Documentație Pentru Tema 1 Computer Vision - Jocul Qwirkle

Rizea Mihai-Marius

2 decembrie 2025

Cuprins

1 Introducere

Acest document detaliază implementarea unui sistem de viziune Computer Vision destinat analizei automate a unui joc de masă de tip *Qwirkle*. Scriptul, realizat în Python utilizând biblioteca OpenCV, procesează o serie de imagini secvențiale pentru a detecta tabla de joc, configurația inițială, mutările efectuate de jucători, precum și culoarea și forma pieselor plasate, calculând automat scorul.

2 Secțiunea 1: Extragerea Cadranului (Pre-procesare Tablă)

Prima etapă a scriptului este izolarea tablei de joc din imaginea inițială și corectarea perspectivei pentru a obține o imagine perfect aliniată, pe care pot fi aplicate ulterior toate etapele de procesare.

2.1 Normalizarea Iluminării

Funcția `normalizeaza_iluminarea` are rolul de a corecta variațiile de luminozitate care pot afecta detectarea ulterioară a culorilor.

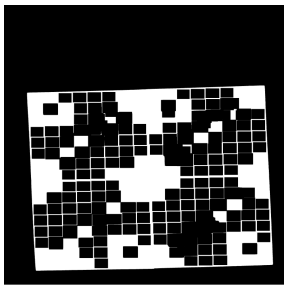
- Imaginea este convertită din spațiul de culoare BGR în **Lab**.
- Se extrage canalul **L** (Luminozitatea) și se aplică algoritmul **CLAHE** (Contrast Limited Adaptive Histogram Equalization) cu un *clipLimit* de 2.0.
- Canalele sunt reîmbinate și imaginea este convertită înapoi în BGR. Această metodă asigură un contrast uniform pe toată suprafața tablei.

2.2 Găsirea Conturului Tablei

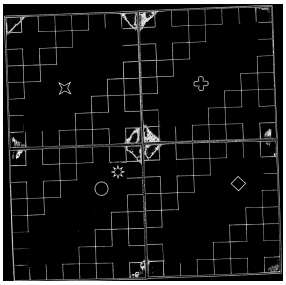
Funcția `gaseste_contur_tabla` izolează tabla folosind segmentarea cromatică:

1. Conversie în spațiul **HSV** și aplicarea unei măști pentru nuanțele de verde (specific tablei de joc). Intervalul utilizat este: $H \in [30, 90]$, $S \in [25, 255]$, $V \in [25, 255]$.
2. Aplicarea operațiilor morfologice (`cv2.MORPHCLOSE`) cu un kernel de 35×35, pentru a elimina zgomotul rămas în mască și pentru a unifica zona tablei într-o singură regiune.
3. Identificarea celui mai mare contur extern, care este considerat conturul tablei.

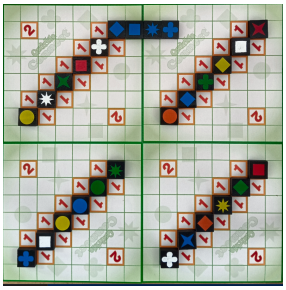
2.3 Extragerea Colțurilor și Transformarea de Perspectivă



Mască după aplicarea intervalelor HSV și operații morfologice



Imaginea binară cu liniile tablei, înainte de detectarea colțurilor.



Perspectivă corectată

Funcția `transforma_perspectiva_tabla` preia aceste 4 puncte (sortate topologic) și aplică o transformare de perspectivă (*warp perspective*), rezultând o imagine finală de **800x800 pixeli**, unde fiecare celulă a grilei are dimensiunea de 50×50 pixeli.

3 Secțiunea 2: Procesarea Imaginii de Start (Configurația Inițială)

La începutul jocului, este necesară determinarea configurației inițiale a diagonalelor preexistente. Funcția `determina_configuratia_initiala` analizează cele 4 cadrane ale tablei (16×16):

- Pentru fiecare cadran, se verifică două ipoteze: prezența pieselor pe diagonala principală vs. diagonala secundară.
- Se extrag patch-uri de 50×50 pixeli pentru celulele de pe aceste diagonale.
- Deoarece piesele au o luminozitate medie diferită față de fundalul tablei, se compară media canalului V (din HSV) al patch-urilor.
- Diagonala cu media valorilor mai mică (indicând patchuri mai întunecate/contrastante față de fundal) este considerată diagonala activă și marcată în matricea de stare cu valoarea 1.

4 Secțiunea 3: Găsirea Pozițiilor Pieselor (Detecție Mișcare)

Detectarea pieselor nou plasate se realizează prin compararea imaginii curente (*warped*) cu imaginea stării anterioare.

4.1 Algoritmul de Diferențiere

Funcția `detecteaza_schimbari_intre_imagini` implementează următoarea logică:

1. Se calculează diferența absolută (`cv2.absdiff`) între cadrul curent și cel anterior.

2. Se aplică un **Gaussian Blur** puternic (kernel 23×23) pentru a elimina zgomotul fin și micile discrepanțe de aliniere.

3. Imaginea diferență este analizată la nivel de patch (50×50).

4.2 Praguri de Decizie (Thresholding)

Pentru a reduce cazurile de *False Positive*, se utilizează un sistem cu două praguri:

- **High Threshold ($>$ 70):** Diferența este semnificativă, patch-ul este clasificat automat ca piesă nouă.
- **Low Threshold (60 - 70):** Zona este incertă (posibilă umbră sau mișcare a camerei). Se verifică diferența mediilor de intensitate între patch-ul vechi și cel nou. Dacă $|Medie_{noua} - Medie_{veche}| > 30$, se confirmă prezența piesei; altfel, este ignorată.

Funcția `identifica_mutari_noi` returnează coordonatele exacte ale pieselor adăugate în tura curentă.

Piesele din tabla de diferență între două imagini consecutive

5 Secțiunea 4: Detecție Culoare

Clasificarea cromatică se realizează în spațiul HSV, prin funcția `determina_culoarea_piesei`.

5.1 Strategia de rezolvare a pieselor poziționate greșit

Pentru a combate plasarea imperfectă a pieselor (care nu sunt perfect centrate în celula de 50×50), algoritmul nu analizează doar centrul patch-ului. Se verifică 5 puncte cheie:

- Centrul geometric al patch-ului.
- Cele 4 colțuri interioare (cu offset de 10px față de margini).

5.2 Intervale HSV

Pentru fiecare punct, se calculează media vecinătății și se verifică apartenența la intervale prestabilite (intervalele au fost extrase folosind ultimele imagini din fiecare dataset și verificarea individuală a fiecărei piese):

- **Alb:** Saturație mică ($S < 55$), Luminozitate mare ($V > 110$).
- **Galben:** $H \in [21, 35]$, $S \in [60, 255]$, $V \in [100, 255]$.
- **Portocaliu:** $H \in [3, 20]$, $S \in [65, 255]$, $V \in [100, 255]$.
- **Albastru:** $H \in [80, 135]$, $S \in [100, 255]$, $V \in [80, 255]$.
- **Verde:** $H \in [28, 79]$, $S \in [30, 255]$, $V \in [40, 220]$.
- **Roșu:** $H \in [0, 10] \cup [160, 180]$, $S \in [80, 255]$, $V \in [80, 255]$.

6 Secțiunea 5: Detecție Formă

Pentru recunoașterea formelor (Cerc, Pătrat, Romb, Trifoi, Stea-4, Stea-8) s-a utilizat metoda **Template Matching**.

6.1 Pre-procesarea pentru Formă

Funcția `preproceseaza_pentru_forma` pregătește patch-ul pentru a maximiza trăsăturile geometrice:

1. **Upscaling:** Imaginea patch-ului este mărită de 4 ori (interpolare cubică) pentru a păstra detaliile muchiilor la operațiile ulterioare.
2. **Binarizare:** Se extrage canalul de intensitate și se aplică *Otsu's Thresholding* după o blurare ușoară, rezultând o imagine binară (alb-negru).
3. **Eliminare Zgomot Margini:** Se aplică un crop de $P = 10$ pixeli pentru a elimina artefactele de la grila tablei.
4. **Centrare:** Se calculează momentele imaginii (M) pentru a găsi centrul de greutate al formei (C_x, C_y). Imaginea este translatată astfel încât centrul de greutate al formei să coincidă cu centrul geometric al imaginii.
5. **Resize Final:** Imaginea este redimensionată la o dimensiune standard de 64×64 pixeli.

Inainte de procesare

După procesare

6.2 Template Matching cu Invarianță la Rotație

Funcția `identifica_forma_piesei` compară patch-ul procesat cu o bază de date de șabloane (încărcate prin `incarca_sabloane_forme`).

- Se utilizează metoda `cv2.TM_SQDIFF_NORMED`, unde un scor mai mic indică o potrivire mai bună (0 = identic).
- **Rotire:** Pentru a compensa rotația pieselor pe tablă, patch-ul de input este rotit succesiv cu un unghi $\theta \in \{-3°, -2°, \ldots, +3°\}$.
- Se reține forma care generează scorul minim global. Dacă scorul este sub un prag foarte strict (0.02), potrivirea este considerată perfectă și căutarea se oprește prematur pentru eficiență.

7 Secțiunea 6: Logica de Joc (Bonusuri și Scor)

Ultima etapă constă în cuantificarea mutărilor conform regulilor jocului.

7.1 Generarea Matricei de Bonusuri

Funcția `genereaza_matrice_bonus` construiește o hartă a punctelor speciale bazată pe configurația inițială detectată în Secțiunea 2.

- **Bonus +2:** Se acordă celulelor aflate în colțurile diagonal opuse direcției principale a pieselor inițiale.
- **Bonus +1:** Se acordă celulelor adiacente diagonalelor.

7.2 Calculul Scorului pe Tură

Funcția `calculeaza_scor_tura` evaluează punctajul:

1. **Bonusuri de poziție:** Dacă o piesă este plasată pe o poziție specială, valoarea bonusului este adăugată la scor și poziția este marcată ca "consumată" (-1).
2. **Linii formate:** Se verifică lungimea liniilor continue formate pe orizontală și verticală care includ noile piese.
3. **Bonus Qwirkle:** Dacă o linie atinge lungimea de 6 piese, se acordă un bonus suplimentar de 6 puncte.

Scorul și configurația pieselor (poziție, formă, culoare) sunt exportate într-un fișier text pentru fiecare imagine procesată.

8 Bonus

Pentru a urmări evoluția tablei, s-a implementat o metodă asemănătoare cu cea folosită in jocul Qwirkle Connect, dar m-am bazat ca fiecare piesa are in medie o lungime si latime de 165 pixeli.

1. Detectarea poziției inițiale a pieselor

Prima imagine din secvență este folosită pentru a determina originea tablei $(0, 0)$ și pozițiile tuturor pieselor inițiale. Algoritmul funcționează astfel:

1. Imaginea este transformată în grayscale și ușor blurată pentru reducerea zgomotului.
2. Se aplică un **threshold** invers, astfel încât piesele să devină albe, iar fundalul negru.
3. Contururile pieselor sunt completate (`cv2.drawContours` cu `thickness=cv2.FILLED`) pentru a obține blocuri solide.
4. Se filtrează contururile prea mici (zgomot sub 500 pixeli) și se sortează după coordonatele X și Y pentru a identifica piesa din stanga ca fiind originea (0, 0).
5. Se calculează centrul fiecărui contur valid și poziția relativă a sub-pieselor din cadrul acestuia în raport cu originea $(0, 0)$.
6. Rezultatul este o listă de coordonate grid pentru toate piesele inițiale și pixelii centrului folosit ca referință pentru detectările ulterioare.

2. Detectarea pieselor noi

Pentru imaginile ulterioare, piesele noi sunt detectate prin comparația imaginii curente cu cea anterioară:

1. Se calculează diferența absolută între imaginea curentă și cea anterioară, apoi se transformă în grayscale.
2. Se aplică un **threshold** binar: pixelii sub 60 devin negri, restul albi.
3. Contururile rezultate sunt completate și filtrate pentru a elimina zgomotul (zone sub 500 pixeli).
4. Fiecare contur valid este împărțit în patrate corespunzătoare dimensiunii piesei (`tile_size`) și se verifică proporția pixelilor albi (minimum 50% pentru a fi considerată piesă).
5. Pentru piesele validate se calculează centrul și poziția relativă față de originea tablei.
6. Se adaugă piesele noi în lista celor ocupate, evitând duplicatele.

Această abordare combinată permite detectarea poziției inițiale a pieselor și urmărirea exactă a noilor plasări pe tabla de joc, oferind o soluție completă pentru arbitrajul automat. Pentru tipul de piesa si culoare, am folosit aceleasi functii, prezentate deja mai sus.

9 Concluzii

9.1 Rularea Scriptului

Scriptul poate fi rulat din terminal folosind comanda:

```
python script.py -i "folder-jocuri" -o output -t template
python bonus.py -i "jocul-bonus" -o output -t template
```

Unde:

- `-i` - folderul de input ce conține imaginile jocului;
- `-o` - folderul unde vor fi salvate rezultatele;
- `-t` - folderul care conține cele 6 template-uri pentru recunoașterea formelor pieselor.

Biblioteci Utilizate

Pentru implementarea și rularea scriptului au fost utilizate mai multe biblioteci Python. Majoritatea fac parte din biblioteca standard Python și nu necesită instalare separată:

- `argparse` – pentru parsarea argumentelor din linia de comandă;
- `sys` – pentru interacțiunea cu interpretorul Python;
- `os` – pentru manipularea căilor și directoarelor;
- `glob` – pentru obținerea listelor de fișiere.

În plus, scriptul utilizează două biblioteci externe. Versiunile folosite în proiectul de față sunt:

- **OpenCV (cv2):** 4.11.0
- **NumPy:** 2.2.3

Scriptul a fost testat și rulat folosind **Python 3.10.9**.