UNIVERSITATEA DIN BUCUREŞTI

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

**LUCRARE DE LICENTA**

**Analiza jocului de dame**

**Coordonator ştiinţific Absolvent**

**Conf. Dr. Alexe Bogdan Nicolae Cristian-Cătălin**

BUCUREŞTI, iunie 2017

**Abstract**

Unul din subdomeniile principale al vederii artificiale este analiza unei scene într-un mod similar celui uman, dar automatizând acest proces. Scenele cele mai adecvate pentru analiză sunt cele ce prezintă reguli structurale bine definite și asupra cărora se pot face cât mai multe deducții cu ajutorul detectorilor actuali. Jocul de dame reprezintă un astfel de sistem bine definit deoarece tabla de șah și piesele de dame sunt obiecte cu o structură geometrică foarte clară, iar regulile jocului nu impun probleme.

Lucrarea de față este o lucrare de sinteză ce încearcă analiza unei partide de dame filmate cu o cameră de deasupra. Fiecare cadru este analizat prin următorii pași: teste inițiale de validitate a cadrului, identificarea colțurilor din caroiajul tablei de joc prin detectorul Harris, distingerea punctelor relevante față de cele irelevante (printr-un algoritm de tip RANSAC) pentru identificarea tablei în cadrul curent, completarea prin regresie liniară a punctelor lipsă de la pasul anterior, separarea tablei în câmpuri individuale și analiza acestora pentru a obține poziția și culoarea fiecărei piese, compararea stării obținute față de starea precedentă pentru a valida sau nu rezultatul și apoi desenarea stării curente în 2D.

Combinația dintre detectorul Harris pentru colțuri, algoritmul de tip RANSAC pentru selecția punctelor relevante ipotezei noastre și regresia liniară pentru completarea selecției cu puncte omise de pașii anteriori oferă o acuratețe foarte mare în determinarea celor 81 de colțuri din geometria tablei de joc. Recunoașterea existenței piesei pe un anumit câmp cât și recunoașterea culorii acesteia reprezintă o problemă dependentă de condițiile scenei, dar se pot găsi valori de prag ce dau rezultate bune.

**Abstract**

One computer vision's main subdomains is that of analyzing a scene in manner similar to what a human would do, but automizing that process. Scenes that are most adequate for this kind of analysis are scenes that have well-defined structural rules and scenes with features that can be deducted with modern detectors. The game of draughts is a well-defined system because the chessboard and draughts pieces are objects with highly-structured geometry and the game itself does not present rules that may cause problems.

This paper is a work of synthesis that tries to analyze a game of draughts filmed from above with a camera. Every frame is analyzed following these steps: identifying the corners in the scene with the Harris detector, making a distinction between the corners that serve toward identifying the board and irrelevant corners (with a RANSAC style algorithm), using linear regression to find the missing points, dividing the board into individual fields and trying find the position and color for each piece, comparing the current state of the board with the previous known state of the board so we can validate our result and show it in a 2D representation.

The combination of the Harris corner detector, the RANSAC-style algorithm for selecting only the points relevant to our hypothesis and linear regression for completing our selection with points mission from earlier steps offers a high level of accuracy in determining the 81 points featured in the geometry of the board. Observing the existence of a piece on certain field and identifying its color is a problem that depends on scene conditions, but good results can be achieved by finding certain thresholds.

# 

# Cuprins

[Cuprins 3](#_Toc482189768)

[Introducere 4](#_Toc482189769)

[CAPITOLUL I – Tehnologiile folosite in dezvoltarea aplicatiei 8](#_Toc482189770)

[1.1 Limbajul C++ 8](#_Toc482189771)

[1.2 Libraria OpenCV 11](#_Toc482189772)

[CAPITOLUL II – Conceptele matematice folosite in dezvoltarea aplicatiei 13](#_Toc482189773)

[2.1 RANSAC (Random sample Consensus) 13](#_Toc482189774)

[2.2 Detectorul Harris 16](#_Toc482189775)

[2.3 Detectorul Canny 24](#_Toc482189776)

[Bibliografie 30](#_Toc482189777)

# Introducere

Inteligenţa artificială şi jocurile pe tabla de şah au un lung istoric. Începând încă din anul 1952, când Arthur Samuel a scris a serie de programe pentru jocul de dame, programe ce eventual au ajuns la nivelul la care puteau să participe în turnee oficiale de dame. În ziua de astăzi este dificial pentru un campion la şah să reuşească o simplă remiză contra celui mai bun program de joc, iar când vine vorba de jocul de dame, Jonathan Schaeffer a "rezolvat" jocul în anul 2007 demonstrând că în condiţii de joc perfect, o partida de dame se va termina mereu remiză.

Cea mai interesantă activitate într-un joc competitiv de o complexitate ridicată este urmărirea jucătorilor profesionişti. În trecut, acest lucru se făcea la radio, pe staţii dedicate sau foarte rar la televizor. Odată cu era Internetului, multe campionate de şah şi dame au început să fie oferite prin stream-uri ce pot fi urmărite direct de pe site-ul sponsorului oficial ar turneului.

Spectatorii de pe stream pot vedea pe ecran o tabla 2D ce prezintă starea jocului, tabla ce se actualizează instant în momentul în care unul dintre jucători a făcut o mutare datorită unor table speciale ce folosesc magneţi pentru a detecta poziţiile pieselor, iar tabla poate transmite starea completă a jocului prin cablu sau Wi-Fi. Acestă soluţie este mult mai costisitoare, dar mai sigură decât soluţia oferită de vederea artificială adică o camera video montată deasupra tablei de joc şi un program ce ar analiza cadrele primite de la cameră pentru a stabili starea curentă.

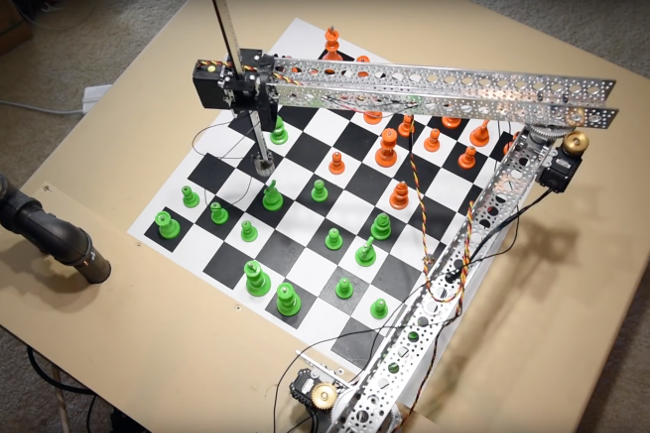


Figura 1.1: **Metoda tablei inteligente versus metoda vederii artificiale.** *Soluţia vederii artificiale este mai simplă, dar de multe ori cere compromisuri precum schimbarea culorii pieselor pentru a avea rezultate acceptabile.*

Dorinta realizării aplicaţiei a pornit din simpla pasiune pentru jocuri, în special jocurile pe tablă şi în special jocul de dame. În căutarile mele am găsit multe aplicaţii ce urmăresc jocul şah, majoritatea oferind numai notaţia algebrică, deci fiind capabile să identifice de pe ce câmp a plecat o piesă şi unde a ajuns, dar să distingă piesele una faţă de alta. Piesele de şah reprezintă o provocare deoarece când se filmează tabla de sus nu se pot distinge trăsăturile specifice pentru fiecare piesă pe cand atunci când se filmează din unghi se întalnesc des situaţii în care anumite piese maschează colţuri din componenţa tablei sau alte piese. Una dintre aplicaţiile pe care le-am întâlnit este descrisă în lucrarea "**A Computer Vision System for Chess Game Tracking"[1],** lucrarescrisă de Can Koray şi Emre Sumer. Aceştia folosesc o metodă din Matlab (metoda detectCheckerboardPoints, similară cu findChessboardCorners din OpenCV) pentru identificarea punctelor, metodă ce foloseşte un algoritm ce detectează patrulatere adiacente în imagine.

Nu am întâlnit însă o lucrare care să ia exclusiv în considerare problema jocului de dame, problemă ce rămâne similară datorită tablei comune pe care se joacă ambele jocuri, dar diferită din puncte de vedere al pieselor. Aplicaţia dezvoltată în această lucrare încearcă să analizeze o filmare a unei partide de dame, ţinând pasul cu mutările făcute de către cei doi jucători şi oferind la final notaţia algebrică ce descrie partida, notaţia algebrică fiind un de arhivare a partidelor. Validatatea mutărilor este verificată astfel încât o noua stare a tablei să fie afişată numai dacă este posibil să se ajungă la acea stare de la starea precedentă (prima stare fiind implicit starea de început clasică).

Aplicaţia a fost dezvoltată în limbajul C++ folosind librăria OpenCV. Limbajul C++ este o alegere foarte bună pentru acest tip de aplicaţii deoarece este unul dintre cele mai rapide limbaje, iar algoritmii de tip RANSAC sunt costisitori din cauza numărului mare iteraţii pe care trebuie să le execute pentru a da rezultate satisfăcătoare. Librăria OpenCV oferă o varietate de module pentru toate subdomeniile vederii artificiale, dar şi o multitudine de clase şi metode în stilul celor din MATLAB ce aduc simplitate în lucrul cu matrice.

În capitolul 1 se vor prezenta instrumentele vederii artificiale, metodele algoritmice şi conceptele matematice folosite în ordinea importanţei acestora în aplicaţie. Începând cu detectorul Harris ce ne oferă punctele din imagine ce se clasifică drept colţuri, urmând o prezentare generală a algoritmului RANSAC (algoritmul folosit în aplicaţie fiind diferit în anumite privinţe faţă de varianta clasică RANSAC), o prezentare generală a regresiei liniare pentru ca în final să trecem în revistă detectorul Canny şi implementarea acestuia în OpenCV.

În capitolul 2 se va prezenta implementarea aplicaţiei împărţită pe etape, oferind detalii specifice despre folosirea funcţiilor din OpenCV ce implementează Harris, Canny şi Hough. De asemenea se vor explica elementele specifice aplicaţiei precum verificările iniţiale asupra cadrului preluat din filmare, completarea selecţiilor rezultate din algoritmul de tip RANSAC cu o metodă ce implică regresia liniară, identificarea existenţei şi culorii pieselor şi nu în ultimul rând algoritmul de verificare a validităţii unei noi stări faţă de cea precedentă.

# CAPITOLUL I – Conceptele matematice folosite in dezvoltarea aplicatiei

**Regulile jocului de dame**

Jocul are o istorie ce se întinde asupra câtorva milenii şi este unul dintre cele mai populare jocuri pe tablă, în mare parte datorită simplităţii sale, simplitate în care se ascunde totuşi o complexitate similară cu cea a jocului de şah. Deşi există zeci de variante diferite ale jocului, variante ce folosesc alte poziţii de start, schimbă numărul de piese sau chiar regulile prin care o piesă poate captura o alta piesă, în aplicaţie avem implementată varianta englezească a jocului.

Jocul se desfăşoară numai pe Fiecare jucător începe cu 12 piese, o piesă fiind e obicei numită "pion" sau "damă". Spre deosebire de şah, în jocul de dame jucătorul ce controlează piesele negre mută primul. Ultimul rând din parte fiecărui jucător este numit "rândul reginei". Dacă un jucător reuşeste să ducă o piesă pe ultimul rând al adversarului, acea piesă va deveni "regină" şi va avea diferite avantaje în funcţie de regulile alese.



Figura 1.1.1: **Poziţia de început în varianta englezească a jocului de table.** *Aplicaţia va presupune ca partida începe din această poziţie.*

Mutările se fac în următoarele doua moduri:

* mutare: o piesă se mută pe diagonală un singur câmp, numai dacă acel câmp este neocupat. Damele nu se pot muta decât în faţă (spre adversar). Reginele pot muta şi în faţă şi în spate.
* captură sau "salt": acest tip de mutare presupune saltul pe diagonală peste o piesă inamică aflată pe, ajungând pe câmpul adiacent acesteia. Piesa inamică este scoasă in joc. Se pot executa salturi multiple în cazul în care după aterizarea pe un nou câmp se poate face o noua captură a unei piese inamice aflată pe diagonală. Este de notat faptul că jucătorii sunt obligaţi să facă o captură atunci când au ocazia, dar pot alege direcţia în care să captureze dacă au mai multe posibilităţi.

La final, jucătorul care a rămas fără piese sau fără o mutare posibilă este învins.

**Notaţia algebrică pentru jocul de dame**

Jocul are dame are un sistem de notaţie ceva mai simplu decât cel de şah. Câmpuri negre/maro sunt numerotate prin anumite valori numerice, câmpuri albe fiind irelevante.

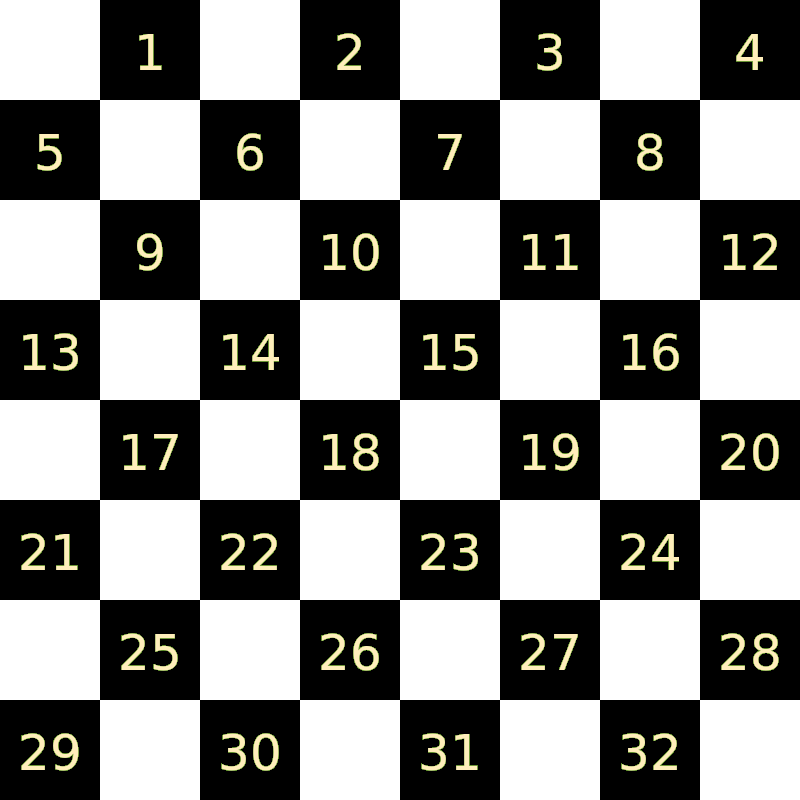


Figura 1.1.2: **Notaţia câmpurilor.** *De obicei piesele albe joacă în partea de jos, lucru respectat şi în aplicaţie.*

O mutare a unei piese de pe câmpul 19 pe câmpul 15 se notează "19-15", iar o captură de pe câmpul 19 pe o piesă de pe câmpul 15 se notează "19x10", deci important este câmpul pe care ajunge piesa după captură. O tură este alcătuită dintr-o mutare a ambelor părţi, negrul înaintea albului (de exemplu: "1. 9-14 23-19"). La final, se notează învingătorul prin "BW" (negrul a câştigat) sau "WW" (albul a câştigat).

Majoritatea notaţiilor se fac cu piesele albe în partea de jos a tablei, deci o ultimă condiţie este ca filmarea să fie făcută cu piesele albe în partea de jos a cadrului.

2.1 RANSAC (Random sample Consensus)

În inteligenţa artificială, domeniul analizei scenelor este foarte preocupat cu interpolarea unor date în anumite modele predefinite. Prima problemă a intrepretării datelor stă în găsirea datelor ce se potrivesc cel mai bine cu modelul pe care îl căutăm (problema clasificării). A două problemă este căutarea valorilor optime pentru parametrii liberi (problema estimării parametrilor), adică acele valori din ipoteza noastră ce nu pot fi deduse precis sau constrânse de către model, deci rămâne numai să fie aproximate experimental sau teoretic. Problema cea mai mare este că în practică cele două probleme nu sunt independente. De exemplu, pentru a face estimarea parametrilor este nevoie de realizarea clasificării.

O tehnică clasică este metoda celor mai mici pătrate. Scopul acestei metode este să minimizeze suma pătratelor diferenţelor dintre date. Grafic, această metodă se referă la suma la pătrat a distanţelor dintre puncte (date) şi linia de regresie; cu cât diferenţele sunt mai mici cu atât mai bine se potrivesc datele pe model. În orice astfel de metodă, punctele ce se potrivesc modelului se numesc "inliers", iar cele eronate se numesc "outliers". Avantajul pe care RANSAC îl ofera faţă de metoda celor mai mici pătrate este faptul că cea din urmă ia în calcul toate punctele din domeniul datelor, inclusiv punctele de tip outlier (cu posibilitatea de a relua algoritmul după ce s-au eliminat punctele foarte diferite de model). RANSAC produce un model construit numai din puncte de tip inlier, cu presupunea că selecţia de puncte exclusiv inlier este destul de precisă.

În loc să se caute o soluție inițială folosind toate datele și apoi să se elimine punctele invalide, RANSAC folosește o submulțime din setul inițial de date și lărgește această submulțime pe cât este posibil. De exemplu, pentru a găsi dreapta de regresie într-o mulțime de puncte, metoda RANSAC ar alege câte două puncte din mulțime (avem nevoie de două puncte pentru a defini o dreaptă), ar calcula ecuația dreptei și ar număra câte dintre celelalte puncte din mulțime sunt aproximativ pe dreaptă. La final, dreapta cu cele mai puncte ar fi rezultatul dorit.

În figura 14 avem un exemplu al unei regresii liniare făcute cu RANSAC. Punctele negre cele două puncte ce definesc dreapta, punctele roşii sunt puncte adiacente dreptei (distanţa faţă de dreaptă fiind un parametru de eroare ce permite punctului să fie valid sau nu), iar punctele verzi sunt puncte de tip outlier.

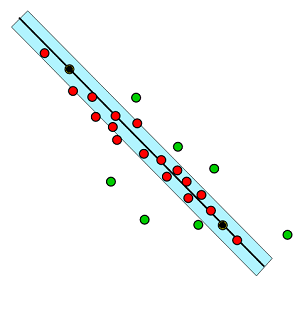


Figura 1. Exemplu regresie liniară cu RANSAC

Metoda RANSAC a fost definită în lucrarea "Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography", lucrare scrisă de către doi programatori, Martin A.Fischler și Robert C.Bolles, în anul 1980. Formal, aceștia au definit paradigma de lucru astfel:

1. Având un model ce necesită un număr N de date pentru a fi instantiat și o mulțime de puncte P cu proprietatea că mulțimea P are cardinalul mai mare decât N. De asemenea, avem un număr X de iteraţii pe catre trebuie să le execute algoritmul.
2. Selectăm în mod aleator o submulțime S de cardinal N din mulțimea datelor P.
3. Instantiem modelul (aplicăm ipoteza modelului) M și determinăm submulțimea S\* din P ce conține puncte aflate sub un anumit prag de eroare față de modelul M. Mulţimea rezultată S\* este numită mulțimea de consens a lui S.
4. Dacă mulțimea S\* are cardinalul mai mare decât o anumită valoare de prag T, numărul de puncte necesare pentru modelul cerut, atunci putem folosi mulțimea S\* pentru a calcula modelul M. Dacă mulțimea S\* are cardinalul mai decât T, selectăm o nouă submulțime S2 și repetăm procesul.
5. Dacă după X iterații nu avem nici o mulțime de consens potrivită pentru valoarea de prag T, fie folosim mulțimea cea mai apropiată de această valoare și încercam să realizăm astfel modelul, fie terminăm procesul cu eșec.

Algoritmul poate fi îmbunătăţit în funcţie de situaţie prin două metode. Prima îmbunătăţire ar fi folosirea unui mod determinist de selecţie a punctelor (în loc de cel aleator). A doua îmbunătăţire se poate face odată ce mulţimea de consens S\* a fost găsită şi modelul M\* a fost instanţiat; se mai pot adăuga în S\* puncte din P ce sunt consistente cu M\* şi apoi reinstanţierea modelului cu o mulţime mai largă.

Metoda RANSAC are trei parametri nespecificaţi:

* ipoteza sau toleranţa erorii, parametrul ce verifică daca un punct este sau nu compatibil cu modelul problemei.
* numărul de iteraţii care vor fi făcute.
* valoarea de prag T, adică numărul de puncte necesare într-o mulţime de consens pentru a putea construi modelul.

Primul parametru necunoscut, toleranţa erorii, este de obicei aflat de către programator prin experimente (de exemplu, cât de aproape să fie punctele de tip inlier faţă de dreapta de regresie).

Al doilea parametru, numărul de iteraţii sau numărul de submulţimi S incercate, poate fi ales de către programator, dar există o formulă ce ne oferă o valoare de încredere. Dacă Q este probabilitatea ca mulţimea de selecţie S să nu conţină vreun punct de tip outlier, atunci (1-Q) este probabilitatea ca S să conţină un punct de tip outlier. Dacă am face X iteraţii, probabilitatea ca toate mulţimile să fie contaminate de un punct de tip outlier este de . Vrem să il alegem pe X astfel încât probabilitatea să fie mai mică decât un anumit ε, ε fiind o valoare de prag.

Astfel ajungem la . Dacă îl includem pe X, inecuaţia poate fi scrisă mai clar:

Al treilea parametru, valoarea de prag T, este un alt parametru dedus de programator în conformitate cu modelul pe care încearcă să il instanţieze. De exemplu, pentru a instanţia o tablă de şah 8x8 avem nevoie de 81 de puncte.

2.2 Detectorul Harris

Detectarea colțurilor este folosită pentru o varietate de sarcini ce apar în procesarea imaginilor precum urmărirea obiectelor, detectarea schimbărilor, determinarea perspectivei camerei și așa mai departe. O parte importantă a algoritmului nostru pentru identificarea jocurilor de dame și șah o reprezintă partea de identificare a colțurilor. Un colț poate fi definit drept intersecția a două margini, dar poate fi de asemenea definit ca un punct pentru care există două direcții dominante ale marginilor în vecinătatea sa.

De multe ori avem de ales între ce tip de caracteristică vrem să căutăm pentru a rezolva o anumită problemă. Colțurile reprezintă o caracteristică foarte convenabilă deoarece acestea sunt relativ distincte, spre deosebire de margini (edge-uri) sau diferite texturi din imagine.

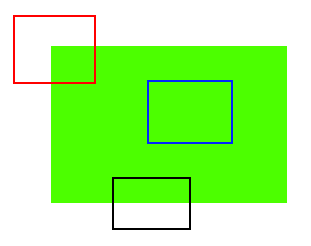


Figura 2. Margini vs edge-uri

În figura 1, dreptunghiul albastru este aproape imposibil de identificat deoarece oricum l-am muta în vecinatea sa, acesta arată la fel. Dreptunghiul negru conține o margine, dar totuși oferă o provocare similară deoarece atunci când mutăm dreptunghiul vertical (în sensul gradientului), acesta se schimbă, iar atunci când îl mutăm orizontal, de-a lungul marginii, acesta rămâne la fel. Dreptunghiul roșu conține un colț și se distinge de celelate două prin faptul ca oricum l-am muta, vom avea o variație, deci poziția sa este unică. Acest atribut al colțurilor le oferă numele de "trăsături bune" ("good features") în literatura de specialitate.

Din exemplul anterior putem de asemenea să deducem că cel mai bun plan pentru a găsi colțuri este să ne centrăm cu un dreptunghi într-un pixel și să calculăm variația dintre dreptunghiul originar și dreptunghiuri translatate cu puțin față de acesta, variațiile maxime oferindu-ne colțuri.

Chris Harris și Mike Stephens au luat această idee de a căuta regiuni din imagine ce au variație mare de intensitate în toate direcțiile și au publicat lucrarea "A Combined Corner and Edge Detector" în 1988, lucrare ce a dat naștere detectorului Harris. Formula ce stă la baza detectorului este următoarea:

=

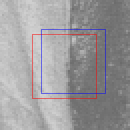


Figura 3. Exemplu alg. Harris

Luând figura 2 ca exemplu pentru a explica formula. Dreptunghiul roșu este cel originar și dreptunghiul albastru este cel mutat. Suma este făcută după X și Y, punctele ce determină dreptunghiul roșu. Formula poate fi impartitita în trei părți:

* W(X,Y) reprezintă funcția window ("fereastra"). În dreptunghiul definit de punctele X și Y funcția returnează valoarea 1, iar în afară lui valoarea 0. Astfel, sumă diferențelor la pătrat va fi relevantă doar atunci când suntem într-un pixel din dreptunghiul roșu definit de punctele X și Y. De cele mai multe ori sunt folosite funcții ce oferă anumite priorități pixelilor din zonă pe care o definesc precum funcția Gauss, deoarece această funcție este simetrică la rotație, eficientă și pune accent pe pixelii din centrul ferestrei.
* I (X+U, Y+V) reprezintă intensitatea punctului din dreptunghiul deplasat. Dacă X și Y sunt coordonatele ce definesc dreptunghiul originar, atunci U și V vor reprezenta coordonatele cu care ne mutăm față de această pentru a defini dreptunghiul albastru.
* I (X,Y) reprezintă intensitatea pixelului din dreptunghiul roșu.

Deci calculăm suma diferențelor la pătrat între dreptunghiul roșu și cel albastru (adică facem diferența pentru fiecare pixel, o ridicăm la pătrat și adunăm rezultatele). Astfel, avem un număr pentru fiecare pereche (U,V) -> E(U,V). Pentru a înțelege cum ne ajută această sumă să găsim punctele ce reprezintă colțuri trebuie să mai luăm câteva exemple de situații pentru valorile lui U și V.

Dacă am avea V=0 constant și U variabil atunci am avea o mulțime de dreptunghiuri albastre la dreapta și la stânga celui roșu, mișcarea făcându-se doar pe orizontală. În cazul nostru variația (suma diferențelor la pătrat) ar fi considerabil de mare deoarece mutarea la stânga sau la dreapta ar aduce o mai mare parte a culorii mai închise sau mai mai deschise în dreptunghiul nostru. Dacă am avea U=0 constant și V variabila, atunci am avea o mulțime de dreptunghiuri albastre deasupra și mai jos de cel roșu, caz în care am avea o variație mai mică deoarece peverticală imaginea nu variază așa mult. Așa ar arăta graficul pentru exemplul V=0, U variabil (exemplu U=0, V variabil fiind similar, doar mai redus).

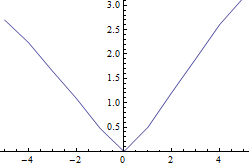


Figura 4. Grafic V=0, U variabil

De asemenea, așa ar arata plotarea graficului de mai sus (fig 4).

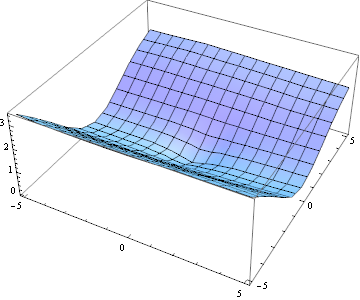


Figura 5. Plotarea graficului fig. 4

Plotarea din figura 4 ne arată faptul că variația este pe o singură direcție. Acesta nu este tipul de dreptunghiuri căutate de algoritmul nostru deoarece acestea sunt dominante vertical sau orizontal și sunt caracteristice marginilor, nu colțurilor.

Fie că luăm U=0 constant și V variabil ori V=0 constant și U variabil, variația va fi puternică.

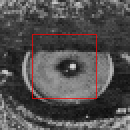


Figura 6. Exemplu de variaţie puternică

Dacă am lua un dreptunghi ca cel din imaginea 5, atunci situația ar fi complet diferită. Fie că luăm U=0 constant și V variabil sau U variabil și V=0 constant, tot vom avea o variație foarte mare la fiecare mișcare, rezultând plotarea din imaginea 6, unde putem vedea că variaţia este în toate direcţiile, nu doar vertical sau orizontal.

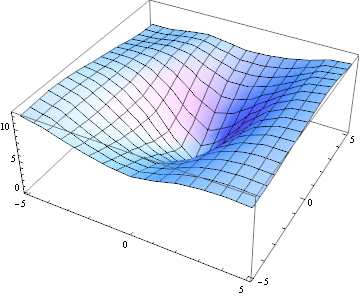


Figura 7. Plotarea fig. 6

Oricum am schimba poziția dreptunghiului, mereu va fi diferită. Forma lui E(U,V) se distinge în 3 cazuri diferite:

* Dacă E(U,V) este 0 aproape peste tot, nu avem nicio textură prezentă în imagine.
* Dacă E(U,V) este de forma figurii 4, dominantă vertical sau orizontal, probabil avem de a face cu o margine sau o textură, dar nu un colț.
* Dacă E(U,V) este de forma figurii 6, cu o textură specific, dar fără orientare domintanta, atunci avem un tip de dreptunghi ce poate indica un colț.

În mod normal, nici nu este nevoie să calculăm E(U,V). Dorim numai să aflăm forma vecinătății sale în jurul punctului (U,V) = (0,0) deci avem nevoie doar de expansiunea Taylor a lui E(U,V) în (0,0).

Detectorul Harris este implementat în librăria OpenCV. Revenind la formula de bază a detectorului:

=

După aplicarea expansiunii Taylor și a altor câțiva pași matematici, ajungem la forma: ≈ M unde M este . si sunt derivatele în direcția x respectiv y. Acestea sunt găsite prin operatorul Sobel, de asemenea implementat în OpenCV.

Acum urmează partea principală a algoritmului, funcția ce ne oferă scorul fiecărui dreptunghi și o indicație asupra prezenței unui colț în imagine.

In formula, avem:

* si sunt valorile proprii ale lui M.

Cu aceste valori se fac deducțiile pe care le-am prezentat și mai devreme.

* Dacă |R| este mic, lucru care se întâmplă atunci când ⋋\_1 și ⋋\_2 sunt mic, avem o regiune care nu este nici margine nici colț.
* Dacă R<0, lucru care se întâmplă când ⋋\_1>> ⋋\_2 sau ⋋\_2>> ⋋\_1, regiunea este o margine.
* Dacă R este mare, lucru care se întâmplă când ⋋\_1 și ⋋\_2 sunt ambele mari, avem de a face cu un colț.

Antetul funcției din OpenCV arată astfel:

void cv::cornerHarris (InputArray **src**,

OutputArray **dst**,

int **blockSize**,

int **ksize**, double k,

int **borderType**=BORDER\_DEFAULT)

Această funcție rulează detectorul Harris și calculează R în toate punctele din imagine, colțurile reprezentând puncte de maxim local. Parametrii funcției:

* src - o imagine 8-bit pe un singur canal pe servita ca intrare (la nivel de cod - matrice).
* dst-imaginea în care se vor pune rezultatele R.
* blocksize-mărimea vecinătății.
* ksize-parametru pentru mărimea operatorului Sobel. Trebuie să fie de valoare 1, 3, 5 sau 7.
* k - parametru specific Harris folosit în formula
* borderType - metoda de extrapolare a pixelilor.

Algoritmul dă rezultate bune și în 2D și în 3D, în condiții relativ potrivite de contrast. Un astfel de exemplu ar este imaginea de mai jos. Se observă că în imaginea 3D cu piesele geometrice există puncte imprecise sau puncte care nu neapărat așteptate că rezultat, dar care totuși convin definiției clasice de colț căutată în detectorul Harris.

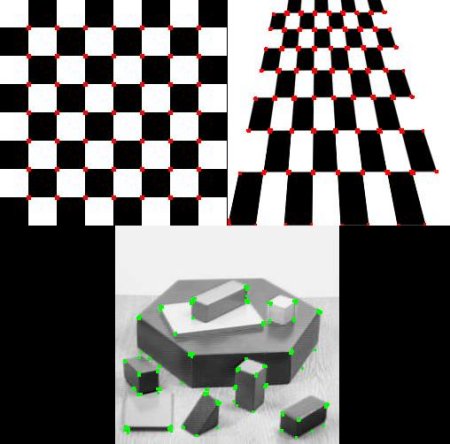


Figura 8. Exemplu aplicare Harris

Au existat totuși optimizări asupra detectorului Harris. În 1994, Jianbo Shi și Carlo Tomasi au propus o modificare asupra formulei cu care este obținut scorul R.

În loc de formula veche: , care odată ce înlocuim cu valorile proprii si devine . Cei doi au propus:

Această modificare reduce acumularea de puncte în vecinătăți ale imaginii, punctele de la ieșire fiind mult mai relevante căutării și mai ușor de manipulat la filtrarile ce urmează a fi făcute.

În aplicația prezența, folosim detectorul Shi-Tomasi deoarece tabla de șah generează uneori puncte multiple în zone foarte restrânse. Detectorul Shi-Tomasi este implementat în OpenCV în funcţia **goodFeaturesToTrack :**

void goodFeaturesToTrack(InputArray **image**,

OutputArray **corners**,

int **maxCorners**,

double **qualityLevel**,

double **minDistance**,

InputArray **mask**=noArray(),

int **blockSize**=3,

bool **useHarrisDetector**=false,

double **k**=0.04 )

Parametrul foarte important pentru eliminarea punctelor foarte aproapiate este **minDistance**. Acest parametru reprezintă distanța euclidiană minimă între două puncte detectate. Parametrul **qualityLevel ne ajută să stabilim o valoare de prag care să respingă anumite colţuri găsite. Dacă cel mai bun colţ găsit are un scor de 1500, atunci dacă setăm acest parametru cu 0.01, produsul dintre cel mai bun scor şi nivelul de calitate va fi limita inferioară pentru celelalte colţuri (în cazul de faţă va fi 15).**

Alt parametru foarte relevant în această funcție și absent în varianta Harris este **maxCorners**, acesta oferindu-ne șansa de a limita numărul de colțuri detectate, foarte util în problemele în care știm numărul de puncte relevante (81 în cazul tablei de șah 8x8). Parametrul **k** este parametrul liber al detectorului Harris, iar **blockSize** este relevant pentru mărimea matricii de covarianţa făcuta pe vecinătatea fiecărui punct.

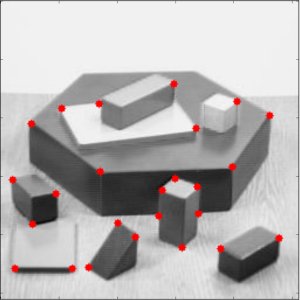


Figura 9. Exemplu Shi-Tomasi

In figura 8 vedem aplicarea detectorului Shi-Tomasi asupra aceleiasi imagini ca in figura 7. Numarul de puncte detectate este mai mic, fie din cauza parametrului maxCorners, fie din cauza distantei euclidiene minime, dar nu mai avem aglomerari de puncte intr-o singura zona.

2.3 Detectorul Canny

Detectoarele de margini sunt o parte esențială în sistemele de vedere artificială. Detectarea marginilor este un proces ce servește la simplificarea analizei imaginilor în mod drastic prin reducerea datelor ce trebuie a fi procesate, dar în același timp prezervarea informației legată de structura obiectelor. Există o mare diversitate în domeniul aplicațiilor de detectare a marginilor, dar cele mai multe au o mulțime comună de cerințe. John Canny a clasificat aceste cerințe în lucrarea să din 1986: "A Computațional Approach to Edge Detection" și a conceput o soluție ce răspunde la acestea:

* Detecția trebuie să se facă cu o rată mică de eroare, cât mai puține fals-pozitive sau margini ratate de detector.
* Punctul de margine detectat trebuie să se situeze în centrul marginii. Distanța trebuie pixelii de margine detectați și pixelii de margine adevărați trebuie să fie minimă.
* Marginile detectate trebuie să fie marcate doar o singură data și când este posibil, zgomotul din imagine nu trebuie creeze fals-pozitive.

Pentru a satisface aceste cerințe, algoritmul lui Canny lucrează în mai multe etape. Pasul 1 presupune filtrarea zgomotolui din imagine înainte de a începe detecția este esențială. Filtrul Gaussian poate fi implementat folosind o simplă mască și este folosit mod exclusiv de către algoritmul lui Canny. Odată calculată o mască, netezirea imaginii prin filtrul Gaussian se face prin metoda standard de convoluție. Masca trebuie să fie iterată peste toată imaginea pe rând deoarece aproape de fiecare dată masca este mai mică decât imaginea. O regulă importantă este aceea că lărgimea măștii Gaussiene scade senzitivitatea detectorului la zgomot cu cât este mai mare.

Un exemplu de masca Gaussiana 5x5 cu **ϭ=1.4**:

După netezire, următorul pas este calcularea gradientului imaginii pentru a obţine marginile. Algoritmul Canny foloseşte 4 filtre pentru a detecta margini orizontale, verticale şi diagonale. Cel mai des este folosit operatorul Sobel (Roberts sau Prewitt fiind alte variante) ce returnează derivata pe orizontală () si pe verticală (). Operatorul Sobel foloseşte măşti de convoluţie 3x3, una ce estimează gradientul orizontal si alta pe cel vertical.

Scorul marginii este dat de formula:

Odată ce direcția marginii este cunoscută, următorul pas este să o legăm de o direcție ce poate fi trasată în imagine. Există 4 direcții la care sunt reduse toate posibilitățile de unghiuri: 0 grade (orizontal), 45 de grade (diagonala cu pantă pozitivă), 90 de grade (vertical), 135 de grade (diagonala cu pantă negativă). Unghiurile intre 0 si 22.5, 157.5 si 180 vor fi unghiuri de 0 grade.

* Unghiurile între 22.5 si 67.5 vor fi unghiuri de 45 de grade.
* Unghiurile între 67.5 si 112.5 vor fi unghiuri de 90 de grade.
* Unghiurile între 112.5 si 157.5 vor fi unghiuri de 135 de grade.

Următorul pas este "subţierea" marginii printr-un proces de eliminare (setare la 0) a pixelilor ce nu sunt consideraţi că fac parte din margine. Această operaţie lasă doar cei mai pronuntaţi pixeli din margine. In multe implementări metoda aleasă este următoarea:

* Dacă unghiul de la pasul anterior a fost 0, comparăm scorul G al pixelului curent cu scorurile pixelilor din stânga și din dreapta. Dacă pixelul curent are un scor mai mare decât ambii pixeli, atunci acesta rămâne, altfel este suprimat.
* Dacă unghiul de la pasul anterior a fost 90, comparăm scorul G al pixelului curent cu scorurile pixelilor de deasupra și de jos. Dacă pixelul curent are un scor mai mare decât ambii pixeli, atunci acesta rămâne, altfel este suprimat.
* Dacă unghiul de la pasul anterior a fost 135, comparăm scorul G al pixelului curent cu scorurile pixelilor din dreaptă sus și stânga jos. Dacă pixelul curent are un scor mai mare decât ambii pixeli, atunci acesta rămâne, altfel este suprimat.
* Dacă unghiul de la pasul anterior a fost 45, comparăm scorul G al pixelului curent cu scorurile pixelilor din dreapta jos și stânga sus. Dacă pixelul curent are un scor mai mare decât ambii pixeli, atunci acesta rămâne, altfel este suprimat.

Nu în ultimul rând, algoritmul folosește două valori de limită (threshold) pentru a evita marginile sacadate și despărțite. Dacă folosim numai o valoare limită inferioară sau numai una superioară, inevitabil vom pixeli ce vor trece și pixeli apropiați ce vor pica, astfel reducând credibilitatea marginii descoperite. Canny folosește două valori de limită și astfel orice pixel cu o valoare mai mare decât valoarea minimă este considerat pixel ce aparține marginii și orice pixel cu valoarea mai mică decât valoarea maximă este de asemenea considerat pixel ce aparține marginii.

Antetul functiei in OpenCV arăta astfel:

void Canny(InputArray **image**,

OutputArray **edges**,

double **threshold1**,

double **threshold2**,

int **apertureSize**=3,

bool **L2gradient**=false )

Semnificatia parametrilor:

**image** - imaginea dată la intrare.

**edges**- mapa ce conţine marginile detectate.

**threshold1** - prima valoare de limită pentru ultimul pas (hysteresis).

**threshold2** - a doua valoare de limită pentru ultimul pas.

**apertureSize** - parametru pentru operatorul Sobel.

**L2gradient** - variabila bool care in cazul true foloseste o formulă mai precisă pentru scorul gradienţilor.

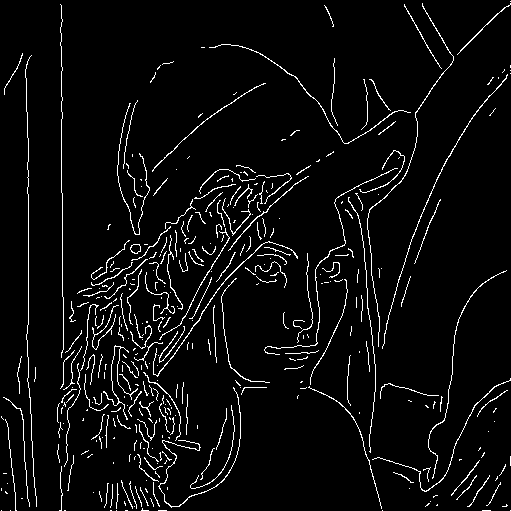


Figura 10. Exemplu Canny

2.4 Regresia liniară

În aplicație vom avea nevoie să găsim dreapta ce se potrivește cel mai bine pentru un grup de puncte, astfel încât să putem găsi cele 8 linii verticale și cele 8 linii orizontale prezente pe tablă de șah și să putem găsi intersecțiile acestora, adică cele 81 de puncte ce definesc tablă. Presupunem că avem un set de N puncte cu asbscisa X si ordonata Y: (), (), (),...,(),... (. Vrem să găsim ecuaţia unei drepte ce potriveste cel mai bine aceste puncte. Ecuaţia unei drepte este de forma:

**M** este panta dreptei definită de ecuație, iar **C** este locul în care dreapta intersectează axa ordonatelor. Astfel, vrem să găsim cel mai bun M și cel mai bun C pentru mulțimea noastră de puncte. Dacă soluția nu este trivială (atunci când punctele sunt deja pe aceeași linie) știm că va exista o eroare (linia nu va intersecta punctele, distanța de la linie către puncte reprezentând eroarea). Cum se observă şi în figura 2.4.1, formula de calcul a reziduului pentru într-un singur punct este:

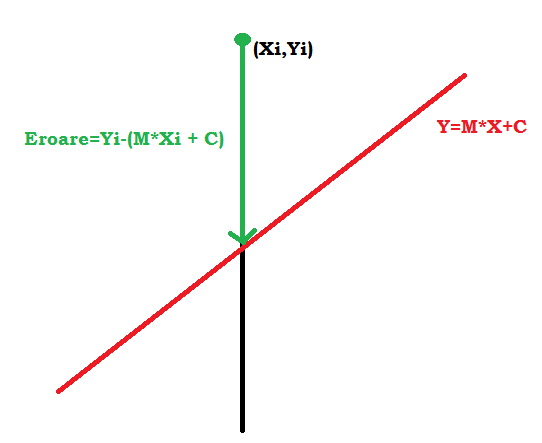


Figura 2.4.1: **Exemplu reziduu regresie liniară.** *Distanţa de la punct către linia de regresie reprezintă reziduul.*

Vrem să găsim dreapta cu proprietatea că suma reziduurilor este minimă. Două probleme ce apar constau în faptul că un reziduu poate fi şi pozitiv şi negativ, dar şi în faptul că este nevoie de o metodă ce face un calcul ponderat între punctele ce sunt aproape de linie şi punctele ce sunt foarte departe. Soluţia pentru ambele probleme este ridicarea la pătrat a reziduurilor.

Pentru a avea suma reziduurilor la pătrat, definim R:

Pentru a găsi minimul acestei ecuaţii trebuie să ne folosim de faptul că valoarea minimă a lui R se află în locul unde prima derivată este 0. Cele doua variabile sunt M şi C deci avem:

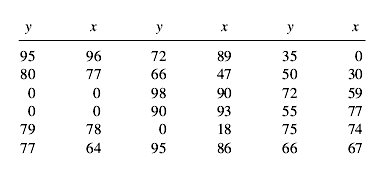
Putem separa cele două ecuaţii astfel:

Dacă simplificăm înmulţirile cu 2 şi aranjăm termenii după semn vom avea următoarele ecuaţii:

Acum putem să scoatem constantele în faţa sumelor (M şi C), dar de asemenea suma de N ori din C este N\*C. Aceste schimbări duc la ecuaţiile:

Într-un final, dacă încercăm să scoatem variabilele care ne interesează, M şi C, vom avea:

Se observă din forma ecuaţiilor că putem avea un caz de împărţire la 0 când toate valorile punctelor de pe abscisă sunt egale. Acest lucru se întâmpla când toate punctele sunt pe aceeaşi dreaptă **verticală**, lucru ce poate fi rezolvat pentru scopul aplicaţiei. Un exemplu de regresie luat din cartea **"Linear Models in Statistics"**:



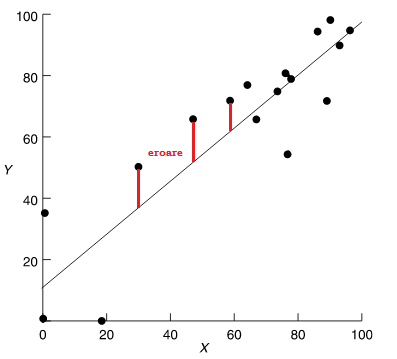
****

Figura 2.4.2: **Exemplu regresie liniară.** *Cele 18 puncte din tabel sunt fitate de linia din graficul de jos. Distanţele de la puncte către linie reprezintă reziduuri.*

**CAPITOLUL III - Implementarea aplicaţiei**

Aplicaţia trece prin mai multe etape pentru a ajunge la datele de ieşire (notaţia algebrică a jocului). Cele 3 ecrane vizibile pe parcursul rulării vor fi: ecranul în care se poate vedea filmarea neafectată de program, ecranul care reprezintă ultimul cadru analizat şi ecranul ce conţine reprezentarea în 2D a cadrului analizat. Deşi fiecare etapă este mai detaliată, în principal programul se împarte astfel:

1. Iniţial se deschide video-ul cu filmarea jocului şi la fiecare 10 cadre derulate se capturează unul pentru a fi analizat.
2. Se verifică dacă una dintre mâinile jucătorilor este în imagine, caz în care programul merge mai departe fără a analiza cadrul.
3. Dacă totul este în regulă până acum se poate începe analiza propriu-zisă, analiză ce începe cu aplicarea detectorului Harris.
4. Odată ce vectorul de colţuri a fost iniţializat, se începe procesul de RANSAC cu 50 de selecţii pe care se aplică ipoteza noastră (punctele din selecţie fac parte din reprezentarea tablei de şah). La final, avem ca date de ieşire selecţie cu cele mai multe colţuri ce respectă ipoteza.
5. Selecţia obţinută la pasul interior nu este aproape niciodată completă cu toate cele 81 de puncte necesare definirii tablei. Completarea se face pe baza unei regresii liniare făcute pentru fiecare linie şi coloana ce defineşte tabla de joc (9 linii verticale şi 9 linii orizontale) urmând să se găsească punctele de intersecţie pentru aceste linii şi să se obţină 81 de puncte. Aceste puncte se reunesc cu punctele din selecţie (dacă un punct lipseşte din selecţie se va completa cu unul generat în urma intersecţiilor dintre drepte). Rezultatul acestui pas este vectorul ce conţine cele 81 de colţuri necesare pentru identificarea tablei de şah în cadru.
6. Se obţin cele 64 câmpuri pe baza celor 81 de colţuri şi se analizează fiecare câmp pentru a se determina dacă avem un câmp liber, un câmp cu o piesă albă sau un câmp cu o piesă neagră. În acelaşi timp se contruieşte şi reprezentarea bidimensională a tablei pentru a fi afişată.
7. În final se face analiza mutărilor şi a validităţii poziţiei din cadrul curent faţă de poziţia anterioară. Dacă de la poziţia precedentă nu se poate ajunge printr-o singură mutare sau captură la poziţia din cadrul curent atunci acesta nu va fi afişat.

3.1 Validitatea filmării date ca intrare

Filmarea dată ca intrare în program trebuie să conţină o partidă de dame ce începe din poziţia de început tipică în varianta englezească a jocului (12 versus 12 pe liniile 1,2,3 respectiv 6, 7 şi 8).

Camera nu trebuie să fie atât de aproape de tablă încât să nu fie vizibile anumite colţuri, distanţa minină oferind cazul în care tabla reprezintă întregul cadru (programul presupune ca un câmp este maxim 1/8 din lungimea şi lăţimea cadrului - cazul în care tabla este întregul cadru), iar maximul depinde de calitatea filmarii la distanţă, dar tabla trebuie să ocupe măcar jumătate din întregul cadru. Este de asemenea de preferat ca fundalul să fie monocrom sau cel puţin lipsit de obiecte ce pot genera colţuri în imagine deşi vom vedea ca ipoteza RANSAC evită colţurile ce nu se află pe tablă. Fundalul poate fi o problemă deoarece detectorul Harris detectează un anumit număr maxim de colţuri, iar unele colţuri ce nu se află pe tablă pot avea un scor mai bun conform detectorului decât colţurile de pe tablă, cauzând lipsa unor colţuri importante încă de la detecţia iniţială. Problema este în general rezolvată de completarea realizată dupa procesul de RANSAC, dar în cazuri extreme fundalul poate face imposibilă determinarea punctelor ce definesc tabla în imagine.

Un ultim punct adus în discuţie când vine vorba de analiza imaginilor este luminozitatea. Trebuie să avem grijă ca filmarea să nu fie mult prea puternic sau mult prea slab luminată, să nu conţină reflexii pe tablă cauzate de proximitatea sursei de lumină faţă de tablă şi să nu existe o diferenţă foarte mare de între modul în care cade lumina asupra unei parţi a tablei faţă de altă parte.

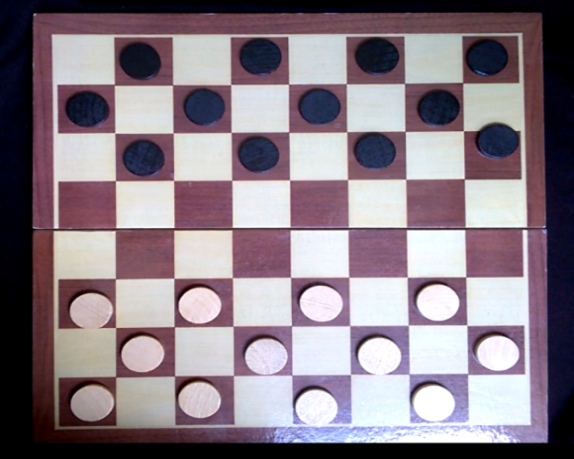


Figura 3.1.1: Exemplu de cadru adecvat. *Această filmare a fost făcută afară deoarece soarele este o sursă de lumină direcţională şi se pot distinge foarte clar piesele negre de câmpurile maro ale tablei. Se pot totuşi observa mici probleme cu reflexiile pe tablă, dar acestea nu sunt grave.*

3.2 Preluarea cadrelor şi primele verificări

Odată ce intrarea în program este corespunzătoare putem începe să vorbim despre execuţia în sine. Cadrele sunt capturate cu ajutorul clasei VideoCapture din OpenCV ce deschide un video (formate comune precum mp4 sau avi sunt suportate de acestă clasă) la o anumită cale din memorie. Programul se desfăşoară într-o buclă "for" infinită ce ia cadru cu cadru filmarea şi se opreşte doar atunci când nu mai există alte cadre în filmare (mai specific, atunci cand instanţa clasei VideoCapture ne oferă o matrice goală).

Deoarece majoritatea camerelor moderne filmează în rezoluţii relativ mari, imediat după preluare cadrul este supus unei redimensionări cu 50% pe ambele dimensiuni. Astfel este mai uşoară afişarea mai multor ecrane pe timpului rulării aplicaţiei.

Se încearcă analiza unui singur cadru la fiecare alte 10 cadre deoarece tot procesul durează în jur de o secundă, în special datorită execuţiei celor 50 de selecţii din RANSAC, lucru care ar îngreuna foarte mult execuţia generală a programului, iar urmărirea execuţiei pe cele 3 ecrane ar fi foarte dificilă.

Prima verificare importantă se face pentru a determina dacă mâna unui jucător se află în cadru, acest lucru fiind detrimental deoarece umbra mâinii poate cauza fals pozitive când vine vorba despre piesele de culoare neagra sau poate ascunde anumite piese. Un cadru generat în aceste două cazuri nu va ajunge să fie afişat pentru că cel mai probabil nu va corespunde stării anterioare, vom avea totuşi foarte multe calcule executate pe nişte cadre ce nu pot oferi rezultatul dorit. Verificarea se face pe baza detectorului de muchii Canny, folosit în contextul funcţiei findContours() din OpenCV. Cum a fost explicat în capitolul anterior, detectorul Canny foloseşte un filtru Gaussian şi un proces de hysteresis pentru a identifica marginile într-o imagine dată. Matricea (imaginea) rezultată nu va mai avea elemente cu 3 canale RGB, va fi o matrice binară cu valori de 0 şi 255.

Funcţia findContours() găseşte componentele conexe din matricea binară dată de Canny, adică contururile obiectelor din imagine. Antetul funcţiei arată astfel:

void findContours( InputOutputArray **image**,

OutputArrayOfArrays **contours**,

int **mode**,

int **method**,

Point **offset**=Point())

Primul parametru, "image", este imaginea rezultată din detectorul Canny ce conţine marginile. Parametrul "contours" este un vector ce va conţine câte un vector pentru fiecare contur din imagine, aceşti vectori conţinând la rândul lor fiecare punct ce face parte din contur. Deoarece funcţia are un parametru opţional prin care ne poate întoarce şi o ierarhie a contururilor (dacă unele contururi sunt interne altor contururi) sub formă de arbore, parametrul de "mode" ne ajută să alegem cum vrem să fie structurat arborele (pe N nivele sau pe 2 nivele) sau dacă vrem să nu fie calculat deloc, caz ales în aplicaţia faţă deoarece noi dorim doar numărul de contururi. Ultimul parametru, "offset, este util numai daca am aplicat funcţia doar o parte din imaginea noastră (o anumită regiune de interes) şi vrem ca în vectorul de contururi punctele să fie mutate cu o anumită valoare.

O abordare timpurie a fost căutarea celui mai lung contur dat de această funcţie, deoarece mâna jucătorului ar trebui să determine o componentă mai lungă decât orice alta de pe tablă. Acest lucru a fost invalidat în teste atunci când s-a observat că nu putem avea cu incredere o singură componentă conexă care să reprezinte întreagă mână, dar şi faptul că natura tablei de şah cauzează contururi foarte lungi între liniile dintre piese, cum se poate observa în figura 3.2 . Soluţia finală a fost bazată pe numărul de contururi prezente în vectorul de ieşire al funcţiei findContours. Mâna unui jucător măreşte cu aproape 100 numărul de contururi prezente în cadru, oferind un foarte bun indiciu.

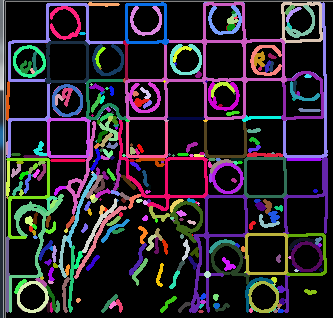
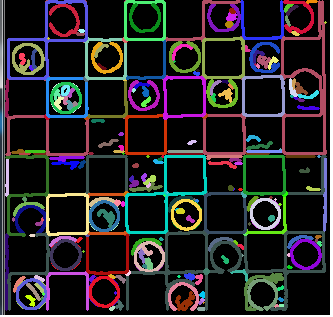


Figura 3.2.1: findContours pe Canny: Cu şi fără mâna unui jucător. *Fiecare contur este desenat cu o culoare aleatoare din spaţiul RBG. Se pot observa în stânga contururile prezente în mod normal pe tablă, iar în dreapta se observă o abundenţă de contururi pe mâna jucătorului*

3.3 Aplicarea detectorului Harris

Am prezentat în capitolul 2 cum arată antetul funcţiei ce implementează detectorul Harris in OpenCV. În aplicaţie folosim detectorul în varianta Shi-Tomasi, lucru ce ne permite să setăm o distanţă minimă între colţuri şi să ne asigurăm că nu vom avea mai multe puncte redundante. Am setat distanţa minimă 10 şi parametrul "qualityLevel" la 0.01, numărul de colţuri maxime 90 şi mărimea matricei de covarianţă 3 (blockSize).

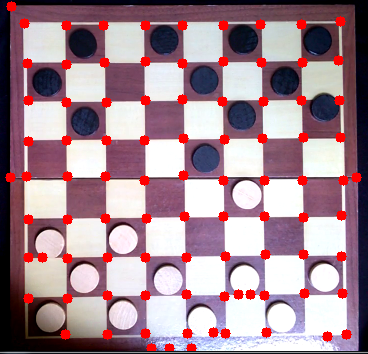
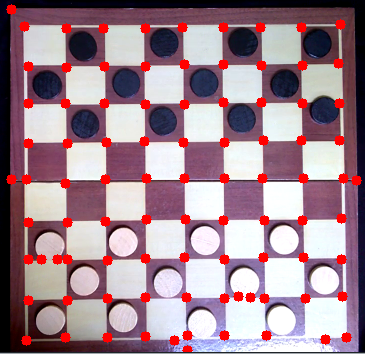
****

Figura 3.3.1: Detectorul Harris în aplicaţie: *Detectorul Harris este setat să returneze primele 90 de puncte (ordonate conform scorului). În cazul de faţă avem cele 81 de puncte ale tablei, dar avem 9 puncte in diverse locuri ale tablei ce cauzează detecţii false (jumătatea tablei din cauza şanţului ce strânge tabla în două bucăţi, marginile tablei faţă de fundal, diverse piese ce nu sunt poziţionate fix in centrul câmpului).*

Uneori se întâmplă ca detectorul Harris să nu returneze toate punctele de pe tablă (cum se vede în imaginea din dreapta din figura 3.3.1 - punctul din stânga jos), fapt cauzat de prezenţa unor colţuri pronunţate ce nu fac parte din tabla noastră. Colţurile "false" scot anumite colţuri de pe tablă din top-ul celor 90 de puncte returnate.

3.4 Metoda RANSAC de selecţie a colţurilor

Următoarea parte din execuţia programului este algoritmul de tip RANSAC. Acesta diferă totuşi de structura tipică RANSAC prezentată în capitolul 2. În cazul nostru, scopul nostru nu este să iniţializăm modelul în momentul în care am găsit o selecţie cu un anumit număr de puncte de tip inlier sau după ce algoritmul returnează cea mai bună selecţie. Numărul de puncte inlier necesare în selecţie pentru definirea tablei de şah este de 81, număr prea mare pentru a fi obţinut cu încredere de către un algoritm de tip RANSAC, mai ales în situaţia în care avem puncte lipsă încă din stadiul de găsire a colţurilor cu detectorul Harris. Algoritmul din aplicaţie returnează între 65 si 75 de puncte din cele 81 necesare, restul fiind completate la pasul următor din aplicaţie. Din acest motiv folosim un algoritm RANSAC ce conţine doar primii 2 paşi din modelul clasic RANSAC şi un pas 3 modificat.

Primul pas este să stabilim numărul de alegem numărul de puncte dintr-o selecţie. Pentru că nu ne aşteptăm să primim toate cele 81 de puncte necesare modelului nostru, putem să alegem mărimea selecţiei chiar de 81. Numărul de selecţii încercate (iteraţii) a fost ales experimental la valoarea 50. Este de notat faptul că puţine iteraţii ne oferă o mulţime de selecţie destul de incompletă (55-65 de puncte inlier) pe când un număr mare iteraţii ar implica un timp de execuţie destul de mare.

Al doilea pas reprezintă pur şi simplu alegerea în mod aleator a 81 de puncte din cele 90, fiind siguri ca nu există duplicate în mulţimea rezultată.

Cel de-al treilea pas implică aplicarea ipotezei ce încearcă să definească tabla de joc. O selecţie de 81 de puncte intră în funcţia de aplică ipoteza. Ipoteza este următoarea:

"Un punct face parte din colţurile tablei de şah dacă şi numai dacă are cel puţin un vecin la o distanţă mai mică decât 1/8 din numărul de coloane al matricei ce reprezintă cadrul, dar mai mare decât 1/16 din numărul de coloane al matricei."

Distanţa dintre două puncte este calculată cu formula:

Distanţa de la punctul pe care este aplicată ipoteza către vecinul său trebuie să încadreze în intervalul (1/16 \* numărul de coloane al matricei cadrului , 1/8 \* numărul de coloane al matricei cadrului) deoarece, cum a fost menţionat la începutul capitolului 3 (validitatea filmarării), tabla trebuie să reprezinte minim jumătate din conţinutul cadrului (o coloană reprezentând 1/16 din imagine) şi maxim tabla să reprezinte întregul conţinut al imaginii (o coloană reprezentând 1/8 din imagine). Totuşi, ştim ca tabla este alcătuită din linii verticale şi orizontale, deci putem alege vecinii doar din punctele ce se află aproximativ pe aceeaşi linie.

Astfel, algoritmul ce verifică ipoteza are 3 paşi:

* pentru un punct din selecţie se caută punctele ce se află pe aceeaşi linie sau coloană cu punctul respectiv. Exemplu figura 3.4.1.
* din mulţimea de puncte de la pasul anterior, se numără vecinii, adică punctele din intervalul intervalul (1/16 \* numărul de coloane al matricei cadrului , 1/8 \* numărul de coloane al matricei cadrului). Exemplu figura 3.4.1.
* dacă punctul are cel puţin un vecin de la pasul anterior, punctul verifică ipoteza şi este pus în mulţimea ce reprezintă selecţia filtrată (de punctele ce nu verifică ipoteza).

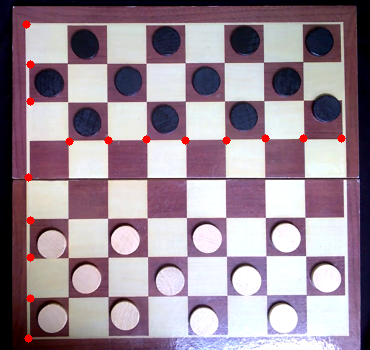
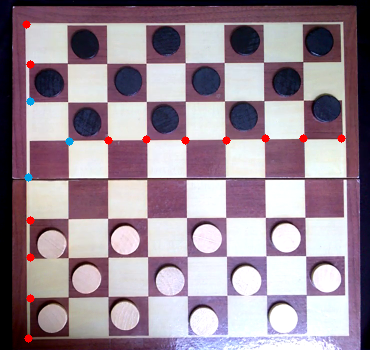
****

Figura 3.4.1: Exemplu paşi ipoteză RANSAC. *Exemplul prezintă ipoteza aplicată pentru pentru primul colţ de pe linia 4 a tablei. În stânga vedem pasul 1, toate punctele ce au abscisa sau ordonata aproximativ egală cu cea a punctului. În dreapta vedem pasul 2, punctele albastre fiind punctele ce se alfă în intervalul de distanţă stabilit şi se pot numi vecini.*

Algoritmul ales ne asigură că nu vom avea puncte exterioare tablei (figura 3.4.2). Aceste puncte ar detrimentale analizei ce urmează a fi prezentată în capitolul 3.6 deoarece un punct exterior ar lua locul unui punct, iar segmentarea pe câmpuri a tablei ar fi incorectă. De asemenea, în figura 3.4.2 se observă şi "ratarea" unor puncte ce se află pe piesele de dame, dar această problemă este cauzată de faptul că acele puncte au abscisa aproximativ egală cu punctele de pe linia tablei. Se mai observă şi că unele puncte sunt eliminate în mod greşit. Ambele probleme sunt rectificate de completarea acestei selecţii prin regresie liniară, metodă prezentată în capitolul 3.5.

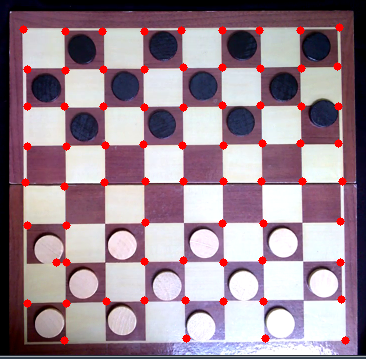
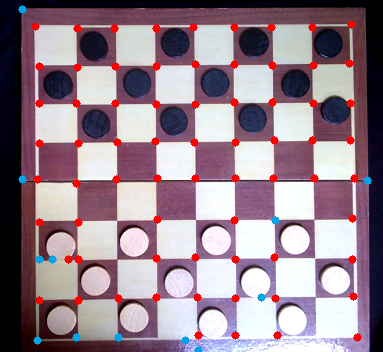


Figura 3.4.2: Exemplu selecţie înainte şi după aplicarea ipotezei. *În partea din stânga avem o selecţie alcătuită din 81 de puncte alese aleator din cele 90 de detectorul Harris (figura 3.3.1, imaginea din stânga). Ipoteza noastră a eliminat punctele albastre, oferind ca rezultat imaginea din dreapta.*

Într-un final, vom avea 50 de selecţii precum cea din figura 3.4.2. Selecţia cu numărul cel mai mare număr de puncte de tip inlier va fi returnată înapoi în program. S-a observat în mod experimental că cea mai bună selecţie are de obicei între 65 şi 75 de puncte.

3.5 Procesarea celei mai bune selecţii - completarea prin regresie liniară

Pentru completarea punctelor vom folosi împărţi selecţia returnată de pasul precedent în linii (deci implicit şi în coloane), astfel încât să avem 9 vectori de puncte, fiecare vector având puncte aflate pe o linie de pe tablă. Fiecare vector ar trebui să aibă 9 puncte, dar pentru că selecţia nu este completă numărul de puncte va fi variat (este important să existe măcar un punct pe fiecare linie/coloană din cele 9).

Vom aplica procesul de regresie liniară prezentat în capitolul 2.4 pentru a obţine ecuaţiile (valorile M şi C) celor 18 drepte (9 drepte verticale date de punctele de pe coloană şi 9 orizontale date de punctele de pe linie). Acum putem face intersecţiile dintre aceste drepte pentru a afla cele 81 de puncte ce pot defini tabla de şah în imagine. Presupunem că avem:

ecuaţia unei drepte orizontale.

ecuaţia unei drepte verticale.

Pentru că panta unei drepte verticale este 0 (aproximativ 0 în cazul nostru), putem spune că , având:

Deci ştim că punctul de intersecţie va avea ordonata în (locul unde dreapta orizontală intersectează axa Y). Dacă înlocuim cu Y în a doua ecuaţie:

După ce îl mutăm pe în dreapta şi apoi împărţim cu :

Odată ce avem cele 81 de puncte, am putea trece direct la segmentarea tablei în câmpuri, dar mai întâi este de preferat să unim rezultatul regresiei liniare cu selecţia incompletă de la care am plecat, deoarece detectorul Harris oferă o mai mare încredere faţă de aproximările făcute de regresie. Vom combina rezultatele cu o simplă regulă: "Dacă un anumit colţ există în selecţia RANSAC, acel punct va ajunge în mulţimea finală de colţuri, altfel va fi înlocuit cu un colţ rezultat din procesul de regresie".

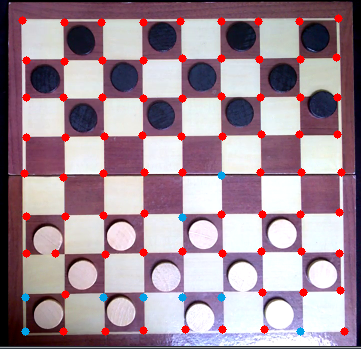
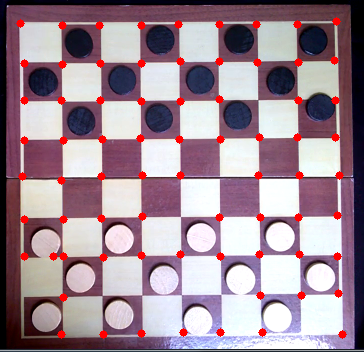
****

Figura 3.5.1: Exemplu completare regresie liniară. *În stanga avem un exemplu de selecţie ce trrebuie completată. Puncte albastre sunt puncte găsite prin intersecţia dreptelor de regresie.*

3.6 Identificarea existenţei şi culorii pieselor

Odată ce ştim unde se află fiecare din cele 81 de colţuri ale tablei putem începe să segmentăm tabla pentru a obţine fiecare câmp individual şi să încercăm să verificăm fiecare câmp. Datorită faptului că avem colţurile împărţite într-un mod organizat (ştim cele 9 puncte pe fiecare linie sau coloană) putem să segmentăm foarte uşor tabla şi să obţinem cele 64 de câmpuri (un dreptunghi este definit de două puncte, deci lucrul cu indecşi este trivial).

Dacă ştim poziţiile celor două puncte ce definesc un câmp atunci putem să începem să deducem dacă pe acel câmp se află o piesă şi care este culoarea acesteia. Putem de la început să nu ţinem cont de câmpurile albe de pe tablă deoarece acestea vor fi mereu libere. Astfel, trebuie doar să distingem între:

* câmpuri pe care se află o piesă neagră.
* câmpuri pe care se află o piesă albă.
* câmpuri libere.

****

Figura 3.6.1: Exemple câmpuri după segmentare. *Câte un exemplu din fiecare tip de câmp pe care îl putem avea.*

Pentru a determina existenţa şi culoarea pieselor ne vom folosi de spaţiul RGB (red-green-blue). Cum este menţionat în nume, spaţiul RGB este compus din culori formate prin adunarea de roşu, verde şi albastru. O culoare din spaţiul RGB este un vector de 3 elemente, fiecare element având valori între 0 şi 255, 0 reprezentând absenţa completă a culorii respective. Astfel, vectorul (0,0,0) este complet negru, iar valori mici în cele 3 componente vor rezulta o nuanţă de negru, pe când vectorul (255,255,255) este complet alb, valori mari în cele 3 componente oferind nuanţe de alb. Diferite nuanţe de maro (pentru identificarea câmpurilor libere) au vectori RGB diferiţi, deci cel mai uşor va fi să verificăm dacă un câmp conţine o piesă albă sau o piesă neagră (altfel, câmpul este liber), deoarece culoarea negru are valori mici pe cele 3 elemente RGB, iar albul are valori mari.

Matricele în care sunt încărcate imaginile în OpenCV au componente de tip RGB. Metoda de identificare a stării câmpurilor este făcută prin încadrarea medierea unui pătrat 5x5 în in centrul câmpului şi apoi sumarea celor 3 canale RGB ale mediei. Suma celor 3 canale poate fi încadrată în anumite intervale specifice celor 3 categorii de stări pe care le avem (câmp liber, piesă neagră, piesă albă). Sensibilitatea la schimbările de lumină este adusă în mare parte de această metodă. Valorile găsite experimental ce oferă rezultate în cazuri optime de luminozitate:

* dacă suma canalelor RGB ale mediei este sub 150, atunci acel câmp conţine o piesă neagră.
* dacă suma canalelor RGB ale mediei este între 150 şi 430, atunci acel câmp este liber.
* dacă suma canalelor RGB ale mediei este mai mare ca 430, atunci acel câmp conţine o piesă albă.

Cea mai dificilă distincţie ce trebuie făcută este cea a câmpurilor libere. Maroul de pe câmpuri se poate încadra atât în intervalul pieselor negre cât şi în intervalul pieselor albe dacă avem probleme cu luminozitatea.

O soluţie alternativă a fost folosirea detectorului Hough pentru forme circulare. Se rula detectorul Hough pe fiecare câmp, iar câmpurile ce conţineau piese ofereau detecţii, pe când câmpurile libere nu. Totuşi, existau cazuri în care era raportată o detecţie pe un câmp liber sau nu erau raportate detecţii pe un câmp ocupat. Tot era nevoie de o anumită valoare de prag pentru a decide dacă există sau nu o piesă pe acel câmp (de exemplu, daca există mai mult de 2 detecţii atunci o piesă există, altfel nu). În plus, era nevoie o altă valoare de prag ce distingea între piesele albe şi cele negre odată ce existenţa piesei era rezolvată.

Reprezentarea 2D este trivială deoarece există doar 4 tipuri de câmpuri în jocul de dame (alb, negru, piesă albă, piesă neagră) deci putem produce 4 imagini standard de aceeaşi dimensiune şi pe măsură ce aflăm starea tablei concatenăm o nouă imagine. În aplicaţie, fiecare dintre cele 4 poze este de 50x50 deci putem declara o matrice de 400x400 care să conţină toata tablă şi să concatenăm o imagine la poziţia corectă până reuşim să umplem matricea. Am mai lăsat loc 20 de pixeli (deci matricea devine 420x400) pentru a putea afişa şi scorul curent pe tablă.

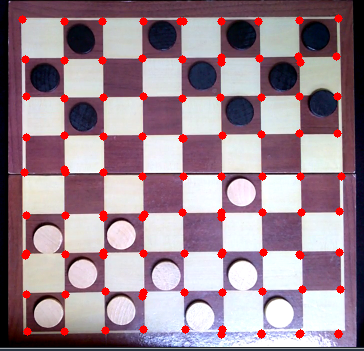
****

Figura 3.6.2 Exemplu identificare câmpuri. *Programul identifică corect fiecare câmp, dată fiind starea uniformă a luminozităţii pe tablă.*

3.7 Validarea stării calculate faţă de starea precedentă

Într-un final, fiecare stare identificată trece printr-un proces de validare. Programul începe de la starea clasică din figura 1.1.1. Această stare este încărcată înainte să se facă orice altă analiză, urmând să se calculeze toate mutările posibile pentru negru (deoarece negrul mută primul). Aceste stări calculate sunt stocate în memoria programului pe timpul rulării. În momentul în care s-a ajuns la o stare pe baza unui cadru analizat, se verifică dacă există o stare stocată identică cu cea la care s-a ajuns în urma analizei. Numai dacă această condiţie este îndeplinită, starea nouă va fi afişată (şi mutarea făcută va fi semnalată prin încercuirea cu roşu a câmpului de pe care a plecat piesa şi a câmpului pe care a ajuns). De asemenea, analiza nu permite ca un anumit jucător să mute de mai multe ori, validitatea unei stări se referă şi la jucătorul ce urmează să mute.

Analiza stărilor ne ajută şi în scopul optimizării programul. Cât timp nu a fost citită o stare greşită, adică o stare ce nu există în mutările posibile din starea anterioară, putem păstra colţurile calculate deci sărim peste detectorul Harris, algoritmul de selecţie şi algoritmul de completare, trecând direct la identificarea câmpurilor. Dacă identificarea câmpurilor eşuează să dea o stare corectă, înseamnă că ceva a intervenit în cadru (camera sau tabla a fost mutată mai mult sau mai puţin) şi este nevoie de o recalculare.

La ultimul cadru din filmare se returnează notaţia algebrică a întregului joc analizat, conform regulilor prezentate în capitolul 2.

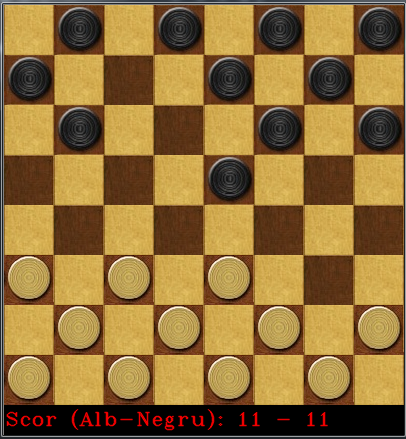
****

Figura 3.7.1: Două stări succesive în program. *Mutarea 23-19 a jucătorului ce controlează piesele albe a fost semnalată.*

Concluzii

Tabla de şah şi-a confirmat statutul de obiect foarte potrivit analizei pe calculator datorită proprietăţilor geometrice clare. Încă de la aplicarea detectorului Harris se observă rezultate satisfăcătoare. Deşi colţurile returnate de detectorul Harris nu sunt ordonate în vreun fel, există cazuri rare în care să fie ratate mai mult de câteva colţuri din cele 81 necesare. Implementarea algoritmului de tip RANSAC a fost de asemenea elegantă şi nu a prezentat probleme majore. Avantajul acestei metode faţă altă încercare bazată pe detectorul Hough este lipsa de valori de prag sau alte elemente statice necesare pentru funţionarea corectă în cat mai multe cazuri. Toate valorile specifice algoritmului de tip RANSAC precum numărul de selecţii sau mărimea selecţiei sunt independente de calitatea pozei, atâta vreme cât detectorul Harris reuşeşte să găsească un număr adecvat de colţuri.

O problemă mai dificilă a fost ridicată când a fost nevoie să completăm selecţia cu punctele lipsă de la paşii anteriori. Ideea iniţială era bazată pe aproximări făcute relativ la mărimea imaginii. Dacă am fi impus ca şi condiţie de filmare ca tot cadrul să fie reprezentat de tablă şi să nu existe fundal la extremităţile tablei, atunci am fi putut face presupunerea că distanţa dintre două puncte pe aceeaşi linie este 1/8 din numărul de coloane al matricei imaginii pe când distanţa dintre două puncte de pe aceeaşi coloană este 1/8 din numărul de linii al imaginii. Această condiţie ar fi îngreunat şi mai mult pregătirea unei filmări adecvate pentru analiză (pe lângă filmarea cu piesele albe în partea de jos a tablei sau condiţiile de luminozitate). Soluţia bazată pe regresie liniară este mult mai flexibilă deoarece ecuaţiile dreptelor ne oferă o vedere generală şi cazurile în care nici un punct nu este detectat pe o linie sau pe o coloană (deci nu se poate regresie) sunt aproape imposibile.

Cea mai mare problemă a fost prezentată de analiza câmpurilor după ce tabla a fost segmentată. Cea mai simplă şi evidentă soluţie este cea bazată pe analiza RGB, soluţie ce ne lasă expuşi la probleme de luminozitate. Totuşi, odată setaţi parametrii ce trebuie să distingă cele 3 tipuri de câmpuri, soluţia oferă un grad de încredere superior soluţiei bazată pe detecţia de cercuri în fiecare câmp cu algoritmul Hough.

În cele din urmă, condiţiile de mediu sunt o problemă comună în multe aplicaţii din domeniul vederii artificiale, iar procesul de identificare a tablei prin cele 81 de puncte este de încredere. O îmbunătăţire a aplicaţiei ar fi lucrul din mai multe unghiuri, deoarece în acest moment aplicaţia presupune că filmarea este făcută de deasupra tablei, ipoteza RANSAC cât şi regresia liniară fiind dependente de această premisă. De asemenea, există multe lucruri ce ar putea fi aduse analizei finale. S-ar putea sugera o nouă mutare pentru următorul jucător, s-ar putea da un scor mutării tinând cont de cât de potrivită a fost, s-ar putea genera variaţii asupra poziţiei curente, prezentate în ordinea corectitudinii. Toate acestea ar implica fie dezvoltarea unui program competent, fie folosirea unui program deja existent ce poate citi notaţia jocului şi poate retuna sugestii.

# Bibliografie

1. L. Jing, M. Gymrek. *The Mathematics of the Rubik’s Cube.* Massachusetts Institute of Technology. 2009. Introducere
2. Alvin C. Rencher, G. Bruce Schaalje*. Linear models in statistics. Second Edition*

LINEAR MODELS IN STATISTICS Second Edition Alvin C. Rencher and G. Bruce Schaalje

[1]https://vision.fe.uni-lj.si/cvww2016/proceedings/papers/21.pdf

[3] Learning Image Processing with OpenCV - Gloria Bueno Garcia

[4] A combined corner and edge detector - Chris Harris and Mike Stephens

http://www.bmva.org/bmvc/1988/avc-88-023.pdf

[6] Artificial Intelligence - A Modern Approach - Stuart J.Russel and Peter Norvig

[7] Computer Vision Algorithms and Applications - Richard Szelinski

[8] Martin A. Fischler & Robert C. Bolles (Martie 1981). "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography"

Surse figuri:

http://www.gadgetify.com/schachcomputer-chessgenius/

http://www.robogeek.ru/files/blogs/0004/2428/\_cache/fit650x800-turkdiy.png

[2] http://docs.opencv.org/trunk/feature\_simple.png

[3] http://dsp.stackexchange.com/questions/3336/mathematics-of-harris-corner-point-detection

[4] http://dsp.stackexchange.com/questions/3336/mathematics-of-harris-corner-point-detection

[5] http://dsp.stackexchange.com/questions/3336/mathematics-of-harris-corner-point-detection

[6] http://dsp.stackexchange.com/questions/3336/mathematics-of-harris-corner-point-detection

[7] http://dsp.stackexchange.com/questions/3336/mathematics-of-harris-corner-point-detection

[8] http://docs.opencv.org/trunk/dc/d0d/tutorial\_py\_features\_harris.html

[9] http://docs.opencv.org/3.0-beta/\_images/shitomasi\_block1.jpg

[10] https://robotics.eecs.berkeley.edu/~sastry/ee20/cademo.html

Harris

=================================================

http://dsp.stackexchange.com/questions/3336/mathematics-of-harris-corner-point-detection

http://docs.opencv.org/trunk/df/d54/tutorial\_py\_features\_meaning.html

http://docs.opencv.org/trunk/dc/d0d/tutorial\_py\_features\_harris.html

http://docs.opencv.org/trunk/dd/d1a/group\_\_imgproc\_\_feature.html#gac1fc3598018010880e370e2f709b4345

http://docs.opencv.org/3.0-beta/doc/py\_tutorials/py\_feature2d/py\_shi\_tomasi/py\_shi\_tomasi.html

Good Features To Track - Jianbo Shi Carlo Tomasi

==============================================

Hough

====================================================

Use of the Hough Transform to detect lines and curves in pictures Richard O Duda. Peter E hart

http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough\_lines/hough\_lines.html

Canny

Canny, J., *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny\_detector/canny\_detector.html

===================================================================

http://docs.opencv.org/trunk/feature\_simple.png LINK imagine 1

//Eventuale adaugiri

//Expansiunea Taylor

// TODO Expansiunea Taylor de pe wikipedia

// incercare de recunoastere naiva doar pe baza de medie -> medie + deviatie -> Canny + medie + deviatie

// am facut identificarea punctelor din colturi adaptativa

//tabla dame

https://upload.wikimedia.org/wikipedia/commons/thumb/e/e0/Damiera.JPG/320px-Damiera.JPG