Procesare de Imagini

Detecţia plăcuţelor cu numerelor de înmatriculare de România ale autovehiculelor

Coldea Cosmin

Grupa 30235

**Overview:**

Acest proiect are ca scop dezvoltarea unui algoritm pentru detectarea automata a placutelor de inmatriculare romanesti din imagini color, capturate intr-o perspectiva frontala. Se va folosi C++ si biblioteca OpenCV.

Detectarea automata a placutelor de inmatriculare este un element esential in sistemele moderne de supraveghere a traficului, avand aplicatii in:

* Monitorizarea traficului rutier – identificarea vehiculelor in zonele publice;
* Parcari inteligente – permiterea accesului pe baza numarului de inmatriculare;
* Control de acces – automatizarea barierelor in zone securizate;
* Sisteme de taxare automata – utilizate in autostrazi si zone de taxare.

Pentru a realiza acest lucru, algoritmul va parcurge mai multe etape esentiale, inclusiv:

* Preprocesarea imaginii pentru reducerea zgomotului si imbunatatirea contrastului.
* Detectarea contururilor si extragerea regiunilor de interes, utilizand tehnici precum operatorii Sobel si Canny.
* Selectia si validarea regiunilor candidate pe baza dimensiunilor si proportiilor caracteristice ale placutelor.
* Vizualizarea rezultatelor prin evidentierea placutei detectate.

**Provocari si solutii abordate:**

* Variatia conditiilor de iluminare – se vor utiliza tehnici precum histogram equalization si adaptive thresholding.
* Prezenta altor elemente similare – filtrarea regiunilor pe baza aspect ratio-ului si a dimensiunilor caracteristice placutelor.
* Fundal zgomotos – utilizarea transformarilor morfologice pentru eliminarea componentelor nedorite.
* Diferente de perspectiva – sistemul va fi testat pe imagini capturate din unghiuri diferite pentru a creste robustetea.

**Project Plan:**

Implementarea proiectului se va realiza in mai multe etape:

1. **Preprocesarea imaginii:**

Aceasta etapa presupune conversia imaginii in tonuri de gri (grayscale), mai apoi aplicarea filtrarii Gaussiene pentru reducerea zgomotului si in final cresterea contrastului prin histogram equalization.

1. **Detectarea contururilor si selectia regiunilor candidate:**

Pentru aceasta vom aplica operatorul Sobel/Canny pentru detectarea marginilor, dupa care vom identifica contururile si vom selecta regiunile de interes pe baza formei si dimensiunilor, iar in final vom extrage bounding box-urile pentru zonele c ear putea fi posibile placate de inmatriculare.

1. **Filtrarea si validarea regiunilor candidate:**

Vom elimina regiunile irelevante bazate pe raportul latime/inaltime. Vom aplica transformarea morfologica(closing) pentru imbunatatirea structurii placutei, iar in final vom valida folosind OCR.

1. **Testare si optimizare:**

Testarea pe un set diversificat de imagini cu unghiuri si conditii diferite de iluminare si compararea diferitelor metode de filtrare si selectie a regiunilor.

**Posibile provocari si solutii:**

**Fundal complex:** Se va aplica filtrare morfologica pentru reducerea zgomotului.

**Luminozitate slaba:** Se va folosi adaptative histogram equalization.

**Detectari false:** Optimizarea parametrilor de filtrare si folosirea unui clasificator ML/CNN.

**Implementation:**

Se va folosi C++ si OpenCV pentru procesarea imaginii. Principalele functii ale proiectului sunt:

**Functia/modulul de preprocesare** care realizeaza conversia imaginii la gri, aplica filtrele si histogram equalization.

**Functia/modulul de detectie a marginilor** unde se aplica operatorii Sobel/Canny pentru extragerea contururilor.

**Functia/Modulul de selectie a regiunilor candidate** unde se analizeaza contururile si se selecteaza bounding box-urile relevante.

**Functia/Modulul de validare si extragere a numerelor.**

**Testing:**

Vom avea mai multe seturi de teste ce vizeaza:

Detectarea corecta a placutei in imagini clare si bine illuminate.

Rezultate in imagini cu lumina scazuta sau umbre.

Testarea algoritmului pe unghiuri diferite ale camerei.

**Improvements:**

Optimizarea selectiei regiunilor.

Adaugarea unui clasificator pentru filtrarea detectarilor false.

Testarea pe un set mare de imagini.

**Algoritmi si tehnici folosite:**

**Filtrarea Gaussiana** (Gaussian Blur): este o metoda utilizata pentru reducerea zgomotului si netezirea imaginii esentiala inainte de detectarea marginilor. Functioneaza aplicand un filtru gaussian asupra fiecarui pixel di imagine.

Aceasta evita detectarea falsa a marginilor si imbunatateste acuratetea detectarii placutelor.

**Histogram Equalization**: imbunatateste contrastul imaginii distribuend uniform intensitatile pixelilor. Este utila in special in cazul imaginilor slab iluminate.

Aceasta tehnica consta in construirea histogramei imaginii si aplicarea unei functii de transformare pentru a redistribui intensitatile astfel incat pixelii intunecati devin mai luminosi, iar cei prea luminosi sunt echilibrati.

Aceasta este utilizata pentru a creste vizibilitatea detaliilor si pentru a face imaginea mai clara pentru procesare.

**Operatorul Sobel:** detecteaza marginile unei imagini calculand derivatele partiale in directiile X si Y. Este utilizat pentru a detecta structura placutelor si permite separarea obiectelor de fundal.

Operatorul Canny: este unul dintre cei mai puternici detector de margini folosind un process in patru etape: filtrare gaussiana, calcularea gradientului, non-maximun suppression si hysteresis thresholding. Este optim pentru detectarea obiectelor.

**Bounding Box** (Delimitarea regiunilor): Pentru a izola placutele de inmatriculare folosim cv::boundingRect() care determina cel mai mic dreptunghi care continue un contur.

Mod de functionare: Se aplica cv::findContours() pentru a gasi regiunile conturate, apoi se verifica dimensiunile si raportul de aspect pentru a selecta doar placutele. In final se deseneaza bounding box in jurul acestei regiuni identificate.

Este utilizata pentru ca permite izolarea placutelor de restul imagii si necesita putina putere de calcul.

**Implementare:**

Pipeline-ul folosit:

Conversie in tonuri de gri: imaginea originala color e convertita la grayscale pentru a simplifica procesarea.

Filtru Gaussian: aplica un filtru Gaussian (kernel 5x5, sigma=1.0) pentru a reduce zgomotul si variatiile mici de intensitate.

Binarizare: prag fix (threshold=100, se poate modifica) aplicat imaginii blurate pentru a evidentia marginile clare.

Detectia marginilor: operatorul Canny este aplicat pe imaginea binara pentru a obtiine contururile.

Dilatarea: folosim o dilatare cu kernel implicit pentru a inchide eventualele spatii mici din contururi si pentru a consolida marginea placutei.

Detectarea placutei:

Metoda 1-Detectare bazata pe contur: se cauta contururi mari dreptunghiulare cu ratio intre 2.5 si 5.5. Daca este gasit un astfel de contur, se considera ca placuta a fost detectata.

Metoda 2-Detectare bazata pe litere: daca metoda 1 nu reuseste se identifica litere individuale si sunt grupate in functie de proximitate si raportul de aspect al grupului, pentru a deduce pozitia placutei.

Proiectul are un fisier .cpp pentru functii folosite pentru implementarea proiectului si un fisier main in care acestea sunt folosite conform pipeline-ului.

Functii:

Initial am implementat filtrul Sobel si Histogram Equalization insa rezultatele obtinute in urma folosirii filtrului Canny, am ales sa raman cu aceasta varianta. Prin urmare, functiile folosite in acest pipeline sunt:



Aceasta functie ia ca parametru o matrice sursa (grayscale), o dimensiune a kernelului si o valoare sigma (deviatia standard a distributiei gaussiene, ce controleaza cat de tare „se întinde” blur-ul).

Scopul ei este de a aplica un filtru Gaussian implementat manual care estompeaza imaginea pentru a reduce zgomotul si detaliile mici.

Se creeaza o matrice de kernel pe baza distributiei gaussiene, iar fiecare pixel nou devine o medie ponderata a vecinilor sai cu greutati date de kernel-ul gaussian.



Este o implementare a binarizarii ce converteste imaginea grayscale intr-o imagine alb-negru ce primeste ca parametru imaginea sursa (grayscale) si un prag (threshold) intre 0-255 care determina daca pixelul devine alb sau negru. Aceasta functie simplifica imaginea si evidentiaza doar regiunile cu contrast mare, ignorand detalii minore).



Se aplica operatorul Canny pentru detectia marginilor. Primeste o imagine grayscale binarizata anterior, un prag inferior si unul superior pentru histerezis.

Mdoul de functionare Canny: La fiecare pixel se calculeaza gradientul (schimbarea intensitaii) folosind filtre sobel pe axele X si Y. Pentru fiecare pixel pastram gradientul doar daca este maxim pe directia gradientului. Scopul este subtierea marginii la o linie de 1 pixel.

Avem 2 praguri. Daca un pixel are valoare > high e sigur margine iar daca e < low nu e margine. Daca e intre praguri se va pastra doar daca e conectat la o margine mai puternica.



Aceasta functie realizeaza dilatarea unei imagini binare folosind un kernel patrat de dimensiune aleasa.

**Parametri:**

* const cv::Mat& src: imaginea binara (0 sau 255) pe care aplicam dilatarea.
* int kernelSize: dimensiunea kernel-ului.
* int iterations: cate iteratii de dilatare sa aplicam.

**Algoritm:**

Initializam rezultatul cu imaginea originala.

Calculam "jumatatea" kernel-ului. Daca kernel-ul este 3x3, k va fi 1 (adica vom analiza vecinii la o distanta de +1/-1 fata de pixelul curent).

Iteram pentru fiecare iteratie de dilatare:

* Folosim un for: for (int iter = 0; iter < iterations; iter++)
* Cream o copie temporara care se foloseste pentru a nu modifica pixelii deja procesati.

Parcurgem fiecare pixel (y, x) din imagine (excludem marginile):

* for (int y = k; y < src.rows - k; y++)
* for (int x = k; x < src.cols - k; x++)
* Evitam marginile pentru a nu iesi din matrice.

Cautam valoarea maxima in jurul pixelului (y, x):

* Initializam uchar maxVal = 0;
* Apoi parcurgem fereastra locala:
  + for (int i = -k; i <= k; i++)
  + for (int j = -k; j <= k; j++)
  + Citim fiecare vecin si verificam daca este mai mare decat maxVal si actualizam.

Setam valoarea pixelului central:

* Daca macar un pixel din fereastra locala este "obiect" (255), pixelul central devine si el 255.
* Daca toti sunt 0, pixelul ramane 0.

La finalul fiecarei iteratii, imaginea rezultata este actualizata si pregatita pentru o noua iteratie daca mai sunt necesare.

Returnam rezultatul final cu return result;

**Scop:**

* Aceasta metoda mareste zonele „obiect” dintr-o imagine (de exemplu, margini sau forme binare).
* Este utila pentru conectarea contururilor intrerupte si pentru inchiderea golurilor mici din obiecte.



**Parametri:**

* const cv::Mat& processed: imaginea pe care s-a aplicat deja Canny/dilatare (alb-negru, contururi evidentiate);
* cv::Mat& output: copia imaginii initiale, pe care vom desena dreptunghiurile gasite.

**Algoritm:**

* std::vector<std::vectorcv::Point> contours; un vector de contururi: fiecare contur este o lista de puncte (pixeli).
* findContours(...) functia OpenCV care detecteaza contururile si le stocheaza in contours.
* std::vectorcv::Rect letterCandidates; lista finala cu dreptunghiuri care incadreaza fiecare litera detectata.
* double totalHeight = 0.0; ➔ suma inaltimilor tuturor contururilor (pentru a calcula media ulterior).

Loop prin contururi:

for (const auto& contour : contours)

{ cv::Rect rect = boundingRect(contour); totalHeight += rect.height; }

* cv::Rect rect = boundingRect(contour); gaseste cel mai mic dreptunghi care incapsuleaza conturul curent.
* avgHeight: media inaltimii tuturor contururilor. Daca nu exista contururi, este 0.

**Stabilirea filtrelor adaptive:**

if (avgHeight > 80)

{ minHeight = 20; maxHeight = 350; minWidth = 10; maxWidth = 150; }

else

{ minHeight = 15; maxHeight = 120; minWidth = 5; maxWidth = 60; }

* Daca media este mare (>80 px), presupunem ca avem un close-up si permitem litere mai mari;
* Altfel, aplicam filtre mai stricte.

**Filtrarea contururilor:**

if (rect.height > minHeight && rect.height < maxHeight && rect.width > minWidth && rect.width < maxWidth)

* Eliminam zgomotul (obiecte prea mici);
* Excludem obiecte prea mari care nu au cum sa fie litere.

Desenare:

rectangle(output, rect, cv::Scalar(0, 255, 0), 1);

Deseneaza dreptunghiul verde pe output pentru fiecare litera detectata.



**Parametri:**

* const std::vectorcv::Rect& letterRects: lista de dreptunghiuri ale literelor detectate;
* cv::Mat& output: imaginea pe care vom desena dreptunghiul placutei deduse.

**Variabile:**

if (letterRects.empty()) { ... } Daca nu exista litere detectate, returnam direct un dreptunghi gol.

Constantele de filtrare:

* const int radius = 150; distanta maxima intre litere pentru a fi considerate parte din acelasi grup.
* const int minLetters = 5; numarul minim de litere care formeaza o placuta.
* const int maxLetters = 8; numarul maxim de litere admise.
* const float minRatio = 2.5f; aspect ratio minim al grupului.
* const float maxRatio = 5.5f; aspect ratio maxim.

Loop-ul principal:

for (size\_t i = 0; i < letterRects.size(); i++)

{ std::vectorcv::Rect currentGroup; cv::Rect base = letterRects[i];

Loop-ul interior (cautam vecini):

A computer screen shot of a program code

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.La final: Daca am gasit un grup valid: rectangle(output, bestPlate, cv::Scalar(255, 0, 0), 2); return bestPlate;



**Parametri:**

* const cv::Mat& processed: imaginea procesata (dupa dilatare), alb-negru;
* cv::Mat& output: copia imaginii originale, pe care se va desena dreptunghiul placutei.

**Variabile explicate:**

* std::vector<std::vectorcv::Point> contours; vector care contine toate contururile gasite.
* cv::Rect bestPlate; dreptunghiul final al placutei gasite.
* float bestRatioDiff = 1e9; variabila care pastreaza cea mai mica diferenta fata de aspect ratio-ul ideal (~4:1).

Loop-ul principal:

for (const auto& contour : contours)

{ cv::Rect rect = boundingRect(contour);

Filtru dimensiuni: if (rect.width < 100 || rect.height < 30) continue;

Calcul aspect ratio: float ratio = static\_cast(rect.width) / rect.height;

Verificare aspect ratio:

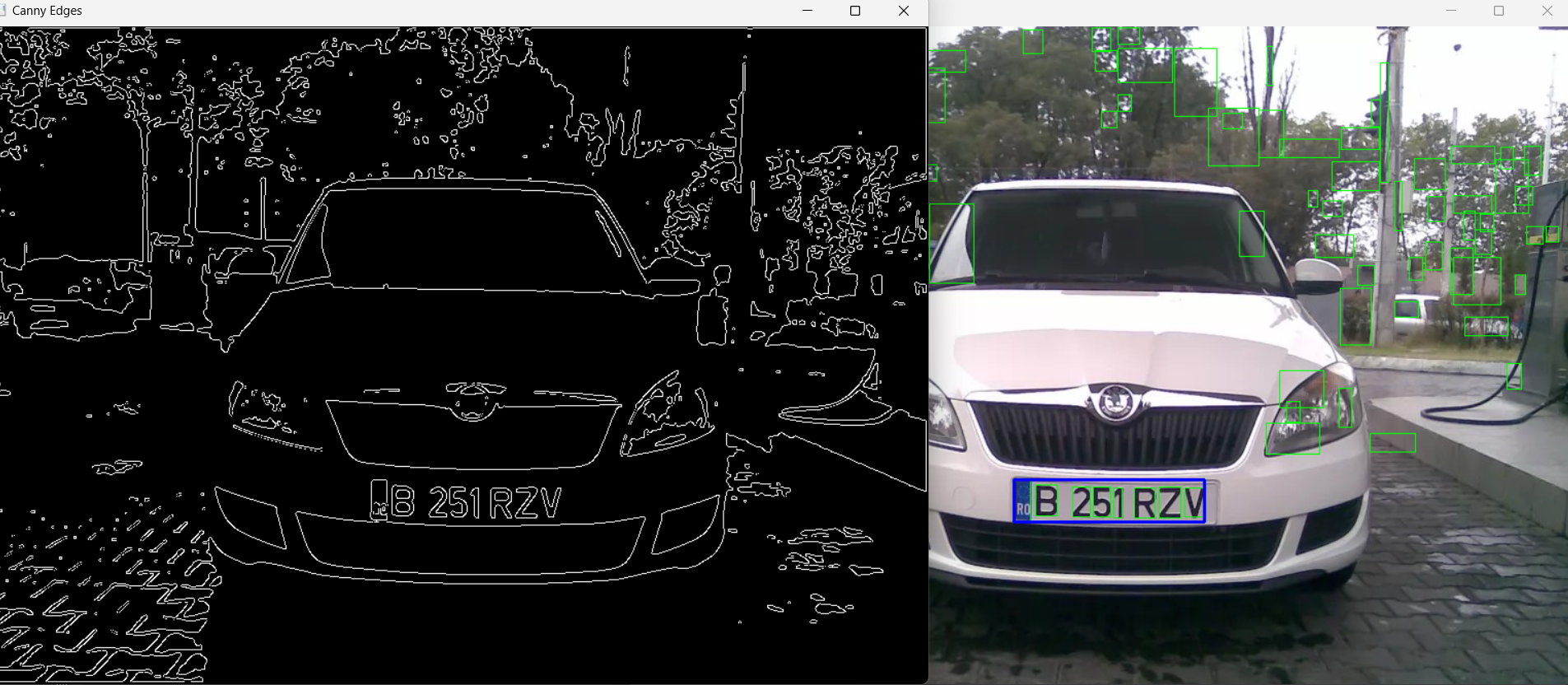
if (ratio > 2.5 && ratio < 5.5)

{ float ratioDiff = std::abs(ratio - 4.0f); if (ratioDiff < bestRatioDiff) { bestRatioDiff = ratioDiff; bestPlate = rect; } }

La final: Daca gasim o placuta (bestPlate.area() > 0), desenam cu galben: rectangle(output, bestPlate, cv::Scalar(0, 255, 255), 2);

**Exemple:**

In primul exemplu, dupa aplicarea filtrului Canny, conturul placutei nu este bine definit, in schimb literele sunt foarte bine delimitate. Astfel aplicam metoda de detectie a literelor mentionata anterior, apoi le grupam dupa criteriile descrise. Cautam un grup intre 5-8 litere (functia de detective a literelor gaseste candidati, nu litere efectiv), care au aproximativ aceeasi forma (dupa height si width al conturului) si sunt grupate intr-o distanta relative mica una de cealalta. Sunt gasite mai multe astfel de grupari de “litere/caractere”, insa vom selecta gruparea cea mai apropiata de ratio-ul cautat. Aceea va fi placuta de inmatriculare.



In al doilea exemplu avem un contur al placutei mult mai bine determinat astfel incat putem aplica direct functia detectPlateContour care selecteaza conturul cu cel mai bun ratio.

A close-up of a car

AI-generated content may be incorrect.

**Precizari:**

Am folosit operatorul Canny implementat in OpenCV deoarece varianta implementata manual nu are rezultate la fel de bune in acest moment. De asemenea, varianta Canny ofera rezultate mai bune decat cea Sobel impreuna cu histogram equalization.

Acest model de implementare a fost testat si functioneaza pentru poze frontale, in care placuta este vizibila. Proiectul nu a fost testat pentru edge case-uri, de exemplu fotografii facute la diferite unghiuri, in care placuta nu este complet vizibila.