



**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**  
**DEPARTAMENTUL CALCULATOARE**

**LIPIREA IMAGINILOR FOLOSIND DIFERITE TIPURI DE  
PUNCTE CHEIE**

**LUCRARE DE LICENȚĂ**

Absolvent: **Adrian BÎZ**

Coordonator **Şl. Dr. Ing. Robert VARGA**  
științific:

**2024**

# Cuprins

<b>Capitolul 1. Introducere.....</b>	<b>3</b>
1.1. Contextul proiectului .....	3
1.2. Motivația.....	4
1.3. Structura lucrării pe capitole.....	4
<b>Capitolul 2. Obiectivele proiectului .....</b>	<b>6</b>
2.1. Obiectivul principal .....	6
2.2. Obiectivele secundare .....	6
2.3. Cerințe.....	7
<b>Capitolul 3. Studiu bibliografic .....</b>	<b>9</b>
3.1. Algoritmi folosiți .....	10
3.1.1. SIFT (Scale-Invariant Feature Transform) .....	10
3.1.2. ORB (Oriented FAST and Rotated BRIEF) .....	10
3.1.3. FREAK (Fast Retina Keypoint) .....	11
3.1.4. BRIEF (Binary Robust Independent Elementary Features) .....	11
3.1.5. FAST (Features from Accelerated Segment Test) .....	11
3.1.6. Detectorul de Colțuri Harris .....	11
3.1.7. Shi-Tomasi (Good Features to Track) .....	12
3.1.8. BFM (Brute-Force Matching) .....	12
3.1.9. FLANN (Fast Library for Approximate Nearest Neighbors) .....	12
3.1.10. RANSAC (Random Sample Consensus).....	13
3.2. Comparație între algoritmi.....	13
<b>Capitolul 4. Analiză și fundamentare Teoretică.....</b>	<b>15</b>
4.1. Modelul cazurilor de utilizare .....	15
4.2. Cazurile de utilizare .....	15
4.3. Filtrul Log-Gabor.....	17
4.4. Resursele necesare .....	19
<b>Capitolul 5. Proiectare de detaliu și implementare .....</b>	<b>20</b>
5.1. Schema generală a aplicației.....	20
5.2. Main-ul și componenta globală .....	21
5.3. Pachetul controlorului.....	23

5.4.	Pachetul evaluatorului .....	24
5.5.	Pachetul cusătoriilor .....	25
5.6.	Pachetul descriptorilor .....	26
5.7.	Pachetul împerechectorului de trăsături .....	27
5.8.	Pachetul detectorilor .....	28
<b>Capitolul 6. Testare și validare .....</b>		<b>32</b>
6.1.	Modul de testare .....	32
6.2.	Parametrii algoritmilor.....	32
6.2.1.	Calibrarea detectorului de colțuri Harris .....	32
6.2.2.	Calibrarea detectorului de colțuri Shi-Tomasi.....	33
6.2.3.	Calibrarea detectorului Log-Gabor.....	33
6.3.	Testarea detectorilor .....	43
6.4.	Testarea descriptorilor .....	51
6.5.	Testarea împerechectorilor .....	52
<b>Capitolul 7. Manual de instalare si utilizare .....</b>		<b>59</b>
<b>Capitolul 8. Concluzii.....</b>		<b>63</b>
8.1.	Contribuție personală.....	63
8.2.	Realizarea obiectivelor .....	63
8.3.	Dezvoltări ulterioare .....	64
<b>Anexa 1 – Aplicarea filtrelor Log-Gabor cod .....</b>		<b>66</b>
<b>Anexa 2 – Lista figurilor .....</b>		<b>67</b>
<b>Anexa 3 – Lista tabelelor .....</b>		<b>69</b>
<b>Anexa 4 – Glosar.....</b>		<b>70</b>

# **Capitolul 1. Introducere**

## **1.1. Contextul proiectului**

Pentru înțelegerea acestei lucrări trebuie definit mai întâi ce reprezintă o imagine în domeniul informatic și a procesării imagistice. O imagine reprezintă o matrice, implicit de două dimensiuni(lungime și lățime), în care la fiecare poziție se află un pixel. Un pixel poate avea diferite formatare pe baza numărului de canale de lumină și a altor caracteristici precum opacitatea: binară(0 și 1), Grayscale(alb-negru, cu valori între 0-255), RGB(trei culori: roșu, galben, albastru), RGBA(RGB cu un canal pentru opacitatea pixelului).

Procesarea digitală de imagini se consideră a fi un subdomeniu a procesării de semnale digitale, aceasta ocupându-se în general de operații prin care se extrag informații din imaginile 2D, se îmbunătățește calitatea acestora sau se automatizează anumite procese. Aceasta se folosește de mediul digital, modele matematice și algoritmi pentru procesare și analizare imaginilor.

Un proces ce cade în acest domeniu este lipirea de imagini sau cusătura imaginilor. Prin aceasta ce urmărește combinarea mai multor imagini cu câmpuri vizuale suprapuse pentru crearea unei viziuni panoramice segmentate sau a unei imagini de rezoluție înaltă. Pentru realizarea acesteia cu succes sunt necesare suprapunerile aproape exacte între imagini și expuneri identice.

În domeniul procesării de imagini și viziunii computerizate, detectarea de elemente cheie și extragerea acestora reprezintă una dintre operațiile de bază acestea jucând un rol critic pentru procese precum recunoașterea de obiecte, analizarea scenelor sau lipirea de imagini. În acest context, elementele cheie sunt denumite caracteristici, acestea pot fi reprezentate de puncte, linii, pete sau alte forme și modele ce pot să definească o anumită imagine și ce reprezintă aceasta. Calitatea acestor caracteristici influențează direct modul, viteza și posibilitatea de a folosi imaginea în diferite aplicații precum recunoașterea facială, reconstrucția 3D, crearea de imagini panoramice sau de definiție înaltă, etc. De-a lungul timpului diferiți algoritmi au fost dezvoltăți în scopul de a obține caracteristici cât mai calitative și cât mai eficient posibil. Printre cele mai cunoscute se numără: [1]detectorul de colțuri Harris și [2]Shi-Tomasi, [3]Scale-Invariant Feature Transform (SIFT), [4]Oriented FAST and Rotated BRIEF (ORB), [5]Features from Accelerated Segment Test (FAST). Aceste detectoare au devenit unele standard și se folosesc constant în diferite aplicații datorită robustei, acurateței și eficienței acestora.

În ciuda inovațiilor aduse de acești algoritmi încă rămâne loc, dacă nu chiar o necesitate pentru îmbunătățiri, în special pentru compensarea în diferențele de mărime, orientare și luminozitate. Fiecare dintre aceste posibile variații poate afecta semnificativ calitatea caracteristicilor detectate și prin urmare rezultatul final al oricărui proces ulterior. Folosind inspirație din natură, se pot găsi diferite noi modalități de creștere a ratei de succes pentru caracteristici de calitate înaltă. O astfel de modalitate implică folosirea de filtre Log-Gabor care sunt dezvoltate pornind de la exemplul ochiului uman. Acestea se arată ca o posibilă alternativă la algoritmii cunoscuți oferind o mai capturare a caracteristicilor ce reprezintă detaliu fine în imagini.

## **1.2. Motivația**

Motivația din spatele acestui proiect vine de la limitările prezente în metodele deja cunoscute de detectare a caracteristicilor și din potențialele beneficii ce le poate aduce filtrele Log-Gabor în procesarea de imaginii. Algoritmii standard precum [3]SIFT, [4]ORB și [5]FAST deși eficienți pot ajunge să aibă dificultăți în anumite condiții precum variațiuni de lumină, zgomot de imagine și diferențe de mărime. În schimb Log-Gabor se prezintă ca fiind soluția în asemenea cazuri fără a sacrifica eficiență.

Acest proiect își propune să exploreze o implementare a algoritmului Log-Gabor pentru detectarea caracteristicilor și să compare din punct de vedere calitativ performanța acestuia în comparație cu celelalte modalități tradiționale de detecție. Prin aceasta se dorește determinarea îmbunătățirii aduse de filtrele Log-Gabor în acuratețe și robustețe, în particular în cazul aplicațiilor pentru lipirea de imagini. Pentru lipirea imaginilor se dorește o aliniere cât mai precisă între caracteristicile comune, care se suprapun, deoarece chiar și erori minore la prima vedere pot crea artefacte vizibile în lipirea rezultată, iar o îmbunătățirea acurateței poate duce la rezultate fără urme vizibile de distorsiuni sau cusături. Pe lângă aceasta demonstrarea eficienței filtrelor Log-Gabor ar putea încuraja folosirea lor și în alte aplicații astfel fiind o posibilă inovație în domeniul procesării de imagini.

## **1.3. Structura lucrării pe capitole**

Capitolul 1 – Introducere, primul capitol prezintă o conturare a problemei abordate de tema proiectului de licență și a domeniului acesteia.

Capitolul 2 – Obiectivele proiectului detaliază obiectivele principale și secundare a proiectului.

Capitolul 3 - Studiu bibliografic conține o analiză detaliată a literaturii consultată în domeniul detecției trăsăturilor și al filtrării Log-Gabor. Sunt prezentate algoritmii folosiți pentru comparație și metodele cunoscute de procesare a caracteristicilor și lipiri imaginilor cu acestea în vederea obținerii imaginilor panoramice.

Capitolul 4 - Analiză și Fundamentare Teoretică, se prezintă cerințe funcționale și non funcționale ale sistemului, resursele necesare rulării proiectului, se evidențiază tehnologiile și mediul de dezvoltare folosit și sunt prezentate formulele matematice necesare creării filtrelor Log-Gabor.

Capitolul 5 - Proiectare de Detaliu și Implementare, capitolul include o descriere a componentelor sistemului, evidențind arhitectura aplicației și procesul de implementare. Sunt prezentate detalii despre modul de implementare a filtrării Log-Gabor, algoritmii de extragere a punctelor cheie, precum și comunicarea între modulele software utilizate. Acesta include, de asemenea, diagrame arhitecturale și secțiuni relevante de cod pentru a ilustra aspectele critice ale implementării.

Capitolul 6 - Testare și Validare în acest capitol sunt prezentate cazurile de testare, rezultatele experimentale obținute și o analiză comparativă a performanței detectorului de puncte cheie Log-Gabor în raport cu algoritmii standard.

Capitolul 7 - Manual de Instalare și Utilizare, acest capitol oferă instrucțiuni detaliate privind instalarea și rularea aplicației dezvoltate.

Capitolul 8 – Concluzii acest capitol summarizează principalele rezultate și descoperirile ale proiectului, evaluând în ce măsură au fost îndeplinite obiectivele stabilite. Sunt analizate contribuțiile aduse de utilizarea filtrelor Log-Gabor în contextul

detectiei trăsăturilor și lipirii imaginilor. De asemenea, sunt propuse direcții de dezvoltare viitoare, inclusiv optimizări.

Glosar, Liste și Anexe, în final sunt incluse un glosar cu termenii specifici din domeniul detectiei punctelor cheie în imagini, precum și liste cu figurile și tabelele folosite în lucrare.

## **Capitolul 2. Obiectivele proiectului**

### **2.1. Obiectivul principal**

Obiectivul principal al proiectului este dezvoltarea și implementarea unui set de filtre de caracteristici pentru imagini, bazate pe algoritmul Log-Gabor. Scopul utilizării acestor filtre este de a obține o detecție mai precisă a trăsăturilor fine ale imaginilor. Aceasta este esențială în procese complexe de prelucrare a imaginii, precum lipirea imaginilor pentru crearea de panorame. Alte metode de detecție a trăsăturilor pot suferi de limitări în ceea ce privește sensibilitatea la zgomot și la variații de scală și orientare. Filtrele Log-Gabor oferă un avantaj distinct prin capacitatea lor de a capta informații de înaltă frecvență din imagini fără a introduce distorsiuni sau erori semnificative.

Un set de filtre va consta în o combinație de filtre Log-Gabor individuale, care diferă între ele prin variațiuni ale parametrilor funcției matematice utilizate. Acești parametrii determină modul în care filtrul răspunde la anumite caracteristici ale imaginii, iar ajustarea lor poate influența semnificativ rezultatul fiecărei iterații de procesare. În principiu un set de filtre adecvat unei astfel de probleme consistă în diferite combinări de mărime și orientare. Totuși o astfel de abordare poate face ca și cea mai simplă procesare să crească exponențial în timp și resurse necesare, ceea ce impune o limitare practică a numărului de combinații de filtre.

Este esențial de menționat că un filtru Log-Gabor funcțional nu este doar o reprezentare imagistică a unei singure funcții matematice. Acesta poate fi o combinație de alte filtre cu scop de calibrare și îmbunătățire. Astfel există o multitudine de filtre ce se pot crea, dar din considerente de resurse și timp, crearea unui singur set este obiectivul principal.

### **2.2. Obiectivele secundare**

Aplicația dezvoltată în cadrul acestui proiect are mai multe obiective secundare, toate contribuind la demonstrările posibilelor utilizări ale filtrelor Log-Gabor în procesul de creare a imaginilor panoramice. În acest context, aplicația va funcționa ca un prototip și ca o platformă de testare, permitând utilizatorilor să experimenteze și să evaluateze eficiența și performanța filtrelor Log-Gabor în comparație cu alte tehnici existente. Pentru a atinge acest scop, aplicația va îndeplini următoarele funcționalități:

- Selectarea de către utilizator a unui set de două imagini ce vor fi procesate.
- Procesarea imaginilor folosindfiltrele Log-Gabor.
- Salvarea de rezultate parțiale și finale aplicării filtrelor.
- Marcarea punctelor cheie ce reprezintă caracteristici ale imaginii.
- Identificarea caracteristicilor comune dintre cele două imagini.
- Salvarea panoramei rezultate după încercarea de a lipi cele două imagini.

Selectarea Imaginilor: Utilizatorii vor putea selecta un set de două imagini pentru procesare. Aceste imagini vor fi utilizate ca input pentru aplicație și vor fi analizate folosind filtre Log-Gabor pentru a extrage caracteristicile cheie.

Procesarea Imaginilor: Aplicația va aplica filtrele Log-Gabor pe imaginile selectate pentru a scoate în evidență posibilele puncte cheie. Acest proces va implica filtrarea frecvențelor relevante și eliminarea zgomotului, maximizând claritatea trăsăturilor importante.

Salvarea Rezultatelor Partiale: În timpul procesului de filtrare și detecție a caracteristicilor, rezultatele intermediere vor fi salvate pentru o analiză ulterioară. Acestea constau în imaginile peste care s-a aplicat filtrul Log-Gabor, filtrul folosit și date referitoare la acesta. Datele și imaginile parțiale vor permite utilizatorilor să evaluateze eficiența fiecărui pas în parte și să facă comparații directe cu alte metode de detecție.

Marcaj Puncte Cheie: După procesarea imaginilor aplicația va salva o versiune a imaginii pe care vor fi marcate punctele detectate ca fiind caracteristici esențiale. Aceasta va permite vizualizarea directă a punctelor de interes identificate de filtrele Log-Gabor.

Identificarea Caracteristicilor Comune: Aplicația va genera un set de puncte comune și o nouă imagine, ce va fi salvată împreună cu celelalte date parțiale. Aceasta va fi formată din cele două imagini sursă, iar caracteristicile comune vor fi vizibile și conectate printr-o linie colorată. Astfel se verifică acuratețea detecției de trăsături și se poate anticipa succesul procesului de lipire a imaginilor.

Generarea și Salvarea Panoramelor: În cele din urmă, aplicația va încerca să combine cele două imagini procesate într-o panoramă unificată, folosind punctele cheie identificate pentru alinierea corectă. Panoramele rezultate vor fi salvate pentru evaluare, oferind o bază solidă pentru a compara performanța metodei Log-Gabor cu alte tehnici standardizate în acest domeniu.

### **2.3. Cerințe**

Pentru ca acest proiect să fie considerat că și-a atins obiectivele următoarele cerințe funcționale trebuie respectate:

- Încărcarea de imagini: Sistemul va permite utilizatorului să aleagă două imagini ce vor fi procesate. Acestea vor fi citite și convertite în format alb-negru.
- Detectare de trăsături: Sistemul va suporta folosirea mai multor algoritmi de detecție. Pornind de la imaginile alb-negru va returna punctele cheie detectate de aceștia.
- Descriere de trăsături: Sistemul va permite folosirea mai multor algoritmi de descriere în diferite combinații. Aceștia vor primi punctele cheie detectate și vor returna descriptori trăsăturilor reprezentate de aceștia.
- Împerecherea de trăsături: Sistemul va suporta multipli algoritmi de împerechere a trăsăturilor și pornind de la o pereche de imagini și descriptori trăsăturilor lor va oferi împerecherea acestora folosind algoritmul selectat.
- Calcularea omografiei: Sistemul trebuie să calculeze matricea de omografie folosind pentru a găsi cea mai bună transformare între cele două imagini, raportul inlier și eroarea de reproiectare.
- Cusătura de imagini: Sistemul va face cusătura imaginilor folosind omografia calculată.
- Măsurarea performanței: Sistemul trebuie să măsoare și să înregistreze timpul de procesare pentru fiecare combinație de detectare a caracteristicilor, descriere și algoritmi de potrivire.
- Salvarea rezultatelor: Sistemul va desena conexiunile dintre imagini și le va salva în fișierul imaginilor originale alături de imaginea cusută și metricile rezultate(raportul inlier, eroarea de reproiectare, numărul de potriviri bune, timpul de procesare).

Din punct de vedere a cerințelor non-funcționale proiectul va trebui să aibă în vedere:

- Performanță: Sistemul ar trebui să gestioneze eficient procesarea imaginilor de înaltă rezoluție și să minimizeze timpul de procesare unde se poate.
- Scalabilitate: Sistemul ar trebui să fie proiectat pentru a adăuga cu ușurință suport pentru detectarea, descrierea și algoritmii de potrivire a caracteristicilor noii.
- Utilizabilitate: Sistemul trebuie să ofere rezultate clare și informative, inclusiv valori și vizualizări ale rezultatelor, iar când e cazul să gestioneze erorile cu grație.
- Mantenabilitate: Codul trebuie să fie modular, sugestiv și bine documentat pentru a facilita întreținerea și îmbunătățirile viitoare.
- Fiabilitate: Sistemul trebuie să ofere rezultate precise și consecvențe pentru detectarea caracteristicilor, descriere, potrivire și îmbinare a imaginii.
- Portabilitate: Sistemul trebuie să fie portabil pe diferite sisteme de operare, cu condiția ca bibliotecile necesare (de exemplu, OpenCV) să fie disponibile.
- Extensibilitate: Sistemul ar trebui să fie proiectat pentru a permite integrarea ușoară a funcționalităților suplimentare de procesare a imaginii, cum ar fi diferite tipuri de transformări sau îmbunătățiri ale imaginii.

## Capitolul 3. Studiu bibliografic

Detectarea punctelor cheie reprezintă una dintre aspectele fundamentale a multor dintre problemele procesării de imagini. Acestea includ creări de panorame, reconstrucții 3D, recunoașterea de obiecte, urmărirea mișcării și corespondența stereo. Pentru realizarea tuturor acestora este nevoie de stabilirea caracteristicilor unei imagini. Acestea reprezintă proprietăți independente de mărime și rotație, dar doar parțial variabile față de luminozitate și perspectivă. Acestea sunt locate în poziții stabile în domeniu spațial și frecvențial fiind foarte improbabil a fi perturbate de o îngărmădire sau zgromot a altor pixeli, de asemenea sunt și distinctive unele față de altele. Astfel o singură astfel de trăsătură de imagine poate fi împerecheată cu o rată foarte mare de succes chiar și într-un set made de date. Dar pentru a putea să le obținem trebuie mai întâi să găsim puncte cheie.

Un punct cheie este un punct specific dintr-o imagine care are o semnificație particulară datorită caracteristicilor sale unice. Aceste puncte sunt adesea alese pentru că sunt distincte și ușor de recunoscut chiar și atunci când imaginea este supusă unor transformări precum rotația, scalarea sau variațiile de iluminare. Aceștia sunt extrași din imagini cu ajutorul algoritmilor de detecție:

- [3]SIFT (Scale-Invariant Feature Transform): Identifică puncte de interes care sunt independente de scară și rotație.
- [5]FAST (Features from Accelerated Segment Test): Un algoritm simplu și rapid utilizat frecvent pentru detectarea puncte cheie în timp real.
- [1]Detectorul de colțuri HARRIS: Succesorul primului algoritm ce detectă puncte cheie.
- [2]Detectorul de colțuri Shi-Tomasi numit și “Good Features to Track”: O îmbunătățire a algoritmului lui Harris.
- [4]ORB (Oriented FAST and Rotated BRIEF): O dezvoltare a algoritmului FAST, aceasta fiind metoda preferată în cazul aplicațiilor în timp real.

Aceștia fiind printre principalii algoritmi standard când vine vorba de o astfel de problemă.

Pentru procesul de găsire și descriere a caracteristicilor unei imagini se urmărește următorul set de operații:

- Preprocesarea Imagini: se încearcă eliminarea zgromotului de fundal și îmbunătățirea imaginii pentru claritate.
- Identificarea punctelor de interes: Folosind algoritmi enunțați se caută diferite puncte sau zone ce au caracteristici distincte și care sunt ușor de recunoscu. Acestea pot fi colțuri, muchii sau regiuni de textură.
- Calculul Descriptorilor de Caracteristici: Odată ce punctele de interes au fost identificate, se calculează descriptorii de caracteristici. Acești descriptori sunt vectori care codifică informații despre regiunea din jurul fiecărui punct de interes. Scopul este de a descrie caracteristicile într-un mod care este robust la schimbări de scară, rotație, sau variații de iluminare.

De multe ori algoritmi standardizați au descriptori dedicați ce oferă cele mai bune rezultate în general, dar se poate ca un detector să nu aibă sau să folosească un algoritm de descriere independent sau de la alt algoritm. Acești descriptori captează informații unice despre zonele din jurul punctelor cheie și le codifică într-un mod care permite

potrivirea robustă și eficientă. În cadrul acestui proiect interesul va fi pe următorii descriptori: [3]SIFT, [4]ORB, [6]FREAK și [7]BRIEF.

În cazul problemei de lipire panoramică se vor căuta caracteristicile în toate imaginile ce se doresc a fi unite, după care se vor căuta corespondențe între acestea. Pentru acest pas, în cadrul proiectului s-au considerat două metode: [8]Brute-Force Matching și [9]FLANN(Fast Library for Approximate Nearest Neighbors).

O dată ce avem un set de trăsături ce sunt regăsite în ambele imagini, ce se doresc a fi cusute, va trebui ca acestea să fie organizate într-un model transformațional. Acesta descrie modul în care o imagine va trebui transformată față de celalătă ca ea să se potrivească și să formeze o continuare a celelalte imagini. În principal acest model se realizează cu o estimare de tip [10]RANSAC, ce încearcă să ofere o transformare în care trăsăturile unei imagini să fie poziționate peste cele corespunzătoare din celalătă imagine. Această transformare se numește omografie și cuprinde multiple operațiuni: scalare, rotire, translatare și aşa mai departe.

Pasul următor constă în deformarea imaginilor conform omografiei astfel încât acestea să se potrivească într-un domeniu de coordonate. Îmbinarea acestora formând o imagine ce cuprinde mulțimea totală a pixelilor din ambele imagini unite. Pentru îmbinare există diferite metode de abordare, dar pentru simplitatea proiectului s-a ales o abordare directă ce nu încearcă să compenseze pentru diferențe de lumină și expunere.

Tot acest procedeu se poate găsi explicat mai în detaliu în cartea [11].

### **3.1. Algoritmi folosiți**

#### **3.1.1. SIFT (Scale-Invariant Feature Transform)**

[3]SIFT este dezvoltat de David Lowe și este unul dintre cei mai utilizati detectori și descriptori pentru extragerea caracteristicilor imaginii. Acesta detectează puncte de interes într-un mod invariant la scală și rotație, utilizând un proces de transformare de imagine pentru a identifica locurile unde contrastul local este ridicat.

SIFT parcurge mai întâi o versiune netezită a imaginii căutând maxime și minime locale. Preia punctele cheie eliminând cele ce reprezintă muchii și cele cu contrast slab, după care le atribuie o orientare folosindu-se de gradientul local a imaginii.

Avantajele algoritmului SIFT constau în o toleranță mai bună la schimbări de scară, rotație, iluminare și chiar la anumite perspective. Acesta având cea mai bună acuratețe dintre algoritmi standard. Unul dintre principalele dezavantaje ale SIFT este complexitatea sa computațională ridicată, ceea ce îl face mai lent comparativ cu alți descriptori. De asemenea, este mai puțin eficient în aplicații în timp real din cauza resurselor necesare.

Acesta este utilizat frecvent în robotică, viziune computerizată pentru detecția și urmărirea obiectelor, realizarea de mozaicuri de imagini, reconstrucție 3D.

#### **3.1.2. ORB (Oriented FAST and Rotated BRIEF)**

[4]ORB combină algoritmul FAST pentru detectarea punctelor de interes cu BRIEF pentru descrierea caracteristicilor, optimizându-le pentru a rezolva problemele de rotație și scală. ORB este conceput pentru a fi rapid și eficient, potrivit pentru aplicațiile în timp real.

ORB este cel mai rapid algoritm standard eficiență și viteza lui fiind sesizabile chiar și fără măsurări cronometrice, de asemenea, este invariant la rotație și parțial invariant la schimbări de scară. Deși este mai rapid, ORB poate fi mai puțin precis decât SIFT în cazurile de iluminare foarte variabilă sau distorsiuni de perspectivă complexe.

Acesta este ideal pentru aplicații încorporate, dar este folosit și în aplicații de realitate augmentată, drone, sisteme de navigație autonomă și alte scenarii unde performanța în timp real este critică.

### 3.1.3. FREAK (Fast Retina Keypoint)

[6] Inspirat de structura retinei umane, FREAK este un descriptor binar care extrage caracteristici prin analizarea unui set de puncte dispuse într-un mod care imită câmpul vizual periferic al ochiului uman. Dezvoltat de Alexandre Alahi în 2012.

Este optimizat pentru a fi eficient din punct de vedere al memoriei, dar și a timpului. Astfel rapiditatea și cerințele scăzute de memorie îl fac potrivit pentru aplicații mobile. FREAK este, de asemenea, robust la zgromot și schimbări moderate de iluminare. Totuși acestea nu vin fără dezavantaje, FREAK tinde să aibă performanțe mai slabe în situații de perspectivă extremă sau în prezența unor distorsiuni geometrice complexe.

Aplicațiile lui sunt regăsite în detectarea și potrivirea caracteristicilor în sisteme de recunoaștere facială, urmărire obiectelor și robotică.

### 3.1.4. BRIEF (Binary Robust Independent Elementary Features)

[7] BRIEF este un descriptor binar care folosește un set de comparații de intensitate aleatorii între perechi de pixeli pentru a crea un vector descriptor pentru fiecare punct cheie. Introdus de Michael Calonder în 2010, nu este foarte acurat, dar compensează prin viteză.

Acesta are viteză foarte bună de procesare. Implementarea sa este directă și necesită resurse minime, dar pe partea minusurilor, performanțele sale scad în prezența schimbărilor semnificative de scară, rotație sau iluminare. Acesta finind, de asemenea, mai vulnerabil la zgromot în comparație cu alți descriptori.

În ciuda dezavantajelor lui, implementările acestuia sunt regăsite în procesarea de imagini în timp real, aplicații de urmărire și detecție rapidă a obiectelor în medii controlate.

### 3.1.5. FAST (Features from Accelerated Segment Test)

[5] FAST este un detector de trăsături propus de Edward Rosten și Tom Drummond în 2006. Aceasta este menit să fie ușor de procesat și adecvat aplicațiilor în timp-real. Aceasta evaluează fiecare pixel folosindu-se de vecinii săi, dacă vecinii diferă de acesta în mod similar și în variațiune considerabilă, pixelul este declarat ca colț și posibilă trăsătură a imaginii.

FAST, precum și abrevierea este rapid în procesare, dar sacrifică stabilitatea față de scală și rotație.

Acesta își găsește utilitatea în alți algoritmi mai optimizați sau în învățare computerizată.

### 3.1.6. Detectorul de Colțuri Harris

[1] Acesta este printre primii algoritmi de detecție a colțurilor, dezvoltat de Harris și Mike Stephens în 1988, rămâne și în ziua de azi foarte relevant pentru robustețea lui. Detectorul funcționează pe baza diferenței de intensitate a pixelilor și recunoaște ca colț o locație unde aceasta diferă în ambele direcții și utilizează valorile proprii ale matricei de al doilea moment a gradientilor de imagine pentru a determina rezistența colțului.

Detectorul Harris rămâne relevant deoarece acesta nu este sensibil la schimbări de rotație și oferă rezultate accurate și precise. Acesta reușește toate aceste calități fără a se baza pe calcule matematice complexe și procese complicate, așa că mai are avantajul de a rămâne ușor de implementat. Totuși acesta necesită calibrări de limite și rămâne sensibil la schimbări de scală și transformări affine. Iar calculele necesare pot ajunge să fie dificil de calculat în cazurile imaginilor mari sau detaliate.

Acesta fiind printre primi algoritmi utilitatea lui rămâne marcată în toți ceilalți algoritmi dezvoltăți ulterior.

### 3.1.7. Shi-Tomasi (Good Features to Track)

[2]Detectorul de colț Shi-Tomasi, cunoscut și ca algoritmul „Good Features to Track”, a fost introdus de Jianbo Shi și Carlo Tomasi în 1994. Este o îmbunătățire a Detectorului de colț Harris și este utilizat pe scară largă pentru detectarea colțurilor și punctelor cheie în imagini, în special pentru funcțiile de urmărire în timp în secvențele video.

Metoda lui Shi-Tomasi se bazează pe detectorul de colț Harris, dar face o modificare crucială în modul în care sunt selectate colțurile. În loc să folosească funcția de răspuns Harris pentru a identifica colțurile, Shi-Tomasi propune utilizarea valorii proprii minime a matricei de covarianță a gradientilor de imagine (cunoscută și ca tensor de structură). Ideea principală este că un punct este considerat un colț bun dacă cea mai mică valoare proprie este mai mare decât un anumit prag.

Acest algoritm reușește să păstreze avantajele detectorului lui Harris și chiar să depășească acesta în aspecte de acuratețe, performanță și eficiență, dar acestea vin cu mai multe restricții față de transformările suportate și limitare în căutare, fiind adecvat numai pentru colțuri.

Similar cu algoritmul precedent, utilitatea acestui detector rămâne ca fiind opțiunea standard și sigură în multiple aplicații și îmbunătățiri.

### 3.1.8. BFM (Brute-Force Matching)

[8]Împerecherea BFM, reprezintă opțiunea cea mai simplă și directă de a crea conexiuni între imagini. Acesta calculează distanța dintre un descriptor și ceilalți descriptori din imaginea secundară, alegându-l pe cel mai apropiat.

Această abordare nu necesită calcule complexe fiind îngreunată numai de un număr mare de date de procesat, în cazul imaginilor mari.

Deși este simplistă această abordare poate oferi rezultate bune atâtă timp cât descriptorii reprezintă trăsături de calitate.

### 3.1.9. FLANN (Fast Library for Approximate Nearest Neighbors)

[9]FLANN, care este prescurtarea de la Fast Library for Approximate Nearest Neighbors, este o bibliotecă, concepută pentru a rezolva problema găsirii celor mai apropiate vecini ai punctelor cheie din spații cu dimensiuni mari, în mod eficient.

Aceasta include o colecție de algoritmi optimizați pentru găsirea celor mai apropiate vecini în seturi de date multidimensionale. Avantajul său principal este că poate efectua aceste căutări eficient chiar și atunci când are de-a face cu un număr mare de puncte, chiar și în format de multiple dimensiuni. Biblioteca realizează acest lucru utilizând mai degrabă metode aproximative decât metode exacte, sacrificând precizia pentru viteze considerabile mai mari. Astfel FLANN caută numai prin aproximare cel mai apropiat vecin, iar algoritmi de căutare sunt aleși automat din cadrul bibliotecii pe baza unor teste predefinite. FLANN are un grad mare de adaptabilitate chiar și pentru

probleme din afara domeniului procesării de imagini putând să folosească diferite metriki pentru distanțe.

Astfel FLANN este extrem de flexibil, eficient în timp, scalabil și pe lângă acestea este ușor de folosit, fiind incorporat în bibliotecile OpenCV (Open Source Computer Vision Library).

Totuși acesta are și dezavantaje, printre care calibrare manuală pentru rezultate optime și necesitatea de consum de memorie ridicată pot face nu cea mai dorită opțiune.

### 3.1.10. RANSAC (Random Sample Consensus)

[10]RANSAC, care înseamnă Random Sample Consensus, este un algoritm iterativ utilizat pentru a estima parametrii unui model matematic dintr-un set de date observate care conține valori aberante. Dezvoltat de Fischler și Bolles în 1981, RANSAC este robust și utilizat pe scară largă în viziunea computerizată și sarcinile de procesare a imaginilor, în special acolo unde datele pot fi contaminate cu zgomot și valori aberante.

RANSAC funcționează pe baza următorilor pași:

- Eșantionare aleatorie: Un mic subset de date de intrare este selectat aleatoriu. Mărimea acestui subset depinde de numărul minim de puncte necesare pentru estimarea parametrilor modelului. De exemplu, pentru a se potrivi unei linii, sunt necesare doar două puncte.
- Estimarea modelului: Un model este estimat folosind doar subsetul aleatoriu de puncte selectat. De exemplu, dacă se încearcă potrivirea unei linii la date, parametrii liniei (panta și interceptarea) sunt calculați folosind punctele selectate.
- Evaluare set de consens: Întregul set de date este testat cu modelul estimat pentru a vedea cât de bine se potrivește. Un punct este considerat un inlier dacă se află într-o anumită distanță de prag predefinită față de model. Toate inliers formează ceea ce se numește setul de consens.
- Verificare model: Se evaluează dimensiunea setului de consens (numărul de inlieri). Dacă este mai mare decât un prag specificat, modelul este considerat valid și procesul se poate termina. În caz contrar, procesul continuă la următoarea iterare.
- Repetare: Pașii anteriori se repetă pentru un număr fix de iterări sau până când se găsește un model satisfăcător. Modelul cu cel mai mare număr de inlieri este selectat ca cea mai bună estimare a datelor.
- Rafinament: După identificarea celui mai bun model, toate inlierele sunt folosite pentru a re-estima parametrii modelului, producând un model final rafinat, care se potrivește cât mai mult cu datele.

RANSAC are avantajele de a suporta un număr mare de date, poate fi calibrat pentru aplicații în timp real și este flexibil suportând diferite modele și tipuri de date. Dar pe celaltă parte RANSAC poate duce la costuri mari de procesare din cauza structurii iterative, este dependent de parametrii predefiniți și nu garantează rezultate optime sau ne eronate din cauza zgomotului în inliners.

## 3.2. Comparație între algoritmi

În următorul tabel se află o comparație rudimentară între algoritmii prezentați anterior și pentru o mai bună înțelegere următorii termeni au fost detaliați:

- Robust la zgomot și outlieri: Aceasta se referă la capacitatea algoritmului de a funcționa corect chiar și în prezența unor date eronate sau neobișnuite (outlieri).
- Complexitate computațională: Aceasta măsoară resursele de calcul necesare pentru a rula algoritmul.

- Aplicabil în timp real: Acest criteriu evaluează dacă algoritmul poate fi utilizat în aplicații care necesită procesare rapidă.
- Precizie: Aceasta reflectă cât de bine pot algoritmii să identifice și să potrivească puncte de caracteristici corect.
- Utilizări comune: Acest criteriu oferă exemple de scenarii comune în care este utilizat fiecare algoritm.

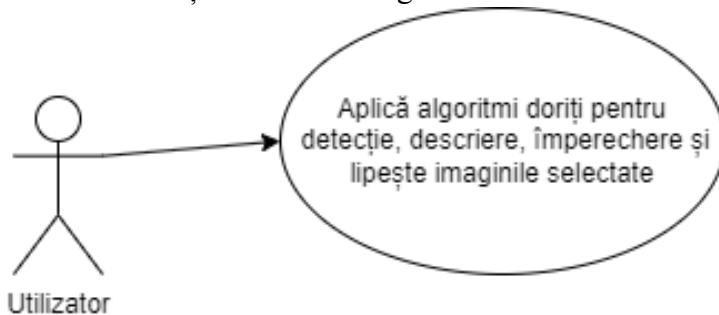
Algoritm	Robust la zgomot și outlieri	Complexitate computațională	Aplicabil în timp real	Precizie	Utilizări comune
RANSAC	Foarte robust	Mare (dependent de numărul de iterări)	Da (cu parametri ajustați)	Medie	Estimarea modelelor, potrivire de caracteristici, 3D reconstrucție
SIFT	Medie	Foarte mare	Nu	Foarte precis	Detectie de caracteristici la scara, recunoaștere de obiecte, panorame
ORB	Medie	Scăzut	Da	Precizie bună	Aplicații în timp real, robotică, AR
Harris Corner	Scăzut	Medie	Da	Precizie bună	Detectia colțurilor, urmărirea mișcării
Shi-Tomasi	Medie	Medie	Da	Precizie bună	"Good Features to Track", urmărirea mișcării
FLANN	Robust	Scăzut (când este folosit cu RANSAC)	Da	Precizie variabilă	Căutare rapidă a vecinilor, potrivire de caracteristici
BRIEF	Medie	Foarte scăzut	Da	Precizie medie	Detectie de caracteristici, potrivire rapidă
FREAK	Medie	Scăzut	Da	Precizie bună	Recunoaștere facială, aplicații mobile
FAST	Scăzut	Foarte scăzut	Da	Precizie medie	Aplicații în timp real, urmărirea mișcării, robotică
BFM (Brute-Force Matching)	Medie	Mare (pentru seturi mari de date)	Nu	Foarte precis	Potrivirea caracteristicilor brute, analiză detaliată

Tabel 3.1: Comparație rudimentară între algoritmi standard.

## Capitolul 4. Analiză și fundamentare Teoretică

### 4.1. Modelul cazurilor de utilizare

Aplicația din cadrul acestui proiect are mai mult scop demonstrativ, așa că aceasta a fost concepută să ofere cât mai multe date fără a pierde timpul utilizatorului în meniuri și în pagini de interfețe. Astfel aplicația suportă numai două cazuri propriu zise de utilizare: Selectarea unui anumit set de algoritmi pentru crearea unei panorame sau folosirea tuturor combinațiilor valide de algoritmi.



Figură 4.1: Cazurile de utilizare

### 4.2. Cazurile de utilizare

Cazurile de utilizare presupun că actorul a pornit aplicația și a selectat cele două imagini ce vor fi procesate. Acesta primește ca răspuns întrebarea dacă vrea să folosească numai un set sau toate seturile valide.

- Cazul singurului set:

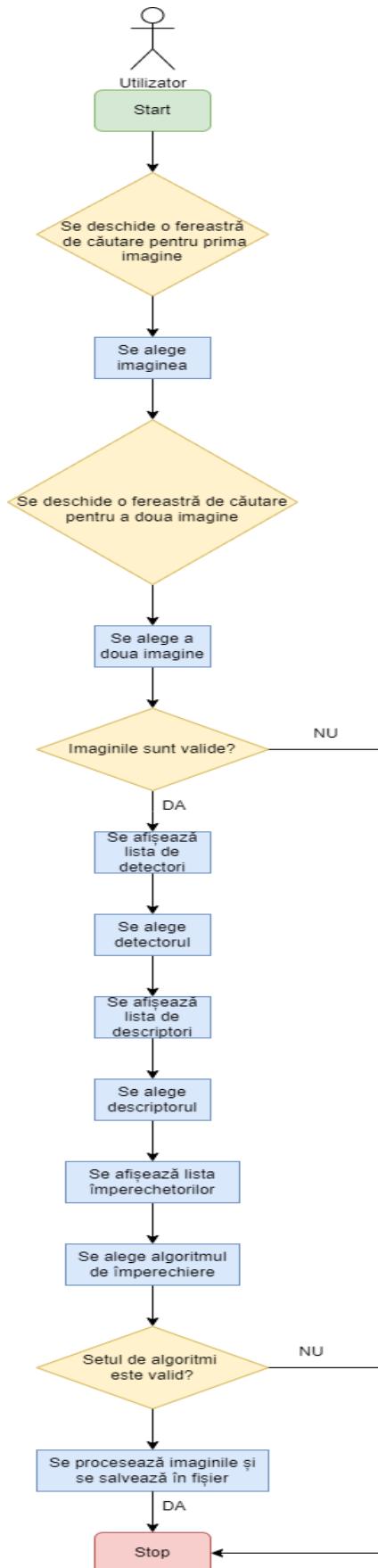
Începutul cazului de utilizare: Utilizatorul alege opțiunea singurului set.

Pasul 1: Utilizatorul va primi ca răspuns un set de algoritmi de detecție implementați în aplicație. Utilizatorul alege algoritmul de detecție dorit.

Pasul 2: Utilizatorul va primi ca răspuns un set de algoritmi de descriere implementați în aplicație. Utilizatorul alege algoritmul de descriere dorit.

Pasul 3: Utilizatorul va primi ca răspuns un set de algoritmi de împerechere implementați în aplicație. Utilizatorul alege algoritmul de împerechere dorit.

Sfârșitul cazului de utilizare: Sistemul afișează pe ecran confirmarea că setul ales este valid și va începe procesarea imaginilor conform acestuia și va slava rezultatele în fișierul cu imaginile selectate. În cazul în care nu este valid atunci sistemul va afișa o eroare.



Figură 4.2: Cazul de utilizare

### 4.3. Filtrul Log-Gabor

Inițial funcția Gabor a fost gândită pentru o singură dimensiune ci numai mai târziu a fost dezvoltată ca un filtru pentru semnalele 2D, conform [12]. Aceasta funcționează prin analizarea conținutului unei imagini atât în domeniu spațial cât și frecvențial. Acest fapt îl face ideal pentru a extrage informații atât despre obiectele din imagine cât și despre texturile lor. Considerat a opera similar funcționalități ochiului uman, acesta a crescut rapid în popularitate și considerare.

Totuși acesta nu este lipsit de dezavantaje componența sa zero, numită DC obținută prin calculul frecvenței folosind funcția sinusoidală, răspunsul limitat la extreame de frecvență și funcția sa Gaussiană ce limitează precizia în spațiul frecvențial sunt doar câteva din neajunsurile acestuia.

Filtrul Gabor poate fi reprezentat prin o funcție complexă sinusoidală modulată de o funcție Gaussiană:

$$G(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{-2\sigma^2}\right) \cdot \cos\left(2\pi \frac{x'}{\lambda} + \psi\right)$$

Unde:

- $(x, y)$ : Coordonatele spațiale în imagine.
- $x' = x\cos(\theta) + y\sin(\theta)$ : Rotația coordonatei  $x$ .
- $y' = -x\sin(\theta) + y\cos(\theta)$ : Rotația coordonatei  $y$ .
- $\lambda$ : Lungimea de undă a factorului sinusoidal (controlează scara filtrului).
- $\theta$ : Orientarea filtrului Gabor (unghi în radiani).
- $\psi$ : Decalaj de fază (deplasează sinusoida).
- $\sigma$ : Abaterea standard a envelopei gaussiene (controlează lățimea de bandă).
- $\gamma$ : Raportul de aspect spațial (controlează elipticitatea Gaussianului, de obicei setat la 1 pentru simetria circulară).

Filtrul Log-Gabor reprezintă o îmbunătățire semnificativă față de filtrul Gabor tradițional, oferind o capacitate mai bună de analiză a frecvențelor și detectare a caracteristicilor fine, ceea ce îl face ideal pentru o gamă largă de aplicații în procesarea și analiza imaginilor. Deși poate necesita o configurație mai complexă și resurse computaționale mai mari, beneficiile sale în precizia și sensibilitatea detecției fac ca Log-Gabor să fie preferat în multe scenarii critice de imagine.

Un filtru Log-Gabor poate fi construit folosind două componente: radială și unghiulară. Componenta radială reglează banda de frecvență și orientarea răspunsului filtrului. Aceasta este inițial definită în [13] ca:

$$G_{s(f)} = \exp\left(-\frac{\left(\log\left(\frac{f}{f_s}\right)\right)^2}{2\left(\log\left(\frac{\sigma_f}{f_s}\right)\right)^2}\right)$$

Unde:

$f_s = 1/\lambda$  : Frecvența radiană a filtrului.

$\lambda$  : Lungimea de undă a filtrului.

$\sigma_{fs}$  : Lățimea de bandă radială de  $B_f$  în octave definite ca:

$$B_f = 2\sqrt{2\log(2)} |\log(\sigma_f/f_s)|$$

Componenta unghiulară a funcției log-Gabor este definită ca:

$$G_o(\theta) = \exp\left(-\frac{(\theta - \theta_o)^2}{2\sigma_{\{\theta_o\}}^2}\right)$$

Unde:

- $\theta_o$  : Unghiul de orientare al filtrului
- $\sigma\theta_o$  : Lățimea de bandă a lui  $B\theta$  în octave.

$B\theta$  este definită ca:

$$B\theta = 2\sigma\theta\sqrt{2\log(2)}$$

Ambele componente radiale și unghiulare sunt înmulțite împreună pentru a construi filtre log-Gabor 2D în jurul unei frecvențe centrale ( $f_s, \theta_o$ ):

$$G_{\{s,o\}(f,\theta)} = G_s(f) \times G_o(\theta)$$

Deoarece acest proiect este dezvoltat folosind limbajul C++ aceste formule reprezintă o problemă datorită necesității de a face calcule în multimea numerelor complexe, în special pentru componenta unghiulară. O soluție alternativă este prezentată pe pagina web [14], unde componenta unghiulară este calculată folosind funcții trigonometrice:

$$\begin{aligned} G_o(\sigma) = \exp & (-| \operatorname{atan2}(\sin(\theta) \cdot \cos(\text{angl}) - \cos(\theta) \cdot \sin(\text{angl}), \cos(\theta) \\ & \cdot \cos(\text{angl}) + \sin(\theta) \cdot \sin(\text{angl}))|^2 \times \frac{1}{2\sigma^2}) \end{aligned}$$

Unde:

$$\theta = \operatorname{atan2}(-y, x)$$

Filtrul rezultat se poate aplica conform articoului [15] unei imagini urmând secvența imaginii inițiale:

- Aplicăm transformata Fourier asupra imaginii.
- Convoluțione cu filtrul Log-Gabor calculat.
- Aplicăm inversa transformatei Fourier asupra imaginii.
- Calculăm valoarea absolută.
- Facem eliminare non-maxima peste matricea rezultată pentru a obține punctele cheie.
- Se repetă procedeul folosind un alt filtru Log-Gabor cu parametrii schimbați pentru a obține rezultate cu multiple scalări și multiple orientări.

În articolul [15] se prezintă și un algoritm calibrat pentru descrierea punctelor cheie rezultate aplicării filtrelor Log-Gabor și recomandă calcule adiționale împerecheri trăsăturilor pentru cele mai bune rezultate. Aceasta este în afara cadrului proiectului interesul fiind în detector și comportarea acestuia folosind algoritmi deja cunoscuți.

Pseudocodul pentru generarea și aplicarea unui filtru Log-Gabor unei imagini:

Function applyLogGaborFilter(src\_gray, angl, sig\_fs, lam, theta\_o, threshold, cutoff, sharpness, imageName, id)

```

radius := createNormalizedRadius(rows, cols);
logGabor := createLogGaborFilter(radius, lam, sig_fs);
lowPassFilter := createLowPassFilter(radius, cutoff, sharpness);
multiply(logGabor, lowPassFilter, logGaborFiltered);
spread := createAngularComponent(cols, rows, angl, theta_o);
multiply(spread, logGaborFiltered, filter);

m := getOptimalDFTSize(rows);
n := getOptimalDFTSize(cols);
copyMakeBorder(src_gray, padded, 0, m - rows, 0, n - cols);
dft(complexI, complexI);
fftShift(complexI, complexI);

```

```
applyFilterManually(complexI, filter);
fftShift(complexI, complexI);

idft(complexI, imgf, DFT_REAL_OUTPUT);
normalize(response, response, 0, 1, NORM_MINMAX);
return response;
```

În acest pseudocod filtrul Log-Gabor este format din componenta unghiulară, angulară și un filtru trece-jos. Filtrul are rolul de a netezi imaginea și a reduce din zgomotul imaginii. La final imaginea este adusă înapoi în domeniul spațial pentru vizualizare.

#### **4.4. Resursele necesare**

Pentru implementarea filtrului Log-Gabor și dezvoltarea aplicației aferente s-a ales mediul de dezvoltare Visual Studio cu limbajul C++ și bibliotecile: iostream (pentru comunicarea cu afișorul), fstream (pentru comunicarea cu fișiere), string (pentru manipularea string-urilor pentru mesaje și căi de acces), chrono (pentru măsurarea eficienței în timp), stdio, tchar, SDKDDKVer, ShlObj, windows, CommDlg (pentru selectarea imaginilor), OpenCV (pentru algoritmi și operați pe imagini), vector, algoritm și cmath (pentru efectuarea calculelor matematice). Bibliotecile au fost importate folosind sistemul Vpckg.

Dezvoltarea sa făcut pe calculator personal cu sistem de operare Windows 11.

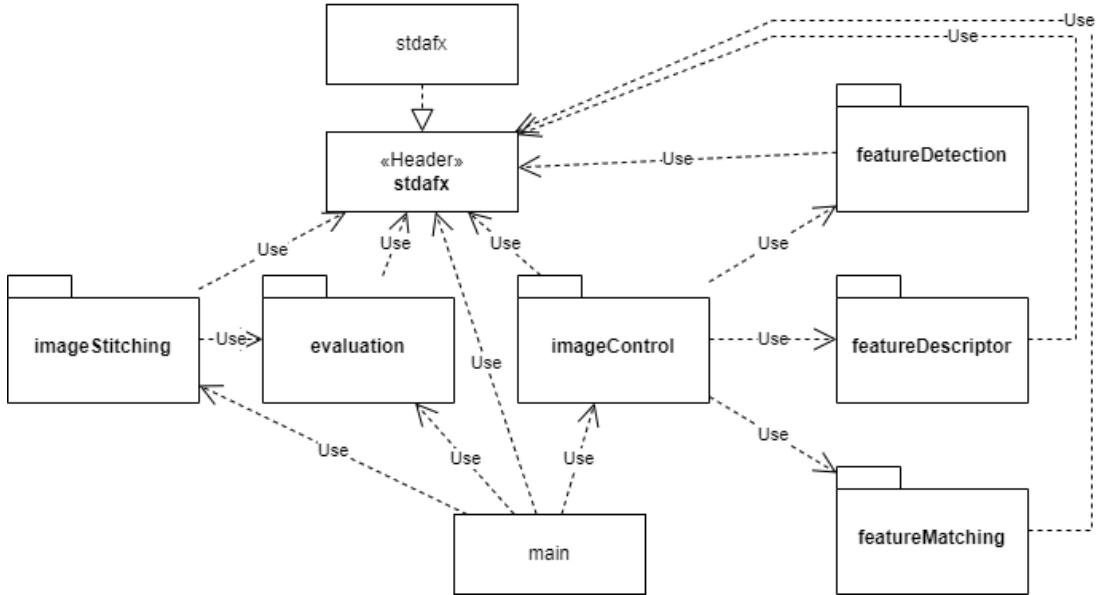
## **Capitolul 5. Proiectare de detaliu și implementare**

### **5.1. Schema generală a aplicației**

Scopul principal al acestei aplicații este de a demonstra utilitatea filtrelor Log-Gabor prin utilizarea acestora în paralel și comparativ cu alți algoritmi pentru rezolvarea problemei de lipire (cusătură) a două imagini. Aplicația este concepută pentru a evidenția cum filtrele Log-Gabor, cunoscute pentru capacitatea lor de a extrage eficient caracteristici de la diferite scări și orientări, îmbunătățesc procesul de îmbinare a imaginilor comparativ cu alte metode tradiționale. Din punct de vedere arhitectural aplicația poate fi separată în șapte părți:

- Main-ul aplicației alături cu biblioteca costum stdax: Acest modul conține funcția principală de pornire a aplicației și elementele de bază care sunt utilizate pe parcursul întregului proces. Biblioteca stdax include definiții globale, setări predefinite și funcții de ajutor comune, facilitând gestionarea și organizarea codului.
- imageStiching : Această bibliotecă conține funcțiile specifice necesare pentru realizarea cusăturii/lipirii imaginilor.
- evaluation : Modulul de evaluare include funcțiile pentru măsurarea performanței algoritmilor de lipire.
- imageControl : Biblioteca funcțiilor de operare pe imagini. Acesta oferă funcții menite pentru manipularea și aplicarea algoritmilor asupra imaginilor.
- featureDetection : Biblioteca dedicată detecției de caracteristici folosește diferiți algoritmi pentru identificarea punctelor de interes din imagini. Algoritmii precum SIFT, ORB, FAST și Log-Gabor sunt utilizati pentru a detecta colțuri, margini și alte caracteristici semnificative care pot fi ulterior folosite pentru alinieri precise între imagini.
- featureDescription : După ce punctele de interes sunt detectate, acest modul se ocupă cu descrierea caracteristicilor. Algoritmii precum SIFT, ORB, FREAK și BRIEF sunt utilizati pentru a genera descriptori care codifică informațiile din jurul punctelor cheie. Acest pas este esențial pentru potrivirea corectă a caracteristicilor între imagini.
- featureMatching : În acest modul, caracteristicile descrise sunt comparate pentru a găsi potriviri între imagini. Algoritmi precum Brute-Force Matching (BFM) și Fast Library for Approximate Nearest Neighbors (FLANN) sunt folosiți pentru a stabili corespondențe între punctele cheie, facilitând astfel alinierea și cusătura imaginilor.

Această abordare a fost aleasă pentru a facilita îmbunătățirea codului cât mai ușor posibil. Prin modularitatea sa putut separa funcțiile necesare de cele non relevante în cazul fiecărei operați, iar navigarea în cadrul proiectului să fie cât mai intuitivă. Procesul de dezvoltare fiind unul destul demeticulos, necesită multiple adaptări și schimbări pentru obținerea rezultatelor stabile ce se pot folosi în paralel cu ceilalți algoritmi.



Figură 5.1: Arhitectura aplicației

Componentele menționate mai sus interacționează între ele prin apele de funcții în timpul procesării imaginilor. O rulare tipică a aplicației implică următoarele etape:

- Încărcarea imaginilor:

Aplicația deschide o fereastră de dialog pentru ca utilizatorul să selecteze prima imagine de procesat. Odată selectată prima imagine, o a doua fereastră de dialog permite alegerea celei de-a doua imagini care va fi lipită cu prima.

- Configurarea opțiunilor de rulare:

Utilizatorul are posibilitatea de a selecta modul de operare al aplicației. Aceasta poate include alegerea algoritmilor de detecție și descriere a caracteristicilor.

- Procesarea imaginilor:

Aplicația începe cu preprocesarea imaginilor pentru îmbunătățirea calității și eliminarea zgomotului. Urmează detectia și descrierea caracteristicilor, potrivirea acestora și, în final, realizarea efectivă a cusăturii imaginilor.

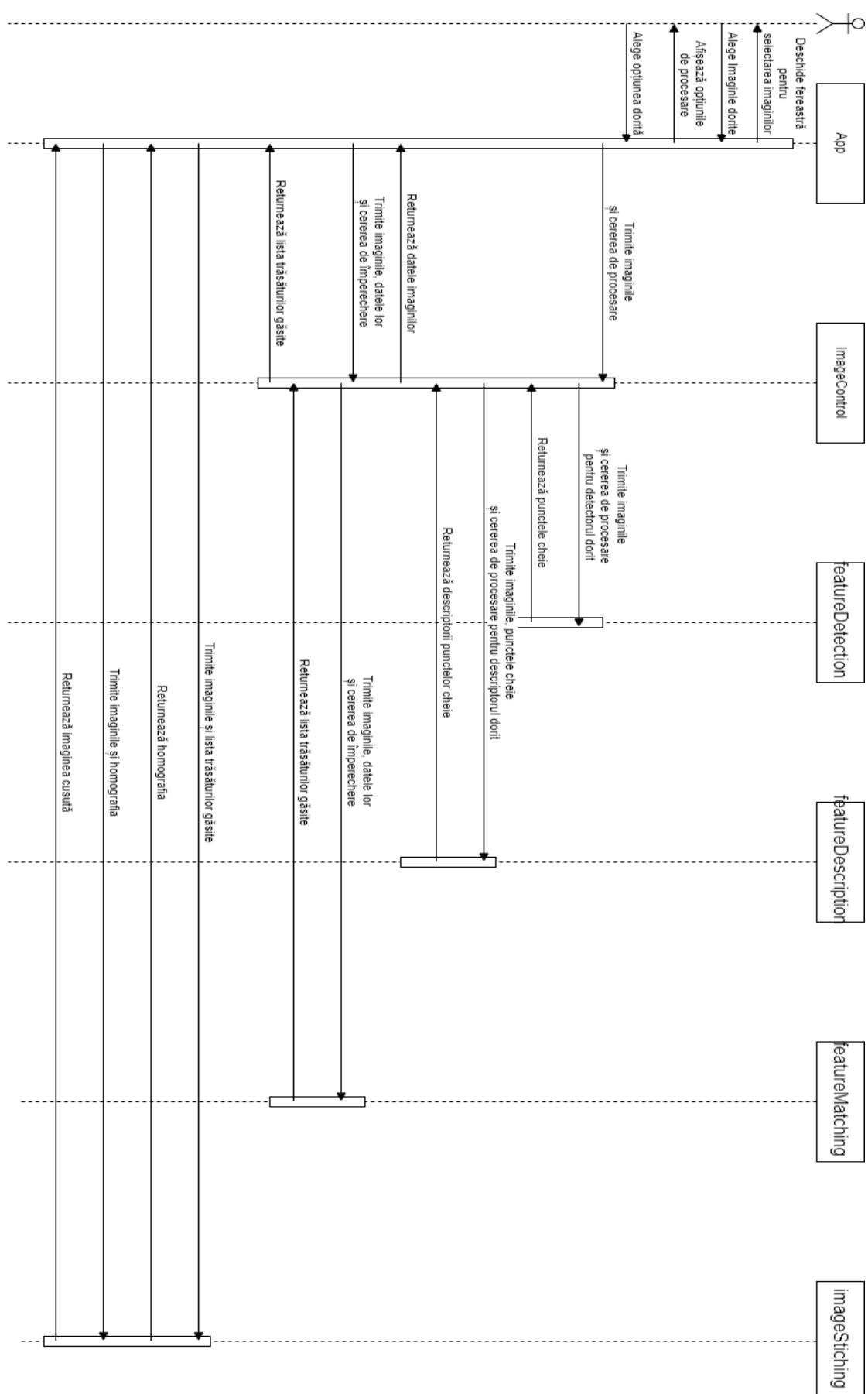
- Salvarea rezultatelor:

Imaginiile finale rezultate din procesul de lipire sunt salvate alături de rezultatele parțiale și metricele de evaluare, oferind utilizatorului un feedback cu privire la calitatea procesării.

## 5.2. Main-ul și componenta globală

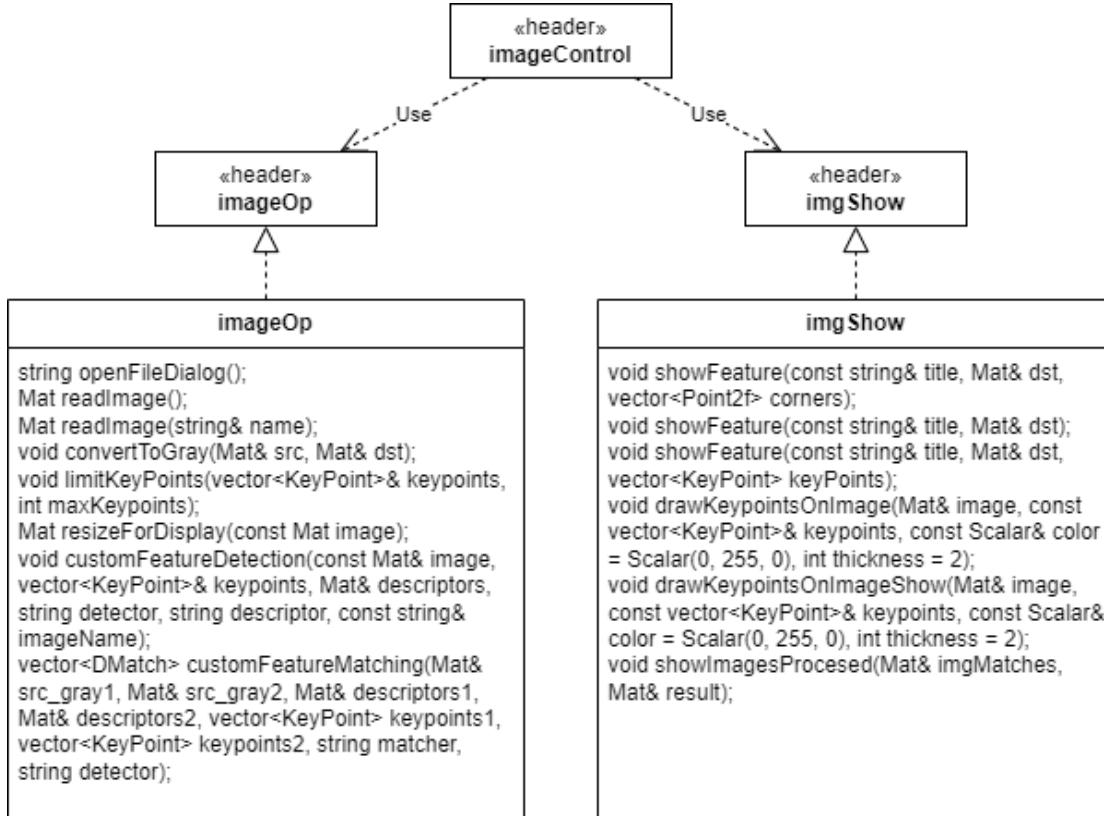
Clasa main are rolul de a servi ca punct de intrare în aplicație, acesta controlează fluxul programului și asigură ca condițiile de avansare sunt respectate. Aceasta este componenta din care pornesc următoarele operații: citirea imaginilor, verificare și validarea imaginilor, conversia acestora, selectarea modului de operare și a algoritmilor, procesarea imaginilor, detectia și descrierea caracteristicilor, împerecherea trăsăturilor, realizarea lipirii imaginilor și salvarea rezultatelor.

Componenta globală este reprezentată de headerul stdax.h. Acesta asigură modularitatea și îmbunătățește compilarea. Head-ărul cuprinde: librării des folosite, constante și valori globale și utilizări de namespace.



Figură 5.2: Diagrama de secvență

### 5.3. Pachetul controlorului



Figură 5.3: Pachetul controlorului

- **imageOp**

În acest fișier sunt implementate funcții de bază pentru deschiderea și citirea imaginilor, conversia imaginilor la scala de gri, limitarea numărului de puncte cheie, redimensionarea imaginilor pentru afișare, detectarea personalizată a caracteristicilor și împerecherea caracteristicilor între două imagini. Codul este flexibil, permitând utilizatorilor să aleagă dintr-o varietate de algoritmi de detecție și descriere, ceea ce îl face potrivit pentru viitoare aplicații în prelucrarea imaginilor și viziunea computerizată.

Funcția `openFileDialog` deschide un dialog de selectare a fișierelor pentru utilizator, permitându-i să aleagă un fișier de pe disc. Utilizează structura `OPENFILENAME` din API-ul Windows pentru a configura și deschide fereastra de dialog. Dacă utilizatorul selectează un fișier, funcția returnează calea către acel fișier. În caz contrar, returnează un sir gol. Aceasta este utilizată în funcțiile de citire a imaginilor ce urmează.

Funcția `readImage`, versiunea ei supraîncărcată primește un parametru de tip referință la string pentru a returna numele fișierului imaginii fără extensie. După citirea imaginii, funcția verifică dacă calea fișierului este validă și dacă imaginea a fost încărcată corect, afișând mesaje de eroare dacă este cazul.

Funcția `convertToGray` face conversia unei imagini în formatul alb-negru.

Funcția `limitKeyPoints` limitează numărul de puncte cheie dintr-un vector la un număr maxim specificat de `maxKeypoints`. Punctele cheie sunt sortate în funcție de răspunsul lor (response) înainte de a fi redimensionate pentru a păstra doar cele mai

importante puncte. Aceasta este utilă în cazurile când numărul de puncte cheie este foarte mare și îngreunează considerabil procesarea imaginilor. Prin aceasta se sacrifică din calitate pentru a avea o rulare în timp util.

Funcția `resizeForDisplay` redimensionează imaginea pentru a se potrivi într-o înălțime maximă specificată de `MAX_DYSPLAY_HEIGHT`. Aceasta este utilă pentru a afișa imagini mari fără a depăși dimensiunile ecranului. Redimensionarea păstrează proporțiile imaginii.

Funcția `customFeatureDetection` realizează detecția și descrierea caracteristicilor dintr-o imagine, utilizând diferite algoritmi specificați de utilizator. Imaginile sunt analizate pentru a detecta puncte cheie (`keypoints`), și apoi descriptorii corespunzători sunt calculați pentru aceste puncte cheie.

Funcția `customFeatureMatching` Această funcție efectuează împerecherea descriptorilor dintre două imagini utilizând diferiți algoritmi de împerechere (FLANN, BFM). Alegerea algoritmului se face pe baza parametrilor de intrare, iar rezultatele sunt returnate sub formă de vector de potriviri (`DMatch`).

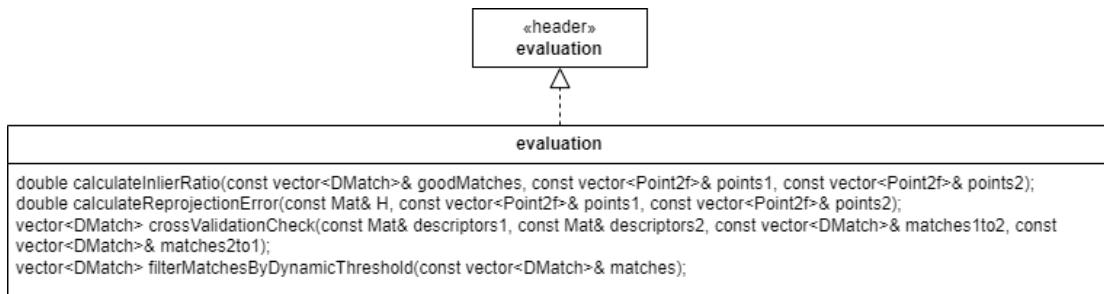
- `imgShow`

Acest fișier cuprinde funcțiile ce ajută la vizionarea imaginilor marcând unde este cazul punctelor cheie.

Funcția `showFeature` vine în multiple forme: cea pentru marcarea colțurilor detectate (concepță pentru algoritmi Harris și Shi-Tomasi), cea pentru puncte cheie și cea pentru afișarea normalizată. În toate cazuri funcția primește un titlu pentru fereastra de afișare, imaginea de destinație. În cazul colțurilor și punctelor cheie un vector adițional cu datele fiecărui punct relevant pe imaginea finală se va marca un cerc colorat aleatoriu poziția acestuia pentru a putea fi mai ușor de recunoscut în evaluarea vizuală. Funcțiile `drawKeypointsOnImage` și `drawKeypointsOnImageShow` funcționează similar cu diferența că în acestea se poate seta prin parametrii grosimea și culoarea cercului.

Funcția `showImagesProcesed` afișează două imagini: una conținând potrivirile dintre două seturi de caracteristici (`imgMatches`) și cealaltă fiind rezultatul procesării ulterioare (cum ar fi îmbinarea imaginii). Imaginile sunt redimensionate pentru a se potrivi ecranului și apoi afișate în ferestre de dialog separate folosind `imshow`.

## 5.4. Pachetul evaluatorului



Figură 5.4: Pachetul evaluatorului

Acest fișier de cod conține funcții pentru evaluarea algoritmilor de detectare, descriere și lipire a imaginilor. Funcțiile sunt esențiale pentru stabilirea consistenței și acurateței potrivirilor în cadrul unei aplicații de procesare a imaginii.

Funcția `calculateInlierRatio` calculează raportul inlierilor față de numărul total de potriviri. Un inlier este o potrivire care este considerată corectă în contextul unui model geometric specific, de exemplu, o transformare omografică. Formula folosită

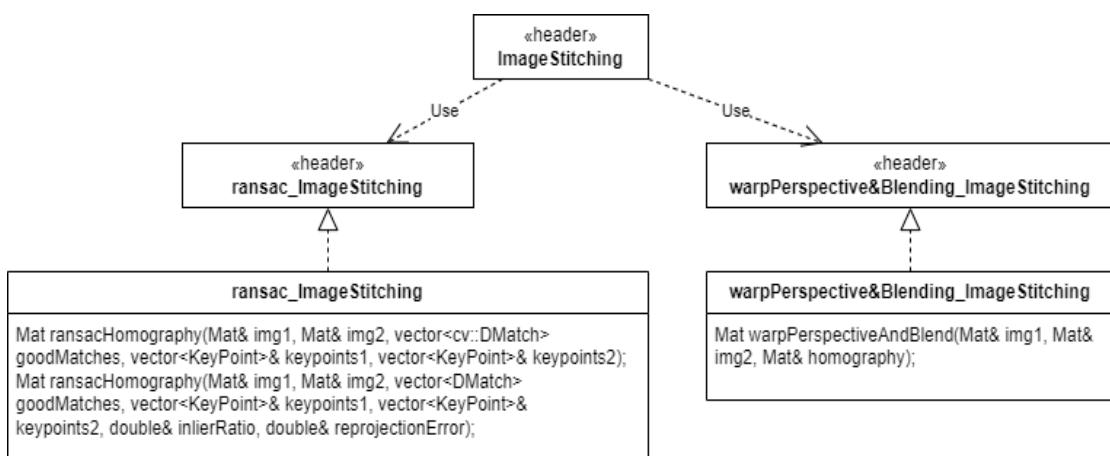
este Inlier Ratio =  $N_m / N_c$ , unde  $N_m$  este numărul de inlieri, iar  $N_c$  este numărul total de potriviri. Un raport mare de inlieri indică faptul că o proporție mare de potriviri sunt inlieri (potriviri corecte). Un raport mic de inlieri indică o proporție redusă de inlieri, sugerând posibilitatea unor erori de potrivire. Ca recomandare un raport de inlieri peste 0.7 este considerat bun.

Funcția crossValidationCheck realizează o verificare de validare încrucișată a potrivirilor pentru a confirma consistența acestora. O potrivire este considerată consistentă dacă este reciprocă între cele două seturi de imagini (prima și a doua imagine selectate la începutul rulării în acest caz).

Funcția calculateReprojectionError calculează eroarea de reproiecție, care reprezintă distanța medie între punctele cheie potrivite și punctele lor proiectate folosind o omografie (parametrul  $H$  în cazul acesta). Formula folosită este  $\text{reprojectionError} = \text{totalError} / \text{numMatches}$ , unde  $\text{totalError}$  este suma distanțelor dintre punctele potrivite și cele proiectate, iar  $\text{numMatches}$  este numărul total de potriviri. O eroare de proiectare mică indică o potrivire bună între punctele cheie și proiectarea lor. Pe când o eroare de proiectare mare sugerează că punctele cheie nu se aliniază bine cu proiectările lor, fapt ce va afecta calitatea potrivirii. Ca recomandare o eroare de re-proiecție ar trebui să fie sub 5.0 pixeli.

Funcția filterMatchesByDynamicThreshold filtrează potrivirile între descriptorii de caracteristici pe baza unei distanțe dinamice calculate. Aceasta ajută la eliminarea potrivirilor care au distanțe semnificativ mai mari decât media, considerându-le pe acestea ca fiind eronate sau nerelevante. Funcția ajută la îmbunătățirea calității potrivirilor într-un set de date de imagini. Pragul este definit ca  $\text{threshold} = \text{meanDistance} + 2 * \text{stdevDistance}$ , unde  $\text{meanDistance}$  este media aritmetică a tuturor distanțelor dintre descriptorii de caracteristici care au fost potriviți între două imagini și  $\text{stdevDistance}$  reprezintă deviația standard a distanțelor dintre descriptorii de caracteristici potriviți. Funcția returnează un vector de obiecte DMatch filtrate, care conține doar potrivirile ale căror distanțe sunt mai mici sau egale cu un prag dinamic calculat.

## 5.5. Pachetul cusătoriilor



Figură 5.5: Pachetul cusătoriilor

- warpPerspectiveAndBlend

Funcția warpPerspectiveAndBlend aplică o transformare de perspectivă asupra celei de-a doua imagini (img2) utilizând o matrice de omografie (homography).

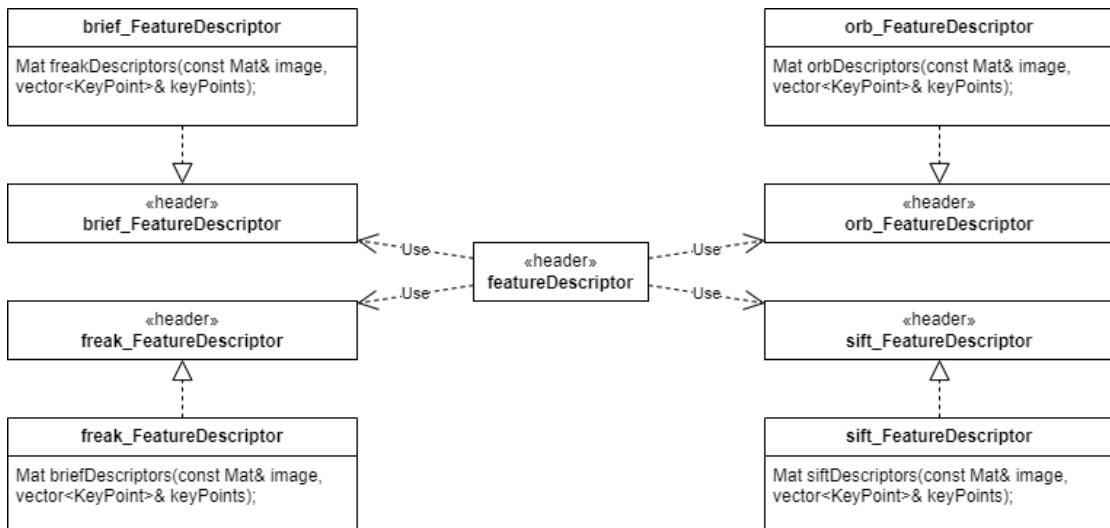
Imaginea transformată este apoi combinată cu prima imagine (img1) într-un rezultat final. Funcția realizează îmbinarea celor două imagini într-o singură imagine mare. Matricea de omografie 3x3 care definește transformarea de perspectivă aplicată asupra celei de-a două imagini. Această matrice proiectează punctele din coordonatele imaginii sursă în coordonatele imaginii destinație.

Imaginea combinată rezultată este returnată aceasta conține atât prima imagine (img1), cât și pe cea de-a două (img2), transformată și lipită. Rezultatul este de dimensiuni mai mari pentru a acomoda ambele imagini după aplicarea transformării de perspectivă.

- `ransac_ImageStitching`

Funcțiile `ransacHomography` calculează matricea de omografie care mapează punctele dintr-o imagine (img2) în celalătă (img1) folosind metoda RANSAC (Random Sample Consensus) pentru a se asigura că omografia este robustă la zgromot și la corespondențe greșite. Omografia este determinată pe baza unui set de potriviri bune (`goodMatches`) între punctele cheie detectate în ambele imagini. Diferența dintre cele două funcții este că cea din urmă are parametrii `inlierRatio` și `reprojectionError` acestea sunt variabile de ieșire ce conțin raportul de inlieri calculat după determinarea omografiei și respectiv eroarea de reproiecție calculată pe baza omografiei obținute.

## 5.6. Pachetul descriptorilor



Figură 5.6: Pachetul descriptorilor

- `brief_FeatureDescriptor`

Funcția `briefDescriptors` extrage descrieri ale caracteristicilor (descriptors) dintr-o imagine dată utilizând algoritmul BRIEF (Binary Robust Independent Elementary Features). BRIEF este un descriptor binar rapid și eficient folosit în recunoașterea și potrivirea caracteristicilor de imagine, ideal pentru aplicații care necesită performanță în timp real.

- `freakDescriptors`

Funcția `freakDescriptors` este utilizată pentru a calcula descriptorii FREAK pentru un set de keypoints într-o imagine. Acești descriptori sunt adesea folosiți în combinație cu detectoare rapide de puncte de interes, cum ar fi FAST sau ORB, pentru aplicații de viziune computerizată care necesită performanță în timp real și rezistență la zgromot și variații de iluminare.

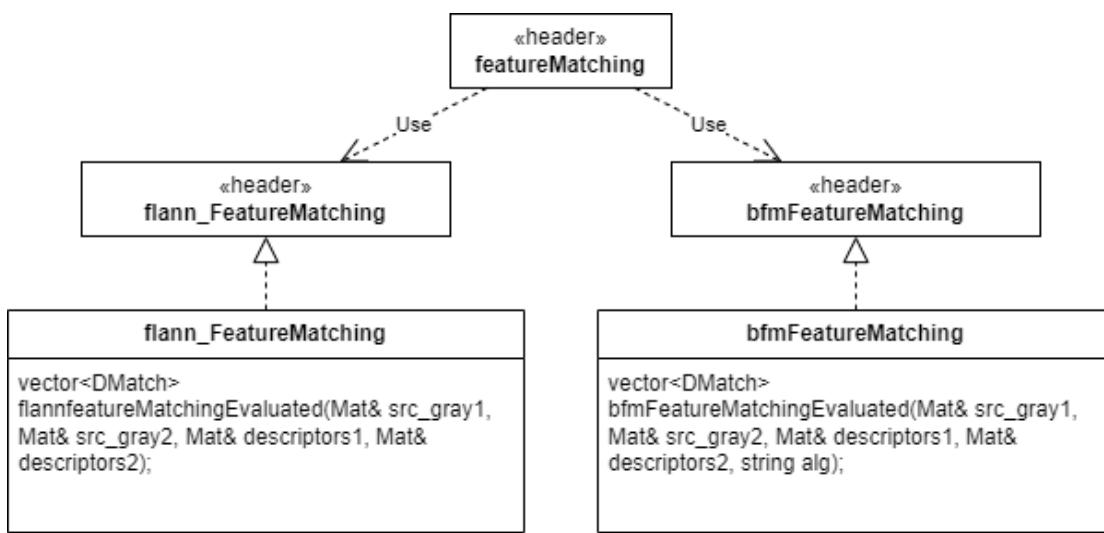
- orbDescriptors

Funcția orbDescriptors este utilizată pentru a calcula descriptorii ORB pentru un set de keypoints într-o imagine. ORB combină detecția rapidă a punctelor de interes prin metoda FAST cu descrierea prin metoda BRIEF modificată pentru a fi insensibilă la rotații.

- siftDescriptors

Funcția siftDescriptors este utilizată pentru a calcula descriptorii SIFT pentru un set de keypoints într-o imagine. Aceasta este compatibilă numai cu punctele cheie obținute cu detectorul SIFT. Descriptorii SIFT sunt recunoscuți pentru robustețea lor în potrivirea caracteristicilor.

## 5.7. Pachetul împerechitorului de trăsături



Figură 5.7: Pachetul împerechitorului de trăsături

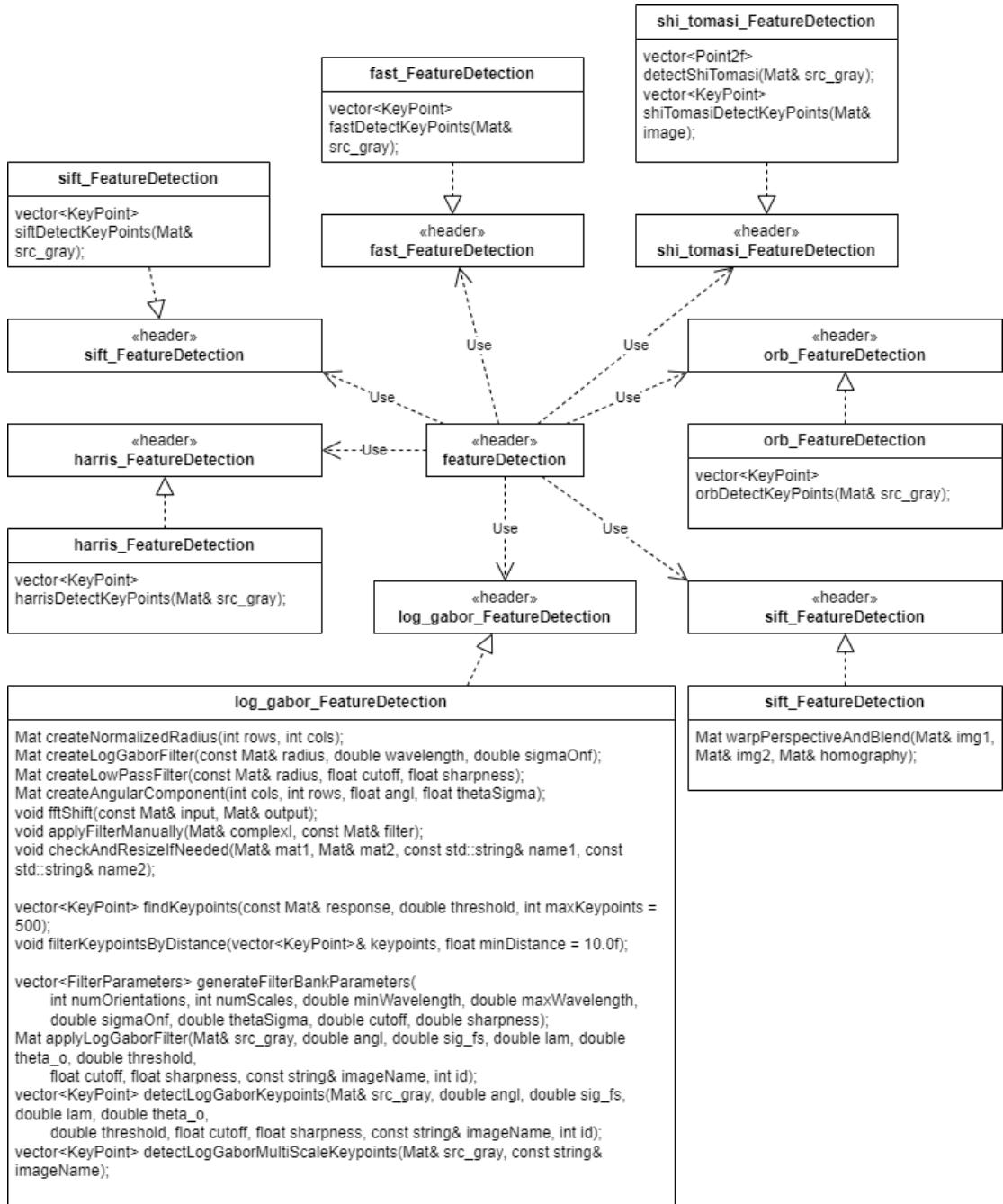
- flann\_FeatureMatching

Funcția flannfeatureMatchingEvaluated realizează potrivirea caracteristicilor între două imagini utilizând un algoritm de potrivire bazat pe FLANN (Fast Library for Approximate Nearest Neighbors). Aceasta evaluează potrivirile folosind o verificare încrucișată (cross-validation) pentru a asigura consistența, apoi filtrează potrivirile utilizând un prag dinamic bazat pe distanța dintre descriptorii de caracteristici. Scopul acestei funcții este de a returna potriviri robuste și fiabile între caracteristicile celor două imagini.

- bfmFeatureMatching

Funcția bfmFeatureMatchingEvaluated utilizează un BFMatcher (Brute-Force Matcher) pentru a potrivi descriptorii de caracteristici între două imagini, utilizând un tip de normă corespunzător algoritmului de descriere. Funcția include, de asemenea, o verificare încrucișată pentru a asigura consistența potrivirilor și aplică un prag dinamic pentru a filtra potrivirile necorespunzătoare. Scopul principal al acestei funcții este de a returna un set de potriviri fiabile între două seturi de descriptorii de caracteristici.

## 5.8. Pachetul detectorilor



Figură 5.8: Pachetul detectorilor

- fast\_FeatureDetection

În această funcție, este utilizat detectorul de caracteristici FAST (Features from Accelerated Segment Test) pentru a detecta punctele cheie într-o imagine alb-negru, aceasta va returna un vector de tipul keypoints ce va conține datele dorite.

- harris\_FeatureDetection

Funcția harrisDetectKeyPoints detectează puncte cheie dintr-o imagine alb-negru folosind detectorul Harris, un algoritm cunoscut pentru detecția de colțuri și puncte de interes. Apoi, aceste puncte de interes sunt convertite într-un format de puncte cheie, care poate fi utilizat pentru descrierea și potrivirea caracteristicilor.

- orb\_FeatureDetection

Funcția orbDetectKeyPoints detectează puncte cheie într-o imagine alb-negru utilizând algoritmul ORB (Oriented FAST and Rotated BRIEF) și returnează un vector cu acestea.

- shi\_tomasi\_FeatureDetection

Funcția detectShiTomasi utilizează algoritmul Shi-Tomasi pentru a detecta colțurile în imaginea dată. Aceasta returnează un vector de puncte Point2f care conțin coordonatele colțurilor detectate în imagine.

Funcția shiTomasIDetectKeyPoints convertește punctele de interes detectate de detectShiTomasi în obiecte de tip KeyPoint, care sunt structuri mai complexe utilizate pentru diverse operațiuni de procesare a imaginilor.

- sift\_FeatureDetection

Funcția siftDetectKeyPoints utilizează algoritmul SIFT (Scale-Invariant Feature Transform) pentru a detecta punctele de interes într-o imagine. Rezultatele returnate de funcție sunt calibrate pentru a putea fi folosite cu descriptorul SIFT și sunt incompatibile cu ceilalți descriptori.

- Log\_gabor\_FeatureDetection

Funcția createNormalizedRadius construiește o matrice care conține distanțele normalize de la fiecare punct dintr-o matrice de dimensiune (rows, cols) la centrul acestuia. Aceste distanțe sunt utilizate frecvent în prelucrarea imaginii și analiza frecvenței, în special pentru a implementa filtre în domeniul frecvenței. Distanțele sunt normalize față de dimensiunea imaginii pentru a obține valori între 0 și 1, unde 1 reprezintă distanța maximă. În cadrul aplicației aceasta este folosită în cadrul creații filtrului Log-Gabor de detecție a punctelor cheie. Distanța este calculată ca distanță euclidiană de la fiecare punct la centru imaginii conform ecuației lui Pitagora  $\text{radius} = \sqrt{x^2 + y^2}$ . Valoarea din centru este setată la 1 pentru a evita problema calcului  $\log(0)$ .

Funcția createLogGaborFilter construiește un filtru Log-Gabor bidimensional bazat pe o matrice de raze normalize (radius). Funcția necesită: o matrice de tip CV\_32F care conține distanțele normalize ale fiecărui punct față de centrul matricii generate folosind funcția createNormalizedRadius (radius), lungimea de undă la care filtrul Log-Gabor are răspunsul maxim (wavelenght) și factorul de scalare care controlează lățimea benzii de frecvență a filtrului (sigmaOnf). Această funcție va calcula filtrul folosind formula componentei unghiulare a filtrului Log-Gabor detaliată la capitolul 4 și va seta valoarea centrală la 0 pentru evitarea artefactelor asociate cu frecvența zero (componenta DC).

Funcția createLowPassFilter construiește un filtru trece-jos bidimensional, utilizând o matrice de raze normalize (radius). Acest filtru este folosit în procesarea imaginilor pentru a elimina frecvențele înalte, permitând doar frecvențele joase să treacă, ceea ce are ca rezultat o estompare sau netezire a imaginii. Acest filtru va fi înmulțit cu cel obținut de funcția createLogGaborFilter în alcătuirea filtrului final Log-Gabor. Funcția necesită o matrice de tip CV\_32F cu distanțele normalize, frecvența de tăiere a filtrului ce reprezintă frecvența de tranziție dintre păstrarea frecvențelor joase și atenuarea frecvențelor înalte și controlul pantei de tăiere ce reprezintă cât de bruscă va fi tranziția dintre frecvențe.

Funcția createAngularComponent creează o componentă unghiulară utilizată pentru filtrele log-Gabor, ceea ce este frecvent în procesarea imaginilor pentru a evidenția caracteristicile bazate pe orientarea și frecvența conținutului imaginii. Această funcție este utilă pentru a calcula distribuția unghiulară a unei imagini în raport cu un unghi specificat, oferind o modalitate de a analiza conținutul în funcție de direcție.

Parametrii funcției constau în : număr de coloane, număr de rânduri, unghiul de referință în radiani și deviatorul standard al distribuției unghiulare.

Unghiul de referință definește orientarea principală a filtrului. De exemplu, dacă angl este setat la  $\pi/4$  (care este echivalentul a 45 de grade), atunci componenta unghiulară va favoriza sau va filtra în principal caracteristicile orientate în această direcție. În cadrul unui filtru log-Gabor, acest parametru ajută la identificarea caracteristicilor orientate de-a lungul acestei direcții specifice. Aceasta este exprimată în radian.

Deviația standard a distribuției unghiulare determină cât de sensibil este filtrul la variațiile în jurul unghiului de referință. Este o măsură fără unități, dar reflectă deviația unghiului în radiani.

Funcția `createAngularComponent` va returna o matrice ce reprezintă componenta angulară a filtrului Log-Gabor conform explicației de la capitolul 5.

Funcția `fftShift` efectuează o operațiune de transpunere a componentelor de frecvență zero ale unui spectru în centrul imaginii. Aceasta este o tehnică utilă în procesarea imaginilor, în special atunci când se utilizează transformata Fourier (FFT) și se dorește vizualizarea și analiza frecvențelor într-un mod mai intuitiv.

Funcția `applyFilterManually` aplică un filtru pe o imagine în domeniul frecvenței. Aceasta se ocupă cu manipularea și aplicarea unui filtru pe imaginile de frecvență complexă, care sunt de obicei rezultatul aplicării transformatei Fourier. Funcția este necesară pentru a putea face calcule pe valori complexe în ciuda faptului că mediul de dezvoltare Visual Studio și limbajul C++ nu suportă date în asemenea format în mod eficient. Calculul complex în acest caz realizându-se prin împărțirea planurilor, unul pentru partea reală și altul pentru partea imaginară. Acest lucru permite manipularea fiecărei componente separat. Se aplică filtrul în mod individual pe fiecare plan și se combină înapoi într-o matrice ce va conține o imagine complexă.

Funcția `findKeypoints` are rolul de a extrage punctele de interes dintr-o matrice de răspuns, aplicând o tehnică de suprimare a non-maximului și filtrând punctele în funcție de un prag de intensitate.

Parametrul `response` constă într-o matrice de răspuns, care conține valori numerice ce indică relevanța fiecărui pixel ca punct de interes. De obicei, această matrice este rezultatul unei funcții de detecție a caracteristicilor, iar în cadrul proiectului va fi folosită matricea rezultat din aplicarea filtrului peste imaginea selectată.

Parametrul `threshold` reprezintă pragul minim pentru ca un punct să fie considerat un punct de interes. Numai valorile de răspuns mai mari decât acest prag sunt considerate.

Parametrul `maxKeypoints` reprezintă numărul maxim de puncte de interes care trebuie returnate. Dacă numărul de puncte detectate este mai mare, sunt păstrate doar cele mai relevante.

Funcția `findKeypoints` parcurge mai întâi matricea de răspuns, ignorând marginile imaginii pentru a evita accesarea pixelilor în afara imaginii și posibilele trăsături false oferite de punctele de la margine. Se verifică dacă valoarea de răspuns a pixelului curent este mai mare decât valoarea vecinilor săi direcți (sus, jos, stânga, dreapta) și decât pragul specificat. Dacă toate condițiile sunt îndeplinite, pixelul este considerat un punct de interes. După care punctele de interes sunt sortate în ordine descrescătoare a valorii răspunsului, astfel încât cele mai relevante puncte sunt plasate la începutul listei și se elimină din cele cu răspuns slab în cazul în care numărul acestora depășește un prag setat de o constantă globală. La final lista de puncte de interes este returnată.

Funcția `filterKeypointsByDistance` este utilizată pentru a elimina punctele de interes (keypoints) care sunt prea aproape unele de altele, pe baza unei distanțe minime specificate. Aceasta este o tehnică importantă pentru a evita redundanță în seturile de puncte de interes și pentru a asigura că punctele selectate sunt bine distribuite în imagine.

Funcția `checkAndResizeIfNeeded` este utilizată pentru a verifica șiajusta dimensiunile a două imagini (sau matrice) pentru a asigura că acestea au dimensiuni identice. Aceasta este utilă atunci când se lucrează cu imagini în procesare vizuală sau în viziune computerizată, unde dimensiunile matricelor trebuie să fie compatibile pentru a efectua operații cum ar fi adunarea, înmulțirea sau alte manipulări. Această funcție mai are scopul adițional de a oferi toleranță la erori în cadrul filtrului Log-Gabor. Aceasta va compara mărimele matricelor primite, dacă mărimele nu se potrivesc atunci există riscul ca programul să se întrerupă în timpul rulării și să se piardă informații esențiale. Așa că pentru a garanta funcționarea continuă, matricele vor fi redimensionate la o dimensiunea celei mai mici. Utilizatorul va fi informat de schimbare prin interfața liniei de comandă.

Funcția `generateFilterBankParameters` creează un set de parametri pentru un bank de filtre Gabor utilizat în analiza imaginii. Acest bank de filtre este folosit pentru a extrage caracteristici din imagini la diferite orientări și scale (scări), ceea ce este foarte util în recunoașterea de pattern-uri. Fiecare filtru din bank este caracterizat de o orientare specifică și o lungime de undă (wavelength) particulară. Alte proprietăți includ sigma (pentru factorul de formă a gaussianei), cutoff (pentru filtrarea de frecvență joasă), și sharpness (pentru claritatea filtrului).

- numOrientations: Numărul de orientări diferite pentru care vor fi generate filtrele. De exemplu, 6 orientări ar împărți 180 de grade în intervale de 30 de grade.
- numScales: Numărul de scale (lungimi de undă) pentru care vor fi generate filtrele. Acest parametru controlează cât de fin sau grosolan sunt capturate detaliile.
- minWavelength: Lungimea de undă minimă pentru filtre. Aceasta controlează frecvența cea mai mare (detalii fine) pe care o va captura un filtru.
- maxWavelength: Lungimea de undă maximă pentru filtre. Aceasta controlează frecvența cea mai mică (detalii grosiere) pe care o va captura un filtru.
- sigmaOnf: Proporția între sigma gaussianei în domeniul frecvenței și lungimea de undă centrală. Aceasta definește lățimea de bandă a filtrului Gabor.
- thetaSigma: Dispersia angulară pentru controlul lățimii filtrului în termeni de orientare.
- cutoff: Frecvența de tăiere pentru filtrul de trecere joasă.
- sharpness: Parametrul de claritate pentru filtrul de trecere joasă.

Funcția `detectLogGaborKeypoints` aplică un filtru Log-Gabor asupra unei imagini alb-negru și folosește funcția de detectare a punctelor cheie pe răspunsul acesteia. După care punctele cheie sunt filtrate pentru a asigura relevanța și calitatea acestora.

Funcția `applyLogGaborFilter` generează un filtru Log-Gabor bazat pe un set de parametrii ce poate fi generat de funcția `generateFilterBankParameters`. Acest filtru este aplicat peste imaginea sursă ce se presupune că este în format alb-negru și o va returna pentru a fi vizualizată sau folosită în funcțiile de detectare a punctelor cheie.

# **Capitolul 6. Testare și validare**

## **6.1. Modul de testare**

Pentru testare am folosit calculatorul personal, acesta având specificațiile următoare:

- OS: Windows 11, x64
- Procesor: Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz 2.40 GHz
- RAM: 16 GB
- Placă grafică: NVIDIA GeForce GTX 1660 TI
- Versiunea build-ului folosit: Visual Studio, Release, x64

O rulare completă a constat în alegerea unui set de imagini și procesarea lor folosind toate combinațiile valide de algoritmi. În cazul aplicării filtrului Log-Gabor aceasta cuprindea și salvarea imaginii filtru, imagini peste care s-a aplicat filtrul și un fișier cu datele filtrului. Astfel timpul necesar unei rulări pentru un set ce implică detectorul Log-Gabor va cuprinde și timpul salvării datelor parțiale. La fiecare set s-au salvat ca rezultate parțiale punctele cheie detectate de fiecare algoritm de detecție.

Combinățiile invalide sunt:

- Detector SIFT cu alte descriptoare încălărită descriptorul SIFT, indiferent de algoritmul de împerechere.
- Orice detector cu descriptorul SIFT, exceptie făcând desigur detectorul SIFT, indiferent de algoritmul de împerechere.
- Descriptoarele FREAK, ORB și BRIEF cu împerechitorul FLANN, indiferent de detector. Deoarece acestea sunt concepute să ofere detalii minime în schimbul eficienței în timp, detalii lipsă, precum orientarea, sunt necesare ca FLANN să ofere rezultate valide.

## **6.2. Parametrii algoritmilor**

Algoritmi folosiți, în special în cazul celor de detecție au necesitat parametrii de funcționare ce în condiții ideale necesită calibrări minuțioase ce implică multiple rulări și recalibrări. Din păcate un timp și capacitatea calculatorului pe care s-a făcut dezvoltarea nu au permis acestea aşa că s-au ales un set ce să fie reprezentativ pentru rularea generală a acestora.

### **6.2.1. Calibrarea detectorului de colțuri Harris**

Algoritmul Harris necesită următorii parametrii:

- Număr maxim de colțuri:

Acesta reprezintă un număr maxim de colțuri care să fie luate în considerare ca puncte cheie. Algoritmul Harris are ca funcționalitate principală detectarea tuturor colțuri posibile, iar procesarea tuturor ar fi foarte extinsă în timp și nu ar fi avut șansă să ofere date relevante în comparația cu ceilalți algoritmi. Așa că s-a ales un număr maxim de colțuri care luate începând cu cele ce au cel mai bun răspuns vor fi considerate posibile puncte cheie. Valoarea stabilită pentru teste este de 100000.

- Nivelul de calitate:

Parametru de calitate minimă pentru un colț. Este un factor de scalare aplicat asupra celei mai bune valori de răspuns pentru un colț. Colțurile care au răspunsuri sub acest nivel nu sunt considerate. Sa folosit valoarea 0.01.

- Distanță minimă:

Distanță minimă dintre două colțuri detectate pentru a fi considerate distincte. În teste s-a folosit valoarea 10.

- Mărimea blocului:

Dimensiunea blocului folosit pentru calculul derivatelor de imagine. Acesta are valoarea 3.

- K: Cu valoarea de 0.04.

#### 6.2.2. Calibrarea detectorului de colțuri Shi-Tomasi

Algoritmul Shi-Tomasi necesită parametrii similari cu cei necesari pentru algoritmul lui Harris ce îndeplinește aproape aceeași funcții, algoritmii diferențiindu-se prin metoda de calcul:

- Număr maxim de colțuri: 100000
- Nivelul de calitate: 0.01
- Distanță minimă: 10
- Mărimea blocului: 3
- K: 0.04

#### 6.2.3. Calibrarea detectorului Log-Gabor

Algoritmul Log-Gabor se folosește de următorii parametri:

- Orientare (orientation):

Reprezintă unghiul la care filtrul va fi sensibil. Filtrul va evidenția acele trăsături ce sunt orientate conform acestui parametru. În cadrul testării s-au creat filtre cu orientarea având valori între 0 și 150 de grade.

- Scala sau lungimea de undă (wavelength):

Permite detectarea caracteristicilor la diferite niveluri de detaliu (scări mari pentru detaliu grosier, scări mici pentru detaliu fin). În cadrul testării s-au creat filtre cu scala având valori între 3.0 și 15.0.

- Raportul dintre deviația standard și frecvența centrală a filtrului (sigmaOnf):

Controlează lățimea benzii de frecvență a filtrului Log-Gabor. O valoare mai mare duce la un filtru cu o bandă mai largă, iar o valoare mai mică face filtrul mai selectiv pe frecvență. Aceasta are valoarea 0.55.

- Deviația standard a distribuției unghiulare (thetaSigma):

Controlează cât de sensibil este filtrul la orientări diferite. Valori mai mici fac filtrul mai specific unei anumite orientări, în timp ce valori mai mari permit detectarea caracteristicilor pe o gamă mai largă de unghiuri. Aceasta are valoarea 1.0 radian.

- Frecvența de tăiere a filtrului de trecere joasă (cutoff):

Controlează cât de mult din spectrul de frecvență trece prin filtrul de trecere joasă. O valoare mai mică reduce zgomotul de înaltă frecvență, în timp ce o valoare mai mare permite mai multe detaliu să treacă. Parametrul are valoarea 0.45.

- Parametru de ascuțire pentru filtrul de trecere joasă (sharpness):

Controlează cât de rapid se estompează trecerea de la frecvențele permise la cele respinse. O valoare mai mare face tranziția mai abruptă, ducând la un filtru mai ascuțit. Aceasta are valoarea 15.0.

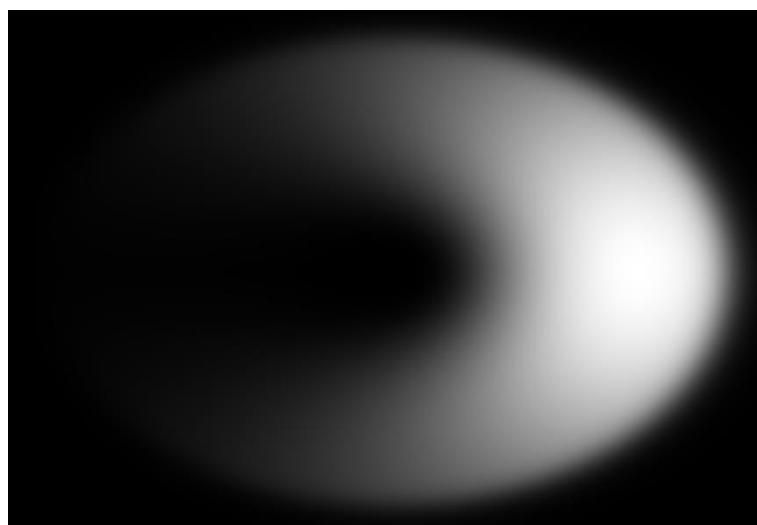
- Limita de detectie (threshold):

Aceasta reprezintă sensibilitatea filtrului de puncte cheie. Cu cât este mai mare cu atât este mai improbabil ca zgromotul imagini să fie detectat ca un fals pozitiv.

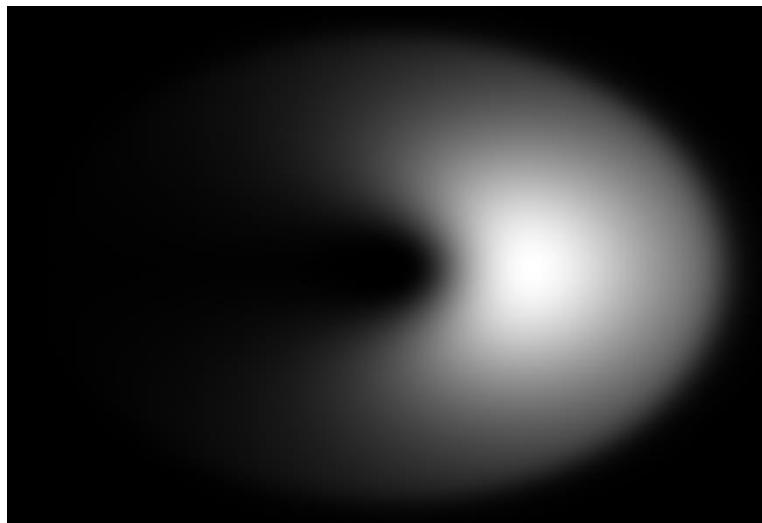
Folosind acești parametrii vom obține următoarele filtre:

- Filtrul 1: orientare: 0, scala: 3, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 2: orientare: 0, scala: 5.12993, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 3: orientare: 0, scala: 8.77205, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 4: orientare: 0, scala: 15, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 5: orientare: 0.523599, scala: 3, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 6: orientare: 0.523599, scala: 5.12993, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 7: orientare: 0.523599, scala: 8.77205, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 8: orientare: 0.523599, scala: 15, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 9: orientare: 1.0472, scala: 3, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 10: orientare: 1.0472, scala: 5.12993, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 11: orientare: 1.0472, scala: 8.77205, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 12: orientare: 1.0472, scala: 15, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 13: orientare: 1.5708, scala: 3, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 14: orientare: 1.5708, scala: 5.12993, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 15: orientare: 1.5708, scala: 8.77205, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detecție : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.

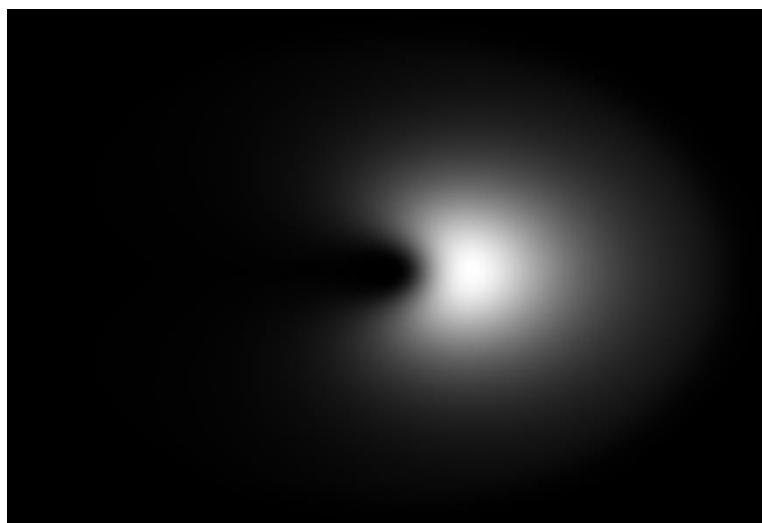
- Filtrul 16: orientare: 1.5708, scala: 15, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detectie : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 17: orientare: 2.0944, scala: 3, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detectie : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 18: orientare: 2.0944, scala: 5.12993, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detectie : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 19: orientare: 2.0944, scala: 8.77205, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detectie : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 20: orientare: 2.0944, scala: 15, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detectie : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 21: orientare: 2.61799, scala: 3, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detectie : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 22: orientare: 2.61799, scala: 5.12993, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detectie : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 23: orientare: 2.61799, scala: 8.77205, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detectie : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.
- Filtrul 24: orientare: 2.61799, scala: 15, raportul dintre deviație și frecvență: 0.55, deviația standard : 1, limita de detectie : 0.7, frecvența de tăiere : 0.45, ascuțimea: 15.



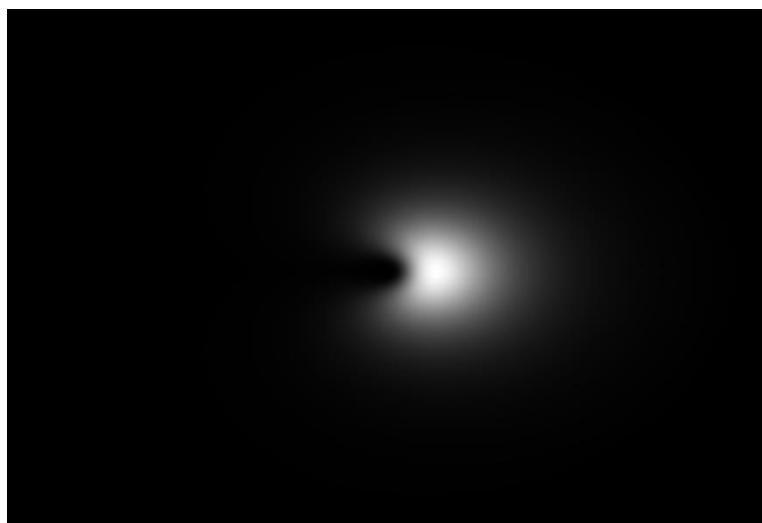
Figură 6.1 : Filtrul 1 Log-Gabor



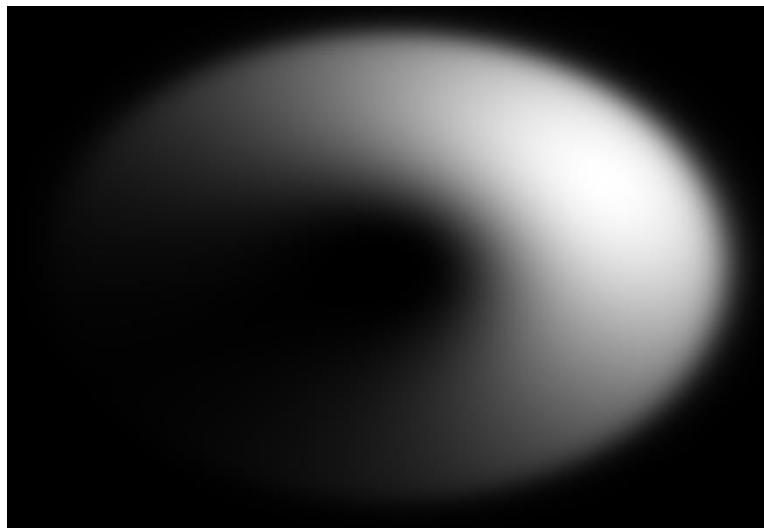
Figură 6.2 : Filtrul 2 Log-Gabor



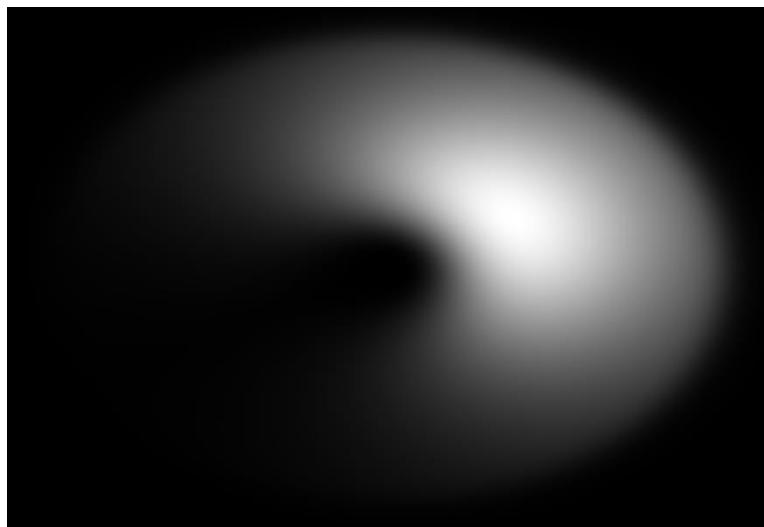
Figură 6.3 : Filtrul 3 Log-Gabor



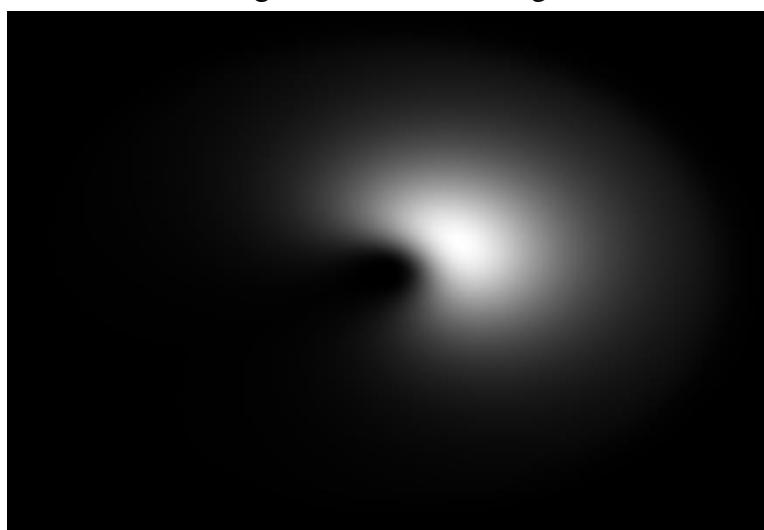
Figură 6.4 : Filtrul 4 Log-Gabor



Figură 6.5 : Filtrul 5 Log-Gabor



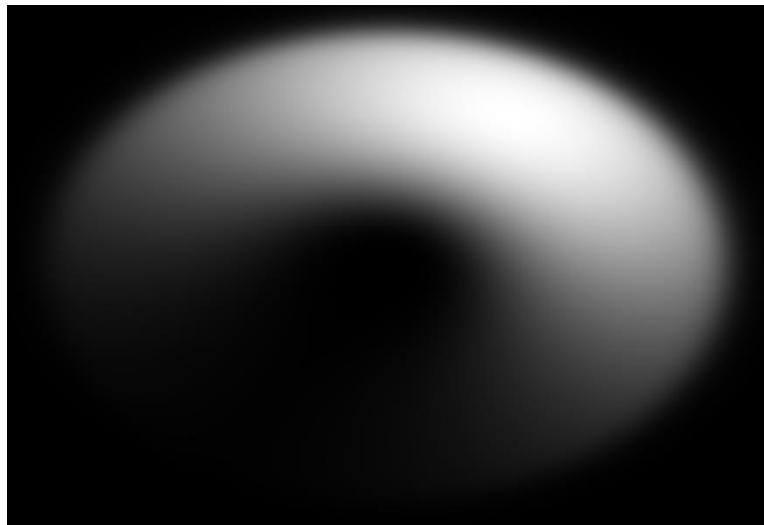
Figură 6.6 : Filtrul 6 Log-Gabor



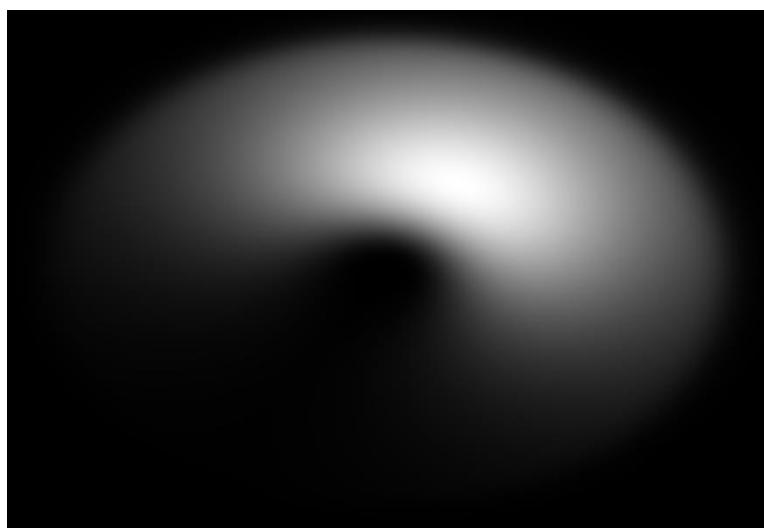
Figură 6.7 : Filtrul 7 Log-Gabor



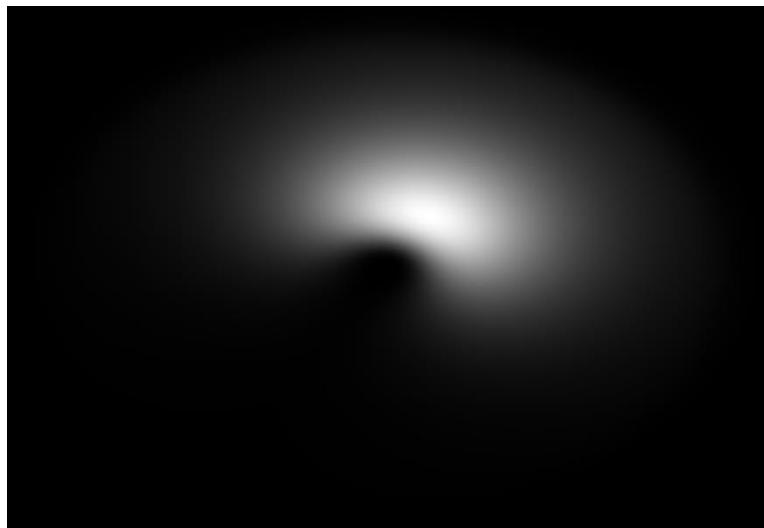
Figură 6.8 : Filtrul 8 Log-Gabor



Figură 6.9 : Filtrul 9 Log-Gabor



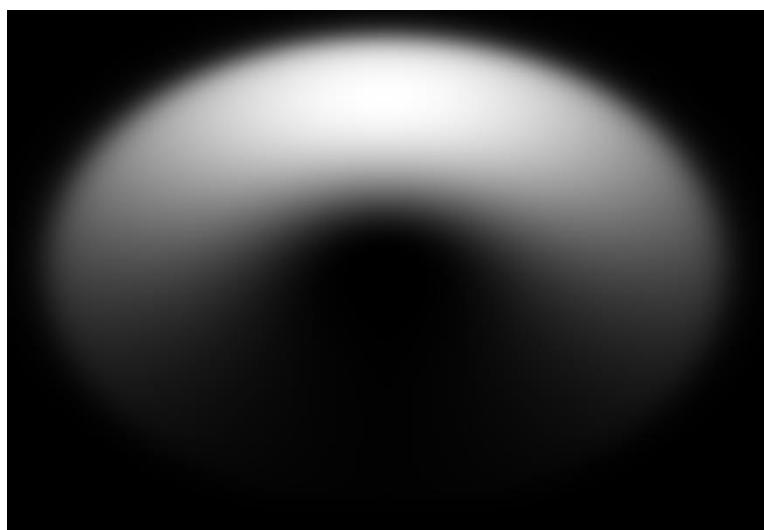
Figură 6.10 : Filtrul 10 Log-Gabor



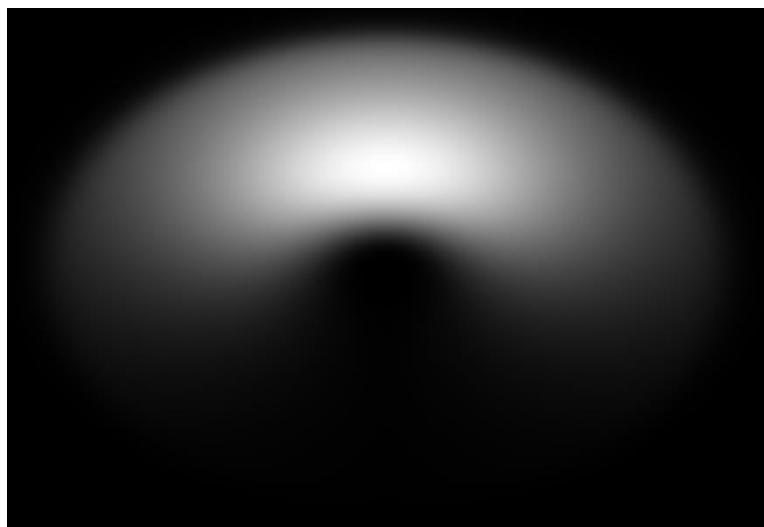
Figură 6.11 : Filtrul 11 Log-Gabor



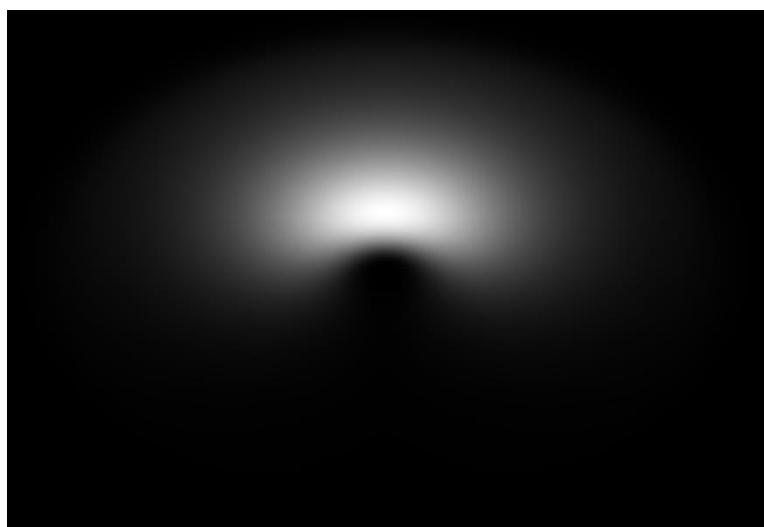
Figură 6.12 : Filtrul 12 Log-Gabor



Figură 6.13 : Filtrul 13 Log-Gabor



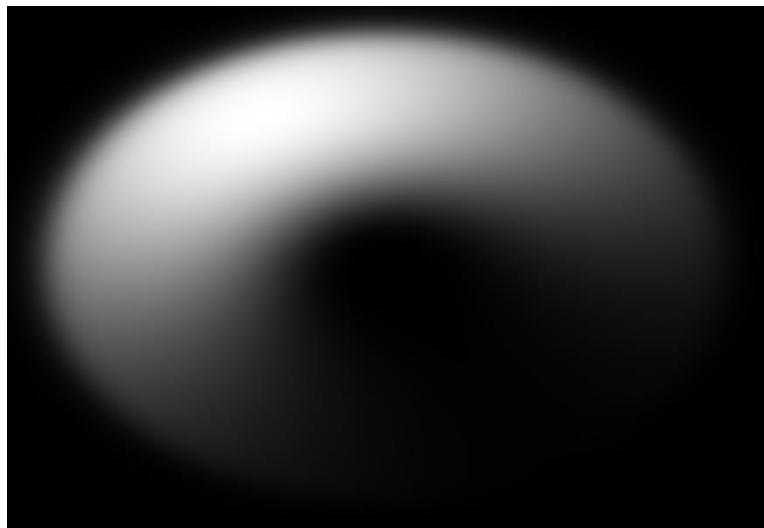
Figură 6.14 : Filtrul 14 Log-Gabor



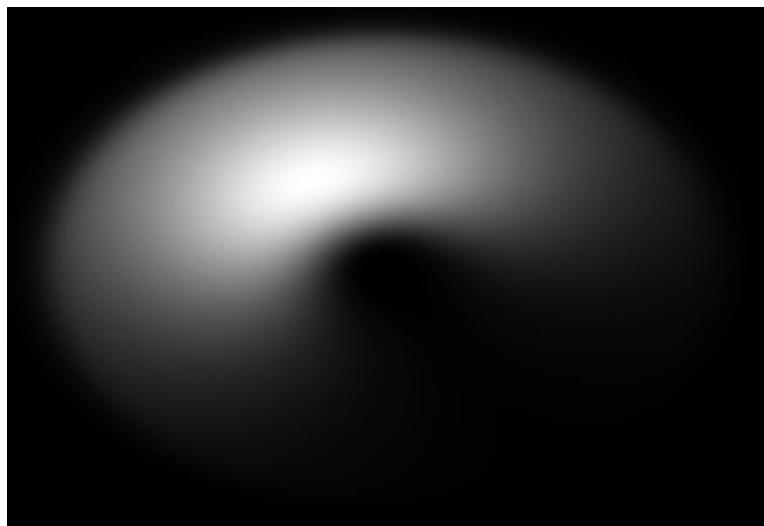
Figură 6.15 : Filtrul 15 Log-Gabor



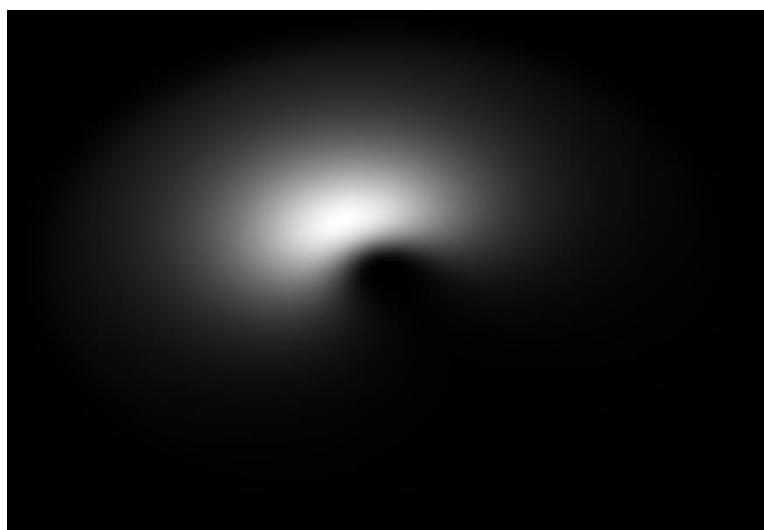
Figură 6.16 : Filtrul 16 Log-Gabor



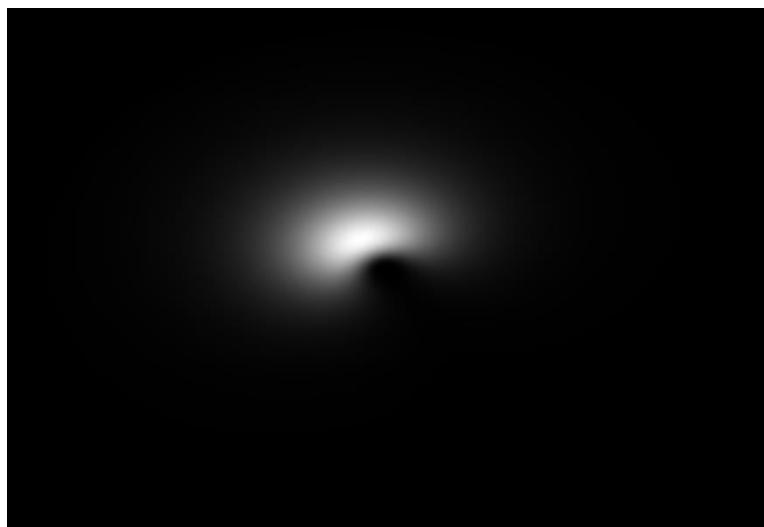
Figură 6.17 : Filtrul 17 Log-Gabor



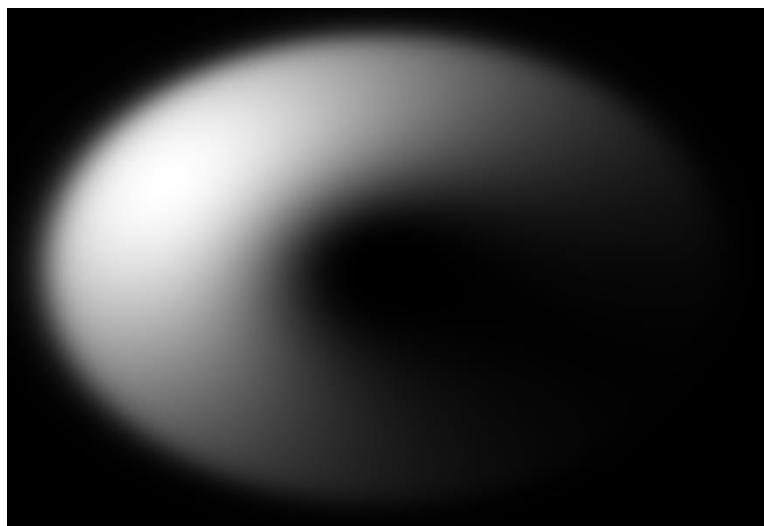
Figură 6.18 : Filtrul 18 Log-Gabor



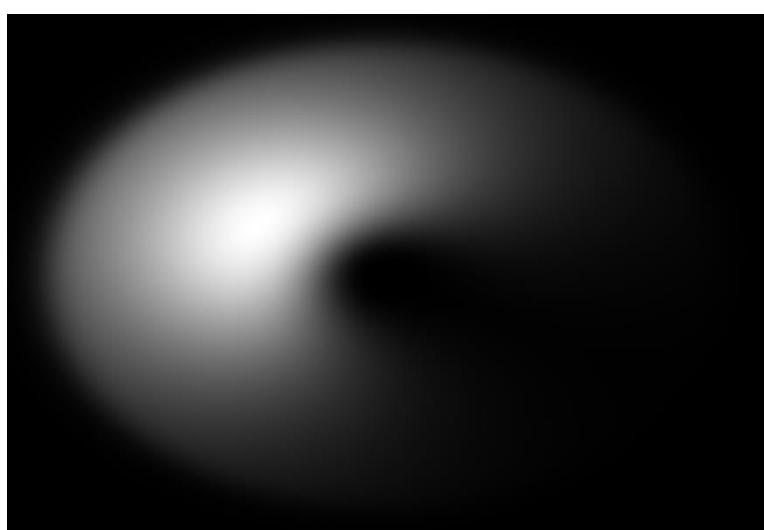
Figură 6.19 : Filtrul 19 Log-Gabor



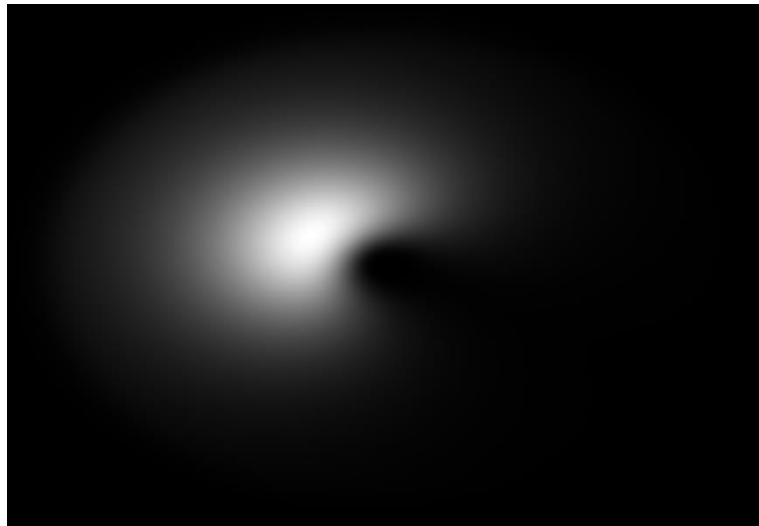
Figură 6.20 : Filtrul 20 Log-Gabor



Figură 6.21 : Filtrul 21 Log-Gabor



Figură 6.22 : Filtrul 22 Log-Gabor



Figură 6.23 : Filtrul 23 Log-Gabor



Figură 6.24 : Filtrul 24 Log-Gabor

### 6.3. Testarea detectorilor

Prima operație semnificativă după citirea și conversia imaginilor în format alb-negru este detectarea punctelor cheie. Pentru următoarele imagini se va urmări procesarea figurilor 6.26 și 6.25 ce au fost preluate din baza de date [16].



Figură 6.26 : DSC\_0295.jpg



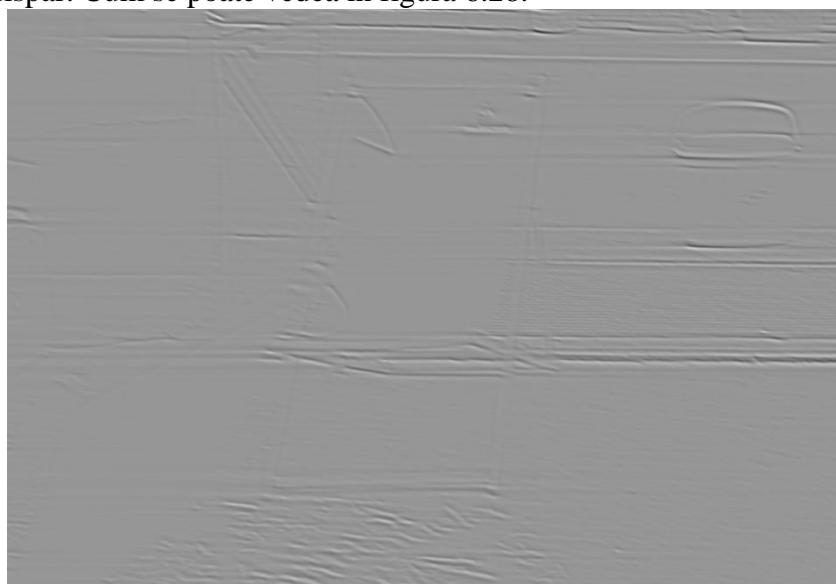
Figură 6.25 : DSC\_0296.jpg

În cadrul detectorului Log-Gabor, deoarece acesta este implementat explicit să putut face și salvarea imaginilor peste care s-a aplicat un filtru Log-Gabor. Astfel pentru imaginea din figura 6.26 la aplicarea filtrului 3 Log-Gabor din figura 6.3 avem ca rezultat figura 6.27 . În aceasta se poate observa că s-au păstrat și evidențiat toate liniile verticale pe când cele orizontale au fost suprimate.



Figură 6.27 : DSC\_0295 cu filtrul 3 Log-Gabor

În comparație la aplicarea filtrului 24 Log-Gabor asupra aceleiași imagini vom avea rezultatul invers, în care trăsăturile orizontale sunt evidențiate pe când cele verticale dispar. Cum se poate vedea în figura 6.28.

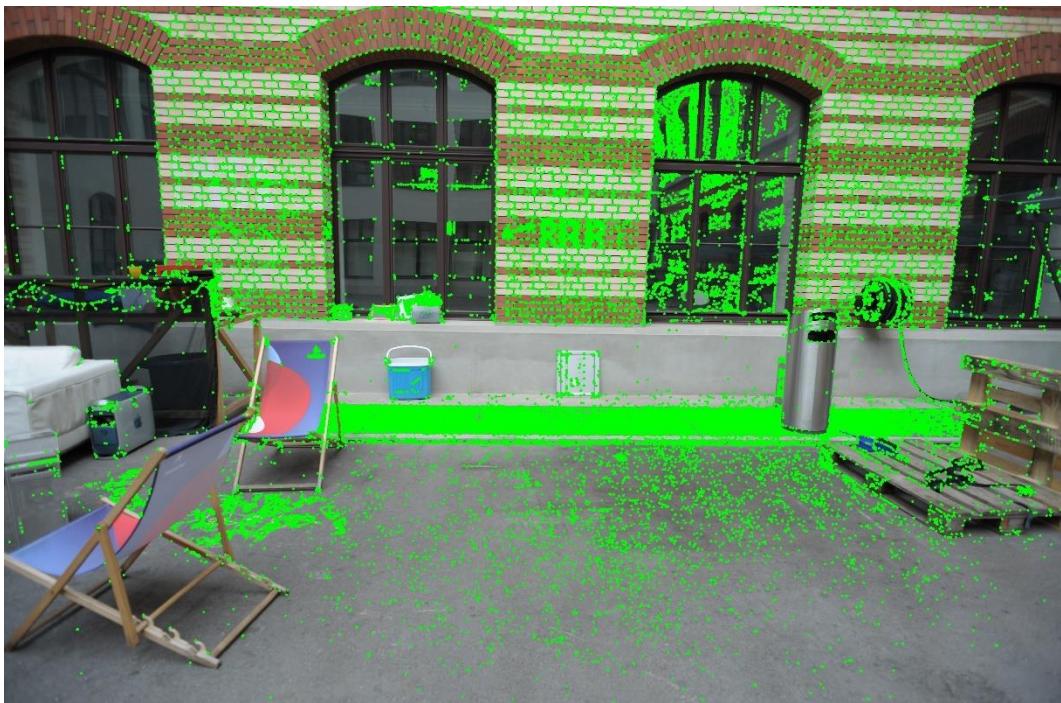


Figură 6.28 : DSC\_0295 cu filtrul 24 Log-Gabor

Figurile 6.27 și 6.28 sunt mărite pe poziția scaunului pentru a evidenția mai bine impactul semnificativ pe care îl are folosirea mai multor filtre Log-Gabor de orientări și lungimi de undă diferite. Astfel prin folosirea tuturor 24 de filtre vom obține un set complet de puncte cheie ce au posibilitatea de a reprezenta trăsăturile necesare lipirii imaginilor. Ceilalți detectori sunt implementați folosind biblioteca OpenCV și nu permit vizualizări parțiale.

Toți detectorii au scopul de a returna un set de puncte cheie ce vor fi analizate de descriptori. Teoretic este nevoie de minim doar patru puncte calitative, dar nu există nicio garanție ca primele patru puncte cheie să respecte criteriul de calitate aşa că se vor căuta toate punctele posibile. Acestea pot fi și în numărul sutelor de mii, în unele cazuri, aşa că ar fi mult prea costisitoare în timp vizualizarea lor individuală. Ce se poate constata la o observație vizuală este numărul acestora și distribuția:

- Detectorul FAST:
  - Puncte cheie în DSC\_0295 : 44880
  - Puncte cheie în DSC\_0296 : 38226



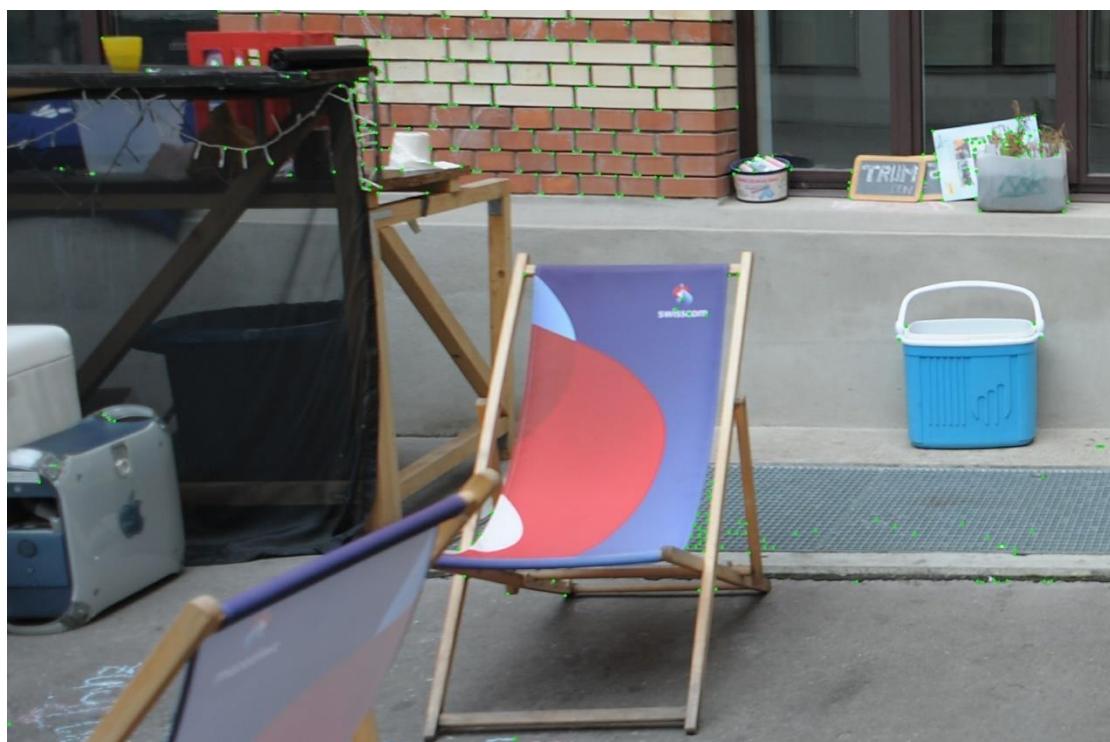
Figură 6.30 : Puncte cheie folosind detectorul FAST pe DSC\_0295



Figură 6.29 : Puncte cheie folosind detectorul FAST pe DSC\_0296

Se poate vedea că detectorul FAST este destul de sensibil acesta detectând grupuri dese de posibile puncte cheie. Acest fapt nu e necesar bun, punctele detectate pot face parte din elemente non-esențiale, iar numărul mare poate face ca împerecherea să nu funcționeze corect.

- Detectorul de colțuri Harris:
  - Puncte cheie în DSC\_0295 : 2250
  - Puncte cheie în DSC\_0296 : 1766



Figură 6.32 : Puncte cheie folosind detectorul Harris pe DSC\_0295, mărită



Figură 6.31 : Puncte cheie folosind detectorul Harris pe DSC\_0296, mărită

Detectorul de colțuri Harris oferă mai puține puncte dar păstrează distribuția acestora.

- Detectorul cu filtre Log-Gabor:
  - Puncte cheie în DSC\_0295 : 7558
  - Puncte cheie în DSC\_0296 : 9417



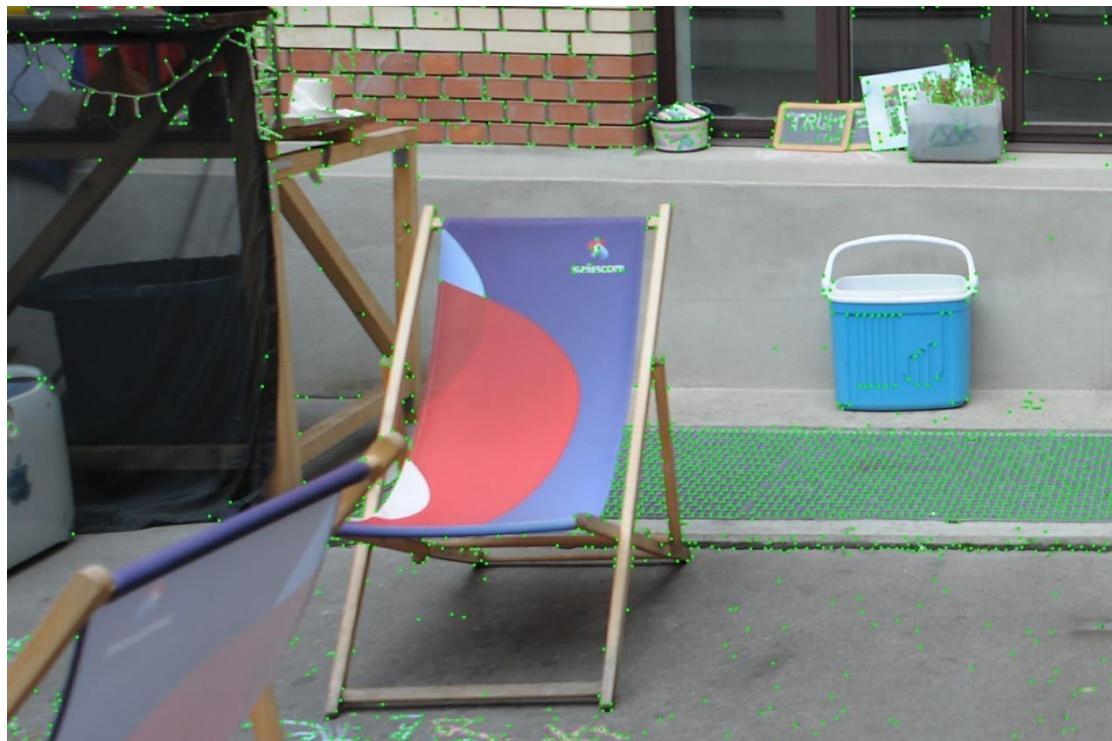
Figură 6.34 : Puncte cheie folosind detectorul Log-Gabor pe DSC\_0295, mărită



Figură 6.33 : Puncte cheie folosind detectorul Log-Gabor pe DSC\_0296, mărită

Detectorul bazat pe filtre Log-Gabor oferă un număr mare de puncte iar distribuția lor este comparabilă cu a detectorului Shi-Tomasi, deși nu este la fel de sensibil.

- Detectorul de colțuri Shi-Tomasi:
  - Puncte cheie în DSC\_0295 : 14524
  - Puncte cheie în DSC\_0296 : 14327



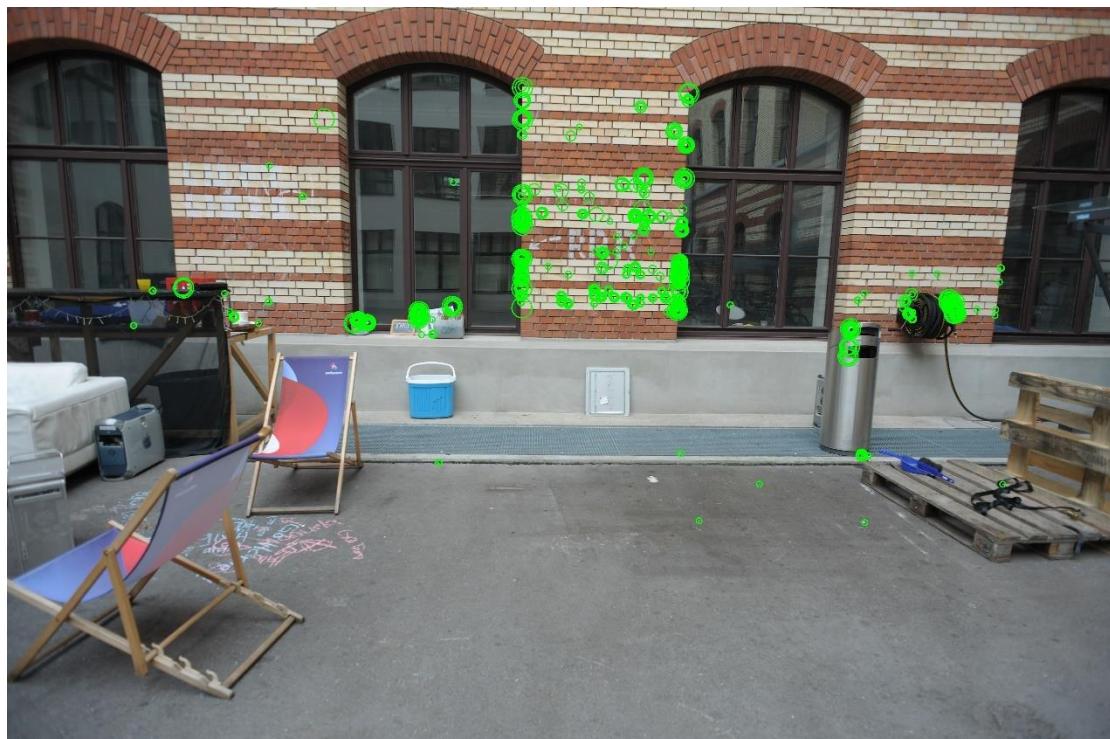
Figură 6.36 : Puncte cheie folosind detectorul Shi-Tomasi pe DSC\_0295, mărită



Figură 6.35 : Puncte cheie folosind detectorul Shi-Tomasi pe DSC\_0296, mărită

Detectorul Shi-Tomasi oferă rezultate comparativ mai bune decât cel dezvoltat de Harris, deși acestea au multiple elemente comune.

- Detectorul ORB:
  - Puncte cheie în DSC\_0295 : 500 (atinge o limită predefinită în cadrul bibliotecii)
  - Puncte cheie în DSC\_0296 : 500 (atinge o limită predefinită în cadrul bibliotecii)



Figură 6.38 : Puncte cheie folosind detectorul ORB pe DSC\_0295



Figură 6.37 : Puncte cheie folosind detectorul ORB pe DSC\_0296

- Detectorul SIFT:
  - Puncte cheie în DSC\_0295 : 24457
  - Puncte cheie în DSC\_0296 : 24259



Figură 6.40 : Puncte cheie folosind detectorul SIFT pe DSC\_0295



Figură 6.39 : Puncte cheie folosind detectorul SIFT pe DSC\_0296

## **6.4. Testarea descriptorilor**

Descriptorii au rolul de a lua datele punctelor cheie și de a oferi informații despre modelele reprezentate de acestea. Numărul descriptorilor nu va fi egal cu cel a punctelor cheie deoarece punctele pot fi false pozitive, provenite din zgromotul de imagine sau să fie foarte îngămădite și astfel o mulțime de puncte vor reprezenta un singur model. Deși nu există o metodă bună de validare sau verificare a acestora se poate evalua calitatea fiecărui pe baza eficienței și calității împerecherilor și a lipirii în pașii ulterioiri.

Datele rezultate rulării tuturor seturilor valide de algoritmi pentru imaginile din figurile 6.25 și 6.26 sunt:

- Detector FAST descris cu BRIEF:
  - Număr de descriptori pentru DSC\_0295 : 44547
  - Număr de descriptori pentru DSC\_0296 : 37954
- Detector FAST descris cu FREAK:
  - Număr de descriptori pentru DSC\_0295 : 44600
  - Număr de descriptori pentru DSC\_0296 : 38006
- Detector FAST descris cu ORB:
  - Număr de descriptori pentru DSC\_0295 : 44510
  - Număr de descriptori pentru DSC\_0296 : 37923
- Detector Harris descris cu BRIEF:
  - Număr de descriptori pentru DSC\_0295 : 2237
  - Număr de descriptori pentru DSC\_0296 : 1764
- Detector Harris descris cu FREAK:
  - Număr de descriptori pentru DSC\_0295 : 2239
  - Număr de descriptori pentru DSC\_0296 : 1764
- Detector Harris descris cu ORB:
  - Număr de descriptori pentru DSC\_0295 : 2237
  - Număr de descriptori pentru DSC\_0296 : 1764
- Detector Log-Gabor descris cu BRIEF:
  - Număr de descriptori pentru DSC\_0295 : 7183
  - Număr de descriptori pentru DSC\_0296 : 8258
- Detector Log-Gabor descris cu FREAK:
  - Număr de descriptori pentru DSC\_0295 : 7190
  - Număr de descriptori pentru DSC\_0296 : 8260
- Detector Log-Gabor descris cu ORB:
  - Număr de descriptori pentru DSC\_0295 : 7179
  - Număr de descriptori pentru DSC\_0296 : 8257
- Detector ORB descris cu BRIEF:
  - Număr de descriptori pentru DSC\_0295 : 500
  - Număr de descriptori pentru DSC\_0296 : 500
- Detector ORB descris cu FREAK:
  - Număr de descriptori pentru DSC\_0295 : 500
  - Număr de descriptori pentru DSC\_0296 : 500
- Detector ORB descris cu ORB:
  - Număr de descriptori pentru DSC\_0295 : 500
  - Număr de descriptori pentru DSC\_0296 : 500
- Detector Shi-Tomasi descris cu BRIEF:
  - Număr de descriptori pentru DSC\_0295 : 14392
  - Număr de descriptori pentru DSC\_0296 : 14184

- Detector Shi-Tomasi descris cu FREAK:
  - Număr de descriptori pentru DSC\_0295 : 14409
  - Număr de descriptori pentru DSC\_0296 : 1420
- Detector Shi-Tomasi descris cu ORB:
  - Număr de descriptori pentru DSC\_0295 : 14380
  - Număr de descriptori pentru DSC\_0296 : 14172
- Detector SIFT descris cu SIFT:
  - Număr de descriptori pentru DSC\_0295 : 24457
  - Număr de descriptori pentru DSC\_0296 : 24259

## **6.5. Testarea împerechelor**

Pentru testarea împerechelor din punct de vedere a calității dar și a eficienței s-au aplicat mai multe teste: testul de deviație a inlinerilor, testul erorii de proiectare și testul de lipire a imaginilor. Testele sunt definite în pachetul de evaluatori și în pachetul cusătoriilor de imagini (imageStitching).

În testul de deviație a inlinerilor se face raportul dintre inliner și numărul total de potriviri. Un inliner în acest context este definit ca o potrivire corectă și utilă în operația de lipire a imaginilor. Testul constă în crearea omografiei folosind algoritmul RANSAC. Acesta returnează lista potrivirilor ce au fost utile în operația de cusătură a imaginilor, cum se poate vedea în figura 6.41.

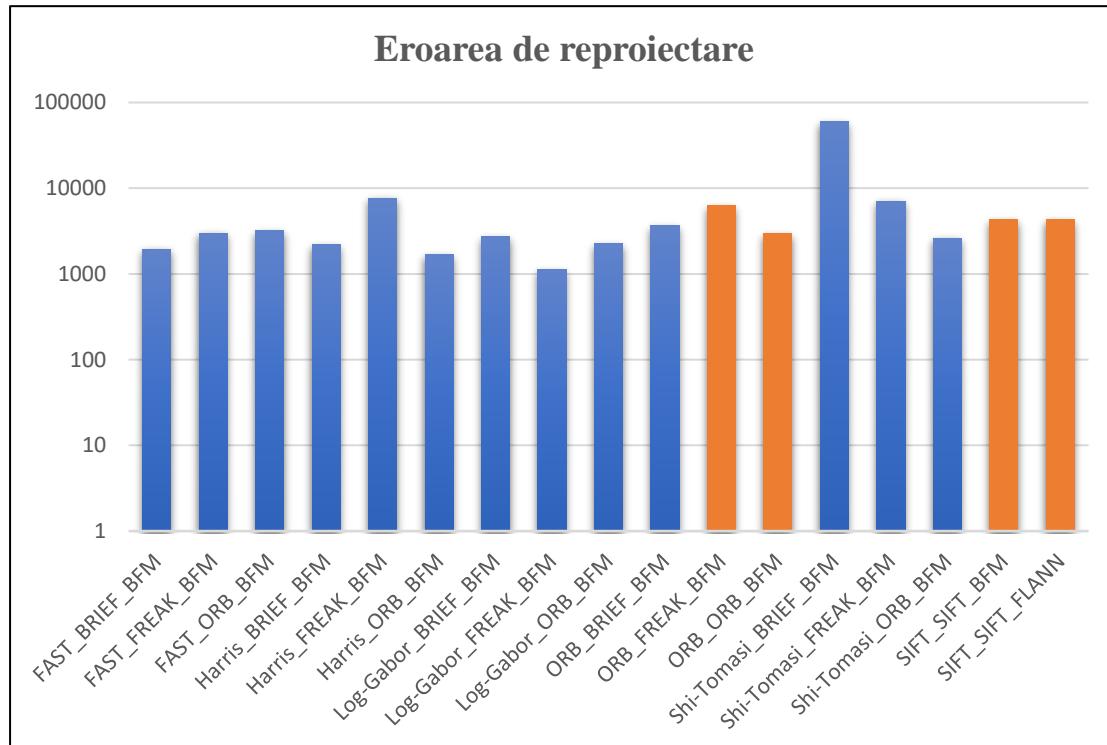
În testul erorii de proiectare se calculează eroarea medie a reproiectării. Aceasta este definită ca distanța de la punctul cheie potrivit în imagine și reprezentarea sa în omografie. Formulă: Reprojection Error = totalError / numMatches, unde totalError este suma distanțelor dintre punctele potrivite și cele proiectate, iar numMatches este numărul total de potriviri. A se vedea figura 6.42 cu graficul valorilor.

Pentru imaginile de la figurile 6.25 și 6.26 avem următoarele rezultate:

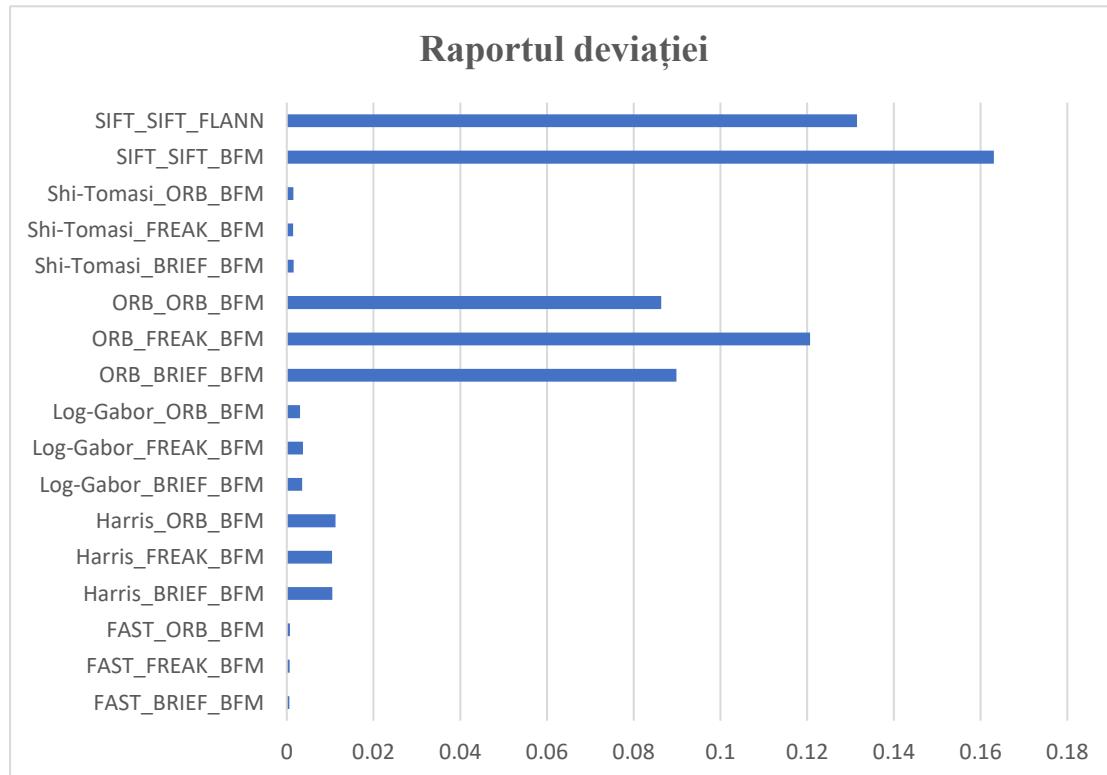
Detector	Descriptor	Împerechitor	Deviația	Eroarea de reproiectare
FAST	BRIEF	BFM	0.000610277	1943.04
FAST	FREAK	BFM	0.000653452	3000.87
FAST	ORB	BFM	0.000691802	3172.59
Harris	BRIEF	BFM	0.0104822	2213.64
Harris	FREAK	BFM	0.0104603	7620.16
Harris	ORB	BFM	0.0112108	1693.47
Log-Gabor	BRIEF	BFM	0.00353565	2720.04
Log-Gabor	FREAK	BFM	0.00375235	1120.51
Log-Gabor	ORB	BFM	0.00305623	2253.53
ORB	BRIEF	BFM	0.0898876	3649.01
ORB	FREAK	BFM	0.12069	6282.36
ORB	ORB	BFM	0.0863309	2968.5
Shi-Tomasi	BRIEF	BFM	0.00155715	60204.2
Shi-Tomasi	FREAK	BFM	0.00145433	7063.62
Shi-Tomasi	ORB	BFM	0.00151837	2619
SIFT	SIFT	BFM	0.163075	4288.07
SIFT	SIFT	FLANN	0.131484	4327.3

Tabel 6.1 : Testul de deviație și eroare de reproiectare

În tabelul 6.1 se poate vedea rezultatele pentru fiecare set de algoritmi. Observând valorile se poate face o anticipare la rata de succes a lipirii celor două. Se poate observa că cele mai bune șanse pentru o lipire corectă vor fi dacă se folosesc una din seturile de algoritmi: SIFT+SIFT+FLANN, SIFT+SIFT+BFM, ORB+FREAK+BFM, și ORB+ORB+BFM.



Figură 6.42 : Reprezentarea grafică a erori de reproiectare

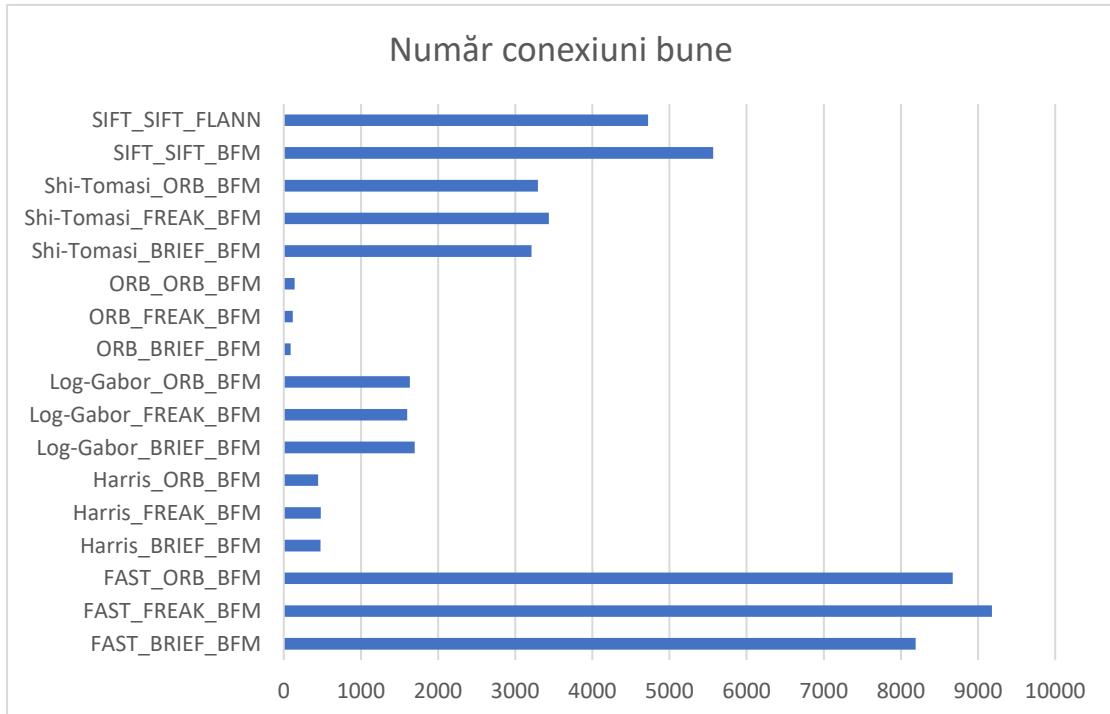


Figură 6.41 : Reprezentarea grafică a raportului deviației pentru fiecare set

Pentru confirmarea ipotezei avem rezultatele finale lipirii:

Detector	Descriptor	Împerechitor	Număr conexiuni bune	Timp (ms)	Lipire realizată cu succes
FAST	BRIEF	BFM	8193	34242	NU
FAST	FREAK	BFM	9182	62301	NU
FAST	ORB	BFM	8673	35522	NU
Harris	BRIEF	BFM	477	1344	NU
Harris	FREAK	BFM	478	1545	NU
Harris	ORB	BFM	446	1397	NU
Log-Gabor	BRIEF	BFM	1697	317850	NU
Log-Gabor	FREAK	BFM	1599	318151	NU
Log-Gabor	ORB	BFM	1636	316068	NU
ORB	BRIEF	BFM	89	498	PARTIAL
ORB	FREAK	BFM	116	637	PARTIAL
ORB	ORB	BFM	139	847	NU
Shi-Tomasi	BRIEF	BFM	3211	19597	NU
Shi-Tomasi	FREAK	BFM	3438	8757	NU
Shi-Tomasi	ORB	BFM	3293	5276	NU
SIFT	SIFT	BFM	5568	33366	DA
SIFT	SIFT	FLANN	4723	15011	DA

Tabel 6.2 : Rezultatul cusăturii



Figură 6.43 : Graficul algoritmilor în funcție de numărul de conexiuni stabile

## *Capitolul 6. Testare și validare*

---

Următoarele imagini prezintă rezultatele individuale:



Figură 6.44 : Panorama FAST + ORB + BFM



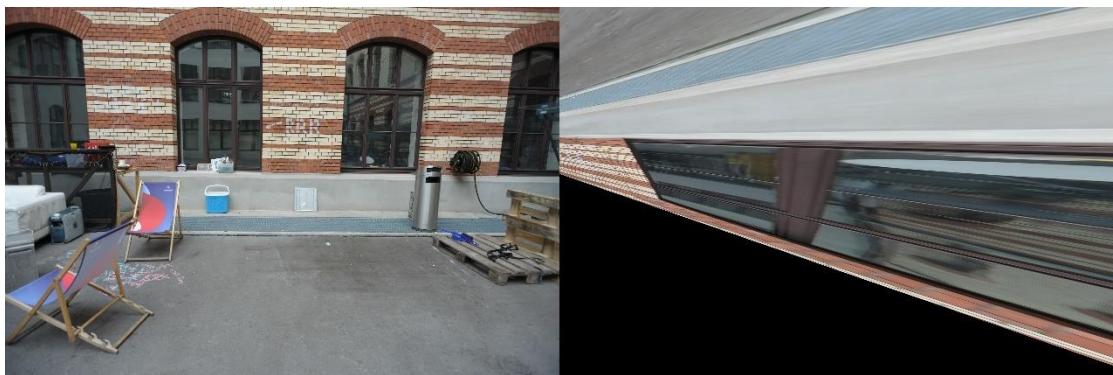
Figură 6.45 : Panorama Harris + BRIEF + BFM



Figură 6.46 : Panorama Harris + FREAK + BFM



Figură 6.47 : Panorama Harris + ORB + BFM



Figură 6.50 : Panorama Log-Gabor + BRIEF + BFM



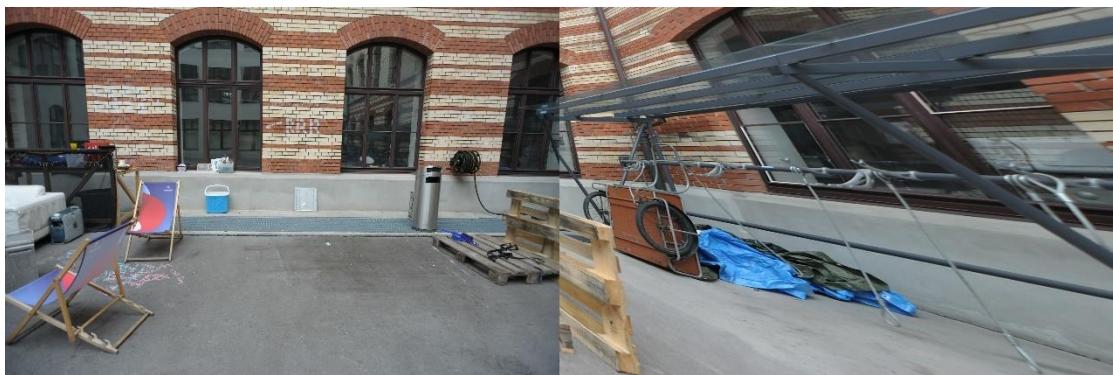
Figură 6.51 : Panorama Log-Gabor + FREAK + BFM



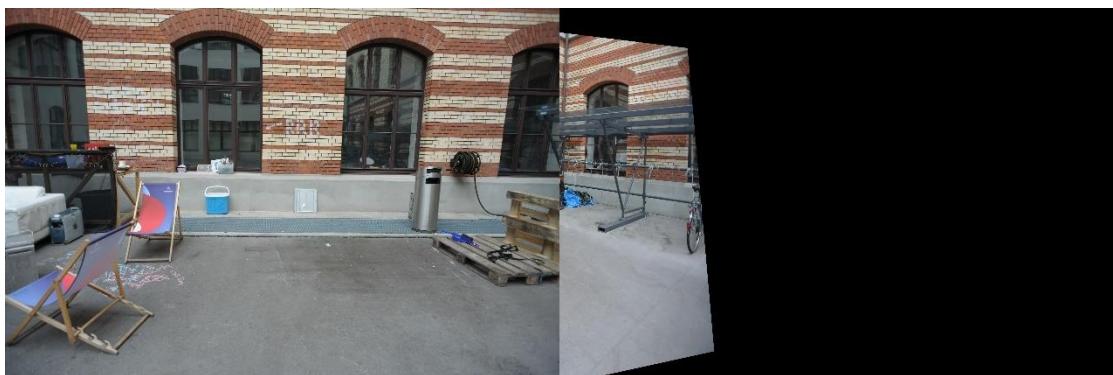
Figură 6.52 : Panorama Log-Gabor + ORB + BFM



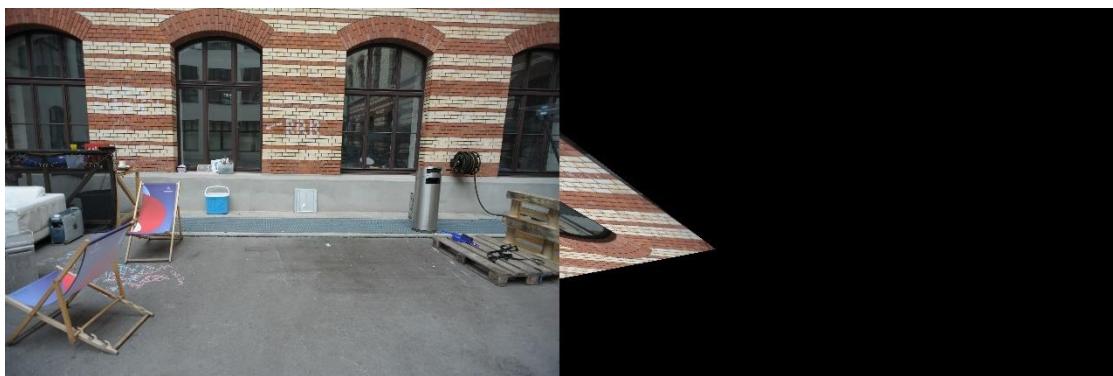
Figură 6.53 : Panorama ORB + BRIEF + BFM



Figură 6.54 : Panorama ORB + FREAK + BFM



Figură 6.55 : Panorama ORB + ORB + BFM



Figură 6.56 : Panorama Shi-Tomasi + BRIEF + BFM



Figură 6.57 : Panorama Shi-Tomasi + FREAK + BFM



Figură 6.58 : Panorama Shi-Tomasi + ORB + BFM



Figură 6.59 : Panorama SIFT + SIFT + BFM



Figură 6.60 : Panorama SIFT + SIFT + FLANN

## Capitolul 7. Manual de instalare si utilizare

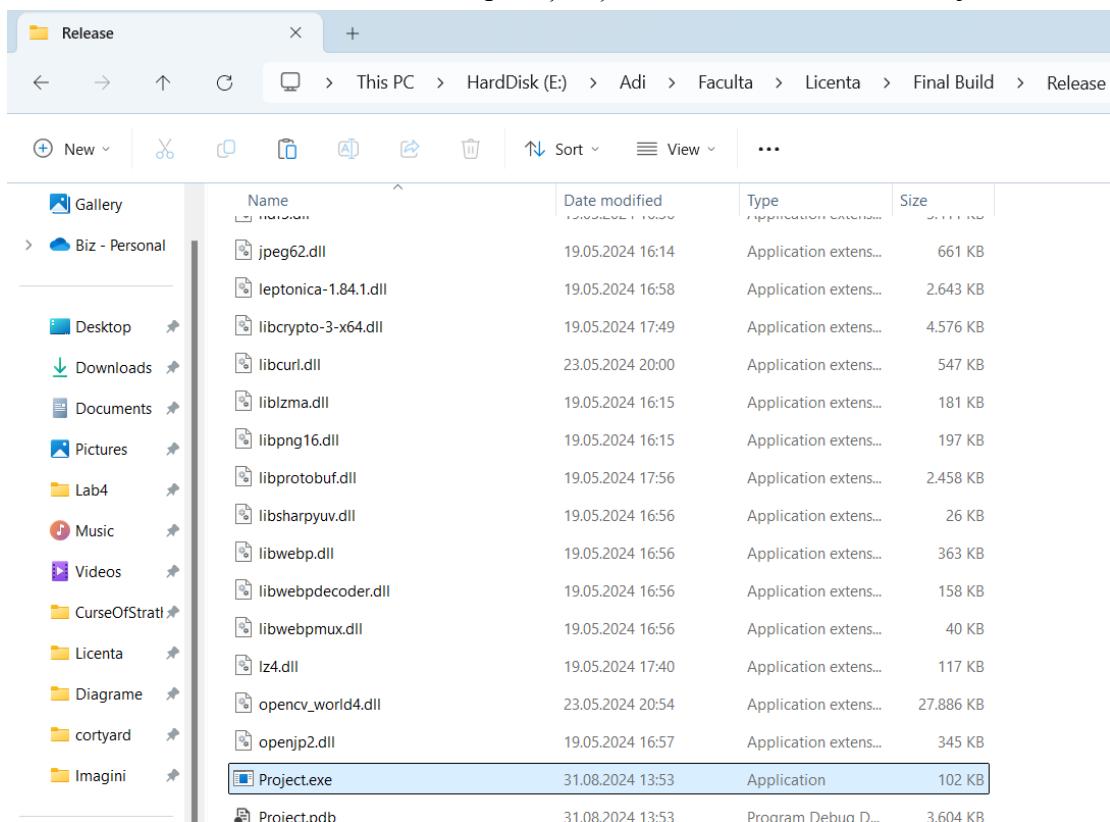
Pentru rularea aplicației aferente acestui proiect, utilizatorul trebuie să aibă instalate pe calculatorul personal sau pe care se face rularea următoarele:

- Windows 7 sau o versiune mai recentă.
- .NET Framework 3.5 sau mai recentă.
- Microsoft Visual C++ 2015-2022 Redistributable

În cazul în care fișierul aplicației se află într-o arhivă acesta trebuie despachetat, de preferat în propriul director.

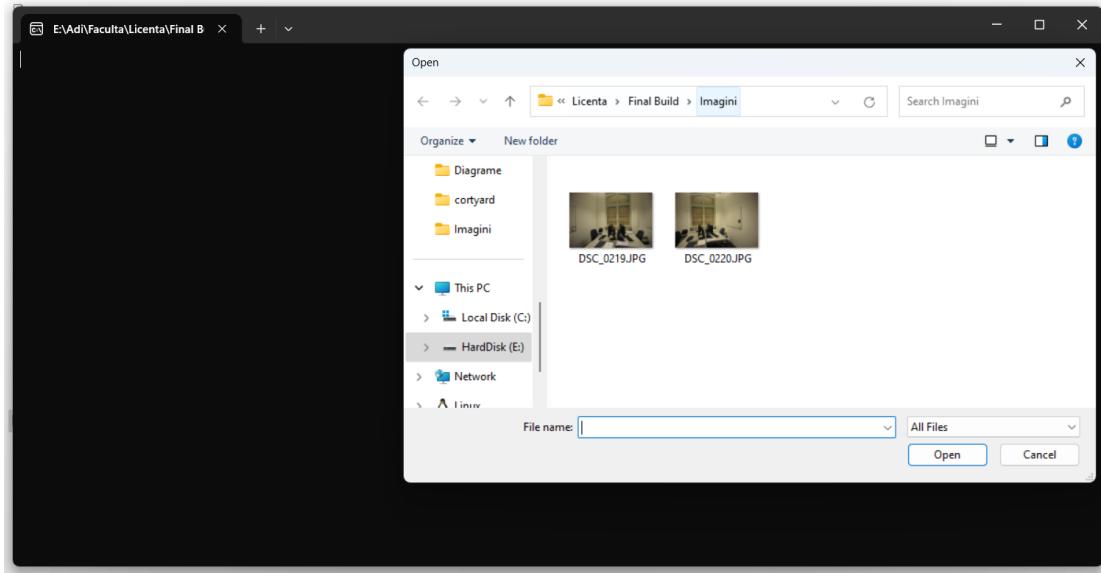
Pașii ce trebuie urmați pentru utilizare aplicației sunt următorii:

1. Se intră în directorul aplicației și se caută executabilul „Project.exe”.



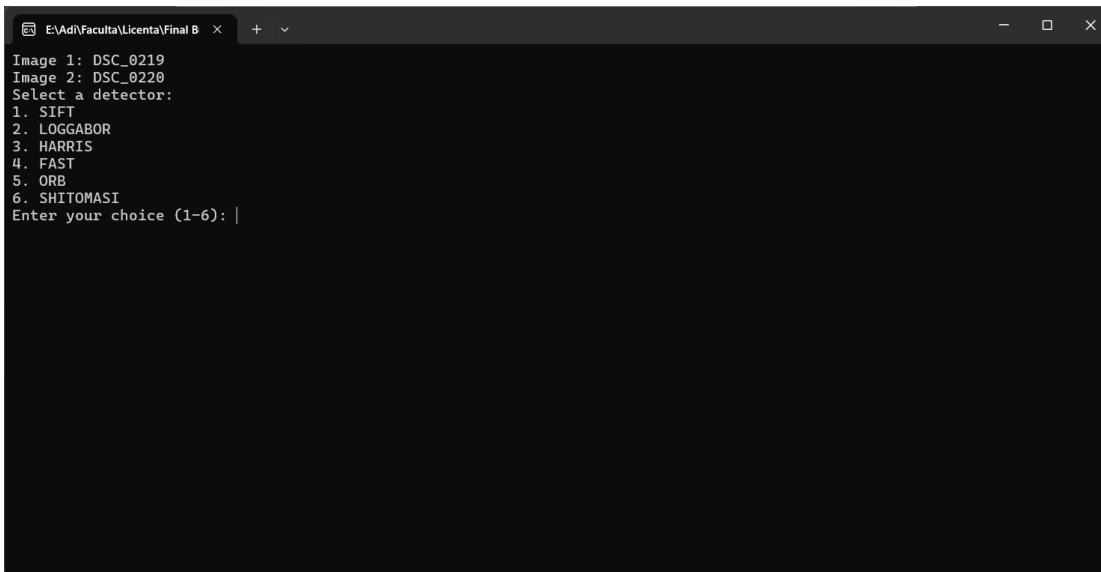
Figură 7.1 : Locația executabilului aplicației

2. Se pornește aplicația prin apăsarea rapidă a clicului din stânga de 2 ori sau prin selectarea acestuia cu un singur clic și apăsarea tastei „Enter”. Aplicația se va porni în linia de comandă și va deschide o fereastră de căutare în fișiere.
3. Utilizatorul va trebui să aleagă prima imagine ce va fi procesată, aceasta va fi imaginea din stânga când se va lipi cu a doua. După ce utilizatorul a ales prima imagine, o nouă fereastră de căutare se va deschide.
4. Utilizatorul va alege cea de a doua imagine ce va fi procesată, acesta va fi imaginea din dreapta și de asemenea cea ce va fi deformată când se va face lipirea imaginilor. După alegerea imaginilor aplicația va afișa o listă a detectoarelor de trăsături ce au fost implementate în cadrul aplicației.



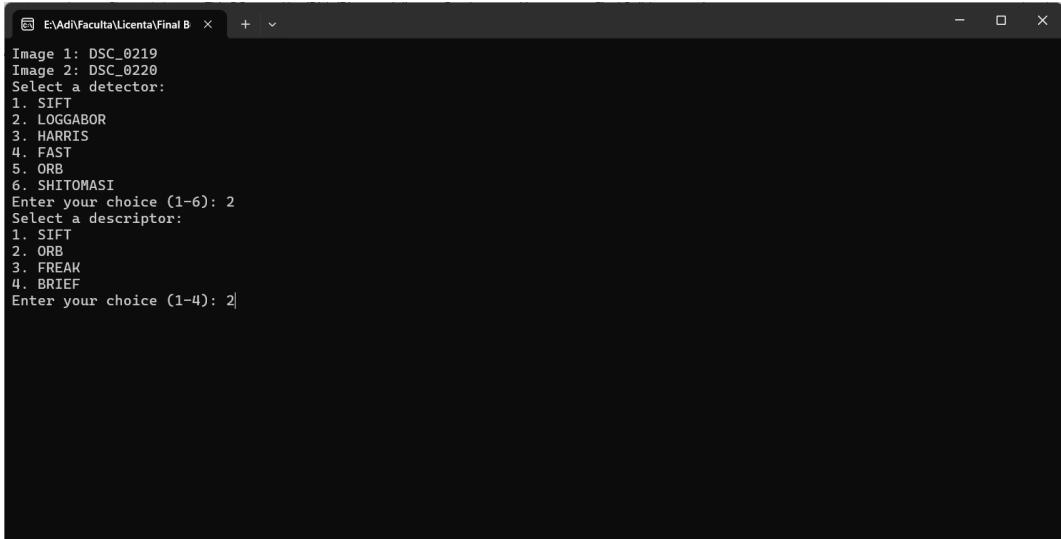
Figură 7.2 : Selectarea imaginilor de procesat

- Utilizatorul va alege detectorul dorit prin oferirea indicelui asociat acestuia. Odată ales utilizatorul va apăsa „Enter” și un nou set de algoritmi va fi afișat. Aceștia reprezintă algoritmii de descriere a trăsăturilor.

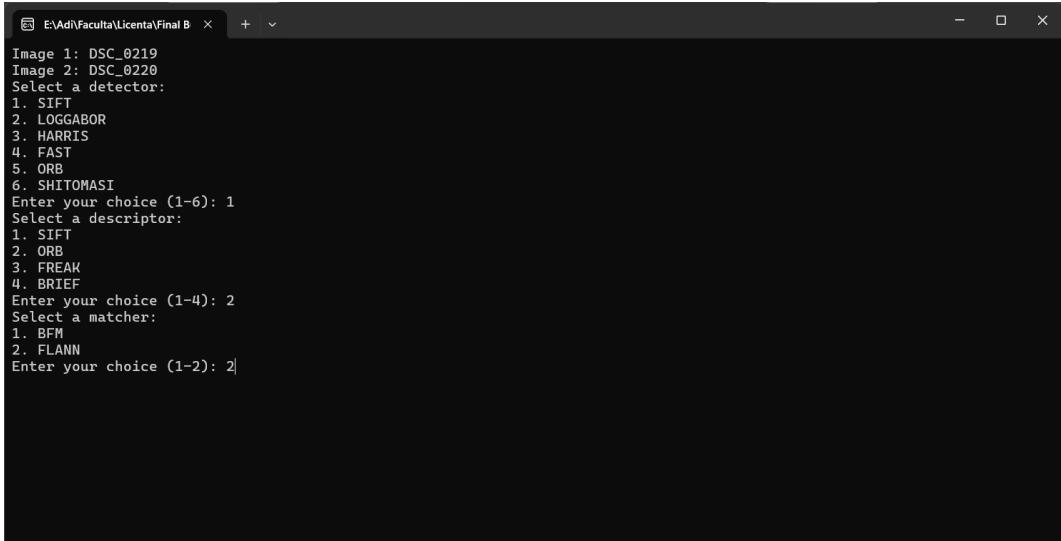


Figură 7.3 : Alegerea detectorului

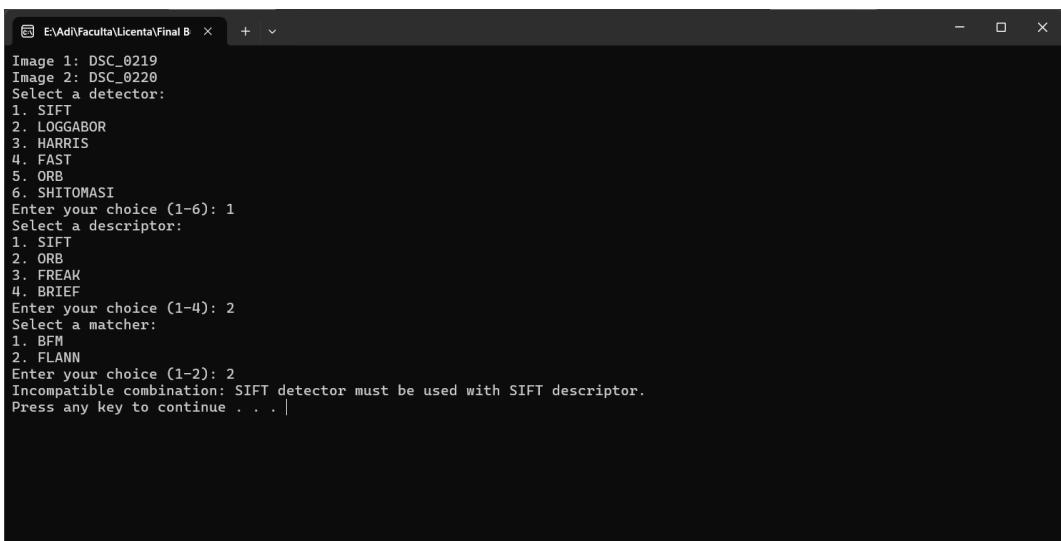
- Utilizatorul va oferi un nou indice ce reprezintă algoritmul de descriere dorit și va apăsa „Enter”. Aplicația va oferi un ultim set de algoritmi reprezentând opțiunile de împerechere a trăsăturilor.
- Utilizatorul va oferi indicele din stânga algoritmului de împerechere dorit și va apăsa „Enter”.
- Dacă setul selectat este o combinație invalidă aplicația va afișa un mesaj de eroare și va necesita apăsarea oricărei taste pentru a se închide. Dacă setul dorit este o combinație validă, atunci aplicația va afișa ca confirmare setul compus și va începe procesarea imaginilor conform acestuia.



Figură 7.6 : Alegerea descriptorului

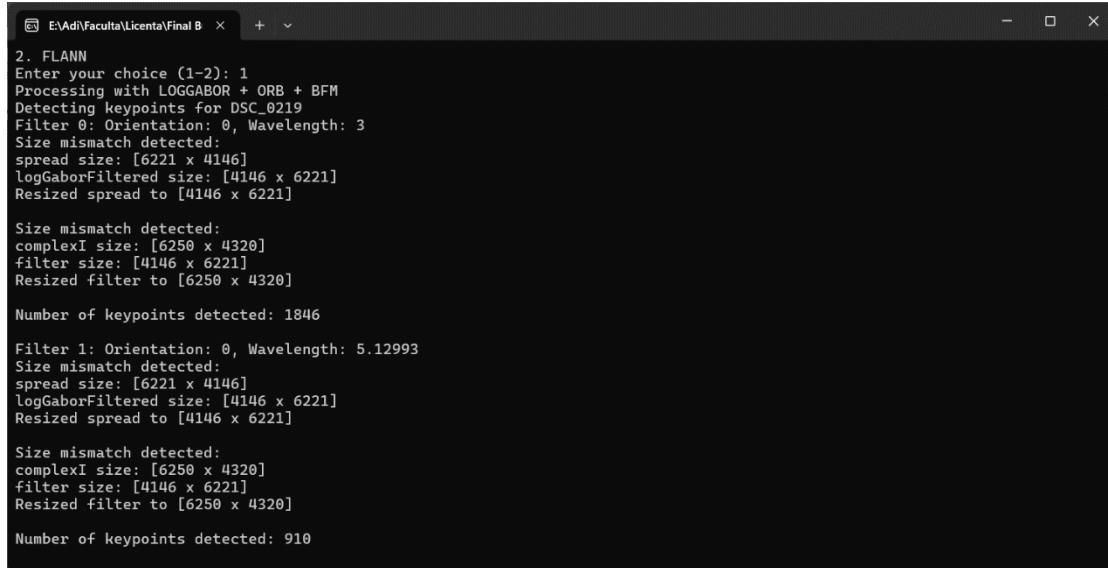


Figură 7.4 : Alegerea împerechelorilor



Figură 7.5 : Eroare algoritmi incompatibili

9. În cazul algoritmului Log-Gabor de detecție aplicația va putea oferi informații cu privire la progresul acesteia. Dacă nu este cazul atunci programul va afișa un mesajul “Press any key to continue...” și așteaptă apăsarea oricărei taste pentru a se închide. Rezultatele sunt salvate în directorul de unde au fost preluate imaginile procesate.



The screenshot shows a terminal window with the following text output:

```
E:\Adi\Faculta\Licenta\Final B > + ^

2. FLANN
Enter your choice (1-2): 1
Processing with LOGGABOR + ORB + BFM
Detecting keypoints for DSC_0219
Filter 0: Orientation: 0, Wavelength: 3
Size mismatch detected:
spread size: [6221 x 4146]
logGaborFiltered size: [4146 x 6221]
Resized spread to [4146 x 6221]

Size mismatch detected:
complexI size: [6250 x 4320]
filter size: [4146 x 6221]
Resized filter to [6250 x 4320]

Number of keypoints detected: 1846

Filter 1: Orientation: 0, Wavelength: 5.12993
Size mismatch detected:
spread size: [6221 x 4146]
logGaborFiltered size: [4146 x 6221]
Resized spread to [4146 x 6221]

Size mismatch detected:
complexI size: [6250 x 4320]
filter size: [4146 x 6221]
Resized filter to [6250 x 4320]

Number of keypoints detected: 910
```

Figură 7.7 : Procesarea Log-Gabor

## **Capitolul 8. Concluzii**

### **8.1. Contribuție personală**

În cadrul proiectului s-a dorit realizarea unui procedeu de lipire a imaginilor, dar și aducerea de îmbunătățiri a modului de procesare a acestora, pornind de la un articol ce aducea în lumină posibilitatea de a detecta trăsăturile unei imagini eficient și cu rezultate calitative, folosind filtrele Log-Gabor. Proiectul realizat încearcă să deschidă posibilitatea de dezvoltare și îmbunătățire a cestei idei prin implementarea filtrelor și în limbajul de programare C++ ce are avantajele de a putea dezvolta aplicații în care memoria și datele sunt manipulate în detaliu, dar și posibilitatea de a converti codul realizat într-un format compatibil cu programarea de procesoare. Astfel deși codul realizat nu este eficient în timp, optimizat sau cu rezultate calitative acesta nu-și pierde potențialul ci doar necesită mai multă muncă și cunoștințe mai avansate decât cele acaparate în prezent.

Articolul de la care s-a început acest proiect este [15]. Acesta oferă o modalitate de a implementa un detector eficient a trăsăturilor unei imagini folosind mediul de dezvoltare Python și Matlab. Aceste medii de dezvoltare oferă funcționalități dedicate procesării imaginilor și soluții ușoare la calculele complexe, dar nu neapărat eficiente în memorie sau rapide, așa că idea inițială este de a oferi o implementare a filtrului Log-Gabor și în limbajul C++. Aceasta are rolul de a servi ca un punct de plecare pentru viitoarele dezvoltări și îmbunătățiri ce necesită manipularea memoriei și eficiență în timp. Totuși algoritmul rezultat se poate considera mai mult decât o simplă copie a originalului, soluția originală a suferind diferite modificări pentru a obține o variată funcțională în mediul de dezvoltare Visual Studio și folosind limbajul C++.

Un prim impediment ce necesită o diferență interpretare a filtrelor Log-Gabor este calculul componentei unghiulare. Formula inițială și descrisă în articolul [15] necesită calcule pentru matrice definite în domeniul numerelor complexe, calcule ce nu sunt suportate de funcționalitățile limbajului C++ și nici a unei biblioteci adiționale. Așa că folosirea unei alternative este necesară pentru evitarea calculelor de matrice în domeniul numerelor complexe.

O altă adaptare constă în funcția de verificare și redimensionare. Algoritmul necesită aplicarea transformatei Fourier și aplicarea de filtre compuse în domeniu frecvențial. Asemenea calcule pot oferi valori în afara matricei sau din cauza unei borduri de siguranță matricele să nu mai fie compatibile. Așa că la fiecare înmulțire se fac verificări și dacă matricele nu sunt compatibile acestea sunt redimensionate la mărimea corespunzătoare.

### **8.2. Realizarea obiectivelor**

Implementare filtrelor Log-Gabor deși nu este ideală este funcțională și aplicația rezultată oferă informații esențiale pentru compararea acestor filtre cu algoritmii standard de detectie a trăsăturilor. Conform datelor prezentate în capitolul 6 această implementare filtrelor Log-Gabor este de departe de a fi varianta ideală. Totuși în ciuda defectelor evidente de calibrare și implementare aceasta oferă metrice mai bune decât anumiți algoritmi recunoscuți și care sunt încă folosiți. Astfel se poate considera că atât implementarea detectorului Log-Gabor cât și dezvoltarea aplicației și-au atins obiectivele.

### **8.3. Dezvoltări ulterioare**

Aplicația implementată se poate considera suficientă pentru scopul acestui proiect, dar este de departe perfectă aceasta având câteva laturi asupra cărora o dezvoltare ulterioară ar putea îmbunătăți:

- Calibrarea în detaliu a filtrelor și detectorilor sau implementarea unui modul ce ar asigura calibrarea automată a acestora.
- Îmbunătățirea algoritmilor de detecție prin implementarea de filtre adiționale ce să reducă din zgomotul de imagine sau a elementelor non relevante.
- Folosirea de metode mai avansate pentru lipirea imaginilor ca rezultatele finale să formeze treceți indistincție între imagini.
- Crearea unei interfețe utilizator ce ar permite salvarea imaginilor și a rezultatelor în mod elegant.
- Portarea într-un mediu cloud a aplicației pentru a asigura accesul mai multor utilizatori și a putea folosi resurse mult mai generoase.
- Implementare de descriptori dedicați pentru fiecare algoritmi.
- Crearea de teste adiționale pentru evaluarea algoritmilor. În cadrul modulului de evaluare există deja concepte de metriki ce nu au fost folosite din cauza necesității unui set de imagini cu trăsături predefinite.

Odată ce aceste îmbunătățiri au fost aduse potențialul aplicației se deschide înspre aplicări mai vaste. Cum ar fi:

- Crearea unui sistem de compunere a imaginilor panoramice sau de mărimi mari.
- Utilizarea funcțiilor de procesare de imagini pentru a obține reconstrucții 3D a obiectelor din cadrul imaginilor.
- Crearea unei aplicații eficiente în timp și memorie pentru recunoașterea persoanelor și a obiectelor.

## Bibliografie

- [1] C. H. și M. Stephens, „A Combined Corner and Edge Detector,” în *Proceedings of the 4th Alvey Vision Conference*, 1988.
- [2] J. S. și C. Tomasi, „Good Features to Track,” în *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1994.
- [3] D. G. Lowe, „Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, nr. 2, pp. 91-110, 2004.
- [4] E. a. R. V. a. K. K. a. B. G. Rublee, „ORB: An efficient alternative to SIFT or SURF,” în *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2011.
- [5] E. a. D. T. Rosten, „Machine learning for high-speed corner detection,” în *Proceedings of the European Conference on Computer Vision (ECCV)*, 2006.
- [6] A. a. O. R. a. V. P. Alahi, „FREAK: Fast Retina Keypoint,” în *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012.
- [7] M. L. V. S. C. & F. P. Calonder, „Brief: Binary robust independent elementary features,” în *European Conference on Computer Vision (ECCV)*, 2010.
- [8] R. a. Z. A. Hartley, *Multiple view geometry in computer vision*, Cambridge university press, 2003.
- [9] M. a. L. D. G. Muja, „Fast approximate nearest neighbors with automatic algorithm configuration,” în *International Conference on Computer Vision Theory and Applications*, 2009.
- [10] M. A. a. B. R. C. Fischler, „Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, nr. 6, pp. 381-395, 1981.
- [11] R. Szeliski, *Image Alignment and Stitching: A Tutorial*, Now Foundations and Trends, 2006.
- [12] J. G. Daugman, „Uncertainty Relation for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two-Dimensional Visual Cortical Filters,” *Journal of the Optical Society of America A*, vol. 2, nr. 7, pp. 1160-1169, 1985.
- [13] F. DJ, „Relations Between the Statistics of Natural Images and the Response Properties of Cortical Cells,” *Journal of the Optical Society of America A*, vol. 4, nr. 12, pp. 2379-2394, 1987.
- [14] P. Kovesi, „What Are Log-Gabor Filters and Why Are They Good?,” [Interactiv]. Available: <https://peterkovesi.com/matlabfns/PhaseCongruency/Docs/convexpl.html>.
- [15] D. B. B. Tze Kian Jong, „An effective feature detection approach for image stitching of near-uniform scenes,” *Signal Processing: Image Communication*, vol. 110, p. 116872, 2023.
- [16] C. V. a. G. Group, „ETH3D,” ETH Zurich, [Interactiv]. Available: <https://www.eth3d.net/datasets#high-res-multi-view>.

## Anexa 1 – Aplicarea filtrelor Log-Gabor cod

```
Mat applyLogGaborFilter(Mat& src_gray, double angl, double sig_fs, double lam,
double theta_o, double threshold,
float cutoff, float sharpness, const string& imageName, int id) {
int rows = src_gray.rows;
int cols = src_gray.cols;
Mat radius = createNormalizedRadius(rows, cols);
Mat logGabor = createLogGaborFilter(radius, lam, sig_fs);

Mat lowPassFilter = createLowPassFilter(radius, cutoff, sharpness);

Mat logGaborFiltered;
checkAndResizeIfNeeded(logGabor, lowPassFilter, "logGabor", "lowPassFilter");
multiply(logGabor, lowPassFilter, logGaborFiltered);

Mat spread = createAngularComponent(cols, rows, angl, theta_o);

Mat filter;
checkAndResizeIfNeeded(spread, logGaborFiltered, "spread", "logGaborFiltered");
multiply(spread, logGaborFiltered, filter);

Mat padded;
int m = getOptimalDFTSize(rows);
int n = getOptimalDFTSize(cols);
copyMakeBorder(src_gray, padded, 0, m - rows, 0, n - cols,
BORDER_CONSTANT, Scalar::all(0));

Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
Mat complexI;
merge(planes, 2, complexI);

dft(complexI, complexI);
fftShift(complexI, complexI);
checkAndResizeIfNeeded(complexI, filter, "complexI", "filter");
applyFilterManually(complexI, filter);
fftShift(complexI, complexI);

Mat imgf;
idft(complexI, imgf, DFT_REAL_OUTPUT);
Mat response;
imgf.convertTo(response, CV_32F);
normalize(response, response, 0, 1, NORM_MINMAX);
return response;
```

## Anexa 2 – Lista figurilor

Figură 4.1: Cazurile de utilizare.....	15
Figură 4.2: Cazul de utilizare .....	16
Figură 5.1: Arhitectura aplicației .....	21
Figură 5.2: Diagrama de secvență.....	22
Figură 5.3: Pachetul controlorului .....	23
Figură 5.4: Pachetul evaluatorului .....	24
Figură 5.5: Pachetul cusătoriilor .....	25
Figură 5.6: Pachetul descriptorilor.....	26
Figură 5.7: Pachetul împerecheterului de trăsături .....	27
Figură 5.8: Pachetul detectorilor.....	28
Figură 6.1 : Filtrul 1 Log-Gabor .....	35
Figură 6.2 : Filtrul 2 Log-Gabor .....	36
Figură 6.3 : Filtrul 3 Log-Gabor .....	36
Figură 6.4 : Filtrul 4 Log-Gabor .....	36
Figură 6.5 : Filtrul 5 Log-Gabor .....	37
Figură 6.6 : Filtrul 6 Log-Gabor .....	37
Figură 6.7 : Filtrul 7 Log-Gabor .....	37
Figură 6.8 : Filtrul 8 Log-Gabor .....	38
Figură 6.9 : Filtrul 9 Log-Gabor .....	38
Figură 6.10 : Filtrul 10 Log-Gabor .....	38
Figură 6.11 : Filtrul 11 Log-Gabor .....	39
Figură 6.12 : Filtrul 12 Log-Gabor .....	39
Figură 6.13 : Filtrul 13 Log-Gabor .....	39
Figură 6.14 : Filtrul 14 Log-Gabor .....	40
Figură 6.15 : Filtrul 15 Log-Gabor .....	40
Figură 6.16 : Filtrul 16 Log-Gabor .....	40
Figură 6.17 : Filtrul 17 Log-Gabor .....	41
Figură 6.18 : Filtrul 18 Log-Gabor .....	41
Figură 6.19 : Filtrul 19 Log-Gabor .....	41
Figură 6.20 : Filtrul 20 Log-Gabor .....	42
Figură 6.21 : Filtrul 21 Log-Gabor .....	42
Figură 6.22 : Filtrul 22 Log-Gabor .....	42
Figură 6.23 : Filtrul 23 Log-Gabor .....	43
Figură 6.24 : Filtrul 24 Log-Gabor .....	43
Figură 6.25 : DSC_0296.jpg .....	43
Figură 6.26 : DSC_0295.jpg .....	43
Figură 6.27 : DSC_0295 cu filtrul 3 Log-Gabor.....	44
Figură 6.28 : DSC_0295 cu filtrul 24 Log-Gabor.....	44
Figură 6.29 : Puncte cheie folosind detectorul FAST pe DSC_0296 .....	45
Figură 6.30 : Puncte cheie folosind detectorul FAST pe DSC_0295 .....	45
Figură 6.31 : Puncte cheie folosind detectorul Harris pe DSC_0296, mărită..	46
Figură 6.32 : Puncte cheie folosind detectorul Harris pe DSC_0295, mărită..	46
Figură 6.33 : Puncte cheie folosind detectorul Log-Gabor pe DSC_0296, mărită ..	47
Figură 6.34 : Puncte cheie folosind detectorul Log-Gabor pe DSC_0295, mărită ..	47

---

Figură 6.35 : Puncte cheie folosind detectorul Shi-Tomasi pe DSC_0296, mărita	48
Figură 6.36 : Puncte cheie folosind detectorul Shi-Tomasi pe DSC_0295, mărita	48
Figură 6.37 : Puncte cheie folosind detectorul ORB pe DSC_0296.....	49
Figură 6.38 : Puncte cheie folosind detectorul ORB pe DSC_0295.....	49
Figură 6.39 : Puncte cheie folosind detectorul SIFT pe DSC_0296.....	50
Figură 6.40 : Puncte cheie folosind detectorul SIFT pe DSC_0295.....	50
Figură 6.41 : Reprezentarea grafică a raportului deviației pentru fiecare set ..	53
Figură 6.42 : Reprezentarea grafică a erori de reproiectare .....	53
Figură 6.43 : Graficul algoritmilor în funcție de numărul de conexiuni stabile	54
Figură 6.44 : Panorama FAST + ORB + BFM .....	55
Figură 6.45 : Panorama Harris + BRIEF + BFM.....	55
Figură 6.46 : Panorama Harris + FREAK + BFM.....	55
Figură 6.47 : Panorama Harris + ORB + BFM .....	55
Figură 6.48 : Panorama FAST + FREAK + BFM .....	55
Figură 6.49 : Panorama FAST + BRIEF + BFM .....	55
Figură 6.50 : Panorama Log-Gabor + BRIEF + BFM .....	56
Figură 6.51 : Panorama Log-Gabor + FREAK + BFM .....	56
Figură 6.52 : Panorama Log-Gabor + ORB + BFM .....	56
Figură 6.53 : Panorama ORB + BRIEF + BFM .....	56
Figură 6.54 : Panorama ORB + FREAK + BFM.....	57
Figură 6.55 : Panorama ORB + ORB + BFM.....	57
Figură 6.56 : Panorama Shi-Tomasi + BRIEF + BFM .....	57
Figură 6.57 : Panorama Shi-Tomasi + FREAK + BFM .....	57
Figură 6.58 : Panorama Shi-Tomasi + ORB + BFM .....	58
Figură 6.59 : Panorama SIFT + SIFT + BFM.....	58
Figură 6.60 : Panorama SIFT + SIFT + FLANN.....	58
Figură 7.1 : Locația executabilului aplicației .....	59
Figură 7.2 : Selectarea imaginilor de procesat.....	60
Figură 7.3 : Alegera detectorului .....	60
Figură 7.4 : Alegera împerechelor.....	61
Figură 7.5 : Eroare algoritmi incompatibili .....	61
Figură 7.6 : Alegera descriptorului .....	61
Figură 7.7 : Procesarea Log-Gabor .....	62

## **Anexa 3 – Lista tabelelor**

Tabel 3.1: Comparație rudimentară între algoritmi standard.....	14
Tabel 6.1 : Testul de deviație și eroare de reproiectare .....	52
Tabel 6.2 : Rezultatul cusăturii .....	54

## **Anexa 4 – Glosar**

inliners	Perechi de puncte cheie
cloud	Suport virtual pentru servere