Militaru Mihai-Alexandru

Grupa 343

Concepte si Aplicatii in Vederea Artificiala - Documentatie Tema1

Calculator automat de scor pentru jocul Double Double Dominos

Obiectiv

Scopul acestei teme este implementarea unui sistem automat de calculare a scorului pentru o varianta a jocului de domino numita Double Double Dominoes.

Implementare

1. Taskul 1

Pentru primul task am citit fiecare imagine din setul de testare folosind functia imread() din libraria CV2. Imaginea a fost ulterior procesata folosind functia extrage_careu(image, margins) ce are ca parametrii imaginea parsata (image) si marginile ce se adauga la rezultatul final.

Functia extrage_careu() proceseaza imaginea in urmatoarele etape:

- Se convertește imaginea data ca parametru din spatiul de culoare BGR în spatiul de culoare HSV utilizand cv.cvtColor.
- Se definește un interval de culori albastre în formatul HSV. Dupa numeroase incercari am ajuns la concluzia ca vectorii lower_blue si upper_blue, respectiv [40, 120, 20] si [143, 255, 255] ofer cele mai satisfacatoare rezultate.
- Se creeaza o masca pentru a selecta doar acele regiuni din imagine care se potrivesc intervalului de culori albastre.
- Se aplica operatia cv.bitwise_and pentru a izola zona care corespunde culorilor albastre in imaginea HSV.
- Se converteste rezultatul la o imagine grayscale utilizand cv.cvtColor.
- Se aplica un filtru Gaussian si un filtru median pentru a îmbunatati imaginea si pentru a evidentia mai bine contururile. Pentru filtrul Gaussian am folosit un kernel de (5,5), iar pentru filtrul median am ales marimea kernel-ului de 5. Pentru a combina cele doua rezultate in urma aplicarii celor doua filtre am folosit functia cv.addWeighted. Parametrul alpha = 1.2 ii corespunde filtrului Gaussian, iar parametrul beta = -0.8 ii corespunde celui median. Rezultatul in urma aplicarii celor doua ponderi se calculeaza dupa formula : $g(x)= \alpha f0(x) + bf1(x)$.

- Se aplica operatia de cv.threshold unde fiecare pixel este adus la valoarea 0 daca este mai mic decat 30, iar in caz contrar valoarea sa este setata la 255. Aceasta operatie are ca scop accentuarea contururilor imaginii date ca parametru.
- Se efectueaza operatii de erodare și dilatare pentru a îmbunătăti contururile și pentru a elimina zgomotul. Pentru aceste doua operatii am folosit un kernel de (3,3). Pentru a ajunge la marginea tablei de joc am aplicat un numar de 8 iteratii pentru functia cv.erode().
- Se utilizează operatia Canny pentru detectarea muchiilor.
- Se foloseste functia cv.findContours pentru a identifica contururile din imaginea binara edges. Functia cv.RETR_EXTERNAL specifica ca se doresc doar contururile exterioare, adica cele care nu sunt continute in altele, iar functia cv.CHAIN_APPROX_SIMPLE stocheaza doar punctele necesare pentru a reprezenta contururile.Variabila max_area va stoca cea mai mare arie data de contururi. Pentru fiecare contur, se itereaza prin punctele componente ale conturului si se identifica colturile posibile. Se calculeaza diferenta intre coordonatele x ale colturilor pentru a identifica colturile din dreapta sus si stanga jos ale patratului. Daca aria poligonului construit pe baza coordonatelor colturilor identificate este maxima se actualizeaza variabila max_area si se salveaza coordonatele colturilor patratului. Se ajusteaza coordonatele colturilor patratului adaugand sau scazand o margine data ca parametru. In final, variabilele top_left, bottom_right, top_right, si bottom_left vor conține coordonatele colturilor patratului identificat în imaginea data ca parametru.
- Se definesc dimensiunile dorite pentru imaginea finala a careului, respectiv width-ul si height-ul, ambele avand valoarea 1350. Se calculeaza matricea de transformare de perspectiva folosind colturile patratului și destinatia dorita. Se aplica transformarea de perspectiva asupra imaginii originale pentru a extrage careul si a-l aduce în forma dorita. Imaginea este ulterior convertita la grayscale.

Dupa citirea imaginii din setul de testare si procesarea ei am citit imaginea cu tabla goala din folderul imagini_auxiliare si am aplicat functia *extrage_careu()* si pe aceasta pentru a avea in vedere doar careul cu tabla de joc. Atat pe poza din setul de testare (*careu*), cat si pe imaginea cu tabla goala de joc (*img_tabla_goala*) am aplicat operatia de *cv.threshold()*. Astfel, variabila *result* va contine rezultatul in urma aplicarii operatiei de *threshold* pe poza din setul de testare, iar *result_tabla_goala* va continue rezultatul in urma aplicarii operatiei de *threshold* pe imaginea cu table goala.

In Figura 1 se poate observa cum arata *result,* respectiv imaginea din setul de testare unde am aplicat doar operatia de threshold.

Figura 1. *Imaginea cu tabla acoperita de piese de domino dupa operatia de threshold.*

Deoarece imaginea unde sunt amplasate domino-urile contine mult zgomot, printre care romburile desenate pe tabla si scrisul din mijloc am facut diferenta intre imaginea cu piesele de domino (cea din Figura 1) si imaginea cu tabla goala in urma operatiei de threshold, respectiv diferenta dintre *result* si *result_tabla_goala.* Rezultatul in urma acestei diferente poate fi vazut in Figura 2.

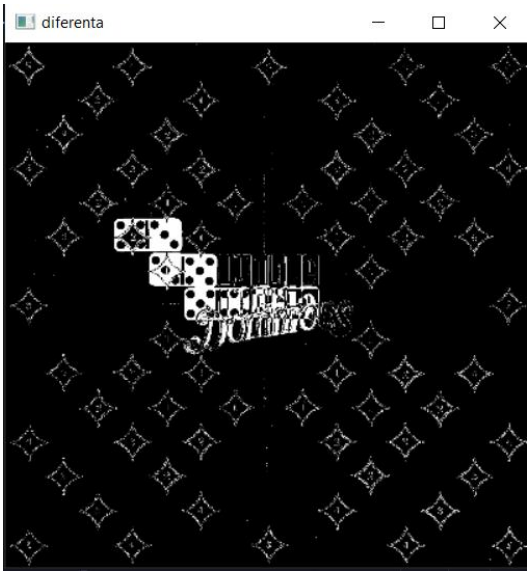

Figura 2. *Diferenta dintre imaginea din setul de testare si imaginea ce contine tabla goala*

Astfel, am reusit sa elimin zgomotul din imagine cu ajutorul functiei *cv.absdiff()* intre *result* si *result_tabla_goala.*

Noile imagini rezultate, respectiv *result, result_tabla_goala* si *diferenta* au dimensiunea de 1350 × 1350. Cunoastem faptul ca o tabla de joc are dimensiunea de 15 × 15 pentru asezarea pieselor. Drept urmare, fiecare casuta din tabla de joc va avea dimensiunea de 90 × 90 pixeli. Va trebui sa impartim imaginea folosind linii orizontale si verticale pentru a delimita fiecare casuta unde va fi amplasata o parte dintr-un domino.

Se creeaza o lista goala *lines_horizontal* pentru a stoca coordonatele liniilor orizontale ale grilei. Se utilizeaza un pas de 90 de pixeli pentru a determina pozitiile orizontale ale liniilor. Pentru fiecare pozitie, se adauga un segment de linie de la (0, i) la (1349, i) în lista *lines_horizontal*.

Analog am procedat si pentru liniile verticale. Se creeaza o lista goala *lines_vertical* pentru a stoca coordonatele liniilor verticale ale grilei. Se utilizeaza acelasi pas de 90 de pixeli pentru a determina pozitiile verticale ale liniilor. Pentru fiecare pozitie, se adauga un segment de linie de la (i, 0) la (i, 1349) în *lista lines_vertical.* Cunoscand coordonatele liniilor verticale si orizontale am trasat linii de grosime egala cu 2 pixeli pe imaginea *result.*

Ulterior, am apelat functia *determina_configuratie_careu(thresh, lines_horizontal, lines_vertical).* Variabilele *, lines_horizontal* si *lines_vertical* contin coordonatele pentru liniile si coloanele tablei de joc, iar pentru parametrul *thresh* am folosit imaginea in urma aplicarii operatiei de *threshold* pe diferenta dintre imaginea cu piese domino si table goala, respectiv *result_tabla_goala.* Parcurgand ambii vectori, atat *, lines_horizontal* cat si *lines_vertical* am extras fiecare patch, mai exact fiecare casuta a tablei de joc. Un exemplu de patch se poate observa in Figura 3.

Figura 3. *Un exemplu de patch extras dintr-o imagine de antrenare in care a fost plasat un domino ce contine o fata cu numarul 6.*

Pentru fiecare astfel de patch am calculat media culorii folosind functia *np.mean(patch).* In Figura 4 se poate observa diferenta dintre un patch cu o piesa de domino si un patch fara piesa de domino. Media culorii este mai mare pentru un patch ce contine o piesa de domino decat un patch gol. Din acest motiv extragerea de patch-uri si calculul mediei culorii s-a realizat pe diferenta dintre poza cu piese si table goala, pentru a ramane evidentiate doar piesele de domino.

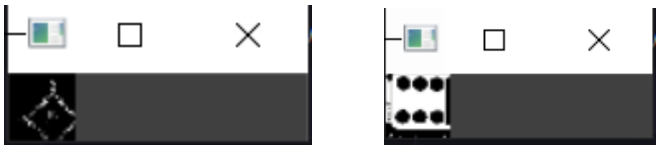

Figura 4. *Un patch ce contine doar o parte din tabla goala si alt patch ce contine o bucata dintr-o piesa de domino.*

Dupa ce am parcurs fiecare patch din imagine am stocat intr-un vector numit *patches* coordonatele liniei, respectiv coloanei din *, lines_horizontal* si *lines_vertical,* pentru a afla pozitia exacta a fiecarui patch pe imagine si media culorii acelui patch. Vectorul *patches* contine elemente sub forma *[medie_patch, i, j]* unde i si j sunt linia si coloana a caror intersectie imi determina patch-ul, iar *medie_patch* este media culorii patch-ului respectiv.

Deoarece media culorii este mai mare in cazul patch-urilor ce contin piese de domino, am sortat descrescator vectorul *patches* dupa parametrul *medie_patch.* Aplicand acest algoritm pentru fiecare poza din setul de testare, in urma sortarii stiu ca primele 2 * *numar_poza* elemente din vectorul *patches* detin coordonatele i si j unde a fost pozitionata piesa, restul elementelor fiind portiuni din tabla neacoperite de vreo piesa.

Spre exemplu, daca citesc prima poza din primul set de antrenare, 1_01.jpg stiu ca aplicand algoritmul descris mai sus, primele 2 patch-uri ($1 \times 2 = 2$) contin un domino. Citind urmatoarea poza din setul de antrenare, 1_02.jpg, aplicand acelasi algoritm stim ca primele 4 patch-uri ($2 \times 2 = 4$) reprezinta primele 2 piese de domino puse pe tabla.

Drept urmare, la fiecare poza nou citita stiu ca mai am inca 2 patch-uri, iar acelea reprezinta piesa de domino pusa la runda curenta. Insa, pentru a rezolva task-ul 1 trebuie afisat intr-un fisier nou piesa care este amplasata in runda respectiva.

Spre exemplu, daca ma aflu la runda 3 si citesc imaginea 1_03.jpg algoritmul va gasi 3 piese de domino pe tabla, deci primele 6 patch-uri din vectorul *patches* imi ofera informatii despre piese. Insa problema care se pune este care din cele 3 piese stiu ca este ultima pusa pe tabla, cu alte cuvinte trebuie sa stiu ce domino a fost pus la runda curenta, cunsocand faptul ca 2 piese au fost deja puse la rundele precedente.

Solutia pentru a distinge ce domino nou a fost adaugat este o matrice de frecventa, unde la fiecare runda marchez cu 1 in *matrix_of_pieces[i][j],* patch-ul gasit de algoritm cu coordonatele i si j. Spre exemplu, daca in imaginea 1_01.jpg am gasit un domino pe pozitiile (0,1) si (0,2) atunci *matrix_of_pieces[0][1] = 1* si *matrix_of_pieces[0][2] = 1.* Continuam citirea imginilor, iar la imaginea 1_02.jpg gasim un nou domino pe pozitiile (1,1) si (1,2). Algoritmul doar stie sa sorteze patch-urile dupa media culorii, deci singura informatie utila era faptul ca s-au gasit 2 piese de domino in cele 4 patch-uri: (0,1), (0,2), (1,1) si (1,2). Insa, eu am marcat mai devreme cu 1 in matricea de frecventa coordonatele (0,1) si (0,2) si pot trage concluzia ca pe acele pozitii din tabla de joc a fost pusa la runda precedenta o piesa. Nu-mi ramane decat sa marchez acum in matricea de frecventa coordonatele (1,1) si (1,2) cu 1 iar acum stiu faptul ca noua piesa este pusa pe tabla de joc la coordonatele (1,1) si (1,2).

La afisarea in fisiere text am tinut cont de faptul ca o piesa este formata din 2 patch-uri, (x,y) si (z,w). Coordonatele x si z reprezinta liniile iar regulile jocului spun ca numerotarea incepe de la 1 pe tabla, deci am incrementat cu 1 aceste coordonate pentru a le trece corect in fisier. Tot regulile jocului spun ca fiecare coloana este indexata folosind litere din alfabet de la A la O (15 in total). Am construit un dictionar pe principiul (key, value) unde key reprezinta indexul coloanei, iar value reprezinta litera corespunzatoare ei. Spre exemplu dictionary[0] = 'A', dictionary[1] = 'B' si asa mai departe pana la dictionary[14] = 'O'.

2. Taskul 2

Cunoscand acum coordonatele pentru fiecare patch in parte, putem sa extragem bucata de piesa din imaginea respectiva si sa numaram cate cercuri se afla in imagine. Pentru acest task am definit functia *detect_circles(image)* unde *image* este un patch. Am aplicat transformata Hough detecta cercuri, *circles = cv.HoughCircles(image, cv.HOUGH_GRADIENT, 1, 10, param1=50, param2=11, minRadius=8, maxRadius=20)*, iar parametrii folositi sunt urmatorii:

- image: Patch-ul ce contine una din fetele domino-ului.
- cv.HOUGH_GRADIENT: Metoda folosita pentru detectarea cercurilor (o metoda bazata pe gradient).
- 1: Raportul invers al rezolutiei acumulatorului fata de rezolutia imaginii.
- 10: Distanta minima dintre centrele cercurilor detectate.
- param1=50: Valoarea pragului mai mare pentru detectorul de margini (Canny). Este utilizat in metoda gradientului.
- param2=11: Pragul acumulatorului pentru centrele cercurilor în stadiul de detectare.
- minRadius=8: Raza minima a cercurilor ce urmeaza a fi detectate.
- maxRadius=20: Raza maxima a cercurilor ce urmeaza a fi detectate.

Daca *circles* are valoarea *None* atunci fata domino-ului din patch-ul dat ca parametru nu contine niciun cerc, deci este 0. In caz contrar, *circles* returneaza un array cu coordonatele cercurilor gasite, iar aplicand functia *len(circles)* returneaza numarul cercurilor gasite.

Modificand codul de la primul task, unde iteram prin vectorul *patches* si luam coordonatele i si j pentru patch-ul respectiv am aplicat functia *detect_circles(patch)* si am gasit numarul de puncte pentru acel patch, deci am aflat fata domino-ului la coordonatele (i,j) de pe tabla de joc.

O problema intampinata la taskul 2 a fost faptul ca unele piese de domino erau pozitionate putin in afara casutei de pe tabla de joc iar in patch-ul obtinut nu erau vizibile toate punctele, sau era vizizbila o piesa vecina. Pentru a rezolva aceasta problema am pus un offset = 4 pentru a mari dimensiunea patch-ului si a ma asigura ca sunt vizibile toate punctele, cu riscul de a extrage informatie si din piesele de domino vecine. Insa, pentru a vedea exact numarul de cercuri pentru un patch am ales un array de margini si am folosit functia *extrage_careu()* unde parametrul *margin* ia valoarea fiecarui element din array. Aplicand diverse margini pe poza initiala, impartirea pe linii si coloane a dus la patch-uri putin diferite. In acest fel am obtinut patch-uri care captureaza aceeasi piesa, dar unele margini nu permit vizualizarea intregii piese. Testand pe

numeroase exemple am ajuns la concluzia ca daca folosesc mai multe margini rezultatul majoritar este cel corect.

Dupa ce am obtinut numarul corect de cercuri pentru fiecare patch am afisat rezultatul in fisierele txt pentru fiecare piesa. Astfel, fiecare linie din fisierul unei runde contine coordonatele fetei din domino-ul respectiv si numarul de cercuri pentru acel patch.

3. Taskul 3

Acest task consta in calcularea scorului pentru fiecare runda. Fisierul *mutari.txt* contine ordinea jucatorilor.

Am ales sa hardcodez table jocului punand la coordonatele specifice numerele trecute pe romburi. De asemenea, am hardcodat traseul pionilor, iar prima pozitie, cea de start am marcat-o cu -1, mai exact *traseu[0]* = -1.

Pentru acest task am citit fisierele unde am scris la taskurile precedente piesa de domino pusa la fiecare runda. Pe masura ce citesc fisierul rundei curente citesc si randul jucatorului curent. Pentru fiecare jucator cand ii soseste randul citesc domino-ul pus de acesta din fisierul text si initializez o variabila *player_index_score* ce reprezinta scorul jucatorului cu numarul *index,* ce are valoarea 0.

O alta variabila numita *player_index_position* arata pozitia juactorului cu numarul *index* pe tabla. La inceputul jocului *player1_position* si *player2_position* sunt 0, deci *traseu*[*player1_position] = -1* si *traseu*[*player1_position] = -1.* Ambii jucatori sunt pe table de joc pe casuta de start.

Daca unul din jucatori are pionul pe o pozitie notata cu numarul x, iar domino-ul curent are una din fete egale cu x, atunci jucatorii ce indeplinesc aceasta conditie primesc 3 puncte si muta pionul 3 casute. Conditia se poate traduce astfel: daca *traseu[player_curent_position] == face_1_domino* sau *traseu[player_curent_position] == face_1_domino,* unde *face_1_domino* si *face_2_domino* sunt cele doua fete ale piesei de domino puse, atunci incrementam cu 3 scorul si pozitia lui *player_curent.*

Daca domino-ul pus de jucatorul curent este pus peste un romb cu o valoare x atunci, doar jucatorul curent primeste punctajul x si muta pionul x pozitii, deci conditia este urmatoarea: *empty_board[x_piece1][y_piece1] != 0* sau *empty_board[x_piece1][y_piece1] != 0*, unde x_piece1 si y_piece1 sunt coordonatele primei fete din domino, iar x_piece2 si y_piece2 sunt sunt coordonatele celei de a doua fete din domino. Daca domino-ul este unul dublu atunci punctajul si pozitia jucatorului sunt incrementate cu inca x puncta. In fisierul text al rundei respective este scris la final, pe o linie noua scorul jucatorului curent din runda curenta.