**Recunoasterea cartilor de joc pentru simularea unor jocuri de carti**

1. **Introducere**

Se doreste dezvoltarea unui algoritm de recunoastere a cartilor de joc dintr-un pachet

de 52 de carti ce poate fi folosit in unele jocuri de carti precum Poker si Blackjack. Pentru a face

posibil acest lucru, trebuie sa ne asiguram ca vom procesa imaginea suficient pentru a

imbunatatii acuratetea recunoasterii. Pentru asta va trebui sa binarizam imaginea si sa eliminam

zgomotul nedorit, sa aplicam algoritmi de detectare a muchiilor si a colturilor si sa aplicam

transformari de perspectiva.

1. **Fundamente Teoretice**

**Studiu Bibligrafic:**

In dezvoltarea acestui proiect, lucrarea: Playing Card Detection and Identification

oferita drept suport a jucat un rol important, indrumandu-ne prin fiecare pas ce necesita parcurs

pentru procesarea si obtinerea unor rezultate cat mai precise. Totodata, lucrarile de laborator sunt

cele care au pus bazele acestor cunostinte si le-au consolidat, facand posibila parcurgerea acestui

proiect. Pentru rezolvarea unor probleme si erori, ne-a stat la dispozitie Stack Overflow, fiind

indispensabil in unele situatii.

**Metode aplicate:**

Pentru realizarea acestui proiect au fost utilizate diverse metode si artificii precum

binarizarea imaginii, metode pentru detectia de contururi si muchii, algoritmi de

detectare a colturilor si metode de transformare a perspectivei si template matching.

Pentru binarizarea imaginii s-a utilizat functia de binarizare cu prag automat regasita si in

lucrarile de laborator. Pragul de binarizare este calculat in functie de imagine, acesta variand de

la o imagine la alta, eliminand nevoia de schimbarea manuala a pragului in cazul imaginilor cu

luminozitati diferite.

Pentru a facilita conturul s-a utilizat functia de filtrarer gaussiana cu un kernel de 5x5 ce elimina

zgomotul nedorit.

Detectia muchiilor si a conturului s-a obtinut cu ajutorul functiilor Canny si findContours puse la

dispozitie de biblioteca OpenCV.

Liniile drepte au fost detectate cu ajutorul transformatei Hough Line ce necesita ca preprocesare

aplicarea functiei de detectare a muchiilor. Acest algoritm ne ajuta la detecatarea colturilor, un

pas obligatoriu pentru a aplica transformari de prespectiva si, ulterior, potrivirea de sablon.

1. **Proiectare si Implementare.**

Solutia pe care noi am ales-o pentru a ajunge la rezultatele obtinute consta in

pregatirea imaginilor de template, adica fotografierea fiecari carti din pachetul de 52 de carti de

joc si decuparea acesteia pentru a fi usor de recunoscut si procesat, imagini ce vor fi ulterior

folosite pentru a realiza template matching. Pentru a usura identificarea cartilor, numele fiecarei

imagini este dat de numarul si simbolul cartii. Acesta este pipeline-ul (fluxul) prin care imaginea

trece pentru a realiza identificarea cartilor:

[Încărcare imagine] -> [Binarizare] -> [Filtrare Gaussiană] -> [Detectare muchii] ->

[Detectare contururi] -> [Detectare muchii si linii drepte] -> [Detectare colțuri] ->

[Corectare perspectivă] -> [O usuara corectie a imagini] -> [Recunoaștere carte] ->

[Afisarea fiecarei carti in mod individual alaturi de denumire]

Dupa pasul de detectie de contururi, fiecare pas care urmeaza este realizat pentru fiecare carte gasita din imagine.

**Descrierea funcționalităților modulelor implementate:**

* prepareInputPhoto(): o imagine la alegere va fi deschisa de catre utilizator in formatul

grayScale urmand sa fie binarizata si redimensionata.

* GaussianBlur(): functie din biblioteca OpenCV pentru a aplica un filtru de tip gaussian pe

imagine cu rol de a reduce zgomotul.

* Canny(): functie a bibliotecii OpenCV pentru detectia de muchii.
* computeContours(): aceasta functie se ocupa cu gasirea de contururi din imagine,

desenandu-le intr-o noua imagine. Doar conturul cartilor este desenat, nu si al simbolurilor

din imagini.

Urmatoarele functii se aplica pe fiecare contur gasit in imagine:

* drawContours(): deseneaza conturul unei singure carti pentru a putea fi usor de identificat

din imaginea initiala.

* HoughLinesP(): transformata probabilistica Hough Line ce detecteaza cele 4 linii ale

cartii, inainte de a o folosi, este nevoia sa apelam din nou Canny.

* computeDistances(): parcurge toate liniile gasite, le calculeaza distantele, le deseneaza

intr-o noua imagine si sorteza vectorul de distante descrescator.

* computeCorners(): iteram prin toate liniile, calculam intersectiile liniilor cu functia

lineIntersection, adaugam punctul gasit in lista de colturi si afisam intr-o imagine separata

colturile.

* computePersepective(): realizeaza transformarea de perspectiva utilizandu-se de functiile

din OpenCV getPerspectiveTransform si warpPerspective, pentru acest lucru este nevoie de

colturi.

* templateImageCorrection(): deoarece cartile nu au colturi ascutite, exista niste pixeli

negrii in fiecare colt pe care ii vom face albi

* getTemplateCards(): incarca imaginile de template in functie de nume si le

redimensioneza cu marimea cartii gasite din imagine pentru ca ambele sa aiba aceleasi

dimensiuni

-matchTemplateBest(): realizeaza potrivirea de sablon utilizand functia din OpenCV matchTemplate().

**Descrierea algoritmilor implementați:**

* binarizarea automata calculeaza un prag optim in functie de histograma imaginii:

se va selecta un prag = media intensitatii maxime si minime si se va segmenta imaginea in functie de acest prag, calculandu-se valorile medii de intensitate pentru pixelii mai mari, respectiv mai mici de cat pragul calculat. Noul prag actualizat va fi egal cu media celor 2 valorii medii. Se repeta acesti pasi pana cand diferenta dintre pragul curent si cel anterior este mai mica decat o eroare aleasa.

* Aplicarea filtrului gaussian: asupra imaginii se realizeaza operatia de convolutie cu un kernel de 5x5:

O imagine care conține Font, captură de ecran, linie, text

Descriere generată automat

Acest pas are rol de a crea un efect de blur pentru a elimina zgomotele.

* Canny(): aceasta functie utilizata pentru a detecta muchiile consta in urmatorii pasi:

1. Se calculeaza gradientul imaginii, gradientul intr-un punct este directia de variatie a

intensitatii imaginii.

1. Se filtreaza imaginea cu un filtru Gaussian(realizat anterior)
2. Se calculeaza modulul si directia gradientului
3. Se realizeaza suprimarea non-maximelor modulului gradientului
4. Se aplica o functie de binarizare adaptiva a punctelor de muchii pentru prelungirea prin

histereza.

* findContours(): aceasta functie scaneaza imaginea si cand gaseste un pixel negru traseaza

limita componentei conectate (conturul) urmarind pixelii care impart intre ei margini sau colturi. Aceasta functie returneaza toate contururile gasite si o ierarhie pentru a putea observa ce contur se afla in interiorul altui contur. Noi avem nevoie doar de primele contururi gasite deci vom lua contururile care in ierarhie au parintii = -1.

* HoughLinesP(): aceasta transformata utilizeaza equatia unei linii in coordonate polare:

x\*cosO + y\* sinO = r. <https://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm>

* computCorners(): pentru a extrage colturile ne-am folosit de formula de intersectie a 2 linii (ecuatia unei linii = ax + by + c = 0):

O imagine care conține Font, text, linie, alb

Descriere generată automat

Se vor calcula lungimile a 2 linii: verticale si orizontale. Normal cartea are liniile verticale mai lungi decat cele orizontale, iar in cazul in care linia verticala este mai mica decat cea orizontala acest lucru denota ca imaginea este intoarsa la 90 de grade, astfel interschimbam lungimile si 2 varfuri pentru a o aduce la normal.

Deoarece algoritmul Hough returneaza mai mult de 4 linii cat are cartea, am ales doar liniile cu o anumita lungime. Uneori acest lucru mai da gres si singura modalitate de rezolvare este de a modifica parametrii functiei HoughLines precum maxLineGap si minLinLength.

* computePerspective(): se calculeaza distantele celor 4 linii ale cartii cu ajutorul celor 4

varfuri gasite, si se aplica getPerspectiveTransform si warpPerspective care mapeaza

varfurile cartii pe o imaginea dreapta de aceeasi dimensiune cu cartea pentru a indrepta

imaginea

* matchTemplateBest(): functia matchTemplate() pur si simplu suprapune imaginea de

input cu cea de template de la stanga la dreapta de sus in jos. Pentru fiecare locatie se calculeaza o metrica ce reprezinta cat de bun sau rau e matching-ul realizat. Se utilizeaza functia minMaxLoc pentru a obtine valorile maxime si minime si pozitiile acestor, pe noi interesandu-ne doar cele maxime. Acest lucru se va realiza pentru fiecare carte de template din cele 52 si se va actualiza maxiumul daca un matching mai bun a fost gasit.

1. **Rezultate**

Pentru urmatorii parametrii HoughLinesP:

HoughLinesP(oneCardContourCanny, lines, 3, CV\_PI / 180, 20, 20, 10);

Input: Output:

**** **O imagine care conține captură de ecran

Descriere generată automat**

Input: Output:

 O imagine care conține text, captură de ecran, monocrom, model

Descriere generată automat

Pentru o carte, simbolul a fost identificat gresit din cauza calitatii imaginii.

Input: Output:

 O imagine care conține text, captură de ecran, model

Descriere generată automat

Input:



Output:

OpenCV Error: Assertion failed (src.checkVector(2, 5) == 4 && dst.checkVector(2, 5) == 4) in cv::getPerspectiveTransform, file C:\build\master\_winpack-build-win64-vc15\opencv\modules\imgproc\src\imgwarp.cpp, line 3232

Schimbam parametrii:

HoughLinesP(oneCardContourCanny, lines, 3, CV\_PI / 180, 40, 20, 10);

O imagine care conține schiță, captură de ecran, model, monocrom

Descriere generată automat

Vom vizualiza pipeline-ul afisand fiecare imagine a procesarii.

Binarizare, Gaussian filter, Canny:

O imagine care conține text, captură de ecran, alb și negru, proiectare

Descriere generată automat

Toate contururile gasite:

O imagine care conține text, Joc de cărți

Descriere generată automat

Conturul primei carti gasite:

O imagine care conține captură de ecran, alb și negru, negru, text

Descriere generată automat

Liniile gasite (se poate observa ca sunt mai multe decat 4):

O imagine care conține diagramă, captură de ecran, text, linie

Descriere generată automat

Colturile gasite:

O imagine care conține diagramă, text, linie, captură de ecran

Descriere generată automat

Cartea dupa transformarea perspectivei si inainte de template matching:

O imagine care conține captură de ecran, text

Descriere generată automat

Final:

O imagine care conține text, captură de ecran

Descriere generată automat

1. **Concluzii:**

Obiectivele propuse au fost fost indeplinite cu success. O inbunatatire ce poate fi adusa acestui program poate fi calcularea automata a parametrilor HoughLinesP pentru a nu mai necesita schimbare manuala in cazul erorilor cum a fost prezentat si anterior.

1. **Bibliografie**

<https://docs.opencv.org/4.x/>

<https://users.utcluj.ro/~rdanescu/teaching_pi.html>

<https://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm>

<https://theailearner.com/tag/cv2-getperspectivetransform/>

[web.stanford.edu/class/ee368/Project\_Winter\_1819/Reports/snyder.pdf](https://web.stanford.edu/class/ee368/Project_Winter_1819/Reports/snyder.pdf)