Universitatea Tehnică ”Gheorghe Asachi” Iași

Facultatea de Automatică și Calculatoare

Detecția plăcuțelor de înmatriculare

Studenți:

Chelarașu Elena-Denisa, 1308B

Miron Alexandru, 1308B

Professor:

Marius Gavrilescu

Github:

Prelucrarea Imaginilor - Proiect

Rezumat

Obiectivele proiectului:

- Segmentarea plăcii (plăcilor) de înmatriculare
- Segmentarea literelor din placă
- Citirea literelor
- Extragerea altor informații utile

Metode existente:

- Machine learning pentru detecție plăcuțelor
  - Metodă versatile, dar trebuie de antrenat un model
  - Nu ține de Prelucrarea Imaginilor
- OCR pentru plăcuțe
  - Tesseract v3 - metodă clasică de detecție a caracterelor
  - Tesseract v4 - neutral net
  - Vom opta pentru implementarea proprie
- GitHub Topic - License Plate Recognition

Rezultate experimentale și concluzii:

- Algoritmii simpli sunt sensitivi la schimbări
- Elaborarea unui algoritm flexibil cere metode complexe de prelucrarea a imaginilor
- Determinarea similitudinii între imagini (două litere, etc.) este dificilă. Este necesar de a elabora mai multe criterii de distanță între imagini, și de a le folosi pentru determinarea literii recunoscute
- Criteriul de distanță după pixeli are o acuratețe decentă, dar e senzitiv la distorsiuni
- Criteriul de gradient poate fi util pentru imagini distorsionate, dar are o acuratețe slabă în lipsa unui algoritm complex
- Literele similare (formă, contur) sunt greu de diferențiat (5 cu S, B cu D, 9 cu 6, 8 cu 0, etc.)
- Pot apărea cazuri de tip **false-positive** (este segmentată o regiune care nu e o plăcuță sau literă), și **false-negative** (nu sunt recunoscute plăcuțele sau literele)

Introducere

Interesul pentru metodele de prelucrare a imaginilor digitale își are rădăcinile în două idei:

- îmbunătățirea informației vizuale pentru interpretarea umană
- procesarea imaginilor pentru stocare, transmitere și ulterioara detecție a informației de către algoritmi specializați

În zilele noastre, procesarea imaginilor este folosită în foarte multe domenii. Spre exemplu, procesarea imaginilor se aplică în medicină (detecția cancerului, detecția organelor, vaselor sangvine, etc.), automotive (detecția plăcilor de înatriculare, automobile self-driving, etc.), și alte domenii.

În cadrul acestui proiect, ne propunem să realizăm detecția plăcuțelor de înmatriculare în imagini. Unele aplicări ale acestui algoritm sunt camerele pentru viteză, parcări private cu număr fix de locuri, vinietă, taxa de pod automată, etc.

OpenCV

OpenCV este o librărie open source, orientată spre vederea computerizată în timp real. Librăria oferă diverse toolkituri pentru lucrul cu 2D și 3D, recunoașterea fețelor, gesturilor, obiectelor, urmărirea mișcărilor, facilități pentru realizarea interfețelor grafice, și altele.

În cadrul proiectului, vom folosi OpenCV pentru facilitarea operațiilor de filtrare, de detecție a contururilor, și de editări ale imaginilor.

Optical Character Recognition (OCR)

OCR reprezintă translatarea imaginilor ce conțin text printat, scris de mână sau tipărit în text editabil.

OCR este un domeniu de cercetare în recunoașterea modelelor, inteligența artificială și vederea mecanică. Recunoașterea optică a caracterelor (folosind metode optice ca oglinzi și lentile) și recunoașterea digitală a caracterelor (folosind scanere și algoritmi pe calculator) au fost, inițial, considerate domenii diferite. Deoarece puține aplicații folosesc tehnici optice, termenul OCR include și procesarea digitală a documentelor.

Programele inițiale necesitau învățarea caracterelor (exemple ale fiecărui caracter) pentru a identifica un font specific. Astăzi există programe „inteligente” care au un grad mai mare de acuratețe, putând identifica majoritatea fonturilor. Unele programe sunt chiar capabile de a aranja textul pe coloane, imaginile și elementele non-textuale în pagină aproape identic cu sursa originală.

Scurtă descriere a aplicației

Aplicația constă în consola de la Visual Studio în care utilizatorul trebuie să introducă calea către imaginea ce se dorește a fi analizată. La apăsarea tastei ENTER, în consolă vor apărea date reprezentative precum literele obținute din "metoda cu pixeli", litera obținută cu magnitudinea și orientația (gradient-ul imaginii).

Pe lângă afișările din consolă, mai apare imaginea originală în care plăcuța de înmatriculare este conturată pentru a se observa pe ce zonă s-au aplicat algoritmii. Din ace zonă, se preiau literele detectate și se deschid n ferestre, unde n este numărul de caractere detectate.

Obiective realizate în etapa intermediară

- Detecția plăcuțelor de înmatriculare - plăcuțele detectate sunt încadrate într-un contur verde
- Segmentarea literelor de pe plăcuță
- Detecția literelor prin prelucrare manuală a segmentelor (fară OCR)

Obiective realizare în etapa finală

- Îmbunătățirea algoritmilor de detecție a plăcuțelor și a literelor
    - Creșterea acurateței
    - Viabilitate și în condiții nefavorabile (imagini cu zgomot, distorsionate, cu rezoluție mică)
    - Implementarea unor criterii adiționale pentru recunoașterea literelor - gradientul literelor

Algoritmi implementați

- pi::simplifyContours() - simplificarea unui contur obținut din cv::findContours()
  - cv::findContours() returnează o listă de puncte ce pot fi aproximate în mai multe linii drepte
  - Se face o iterație pe acea listă, selectând câte 3 puncte consecutive (A, B, C)
  - Se elimină punctul B din listă pentru fiecare grup care îndeplinește una din condiții:
    - len(AB) sau len(BC) e sub un prag prestabilit - B poate fi aproximat ca fiind identic cu A sau C
    - unghiul dintre AB și BC e apropiat de 180 grade - segmentul AC aproximează grupul ABC
  - Se selectează și grupe (A, B, C, D)
  - Se reduce segmentul BC la punctul de mijloc (X), dacă:
    - Segmentul BC e cu mult mai mic față de AB și CD
    - unghiurile ABC și BCD sunt apropiate de 180 de grade (folosind un prag mai relaxat decât la punctul de sus)
  - Se repetă pașii precedenți pentru mai multe iterații, pentru a simplifica și segmentele noi formate
- pi::isLikeALicensePlate()
  - Returnează True dacă conturul dat ca parametru întrunește următorele condiții:
    - Are 4 puncte în total
    - Unghiul între două laturi este apropiat de 90 grade
    - Laturile adiacente au o proporție de 40 / 9 (plăcuțe UE)
- pi::getImageDistance()
  - Algoritmul calculează distanța între două imagini ca fiind un număr între 0.0 și 1.0
  - Iterăm prin ambele imagini simultan folosind doi pași flotanți pentru fiecare imagine - unul pentru rânduri, altul pentru coloane. La fiecare etapă a iterației, incrementăm o variabilă contor.

- Imaginea cu lățimea (înălțimea) mai mare va avea un pas de 1.0 pentru coloane (rânduri), iar cea mai mică va avea un pas subunitar: înălțimea mică / înălțimea mare (lățimea mică / lățimea mare). Efectiv, o imagine este ”mapată” peste altă imagine
- Pentru fiecare etapă a iterației, se calculează coordonatele discrete în imagini, și se extrag câte un pixel. Valoarea abs(pixel1 - pixel2) / 255.0 este adăugată la o variabilă total.
- Iterația se termină când indexul flotant pentru rânduri depășește numărul de rânduri al imaginii respective
- Totalul astfel calculat este împărțit la variabila contor, și rezultatul reprezintă distanța între cele două imagini. Distanța crește cu cât pixelii din regiuni corespunzătoare diferă mai puternic în luminozitate

- pi::contour_gradient()
  - calculează gradientul unei imagini
    - se calculează derivatele în direcțiile x și y cu ajutorul kernelurilor de tip Sobel de 3x3
    - se calculează magnitudinea și orientația cu ajutrul celor două funcții `calculate_orientation()` și `calculate_magnitude()`
    - se combină reprezentările gradientului într-o singură imagine

- Detectarea și segmentarea plăcuței
  - Procesare inițială: aplicăm conversia la Grayscale, filtrăm imaginea folosind un filtru Gauss 3x3, egalizăm histograma imaginii cu cv::equalizeHist(), și aplicăm cv::Canny() pentru a obține contururile pe imagine
  - Se extrag contururile cu cv::findContours()
  - Se simplifică contururile folosind pi::simplifyContours()
  - Se elimină contururile ce nu întrunesc criteriul pi::isLikeARectangle()
  - Se afișează contururile rămase cu verde pe imaginea originală
  - Se segmentează plăcuța folosind bounding box-ul conturului
- Detecția și segmentarea caracterelor de pe plăcuță

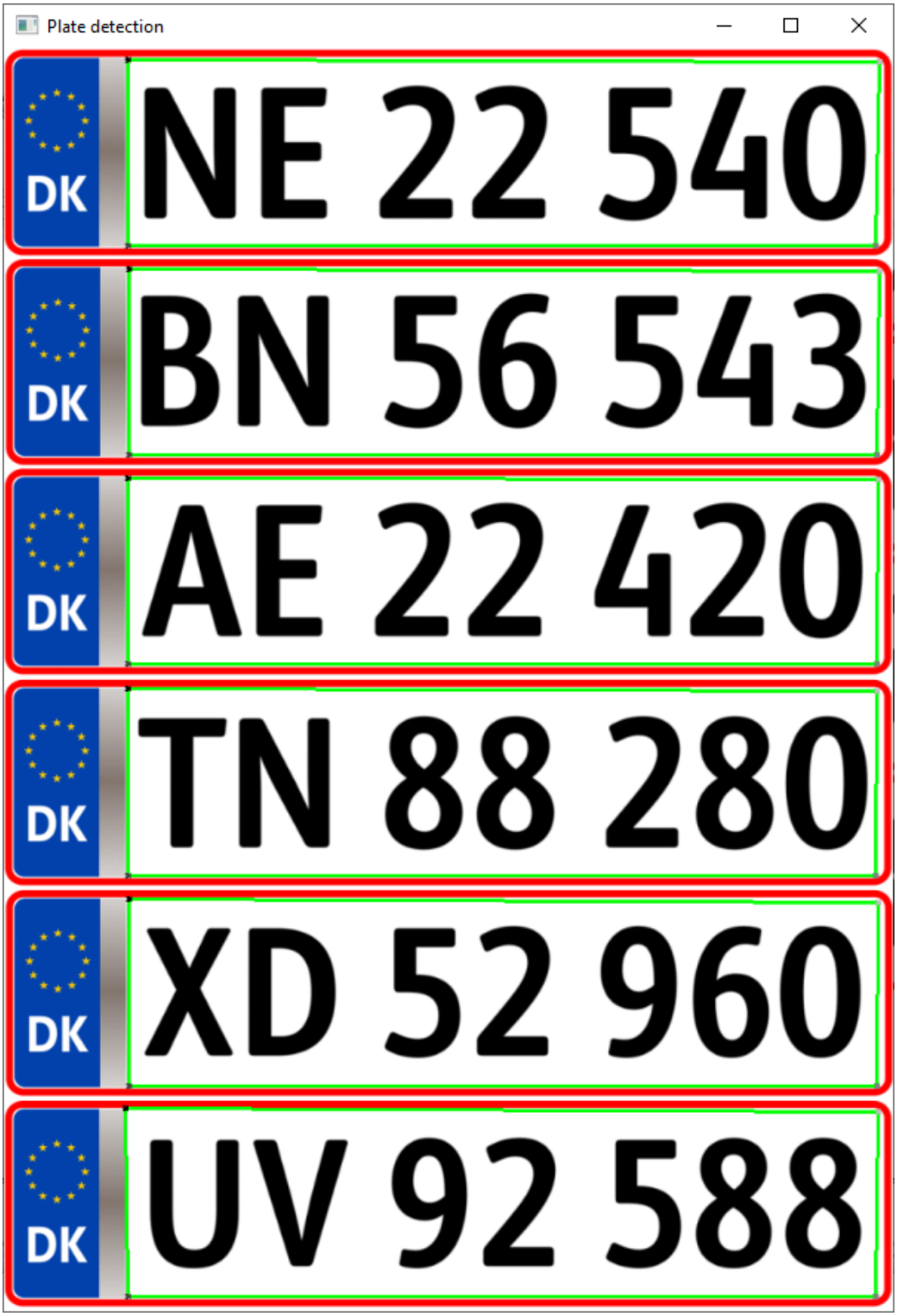
    - Procesare inițială: Grayscale, aplicăm cv::threshold() folosind algoritmul Otsu pentru segmentare, aplicăm cv::Canny() pentru a obține contururile din interiorul plăcuței de înmatriculare
    - Pentru fiecare contur, se determină bounding box-ul lui
    - Se calculează mediana lățimii și înălțimii bounding box-urilor
    - Considerând că literele au același font, putem afirma că variația înălțimii între litere este mică, iar variația lățimii este mai mare (caracterele I și 1 sunt cu mult mai subțiri decât W și M, dar toate au aceeași înălțime). De asemenea, literele vor fi contururile majoritare de pe plăcuță
    - Se elimină contururile ce deviază cu mai mult de 10% față de înălțimea mediană, sau cu mai mult de 75% din lățimea mediană - conform observației de mai sus, contururile non-caractere sunt eliminate
    - Restul contururilor sunt folosite pentru segmentarea literelor. Modalitatea e aceeași ca la plăcuță
    - La final, sortăm contururile după coordonata X și Y pentru a impune ordinea literelor de la stânga la dreapta
- Recunoașterea numărului de înmatriculare de pe plăcuță
    - Pentru plăcuțele UE, am determinat că fontul DIN 1451 Mittelschrift are o similaritate extrem de mare (sau chiar e cel utilizat)
    - Am pregătit o imagine și un fișier cu date, prin care extragem literele din fontul Mittelschrift. Dimensiunea literelor de referință este de 36, dar nu contează cât timp e suficient de mare
    - Pentru fiecare literă de pe plăcuță, aplicăm algoritmul pi::getLetterDistance() între ea și litera din fontul de referință. Litera de referință cu cea mai mică distanță față de cea segmentată este caracterul recunoscut de către program
    - Eventual vom calcula și orientarea conturului literelor pe regiuni ca un criteriu adițional, pentru a crește acuratețea programului

Tehnologii folosite

Aplicația va fi realizată în C++, folosind mediul de dezvoltare Visual Studio 2017 / 2019. Se va folosi OpenCV pentru prelucrarea imaginilor, dar și pentru interfața grafică (modulul High GUI).

Exemple din implementația finală:

Testăm eficacitatea recunoașterii caracterelor:



Plăci:

Placa 0: UV 92 588

Placa 1: XD 52 960

Placa 2: TN 88 290

Placa 3: AE 22 420

Placa 4: BN 56 543

Placa 5: NE 22 540

Text detectat prin ”distanța” imaginilor după pixeli:

Placa 0: UV 92 S88 (6 / 7)

Placa 1: XD S2 066 (4 / 7)

Placa 2: TN 88 280 (6 / 7)

Placa 3: AE 22 420 (7 / 7)

Placa 4: B (FM) S6 S43 (4 / 7, a eșuat la detecția lui N)

Placa 5: NE 22 S40 (6 / 7)

Referitor la metoda gradient (magnitudine, orientație), se obțin rezultate cu acuratețe mai mică. Ca exemplu, luăm placa 1:

Original: XD 52 960

Metoda cu pixeli: XD S2 066 (4 / 7)

Magnitudine: YL 11 J10 (1 / 7)

Orientație: YI II I60 (2 / 7)

Exemple din implementația intermediară

```
Median Width: 45 | Median Height: 58
Skipped a bounding box: 9, 4 at 321, 85
Skipped a bounding box: 12, 5 at 342, 82
Skipped a bounding box: 15, 6 at 358, 80
Skipped a bounding box: 367, 44 at 21, 47
Skipped a bounding box: 388, 90 at 0, 1
45 - 58
47 59
Letter 41 is H, distance: 0.08
43 60
Letter 467 is M, distance: 0.09
19 58
Letter 334 is 1, distance: 0.08
36 58
Letter 500 is 4, distance: 0.11
53 61
Letter 169 is W, distance: 0.10
45 58
Letter 724 is O, distance: 0.10
52 59
Letter 478 is W, distance: 0.17
```

Imaginea de referință pentru Mittelschrift

ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789

Exemple din implementația preliminară

”Plăcuțele” detectate sunt reprezentate printr-un dreptunghi verde!

Caz nefavorabil, unde pot fi eșuări:

Bibliografie

Wikipedia: OCR

OpenCV Documentation

Wikipedia: Canny Edge Detector

DIN 1451 Mittelschrift

GitHub Topic - License Plate Recognition

Wikipedia: Image Gradient

Image Gradients - Implementation