma^2}}$$

Kernelurile Gaussian sunt singurele kerneluri simetrice circulare și separabile, ceea ce le conferă avantaje computaționale similare cu filtrele de tip cutie, dar cu proprietăți suplimentare care le fac ideale pentru procesarea imaginii.

Prin  $r = \sqrt{s^2 + t^2}$ , putem scrie ecuația:

$$G(r) = K e^{-\frac{r^2}{2\sigma^2}}$$

unde  $K$  este o constantă de normalizare, iar  $\sigma$  este deviația standard a distribuției Gaussian.

Rezultatul conoluției este deosebit de important în filtrare. De exemplu, filtrarea se poate face în etape succesive, iar același rezultat poate fi obținut printr-o singură etapă de filtrare cu un kernel compozit format ca și conoluție a kernelurilor individuale.

## 2.8 Filtre trece-sus

Detectarea marginilor este un pas esențial în procesarea imaginilor, deoarece ne permite să identificăm schimbările în intensitate și să găsim contururile obiectelor din imagine. În general, detectarea marginilor implică utilizarea derivatelor de ordinul întâi sau al doilea pentru a determina schimbările în intensitatea pixelilor.

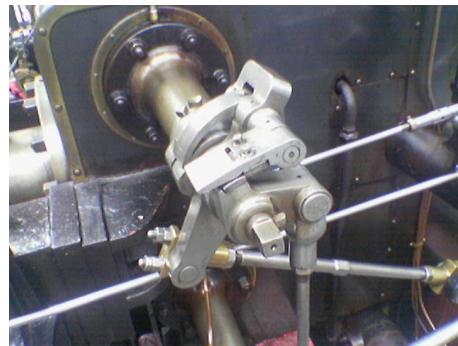


Figura 2.9: Imagine de Input

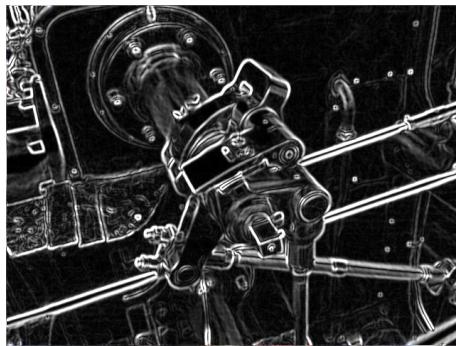


Figura 2.10: Operatorul Sobel

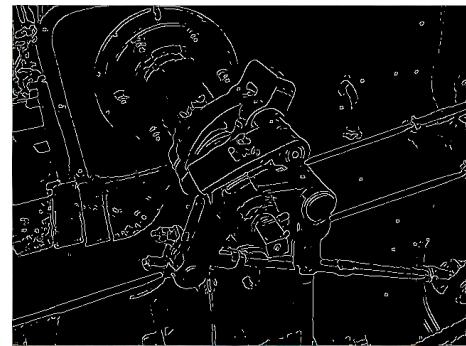


Figura 2.11: Filtru Laplace

Instrumentul principal pentru găsirea direcției și intensității marginii într-o imagine este gradientul. Gradientul unei imagini  $f(x, y)$  la un punct arbitrar  $(x, y)$  este notat și definit ca vectorul:

$$\nabla f(x, y) \equiv \text{grad}[f(x, y)] \equiv \begin{bmatrix} g_x(x, y) \\ g_y(x, y) \end{bmatrix} \equiv \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$

Acest vector are proprietatea binecunoscută că indică direcția în care rata de schimbare a  $f$  la  $(x, y)$  este maximă.

Magnitudinea  $M(x, y)$  a acestui vector gradient la un punct  $(x, y)$  este dată de norma Euclidiană:

$$M(x, y) = \|\nabla f(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)}$$

Aceasta este valoarea ratei de schimbare în direcția vectorului gradient la punctul  $(x, y)$ .

Direcția vectorului gradient la un punct  $(x, y)$  este dată de:

$$\alpha(x, y) = \tan^{-1} \left( \frac{g_y(x, y)}{g_x(x, y)} \right)$$

Unghiiurile sunt măsurate în sensul acelor de ceasornic față de axa  $x$ .

Obținerea gradientului unei imagini necesită calcularea derivatelor partiale  $\partial f / \partial x$  și  $\partial f / \partial y$  la fiecare locație a pixelilor din imagine. Pentru gradient, utilizăm de obicei o diferență finită înainte sau centrată. Utilizând diferențele înainte, obținem:

$$g_x(x, y) = \frac{\partial f(x, y)}{\partial x} \approx f(x+1, y) - f(x, y)$$

$$g_y(x, y) = \frac{\partial f(x, y)}{\partial y} \approx f(x, y+1) - f(x, y)$$

Kernelurile de mărime 2x2 sunt simple conceptual, dar nu sunt la fel de utile pentru calcularea direcției marginilor precum kernelurile care sunt simetrice în jurul centrelor lor. Cele mai mici astfel de kerneluri au dimensiunea 3x3.

Operatorii Prewitt sunt un alt exemplu de operatori de gradient care utilizează kerneluri de dimensiune 3x3:

$$g_x = \frac{\partial f}{\partial x} \approx (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$

$$g_y = \frac{\partial f}{\partial y} \approx (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$$

Acești operatori sunt destul de eficienți în calcularea gradientului, deoarece iau în considerare natura datelor pe părțile opuse ale punctului central și astfel transportă mai multe informații privind direcția unei margini. Operatorii Prewitt, de exemplu, sunt adesea utilizati datorită capacitatea lor de a oferi o aproximare bună a derivatelor partiale în imagini.

Un alt set de operatori utilizat frecvent pentru calculul gradientului este setul de operatori Sobel. Acesta folosește, de asemenea, kerneluri de dimensiune 3x3, dar cu un factor suplimentar de ponderare în centrul kernelului, ceea ce ajută la îmbunătățirea sensibilității la zgromot. Kernelurile pentru operatorii Sobel sunt:

$$g_x = \frac{\partial f}{\partial x} \approx \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$g_y = \frac{\partial f}{\partial y} \approx \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Utilizând aceste kerneluri, operatorii Sobel calculează gradientul pe direcțiile  $x$  și  $y$ , oferind o estimare robustă a direcției și magnitudinii gradientului, ceea ce face ca acești operatori să fie extrem de utili în detectarea marginilor în imagini.

Gradientul morfologic este o altă tehnică în procesarea imaginilor pentru evidențierea marginilor obiectelor. Această tehnică se bazează pe operații morfologice fundamentale precum dilatarea și eroziunea.

Gradientul morfologic este obținut prin calcularea diferenței dintre o imagine dilată și una erodată. Această operație evidențiază marginile obiectelor din imagine, deoarece dilatarea și eroziunea modifică în mod complementar marginile acestora. Gradientul morfologic  $G$  este definit ca:

$$G = (I \oplus B) - (I \ominus B)$$

unde  $I$  este imaginea binară inițială și  $B$  este elementul structurant utilizat pentru dilatare și eroziune.

## 2.9 Componente conexe

Identificarea componentelor conexe este o operațiune esențială în procesarea imaginii, utilizată pentru a găsi regiuni de pixeli adiacenți care au aceeași valoare de intensitate. Aceste regiuni sunt denumite componente conexe. Pixelii sunt considerați  $N4$ -adiacenți dacă sunt adiacenți fie orizontal, fie vertical, și  $N8$ -adiacenți dacă pot fi și adiacenți diagonali.

Componentele conexe sunt utilizate pe scară largă în diverse aplicații, cum ar fi recunoașterea literelor dintr-un document scanat sau identificarea obiectelor (de exemplu, celule) într-o imagine segmentată și calcularea statisticilor de arie. De-a lungul anilor, au fost dezvoltate o varietate de algoritmi eficienți pentru a găsi astfel de componente. Astfel de algoritmi sunt de obicei inclusi în bibliotecile de procesare a imaginii.

După ce o imagine binară sau multivalorată a fost segmentată în componente sale conexe, este util să calculăm statisticile de arie pentru fiecare regiune individuală  $\mathcal{R}$ . Astfel de statistici includ:

- **Aria** (numărul de pixeli);
- **Perimetru** (numărul de pixeli de la margine);
- **Centroidul** (valorile medii  $x$  și  $y$ );
- **Momentul de ordinul doi**,

$$M = \sum_{(x,y) \in \mathcal{R}} \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix} \begin{bmatrix} x - \bar{x} & y - \bar{y} \end{bmatrix},$$

de la care se pot calcula orientarea și lungimile axelor major și minor folosind analiza valorilor proprii.

Aceste statistici pot fi apoi utilizate pentru prelucrarea ulterioară, de exemplu, pentru sortarea regiunilor în funcție de dimensiune (pentru a considera mai întâi regiunile cele mai mari) sau pentru potrivirea preliminară a regiunilor din imagini diferite.

Boxurile de delimitare sunt utilizate pentru a încadra componentelete conexe identificate într-o imagine. Fiecare componentă conexă este înconjurată de un dreptunghi minim care cuprinde toți pixelii din componenta respectivă. Parametrii acestui dreptunghi sunt:

- **Coordonatele colțului din stânga-sus** ( $x_{min}, y_{min}$ );
- **Lățimea și înălțimea** ( $w, h$ ).

Aceste boxuri de delimitare sunt esențiale în diverse aplicații de vizualizare și analiză a imaginii, oferind un mijloc simplu și eficient de localizare și izolare a componentelor de interes.

Componentelor conexe și utilizarea boxurilor de delimitare sunt instrumente fundamentale în procesarea imaginilor, permitând izolarea eficientă și analiza precisă a regiunilor de interes în imagini binare și multivalorate.

## 2.10 Contururi

Algoritmii precum detectorul de margini Canny pot fi utilizați pentru a găsi pixeli de margine care separă diferite segmente într-o imagine, dar nu ne oferă informații despre acele margini ca entități în sine. Pasul următor este să asamblăm acei pixeli de margine în contururi. Un contur este o listă de puncte care reprezintă, într-un fel sau altul, o curbă într-o imagine. Această reprezentare poate varia în funcție de circumstanțele specifice.

Există multe modalități de a reprezenta o curbă. Contururile pot fi reprezentate prin secvențe de puncte 2D. De exemplu, o astfel de reprezentare este lanțul Freeman, în care fiecare punct este reprezentat ca un „pas” într-o direcție dată de la punctul anterior. Deși secvențele de puncte 2D sunt cele mai comune, există și alte modalități de a reprezenta contururile.

Funcțiile de calcul al contururilor pornesc de la imagini binare. Acestea pot folosi imagini create de algoritmi de detectare a marginilor, cum ar fi detectorul Canny, sau imagini create prin aplicarea unor funcții de prag, în care marginile sunt implicate ca limite între regiuni pozitive și negative.

Înainte de a detalia modul de extragere a contururilor, este important să înțelegem ce este un contur și cum pot fi relaționate grupurile de contururi între ele. Un concept deosebit de important este arborele de contururi, care encodează relațiile de includere între contururi.

Un arbore de contururi este o structură ierarhică în care fiecare nod reprezintă un contur. Fiecare nod poate avea copii care sunt contururi incluse în conturul părinte. Această structură ierarhică poate fi reprezentată prin array-uri în care fiecare intrare reprezintă un contur specific. Fiecare intrare conține un set de patru întregi care indică alte noduri în ierarhie cu o relație particulară față de nodul curent. Dacă o relație specifică nu există, elementul corespunzător din structură este setat la -1 (de exemplu, ID-ul părintelui pentru nodul rădăcină ar avea valoarea -1 deoarece nu are părinte).

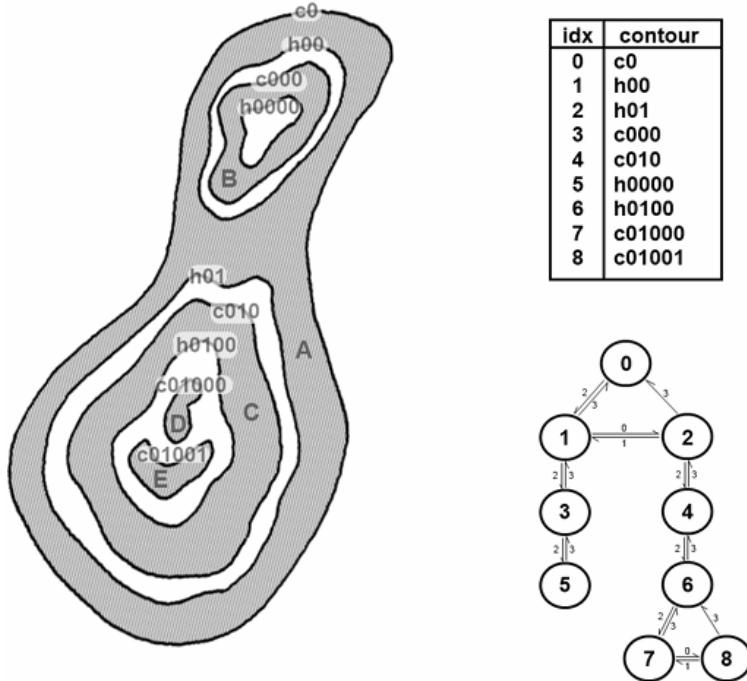


Figura 2.12: Regiuni

Imaginea de mai sus conține mai multe regiuni pe un fundal alb. Contururile acestor regiuni pot fi găsite și aranjate într-un arbore de contururi. De exemplu, dacă avem cinci regiuni, rezultând un total de nouă contururi (inclusiv atât marginile exterioare, cât și cele interioare ale fiecărei regiuni), fiecare nod din arbore va avea ca și copii acele contururi care sunt incluse în conturul respectiv. Arboarele rezultat poate fi vizualizat, arătând legăturile valide și relațiile dintre noduri.

Contururile sunt structuri esențiale în procesarea imaginii, reprezentând margini și forme într-o manieră organizată. Înțelegerea și utilizarea contururilor permite realizarea unor operațiuni avansate de analiză și manipulare a imaginii, inclusiv identificarea obiectelor și segmentarea precisă a regiunilor de interes.

## 2.11 Analiza componentelor conexe

După segmentarea unei imagini, de obicei prin binarizare, putem folosi analiza componentelor conexe pentru a izola și procesa eficient regiunile rezultante ale imaginii una câte una. Analiza componentelor conexe are ca scop etichetarea regiunilor conectate dintr-o imagine binară. Prin etichetarea acestor regiuni, putem izola și analiza fiecare componentă în parte.

- Intrare:** Intrarea este o imagine binară în care prim-planul și fundalul sunt clar definite.
- Etichetarea:** Fiecare pixel din prim-plan primește o etichetă, indicând care componentă conexă o reprezintă. Acest lucru are ca rezultat o hartă de pixeli etichetați unde toți pixelii din aceeași componentă conexă împărtășesc aceeași etichetă.

3. **Ieșire:** ieșirea este o imagine etichetată și, optional, un set de statistici pentru fiecare componentă conexă, cum ar fi aria, dreptunghiul de delimitare și centrul de masă.

Analiza componentelor conexe este destul de populară în algoritmii de segmentare a fundalului ca filtru post-procesare care elimină petele mici de zgomot și în probleme cum ar fi OCR, unde există un prim-plan bine definit de extras. Este esențial ca această operație de bază să fie rapidă.

## 2.12 Operații logice

Operațiile logice lucrează cu variabile și expresii TRUE (de obicei denotate prin 1) și FALSE (de obicei denotate prin 0). În cazul nostru, aceasta înseamnă imagini binare compuse din pixeli de prim-plan (valorați cu 1) și un fundal format din pixeli cu valoarea 0.

Utilizăm operatori logici și de set pe imagini binare folosind două abordări principale:

1. Utilizăm coordonatele regiunilor individuale de pixeli de prim-plan într-o singură imagine ca seturi.
2. Lucrăm cu una sau mai multe imagini de aceeași dimensiune și efectuăm operații logice între pixelii corespunzători din aceste matrice.

În prima categorie, o imagine binară poate fi privită ca o diagramă Venn în care coordonatele regiunilor de pixeli valorați cu 1 sunt tratate ca seturi. Uniunea acestor seturi cu setul compus din pixeli valorați cu 0 formează universul setului,  $\mathcal{E}$ . De exemplu, pentru o imagine binară cu două regiuni de pixeli valorați cu 1,  $R_1$  și  $R_2$ , putem determina dacă regiunile se suprapun (au cel puțin o pereche de coordonate comune) prin efectuarea operației de intersecție a seturilor  $R_1 \cap R_2$ .

În a doua abordare, efectuăm operații logice pe pixelii unei imagini binare sau pe pixelii corespunzători din două sau mai multe imagini binare de aceeași dimensiune. Operatorii logici pot fi definiți în termeni de tabele de adevăr. Operația logică AND (denotată și  $\wedge$ ) produce 1 (TRUE) doar când ambii  $a$  și  $b$  sunt 1. În mod similar, OR ( $\vee$ ) produce 1 când  $a$  sau  $b$  sau ambii sunt 1, iar operatorul NOT ( $\neg$ ) este auto-explicativ. Operatorii AND și OR sunt *elementwise* în acest context, operând pe perechi de pixeli corespunzători între imagini.

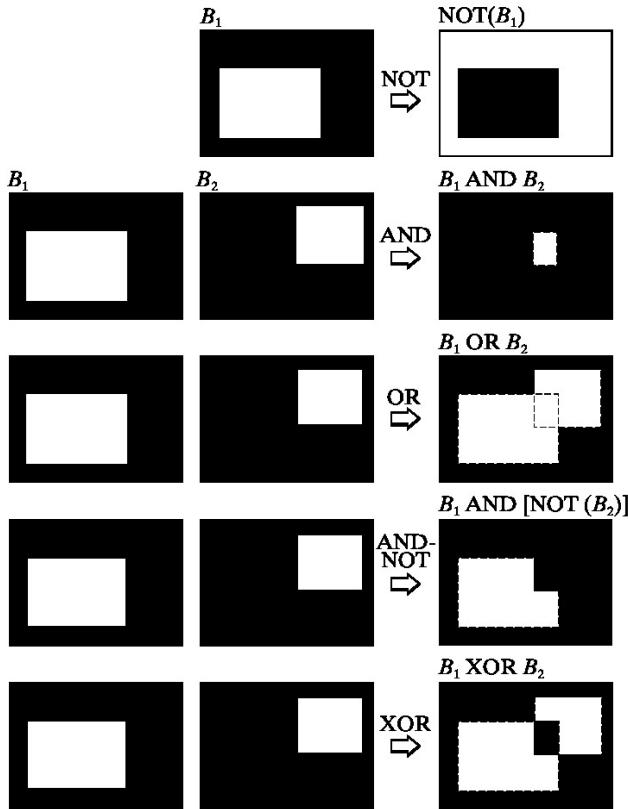


Figura 2.13: Operații logice

Figura ilustrează operații logice folosind a două abordare discutată. NOT al imaginii binare  $B_1$  este o matrice obținută prin schimbarea tuturor pixelilor valorați cu 1 în 0 și invers. AND dintre  $B_1$  și  $B_2$  conține 1 în toate locațiile unde elementele corespunzătoare ale  $B_1$  și  $B_2$  sunt 1; operația produce 0 în altă parte. OR dintre cele două imagini este o matrice care conține 1 în locațiile unde elementele corespunzătoare ale  $B_1$  sau  $B_2$  sau ambele sunt 1. XOR (exclusive OR) produce 1 în locațiile unde elementele corespunzătoare ale  $B_1$  sau  $B_2$  (dar nu ambele) sunt 1.

Putem obține aceleași rezultate folosind prima abordare discutată mai sus. Începem prin etichetarea regiunilor individuale valorați cu 1 în fiecare dintre cele două imagini. Să notăm cu A și B seturile de coordonate ale tuturor pixelilor valorați cu 1 în imaginile  $B_1$  și  $B_2$ . Apoi formăm o singură matrice prin aplicarea operației OR între cele două imagini, păstrând etichetele A și B. Rezultatul ar arăta ca matricea  $B_1 \vee B_2$ , dar cu cele două regiuni albe etichetate A și B, asemănătoare unui diagramă Venn.

## 2.13 Operații morfologice

Morfologia imaginii este un subiect de sine stătător ce cuprinde un număr mare de operații morfologice care au fost dezvoltate în special în primii ani ai viziunii compute-rizate. Majoritatea operațiilor au fost dezvoltate pentru un scop specific sau altul, iar unele dintre acestea și-au găsit o utilitate mai largă de-a lungul anilor. Practic, toate operațiile morfologice se bazează pe doar două operații primitive.

Transformările morfologice de bază se numesc dilatare și eroziune și apar într-o varietate de contexte, cum ar fi eliminarea zgomotului, izolarea elementelor individuale

și unirea elementelor disparate într-o imagine. Operațiile morfologice mai sofisticate, bazate pe aceste două operații de bază, pot fi utilizate pentru a găsi vârfuri de intensitate (sau găuri) într-o imagine și pentru a defini (încă o dată) o formă particulară a gradientului de imagine.

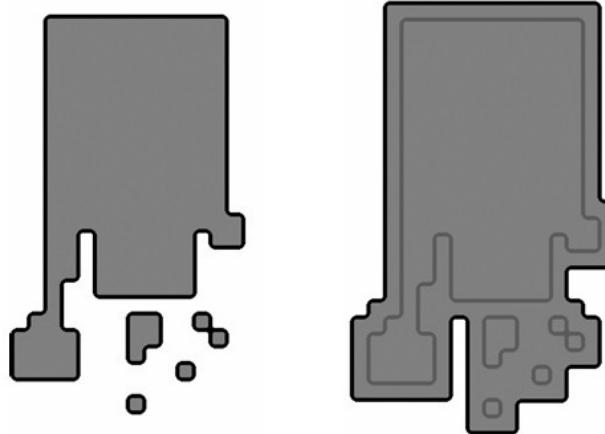


Figura 2.14: Imagine de Input

Figura 2.15: Operația de dilatare

Dilatarea este o conoluție a unei imagini cu un nucleu în care orice pixel dat este înlocuit cu maximul local al tuturor valorilor pixelilor acoperiți de nucleu. Așa cum am menționat anterior, aceasta este un exemplu de operație neliniară, deci nucleul nu poate fi exprimat în forma arătată. Cel mai adesea, nucleul folosit pentru dilatare este un nucleu pătrat „solid” sau uneori un disc, cu punctul de ancorare în centru. Efectul dilatării este de a face ca regiunile umplute dintr-o imagine să crească.

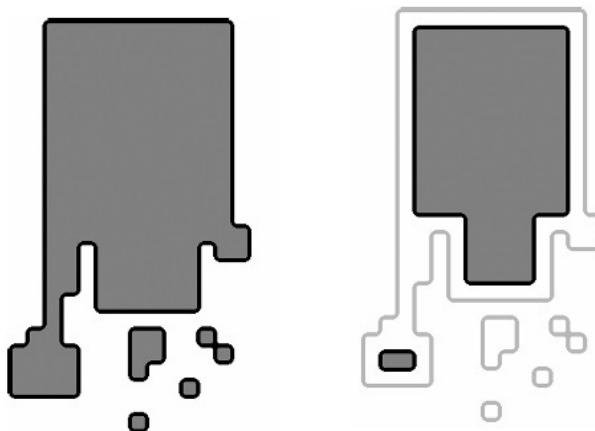


Figura 2.16: Imagine de Input

Figura 2.17: Operația de erodare

Eroziunea este operația opusă. Acțiunea operatorului de eroziune este echivalentă cu calcularea unui minim local peste zona nucleului. În general, în timp ce dilatarea extinde o regiune luminoasă, eroziunea reduce o astfel de regiune luminoasă. Mai mult, dilatarea va tinde să umple concavitățile, iar eroziunea va tinde să eliminate proeminențele. Desigur, rezultatul exact va depinde de nucleu, dar aceste afirmații sunt în general adevărate atât timp cât nucleul este convex și umplut.

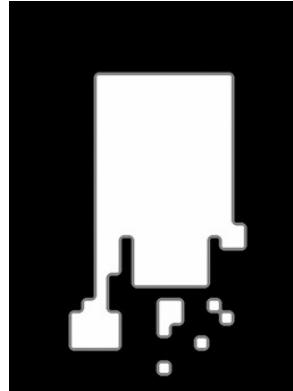


Figura 2.18: Imagine de Input

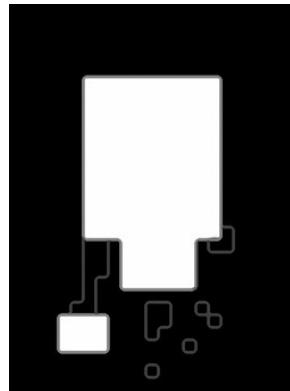


Figura 2.19: Operația de deschidere

Operațiile de deschiderea și închiderea, sunt de fapt combinații simple ale operațiilor de eroziune și dilatare. În cazul deschiderii, erodăm mai întâi și apoi dilatăm. Deschiderea este adesea folosită pentru a număra regiunile într-o imagine binară. De exemplu, dacă am pragat o imagine a celulelor pe o lamă de microscop, am putea folosi deschiderea pentru a separa celulele care sunt aproape una de cealaltă înainte de a număra regiunile.

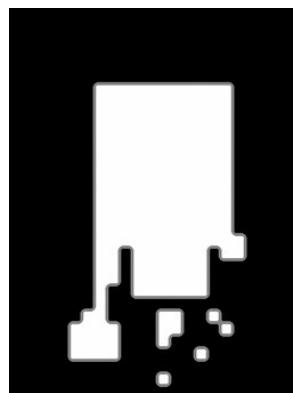


Figura 2.20: Imagine de Input

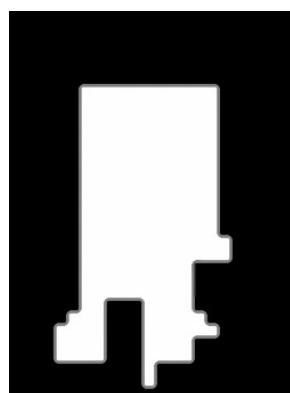


Figura 2.21: Operația de închidere

În cazul închiderii, dilatăm mai întâi și apoi erodăm. Închiderea este utilizată în majoritatea algoritmilor de componente conexe mai sofisticăți pentru a reduce segmentele nedorite sau generate de zgomot. Pentru componentele conexe, de obicei se efectuează mai întâi o eroziune sau o operație de închidere pentru a elimina elementele care apar doar din zgomot, și apoi se folosește o operație de deschidere pentru a conecta regiunile mari din apropiere. Observați că, deși rezultatul final al utilizării deschiderii sau închiderii este similar cu utilizarea eroziunii sau dilatarii, aceste noi operații tend să păstreze mai precis aria regiunilor conexe.

## 2.14 Transformări geometrice

Utilizăm transformările geometrice pentru a modifica aranjamentul spațial al pixelilor dintr-o imagine. Transformările geometrice ale imaginilor digitale constau în două operații de bază:

1. Transformarea spațială a coordonatelor.
2. Interpolarea intensității care atribuie valori de intensitate pixelilor transformați spațial.

Transformarea coordonatelor poate fi exprimată astfel:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = T \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

unde  $(x, y)$  sunt coordonatele pixelilor în imaginea originală și  $(x', y')$  sunt coordonatele pixelilor corespunzători din imaginea transformată. De exemplu, transformarea  $(x', y') = (x/2, y/2)$  micșorează imaginea originală la jumătate în ambele direcții spațiale.

Ne interesează așa-numitele *transformări affine*, care includ scalarea, translația, rotația și deformarea. Caracteristica principală a unei transformări affine în 2D este că păstrează punctele, liniile drepte și planele. Ecuată definită anterior poate fi folosită pentru a exprima transformările menționate mai sus, cu excepția translației, care ar necesita adăugarea unui vector constant 2D în partea dreaptă a ecuației. Cu toate acestea, este posibil să folosim coordonate omogene pentru a exprima toate cele patru transformări affine folosind o singură matrice  $3 \times 3$  în următoarea formă generală:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Această transformare poate scala, roti, translata sau deforma o imagine, în funcție de valorile alese pentru elementele matricei A.

Transformarea anterioară mută coordonatele pixelilor dintr-o imagine în locații noi. Pentru a finaliza procesul, trebuie să atribuim valori de intensitate pixelilor relocați. Această sarcină este realizată folosind *interpolarea intensității*.

Pot utiliza ecuația în două moduri de bază. Primul este *maparea directă*, care constă în scanarea pixelilor imaginii de intrare și, la fiecare locație  $(x, y)$ , calcularea locației spațiale  $(x', y')$  a pixelului corespunzător în imaginea de ieșire folosind direct ecuația.

Al doilea mod, numit *mapare inversă*, scană locațiile pixelilor de ieșire și, la fiecare locație  $(x', y')$ , calculează locația corespunzătoare în imaginea de intrare folosind  $(x, y) = A^{-1}(x', y')$ . Apoi interpolează printre cei mai apropiati pixeli de intrare pentru a determina intensitatea valorii pixelului de ieșire.

## 2.15 Transformata Hough

Adesea, trebuie să lucrăm în medii neorganizate în care avem doar o hartă de margini și nicio informație despre unde ar putea fi obiectele de interes. În astfel de situații, toți pixelii sunt candidați pentru legături și, astfel, trebuie să fie acceptați sau eliminați pe baza unor proprietăți globale definite. În această secțiune, dezvoltăm o abordare bazată pe faptul dacă seturile de pixeli se află pe curbe de o anumită formă. Odată detectate, aceste curbe formează marginile sau contururile regiunilor de interes.

Dat fiind  $n$  puncte într-o imagine, presupunem că dorim să găsim submulțimi ale acestor puncte care se află pe linii drepte. O soluție posibilă este să găsim toate liniile determinate de fiecare pereche de puncte, apoi să găsim toate submulțimile de puncte care sunt aproape de liniile particulare. Această abordare implică găsirea  $\frac{n(n-1)}{2} \sim n^2$  liniilor, apoi efectuarea  $n \left( \frac{n(n-1)}{2} \right) \sim n^3$  comparației ale fiecărui punct pentru toate liniile.

Transformata Hough, propusă de Hough în 1962, este o metodă alternativă pentru detectarea liniilor. Fie  $(x_i, y_i)$  un punct în planul  $xy$  și considerăm ecuația generală a unei liniilor în formă de panta-interceptare:  $y_i = ax_i + b$ . Infinit de multe liniile trec prin  $(x_i, y_i)$ , dar toate satisfac ecuația  $y_i = ax_i + b$  pentru valori variante ale lui  $a$  și  $b$ . Totuși, scriind această ecuație ca  $b = -x_i a + y_i$  și considerând planul  $ab$  (numit și spațiul parametrilor), obținem ecuația unei singure liniilor pentru un punct fix  $(x_i, y_i)$ .

$$x \cos \theta + y \sin \theta = \rho$$

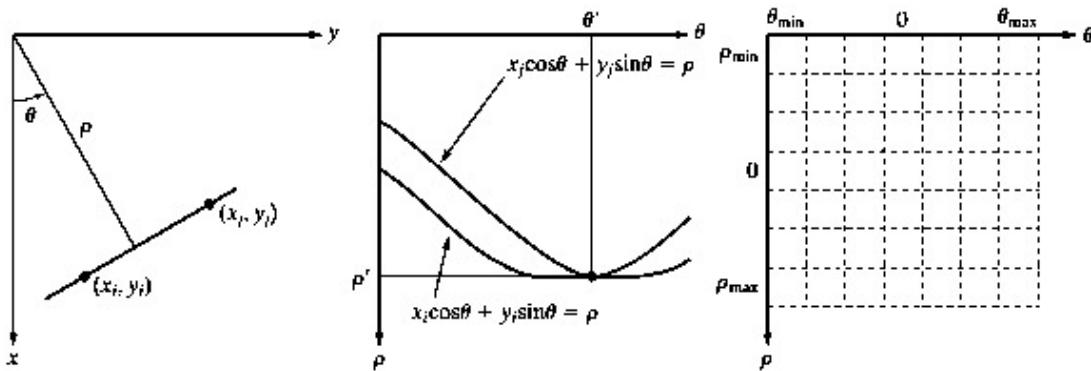


Figura 2.22: Interpretare geometrică

Figura ilustrează interpretarea geometrică a parametrilor  $\rho$  și  $\theta$ . O linie orizontală are  $\theta = 0^\circ$ , cu  $\rho$  egal cu interceptul pozitiv pe axa  $x$ . Similar, o linie verticală are  $\theta = 90^\circ$ , cu  $\rho$  egal cu interceptul pozitiv pe axa  $y$ , sau  $\theta = -90^\circ$ , cu  $\rho$  egal cu interceptul negativ pe axa  $y$ .

Atractia computațională a transformatei Hough provine din subdivizarea spațiului parametrilor  $\rho, \theta$  în celule de acumulare. Procedura constă în incrementarea  $\theta$  și calcularea valorilor corespunzătoare  $\rho$  folosind ecuația:

$$\rho = x_i \cos \theta + y_i \sin \theta$$

Celula la coordonatele  $(i, j)$  cu valoare de acumulare  $A(i, j)$  corespunde pătratului asociat cu coordonatele spațiului parametrilor  $(\rho, \theta)$ . Inițial, aceste celule sunt setate la zero. Apoi, pentru fiecare punct non-fundamental  $(x_k, y_k)$  în planul  $xy$ , incrementăm  $\theta$  și calculăm  $\rho$  folosind ecuația de mai sus. Valorile rezultante  $\rho$  sunt rotunjite la cea mai apropiată valoare permisă de-a lungul axei  $\rho$  și se actualizează celula de acumulare  $A(p, q)$ .

Transformata Hough este un instrument puternic pentru detectarea formelor într-o imagine și este folosită în multe aplicații practice, de la analiza imaginilor medicale la navigația autonomă.

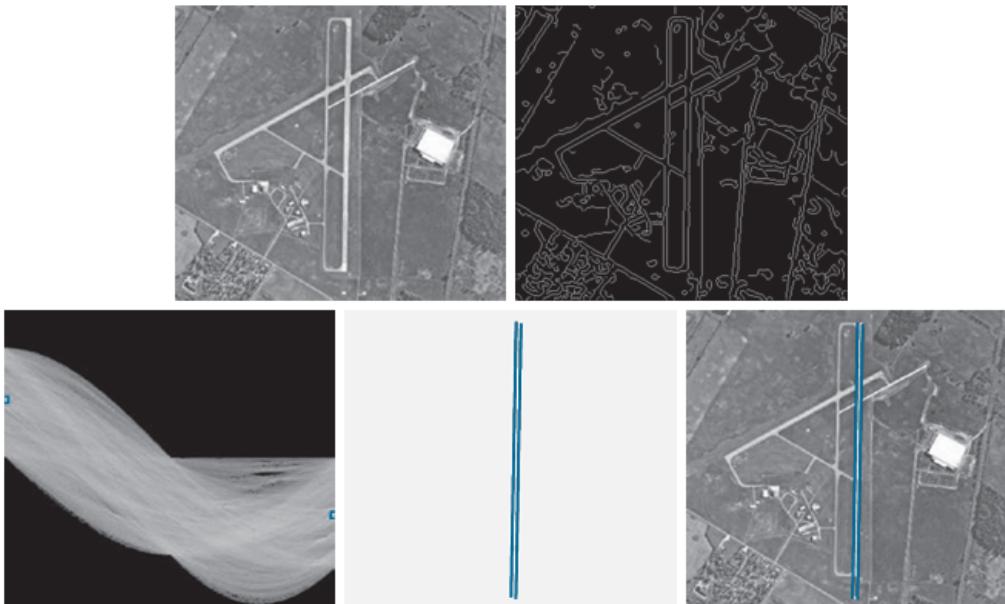


Figura 2.23: Detectarea liniilor unei piste de aterizare

Figura de mai sus arată o imagine aeriană a unui aeroport. Obiectivul acestui exemplu este să utilizăm transformata Hough pentru a extrage cele două margini care definesc pistă principală. O soluție la o astfel de problemă ar putea fi de interes, de exemplu, în aplicații care implică navigația aeriană autonomă.

Primul pas este obținerea unei hărți de margini. Figura a doua arată harta de margini obținută folosind algoritmul Canny. În scopul calculării transformatei Hough, rezultate similare pot fi obținute utilizând oricare dintre celelalte tehnici de detecție a marginilor existente. Figura a treia arată spațiul parametrilor Hough obținut folosind incrementări de  $1^\circ$  pentru  $\theta$  și incrementări de un pixel pentru  $\rho$ .

Pista de interes este orientată aproximativ  $1^\circ$  față de direcția nord, astfel încât selectăm celulele corespunzătoare la  $\pm 90^\circ$  și care conțin numărul cel mai mare, deoarece pistele sunt cele mai lungi linii orientate în aceste direcții. Cutiile mici de pe marginile imaginii trei evidențiază aceste celule. După cum s-a menționat anterior în legătură cu figura a doua, transformata Hough arată adiacență la marginile imaginii. O altă modalitate de interpretare a acestei proprietăți este că o linie orientată la  $+90^\circ$  și o linie orientată la  $-90^\circ$  sunt echivalente (adică, ambele sunt verticale). Imaginea a patra arată liniile corespunzătoare celor două celule de acumulare discutate anterior, iar figura a cincea arată liniile suprapuse pe imaginea originală. Liniile au fost obținute prin unirea tuturor spațiilor care nu depășesc 20% (aproximativ 100 pixeli) din înălțimea imaginii. Aceste lini îcorespund clar marginilor pistei de interes.

Observați că singura informație necesară pentru a rezolva această problemă a fost orientarea pistei și poziția observatorului în raport cu aceasta. Cu alte cuvinte, un vehicul care navighează autonom ar ști că, dacă pistă de interes este orientată spre nord, iar direcția de deplasare a vehiculului este, de asemenea, spre nord, pistă ar trebui să apară vertical în imagine. Alte orientări relative sunt tratate într-un mod similar. Orientările pistelor din întreaga lume sunt disponibile în hărțile de zbor, iar direcția de deplasare poate fi obținută cu ușurință folosind informațiile de la GPS (Global Positioning System). Aceste informații ar putea fi folosite, de asemenea, pentru a calcula distanța dintre vehicul și pistă, permitând astfel estimarea unor parametri, cum ar fi lungimea așteptată a liniilor în raport cu dimensiunea imaginii.

## 2.16 Distanța de la un punct la o dreaptă

Considerăm o dreaptă în planul  $xy$  definită de două puncte  $(x_1, y_1)$  și  $(x_2, y_2)$ . Ecuția generală a acestei drepte poate fi scrisă în forma:

$$ax + by + c = 0$$

unde:

- coeficientul  $a$  este diferența dintre ordonatele celor două puncte ale dreptei:

$$a = y_1 - y_2$$

- coeficientul  $b$  este diferența dintre abscisele celor două puncte ale dreptei:

$$b = x_2 - x_1$$

- termenul liber  $c$  este dat de formula:

$$c = x_1y_2 - x_2y_1$$

De asemenea, considerăm un punct  $(x_0, y_0)$  în planul  $xy$ . Distanța de la acest punct la dreaptă este dată de formula:

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

Această formulă ne permite să calculăm distanța perpendiculară de la un punct dat la o dreaptă specificată prin două puncte în planul  $xy$ .

## 2.17 Punctul de intersecție a două drepte

Considerăm două drepte în planul  $xy$ , fiecare definită de două puncte. Prima dreaptă este definită de punctele  $(x_1, y_1)$  și  $(x_2, y_2)$ , iar a doua dreaptă este definită de punctele  $(x_3, y_3)$  și  $(x_4, y_4)$ .

Ecuția generală a unei drepte în planul  $xy$  poate fi scrisă în forma:

$$Ax + By = C$$

Pentru prima dreaptă, ecuația este:

$$A_1x + B_1y = C_1$$

unde:

$$A_1 = y_2 - y_1, \quad B_1 = x_1 - x_2, \quad C_1 = A_1x_1 + B_1y_1$$

Pentru a doua dreaptă, ecuația este:

$$A_2x + B_2y = C_2$$

unde:

$$A_2 = y_4 - y_3, \quad B_2 = x_3 - x_4, \quad C_2 = A_2x_3 + B_2y_3$$

Punctul de intersecție  $(x, y)$  al celor două drepte poate fi determinat rezolvând sistemul de ecuații liniare:

$$\begin{cases} A_1x + B_1y = C_1 \\ A_2x + B_2y = C_2 \end{cases}$$

Pentru a rezolva acest sistem, folosim determinantul:

$$D = A_1B_2 - A_2B_1$$

Dacă  $D \neq 0$ , sistemul are o soluție unică, iar coordonatele punctului de intersecție sunt date de:

$$x = \frac{B_2C_1 - B_1C_2}{D}$$

$$y = \frac{A_1C_2 - A_2C_1}{D}$$

Dacă  $D = 0$ , dreptele sunt paralele și nu se intersectează într-un punct finit.

## 2.18 Distanța dintre două drepte

Considerăm două drepte în planul  $xy$ . Prima dreaptă este definită de coordonatele  $(x_1, y_1)$  și  $(x_2, y_2)$ , iar a doua dreaptă de referință este definită de coordonatele  $(x_r, y_r)$  și  $(x_s, y_s)$ .

Pentru a determina poziția relativă a unei drepte față de dreapta de referință, mai întâi calculăm punctul mediu al primei drepte. Coordonatele punctului mediu  $(x_m, y_m)$  sunt date de:

$$x_m = \frac{x_1 + x_2}{2}$$

$$y_m = \frac{y_1 + y_2}{2}$$

Distanța dintre punctul mediu  $(x_m, y_m)$  și dreapta de referință este calculată folosind formula:

$$d = (x_m - x_r) \cdot (y_s - y_r) - (y_m - y_r) \cdot (x_s - x_r)$$

Această formulă este derivată din proprietățile determinantului și reprezintă o măsură a distanței semnate dintre punctul mediu și dreapta de referință. Semnul distanței indică poziția relativă a punctului mediu față de dreapta de referință. Astfel:

- dacă  $d > 0$ , punctul mediu  $(x_m, y_m)$  se află de o parte a dreptei de referință,
- dacă  $d < 0$ , punctul mediu  $(x_m, y_m)$  se află de cealaltă parte a dreptei de referință,
- dacă  $d = 0$ , punctul mediu  $(x_m, y_m)$  se află exact pe dreapta de referință.

## 2.19 Determinarea unei drepte care trece printr-un punct

Considerăm un punct dat  $(x_0, y_0)$  în planul  $xy$  și o pantă  $m$ . Obiectivul este de a determina ecuația dreptei care trece prin acest punct și are panta specificată.

Ecuația generală a unei drepte în planul  $xy$  este dată de:

$$y = mx + b$$

unde  $m$  este panta dreptei și  $b$  este interceptul pe axa  $y$ .

Pentru a determina interceptul  $b$ , utilizăm coordonatele punctului  $(x_0, y_0)$ . Substuiuim aceste coordonate în ecuația generală a dreptei:

$$y_0 = mx_0 + b$$

Rezolvăm pentru  $b$ :

$$b = y_0 - mx_0$$

Pentru a determina punctele de start și de sfârșit ale dreptei în cadrul unui sistem de coordonate de dimensiune  $(W, H)$ , procedăm în funcție de direcția în care dorim să extindem dreapta.

1. Extinderea dreptei pe direcția orizontală

■ Punctul de start  $(x_s, y_s)$ :

$$x_s = 0, \quad y_s = b$$

■ Punctul de sfârșit  $(x_e, y_e)$ :

$$x_e = W, \quad y_e = mW + b$$

2. Extinderea dreptei pe direcția verticală

■ Dacă  $m \neq 0$ :

– Punctul de start  $(x_s, y_s)$ :

$$x_s = -\frac{b}{m}, \quad y_s = 0$$

– Punctul de sfârșit  $(x_e, y_e)$ :

$$x_e = \frac{H - b}{m}, \quad y_e = H$$

■ Dacă  $m = 0$ :

– Punctul de start  $(x_s, y_s)$  și punctul de sfârșit  $(x_e, y_e)$ :

$$x_s = x_e = x_0, \quad y_s = 0, \quad y_e = H$$

Astfel, ecuația dreptei și punctele de start și sfârșit pot fi determinate folosind panta  $m$  și punctul dat  $(x_0, y_0)$ . Acest proces ne permite să trasăm o dreaptă definită clar în cadrul unui sistem de coordonate specificat.

# Capitolul 3

## Tehnologii folosite

### 3.1 Limbajul de programare C++

C++ este un limbaj de programare de nivel înalt, cu scop general, creat de omul de știință danez Bjarne Stroustrup. Prima versiune a fost lansată în 1985 ca o extensie a limbajului de programare C și s-a extins semnificativ de-a lungul timpului. Începând cu 1997, C++ a inclus caracteristici orientate pe obiecte, generice și funcționale, pe lângă facilitățiile pentru manipularea memoriei la nivel scăzut. C++ este aproape întotdeauna implementat ca un limbaj compilat, iar mulți furnizori oferă compilatoare.

C++ a fost proiectat având în vedere programarea sistemelor și software-ului embedded, software-ul cu resurse limitate și sistemele mari, punând accent pe performanță, eficiență și flexibilitate de utilizare. C++ s-a dovedit util în multe alte contexte, cu puncte forte cheie în infrastructura software și aplicațiile cu resurse limitate, inclusiv aplicații desktop, jocuri video, servere (de exemplu, comerț electronic, căutare web sau baze de date) și aplicații critice pentru performanță (de exemplu, comutatoare telefonică sau sonde spațiale).

În 1979 a început dezvoltarea unui limbaj numit "C with Classes", care a evoluat ulterior în C++. Motivația pentru acest nou limbaj a venit din necesitatea de a combina caracteristicile utile ale limbajului Simula cu viteza limbajului BCPL. Inițial, "C with Classes" a adăugat la compilatorul C caracteristici precum clase, clase derivate, tipizare puternică, înliniere și argumente implicate. În 1982, limbajul a fost redenumit "C++" și au fost introduse funcții virtuale, supraincarcare de nume de funcții și operatori, referințe, constante, alocare de memorie sigură, verificare îmbunătățită a tipurilor și comentarii pe o singură linie. În 1989, a apărut C++ 2.0, cu caracteristici noi precum moștenirea multiplă și clase abstracte. O evoluție majoră a avut loc în 2011 cu C++11, care a adus numeroase caracteristici noi și extinderea bibliotecii standard.

De-a lungul vieții sale, dezvoltarea și evoluția C++ au fost ghidate de un set de principii:

- Trebuie să fie condus de probleme reale și caracteristicile sale ar trebui să fie imediat utile în programele din lumea reală.
- Fiecare caracteristică ar trebui să fie implementabilă (cu o modalitate destul de evidentă de a face acest lucru).
- Programatorii ar trebui să fie liberi să își aleagă propriul stil de programare, iar

acel stil ar trebui să fie pe deplin susținut de C++.

- Permițând o caracteristică utilă este mai important decât prevenirea fiecărei posibile utilizări greșite a C++.
- Ar trebui să ofere facilități pentru organizarea programelor în părți separate, bine definite și să ofere facilități pentru combinarea părților dezvoltate separat.
- Nu ar trebui să existe încălcări implicate ale sistemului de tipuri (dar să permită încălcări explice; adică, cele solicitate explicit de programator).
- Tipurile create de utilizator trebuie să aibă același suport și performanță ca tipurile încorporate.
- Caracteristicile neutilizate nu ar trebui să afecteze negativ executabilele create (de exemplu, în performanță mai scăzută).
- Nu ar trebui să existe niciun limbaj sub C++ (cu excepția limbajului de asamblare).
- C++ ar trebui să funcționeze alături de alte limbi de programare existente, mai degrabă decât să încurajeze propriul său mediu de programare separat și incompatibil.
- Dacă intenția programatorului este necunoscută, permiteți-i programatorului să o specifice furnizând control manual.

Limbajul C++ are două componente principale: o mapare directă a caracteristicilor hardware oferite în principal de subsetul C și abstracții fără costuri suplimentare bazate pe acele mapări. Stroustrup descrie C++ ca un "limbaj de programare de abstracție ușoară, proiectat pentru construirea și utilizarea de abstracții eficiente și elegante" și "oferind atât acces la hardware, cât și abstracție. Realizarea acestui lucru în mod eficient este ceea ce îl distinge de alte limbi."

C++ moștenește majoritatea sintaxei din C. Un program care respectă standardul C este de asemenea un program valid în C++. Ca și în C, C++ suportă patru tipuri de gestionare a memoriei: obiecte cu durată statică de stocare, obiecte cu durată de stocare pe fir, obiecte cu durată automată de stocare și obiecte cu durată dinamică de stocare.

Obiectele cu durată statică de stocare sunt create înainte de a intra în `main()` și distruse în ordinea inversă creării după ce `main()` iese. Ordinea exactă a creării nu este specificată de standard pentru a permite implementărilor o anumită libertate în modul de organizare a implementării lor. Mai formal, obiectele de acest tip au o durată de viață care "trebuie să dureze pe durata programului".

Variabilele cu durată de stocare pe fir sunt foarte similare cu obiectele cu durată statică de stocare. Principala diferență este că timpul de creare este imediat înainte de crearea firului și distrugerea se face după ce firul a fost unit.

Cele mai comune tipuri de variabile în C++ sunt variabilele locale dintr-o funcție sau bloc și variabilele temporare. Caracteristica comună a variabilelor automate este că au o durată de viață limitată la domeniul variabilei. Ele sunt create și eventual inițializate în punctul de declarație și distruse în ordinea inversă a creării atunci când domeniul este părăsit. Aceasta este implementată prin alocare pe stivă.

Obiectele cu durată dinamică de stocare au o durată de viață dinamică și pot fi create direct cu un apel la `new` și distruse explicit cu un apel la `delete`. C++ suportă de asemenea `malloc` și `free` din C, dar acestea nu sunt compatibile cu `new` și `delete`. Utilizarea

`new` returnează o adresă a memoriei alocate. Ghidurile de bază C++ recomandă evitarea utilizării directe a `new` pentru crearea obiectelor dinamice în favoarea pointerilor inteligenți prin `make_unique<T>` pentru proprietatea unică și `make_shared<T>` pentru proprietatea referențiată multiplu, care au fost introduse în C++11.

## 3.2 Programarea orientată pe obiect

La baza programării orientate pe obiecte (POO) se află conceptul de clasă. O clasă reprezintă o colecție de câmpuri (date) și metode (funcții) care manipulează și utilizează aceste date. Metodele sunt cunoscute și sub denumirea de funcții membre ale clasei. În C++, o clasă poate fi considerată o versiune îmbunătățită a tipului struct din C.

C++ introduce caracteristici de programare orientată pe obiecte în C. Oferă clase, care furnizează cele patru caracteristici comune în limbajele POO: abstractizare, încapsulare, moștenire și polimorfism. O caracteristică distinctivă a claselor C++ comparativ cu clasele din alte limbaje de programare este suportul pentru destructorii deterministici, care, la rândul lor, furnizează suport pentru conceptul de Achiziție a Resurselor este Inițializare (RAII).

O clasă în C++ este un tip de date definit de utilizator, care poate conține atât date (variabile membre), cât și funcții (funcții membre). Clasele permit abstractizarea și organizarea datelor și comportamentului în structuri bine definite. O clasă este definită folosind cuvântul cheie `class` urmat de numele clasei și un bloc de definiții între accolade. Un obiect este o instanță a unei clase. Când o clasă este definită, nu se aloca memorie până când nu se creează un obiect al acelei clase. Obiectele sunt utilizate pentru a accesa datele și funcțiile membre ale unei clase. Însă când o clasă conține metode statice, aceasta nu necesită instanțiere, metodele putând fi apelate în mod direct.

Principiile fundamentale ale programării orientate pe obiecte în C++ includ:

- **Abstractizare:** Ascunderea detaliilor complexe și expunerea unei interfețe simple. O clasă definește o interfață publică prin care utilizatorii pot interacționa cu obiectele fără a cunoaște detaliile interne.
- **Încapsulare:** Gruparea datelor și a funcțiilor care operează pe acele date într-o singură unitate (clasa). Încapsularea ajută la protejarea datelor prin ascunderea lor de accesul neautorizat și expunerea doar a funcțiilor necesare.
- **Moștenire:** Permite unei clase (clasa derivată) să moștenească proprietățile și comportamentul unei alte clase (clasa de bază). Moștenirea promovează reutilizarea codului și stabilirea unor relații ierarhice între clase.
- **Polimorfism:** Permite obiectelor de diferite clase să fie tratate ca obiecte ale unei clase comune de bază. Polimorfismul este realizat prin funcții virtuale, permitând implementări diferite să fie apelate printr-o interfață comună.

Constructorii sunt funcții membre speciale ale unei clase care sunt apelate automat atunci când un obiect al acelei clase este creat. Constructorii au același nume ca și clasa și nu au tip de return. Ei sunt folosiți pentru a inițializa obiectele și pentru a aloca

resursele necesare. Destructorii sunt funcții membre speciale care sunt apelate automat atunci când un obiect este distrus. Destructorii au același nume ca și clasa, dar precedate de un tildă ( ) și nu au parametri. Ei sunt utilizati pentru a elibera resursele alocate de constructori.

Funcțiile virtuale permit implementarea polimorfismului în C++. O funcție virtuală este declarată cu cuvântul cheie `virtual` și poate fi suprascrisă de clasele derivate. Apelurile la funcții virtuale sunt rezolvate la timpul de execuție pe baza tipului obiectului actual. O clasă abstractă este o clasă care conține cel puțin o funcție pur virtuală. O funcție pur virtuală este declarată cu `= 0` și nu are o implementare în clasa de bază, dar obligă clasele derivate să aibă o implementare proprie a acesteia. Clasele abstracte nu pot fi instantiatе și sunt folosite ca interfețe pentru alte clase.

Clasele și obiectele în C++ permit programatorilor să creeze structuri de date complexe și să dezvolte aplicații robuste prin utilizarea principiilor programării orientate pe obiecte.

### 3.3 Dynamic-link library

O bibliotecă de legături dinamice (Dynamic-Link Library, DLL) este o bibliotecă partajată în sistemul de operare Microsoft Windows sau OS/2. Un DLL poate conține cod executabil (funcții), date și resurse, în orice combinație. Un fișier DLL are adesea extensia de fișier .dll, dar dezvoltatorii pot alege să utilizeze o extensie de fișier care descrie conținutul fișierului respectiv, cum ar fi .ocx pentru controale ActiveX și .drv pentru un driver de dispozitiv vechi. Un DLL care conține doar resurse poate fi numit un DLL de resurse. Exemple includ biblioteca de iconițe, uneori având extensia .icl, și biblioteca de fonturi având extensiile .fon și .fon.

Formatul fișierului unui DLL este același ca pentru un executabil (EXE), dar versiunile diferite de Windows folosesc formate diferite. Versiunile de Windows pe 32 și 64 de biți folosesc Portable Executable (PE), iar versiunile de Windows pe 16 biți folosesc New Executable (NE). Principala diferență între DLL și EXE este că un DLL nu poate fi rulat direct, deoarece sistemul de operare necesită un punct de intrare pentru a începe execuția. Windows furnizează un program utilitar (RUNDLL.EXE/RUNDLL/32.EXE) pentru a executa o funcție expusă de un DLL.

Deși tehnologia DLL este esențială pentru arhitectura Windows, ea are dezavantaje. "DLL Hell" descrie comportamentul necorespunzător al unei aplicații atunci când este utilizată versiunea greșită a unui DLL. Alte limitări includ lipsa protecției pentru programul apelant dacă DLL-ul are un bug. Tehnologia DLL permite modificarea unei aplicații fără a necesita recompilarea sau re-legarea componentelor consumatoare. Un DLL poate fi înlocuit astfel încât, la următoarea rulare a aplicației, aceasta să utilizeze noua versiune a DLL-ului. În API-ul Windows, fișierele DLL sunt organizate în secțiuni, fiecare având propriul set de atribută.

Ca și bibliotecile statice, bibliotecile de import pentru DLL-uri sunt notate prin extensia .lib. De exemplu, kernel32.dll este legat prin kernel32.lib. Încărcarea dinamică a unui DLL poate fi realizată la timpul de execuție folosind funcțiile LoadLibrary și GetProcAddress din API-ul Windows. Microsoft Visual C++ furnizează mai multe extensii pentru C++ standard, care permit specificarea funcțiilor ca importate sau exportate direct în codul C++. Aceste extensii folosesc atributul `_declspec` înainte de declarația

unei funcții. Funcțiile importate sau exportate pot fi listate și în secțiunea IMPORT sau EXPORT a fișierului DEF utilizat de proiect.

## 3.4 Testarea unitară

Testarea unitară, cunoscută și ca testarea componentelor sau a modulelor, este o formă de testare a software-ului prin care codul sursă izolat este testat pentru a valida comportamentul așteptat.

Principiul testării unitare, care presupune testarea separată a unor părți mai mici dintr-un sistem software mare, datează din primele zile ale ingineriei software. Unitatea implică, în general, o cantitate relativ mică de cod, cod care poate fi izolat de restul unei baze de coduri. În programarea procedurală, o unitate este de obicei o funcție sau un modul. În programarea orientată pe obiecte, o unitate este de obicei o metodă, un obiect sau o clasă.

Testele unitare pot fi realizate manual sau prin execuție automată. Testele automate includ beneficii precum: rularea testelor frecvent, rularea testelor fără costuri de personal, testare consistentă și repetabilă. Testarea este adesea efectuată de programatorul care scrie și modifică codul testat și poate fi privită ca parte a procesului de scriere a codului.

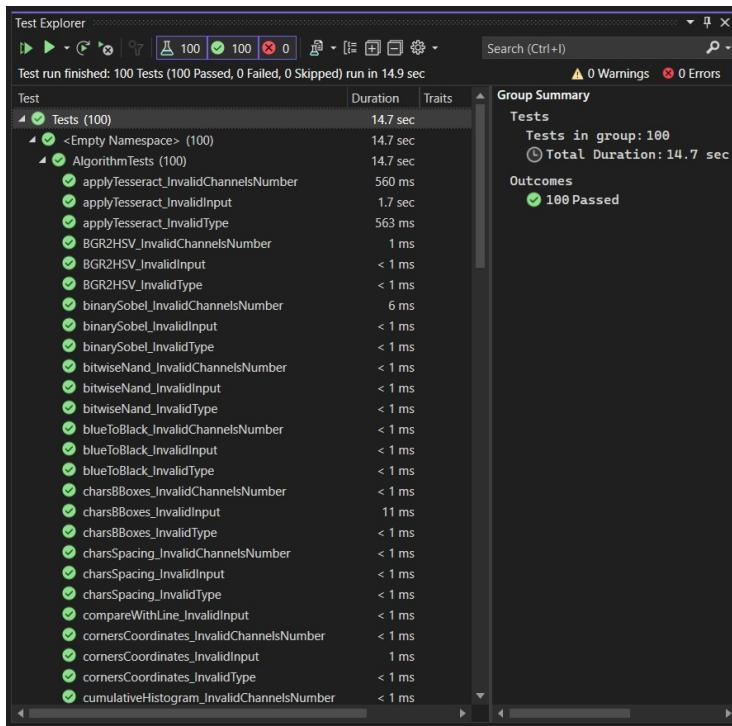


Figura 3.1: Rularea testelor implementate pentru metodele aplicației

În timpul dezvoltării, un programator poate scrie în teste criterii sau rezultate care sunt cunoscute ca fiind bune pentru a verifica corectitudinea unității. În timpul execuției testelor, cadrele de testare înregistrează testele care nu îndeplinesc niciun criteriu și le raportează într-un rezumat. Cea mai utilizată abordare este test - funcție - valoare așteptată. În ingineria software, un caz de testare este o specificație a intrărilor, condițiilor de execuție, procedurii de testare și rezultatelor așteptate care definesc un singur

test ce trebuie executat pentru a atinge un obiectiv specific de testare a software-ului, cum ar fi exercitarea unei anumite căi de program sau verificarea conformității cu o cerință specifică. Cazurile de testare stau la baza testării metodice mai degrabă decât întâmplătoare.

Testarea unitară este destinată să asigure că unitățile își îndeplinesc designul și se comportă aşa cum este intenționat. Unul dintre obiectivele testării unitare este de a izola fiecare parte a programului și de a arăta că părțile individuale sunt corecte. Un test unitar furnizează un contract strict, scris, pe care trebuie să-l satisfacă bucată de cod. Testarea unitară, prin definiție, testează doar funcționalitatea unităților în sine. Prin urmare, nu va detecta erori de integrare sau erori la nivel de sistem mai larg.

Utilizarea testelor unitare ca specificații de design are un avantaj semnificativ față de alte metode de design: documentația de design (testele unitare în sine) poate fi folosită pentru a verifica implementarea. Testele nu vor trece decât dacă dezvoltatorul implementează o soluție conform designului.

Un cadru de testare automatizată oferă funcționalități pentru automatizarea execuției testelor și poate accelera scrierea și rularea testelor. Cadrele de testare sunt de obicei terțe și nu sunt distribuite cu un compilator sau mediu de dezvoltare integrat (IDE). Testele pot fi scrise fără a utiliza un cadru pentru a exercita codul testat utilizând aserții, tratarea exceptiilor și alte mecanisme de control al fluxului pentru a verifica comportamentul și a raporta eșecul.

## 3.5 Visual Studio

Un mediu de dezvoltare integrat (IDE) este o aplicație software care oferă facilități complete pentru dezvoltarea software-ului. Un IDE constă, în mod normal, cel puțin dintr-un editor de cod sursă, unelte de automatizare a compilării și un depanator. Unele medii de dezvoltare integrat conțin chiar și un sistem de control al versiunilor sau diverse unelte pentru simplificarea construirii unei interfețe grafice pentru utilizator (GUI). Multe IDE-uri moderne au, de asemenea, un browser de clase, un browser de obiecte și o diagramă a ierarhiei claselor pentru utilizarea în dezvoltarea software-ului orientat pe obiecte.

IDE-urile sunt proiectate pentru a maximiza productivitatea programatorilor prin oferirea de componente bine integrate cu interfețe de utilizator similare. Acestea prezintă un singur program în care se realizează toată dezvoltarea și oferă de obicei multe funcții pentru scrierea, modificarea, compilarea, implementarea și depanarea software-ului.

Unul dintre obiectivele IDE-ului este reducerea configurației necesare pentru a combina mai multe unelte de dezvoltare. În schimb, oferă același set de capabilități ca o unitate coerentă. Reducerea timpului de configurare poate crește productivitatea dezvoltatorului. Integrarea strânsă a tuturor sarcinilor de dezvoltare are potențialul de a îmbunătăți productivitatea generală, dincolo de ajutorul oferit în sarcinile de configurare. De exemplu, codul poate fi analizat continuu în timp ce este editat, oferind feedback instantaneu atunci când sunt introduse erori de sintaxă, permitând astfel dezvoltatorilor să depaneze codul mult mai rapid și mai ușor cu un IDE.

Printre funcționalitățile principale oferite de mediile de dezvoltare se numără:

- Sublinierea sintaxei, evidențierănd structurile, cuvintele cheie ale limbajului și erorile de sintaxă cu culori și efecte de font distincte vizual.
- Completarea codului, menit să accelereze programarea. IDE-urile moderne au chiar și completare intelligentă a codului.
- Suport pentru refactorizare automată.
- Opțiunea de versionare a proiectelor prin care se poate interacționa cu depozitele de cod sursă.
- Depanarea, folosind un depanator integrat, cu suport pentru setarea punctelor de intrerupere în editor și redarea vizuală a pașilor.

In dezvoltarea aplicației s-a folosit Visual Studio, care este un mediu de dezvoltare integrat dezvoltat de Microsoft. Acesta este utilizat pentru dezvoltarea programelor de calculator, inclusiv site-uri web, aplicații web, servicii web și aplicații mobile. Acestea utilizează platformele de dezvoltare software Microsoft, inclusiv Windows API, Windows Forms, Windows Presentation Foundation (WPF), Windows Store și Microsoft Silverlight. Poate produce atât cod nativ, cât și cod gestionat.

Visual Studio include un editor de cod care suportă IntelliSense (componenta de completare a codului) precum și refactorizarea codului. Debugger-ul integrat funcționează atât ca un debugger la nivel de sursă, cât și ca un debugger la nivel de mașină. Alte instrumente integrate includ un profiler de cod, un designer pentru construirea aplicațiilor GUI, un designer web, un designer de clase și un designer de scheme de baze de date. Acceptă plug-in-uri care extind funcționalitatea la aproape orice nivel, inclusiv adăugarea de suport pentru sisteme de control al versiunilor (cum ar fi Subversion și Git) și adăugarea de noi seturi de instrumente precum editoare și designeri vizuali pentru limbi de domeniu specific sau seturi de instrumente pentru alte aspecte ale ciclului de viață al dezvoltării software.

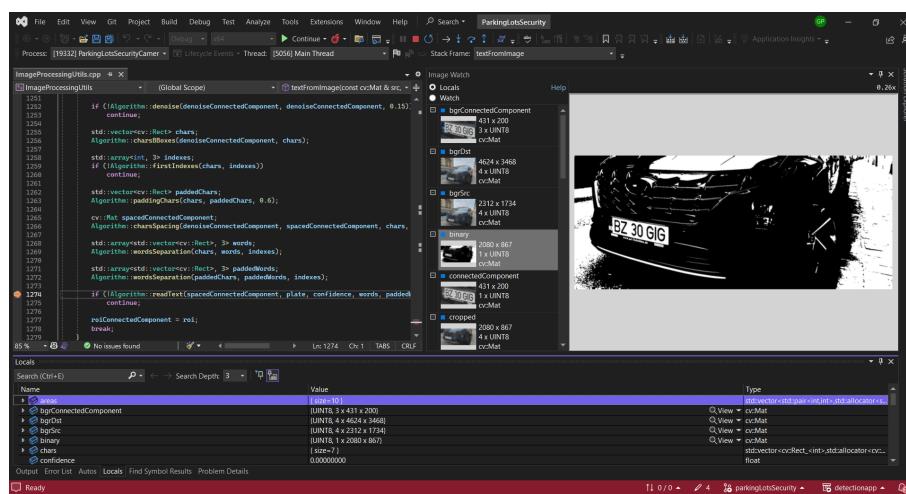


Figura 3.2: Visual Studio in modul de depanare

Visual Studio suportă 36 de limbi de programare diferite și permite editorului de cod și debugger-ului să suporte (în diferite grade) aproape orice limbaj de programare, cu condiția să existe un serviciu specific limbajului respectiv. Limbajele incluse nativ sunt C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML și CSS. Suportul pentru alte limbi precum Python, Ruby, Node.js și M, printre altele, este disponibil prin plug-in-uri. Java (și J#) au fost suportate în trecut.

## 3.6 CMake

În dezvoltarea de software, CMake este un instrument gratuit și open-source, disponibil pe mai multe platforme, utilizat pentru automatizarea, testarea, ambalarea și instalarea software-ului printr-o metodă independentă de compilator. Deși CMake nu este un sistem de compilare propriu-zis, acesta generează fișiere de compilare pentru alte sisteme de compilare.

Principalele caracteristici oferite de CMake sunt:

- **Arbore de construcție separat:** Oferă posibilitatea de a plasa fișierele de ieșire ale compilatorului (cum ar fi fișierele obiect) într-un arbore de construcție care este situat în afara arborelui sursă. Aceasta permite multiple construcții din același arbore sursă și cross-compilare.
- **Gestionarea dependențelor:** CMake păstrează evidența și recomplarea tuturor dependențelor upstream ale unui submodul dat dacă sursele acestuia sunt schimbate.
- **Structură flexibilă a proiectului:** CMake poate localiza executabile, fișiere și biblioteci specificate de utilizator sau sistem-wide. Aceste locații sunt stocate într-o cache, care poate fi apoi personalizată înainte de generarea fișierelor de construcție întă.
- **Suport pentru configurarea IDE:** CMake poate genera fișiere de proiect pentru mai multe IDE-uri populare, cum ar fi Microsoft Visual Studio, Xcode și Eclipse CDT.
- **Detectarea caracteristicilor compilatorului:** Permite specificarea caracteristicilor pe care compilatorul trebuie să le suporte pentru a compila programul sau biblioteca întă.
- **Sistem de pachete:** Deși CMake nu este un manager de pachete, oferă funcții de bază pentru instalarea binarelor și informațiilor de pachet declarate de scriptul CMakeLists.txt pentru a fi folosite de proiectele CMake consumatoare.

Construcția unui program sau a unei biblioteci cu CMake este un proces în două etape. Mai întâi, fișierele de construcție (de obicei scripturi) sunt create (generate) din fișierele de configurare (scripturi CMakeLists.txt) scrise în limbajul CMake. Apoi, uneltele native de construcție ale platformei sunt invocate pentru construcția efectivă a programelor. Fișierele de construcție sunt configurate în funcție de generatorul utilizat (de exemplu, Makefiles pentru make) și fișierele de toolchain asociate. Utilizatorii avansați pot crea și încorpora generatoare suplimentare de makefile pentru a susține nevoile specifice ale compilatorului și ale sistemului de operare.

CMake are un limbaj de scripting relativ simplu, interpretat, imperativ. Suportă variabile, metode de manipulare a sirurilor, array-uri, declarații de funcții/macro-uri și includerea de module (import). Comenzile limbajului CMake sunt citite de cmake dintr-un fișier numit CMakeLists.txt. Acest fișier specifică fișierele sursă și parametrii de construcție, pe care CMake le va plasa în specificația de construcție a proiectului (cum ar fi un Makefile). CMake vine cu numeroase module .cmake și unelte care facilitează

munca, cum ar fi găsirea dependențelor, testarea mediului toolchain și a executabilelor, pachetizarea versiunilor (modulul CPack și comanda cpack) și gestionarea dependențelor de proiecte externe. CPack este un sistem de pachetizare pentru distribuțiile software. Este strâns integrat cu CMake, dar poate funcționa și fără acesta. CPack poate fi folosit pentru a genera pachete RPM, deb și gzip pentru Linux, fișiere NSIS pentru Microsoft Windows și pachete pentru macOS.

## 3.7 Doxygen

Doxygen este un generator de documentație și un instrument de analiză statică pentru arborii sursă ai software-ului. Atunci când este utilizat ca generator de documentație, Doxygen extrage informații din comentariile formate special din cadrul codului. Atunci când este utilizat pentru analiză, Doxygen folosește arborele său de parsare pentru a genera diagrame și grafice ale structurii codului. Doxygen poate realiza referințe încrucișate între documentație și cod, astfel încât cititorul unui document poate referi ușor codul efectiv.

Limbajele de programare suportate de Doxygen includ C, C++, C#, D, Fortran, IDL, Java, Objective-C, Perl, PHP, Python și VHDL. Alte limbiage pot fi suportate prin adăugarea de cod suplimentar. Are suport încorporat pentru a genera diagrame de moștenire pentru clasele C++. Pentru diagrame și grafice mai avansate, Doxygen poate folosi instrumentul "dot" de la Graphviz.

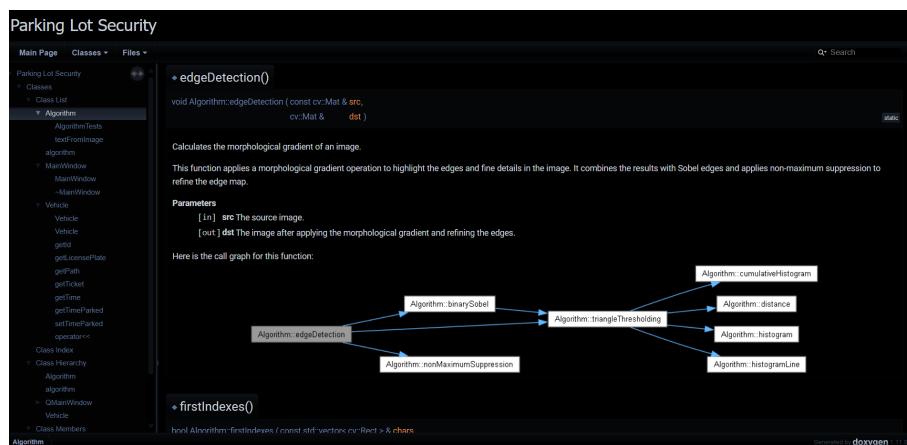


Figura 3.3: Documentația Doxygen a aplicației

Sursele Doxygen sunt găzduite în prezent pe GitHub, sunt scrise în C++ și constau din aproximativ 300.000 de linii de cod sursă. Pentru analiza lexicală, instrumentul standard Lex (sau înlocuitorul său Flex) este rulat prin aproximativ 35.000 de linii de script lex. Instrumentul de parsare Yacc (sau înlocuitorul său Bison) este, de asemenea, utilizat, dar doar pentru sarcini minore, cea mai mare parte a analizei limbajului este realizată de codul C++ nativ. Procesul de construire se bazează pe CMake și implică, de asemenea, unele scripturi Python.

Graphviz (abreviere pentru Graph Visualization Software) este un pachet de instrumente open-source pentru desenarea grafurilor (ca în noduri și muchii, nu ca în grafice de tip bară) specificate în scripturi de limbaj DOT având extensia de fișier "gv".

De asemenea, oferă biblioteci pentru aplicațiile software care utilizează aceste instrumente. Graphviz include o varietate de instrumente precum dot pentru producerea

desenelor grafice stratificate, neato și fdp pentru grafuri nedirectionate, sfdp pentru grafuri nedirectionate mari, twopi pentru layout-uri radiale, circo pentru layout-uri circulare, dotty pentru vizualizarea și editarea grafurilor, și lefty un widget programabil pentru interacțiunea cu grafurile. De asemenea, include utilitare pentru conversia între diferite formate de fișiere grafice, precum GML, GraphML și GXL. Graphviz este un software gratuit licențiat sub Eclipse Public License.

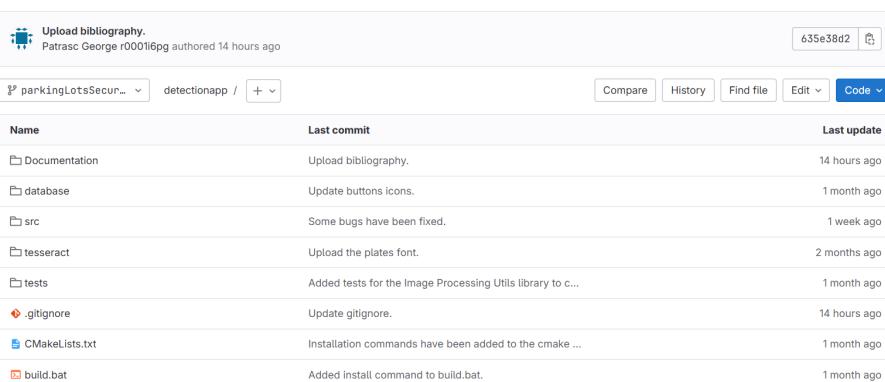
## 3.8 GitLab

Git este un sistem de control al versiunilor distribuit care urmărește versionarea fișierelor. Este adesea folosit pentru a controla codul sursă de către programatorii care dezvoltă software în mod colaborativ.

Obiectivele de design ale Git includ viteza, integritatea datelor și suportul pentru fluxuri de lucru distribuite și non-lineare, cu mii de ramuri paralele care rulează pe calculatoare diferite.

Git menține o copie locală a întregului depozit, cunoscut sub numele de repo, cu abilități de urmărire a istoricului și a versiunilor, independent de accesul la rețea sau un server central. Un repo este stocat pe fiecare computer într-un director standard cu fișiere suplimentare ascunse pentru a oferi capabilități de control al versiunilor. Git oferă funcții pentru sincronizarea schimbărilor între repo-uri care împărtășesc istoricul, copiate (clonate) unul de la celălalt. Pentru colaborare, Git suportă sincronizarea cu repo-uri pe mașini remote. Deși toate repo-urile (cu același istoric) sunt egale, dezvoltatorii folosesc adesea un server central pentru a găzdui un repo pentru a menține o copie integrată.

In dezvoltarea aplicației s-a folosit GitLab, care este un pachet software DevOps care poate dezvolta, securiza și opera software. Proiectul software open-source a fost creat de dezvoltatorul ucrainean Dmytro Zaporozhets și dezvoltatorul olandez Sijbrandij. GitLab permite gestionarea eficientă a codului sursă și colaborarea între membrii echipei. Structura depozitului GitLab a proiectului poate fi observată în figura de mai jos.



The screenshot shows a GitLab repository interface. At the top, there's a message: "Upload bibliography. Patrasc George r001f6pg authored 14 hours ago". To the right is a copy icon. Below the message are navigation buttons: "Compare", "History", "Find file", "Edit", and a "Code" dropdown menu. The main area displays a table of files and their last commits:

Name	Last commit	Last update
Documentation	Upload bibliography.	14 hours ago
database	Update buttons icons.	1 month ago
src	Some bugs have been fixed.	1 week ago
tesseract	Upload the plates font.	2 months ago
tests	Added tests for the Image Processing Utils library to c...	1 month ago
.gitignore	Update gitignore.	14 hours ago
CMakeLists.txt	Installation commands have been added to the cmake ...	1 month ago
build.bat	Added install command to build.bat.	1 month ago

Figura 3.4: Repository

GitLab permite urmărirea detaliată a istoricului modificărilor într-un depozit. Imaginea prezintă un exemplu de istoric al commit-urilor pentru aplicatie. Acest istoric arată activitatea recentă și modificările importante aduse proiectului.

May 26, 2024	
Upload bibliography. Patrasc George r00016pg authored 14 hours ago	635e38d2
Update gitignore. Patrasc George r00016pg authored 14 hours ago	1350bce
The theory chapter was added to the documentation. Patrasc George r00016pg authored 14 hours ago	753ff6ea
May 13, 2024	
Some bugs have been fixed. Patrasc George r00016pg authored 1 week ago	4e08542a
Chapters related to CMake and application architecture have been added to the documentation. Patrasc George r00016pg authored 1 week ago	eeafdf173
Apr 28, 2024	
Update gitignore. Patrasc George r00016pg authored 3 weeks ago	9ba9399f
Added LaTeX Documentation. Patrasc George r00016pg authored 3 weeks ago	684e22ed
Apr 21, 2024	
Update Documentation. Patrasc George r00016pg authored 1 month ago	9e792acd
Apr 19, 2024	
Added button icons to CMakeLists.txt. Patrasc George r00016pg authored 1 month ago	1887a429
Added a poster and a PowerPoint presentation. Patrasc George r00016pg authored 1 month ago	5ff77588
Apr 17, 2024	
Update buttons icons. Patrasc George r00016pg authored 1 month ago	8ddafe76
Added icons for the enter and exit buttons. Patrasc George r00016pg authored 1 month ago	0c2ef41d
Apr 16, 2024	
The paths to the database have been changed depending on the runtime. Patrasc George r00016pg authored 1 month ago	bcd8d4ed
Added install command to build.bat. Patrasc George r00016pg authored 1 month ago	bac14295
Installation commands have been added to the cmake files. Patrasc George r00016pg authored 1 month ago	fc739ae5

Figura 3.5: Istorice commit-uri

## 3.9 OpenCV

OpenCV (Open Source Computer Vision Library) este o bibliotecă de funcții de programare utilizate în principal pentru viziunea computerizată în timp real. Biblioteca este multiplatformă și licențiată ca software liber și open-source. Începând din 2011, OpenCV oferă accelerare GPU pentru operațiuni în timp real.

Proiectul OpenCV a fost lansat oficial în 1999 ca o inițiativă de cercetare pentru a avansa aplicațiile CPU-intensive, parte a unei serii de proiecte care includ ray tracing în timp real și pereți de afișare 3D. Obiectivele inițiale ale proiectului au fost avansarea cercetării în domeniul viziunii prin furnizarea de cod optimizat și deschis pentru infrastructura de bază a viziunii, diseminarea cunoștințelor în domeniul viziunii prin furnizarea unei infrastructuri comune pe care dezvoltatorii să o poată folosi, și avansarea aplicațiilor comerciale bazate pe viziune prin furnizarea de cod optimizat pentru performanță, disponibil gratuit.

OpenCV include și o bibliotecă de învățare automată statistică ce conține algoritmi precum:

- **Boosting**
- **Învățarea arborilor de decizie**
- **Arbore de boosting gradient**
- **Algoritmul de așteptare-maximizare**
- **Algoritmul k-nearest neighbor**

- **Clasificatorul Naive Bayes**
- **Rețele neuronale artificiale**
- **Pădurea aleatoare**
- **Mașină vectorială de suport (SVM)**
- **Rețele neuronale profunde (DNN)**

OpenCV este scris în limbajul de programare C++, la fel ca interfața sa principală, dar păstrează și o interfață mai veche, mai puțin cuprindătoare, în C. Toate dezvoltările și algoritmii noi apar în interfață C++. Există legături de limbaj în Python, Java și MATLAB/Octave. În versiunea 3.4, au fost lansate legături JavaScript pentru un subset selectat de funcții OpenCV, cunoscut sub numele de OpenCV.js, pentru a fi utilizate pe platformele web. Pentru construirea proiectelor se folosește CMake.

Dacă biblioteca găsește Primitivii de Performanță Integrată ai Intel pe sistem, va utiliza aceste rutine optimizate proprietare pentru a se accelera. O interfață bazată pe CUDA pentru unitatea de procesare grafică (GPU) este în dezvoltare din septembrie 2010. O interfață GPU bazată pe OpenCL este în dezvoltare din octombrie 2012.

## 3.10 Tesseract OCR

Recunoașterea optică a caracterelor sau cititorul optic de caractere (OCR) este conversia electronică sau mecanică a imaginilor de text dactilografiat, scris de mână sau tipărit în text codificat digital, fie dintr-un document scanat, o fotografie a unui document, o fotografie de scenă (de exemplu, textul de pe semne și panouri publicitare într-o fotografie peisagistică) sau din textul subtitrat suprapus pe o imagine.

Folosită pe scară largă ca metodă de introducere a datelor din înregistrările pe hârtie tipărită, fie documente de pașaport, facturi, extrase bancare, chitanțe computerizate, cărți de vizită, corespondență, date tipărite sau orice documentație adecvată, este o metodă comună de digitizare a textelor tipărite pentru a putea fi editate electronic, căutate, stocate mai compact, afișate online și utilizate în procesele mecanizate, cum ar fi calculul cognitiv, traducerea automată, text-to-speech (text extras) și extragerea datelor esențiale. OCR este un domeniu de cercetare în recunoașterea modelelor, inteligență artificială și viziunea computerizată.

Recunoașterea optică are ca principale domenii:

- **Recunoaștere optică a caracterelor (OCR):** țintește textul tipărit, un glif sau caracter la un moment dat.
- **Recunoaștere optică a cuvintelor:** țintește textul tipărit, un cuvânt la un moment dat. De obicei, este denumit simplu "OCR".
- **Recunoaștere inteligentă a caracterelor (ICR):** țintește textul scris de mână, printscript sau cursiv, un glif sau caracter la un moment dat, de obicei implicând învățarea automată.
- **Recunoaștere inteligentă a cuvintelor (IWR):** țintește textul scris de mână, printscript sau cursiv, un cuvânt la un moment dat.

Software-ul OCR preprocesează adesea imaginile pentru a îmbunătăți șansele de recunoaștere cu succes. Tehnicile includ:

- **Rectificare:** dacă documentul nu a fost aliniat corect la scanare, acesta poate necesita o înclinare câteva grade pentru a face liniile de text perfect orizontale sau verticale.
- **Eliminarea zgomotelor:** îndepărțarea punctelor pozitive și negative, netezirea marginilor.
- **Binarizare:** conversia unei imagini din color sau gri în alb-negru (denumită imagine binară deoarece există două culori).
- **Eliminarea liniilor:** curățarea casetelor și liniilor non-glif.
- **Analiza layout-ului:** identificarea coloanelor, paragrafelor, legendelor etc. ca blocuri distințe.
- **Detectarea liniilor și cuvintelor:** stabilirea unei linii de bază pentru formele cuvintelor și caracterelor, separarea cuvintelor după necesitate.
- **Recunoașterea scripturilor:** în documentele multilingve, scripturile pot schimba la nivel de cuvinte, astfel identificarea scriptului este necesară înainte de a invoca OCR-ul corect pentru scriptul specific.
- **Izolarea caracterelor:** pentru OCR per-caracter, multiple caractere conectate din cauza artefactelor de imagine trebuie separate.
- **Normalizarea:** raportului de aspect și a scalei.

Există două tipuri de bază de algoritmi OCR: potrivirea matricială, care implică compararea unei imagini cu un glif stocat pe bază de pixeli, cunoscută și sub numele de potrivire de pattern, recunoaștere de pattern sau corelare de imagini. Al doilea tip, extracția caracteristicilor, descompune glifurile în "caracteristici" precum linii, bucle închise, direcția liniilor și intersecțiile liniilor.

Acuratețea OCR poate fi crescută dacă ieșirea este constrânsă de un lexicon, o listă de cuvinte care sunt permise să apară într-un document. Acest lucru poate fi problematic dacă documentul conține cuvinte care nu sunt în lexicon, cum ar fi numele proprii. Furnizorii principali de tehnologie OCR au început să ajusteze sistemele OCR pentru a se ocupa mai eficient de tipuri specifice de intrări. Această strategie este cunoscută sub numele de "OCR orientat pe aplicații" sau "OCR personalizat".

Tesseract OCR este un motor de recunoaștere optică a caracterelor pentru diverse sisteme de operare. Acesta este modelul AI folosit în aplicație pentru citerea numerelor de înmatriculare, care a și fost antrenat pe baza font-ului DIN1451Mittelschrift folosit de numerele de înmatriculare românești.

Motorul Tesseract este un software scris inițial în limbajul de programare C, care mai apoi a suferit o conversie în C++. De atunci, tot codul a fost convertit pentru a fi compilat cu un compilator C++. Versiunea 4 a adăugat un motor OCR bazat pe LSTM și modele pentru multe limbi și scripturi suplimentare, aducând totalul la 116 limbi. De asemenea, 37 de scripturi sunt suportate. Este posibil, de exemplu, să recunoască text cu un mix de limbi europene occidentale și centrale folosind modelul pentru scriptul latin în care este scris.

Până la versiunea 2 inclusiv, Tesseract putea accepta doar imagini TIFF de text simplu pe o singură coloană ca intrări. Aceste versiuni timpurii nu includeau analiza layout-ului, astfel încât introducerea textului pe mai multe coloane, imagini sau ecuații producea rezultate neclare. Începând cu versiunea 3.00, s-a suportat formatarea textului de ieșire, informațiile poziționale hOCR și analiza layout-ului paginii. Suportul pentru un număr de noi formate de imagine a fost adăugat folosind biblioteca Leptonica, bibliotecă dezvoltată pentru a oferi suport în procesarea și analiza imaginilor. Tesseract poate detecta, de asemenea, dacă textul este monospațiat sau proporțional spațiat.

Versiunile inițiale puteau recunoaște doar textul în limba engleză, însă de la versiunea a doua s-au adăugat șase limbi occidentale suplimentare (franceză, italiană, germană, spaniolă, portugheză braziliană, olandeză). Versiunea 3 a extins semnificativ suportul pentru limbi pentru a include limbi ideografice (chineză și japoneză) și limbi citite de la dreapta la stânga (de exemplu, arabă, ebraică), precum și multe alte scripturi. Modelul poate fi antrenat să funcționeze în alte limbi și poate procesa text de la dreapta la stânga, cum ar fi arabă sau ebraică, multe scripturi indiene, precum și CJK destul de bine.

### 3.11 Qt

Qt este utilizat pentru dezvoltarea interfețelor grafice cu utilizatorul (GUI) și a aplicațiilor multi-platformă care rulează pe toate platformele desktop majore și pe platformele mobile sau embedded. Majoritatea programelor GUI create cu Qt au o interfață care pare nativă, în acest caz Qt fiind clasificat ca un toolkit de widget-uri. Pot fi dezvoltate și programe non-GUI, cum ar fi unelte de linie de comandă și console pentru servere. Un exemplu de astfel de program non-GUI care utilizează Qt este framework-ul web Cutelyst.

Qt suportă diverse compilatoare C++, inclusiv GCC și Clang, precum și suitele Visual Studio. Suportă alte limbaje prin legături sau extensii, cum ar fi Python prin legăturile Python și PHP printr-o extensie pentru PHP5, și oferă suport extins pentru internaționalizare. Qt oferă și Qt Quick, care include un limbaj de scripting declarativ numit QML ce permite utilizarea JavaScript pentru a oferi logica. Cu Qt Quick, dezvoltarea rapidă a aplicațiilor pentru dispozitive mobile a devenit posibilă, în timp ce logica poate fi scrisă și în cod nativ pentru a obține cea mai bună performanță posibilă.

Alte caracteristici includ accesul la baze de date SQL, parsarea XML, parsarea JSON, gestionarea firelor de execuție și suportul pentru rețea.

Qt este construit pe baza acestor concepte cheie:

- **Abstracție completă a GUI-ului:** Folosește API-urile native de stil ale diferitelor platforme pentru a interoga metrii și a desena majoritatea controalelor, reducând astfel discrepanțele între platforme.
- **Semnale și sloturi:** Un construct de limbaj introdus în Qt pentru comunicarea între obiecte, facilitând implementarea pattern-ului observer și evitând codul redundant.
- **Compilator de metaobiecte (moc):** Un instrument care interpretează anumite macro-uri din codul C++ ca adnotări și generează cod C++ suplimentar cu meta-informări despre clasele utilizate în program.

- **Legături de limbaj:** Qt poate fi utilizat în diverse limbi de programare, cum ar fi Python, JavaScript, C# și Rust prin intermediul legăturilor de limbaj.

Framework-ul este alcătuit din mai multe module esențiale și module suplimentare:

- **Qt Core:** Singurul modul obligatoriu al Qt, conține clase utilizate de alte module, inclusiv sistemul de meta-obiecte, concurență și threading, containere, sistemul de evenimente, plugin-uri și facilități de I/O.
- **Qt GUI:** Modul central GUI. În Qt 5, acest modul depinde acum de OpenGL.
- **Qt Widgets:** Conține clase pentru aplicații GUI bazate pe widget-uri clasice.
- **Qt QML:** Modul pentru limbajele QML și JavaScript.
- **Qt Quick:** Modul pentru aplicații GUI scrise folosind QML2.
- **Qt Network:** Strat de abstractizare a rețelei.
- **Qt Multimedia:** Clase pentru funcționalități audio, video, radio și cameră.
- **Qt SQL:** Conține clase pentru integrarea bazelor de date folosind SQL.
- **Qt Test:** Clase pentru testarea unitară a aplicațiilor și bibliotecilor Qt.
- **Qt Charts:** Oferă funcționalități și widget-uri pentru a crea grafice.
- **Qt WebChannel:** Oferă acces la obiectele Qt pentru HTML/Javascript prin WebSockets.
- **Qt WebSockets:** Oferă o implementare WebSocket.

Qt vine cu propriul set de unelte pentru a facilita dezvoltarea cross-platform, care poate fi altfel complicată din cauza diferențelor seturi de unelte de dezvoltare. Qt Creator este un IDE cross-platform pentru C++ și QML. Funcționalitatea de layout/design GUI a Qt Designer este integrată în IDE, deși Qt Designer poate fi încă pornit ca o unealtă independentă. În plus față de Qt Creator, Qt oferă qmake, o unealtă cross-platform de generare a scripturilor de construcție care automatizează generarea Makefile-urilor pentru proiectele de dezvoltare pe diferite platforme. Alte unelte disponibile în Qt includ constructorul de interfețe Qt Designer și browserul de ajutor Qt Assistant (ambele fiind incorporate în Qt Creator), unealta de traducere Qt Linguist, uic (compiler de interfață cu utilizatorul) și moc (Meta-Object Compiler).

# **Capitolul 4**

## **Prezentarea aplicației**

### **4.1 Arhitectura**

Vom începe prin a detalia arhitectura modulară a aplicației, accentuând modul în care fiecare componentă specifică aduce un aport semnificativ la funcționalitatea de ansamblu. Această arhitectură este concepută pentru a sprijini extensibilitatea, ușurința în mențenanță și posibilitatea de a testa independent fiecare modul, facilitând astfel adaptarea și scalarea aplicației în funcție de necesități.

Intrarea în aplicație este gestionată de proiectul Application, care cuprinde funcția main. Aceasta are rolul de a inițializa și lansa interfața grafice, fiind punctul de interacție primar pentru utilizatori. Structura simplificată a funcției main este proiectată pentru a oferi un punct de start clar și eficient, minimizând dependențele inițiale și asigurând o încărcare rapidă și neîntreruptă a aplicației.

Interfața de utilizator a aplicației este orchestrată prin intermediul proiectului UserInterface, care este subordonat proiectului principal Application. Acest modul este esențial pentru gestionarea interacțiunii cu utilizatorul și include toate elementele de GUI necesare pentru o navigare intuitivă și eficientă în cadrul aplicației.

Proiectul UserInterface cuprinde, în principal, două componente cheie: clasa Vehicle și clasa MainWindow. Clasa Vehicle are rolul de a administra datele și operațiunile asociate cu vehiculele procesate de aplicație. Aceasta funcționează ca o abstracție a datelor vehiculului, facilitând manipularea și accesul la informațiile relevante într-un mod organizat și eficient.

Pe de altă parte, clasa MainWindow reprezintă fereastra principală a aplicației și este responsabilă pentru gestionarea layout-ului și a elementelor UI principale. Aceasta servește ca punct central al interacțiunii utilizatorului, integrând diverse controale și afișând informații într-un mod accesibil și prietenos. MainWindow utilizează instanțe ale clasei Vehicle pentru a popula interfața cu date specifice fiecărui vehicul, permitând utilizatorilor să vizualizeze și să interacționeze cu aceste detalii într-un context relevant. Utilizarea clasei Vehicle în cadrul MainWindow asigură o separare clară între logica de prezentare și logica din spate a aplicației.

Procesarea imaginilor este un aspect central al aplicației și este gestionată de modulul ImageProcessingUtils, implementat ca un DLL (Dynamic-Link Library). Acest modul este dedicat procesării vizuale, având ca scop principal recunoașterea numerelor de înmatriculare din imagini, funcționalitatea principală a aplicației.

Modulul ImageProcessingUtils este integrat în proiectul UserInterface, contribuind direct la extragerea și furnizarea detaliilor legate de vehicule. Acesta include clasa Algorithm, care încapsulează toți algoritmii și funcțiile necesare pentru suportarea procesului de recunoaștere a textului. Funcțiile din această clasă sunt esențiale pentru interpretarea și transformarea imaginilor în date textuale utilizabile, facilitând astfel recunoașterea și verificarea numerelor de înmatriculare.

Metodele din clasa Algorithm sunt folosite intr-o funcție externă clasei, care este apelată în clasa MainWindow, și care are ca scop chiar funcționalitatea principală a aplicației, returnarea textului numerelor de înmatriculare. Această funcție este declarată ca fiind „friend” (prietenă) pentru a putea apela metodele clasei Algorithm și pentru a simplifica procesul de recunoaștere a textului din imagini fără a compromite encapsularea și securitatea datelor procesate.

De asemenea, există și proiectul separat de teste, situat în afara directorului sursă, dedicat verificării funcționalității modulului ImageProcessingUtils. Testele unitare din acest proiect sunt esențiale pentru asigurarea calității și fiabilității algoritmilor implementați în clasa Algorithm. Aceste teste se concentrează pe evaluarea fiecărei componente software pentru a garantat că recunoașterea numerelor de înmatriculare se efectuează corect și eficient.

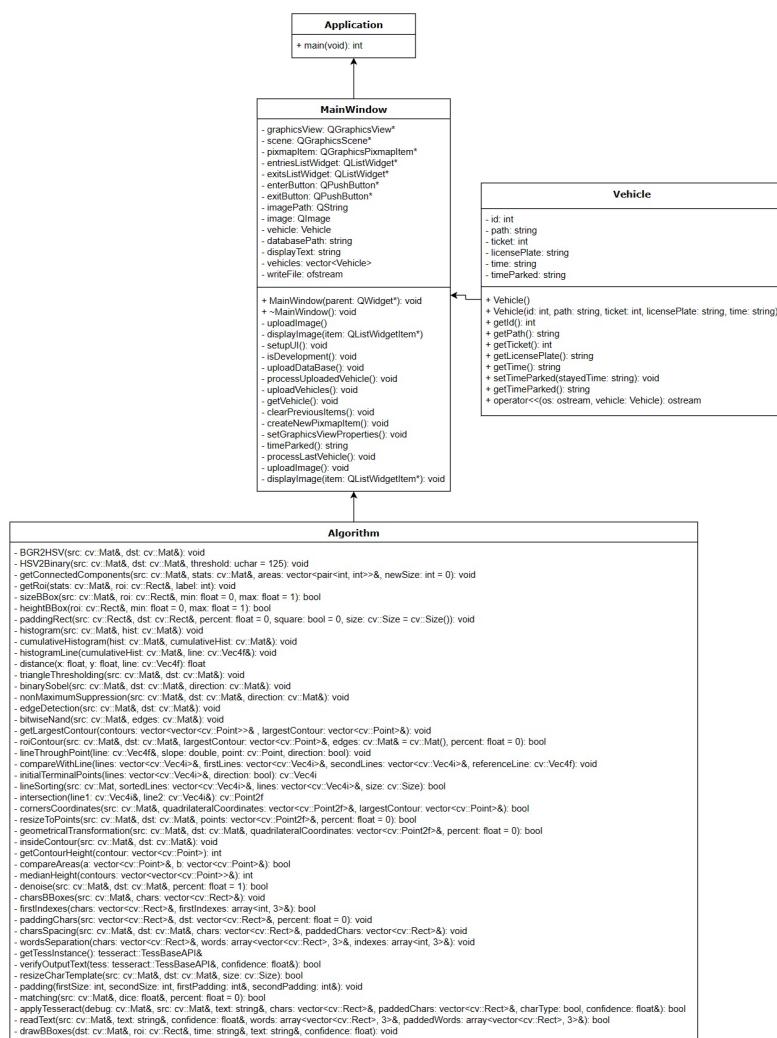


Figura 4.1: Diagrama UML de clase

## 4.2 Segmentarea, rectificarea și recunoașterea textului

În continuare, dorim să aprofundăm pașii de execuție ai programului Toate aceste funcții se regăsesc într-o clasă denumită "Algorithm", din interiorul unei biblioteci dinamice "ImageProcessingUtils". În paralel, vom lua în considerare și o imagine drept exemplu pentru a ilustra fiecare etapă, oferind o perspectivă mai clară și mai cuprinzătoare asupra procesului de funcționare a aplicației.



Figura 4.2: Imaginea de input

Pentru început, să analizăm imaginea de input. Un aspect cheie pentru program este poziționarea camerei de fotografiat, care influențează în mod direct încadrarea numărului de înmatriculare în imagine. În fotografie dată, precum și în setul de date pe care a fost testat programul, numărul de înmatriculare este situat în partea inferioară a imaginii, deoarece acesta este amplasat în general în partea de jos a vehiculelor.

Unghiul de fotografiere și distanța de la care se face poza nu influențează semnificativ procesul de segmentare, cel puțin în anumite limite. Deși, în contextul parcărilor auto, camerele de fotografiat de la intrare sunt fixate astfel încât să se obțină imagini clare din unghiuri optime, în acest exemplu, imaginea este surprinsă într-o manieră neobișnuită pentru a înțelege mai bine impactul anumitor algoritmi folosiți asupra imaginii. Acest lucru permite examinarea mai detaliată a modului în care algoritmii programului gestionează imagini cu condiții diverse de captare și încadrate atipic.

Dimensiunile imaginilor reprezintă un aspect esențial pentru performanța programului. În cazul setului de date pe care a fost testată aplicația, lucrăm cu imagini de înaltă calitate, cu o rezoluție de 4624 x 3468 x 3 și un raport de aspect de 4:3. Această dimensiune generează un număr mare de pixeli, peste 48 de milioane, ceea ce are un impact direct asupra timpului de execuție al aplicației.

Având imagini de aceste dimensiuni, este de așteptat să întâmpinăm provocări legate de viteză și eficiență. O soluție practică este redimensionarea imaginii de input la jumătate, ceea ce reduce numărul total de pixeli, menținând totuși o calitate superioară. Prin acest proces, timpul de execuție este îmbunătățit semnificativ, facilitând o procesare mai rapidă și eficientă, fără a sacrificia prea mult din calitatea imaginii. În acest fel, programul poate gestiona seturi de date mari într-un timp mai scurt, permitându-i să fie mai scalabil și să suporte un volum mai mare de procesare fără a compromite rezultatele.

Pentru a îmbunătăți timpul de execuție și mai mult, putem optimiza procesarea imaginilor concentrându-ne doar pe jumătatea inferioară a acestora. Știind că numărul de înmatriculare este de obicei amplasat în partea inferioară a vehiculului, putem decupa

imagină astfel încât să păstrăm doar jumătatea de jos a acesteia. Eventual, se pot aplica mici decupaje și din părțile laterale pentru a elimina porțiuni care nu conțin informații relevante.



Figura 4.3: Partea inferioară din imaginea de input

Această decizie are un impact direct asupra numărului de pixeli pe care algoritmul trebuie să-i proceseze, reducându-l considerabil. Cu mai puțini pixeli de analizat, timpul de execuție va fi mai rapid, iar aplicația va fi mai eficientă în procesarea imaginilor. În plus, această strategie poate ajuta la focalizarea algoritmilor pe zonele relevante ale imaginii, eliminând părțile neesentiale și reducând riscul de confuzie sau erori.

După ce imaginea a fost decupată, următorul pas este aplicarea unui filtru Gaussian pentru a elimina zgomotul generat de factori externi. Zgomotul din imagini poate apărea din diverse motive, cum ar fi condițiile de iluminare sau interferențele electronice. Filtrul Gaussian este un instrument eficient pentru reducerea zgomotului și pentru netezirea imaginii, fără a afecta contururile în mod semnificativ.



Figura 4.4: Eliminarea zgomotului

Cât despre mască, am considerat că una de dimensiune  $3 \times 3$  este suficientă pentru imagine, însă nu afectează considerabil contururile aşa cum ar face o mască de dimensiuni mai mari, dar totodată elimină o suficientă parte din zgomotul imaginii.

Prin aplicarea acestui filtru, imaginea decupată devine mai curată și mai ușor de procesat de algoritmi, ceea ce contribuie la rezultate mai precise. Contururile rămân clare, facilitând identificarea obiectelor sau a detaliilor importante, în timp ce zgomotul nedorit este redus semnificativ.

În următoarea etapă a procesului de prelucrare a imaginii, un pas esențial în pipeline-ul programului este binarizarea. Scopul este separarea numărului de înmatriculare de restul imaginii, iar una dintre caracteristicile cheie ale unui număr de înmatriculare este culoarea sa, care, în cazul celor din România, este albă.

Imaginile sunt de obicei reprezentate în spațiul de culoare BGR (albastru, verde, roșu), în care culoarea albă are un tuplu de valori de 255 pentru fiecare canal. Totuși, în imagini naturale, factorii externi precum vremea și lumina pot afecta culorile, astfel încât este improbabil să întâlnim albul perfect (255, 255, 255). Pentru a face procesul de binarizare mai robust, putem converti imaginea din BGR în spațiul de culoare HSV (nuanță, saturatie, valoare). În acest spațiu, un obiect alb are o saturatie scăzută și o valoare ridicată, în timp ce nuanța nu este un factor semnificativ.



Figura 4.5: Hue (culoare)



Figura 4.6: Saturație



Figura 4.7: Value (luminositate)

Cu imaginea convertită în spațiul HSV, putem parcurge fiecare pixel și aplica următoarea condiție: dacă pixelul are o saturație scăzută (al doilea canal) și o valoare ridicată (al treilea canal), acesta este considerat ca fiind parte a unui obiect alb, cum ar fi numărul de înmatriculare. În acest caz, pixelul este transformat în alb (255). Dacă nu îndeplinește aceste criterii, este transformat în negru (0). Această abordare creează o imagine binarizată, în care numărul de înmatriculare se va afla într-o regiune albă, față de fundalul imaginii, care va fi negru.



Figura 4.8: Imaginea de input cu filtru binar

Următorul pas în procesul de analiză a imaginii binarizate este extragerea tuturor componentelor conexe. O componentă conexă este o mulțime de pixeli albi învecinați și conectați între ei, înconjurați la exterior de pixeli negri, cum este și cazul numărului de înmatriculare din imaginea binară. În funcție de valorile stabilite anterior ca praguri la pasul de binarizare, imaginea rezultată va avea mai multe sau mai puține componente conexe. Cu toate acestea, de cele mai multe ori, nu va conține doar componenta căutată, adică numărul de înmatriculare, ci probabil și alte componente conexe nedeterminate.



Figura 4.9: Conturul componentelor conexe

Pentru a reduce din numărul componentelor conexe, putem utiliza o metodă de filtrare bazată pe dimensiunea ariei. Numărul de înmatriculare, fiind un element semnificativ în imagine, trebuie să aibă o arie mai mare în comparație cu alte componente

conexe. Prin urmare, putem ordona descrescător toate componentele conexe după dimensiune și să păstrăm doar primele  $n$ , unde  $n$  este stabilit în funcție de contextul imaginii. În acest exemplu, alegem  $n = 10$ , ceea ce înseamnă că rezultatul va fi un vector de dimensiune 10 care conține coordonatele colțului din stânga sus ale fiecărei componente conexe, precum și înălțimea și lățimea fiecărei componente.



Figura 4.10: Conturul celor mai mari 10 componente conexe

Pentru a identifica componenta conexă ce conține numărul de înmatriculare, vom face segmentări din imagine, numite și regiuni de interes (ROI), pe baza vectorului rezultat, și vom analiza fiecare regiune obținută, care încadrează câte o componentă conexă. Știm că un număr de înmatriculare are o formă dreptunghiulară, cu o înălțime mică în comparație cu lățimea și cu o arie destul de mică în raport cu întreaga imagine. Pe baza acestor informații, putem stabili condiții pentru a filtra componentele conexe și a ne concentra pe cea care ar putea corespunde unui număr de înmatriculare.

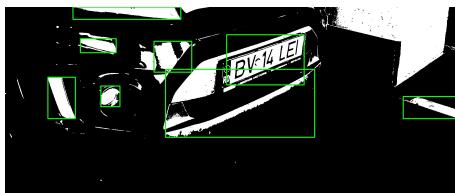


Figura 4.11: Bounding box-urile componentelor conexe

O condiție importantă pentru selecție este raportul dintre înălțimea și lățimea regiunii de interes. Pentru a se califica ca potențial candidat pentru numărul de înmatriculare, înălțimea segmentării trebuie să fie mai mare decât un anumit procent din lățimea acesteia, dar nu mai mare decât un alt procent. De exemplu, putem stabili că înălțimea trebuie să fie între 20% și 90% din lățime, permitând o flexibilitate suficientă pentru a include numerele de înmatriculare care ar putea fi înclinate, rezultând forme aproape pătratice.

$$ROI.height \in [ROI.width \times min, ROI.width \times max]; min, max \in [0, 1]$$

Dacă această condiție este îndeplinită, următorul pas este calcularea ariei regiunii de interes și compararea acesteia cu aria totală a imaginii. Pentru a se califica, aria trebuie să fie între anumite limite procentuale, de exemplu, între 1% și 15% din aria totală.

$$ROI.area \in [Sursa.area \times min, Sursa.area \times max]; min, max \in [0, 1]$$

Acest proces de filtrare bazat pe aspectul dreptunghiular și dimensiunea ariei ajută la reducerea numărului de componente conexe care nu reprezintă numere de înmatriculare, asigurând că ne concentrăm pe candidații cei mai probabili. În plus, flexibilitatea

condițiilor permite ajustări în funcție de imagini cu numere de înmatriculare în poziții variabile sau înclinate.



Figura 4.12: Segmentarea din imaginea de input

Acum că regiunea de interes (ROI) pare să aibă dimensiuni similare cu un număr de înmatriculare, putem aplica diferenți algoritmi pentru a detecta contururile, care vor fi folosiți în pașii următori ai programului. Începem prin adăugarea unui padding la regiunea noastră de interes pentru a asigura continuitatea contururilor, evitând astfel intreruperi cauzate de dimensiunile măștilor utilizate pentru detectarea marginilor. Padding-ul este important pentru a ne asigura că niciun pixel relevant nu este omis în zonele limită.



Figura 4.13: Imaginea segmentată cu padding adăugat

După aceasta, convertim segmentarea în format grayscale, deoarece algoritmii de detectare a marginilor necesită imagini în tonuri de gri.



Figura 4.14: Imaginea segmentată în format grayscale

Primul pas este aplicarea filtrului Sobel, care detectează marginile bazându-se pe derivata imaginii. Pentru a obține o binarizare clară, aplicăm un filtru de binarizare Triangular Thresholding pentru a păstra doar pixelii relevanți detectați de Sobel, care sunt cu adevărat contururi.



Figura 4.15: Filtru Sobel

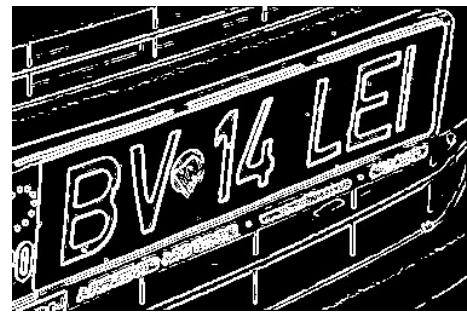


Figura 4.16: Filtru Sobel binarizat cu metoda Triangle Thresholding

Pentru a ne asigura că am captat toți pixelii relevanti din imagine, aplicăm și un filtru de obținere a gradientului morfologic tot pe regiunea în format grayscale, urmat de încă un filtru de binarizare Triangular Thresholding. Astfel, avem două imagini binare: una cu rezultatele filtrului Sobel și alta cu gradientul morfologic.



Figura 4.17: Gradientii morfologici



Figura 4.18: Gradientii morfologici binarizati cu metoda Triangle Thresholding

Următorul pas este combinarea acestor două imagini binare, păstrând în imaginea cu gradientii morfologici doar acei pixeli care apar în cel puțin una dintre imaginile binare.



Figura 4.19: Reuniunea celor două imagini binare (masca)



Figura 4.20: Gradientii morfologici după mască

Acum, că avem doar pixelii care ar putea apartine unor contururi, aplicăm un filtru de subțiere Non-Maximum Suppression pentru a obține margini de un pixel grosime, astfel încât contururile să fie clare și precise.



Figura 4.21: Edge map de un pixel grosime

În următorul pas, revenim la regiunea de interes în formatul grayscale și aplicăm un filtru de binarizare automatizat, cum ar fi Triangle Thresholding sau Otsu. Segmentarea este un dreptunghi în care ar trebui să fie inclus numărul de înmatriculare (dacă este regiunea corectă), dar în acesta pot exista și alte elemente din imaginea inițială care nu ne interesează.



Figura 4.22: Imaginea segmentată cu filtru binar

După aplicarea filtrului de binarizare, în imagine ar trebui observat un chenar alb care conține literele și cifrele numărului de înmatriculare și care ar trebui să fie separat de restul imaginii. Acum putem căuta din nou componentele conexe și păstra strict pe cea de dimensiunea cea mai mare, care ar trebui să fie chenarul alb. Însă pot apărea situații în care această binarizare este eronată și chenarul alb este conectat cu alte zone formate din pixeli albi.

Pentru a corecta acest lucru, folosim edge map-ul obținut anterior. Știm că în edge map contururile sunt reprezentate ca pixeli de valoare 255, deci putem verifica că pixelii albi (255) din edge map sunt negri (0) în regiunea binarizată. Acest lucru asigură că pixelii contururilor rămân separați și nu conectează eronat zonele între ele.



Figura 4.23: Corecția imaginii segmentate

Pentru a fi complet siguri că nu există legături incorecte, putem aplica o operație

morfologică de tip Opening. Această operație separă obiectele legate incorect prin linii fine.

După operația de Opening, este posibil să apară noi componente conexe, deci aplicăm din nou căutarea componentelor conexe și păstrăm doar componenta de dimensiunea cea mai mare. Aceasta ar trebui să fie chenarul alb care conține numărul de înmatriculare.

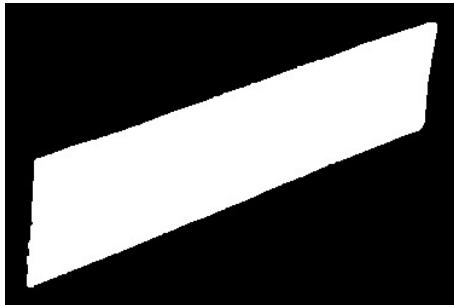


Figura 4.24: Regiunea textului

Pentru a reduce numărul de segmente și pentru a ne apropiă de regiunea care conține numărul de înmatriculare, putem adăuga o nouă condiție pentru imaginea segmentată, care ar trebui să conțină numai zona textului numărului de înmatriculare. Această condiție stabilește că înălțimea zonei respective ar trebui să fie aproape egală cu întreaga înălțime a imaginii de segmentare. Prin urmare, putem compara înălțimea acestei zone cu un procentaj, să zicem 80%, din înălțimea regiunii de interes pentru a valida corectitudinea segmentării.

$$width > ROI.width \times percent; \quad percent \in [0, 1]$$

În pasul de rectificare, trebuie să obținem 4 coordonate pe baza cărora se va face rectificarea, aceste 4 coordonate fiind chiar colțurile chenarului alb în care se află numărul de înmatriculare. Pentru început, scădem din imaginea sursă imaginea sursă erodată pentru a obține exact conturul chenarului alb. Pe imaginea rezultată putem aplica algoritmul Hough, care ne va returna liniile din imagine, din intersectarea cărora putem obține coordonatele colțurilor.

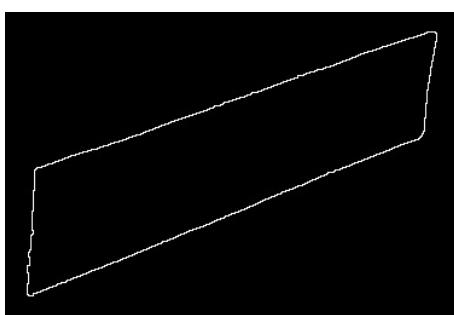


Figura 4.25: Conturul regiunii

Pentru a face acest lucru, primul pas este să stabilim parametrii funcției Hough. Doi parametri importanți din funcția Hough sunt lungimea minimă admisă pentru o linie ca aceasta să fie adăugată ca rezultat și distanța maximă dintre două linii ca acestea să fie considerate de fapt aceeași linie, dar cu o întrerupere eronată între ele, și astfel în vectorul rezultat, în loc de două linii, să fie adăugată una singură formată din aceste

două. Valorile acestor doi parametri sunt obținute în mod automatizat pe baza bounding box-ului rotit.

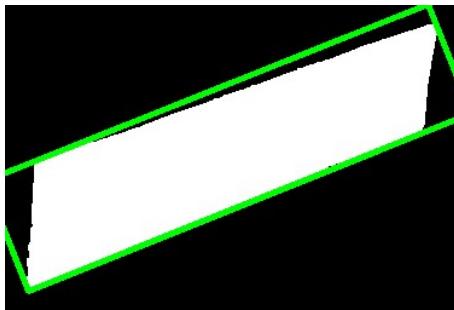


Figura 4.26: Bounding box-ul rotit al regiunii

Având acest bounding box în care componenta conexă este încadrată exact, putem estima dimensiunile acesteia, deci și dimensiunile liniilor Hough. Astfel, pentru lungimea minimă a liniilor și distanța maximă dintre acestea, vom considera anumite procente din înălțimea bounding box-ului. Totuși, aceste procente pot fi destul de mici, de exemplu, 30% pentru lungimea minimă a liniilor și 15% pentru distanța maximă dintre acestea.

Această abordare este necesară deoarece, cu cât unghiul de fotografiere este mai inclinat, cu atât și numărul de înmatriculare va fi mai inclinat în imagine. În astfel de cazuri, algoritmul Hough, în loc să detecteze puține liniile de dimensiuni mari, detectează multe liniile de dimensiuni mici. Astfel, dacă stabilim valori mari parametrilor funcției, multe liniile nu vor fi luate în calcul când numărul de înmatriculare este mai inclinat. Chiar și așa, dacă în continuare nu am obținut toate liniile de care aveam nevoie, putem decrementa treptat lungimea minimă a acestora, pentru a obține în final mai multe liniile.

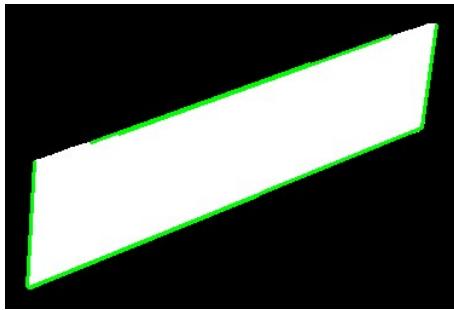


Figura 4.27: Liniile Hough

Liniile de care avem nevoie sunt câte una din fiecare parte a chenarului alb, mai exact partea de sus, jos, stânga, dreapta, pentru că mai apoi să le putem găsi punctele de intersecție. Însă, algoritmul Hough nu ne oferă și informația de care parte aparține fiecare linie, așa că următoarea etapă este o sortare. Pentru început, vom sorta liniile ca fiind verticale sau orizontale. Putem obține această informație în funcție de pantă fiecărei drepte:

$$slope = \frac{y_2 - y_1}{x_2 - x_1}$$

Unghiul de înclinare se calculează apoi astfel:

$$angle = |\arctan(slope) \times \frac{180}{\pi}|$$

Deci, dacă unghiul este aproape de 0 grade, linia este considerată orizontală, iar dacă este aproape de 90 de grade, linia este considerată verticală. Trebuie să ținem cont și de cazurile în care panta este complet verticală, pentru a evita împărțirea la zero.

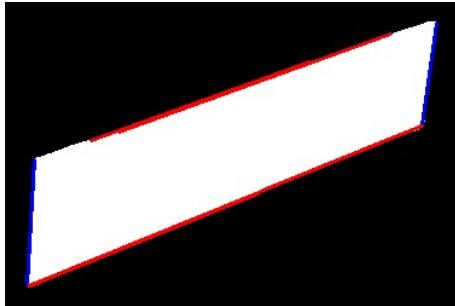


Figura 4.28: Separarea liniilor verticale și orizontale

După sortarea liniilor în verticale și orizontale, putem identifica liniile de care avem nevoie pentru fiecare parte a chenarului alb: partea de sus, jos, stânga, dreapta. Pentru a face acest lucru, luăm ca referință o linie verticală și una orizontală prin centrul imaginii, față de care se compară și se clasifică fiecare linie detectată de algoritmul Hough.

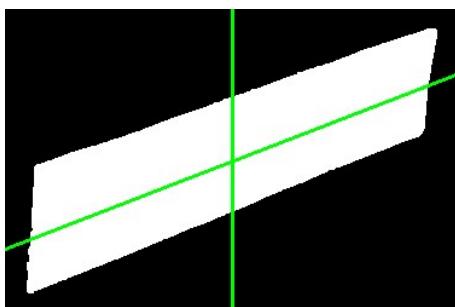


Figura 4.29: Liniile de comparație

În trasarea celor două linii de referință, vom ține cont și de pante, cel puțin în cazul celei orizontale, deoarece numărul de înmatriculare poate fi inclinat la un unghi atât de mare încât unele linii să fie clasificate eronat. În cazul în care numărul de înmatriculare este inclinat la un anumit unghi, și liniile trasate prin centrul imaginii trebuie să fie inclinate la unghiul respectiv pentru a se face clasificarea într-un mod corect.

După ce avem vectorii cu liniile orizontale și verticale, putem să-i parcurgem și să comparăm fiecare linie cu liniile de referință pentru a determina orientarea lor. Linia verticală de referință este utilizată pentru a compara liniile verticale, iar linia orizontală de referință pentru cele orizontale.

Pentru a face această comparație, putem folosi formula distanței dintre două drepte:

$$distance = (x - x_1) \cdot (y_2 - y_1) - (y - y_1) \cdot (x_2 - x_1)$$

Semnul distanței ajută să separăm liniile detectate. Cele de deasupra liniei de referință vor avea distanțe pozitive, iar cele de dedesubt vor avea distanțe negative. La fel, liniile din dreapta vor avea distanțe pozitive față de linia de referință verticală, iar cele din stânga vor avea distanțe negative.

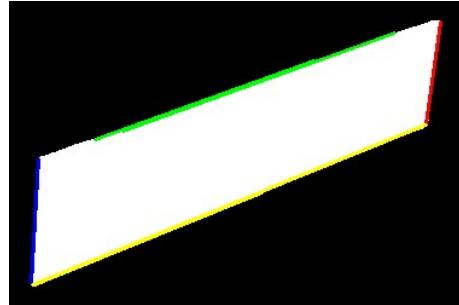


Figura 4.30: Clasificarea Liniile Hough

Ideal, rezultatul ar trebui să fie compus din patru linii, câte una pentru fiecare parte a chenarului alb. Cu toate acestea, din cauza erorilor de detectare sau a înclinațiilor, pot apărea mai multe segmente în loc de o singură linie. Dacă avem mai multe segmente, putem căuta capetele cele mai îndepărțate în funcție de orientare pentru a le lega între ele și a obține o singură linie pentru fiecare parte.

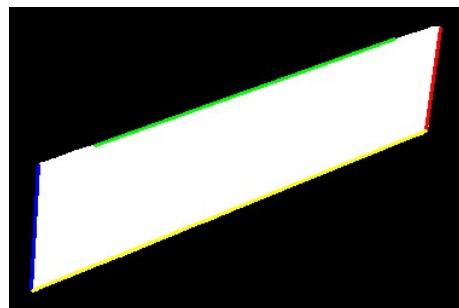


Figura 4.31: Cele 4 linii rezultate

Acesta este și momentul în care verificăm dacă trebuie să aplicăm din nou algoritmul Hough cu parametrii ajustați sau dacă am găsit toate liniile necesare. Dacă avem cele patru linii căutate, pasul următor este să calculăm punctele de intersecție între ele pentru a obține coordonatele colțurilor chenarului alb. Ordinea este următoarea:

- Punctul din stânga sus se obține prin intersectarea liniei din stânga cu cea de sus.
- Punctul din dreapta sus se obține prin intersectarea liniei din dreapta cu cea de sus.
- Punctul din dreapta jos se obține prin intersectarea liniei din dreapta cu cea de jos.
- Punctul din stânga jos se obține prin intersectarea liniei din stânga cu cea de jos.

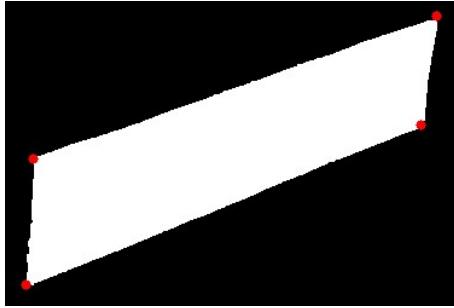


Figura 4.32: Punctele de rectificare

O problemă des întâlnită cu liniile detectate este că punctele de intersecție pot fi în afara imaginii, ceea ce necesită un padding suficient de mare pentru a cuprinde toate cele patru puncte de intersecție. Pentru a rezolva această situație, trebuie să căutăm coordonatele cele mai mici și cele mai mari pentru X și Y dintre punctele calculate anterior:

$$\min X = ROI.width, \min Y = ROI.height, \max X = 0, \max Y = 0,$$

Pentru fiecare punct  $(x, y)$ :

$$\begin{aligned} \min X &= \min(\min X, x); \\ \min Y &= \min(\min Y, y); \\ \max X &= \max(\max X, x); \\ \max Y &= \max(\max Y, y). \end{aligned}$$

După ce avem aceste valori, putem calcula dimensiunea padding-ului necesar pentru fiecare parte a imaginii. Dacă punctele de intersecție sunt în afara imaginii, vom ajusta padding-ul pentru a le include:

$$\begin{aligned} paddingRight &= \max(0, \max X - (\text{src.cols} - 1)), \\ paddingBottom &= \max(0, \max Y - (\text{src.rows} - 1)), \\ paddingLeft &= \max(0, -\min X), \\ paddingTop &= \max(0, -\min Y). \end{aligned}$$

Această abordare ne permite să ajustăm imaginea astfel încât toate punctele de intersecție să fie incluse în limitele acesteia, inclusiv punctele care au coordonate mai mari decât dimensiunile imaginii inițiale. Însă, dacă există și puncte cu coordonate negative pentru care a fost adăugat padding în partea de sus a imaginii sau în partea din stânga, suntem nevoiți să ajustăm toate punctele de intersecție cu valoarea padding-ului adăugat pentru a reflecta noua poziție în cadrul imaginii extinse:

Pentru fiecare punct  $(x, y)$ :

$$\begin{aligned} x &= paddingLeft + x, \\ y &= paddingTop + y. \end{aligned}$$

Acum că avem cele patru coordonate ale colțurilor chenarului alb și ne-am asigurat că sunt în interiorul imaginii, putem trece la pasul de rectificare. Rectificarea presupune transformarea geometrică a regiunii de interes pentru a alinia numărul de înmatriculare într-o poziție corectă, obținând astfel o imagine rectificată.

Pentru a începe procesul de rectificare, calculăm dimensiunile imaginii rezultate. Pentru a determina înălțimea corectă a numărului de înmatriculare, scădem coordonata Y a punctului din stânga sus din coordonata Y a punctului din stânga jos:

$$height = DownLeft.y - UpLeft.y$$

Pentru a obține lățimea, folosim raportul cunoscut pentru numărul de înmatriculare, unde lățimea este de 4.3 ori mai mare decât înălțimea:

$$width = 4.3 \times height$$

Cu aceste dimensiuni calculate, putem stabili punctele corespunzătoare pentru fiecare colț al chenarului alb în imaginea destinată. Ordinea punctelor pentru colțurile chenarului alb este: stânga sus, dreapta sus, dreapta jos, stânga jos. Pentru imaginea rectificată, punctele corespondente vor fi:

- (0, 0) pentru stânga sus,
- (width - 1, 0) pentru dreapta sus,
- (width - 1, height - 1) pentru dreapta jos,
- (0, height - 1) pentru stânga jos.



Figura 4.33: Punctele de rectificare în imaginea destinație

Cu aceste puncte corespondente, putem calcula matricea de transformare utilizând funcția de transformare geometrică. Aplicând această transformare pe imaginea segmentată în formatul grayscale, obținem în final numărul de înmatriculare într-o formă rectificată.



Figura 4.34: Imaginea rectificată

Imaginea rezultată ar trebui să conțină literele și cifrele numărului de înmatriculare pe fundal alb. Însă, există și situații în care rectificarea nu este tocmai ideală, iar astfel pot apărea diverse probleme: anumite caractere nu sunt în totalitate incadrate în imagine, iar astfel, spre exemplu, un caracter este ușor tăiat într-o parte anume, sau, în cazul opus, zona rectificată este ușor mai mare și astfel se obțin detalii în plus la extremitățile imaginii, detalii ce nu sunt necesare.

Ca o primă soluție este decuparea de pe fiecare parte a imaginii a câte o linie sau a câte o coloană, ca mai apoi să se adauge un padding alb în locul liniilor și coloanelor decupate. Decupând o mică parte din imagine, se reduce din posibila zonă care este în plus. De asemenea, padding-ul alb de jur împrejurul imaginii asigură că nu va exista un contur discontinuu al chenarului alb, cum s-ar întâmpla în cazul în care o literă nu este întru totul încadrată în imagine și îintrerupe conturul dreptunghiular al chenarului.



Figura 4.35: Imaginea rectificată cu padding adăugat

Următorul pas este aplicarea unui filtru Binary Thresholding pe imaginea rectificată. Acest lucru va crea o imagine binarizată în care literele și cifrele numărului de înmatriculare vor fi separate de fundal. Însă, se aplică și un filtru de inversare a culorilor deoarece caracterele sunt de culoare neagră pe un fundal alb, pe când acestea ar trebui să fie albe pe un fundal negru.



Figura 4.36: Imaginea rectificată cu filtru binar

Caracterele numărului de înmatriculare pot fi considerate și componente conexe în imaginea binară. De asemenea, știm că numerele de înmatriculare în formatul din România au între 6 și 8 caractere, deci tot atâtea componente conexe trebuie să existe în imaginea binară. Însă, nu este întotdeauna așa, căci uneori numărul de înmatriculare poate avea diferite însemnări pe care le vom considera artefacte și pe care dorim să le eliminăm.

O primă etapă este detectarea tuturor componentelor conexe din imagine și ordonarea acestora după aria lor. Cele mai mari componente conexe tind să fie caracterele numărului de înmatriculare. După ordonare, păstrăm primele opt componente, deoarece, așa cum am stabilit, un număr de înmatriculare nu are mai mult de opt caractere.



Figura 4.37: Primele 8 componente conexe

Cu toate acestea, tot este posibil ca una sau două componente conexe din imagine să fie de fapt zgomot. Pentru a detecta și elmina posibilul zgomot, putem folosi înălțimea ca un factor comun între caractere. Calculăm înălțimea mediană a componentelor conexe și o folosim ca referință pentru a compara înălțimea fiecărei componente cu cea mediană:

$$|median - height| > median \times percent; \quad percent \in [0, 1]$$

Dacă diferența dintre înălțimea componentei conexe curente și cea mediană este mai mare decât un anumit procent din înălțimea mediană, de exemplu 20%, putem considera acea componentă ca fiind zgomot și se elimină. Prin aplicarea acestui filtru bazat pe înălțime, ne putem asigura că imaginea finală conține doar componentele conexe care corespund literelor și cifrelor numărului de înmatriculare.



Figura 4.38: Eliminarea componentelor conexe eronate

De asemenea, ca la un pas anterior, pentru a fi complet siguri că nu există legături incorecte între componentele conexe, putem aplica și o operație morfologică de tip Opening. Această operație separă obiectele legate incorect prin linii fine.

După operația de Opening, este posibil să apară noi componente conexe, motiv pentru care se aplică din nou funcția descrisă anterior și astfel se elimină noile componente conexe, care nu sunt caractere, rezultate din urma operației morfologice.

În urma aplicării tuturor tehniciilor de procesare a imaginilor, am obținut o imagine în care textul numărului de înmatriculare arată ca și cum ar fi fost tastat manual într-un fișier. Aceasta este o imagine potrivită pentru modelul de recunoaștere a caracterelor Tesseract OCR. Totuși, pentru a obține rezultate mai bune, este util să furnizăm modelului AI fiecare caracter separat, pe rând.

Pentru a realiza acest lucru, aplicăm pe imaginea binară rectificată funcția implementată într-un pas anterior ce detectează componentele conexe, pentru a obține bounding box-urile tuturor caracterelor. Rezultatul este din nou un vector ce conține coordonatele colțului din stânga sus ale fiecărui bounding box, precum și înălțimea și lățimea fiecărei componente. Este important să cunoaștem ordinea corectă a caracterelor în imagine pentru a putea reconstrui textul numărului de înmatriculare din caracterele recunoscute.



Figura 4.39: Bounding box-urile caracterelor

Cu toate acestea, ordinea în care componentele conexe sunt stocate în vector poate să nu coincidă cu ordinea reală a caracterelor în imagine. Pentru a remedia acest lucru, sortăm vectorul în ordine crescătoare după coordonata X a fiecărui bounding box. Astfel, obținem o ordine a componentelor conexe care corespunde ordinii reale a caracterelor din imagine.

Stim că numerele de înmatriculare care respectă formatul din România conțin una sau două litere, urmate de două sau trei cifre, și încheiate cu alte trei litere. Această structură ne poate ajuta să îmbunătățim recunoașterea caracterelor. În următoarea etapă, vom partitiona caracterele numărului de înmatriculare în trei regiuni.

Deși stim numărul total de caractere din imagine, primele două regiuni ale numărului de înmatriculare pot avea un număr variabil de caractere, ceea ce înseamnă că nu stim exact unde se termină prima regiune și începe a doua. Pentru a determina acest lucru, trebuie să calculăm distanțele dintre colțurile bounding box-urilor vecine și să identificăm distanța cea mai mare, deoarece aceasta va fi între prima și a doua regiune a numărului de înmatriculare.

Odată ce am identificat cea mai mare distanță între componente conexe, putem partitiona caracterele în funcție de regiuni:

- Prima regiune: cuprinde toate componentelete conexe de la început până la cea dinaintea distanței calculate.
- A doua regiune: începe de la componenta conexă imediat după distanța cea mai mare și se întinde până la componenta conexă dinaintea ultimelor trei.
- A treia regiune: include ultimele trei componente conexe, începând de la antepenultima și terminând cu ultima.

Intr-un final, se memoreaza indexul de început pentru fiecare regiune.



Figura 4.40: Clasificarea caracterelor

Tesseract OCR poate primi ca input imaginea binară cu textul complet și apoi i se pot specifica zonele pentru recunoașterea caracterelor, bazându-ne pe bounding box-uri. Cu toate acestea, bounding box-urile nu includ spațiu pentru padding, ceea ce este esențial pentru model AI. De aceea, vom segmenta fiecare caracter din imagine conform bounding box-urilor și îi vom adăuga padding.

Numărul de linii și coloane pentru padding poate fi calculat automat, folosind un procent din dimensiunile bounding box-ului. Vom considera un procent semnificativ, cum ar fi 60%. Prin înmulțirea acestui procent cu înălțimea și lățimea bounding box-ului, putem determina numărul de linii și coloane care trebuie adăugate în jurul imaginii segmentate pentru a oferi suficient spațiu:

$$\begin{aligned} paddingX &= BBox.width \times percent \\ paddingY &= BBox.height \times percent \end{aligned}$$

Însă, înainte de a adăuga padding fiecărui caracter, este necesar să actualizăm informațiile legate de bounding box. Dimensiunile padding-ului trebuie adăugate la dimensiunile inițiale ale bounding box-ului, de două ori, deoarece padding-ul este aplicat pe ambele părți ale imaginii, atât pe axa X, cât și pe axa Y:

$$\begin{aligned} BBox.width &= BBox.width + paddingX \times 2, \\ BBox.height &= BBox.height + paddingY \times 2, \end{aligned}$$

Revenim la bounding box-urile inițiale, pe baza cărora vom realiza segmentările în imaginea binară pentru a extrage fiecare caracter. Din bounding box-uri cu padding-ul asociat, putem extrage valoarea specifică a padding-ului pentru fiecare caracter și apoi o putem adăuga la imaginea segmentată.

După ce toate caracterele segmentate sunt obținute, fiecare având padding-ul adăugat, următorul pas este concatenarea lor într-un singur sir, recreând astfel textul initial cu această nouă distanțare.

Va trebui să generăm, de asemenea, o imagine complet neagră de dimensiuni suficiente de mari pentru a cuprinde toate caracterele, împreună cu padding-ul lor. În primul rând, vom determina înălțimea acestieia, care va fi echivalentă cu înălțimea celui mai

înalt bounding box. Lățimea va fi calculată ca suma tuturor lățimilor bounding box-urilor.

Anterior, am ajustat dimensiunile bounding box-urilor pentru a se potrivi cu padding-ul, însă acum trebuie să modificăm și colțurile acestora. Coordonata X a fiecărui bounding box poate fi calculată ca suma lățimilor bounding box-urilor anterioare acestuia, iar coordonata Y poate fi calculată ca diferența dintre înălțimea imaginii și înălțimea bounding box-ului, împărțită la 2, pentru a centra caracterul pe axa Y:

$$width = 0,$$

Pentru fiecare bounding box:

$$y = \frac{height - BBox.height}{2},$$

$$x = width,$$

$$width = width + BBox.width.$$

Din acest punct, revenim la segmentele realizate anterior pentru fiecare caracter și le inserăm, pe rând, în imaginea complet neagră generată anterior, conform noilor bounding box-uri.



Figura 4.41: Spațierea caracterelor

Acum, cunoscând atât bounding box-urile cu padding, cât și cele fără padding, putem folosi indicii obținuți anterior pentru a separa aceste bounding box-uri în funcție de regiunile de care aparțin.

Pe baza acestor indici, se pot impune condiții suplimentare pentru a ne asigura că segmentarea imaginii originale conține într-adevăr numărul de înmatriculare. Se verifică dacă al doilea indice este egal cu 2 sau 3, deoarece s-a stabilit că prima regiune poate avea maximum două caractere. De asemenea, se verifică dacă intervalul dintre al doilea și al treilea indice conține exact două sau trei caractere.

Următorul pas implică utilizarea modelului AI Tesseract OCR pentru recunoașterea textului în imagine. Se creează o instanță de tip `tesseract::TessBaseAPI` și se dezactivează toate dicționarele implicate, deoarece acestea nu sunt necesare în acest context. Se setează și un whitelist care indică dacă modelul trebuie să caute litere sau cifre, în funcție de regiunea în care se află caracterul dintre cele trei regiuni obținute anterior.

Modelului i se oferă ca input imaginea binară cu caracterele distanțate și se parcurg, pe rând, fiecare dintre bounding box-uri pentru a specifica zonele în care să se recunoască textul.



Figura 4.42: Bounding box-urile caracterelor cu padding adăugat (input-uri OCR)

Dacă în segmentul de imagine nu a fost recunoscut niciun caracter sau au fost recunoscute mai multe, se decrementează dimensiunea bounding box-ului și se încearcă

din nou recunoașterea, sperând că de această dată să fie identificat caracterul corect. Decrementarea continuă până când bounding box-ul revine la dimensiunile inițiale, închadrând caracterul cu precizie minimă:

$$\begin{aligned}x &= x - 1, \\y &= y - 1, \\width &= width - 2, \\height &= height - 2.\end{aligned}$$



Figura 4.43: Decrementarea bounding box-urilor până la recunoaștere corectă a caracterelor

Dacă totuși bounding box-ul a revenit la dimensiunile inițiale și modelul AI nu a reușit să recunoască caracterul din imagine, este posibil ca problema să fie cauzată de litera "l". În anumite cazuri, Tesseract OCR nu recunoaște acest caracter, aşa că se utilizează o tehnică de Template matching pentru a determina dacă respectivul caracter nerecunoscut este "l" sau nu.



Figura 4.44: Segmentarea literei nerecunoscute

În pasul de template matching, se va încărca în program o imagine template pentru litera "l".



Figura 4.45: Template-ul literei "l"

Pentru început, redimensionăm această imagine template la dimensiunile caracterului segmentat. Vom calcula raportul de aspect pentru imaginea template și noua lățime pe care o va avea:

$$\begin{aligned}aspectRatio &= Template.width / Template.height, \\Template.width &= Sursa.height * aspectRatio.\end{aligned}$$

Pe imaginea redimensionată, se aplică un filtru de tip "Binary Thresholding" pentru a obține regiunea exactă a literei.



Figura 4.46: Template-ul literei "i" cu filtru binar

În acest moment, litera poate fi înconjurate neuniform de linii și coloane negre, motiv pentru care dorim să o extragem din imaginea inițială și să creăm o imagine în care litera să aibă un spațiu uniform pe toate laturile. Astfel, se caută contururile în imagine pentru a identifica conturul singurei regiuni albe existente, care reprezintă litera. Pe baza acestui contur, se extrage litera din imagine și se inserează într-o imagine complet neagră cu dimensiuni egale cu cele ale literei, dar cu un padding de o unitate pe toate laturile.

Deși template-ul a fost redimensionat la înălțimea segmentării, din cauza uniformizării padding-ului, este posibil ca dimensiunea să nu rămână cea stabilită inițial. Prin urmare, vom calcula diferența dintre aceste două înălțimi pentru a obține padding-ul suplimentar care trebuie adăugat la şablon în partea de sus și de jos. Această diferență se împarte la jumătate pentru a adăuga aceeași cantitate de padding pe fiecare parte a imaginii. De asemenea, se verifică dacă această diferență este pară, deoarece, în caz contrar, va trebui incrementat padding-ul pe una dintre părți:

$$\begin{aligned} firstPadding &= \frac{Sursa.height - Template.height}{2}, \\ secondPadding &= firstPadding, \\ secondPadding &= secondPadding + 1, \\ \text{dacă } (Sursa.height - Template.height) \bmod 2 &= 1. \end{aligned}$$

Pentru a stabili și lățimea imaginii template, se va adăuga un padding similar cu cel anterior, calculat pe baza diferenței dintre lățimea imaginii sursă și lățimea imaginii template. La fel ca înainte, diferența se împarte la jumătate pentru a determina padding-ul care trebuie adăugat de fiecare parte a şablonului și se verifică dacă diferența este pară, astfel încât padding-ul să fie distribuit uniform:

$$\begin{aligned} firstPadding &= \frac{Sursa.width - Template.width}{2}, \\ secondPadding &= firstPadding, \\ secondPadding &= secondPadding + 1, \\ \text{dacă } (Sursa.width - Template.width) \bmod 2 &= 1. \end{aligned}$$

Acum că cele două imagini sunt ajustate la aceeași dimensiune și cu litera "I" reprezentată central după adăugarea padding-ului, se realizează intersectarea celor două regiuni albe utilizând operația logică 'și'.



Figura 4.47: Intersecția celor două imagini

Scorul Dice poate fi calculat pe baza acestei intersecții, indicând cât de mult se suprapun cele două regiuni. Dacă scorul obținut este semnificativ, de exemplu cel puțin 80%, se poate concluziona că respectivul caracterul nerecunoscut este 'I'. În caz contrar, se consideră că segmentarea inițială nu reprezintă numărul de înmatriculare și se continuă cu următoarea iterație.

În ultima etapă, se compune textul numărului de înmatriculare folosind caracterele recunoscute, se calculează o medie a nivelului de încredere pe baza tuturor scorurilor de încredere obținute pentru fiecare caracter, se memorează data și ora curente, iar în imaginea sursă se desenează un chenar care încadrează numărul de înmatriculare, pe baza segmentării inițiale.

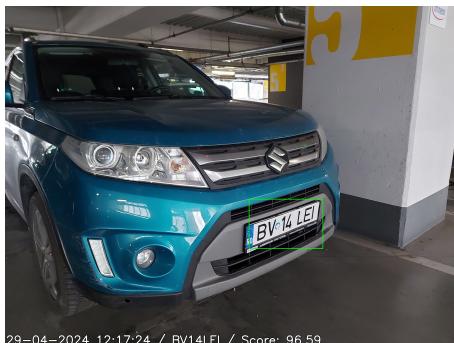


Figura 4.48: Imaginea de output

În situația în care nu a fost recunoscut textul integral al numărului de înmatriculare, acesta este catalogat drept "necunoscut".

Pentru toate componentele descrise anterior, s-au dezvoltat un set de teste în cadrul proiectului „Tests” pentru a asigura robustețea și fiabilitatea aplicației. Aceste teste sunt concepute pentru a evalua comportamentul fiecărei funcții în condiții diverse, cum ar fi procesarea imaginilor neinițializate sau cu un număr diferit de canale față de cel așteptat, și manipularea pixelilor de un tip de date diferit. Aceasta abordare ajută în identificarea punctelor slabe ale procesării și asigură că modulele pot gestiona orice tip de intrare în mod corect.

Pentru funcțiile matematice în special, s-au implementat teste pentru a preveni erori precum împărțirile la zero și alte situații excepționale care pot apărea în timpul execuției. Aceste teste sunt vitale pentru a evita crash-uri ale aplicației și pentru a asigura că output-ul matematic este întotdeauna valid și precis.

Ca rezultat al acestor teste, în codul fiecărei funcții s-au adăugat multiple condiții de verificare a datelor de intrare. Aceste verificări suplimentare asigură că, indiferent de variațiile datelor de intrare, funcțiile vor răspunde în mod adecvat, trecând testele și evitând posibilele erori care ar putea să compromită funcționalitatea aplicației.

## 4.3 Interfață grafică

Pentru construcția interfeței grafice a aplicației, s-a optat pentru utilizarea framework-ului Qt, recunoscut pentru flexibilitatea și eficiența sa în dezvoltarea de interfețe moderne pentru utilizator.

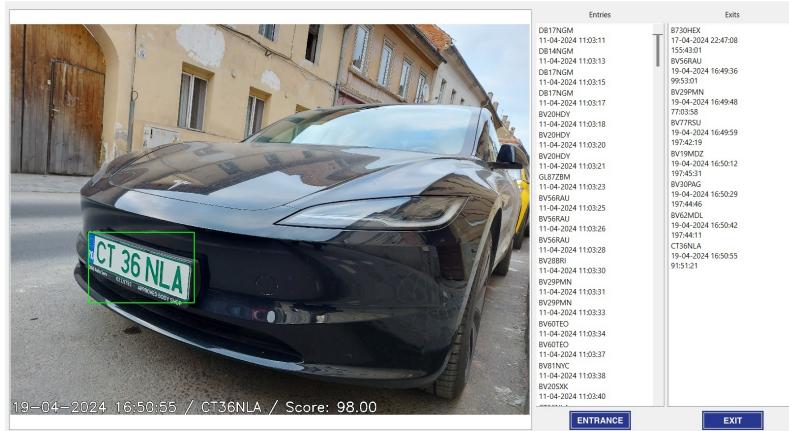


Figura 4.49: Interfață grafică

Pentru început, se poate observa că în partea stângă este afișată imaginea de input, prezentând vehiculul surprins, cu numărul de înmatriculare clar vizibil în cadrul bounding box-ului său. În partea inferioară a imaginii, sunt afișate: data și ora captării fotografiei, în format zi-lună-an ore:minut:secunde, textul recunoscut din numărul de înmatriculare și nivelul de confidență asociat recunoașterii.

Mai departe, în partea dreaptă a interfeței, se pot găsi două liste detaliate care documentează istoricul vehiculelor care au intrat și respectiv ieșit din parcarea. Pentru fiecare vehicul înregistrat, listele afișează textul numărului de înmatriculare și data și ora la care imaginea a fost captată. În plus, lista vehiculelor care au părăsit parcarea include și durata de staționare a fiecărui vehicul, oferind astfel o perspectivă completă asupra traficului și utilizării spațiului de parcare.

Sub fiecare listă, este situat câte un buton specific, similar celor întâlnite în terminalele de acces din parcările auto: un buton pentru intrare sub lista intrărilor și unul pentru ieșire sub lista ieșirilor. Aceste butoane permit utilizatorilor să încarce imagini pentru procesare, similare cu cele obținute de camerele video plasate la intrările în parcări.

În continuare, făcând o analiză și asupra codului sursă, conform principiilor programării orientate pe obiect, este inclusă implementarea unei clase denumite "Vehicle", din interiorul unei biblioteci dinamice "UserInterface". Această clasă permite instantierea unui obiect pentru fiecare vehicul înregistrat, utilizând următorii parametri:

- **id:** identificator unic pentru fiecare vehicul,
- **path:** calea către imaginea asociată vehiculului,
- **ticket:** numărul biletului de parcare,
- **licensePlate:** numărul de înmatriculare al vehiculului,
- **time:** data și ora la care vehiculul a fost înregistrat,

■ **timeParked:** durata de staționare în parcare.

Pentru fiecare parametru al clasei Vehicle, sunt implementate funcții de getter și setter, care îndeplinește roluri esențiale în managementul datelor. Funcțiile de getter sunt utilizate pentru a accesa valorile atributelor instanței, permitând consultarea informațiilor despre vehicule într-un mod sigur și controlat. Pe de altă parte, funcțiile de setter oferă mecanismul necesar pentru actualizarea valorilor acestor attribute, asigurând că modificările de stare sunt gestionate corespunzător, reflectând orice schimbări în comportamentul sau statutul vehiculelor. Implementarea acestor funcții sprijină încapsularea datelor și ajută la protecția integrității acestora și la menținerea unei arhitecturi clare și organizate a codului.

De asemenea, a fost inclusă și o funcție de supraincarcare a operatorului de ieșire «, destinată afișării informațiilor despre vehicul. Această funcție permite scrierea ușoară și directă a tuturor parametrilor clasei, menționată anterior, în fluxurile de ieșire.

Pentru păstrarea datelor chiar și după închiderea programului, s-a optat pentru folosirea unei baze de date. În momentul instantierii unui obiect de tip Vehicle, toate informațiile obiectului sunt automat înregistrate în baza de date, permitând stocarea structurată și accesul rapid la informații la nevoie.

Mai mult, la lansarea programului, sistemul inițiază automat o procedură de încarcare a datelor din baza de date, restabilind starea anterioară a listei de vehicule. Această funcționalitate asigură că toate informațiile necesare sunt disponibile imediat după pornirea aplicației, facilitând o operare continuă și fără întreruperi a sistemului.

De asemenea, pentru fereastra principală a aplicației există clasa MainWindow, aflată tot în interiorul bibliotecii dinamice "UserInterface", care este esențială pentru gestionarea interfeței utilizatorului și a datelor procesate. Aceasta include o serie de membri și metode care facilitează manipularea și afișarea informațiilor despre vehicule, precum și interacțiunea cu utilizatorul.

Printre membrii săi, în primul rând, regăsim elementele de Qt care compun interfața grafică, cum ar fi butoanele, listele, etichetele și zonele de vizualizare. Acestea sunt configurate pentru a oferi o experiență de utilizare intuitivă și accesibilă, permitând utilizatorilor să interacționeze eficient cu aplicația. De asemenea, se regăsește și fișierul bazei de date și calea către acesta, asigurând gestionarea persistentă și sigură a datelor vehiculelor.

De asemenea, tot ca membru al clasei este și instantierea unui obiect de tip Vehicle curent selectat și informațiile legate de acesta, cum ar fi imaginea vehiculului, calea către această imagine și textul format pentru afișare, dar și o listă de obiecte de tip Vehicle în care sunt stocate toate instanțe ale clasei. Acest vector facilitează accesul rapid și eficient la informații complete despre fiecare vehicul, permitând referințarea și manipularea acestora după necesități.

Pentru construcția interfeței grafice, se inițializează un QGraphicsView care funcționează ca un container pentru elementele grafice. Acesta este asociat cu un QGraphicsScene, responsabil pentru gestionarea obiectelor grafice, și un QGraphicsPixmapItem, adăugat la scena respectivă, care permite afișarea eficientă a imaginilor.

Fiecare dintre cele două butoane QPushButton este încărcat cu o imagine reprezentativă pentru funcțiile de intrare, respectiv ieșire, extrase din baza de date prin intermediul căilor specificate. Dimensiunile și iconițele butoanelor sunt ajustate pentru a se integra în designul interfeței.

De asemenea, se face inițializarea a două liste QListWidget care sunt utilizate pentru a afișa înregistrările vehiculelor care intră, respectiv ies din parcare. Fiecare listă este asociată cu câte o etichetă QLabel descriptivă, care este configurată pentru a avea textul centrat.

Organizarea componentelor interfeței este realizată prin structurarea mai multor layout-uri:

- **Layout-ul cu butoanele:** Este un layout orizontal QHBoxLayout în care cele două butoane sunt amplasate central. Acest aranjament asigură că butoanele sunt ușor accesibile și vizibil poziționate, contribuind la o interfață estetic plăcută și funcțională.
- **Layout-ul de intrări:** Acesta este un layout vertical QVBoxLayout care începe cu eticheta de intrări declarată anterior, urmată de lista în care sunt memorate intrările vehiculelor. Organizarea verticală asigură că eticheta este afișată clar deasupra listei, creând o structură logică și estetic plăcută.
- **Layout-ul de ieșiri:** Similar cu layout-ul de intrări, acest layout vertical QVBoxLayout conține eticheta de ieșiri și o listă dedicată ieșirilor vehiculelor. Din nou, această organizare verticală menține coerenta vizuală și funcționalitatea, facilitând accesul și vizualizarea datelor.
- **Layout-ul cu listele:** Este un layout orizontal QHBoxLayout care adaugă layouturile de intrări și ieșiri unul lângă celălalt. Această configurație facilitează afișarea simultană a listelor de intrări și ieșiri, permitând utilizatorilor să compare și să acceseze rapid informațiile.
- **Layout-ul din dreapta:** Acest layout vertical QVBoxLayout adaugă layout-ul cu listele, care include ambele liste și etichetele lor, precum și layout-ul care cuprinde butoanele de interacțiune. Organizarea verticală optimizează utilizarea spațiului și asigură o separare clară între zonele de afișare a datelor și cele de acțiune.
- **Layout-ul din stânga:** Este tot un layout vertical QVBoxLayout folosit pentru a include QGraphicsView, spațiul în care sunt afișate imaginile cu vehiculele. Acest layout oferă o vizualizare centrală și neîntreruptă a elementelor grafice relevante.
- **Layout-ul principal:** Este un layout-ul de tip orizontal QHBoxLayout, ce combină layout-ul din dreapta cu cel din stânga. Alocă o proporție mai mare pentru cel din stânga comparativ cu cel din dreapta, asigurând astfel un spațiu adecvat pentru vizualizarea graficelor și o eficiență în utilizarea spațiului de afișare.

După inițializarea interfeței grafice, începe procesul de încărcare a datelor din baza de date prin deschiderea unui fișier text care conține informațiile salvate despre vehicule. Fișierul este citit linie cu linie, iar datele colectate pentru fiecare vehicul sunt folosite pentru a instantia câte un obiect Vehicle. Dacă se cunosc informații despre timpul staționat al vehiculului, acest lucru indică faptul că vehiculul este înregistrat la ieșirea din parcare, și astfel, timpul staționat al instanței este actualizat corespunzător. Fiecare obiect Vehicle creat este adăugat la lista de vehicule, iar procesul continuă astfel până când toate liniile din fișier sunt citite, după care fișierul este închis.

Odată ce toate datele sunt încărcate local din baza de date, informațiile trebuie afișate în interfața grafică. Această operațiune se realizează iterând prin lista vehiculelor încărcate și actualizând interfața de utilizator corespunzător. Pentru fiecare vehicul, se compune un text de afișare care include numărul de înmatriculare, data și ora înregistrării vehiculului, și, dacă este disponibil, timpul staționat în parcare. Se creează un element QListWidgetItem cu textul format și se adaugă în lista de intrări dacă nu există informații despre timpul staționat, sau în lista de ieșiri, dacă aceste informații sunt disponibile.

In final, se fac și conexiunile pentru butoane. Când unul dintre cele două butoane de ieșire sau intrare sunt apăsate, se emite semnalul "clicked", care declanșează executarea metodei ce permite utilizatorului să încarce imagini ale vehiculelor pentru a fi procesate, fie la intrarea, fie la ieșirea din parcare, simbolizând momentele când vehiculele trec prin punctele de control.

După ce imaginea a fost încărcată și validată, se apelează metoda de procesare care returnează: textul numărului de înmatriculare, data și ora fotografierii și imaginea de output cu numărul de înmatriculare încadrat într-un bounding box și informațiile obținute scrise pe aceasta.

Pentru simularea unui sistem de tichete, fiecare vehicul este catalogat cu câte un ID comun în denumirea tuturor imaginilor în care acesta apare. Se obține acest ID din denumirea imaginii. De asemenea, fiecare imagine de output este salvată pe calculatorul personal, astfel că se creează calea de salvare, care va fi formată din calea către baza de date, la care se adaugă indexul vehiculului respectiv din întreaga listă de vehicule. Acest nou obiect de tip vehicle este memorat în listă.

După încărcarea și validarea imaginii, se inițiază metoda de procesare care extrage și returnează: textul numărului de înmatriculare, data și ora la care fotografia a fost realizată și imaginea de output cu numărul de înmatriculare încadrat într-un bounding box și cu detaliile extrase afișate direct pe aceasta.

Pentru a simula un sistem de tichete, fiecare vehicul este identificat printr-un ID unic, care este inclus în denumirea tuturor imaginilor în care vehiculul apare. Acest ID este extras și memorat din numele fișierului imaginii. Fiecare imagine procesată este salvată local pe calculator, aşadar și aceasta curentă va fi salvată. Calea de salvare este generată automat, începând de la directorul bazei de date și adăugându-se indexul vehiculului din lista totală de vehicule. În final, se creează un nou obiect de tip Vehicle cu toate aceste informații obținute și se adaugă la listă.

Dacă butonul apăsat este cel de ieșire, se inițiază calculul timpului de staționare al vehiculului. Dacă numărul de înmatriculare al vehiculului a fost recunoscut cu succes în imagine, se caută în lista de vehicule pentru a identifica cea mai recentă apariție a acestui vehicul. În cazul în care numărul de înmatriculare nu poate fi recunoscut și este etichetat ca "nerecunoscut", căutarea se va baza pe numărul tichetului asociat vehiculului.

După identificarea vehiculului în listă, se extrage data și ora intrării respective. Utilizând aceste informații împreună cu data și ora ieșirii, care sunt disponibile în momentul apăsării butonului de ieșire, se calculează diferența de timp între cele două momente. Acest calcul va determina durata totală de staționare a vehiculului în respectiva parcare. Această durată este apoi folosită pentru a actualiza înregistrarea vehiculului.

Ulterior, se alcătuiește textul de afișare cu numărul de înmatriculare recunoscut, data și ora și, dacă există, timpul staționat. Acest text este adăugat în lista corespun-

zătoare în funcție de butonul apăsat. Toate informațiile legate de vehiculul curent sunt scrise în baza de date, folosindu-se funcția de scriere supraincarcată din clasa Vehicle.

Pentru a afișa noua imagine de output în interfața grafică, procesul începe cu eliminarea QGraphicsItem-ului existent din scenă, asigurându-se că nu există suprapunerii sau conflicte vizuale. Acest lucru se realizează prin îndepărțarea și stergerea elementului grafic curent, făcând loc pentru adăugarea unui nou conținut.

După curățarea scenei, se creează un nou QGraphicsPixmapItem, care este inițializat cu noua imagine procesată. Acest element nou este apoi adăugat la scena grafică, reprezentând noua imagine de afișare.

Următorul pas implică ajustarea proprietăților QGraphicsView. Aceasta este configurat pentru alinierea la centru, asigurând că imaginea este prezentată în mod egal și centralizat în spațiul disponibil. În plus, se obține dreptunghiul delimitator al noii imaginii, care este utilizat pentru a seta zona de vizualizare. Această setare definește limitele în care elementele grafice pot fi vizualizate, garantând că întreaga imagine este încadrată corespunzător.

În final, imaginea este scalată și încadrată în vizualizator, cu respectarea proporțiilor originale ale acesteia, indiferent de dimensiunile și orientarea vizualizatorului.

De asemenea, și listele de intrări și ieșiri din aplicație sunt conectate printr-un mecanism de semnale și sloturi care leagă selecția unui element din listă la o metodă dedicată afișării imaginii asociate vehiculului.

Pentru a face acest lucru, se obține ID-ul imaginii din datele atașate elementului de listă QListWidgetItem. Utilizând ID-ul obținut, se construiește calea către fișierul de imagine și se încarcă imaginea de la această cale pentru a fi procesată ulterior. La fel ca metodele asociate butoanelor anterioare, se elimină elementul grafic anterior din scenă pentru a se evita suprapunerile. Apoi, se creează un nou QGraphicsPixmapItem din imaginea încărcată și se adaugă acest element în scenă. Apoi, se ajustează proprietățile QGraphicsView pentru a asigura că noua imagine este afișată corect și complet.

În final, la închiderea aplicației, se apelează în mod automat destructorul clasei, care are rolul de a închide fișierul de scriere.

Întregul proiect este încorporat în cadrul proiectului Application, care include funcția main. Aceasta funcție este responsabilă pentru inițierea și lansarea interfeței grafice.

## 4.4 Generarea și configurarea proiectului

Pentru configurarea proiectului, s-a optat pentru utilizarea software-ului CMake, care este foarte popular în rândul dezvoltatorilor software de C++. Conform structurii stabilită anterior, soluția este compusă din trei proiecte principale, la care se adaugă un proiect dedicat testelor. Astfel, pentru a configura întreaga soluție, sunt necesare cinci fișiere CMake: câte unul pentru fiecare proiect și proiectul de teste, plus un fișier suplimentar pentru configurarea globală a întregii soluții. Această abordare modulară asigură o organizare clară și facilitează gestionarea independentă și eficientă a fiecărui segment al proiectului.

Configurația CMake pentru soluție începe cu specificarea versiunii minime necesare de CMake prin funcția `cmake_minimum_required()`, stabilită la 3.16, pentru a asigura compatibilitatea cu caracteristicile folosite în fișierele de configurație. Acest lucru ga-

rantează că toate funcționalitățile moderne ale CMake sunt disponibile și că proiectul poate fi construit în medii diverse de dezvoltare.

Se folosește comanda `project()` pentru a defini numele proiectului ca fiind "ParkingLotSecurity", subliniind scopul său și facilitând gestionarea modulară a setărilor asociate în medii de dezvoltare integrate.

Directoarele de ieșire pentru executabile și biblioteci sunt setate cu ajutorul funcției `set()`, astfel încât toate fișierelor executabile și bibliotecile generate să fie plasate în directorul bin sub directorul binar al proiectului (`CMAKE_BINARY_DIR`). Această structură de director ajută la organizarea clară a fișierelor de output, făcându-le mai ușor de accesat și gestionat. Directorul rădăcină al proiectului este stabilit la locația actuală a fișierului CMake (`CMAKE_CURRENT_SOURCE_DIR`), furnizând o referință centrală pentru alte scripturi și module care pot necesita calea către rădăcina proiectului. Proiectul de start pentru Visual Studio este configurat pentru a deschide automat proiectul Application când soluția este încărcată, facilitând dezvoltarea și depanarea rapidă în acest IDE.

De asemenea, CMake configuraază și caută pachetele necesare folosindu-se de comanda `find_package()`, bibliotecile necesare fiind: OpenCV, Qt6 și Tesseract.

Proiectul include mai multe subdirectoare cu ajutorul funcției `add_subdirectory()`, fiecare dedicat unei părți specifice a aplicației:

- `tests`: pentru unit testing și asigurarea calității codului,
- `src/Application`: care conține codul principal al aplicației,
- `src/ImageProcessingUtils`: pentru utilitățile de procesare a imaginilor,
- `src/GUI`: care gestionează componentele interfeței grafice.

Această structură modulară creată de CMake ajută la menținerea unei organizări clare a codului sursă și la separarea responsabilităților în cadrul proiectului, ceea ce este vital pentru întreținerea și scalabilitatea pe termen lung a aplicației.

Pentru început, ne concentrăm asupra scriptului CMake dezvoltat pentru configurarea proiectului `ImageProcessingUtils`, dedicat gestionării tuturor aspectelor legate de procesarea imaginilor în cadrul aplicației. Folosind comanda `project()`, se inițiază un nou proiect în CMake cu denumirea „`ImageProcessingUtils`”, stabilind acest modul ca responsabil principal pentru funcțiile de procesare a imaginilor.

Comanda `add_definitions()` introduce definiția preprocesorului `-DIMAGEPROCESSINGUTILS_EXPORTS`. Acest macro este utilizat frecvent pentru a reglementa vizibilitatea funcțiilor și claselor în cadrul bibliotecilor partajate, semnalând că acestea sunt exportate din bibliotecă. Comenzile `file()` aplică metoda `GLOB` pentru a colecta automat toate fișierele header (.h) și sursă (.cpp) din directorul curent al proiectului. Această metodă simplifică adăugarea fișierelor la proiect și garantează includerea tuturor resurselor necesare.

Utilizând comanda `add_library()`, se construiește o bibliotecă partajată denumită conform variabilei `PROJECT_NAME`, în acest caz, `ImageProcessingUtils`. Biblioteca include toate fișierele header și sursă colectate. Bibliotecile partajate sunt avantajoase în reducerea dimensiunii totale a programului și permit actualizarea componentelor individuale fără necesitatea recomplirii întregii aplicații.

În continuare, comanda `target_include_directories()` specifică directoarele unde compilatorul trebuie să caute fișierele de antet necesare pentru bibliotecile OpenCV și Tesseract, asigurând acces la API-urile și funcționalitățile acestor biblioteci esențiale pentru proiect. Comanda `target_link_libraries()` asociază proiectul ImageProcessingUtils cu bibliotecile OpenCV și Tesseract. Aceasta include versiunile de release și de debug ale Tesseract, garantând că proiectul are acces la funcționalitățile necesare pentru procesarea avansată a imaginilor și recunoașterea optică a caracterelor.

Această configurare a modulului pregătește proiectul să abordeze diverse sarcini de procesare a imaginilor, beneficiind de capabilitățile avansate ale OpenCV și Tesseract, vitale pentru analiza detaliată și manipularea datelor vizuale.

În continuare, să examinăm scriptul CMake creat pentru configurarea proiectului UserInterface, care se concentrează pe dezvoltarea interfeței grafice a aplicației. Fiecare linie de cod din acest script are un rol esențial în structurarea și pregătirea proiectului pentru compilare. Comanda `project()` inițiază un nou proiect în CMake sub denumirea "UserInterface". Aceasta este importantă pentru coordonarea setărilor specifice necesare modulelor interfeței grafice ale aplicației.

Comanda `add_definitions()` introduce o definiție globală a preprocesorului, `-DUI_EXPORTS`, utilă în gestionarea vizibilității funcțiilor și claselor în cadrul bibliotecilor dinamice. Aceasta facilitează administrarea exporturilor de componente în interfață grafică.

Prin intermediul funcției `set()`, `CMAKE_AUTOUIC` este dezactivat (OFF) pentru a opri procesarea automată a fișierelor de interfață Qt (.ui) de către uic (Qt User Interface Compiler). Astfel, se oferă posibilitatea controlului manual asupra acestui proces, dacă este necesar. În schimb, `CMAKE_AUTOMOC` este activat (ON), permitând CMake să gestioneze automat compilarea fișierelor moc (Meta-Object Compiler) pentru codul Qt ce implică utilizarea semnalelor și sloturilor, asigurând o integrare adecvată a mecanismelor Qt.

Utilizând comanda `file()`, scriptul colectează toate fișierele sursă (.c) și header (\*.h) din directorul curent al proiectului, facilitând astfel includerea lor automată și completă în proiect. De asemenea, se extinde colecția de fișiere pentru a include toate fișierele sursă și header din subdirectoare, utilizând opțiunea `GLOB_RECURSE` pentru a efectua căutări recursive, iar opțiunea `LIST_DIRECTORIES false` indică faptul că directoarele nu sunt incluse în rezultate. Aceasta asigură o gestionare eficientă și organizată a resurselor proiectului.

Funcția `add_library()` creează un nou modul bibliotecă numit conform variabilei `PROJECT_NAME` (denumită "UserInterface"), definit ca bibliotecă statică. Se utilizează toate fișierele GUI colectate anterior pentru compilarea acestei biblioteci. Bibliotecile statice sunt integrate direct în executabil în timpul compilării, ceea ce poate duce la performanțe îmbunătățite în timpul rulării.

Comanda `add_dependencies()` stabilește că UserInterface depinde de modulul ImageProcessingUtils. Aceasta înseamnă că ImageProcessingUtils trebuie compilat înainte de UserInterface, garantând că toate dependințele sunt rezolvate înainte de linkarea finală.

Cu `target_include_directories()`, se configuraază directoarele care vor fi incluse în calea de căutare pentru fișierele de antet necesare compilării UserInterface. Aceasta asigură că compilatorul poate accesa toate antetele necesare din ImageProcessingUtils, directorul GUI și bibliotecile OpenCV. De asemenea, se utilizează comanda

`target_link_libraries()`, care asociază `UserInterface` cu bibliotecile necesare pentru funcționalitatea sa. Aceasta include `ImageProcessingUtils` pentru procesarea imaginilor, modulele `Qt6::Widgets` și `Qt6::Core` pentru funcționalitățile interfeței grafice și bibliotecile `OpenCV` pentru procesare avansată a imaginilor.

Prin comanda `target_link_directories()`, se adaugă directorul de output al runtime-ului la calea de linkare, asigurând că executabilele și bibliotecile sunt detectabile de sistemul de operare în timpul execuției.

Aceste setări configurează integral modulul `UserInterface`, permitând-i să interacționeze eficient cu restul componentelor aplicației și să acceseze resursele necesare pentru a funcționa optim în cadrul proiectului.

Proiectului `Application` include cele două proiecte descrise anterior și este folosit în construirea executabilului. Cu ajutorul comenzi `project()`, se initializează proiectul CMake cu numele "Application".

Pentru a găsi toate fișierele header (\*.h) și sursă (\*.cpp) în directorul curent, se va folosi comanda `GLOB`. Acestea sunt stocate în variabilele `HEADER_FILES` și `SOURCE_FILES`. Apoi, comanda `add_executable()` adaugă un executabil sub numele `PROJECT_NAME` (care este "Application"), specificând că este o aplicație Windows (`WIN32`) și incluzând toate fișierele header și sursă colectate anterior. Cu `set_target_properties()` se setează proprietățile pentru ținta specificată, aici schimbând numele output-ului executabilului în "ParkingLotSecurityCamera".

Pentru a adăuga dependințele proiectului, se folosește comanda `add_dependencies()`, indicând că `Application` depinde de subproiectele `ImageProcessingUtils` și `UserInterface`. Astfel, se asigură că aceste subproiecte sunt construite înainte de `Application`. Pentru a specifica și directoarele care trebuie să fie disponibile proiectului pentru includere, este necesar ca acestea să fie incluse în comanda `target_include_directories()`. Directoarele necesare în cazul de față sunt cele de dezvoltare ale bibliotecii `OpenCV` și subdirectoarele proprii ale proiectului, mai exact `UserInterface` și `ImageProcessingUtils`.

Comanda `target_link_libraries()` leagă bibliotecile specificate la executabilul `Application`. Aceasta include linkarea la modulele Qt necesare (Core, Widgets) și bibliotecile `OpenCV`, precum și bibliotecile create în subproiecte (`ImageProcessingUtils` și `UserInterface`). Cu `target_link_directories()` se adaugă directorul specificat la locațiile în care linkerul va căuta biblioteci la runtime, asigurându-se că toate dependințele necesare sunt accesibile.

Funcția `install(TARGETS ...)` instalează executabilul proiectului în directorul `DeliveryPack` din directorul sursă al proiectului. Acesta este pasul final care face executabilul disponibil pentru distribuire. Cu ajutorul comenzi `install(FILES $<TARGET_RUNTIME_DLLS:> ...)` se instalează toate DLL-urile necesare rulării executabilului în același director. `TARGET_RUNTIME_DLLS` este un generator care listează toate DLL-urile la care executabilul face referință, asigurând că toate dependințele din timpul rulării sunt incluse.

Următoarea secțiune de cod folosește funcția `set()` pentru a defini calea către plugin-urile Qt necesare la "`Qt6_DIR`/.../.../plugins" și le instalează în directorul `DeliveryPack` folosind comanda `install(DIRECTORY ...)`. Opțiunea `USE_SOURCE_PERMISSIONS` este utilizată pentru a asigura păstrarea permisiunilor originale ale fișierelor. Comanda `file(GLOB ...)` este utilizată pentru a identifica toate fișierele DLL din directorul binar al bibliotecii `Tesseract OCR`, cu

`Tesseract_INCLUDE_DIRS` indicând calea către acest director. Fișierele identificate sunt apoi instalate în pachetul de livrare folosind `install(FILES ...)`.

Mai departe, `file(GLOB ...)` este folosită din nou pentru a găsi diverse fișiere din directorul bazei de date, inclusiv fișierul text care stochează informații despre istoricul vehiculelor, şablonul literei 'l' pentru tehnica de template matching și două imagini pentru butoanele din interfața grafică. Toate aceste fișiere sunt instalate în pachetul de livrare folosind din nou comanda `install(FILES ...)`.

Astfel, proiectul Application integrează subproiectele ImageProcessingUtils și UserInterface pentru a crea un executabil complet. Acesta colectează fișierele necesare, gestionează dependințele și leagă bibliotecile adecvate, asigurând o compilare eficientă și o integrare fără cusur. Finalizarea prin instalarea executabilului și a DLL-urilor necesare în directorul DeliveryPack pregătește aplicația pentru distribuire, transformând-o într-un produs gata de utilizare.

Separat de scripturile anterioare, acest script CMake este destinat configurării proiectului Tests, dedicat testelor unitare pentru aplicație. Inițierea proiectului se face prin comanda `project()`, care stabilește "Tests" ca nume al proiectului.

Comanda `file(GLOB ...)` este utilizată pentru a colecta automat toate fișierele sursă C++ din directorul curent, simplificând procesul de adăugare a acestora la proiect. Proiectul utilizează aceste fișiere pentru a crea o bibliotecă partajată, prin comanda `add_library()`. Această abordare favorizează reutilizarea codului și eficientizează testarea componentelor.

Funcția `target_include_directories()` specifică directoarele de căutare pentru fișierele de antet necesare utilizării bibliotecilor OpenCV și Tesseract, garantând accesul la aceste resurse în timpul compilării. Funcția `target_link_libraries()` leagă proiectul Tests de bibliotecile necesare, inclusiv ImageProcessingUtils și cele externe OpenCV și Tesseract, asigurând disponibilitatea tuturor funcționalităților pentru testare. Îar setarea `target_link_directories()` asociază directorul de output runtime, specificat prin `CMAKE_RUNTIME_OUTPUT_DIRECTORY`, facilitând localizarea executabilului și a bibliotecilor în timpul execuției testelor.

Ultima parte a configurării folosește `add_test()` pentru a defini un test specific în infrastructura CTest a CMake, specificând utilizarea `vtest.console.exe` pentru executarea testelor, cu fișierul "Tests.dll" ca sănătă și directorul de lucru specificat pentru desfășurarea testului.

Structura pregătește proiectul Tests pentru realizarea unor teste unitare eficiente.

# Capitolul 5

## Concluzii și rezultate

### 5.1 Direcții și dezvoltări viitoare

Pentru a asigura o alternativă eficientă în cazul unor erori ale sistemului principal de recunoaștere a numerelor de înmatriculare, se prevede implementarea unui sistem de backup bazat pe tickete cu coduri de bare. Această soluție va permite accesul vehiculelor în parcare auto prin scanarea unui cod de bare generat la intrare, asigurând astfel continuitatea serviciului chiar și în condiții neprevazute a sistemului principal.

De asemenea, se dorește dezvoltarea și configurarea unui model propriu de inteligență artificială, conceput special pentru a se adapta specificului și cerințelor sistemului de monitorizare a parcărilor. Acest model AI se vrea a fi dezvoltat utilizând tehnici de învățare automată și seturi de date extinse, pentru a îmbunătăți acuratețea și eficiența recunoașterii numerelor de înmatriculare. Procesul va include etape de antrenare și testare ale modelului, cu scopul de a optimiza performanța generală a sistemului.

Aceste două funcționalități dovedesc inovarea continuă pentru astfel de sisteme de gestionare a accesului în parcările auto și aduc un plus de securitate, facilitând o gestionare mai flexibilă și adaptată nevoilor utilizatorilor.

### 5.2 Concluzii

În procesul de dezvoltare a sistemului, competențele și cunoștințele legate de tehnologiile aplicate au cunoscut îmbunătățiri, dar mai ales în mod semnificativ următoarele aspecte:

- **Algoritmica specifică procesării de imagini:** S-au aprofundat tehnici care sunt esențiale pentru manipularea și analiza imaginilor
- **Programare în C++:** Abilitățile de programare în acest limbaj au fost extinse, îmbunătățind capacitatea de a scrie cod eficient și optimizat pentru procesarea datelor.
- **Utilizarea bibliotecii OpenCV:** Expertiza în folosirea acestei biblioteci esențiale pentru procesarea imaginilor a fost amplificată, facilitând implementarea de soluții eficiente.

- **Modelul de inteligență artificială Tesseract OCR:** Experiența cu acest model a crescut, îmbunătățind acuratețea recunoașterii textului din imagini.
- **Framework-ul Qt:** Cunoștințele legate de acest framework au fost extinse, dezvoltând interfețele grafice pentru utilizator.
- **Unelele CMake pentru configurarea proiectelor:** Utilizarea acestora a fost rafinată, asigurând o gestionare mai eficientă și automatizată a construcției software.
- **Implementarea testelor automate:** Competențele în testarea automată au fost aprofundate, garantând stabilitatea și funcționalitatea fiecărei componente a sistemului.
- **Doxxygen pentru documentația tehnică:** Aplicarea Doxygen a fost îmbunătățită, contribuind la atingerea unui standard în documentarea codului și facilitarea înțelegerei acestuia.
- **Mediul de integrare Visual Studio:** Experiența cu acest mediu de dezvoltare a fost consolidată, sprijinind eficiența în scrierea codului.
- **GitLab pentru versionarea codului:** Utilizarea GitLab a fost îmbunătățită, în special în gestionarea versiunilor de software.

Proiectul de automatizare a accesului în parcările auto a demonstrat o serie de îmbunătățiri semnificative în eficiența operațională, securitatea sporită și gestionarea optimizată a spațiilor de parcare.

Automatizarea procesului a redus considerabil necesitatea de personal pentru verificarea manuală a accesului și a accelerat fluxul de vehicule, minimizând aglomerațiile la intrări și ieșiri.

Monitorizarea constantă și înregistrarea detaliată a tuturor vehiculelor care accesează parcareau consolidat securitatea, facilitând identificarea rapidă a vehiculelor neautorizate sau a activităților suspecte.

Prin urmărire exactă a vehiculelor prezente, sistemul optimizează utilizarea spațiilor disponibile și sporește planificarea mai eficientă a necesarului de parcare.

De asemenea, flexibilitatea și scalabilitatea sistemului oferă posibilitatea de extindere pentru a include mai multe camere sau pentru integrarea în rețele mai mari de parcări, furnizând o soluție adaptabilă.

Prin urmare, în urma implementării soluției descrise, s-a dezvoltat un sistem eficient de recunoaștere a numerelor de înmatriculare, care nu doar că a îmbunătățit gestionarea accesului în parcările auto, dar a și consolidat semnificativ cunoștințele de programare și competențele tehnice. Această soluție integrată a demonstrat capacitatea de a procesa și analiza imagini în condiții variate, garantând o recunoaștere rapidă și precisă a numerelor de înmatriculare.

Experiența acumulată în utilizarea tehnologiilor avansate a contribuit la dezvoltarea profesională, oferind o bază solidă pentru proiectele viitoare și pentru adaptarea continuă la noutățile tehnologice.

# Capitolul 6

## Bibliografie

- [1] Rafael C. Gonzalez. Digital image processing. 2002.
  - [2] Adrian Kaehler and Gary Bradsk. Learning opencv 3. 2016.
  - [3] Richard Szeliski. Computer vision: Algorithms and applications. 2001.
  - [4] Wikipedia Contributors. C++, 2024. URL: <https://en.wikipedia.org/wiki/C%2B%2B>.
  - [5] Wikipedia Contributors. Cmake, 2024. URL: <https://en.wikipedia.org/wiki/CMake>.
  - [6] Wikipedia Contributors. Doxygen, 2024. URL: <https://en.wikipedia.org/wiki/Doxygen>.
  - [7] Wikipedia Contributors. Dynamic-link library, 2024. URL: [https://en.wikipedia.org/wiki/Dynamic-link\\_library](https://en.wikipedia.org/wiki/Dynamic-link_library).
  - [8] Wikipedia Contributors. Git, 2024. URL: <https://ro.wikipedia.org/wiki/Git>.
  - [9] Wikipedia Contributors. Integrated development environment, 2024. URL: [https://en.wikipedia.org/wiki/Integrated\\_development\\_environment](https://en.wikipedia.org/wiki/Integrated_development_environment).
  - [10] Wikipedia Contributors. Microsoft visual studio, 2024. URL: [https://ro.wikipedia.org/wiki/Microsoft\\_Visual\\_Studio](https://ro.wikipedia.org/wiki/Microsoft_Visual_Studio).
  - [11] Wikipedia Contributors. Object-oriented programming, 2024. URL: [https://en.wikipedia.org/wiki/Object-oriented\\_programming](https://en.wikipedia.org/wiki/Object-oriented_programming).
  - [12] Wikipedia Contributors. Opencv, 2024. URL: <https://en.wikipedia.org/wiki/OpenCV>.
  - [13] Wikipedia Contributors. Qt (software), 2024. URL: [https://en.wikipedia.org/wiki/Qt\\_\(software\)](https://en.wikipedia.org/wiki/Qt_(software)).
  - [14] Wikipedia Contributors. Tesseract (software), 2024. URL: [https://en.wikipedia.org/wiki/Tesseract\\_\(software\)](https://en.wikipedia.org/wiki/Tesseract_(software)).
  - [15] Wikipedia Contributors. Unit testing, 2024. URL: [https://en.wikipedia.org/wiki/Unit\\_testing](https://en.wikipedia.org/wiki/Unit_testing).
- [3] [1] [2] [4] [11] [7] [12] [14] [13] [5] [8] [9] [10] [6] [15]

# Capitolul 7

## Anexă

Nr.	Input	Numar de înmatriculare	Text întărit	Text prezis	Confidență
1		<b>DB 17 NGM</b>	DB17NGM	DB17NGM	98.97%
2		<b>DB 17 NGM</b>	DB17NGM	DB17NGM	98.92%
3		<b>DB 17 NGM</b>	DB17NGM	DB17NGM	98.91%
4		<b>DB 17 NGM</b>	DB17NGM	DB17NGM	97.48%
5		<b>BV 20HDY</b>	BV20HDY	BV20HDY	98.74%
6		<b>BV 20HDY</b>	BV20HDY	BV20HDY	98.61%
7		<b>BV 20HDY</b>	BV20HDY	BV20HDY	98.18%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
8		<b>GL 87 ZBM</b>	GL87ZBM	GL87ZBM	96.65%
9		<b>BV 56 RAU</b>	BV56RAU	BV56RAU	97.34%
10		<b>BV 56 RAU</b>	BV56RAU	BV56RAU	96.86%
11		<b>BV 56 RAU</b>	BV56RAU	BV56RAU	97.37%
12		<b>BV 28 BRI</b>	BV28BRI	BV28BRI	97.98%
13		<b>BV 29 PMN</b>	BV29PMN	BV29PMN	98.98%
14		<b>BV 29 PMN</b>	BV29PMN	BV29PMN	98.91%
15		<b>BV 60 TEO</b>	BV60TEO	BV60TEO	98.38%
16		<b>BV 60 TEO</b>	BV60TEO	BV60TEO	98.19%
17		<b>BV 81 NYC</b>	BV81NYC	BV81NYC	97.73%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
18		<b>BV 20SXK</b>	BV20SXK	BV20SXK	98.88%
19		<b>CT 36 NLA</b>	CT36NLA	CT36NLA	97.66%
20		<b>CT 36 NLA</b>	CT36NLA	CT36NLA	98.40%
21		<b>CT 36 NLA</b>	CT36NLA	CT36NLA	98.52%
22		<b>BV 18 UFN</b>	BV18UFN	BV18UFN	97.16%
23		<b>BV 18 UFN</b>	BV18UFN	BV18UFN	98.20%
24		<b>BV 37RED</b>	BV37RED	BV37RED	98.91%
25		<b>BV 37RED</b>	BV37RED	BV37RED	98.93%
26		<b>BZ 30 GIG</b>	BZ30GIG	BZ30GIG	97.20%
27		<b>CJ 25 SGL</b>	CJ25SGL	CJ25SGL	98.44%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
28		<b>AR 15 YCM</b>	AR15YCM	AR15YCM	97.88%
29		<b>AR 15 YCM</b>	AR15YCM	AR15YCM	97.28%
30		<b>B 708 VDF</b>	B708VDF	B708VDF	98.61%
31		<b>B 708 VDF</b>	B708VDF	B708VDF	98.63%
32		<b>BV 90 WLF</b>	BV90WLF	BV90WLF	98.58%
33		<b>BV 77 RSU</b>	BV77RSU	BV77RSU	98.96%
34		<b>BV 02 ALA</b>	BV02ALA	BV02ALA	99.13%
35		<b>B 730 HEX</b>	B730HEX	B730HEX	98.96%
36		<b>B 88 FBU</b>	B88FBU	B88FBU	98.89%
37		<b>B 88 FBU</b>	B88FBU	B88FBU	98.82%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
38		<b>B 88 FBU</b>	B88FBU	B88FBU	98.90%
39		<b>DB 20PMD</b>	DB20PMD	DB20PMD	98.67%
40		<b>B 496 AMI</b>	B496AMI	B496AMI	98.33%
41		<b>BV 15 ZTP</b>	BV15ZTP	BV15ZTP	97.09%
42		<b>BV 15 ZTP</b>	BV15ZTP	BV15ZTP	97.88%
43		<b>BV 20 ZZB</b>	BV20ZZB	BV20ZZB	98.21%
44		<b>BV 20 ZZB</b>	BV20ZZB	BV20ZZB	97.97%
45		<b>BV 15 TWB</b>	BV15TWB	BV15TWB	97.69%
46		<b>BV 15 TWB</b>	BV15TWB	BV15TWB	97.95%
47		<b>B 808 JDM</b>	B808JDM	B808JDM	99.12%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
48		<b>B 808 JDM</b>	B808JDM	B808JDM	99.04%
49		<b>B 800 JDM</b>	B800JDM	B800JDM	98.86%
50		<b>B 800 JDM</b>	B800JDM	B800JDM	98.62%
51		<b>BV 13 KPM</b>	BV13KPM	BV13KPM	97.88%
52		<b>BV 13 KPM</b>	BV13KPM	BV13KPM	96.99%
53		<b>BV 13 KPM</b>	BV13KPM	BV13KPM	98.45%
54		<b>BV 19 MDZ</b>	BV19MDZ	BV19MDZ	98.38%
55		<b>BV 19 MDZ</b>	BV19MDZ	BV19MDZ	97.62%
56		<b>BV 88 UMK</b>	BV88UMK	BV88UMK	98.91%
57		<b>BV 17 XSA</b>	BV17XSA	BV17XSA	98.82%

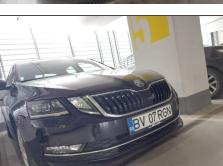
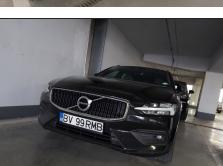
Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
58		<b>BV 17 XSA</b>	BV17XSA	BV17XSA	98.54%
59		<b>B 500FMP</b>	B500FMP	B500FMP	97.63%
60		<b>VN 61 AVM</b>	VN61AVM	VN61AVM	98.46%
61		<b>BV 88 BMG</b>	BV88BMG	BV88BMG	98.93%
62		<b>BV 88 BMG</b>	BV88BMG	BV88BMG	98.91%
63		<b>B 110 SRH</b>	B110SRH	B110SRH	98.09%
64		<b>B 110 SRH</b>	B110SRH	B110SRH	97.57%
65		<b>CV 89 COX</b>	CV89COX	CV89COX	98.83%
66		<b>CV 89 COX</b>	CV89COX	CV89COX	97.89%
67		<b>BV 99 VDV</b>	BV99VDV	BV99VDV	99.01%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
68		<b>BV 99VDV</b>	BV99VDV	BV99VDV	98.85%
69		<b>BV 32GHE</b>	BV32GHE	BV32GHE	98.85%
70		<b>BV 32GHE</b>	BV32GHE	BV32GHE	98.95%
71		<b>B 212 FAD</b>	B212FAD	B212FAD	99.08%
72		<b>B 212 FAD</b>	B212FAD	B212FAD	99.11%
73		<b>BV 01UBC</b>	BV01UBC	BV01UBC	98.28%
74		<b>NT 11 TRP</b>	NT11TRP	NT11TRP	98.16%
75		<b>B 321 KWL</b>	B321KWL	B321KWL	99.13%
76		<b>B 321 KWL</b>	B321KWL	B321KWL	99.02%
77		<b>B 321 KWL</b>	B321KWL	B321KWL	98.42%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
78		<b>BV 83ALE</b>	BV83ALE	BV83ALE	99.07%
79		<b>BV 83ALE</b>	BV83ALE	BV83ALE	98.99%
80		<b>SV 10URK</b>	SV10URK	SV10URK	98.43%
81		<b>IS 99 GTM</b>	IS99GTM	IS99GTM	98.73%
82		<b>CV 09BDL</b>	CV09BDL	CV09BDL	98.64%
83		<b>BV 50MBA</b>	BV50MBA	BV50MBA	98.50%
84		<b>BV 94MTN</b>	BV94MTN	BV94MTN	98.80%
85		<b>BV 18 GBT</b>	BV18GBT	BV18GBT	98.80%
86		<b>BV 26USA</b>	BV26USA	BV26USA	98.83%
87		<b>BV 43BDL</b>	BV43BDL	BV43BDL	98.31%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
88		<b>BV 43BDL</b>	BV43BDL	BV43BDL	98.23%
89		<b>CV 07BJM</b>	CV07BJM	CV07BJM	99.04%
90		<b>CV 07BJM</b>	CV07BJM	CV07BJM	98.64%
91		<b>BV 30PAG</b>	BV30PAG	BV30PAG	98.90%
92		<b>BV 30PAG</b>	BV30PAG	BV30PAG	98.48%
93		<b>BV 63TOY</b>	BV63TOY	BV63TOY	98.15%
94		<b>BV 63TOY</b>	BV63TOY	BV63TOY	97.95%
95		<b>BV 16 EWM</b>	BV16EWM	BV16EWM	98.50%
96		<b>BV 16 EWM</b>	BV16EWM	BV16EWM	98.01%
97		<b>VN 97CRU</b>	VN97CRU	VN97CRU	98.91%

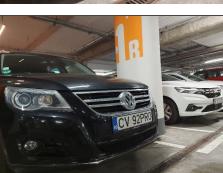
Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
98		<b>VN 97CRU</b>	VN97CRU	VN97CRU	98.78%
99		<b>BV 13 LIE</b>	BV13LIE	BV13LIE	97.41%
100		<b>B 110 NPA</b>	B110NPA	B110NPA	98.34%
101		<b>B 110 NPA</b>	B110NPA	B110NPA	98.66%
102		<b>BV 26BBZ</b>	BV26BBZ	BV26BBZ	97.90%
103		<b>BV 26BBZ</b>	BV26BBZ	BV26BBZ	97.29%
104		<b>B 675 GEO</b>	B675GEO	B675GEO	98.29%
105		<b>B 675 GEO</b>	B675GEO	B675GEO	96.96%
106		<b>BV 24 CAI</b>	BV24CAI	BV24CAI	98.87%
107		<b>BV 24 CAI</b>	BV24CAI	BV24CAI	96.98%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
108		<b>BV 17 CLT</b>	BV17CLT	BV17CLT	98.68%
109		<b>BV 17 CLT</b>	BV17CLT	BV17CLT	98.80%
110		<b>BV 19 BRO</b>	BV19BRO	BV19BRO	98.80%
111		<b>BV 19 BRO</b>	BV19BRO	BV19BRO	98.25%
112		<b>BV 97 DIA</b>	BV97DIA	BV97DIA	97.97%
113		<b>BV 97 DIA</b>	BV97DIA	BV97DIA	97.11%
114		<b>BV 07 RGN</b>	BV07RGN	BV07RGN	98.65%
115		<b>BV 99 RMB</b>	BV99RMB	BV99RMB	98.90%
116		<b>BV 99 RMB</b>	BV99RMB	BV99RMB	99.05%
117		<b>BV 41 AIG</b>	BV41AIG	BV41AIG	96.82%

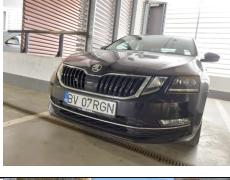
Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
118	A photograph of an orange Volkswagen SUV parked in a parking garage. The license plate area is obscured by a black redaction box.	<b>BV 17 CCO</b>	BV17CCO	BV17CCO	98.75%
119	A photograph of an orange Volkswagen SUV parked in a parking garage. The license plate area is obscured by a black redaction box.	<b>BV 17 CCO</b>	BV17CCO	BV17CCO	98.63%
120	A photograph of a black Hyundai car parked in a parking garage. The license plate area is obscured by a black redaction box.	<b>BV 17 NFU</b>	BV17NFU	BV17NFU	97.43%
121	A photograph of a grey Volkswagen car parked in a parking garage. The license plate area is obscured by a black redaction box.	<b>BV 62MDL</b>	BV62MDL	BV62MDL	98.06%
122	A photograph of a grey Toyota car parked in a parking garage. The license plate area is obscured by a black redaction box.	<b>B 520 DGY</b>	B520DGY	B520DGY	98.39%
123	A photograph of a grey Toyota car parked in a parking garage. The license plate area is obscured by a black redaction box.	<b>B 520 DGY</b>	B520DGY	B520DGY	98.50%
124	A photograph of a blue Volkswagen car parked in a parking garage. The license plate area is obscured by a black redaction box.	<b>BV 92 BBU</b>	BV92BBU	BV92BBU	98.99%
125	A photograph of a blue Volkswagen car parked in a parking garage. The license plate area is obscured by a black redaction box.	<b>BV 92 BBU</b>	BV92BBU	BV92BBU	99.02%
126	A photograph of a dark grey Volkswagen car parked in a parking garage. The license plate area is obscured by a black redaction box.	<b>VN 99 CZJ</b>	VN99CZJ	VN99CZJ	97.41%
127	A photograph of a red Skoda car parked in a parking garage. The license plate area is obscured by a black redaction box.	<b>BV 28 NIM</b>	BV28NIM	BV28NIM	98.64%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
128		<b>BV 28 NIM</b>	BV28NIM	BV28NIM	97.73%
129		<b>BV 83 ARC</b>	BV83ARC	BV83ARC	99.00%
130		<b>BV 83 ARC</b>	BV83ARC	BV83ARC	99.03%
131		<b>BV 02 LTV</b>	BV02LTV	BV02LTV	98.62%
132		<b>BV 70 NOA</b>	BV70NOA	BV70NOA	98.01%
133		<b>BV 70 NOA</b>	BV70NOA	BV70NOA	98.65%
134		<b>BV 14 LEI</b>	BV14LEI	BV14LEI	95.51%
135		<b>BV 14 LEI</b>	BV14LEI	BV14LEI	96.56%
136		<b>BV 61 MTS</b>	BV61MTS	BV61MTS	98.45%
137		<b>BV 61 MTS</b>	BV61MTS	BV61MTS	98.69%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
138		<b>BV 77MRU</b>	BV77MRU	BV77MRU	98.88%
139		<b>B 77 RHY</b>	B77RHY	B77RHY	99.09%
140		<b>B 77 RHY</b>	B77RHY	B77RHY	99.12%
141		<b>B 77 RHY</b>	B77RHY	B77RHY	95.17%
142		<b>AG 95 BFG</b>	AG95BFG	AG95BFG	98.62%
143		<b>AG 95 BFG</b>	AG95BFG	AG95BFG	97.17%
144		<b>BV 70 AAC</b>	BV70AAC	BV70AAC	99.12%
145		<b>BV 70 AAC</b>	BV70AAC	BV70AAC	99.02%
146		<b>BV 03 BYT</b>	BV03BYT	BV03BYT	98.92%
147		<b>CV 92 NAT</b>	CV92NAT	CV92NAT	99.11%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
148		<b>AG 93 WAY</b>	AG93WAY	AG93WAY	99.16%
149		<b>AG 93 WAY</b>	AG93WAY	AG93WAY	99.16%
150		<b>BV 19 PDM</b>	BV19PDM	BV19PDM	98.91%
151		<b>BV 19 PDM</b>	BV19PDM	BV19PDM	98.77%
152		<b>BV 12 PDM</b>	BV12PDM	BV12PDM	98.48%
153		<b>BV 45 MCR</b>	BV45MCR	BV45MCR	98.73%
154		<b>CV 92 PRO</b>	CV92PRO	CV92PRO	97.72%
155		<b>B 211 ABV</b>	B211ABV	B211ABV	98.80%
156		<b>IS 03 BMI</b>	IS03BMI	IS03BMI	96.44%
157		<b>CV 07 BHS</b>	CV07BHS	CV07BHS	98.76%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
158		<b>CV 05 DIW</b>	CV05DIW	CV05DIW	91.78%
159		<b>BV 50 MSL</b>	BV50MSL	BV50MSL	98.14%
160		<b>BV 09 FVK</b>	BV09FVK	BV09FVK	98.62%
161		<b>AG 95 NRS</b>	AG95NRS	AG95NRS	97.60%
162		<b>BV 77 RSU</b>	BV77RSU	BV77RSU	98.84%
163		<b>BV 77 ZZP</b>	BV77ZZP	BV77ZZP	98.29%
164		<b>BV 17 PVU</b>	BV17PVU	BV17PVU	98.69%
165		<b>IF 37 MRY</b>	IF37MRY	IF37MRY	98.26%
166		<b>BV 94 TRT</b>	BV94TRT	BV94TRT	98.19%
167		<b>BV 30 PLK</b>	BV30PLK	BV30PLK	98.87%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
168		<b>BV 07 RGN</b>	BV07RGN	BV07RGN	93.96%
169		<b>BC 74 DNS</b>	BC74DNS	BC74DNS	97.35%
170		<b>BC 74 DNS</b>	BC74DNS	BC74DNS	91.62%
171		<b>BC 89 BEJ</b>	BC89BEJ	BC89BEJ	98.99%
172		<b>BC 89 BEJ</b>	BC89BEJ	BC89BEJ	99.06%
173		<b>BC 89 MSW</b>	BC89MSW	BC89MSW	99.04%
174		<b>BC 89 MSW</b>	BC89MSW	BC89MSW	99.11%
175		<b>BC 99 CSI</b>	BC99CSI	BC99CSI	98.45%
176		<b>BC 99 CSI</b>	BC99CSI	BC99CSI	98.09%
177		<b>B 154 HIL</b>	B154HIL	B154HIL	97.15%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
178		<b>B 154 HIL</b>	B154HIL	B154HIL	97.05%
179		<b>BC 21GUZ</b>	BC21GUZ	BC21GUZ	98.39%
180		<b>BC 21GUZ</b>	BC21GUZ	BC21GUZ	98.55%
181		<b>BC 27 TDI</b>	BC27TDI	BC27TDI	97.97%
182		<b>BC 05 MKA</b>	BC05MKA	BC05MKA	98.32%
183		<b>BC 05 MKA</b>	BC05MKA	BC05MKA	97.96%
184		<b>BC 61BSV</b>	BC61BSV	BC61BSV	98.41%
185		<b>BC 61BSV</b>	BC61BSV	BC61BSV	98.54%
186		<b>BC 27LYV</b>	BC27LYV	BC27LYV	98.78%
187		<b>BC 03VEL</b>	BC03VEL	BC03VEL	98.80%

Nr.	Input	Numar de înmatriculare	Text întă	Text prezis	Confidență
188		<b>BC 77NDN</b>	BC77NDN	BC77NDN	98.60%
189		<b>BC 77NDN</b>	BC77NDN	BC77NDN	99.13%
190		<b>BC 23 NIN</b>	BC23NIN	BC23NIN	98.26%
191		<b>BC 23 NIN</b>	BC23NIN	BC23NIN	97.64%
192		<b>BC 23 NIN</b>	BC23NIN	BC23NIN	97.90%