UNIVERSITATEA ALEXANDRU IOAN CUZA IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

INTERACȚIUNE NATURALĂ CU UTILIZATORUL LA NIVEL DE CLIENT WEB

Propusă de

Păvăloi Alexandru

Sesiunea: iulie, 2016

Coordonator științific

Conf. Dr. Sabin-Corneliu Buraga

Rezumat: În această lucrare vom descrie o abordare nouă și promițătoare a interacțiunii om-mașină folosind gesturi ale palmei. Cu alte cuvinte, o bibliotecă JavaScript ce recunoaște și urmărește palmele utilizatorului, oferind astfel o modalitate nativă și în timp real de control a siturilor web. La baza acestei metode stau concepte bine cunoscute din aria recunoașterii de imagini împreună cu noi facilități introduse prin HTML5 precum elementul *canvas*.

Vom discuta în detaliu despre fiecare modul al aplicației, începând cu cele ce se ocupa de recunoașterea mâinii și sfârșind cu modul de utilizare și personalizare a aplicației. De asemenea, vom aduce in discuție, acolo unde este cazul, avantajele și dezavantajele tehnicilor folosite împreuna cu rezultatele preliminarii din timpul dezvoltării aplicației.

Rezultatele obținute, deși depind de factori externi, sunt promițătoare și indică fezabilitatea acestui nou mod de control al siturilor web. In final, ca studiu de caz a fost creată o simplă aplicație Web de sine stătătoare pentru a demonstra ușurința de folosire și personalizare a acestei biblioteci.

Cuvinte cheie: JavaScript, mână, web, interacțiune naturală

Cuprins

Introducere ... 7
1.1 Motivație și scop ... 7
1.2 Abordări existente ... 8
1.3 Recunoașterea palmei ... 8
1.4 Utilizarea librăriei ... 8
1.5 Contribuții ... 9
Abordări existente ... 10
2.1 OpenCV ... 10
2.2 JSFeat ... 11
2.3 js-handtracking ... 12
2.4 JS-ObjectDetect ... 12
Recunoașterea mâinii ... 13
3.1 Strategia aleasă ... 13
3.2 Fundamentele implementării ... 14
3.3 Extragerea de fundal ... 16
3.4 Filtrarea pixelilor de culoarea pielii ... 17
3.5 Detectarea feței ... 19
3.6 Procesări morfologice ... 20
3.7 Eliminarea componentelor mici ... 22
3.8 Calcularea înfășurătorii convexe ... 23
3.9 Diferențierea acțiunii mâinii ... 25
3.10 Optimizări notabile ... 27
Utilizarea librăriei ... 31
4.1 Implementare și utilizare ... 31
4.2 Feedback-ul utilizatorului ... 33
4.3 Aplicația web de testare ... 36
4.4 Rezultatele obținute ... 37
Concluzii ... 38
Bibliografie ... 40

Capitolul 1

Introducere

Gesturile umane sunt unelte indispensabile in cadrul oricărei conversații, gesticulația mâinilor, expresivitatea feței și pozițiile corpului oferind o serie de indicii valoroase despre mesajul transmis. Cu toate acestea, în contextul interacțiunii om-calculator ne rezumăm în mare parte la folosirea *mouse*-ului și a tastaturii. Aparatura de tipul „Kinect" oferă o nouă dimensiune de control asupra jocurilor dar nu exista nimic asemănător și pentru utilizarea obișnuită a calculatorului.

Mai mult decât atât, interacțiunea om-calculator este un subiect de interes global [1], mai ales in contextul realității virtuale, *smart homes* și roboticii, iar recunoașterea mâinilor reprezintă un important punct de pornire în dezvoltarea de aplicații interactive, ce-i permit utilizatorului să comunice cât mai natural.

Așadar scopul nostru este de a aduce interacțiunea prin gesturi in fiecare aplicație web prin prisma unei biblioteci JavaScript, ultimul standard HTML oferind o unealtă de mare ajutor în realizarea acestui scop. Elementul *canvas* împreună cu *API-ul* camerei web sunt o combinație esențială și suficientă pentru a avea recunoaștere în timp real in browser.

1.1 Motivație și scop

Principala motivație în crearea acestei biblioteci o reprezintă lipsa generală de modalități de control a siturilor web prin altceva decât *mouse* și tastatură. Deși subiectul detecției de mâini folosind JavaScript a mai fost tratat în biblioteci existente, acestea oferă doar detecția propriu-zisă în scop demonstrativ, fără o utilizare clară. Mai mult decât atât, deși acestea oferă rezultate bune pentru detecție, considerăm că putem crea ceva mai performant și util. De aceea, scopul nostru îl reprezintă crearea unei biblioteci care să suporte evenimente de interacțiune, mai exact cel de *click*. Această bibliotecă e dezirabil să aibă o dimensiune redusă, plus o documentație completă și o integrare imediata cu orice aplicație web modernă.

Din cauza varietății de factori externi ce pot afecta imaginile de la camera web dintre care amintim cantitatea de lumină prezentă și rezoluția camerei propriu-zise, am impus o serie de condiții necesare pentru buna funcționare a acestei aplicații:

1. fundalul imaginilor în care e prezent utilizatorul trebuie să fie static
2. utilizatorul trebuie să poarte haine de culori diferite de cea a pielii
3. pentru precizie acesta e nevoit să poarte haine cu mânecă lungă
4. cantitatea de lumină prezentă în cadru nu trebuie sa fie extremă

Așadar, vom considera atins scopul nostru dacă librăria este utilizabilă doar în cazul condițiilor amintite mai sus. Deși acest lucru restrânge spațiul total de utilizare, considerăm că cerințele actuale oferă totuși destulă libertate pentru o utilizare obișnuită și folositoare.

1.2 Abordări existente

În acest capitol vom sintetiza cele mai importante librării deja existente ce se ocupă cu detectarea în timp real a mâinii utilizatorului folosind *stream*-uri video.

In primul rând vom analiza biblioteca C++ OpenCV. Aceasta reprezintă un standard în aria analizei de imagini, oferind o suită de funcționalități indispensabile pentru orice programator. Cu toate acestea, nu o putem folosi pentru soluția actuală deoarece este implementată în limbajul C++, bazându-se extensiv pe tehnologia multiplelor fire de execuție.

Căutând totuși soluții in timp real ce folosesc doar JavaScript la nivelul clientului, observăm o serie de biblioteci *open source* precum *JSFeat*, *ObjectDetect* si *HandJs* ce propun soluții pentru detectarea mâinii. Acestea sunt un bun punct de plecare in construirea unei librării robuste așa că le vom analiza pe fiecare in parte pentru inspirație.

1.3 Recunoașterea palmei

Acesta reprezintă capitolul de bază al lucrării curente căci vom prezenta structura librăriei, punând accent asupra API-urilor de bază folosite, a modulelor acesteia și a interacțiunii dintre ele. În primul rând vom arunca o privire de ansamblu asupra arhitecturii aplicației enunțând cele mai importante tehnici si algoritmi utilizați in crearea ei.

În continuare vom descrie în detaliu implementarea librarii punând accentul pe fiecare tehnică folosită pentru recunoașterea mâinii. Vom compara diferite metode și apoi vom arăta de ce o combinație intre extragerea de fundal și detecția culorii pielii produce cele mai bune rezultate iar în finalul acestui capitol vom face din nou o trecere în revista a metodelor utilizate.

1.4 Utilizarea librăriei

În acest ultim capitol ne vom concentra, din prisma unui utilizator, asupra folosirii acestei librării. Vom prezenta ușurința de integrare folosind o aplicație web separat creată doar pentru acest scop. De asemenea, vom detalia toate modurile posibile de configurare a acestei aplicații, facilitate ce-i oferă librăriei abilitatea de a se adapta la situații cât mai diverse.

În final vom discuta despre informațiile de funcționare pe care un utilizator le primește în timp real. Această funcționalitate o considerăm extrem de importantă pentru a crea o experiență cât mai plăcută.

1.5 Contribuții

Codul scris pentru crearea acestei librarii si a aplicației demonstrative este original si creat in totalitate de către autor. Singura excepție o reprezintă folosirea librăriei *JSFeat* pentru detectarea feței. Licența acelei librării este una deschisă, de tip MIT, [4] ce permite folosirea și alterarea conținutului după bunul plac. În acest sens mulțumim autorului Eugene Zatepyakin.

Toate tehnicile folosite in această lucrare sunt fie creație proprie, fie preluate din documente de cercetare in analiza imaginilor, întreaga bibliografie fiind prezentată la finalul lucrării. Metoda de recunoaștere a mâinii este bazată pe concepte bine cunoscute în industrie. Deși ideea nu e originala, implementarea acestora in timp real folosind limbajul JavaScript alături de optimizările aduse sunt.

Mai mult decât atât, modalitatea de configurare a aplicației reprezintă o idee originală și proprie autorului ce dorește ca aceasta creație să fie liber disponibilă întregii lumi. În final mulțumim coordonatorului Conf. Dr. Sabin-Corneliu Buraga pentru sprijinul acordat în dezvoltarea acestei librării.

Capitolul 2

Abordări existente

2.1 OpenCV

OpenCV [5] este cea mai populară și robustă bibliotecă pentru analiza și procesarea imaginilor. Optimizată pentru procesări în timp real, OpenCV oferă o serie de algoritmi indispensabili pentru orice aplicație de acest tip, precum manipularea culorilor imaginii, detecția de obiecte folosind clasificatori Haar etc. Limbajele în care această bibliotecă este disponibilă sunt C++, C, Java și Python.

Marea majoritate a algoritmilor necesari acestui proiect sunt deja implementați în OpenCV. De exemplu, convertirea unei culori din spațiul RGB la cel HSV, aflarea înfășurătorii convexe ale unui set de puncte sau realizarea de operații morfologice sunt deja implementate în această librărie. O soluție care folosește acești algoritmi, testați de-a lungul timpului, pentru recunoașterea palmei, ar fi mai rapid de dezvoltat si mult mai stabila decât implementarea de la zero a acestor metode.

Cu toate acestea, cum OpenCV nu oferă suport pentru JavaScript, singura modalitate de a utiliza algoritmii acestei librarii pentru scopul nostru ar fi împreună cu un limbaj de back-end. Deci, fiecare dezvoltator de aplicații web care își dorește să ofere această nouă dimensiune de control ar trebui, in cel mai fericit caz, doar să configureze apeluri client-server, iar in cel mai nefericit, sa apeleze metode externe din limbaje in care OpenCV este suportat. Cu alte cuvinte, scopul principal al acestui proiect, acela de a crea o soluție unitară, independentă de platforma unde rulează aplicația si limbajul de server, ar fi imposibil de atins.

Deși nu am putut folosi librăria OpenCV pentru implementarea propriu-zisă, aceasta ne-a fost de folos în mai multe moduri pe parcursul proiectului, după cum vom prezenta în continuare:

În primul rând, a reprezentat o privire de ansamblu asupra a ceea ce este deja posibil și realizabil. Găsind în documentația OpenCV o multitudine de algoritmi ce oferă o procesare în timp real a imaginilor am ajuns la concluzia că, daca a fost posibil in C++, este cu siguranță posibil și în contextul web-ului modern folosind JavaScript.

În al doilea rând, o analiză asupra modului în care OpenCV stochează și utilizează imaginile, sub formă de matrice de pixeli, a oferit premise pentru crearea unei librării cât mai robuste. De asemenea siturile de discuții specializate pe aceasta tema [6] au oferit sugestii de implementări folosind funcții/metode din OpenCV, lucru de mare ajutor în a ne forma o idee despre posibile abordări ale problemelor apărute.

În încheiere, această bibliotecă axată pe analiza și procesarea imaginilor reprezintă un pilon de bază pentru fiecare aplicație cu interes in aceste arii. Din nefericire însă, nu există suport OpenCV pentru JavaScript, unul dintre principalele motive, din punctul nostru de vedere, pentru lipsa marcantă de aplicații web, în timp real cu accent pe analiza imaginilor.

2.2 JSFeat

JSFeat este o bibliotecă pentru analiza și recunoașterea imaginilor scrisă in JavaScript.

„Acest proiect își propune să exploreze posibilitățile JavaScript/HTML5 folosind algoritmi moderni de analiză a imaginilor" (Eugene Zatepyakin – traducere în română [7])

Existența acestei librării a însemnat un important punct de plecare pentru proiectul nostru. In special, detecția feței în timp real folosind clasificatori Haar a marcat dovada că algoritmi de detecție pot fi creați și utilizați în *browserele* moderne.

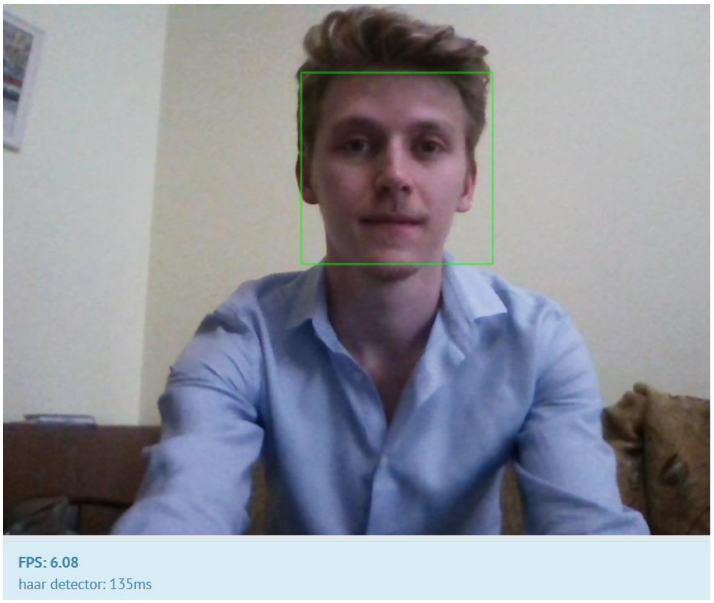

Figura 1 - Detecția feței folosind librăria JSFeat

O altă descoperire promițătoare din timpul perioadei de documentare a reprezentat prezența clasificatorilor pentru palmă și pumn. Aceștia ar putea fi folosiți, la fel ca cel pentru fată, pentru detecția în timp real a palmei sau pumnului. Însă rezultatele oferite au fost extrem de slabe. După mai multe încercări de recunoaștere am reușit o singură detecție corectă dar folosind o imagine extrem de clară și fără fundal.

Pe scurt, biblioteca JSFeat oferă o suită de algoritmi pentru procesarea imaginilor, începând cu procesări simpliste de culoare si până la operații complexe precum detecția marginilor și a obiectelor. Vom folosi această bibliotecă pentru a segmenta fața utilizatorului de restul imaginii, îmbunătățind astfel segmentarea palmei.

2.3 js-handtracking

Librăria *js-handtracking [6]* ne-a convins în privința tehnicilor pe care le vom folosi în etapa de detecție a mâinii. Această bibliotecă detectează și urmărește mâna folosind în primul rând culoare pielii iar apoi optimizează detecția folosind operații morfologice și calculând înfășurătoarea convexă a pixelilor. Rezultatele acestei librării însă, nu sunt extraordinare, detectând de multe ori pixeli din fundal ca fiind pixeli ai palmei.

Totuși, în lipsa unui clasificator Haar robust pentru palmă considerăm că abordările acestei librării sunt cele mai bune în acest moment, oferind și cea mai mare abilitate de personalizare. Astfel, am împrumutat ideile din spatele acestor tehnici pe care le-am implementat în stilul propriu bazându-ne pe nevoile librăriei noastre. Mai mult decât atât am introdus și alte metode pentru a realiza o detectare cât mai curată.

2.4 JS-ObjectDetect

Js-objectdetect [7] este librăria JavaScript ale cărei funcționalități se apropie cel mai mult de scopul acestui proiect. Folosind clasificatori Haar această bibliotecă oferă detecție, printre altele, special pentru mână și pumn. Mai mult decât atât. exemplul prezent pe pagina lor de *GitHub* [8] propune o modalitate de *scroll* a paginii folosind gesturi ale palmei. Deci, această abordare, oferă rezultate bune și ar putea fi folosită in lucrarea curentă.

Cu toate acestea, un clasificator Haar este asemănător unei cutii negre din cauza faptului ca nu poate fi personalizat odată antrenat. Totul depinde de nivelul de antrenare al acestui clasificator, și, din păcate, deși clasificatorii din această bibliotecă sunt mai bine antrenați ca cei din JSFeat, alegem să să folosim în continuare detecția bazată pe culoare tocmai din acest avantaj al personalizării.

Capitolul 3

Recunoașterea mâinii

3.1 Strategia aleasă

În urma analizării librăriilor prezentate în capitolul anterior și a diverselor publicații științifice enumerate în bibliografie am creat o strategia de recunoaștere bazată special pe nevoile acestui proiect. Astfel, inițial vom folosi API-ul HTML5 pentru preluarea fluxului de date video de la camera Web. Imediat ce avem imaginea, folosind JSFeat vom detecta fața utilizatorului, evident doar dacă aceasta este prezentă. Apoi, bazându-ne pe informațiile de culoare și locație ale fiecărui pixel îi vom filtra doar pe aceea care aparțin pielii și nu se afla în locația detectată drept față. În continuare, ne dorim filtrarea eventualului zgomot, deci vom aplica o serie de operații morfologice si eliminări de componente cu dimensiuni reduse. În acest moment ar trebui să avem detectați doar pixelii palmei, așadar, vom calcula înfășurătoarea lor convexă pentru a-i delimita în spațiu. Pasul final este acela de a determina, în funcție de dimensiunea înfășurătorii convexe, daca mâna se află deschisă sau închisă pentru a efectua sau nu operațiunea de *click*.

Dedesubt am ilustrat prin imagini cele doua poziții recunoscute ale mâinii. Cea din stânga o vom num *mână deschisă* în timp ce cea din dreapta este identificată drept acțiune de click sau *mână închisă.*

Figura 2- Ilustrarea acțiunii de mână deschisă'

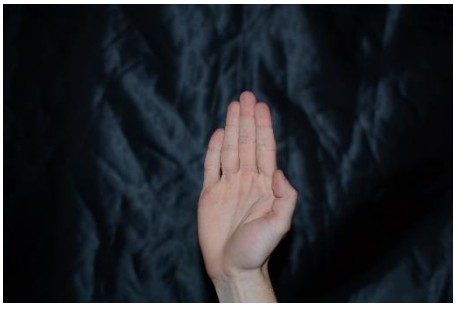

Figura 3-Ilustrarea acțiunii de 'mână închisă'

În diagrama următoare am ilustrat întregul proces de detecție a mâinii, începând cu preluarea fluxului de date video și până la diferențierea gesturilor. Acest procedeu este unul liniar, format din 8 pași, fiecare primind drept intrare rezultatul obținut în pasul anterior, realizând diverse calcule asupra sa și transmițându-l mai departe.

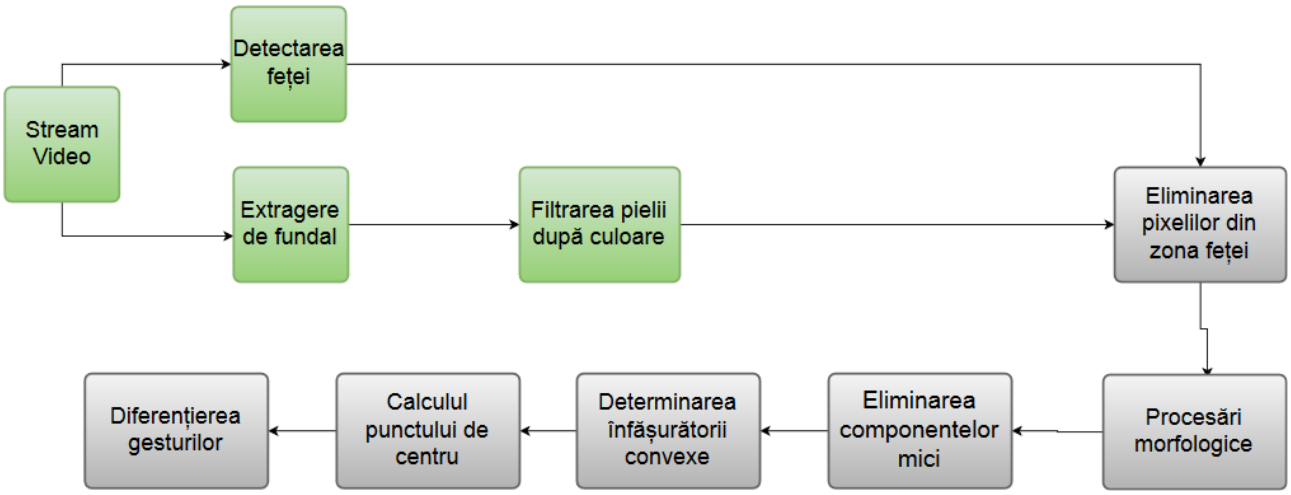

Figura 4 - Etapele detecției mâinii

Această lucrare folosește o serie de structuri proprii pentru manipularea imaginilor, structuri pe care le vom prezenta pe măsură ce sunt folosite pentru fiecare pas în parte.

3.2 Fundamentele implementării

WebRTC [8] (Web Real-Time Communication) este un API HTML5 ce a apărut din nevoia de a crea aplicații web necesitând suport nativ pentru comunicare în timp real audio sau video. Astfel, aplicațiile ce se folosesc în timp real de aceste medii de comunicare pot fi dezvoltate unitar, după o metoda standardizată de Consorțiul Web [9], eliminând astfel nevoia de biblioteci sau aplicații externe și funcționând pe toate browserele moderne. Această tehnologie a fost folosită în această lucrare pentru a obține imaginile camerei Web ale calculatorului utilizatorului. Bineînțeles, nevoia prezenței unui dispozitiv de tipul cameră web este necesară acestui proiect. În ceea ce privește securitatea imaginilor captate de la camera web, API-ul WebRTC criptează atât transmisiunile audio cât și pe cele video. Acest lucru este în special benefic în cazul rețelelor publice de internet unde vulnerabilitatea atacurilor este mai mare.

Pentru a obține în timp real imaginile am folosit API-urile *getUserMedia* și *getMedia* ce sunt implementate de toate browserele moderne. Înainte de a prelua fluxul de informații de la dispozitivul utilizatorului acestea cer permisiune de acces, lucru extrem de binevenit pentru siguranța intimității. De asemenea, în timpul rulării aplicației, browsere moderne oferă indicii vizuale pentru a semnaliza drepturile oferite de către utilizator.

În continuare am folosit elementul *canvas*, adăugat în HTML5, ce poate fi utilizat pentru a desena forme și imagini într-o aplicație web folosind JavaScript. Imaginea obținută de la camera web a utilizatorului conține informații de culoare despre fiecare pixel ce o compune.

Astfel, o imagine este codificata într-o structura de tipul *Uint8ClampedArray,* ce reprezintă un vector in care fiecare element ocupa exact 8 biți si are valoarea în intervalul 0-255. Această structură poate fi accesată doar printr-un element *canvas* așadar am recurs la crearea dinamică a unui astfel de element, folosind JavaScript, pe care nu l-am introdus în pagină decât pentru a vizualiza rezultatele detecției.

Spațiul de culoare folosit pentru codificarea valorilor este RGBA (Red-Green-Blue-Alpha). Acesta are la bază 4 canale, conținând informații pentru canalele roșu, verde, albastru dar și pentru canalul alpha ce este folosit pentru a reține informații despre opacitatea culorii. Astfel, pentru fiecare pixel al imaginii structura de stocare *Uint8ClampedArray* va reține 4 valori in intervalul 0-255, câte una pentru fiecare din cele 4 canale. Dimensiunea aleasă pentru crearea elementului canvas este extrem de importantă deoarece, cu cât rezoluția este mai mare suntem nevoiți să procesăm un număr mult mai mare de elemente. Așadar am ales o dimensiune de 640 * 480 pixeli lucru ce duce la un număr total de 1.228.800 de elemente necesare pentru a stoca o imagine. În practica acest număr s-a dovedit a fi îndeajuns de mic pentru a menține abilitatea librăriei de a recunoaște mâna în timp real.

Web workers sunt o modalitate simplă de a rula cod JavaScript într-un fir de execuție diferit de cel principal al aplicației. Cu alte cuvinte, este modalitatea web-ului de a folosi conceptul de multiple fire de execuție pentru îmbunătățirea performanței aplicațiilor. Aceștia sunt extrem de utili in cazul optimizării aplicațiilor ce realizează cantități mari de procesare, de exemplu procesări ale unor fișiere de tip JSON de dimensiuni mari. În aceste cazuri interfața cu utilizatorul se blochează până la finalul procesărilor distrugând astfel experiența utilizatorului.

Librăria JavaScript creata de noi se confrunta cu aceeași problema. Dorind fezabilitatea folosirii alături de orice aplicație web existentă, trebuie să ne asigurăm că permitem operarea în condiții optime și neblocante a aplicației utilizatorilor în timp ce realizăm calculele necesare recunoașterii mâinii.

O primă soluție la această problemă ar fi procesarea pe server și comunicarea rezultatelor, în timp real, folosind WebSockets. Deși o soluție viabilă în multe cazuri, noi dorim crearea unei soluții independente de limbajul de pe server deci folosirea WebSockets iese din discuție. Așadar utilizarea unui WebWorker pentru izolarea codului librăriei este cea mai bună alegere pentru cerințele actuale. Comunicare dintre aplicația principală și codul ce rulează într-un *Web Worker* se realizează printr-un sistem de mesaje unidirecțional. Așadar, folosind metoda *postMessege(message)* putem trimite instrucțiuni de procesare spre *worker* și rezultatul procesării spre aplicația principală. O ilustrare a acestei scheme de procesare este prezentată în continuare:

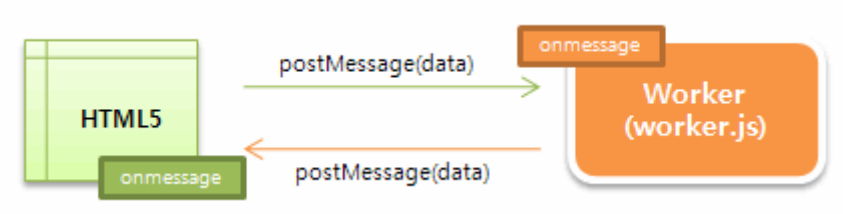

Figura 5 - Schema de comunicare cu Web Workeri

Axându-ne asupra aplicației de față, observăm că la baza ei stau două tipuri de mesaje ce trebuie transmise: informații despre imagine către worker și informații despre tipul și poziția gestului către firul principal. Din punctul de vedere al structurilor implicate în comunicare, am ales obiecte JSON fiecare având exact două proprietăți: *type* și *data.*

Proprietatea *type* va semnaliza tipul mesajului transmis. În cazul în care trimitem o imagine va avea valoarea IMAGE, în cazul în care trimitem coordonatele click-ului va avea valoarea CLICK, în cazul mișcării mâinii valoarea MOVE iar în cazul în care worker-ul eșuează în a detecta mâna, valoarea MISS.

În ceea ce privește proprietatea *data*, aceasta va conține fie informația despre imagine, stocată sub forma unui *Uint8ClampedArray* fie informația despre coordonatele X și Y are locului unde s-a detectat gestul.

În concluzie, dezvoltarea tehnologiei WebWorker reprezintă un aspect fără de care aplicația curentă nu ar putea fi folosită în practică. Și cum aceștia sunt parte a standardului HTML5 deci suportați de toate browsere moderne, se dovedesc a fi soluția perfectă pentru păstrarea unei experiențe fluide si neblocantă.

3.3 Extragerea de fundal

Primul procedeu folosit pentru recunoașterea mâinii îl reprezintă identificarea si eliminarea programatică a fundalului imaginilor. Aplicăm această tehnică pentru a elimina eventualii pixeli de culoarea pielii care se afla în obiecte ce apar în fundal, scopul final fiind acela de a segmenta numai adevăratele puncte specifice pielii. În cazul în care fundalul este unul static această tehnică este aplicată cu mare succes însă suferă o scădere semnificativă în performanță în cazul unui fundal dinamic.

Modul de funcționare al acestei tehnici este relativ simplu. Imediat ce biblioteca este pornită, atenționăm utilizatorul că procedura de captură a fundalului este pe cale să înceapă pentru a și să părăsească cadrul. Apoi, vom realiza 20 de capturi de ecran în următoarele 5 secunde iar la finalul acestui timp vom crea o imaginea de referință pentru fundal făcând o simplă medie între valorile pixelilor din cele 20 poze. Imaginea rezultată va fi folosită drept fundal

pe tot parcursul rulării librăriei. Am ales sa capturam nu una ci mai multe imagini iar apoi să facem media între ele pentru a normaliza diferențele de lumină și zgomot ce pot apărea pe parcursul folosirii librăriei. Așadar, pentru fiecare pixel din fiecare imagine preluată, calculăm diferența față de pixelul corespunzător din fundal. Dacă acesta s-a schimbat cu mai mult de 10 unități pe oricare din canalele R,G sau B atunci îl considerăm ca neaparținând fundalului.

Un dezavantaj al acestei metode este acela că va elimina pixelii cu nuanța pielii, dacă fundalul are de asemene o culoare corespunzătoare acesteia. De aceea am implementat un mecanism de protecție ce verifică culoarea pixelului din fundal. Dacă acesta se încadrează ca fiind piele atunci nu vom aplica procedeul de extragere de fundal asupra sa.

În practică extragerea de fundal s-a dovedit a fi o idee bună, deoarece locurile cele mai obișnuite în care utilizatorii folosesc dispozitive de calcul sunt încăperi ce oferă un fundal total static.

Următorul pas pentru detectarea palmei îl reprezintă filtrarea pixelilor pielii folosind intervale de culoare, procedeu despre care vom vorbi în următorul capitol.

3.4 Filtrarea pixelilor de culoarea pielii

Al doilea pas în crearea acestei librării îl reprezintă identificarea zonelor de piele din cadrul imaginii folosind informații despre culoare. Această abordare este rapidă și ușor de implementat oferind o complexitate liniară căci fiecare pixel este independent analizat față de ceilalți. Metoda folosită este o combinație între două metode diferite prezente într-un articol de interes [10].

In primul rând, articolul menționat propune o metodă de a filtra culoarea pielii bazată pe spațiul de culoare RGB. Această metodă conține practic 2 intervale, unul potrivit pentru pielea aflată sub lumina naturală a zilei:

$$(R > 95) \text{ și } (G > 40) \text{ și } (B > 20) \text{ și}$$

$$(\max\{R, G, B\} - \min\{R, G, B\} > 15) \text{ și}$$

$$(|R - G| > 15) \text{ și } (R > G) \text{ și } (R > B)$$

iar altul pentru pielea aflată sub lumina lanternei sau sub lumină laterală puternică:

$$(R > 220) \text{ și } (G > 210) \text{ și } (B > 170) \text{ și}$$

$$(|R - G| \leq 15) \text{ și } (R > B) \text{ și } (G > B)$$

Aplicând aceste 2 reguli și selectând pixelii care se potrivesc în cel puțin una dintre ele, obținem o primă hartă a zonelor cu piele bazată pe spațiul RGB. Diferența dintre aplicarea unei singure reguli și reuniunea acestora este evidentă și dovedește o detecție sporită în cel de-al doilea caz.

În al doilea rând, același articol propune și o metodă secundară de descoperire a pielii folosind spațiul de culoare HSV (Hue-Saturation-Value). HSV este o schemă de culoare cilindrica și o alternativă populara la tradiționalul RGB, oferind informații nu despre fiecare culoare in parte ci despre rezultatul final. Așadar,

- canalul *Hue* reprezintă nuanța culorii (de ex: roșiatică, albastră etc)
- canalul *Saturation* semnifică intensitatea acelei culori
- și canalul *Value* ce reprezintă tonul culorii, mai exact cantitatea de lumină a acesteia

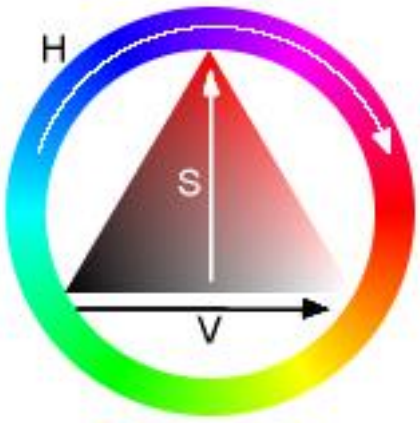

Figura 6 - Schema de culoare HSV

Metoda propusă pentru spațiul de culoare HSV este bazată doar pe canalul Hue. Astfel, un pixel ale cărui valori convertite la HSV satisfac cel puțin una din regulile de mai jos considerăm că acel pixel este piele:

$$H < 25$$

sau

$$H > 230$$

Cu toate acestea, trebuie sa convertim culoarea obținută din format RGBA în spațiul de culoare HSV, pentru a o putea filtra conform regulii de mai sus. Pentru acest lucru am folosit formula [11] pe care am testat-o folosind o serie de aplicații de conversie pentru a ne asigura de veridicitatea ei [12,13,14].

În continuare, același articol propune o intersecție a mulțimii pixelilor identificați după prima regula cu cei identificați după cea de-a doua. Motivația din spatele acestui lucru este evidentă și reiese din faptul că ambele reguli recunosc nuanțele pielii dar si nuanțe asemănătoare cu acestea deci putem reduce cantitatea de zgomot prin combinare lor.

După cum se poate observa în imaginile de mai sus, detecțiile incorecte ale fiecărui prag sunt eliminate prin intersecția lor. Așadar, am ajuns la o formula promițătoare pentru detecția pielii umane. În continuare vom analiza o optimizare adusa procesului de detecția a mâinii prin detectarea feței utilizatorului.

3.5 Detectarea feței

Procedura de recunoaștere a pielii folosind filtrare în funcție de spațiile RGB si HSV este o metodă rapidă ce produce rezultate bune în condiții de lumină favorabile. Cu toate acestea, în procesele de testare a apărut un dezavantaj major. În afară de pielea palmei, această tehnică va recunoaște și pielea mâinilor (in cazul in care acestea sunt descoperite), cat si pielea feței. Putem ignora pielea mâinilor datorită constrângerilor enunțate în capitolul 1 dar nu putem proceda la fel cu pielea feței. Considerăm că o constrângere de același tip și pentru fața utilizatorului ar face librăria total neutilizabilă în realitate, căci fața este în marea majoritate a timpului în aria de vizualizare a camerei web.

Pentru a rezolva aceasta situație am ales o soluție ce are la bază procedee de detectare a feței, mai exact faimoasa tehnică a lui Voila și Jones, clasificatorii Haar [12]. Odată ce chenarul corespunzător feței a fost determinat, vom elimina din procesare toți pixelii cuprinși în acest chenar cât și cei aflați cu cel mult jumătate din înălțimea chenarului deasupra sau dedesubtul acestuia. În acest mod ne asigurăm că pielea feței si a gâtului vor fi total eliminate, obținând o segmentare curata a pielea palmei.

În continuare vom discuta despre implementarea acestei metode folosind JavaScript. Din fericire biblioteca *JSFeat* oferă o soluție deschisă și robustă de detecție a feței folosind chiar clasificatori Haar. Aceasta pune la dispoziție un algoritmul de detecție alături de o serie de clasificatori dintre care, cel mai robust fiind cel pentru față. După o cercetare atentă asupra librării am decis să utilizăm, doar pentru această funcție, structurile specifice ei și nu cele proprii. Așadar, pentru detecția feței folosind JSFeat am utilizat următoarele structuri de date puse la dispoziție de librărie:

- *matrix_t*: structura de bază a librăriei, fiind în esență o structură ce înglobează o matrice. Librăria JSFeat oferă posibilitatea de creare de structuri *matrix_t* diferite în funcție de tipul valorilor conținute
- *data_t*: reprezintă o simplă învelitoare peste tipul *ArrayBuffer* prezent în JavaScript
- *pyramid_t*: conține mai multe obiecte de tipul *matrix_t* fiecare redimensionat la o scara de două ori mai mică decât primul

Folosind aceste structuri și documentația librăriei [13] am integrat detectarea feței în proiectul curent. Pe scurt, pentru fiecare cadru de imagine căutăm fața utilizatorului. În caz

că aceasta este găsită vom elimina din imaginea rezultată din pasul anterior aria detectată drept față.

Rezultatele obținute concluzionează că utilizarea procedeului de detecție a feței folosind librăria *JSFeat* duce la o segmentare superioară doar a zonei palmei. Imaginile următoare ilustrează rezultatul segmentării ambele cazuri.

În cele ce urmează, vom explica atât bazele operațiilor morfologice asupra imaginilor cât și procesările realizate în lucrarea curentă.

3.6 Procesări morfologice

Morfologia matematică, ce stă în prezent la baza procesării imaginilor, este o disciplină creată între 1964 și 1968 de către profesorul Georges Martheron împreună cu colegul său Jean Serra. Contribuția acestora a cuprins și crearea operațiilor morfologice asupra imaginilor binare precum eroziunea și dilatarea, ce au fost folosiți și în această lucrare pentru a elimina micile imperfecțiuni datorate texturii sau zgomotului imaginii.

Procesarea morfologică nu este o singură metodă ci o tehnică de transformare a imaginilor ce poate fi adaptată în funcție de situație și rezultatele dorite. Operațiile morfologice la nivelul imaginilor sunt independente, cu alte cuvinte, imaginea e transformată câte un pixel pe rând, aceasta transformare ținând cont doar de vecinii pixelului și de un element folosit pentru transformarea întregii imagini ce-l vom numi element morfologic și care este de asemenea o imagine binară de dimensiuni reduse.

O operație morfologica pe o imagine binara creează evident o nouă imagine binară în care un pixel are valoare pozitivă doar dacă testul morfologic pentru acel pixel este pozitiv. Prin test morfologic înțelegem o regula bazată pe elementul morfologic și vecinii pixelului testat, ce decide valoarea acelui pixel în imaginea rezultată.

Cele doua operații morfologice de bază, sunt eroziunea și dilatarea, ambele realizându-se de obicei cu elemente morfologice complete. Eroziunea produce o nouă imagine binară cu valori pozitive în toți pixelii unde, dacă elementul morfologic ar fi centrat, acesta ar încăpea în imagine. În urma acestei transformări imaginea se micșorează, fiind eliminat un strat de pixeli de la margini.

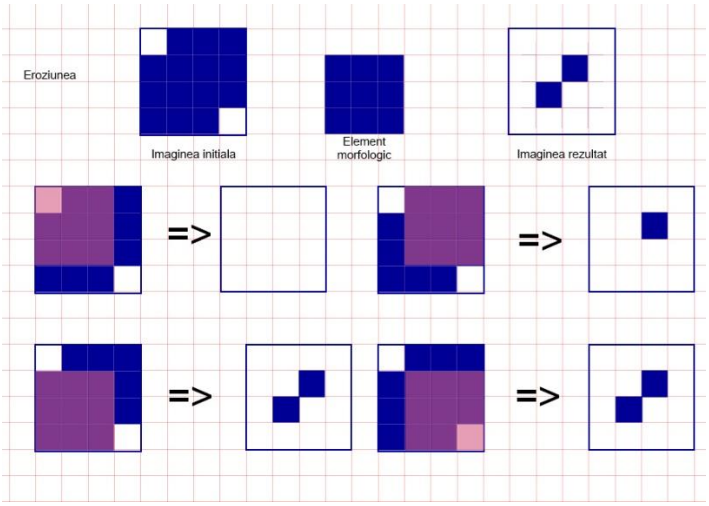

Figura 7 - Operația de eroziune asupra unei imagini

Un aspect interesant de utilizare al eroziunii, deși nefolosit în lucrarea de față, îl reprezintă obținerea conturului imaginii inițiale. Astfel, daca extragem imaginea erodată din cea originala atunci obținem o imagine ce conține doar pixelii de la conturul imaginii inițiale.

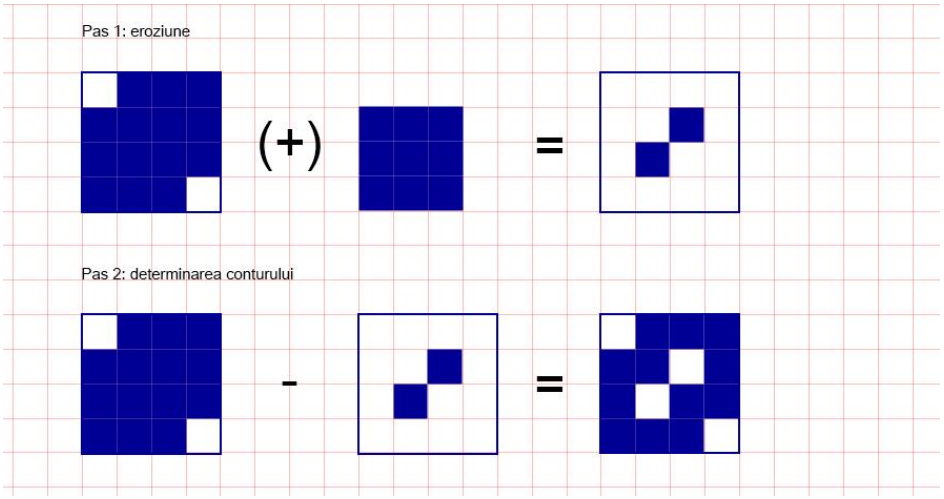

Figura 8 - Calcularea conturului unei imagini

De cealaltă parte, dilatarea produce o nouă imagine binară cu valori pozitive în pozițiile în care, dacă elementul morfologic ar fi centrat, acesta ar intersecta cel puțin un pixel al imaginii. Dilatarea are efectul invers eroziunii adăugând un rând de pixeli în regiunile exterioare dar și interioare.

Pentru îmbunătățirea calității imaginii obținute din pasul anterior am ales să combinăm aceste două operații, procedeul numindu-se în literatură: *închidere* [13]. Practic aplicăm operațiunea de dilatare urmată de cea de eroziune pe imaginea rezultată pentru a umple micile goluri prezente prezente. Extrem de important pentru acest procedeu îl reprezintă folosirea aceluiași element morfologic, iar în abordarea noastră am folosit un element morfologic complet de dimensiune 11. Structura de date folosită este evident o matrice

binară pe care am denumit-o *FullMorphoElement* și care primește la intrare doar dimensiunea dorită.

3.7 Eliminarea componentelor mici

Acest procedeu reprezintă ultima etapă înainte de recunoașterea propriu-zisă a gesturilor palmei. Toate procedeele anterioare au avut același scop: să creeze o imagine cât mai curată în care mâna să fie corect segmentată. Deși imaginea obținută este mult îmbunătățită față de imaginea cu care am pornit acest lung proces, există totuși un caz în care nici unul din procedeele anterioare nu va da rezultate. Să presupunem că în urma detectării și eliminării feței imaginea conține petice detectate incorect drept piele. O primă variantă pentru a le elimina o reprezintă procesările morfologice. Deși folosite aici în alt scop, dilatarea cu un element morfologic mai mare ar elimina în practică toate aceste zone mici de eroare dar ar distruge și forma mâinii. Chiar dacă combinăm eroziunea cu o dilatare ulterioară vom afecta și pixelii mâinii schimbând forma generală a acesteia.

Așadar procesările morfologice nu sunt o idee bună pentru o filtrare la asemenea scară și deci am ales un alt procedeu ce oferă rezultatele așteptate. Această metodă se bazează pe presupunerea conform căreia toate aceste detecții eronate neaparținând mâinii au dimensiuni mult mai reduse decât aceasta. Astfel considerăm o idee bună să descoperim și apoi să eliminăm toate acele componente cu dimensiuni mai mici decât o anumită valoare. Printr-o componentă înțelegem acele regiuni maximale de pixeli în care putem ajunge de la un pixel la oricare altul printr-un drum ce poate trece doar prin pixelii vecini. Acest termen este asemănător cu cel al componentelor conexe din teoria grafurilor [15].

În continuare, vom descrie pseudocodul algoritmului folosit pentru filtrarea acestor componente ale imaginii. Acest algoritm, compus din două etape separate, primește la intrare o matrice binară și un număr întreg, numit *PRAG*, ce reprezintă numărul minim de pixeli pe care trebuie să-i conțină o componentă pentru a nu fi ștearsă din imagine. În teorie, o alegere bună a valorii pragului ar cauza ștergerea tuturor erorilor de detecție obținând o imagine ce conține doar pixeli ai palmei.

Etapa de marcare a imaginii

```
parcurge în ordine, de la stânga la dreapta și de sus în jos fiecare pixel P al
imaginii

      dacă P nu are nici un vecin => asignează-i o nouă etichetă L și adaugă-l în
      structura SL a pixelilor cu această etichetă

      dacă P are un singur vecin în partea din stânga sau în sus => asignează-i
      eticheta acelui vecin și adaugă-l la structura corespunzătoare etichetei

      daca P are vecini atât sus cât și în stânga atunci =>
```

```
daca ambii vecini au aceeași etichetă => asignează-i lui P această
eticheta și adaugă-l la structura corespunzătoare

daca cei doi vecini au etichete diferite => asignează-i lui P oricare
din cele două etichete și adaugă-l structura corespunzătoare, iar apoi
marchează într-o structură de referință faptul că cele 2 etichete sunt
acum echivalente
```

La finalul acestei etape vom obține o serie de structuri, fiecăreia fiind-u-i asignată o etichetă și conținând doar pixeli cu acea etichetă. În plus vom avea și o structură ce reține etichetele echivalente.

Etapa de filtrare

```
pentru fiecare etichetă L folosită

   fie SL structura in care am memorat toți pixelii cu eticheta L

   daca numărul de pixeli din SL împreună cu numărul total de pixeli cu etichetă
   echivalentă cu L >= PRAG atunci adaugă toți acești pixeli în imaginea finală
```

În urma acestui algoritm obținem o nouă imagine binară ale căror componente au o dimensiune în pixeli mai mare decât pragul algoritmului. În practică algoritmul nu depinde numai de valoarea pragului ci și de golurile din zona mâinii, motiv pentru care am realizat acele procesări morfologice. Deși complexitatea ca spațiu este ridicată, folosind de două ori mai multă memorie decât e necesar pentru a memora toți pixelii, acest lucru reprezintă un dezavantaj minor față de rularea performantă ca timp. De asemenea, rezultatele obținute sunt extrem de promițător obținând în multe teste doar imagini ale căror pixeli sunt toți ai mâinii.

3.8 Calcularea înfășurătorii convexe

În urma ultimului pas de procesare a imaginii și dacă pragul a fost bine ales, am obținut o imagine binara ce segmentează complet doar palma. Cu toate acestea informațiile sunt sub formă de coordonate de pixel lucru ce nu ne oferă nici o informație directă despre forma și aria palmei. Cum avem nevoie de aceste informații pentru a diferența între gesturi am decis să ne folosim de tehnica numita *înfășurătoare convexă* pentru a le afla. Astfel, în acest capitol vom detalia atât procedeul înfășurătorii convexe cât și algoritmul folosit pentru calculul acesteia.

Fiind dată o mulțime de puncte in plan, înfășurătoarea convexă reprezintă acea submulțime minima ale cărei puncte formează un poligon convex înăuntrul căruia, sau pe laturi, se găsesc toate punctele din mulțime. Numeroase articole și metode de calcul au fost propuse de-a lungul timpului pentru rezolvarea acestei problema [16,17,18]. Împreună cu acestea, o

serie de algoritmi precum *Graham scan* și *Gift wrapping algorithm* și-au facut apariția. Deși eficienți ca și complexitate, timpul minim al acestora este de ordinul O(n * log(n)) unde *n* reprezintă numărul total de puncte. Din acest motiv am ales să implementăm propriul algoritm ce atinge o complexitate medie de O(n), acest lucru fiind posibil deoarece mulțimea punctelor este deja aranjată sub forma unei matrice binare.

Algoritmul este unul iterativ, parcurgând imaginea binară linie după linie și obținând la fiecare pas înfășurătoarea convexă parțială până în acel moment. Calculul iterativ al înfășurătorii convexe parțiale se bazează pe punctele extreme de pe fiecare rând, adică cel mai din stânga și cel mai din dreapta punct. La fiecare pas aflăm aceste puncte, iar apoi calculăm poziția fiecăruia față de dreapta formată din ultimele două puncte de pe acea parte. În funcție de acest calcul vom lua una dintre două decizii posibile: dacă punctul se află în interiorul dreptei, adică în poziția dreaptă pentru punctul stâng și invers pentru punctul drept, îl ignoram căci este deja cuprins de înfășurătoarea convexă parțială. Altfel trebuie să-l adăugăm la înfășurătoarea convexă dar, pentru a păstra constrângerea de minimizare, vom elimina mai întâi, în ordine inversă, toate punctele introduse pe acea parte atâta timp cât ele ar fi incluse în noua înfășurătoare convexă parțială.

În continuare, vom prezenta sub formă de pseudocod algoritmul descris mai sus. Merită menționat faptul că vom folosi o serie de funcții ce semnifică operații de bază asupra stivelor, după cum urmează:

```
PUSH(A,b) : inserează elementul b în stiva A
POP(A):     elimină elementul din vârful stivei A
TOP(A):     returnează elementul din vârful stivei fără a o modifica

START ALGORITM


S <- stiva goala
D <- stiva goala
pentru fiecare rând i al imaginii binare
        dacă linia i nu are nici un element => avansează la următorul rând
        altfel =>
                l <- cel mai din stânga punct de pe linia i
                r <- cel mai din dreapta element de pe linia i



                dacă i este primul sau al doilea rând ce conține elemente =>
                        PUSH(S, l)
                        PUSH(S, r)
                altfel =>
                        uS <- TOP(S)
                        pS <- penultimul element din S
                        dacă l se află în stânga dreptei (uS, pS) =>
                                cât timp ultimele doua elemente din S formează o
                                dreapta față de care l se află în stânga =>
                                        POP(S)
                        PUSH(S,l)
```

```
                        uR <- TOP(R)
                        pR <- penultimul element R
                        dacă r se află în dreapta dreptei (uR, pR) =>
                                cât timp ultimele două elemente din R formează o
                                dreaptă față de care r se află în dreapta =>
                                        POP(R)
                        PUSH(R,r)
inversează ordinea elementelor din R
atașează elementele lui R la finalul lui S

FINAL ALGORITM
```

În final, în stiva *S* vom obține, în ordine, punctele de pe înfășurătoarea convexă a mulțimii totale. Așadar, dacă am trasa linii între aceste puncte vom desena poligonul convex ce înfășoară toate punctele, după cum se poate observa în imaginea de mai jos.

Pentru a determina poziția punctului A față de dreapta formată din punctele Q și W vom folosi următoarea formulă:

```
d = (A.x - Q.x) * (W.y - Q.y) - (A.y - Q.y) * (W.x - Q.x) [20]
```

unde A, Q și W sunt structuri de tipul punct în spațiul cartezian 2D.

În cazul în care valoare lui *d* este egala cu zero atunci punctul testat se află exact pe dreaptă. Altfel, o valoare pozitiva pentru punctele extreme stânga semnifica poziționarea în parte exterioară a dreptei, în timp ce o valoare negativă pentru punctele din dreapta semnifica poziționarea in partea dreaptă, adică exterioară a dreptei.

Corectitudinea acestui algoritm este ușor de observat datorită faptului că algoritmul de mai sus se bazează pe punctele extreme de pe fiecare rând pentru a calcula înfășurătoarea convexă. Astfel, odată ce punctele extreme curente împreuna cu toate punctele extreme interioare aparțin înfășurătorii este evident că același lucru este adevărat și pentru punctele interioare rândului curent.

În concluzie, acest algoritm oferă o soluție corectă și rapidă de calcul a înfășurătorii convexe folosind-u-se de avantajele structurilor folosite în această bibliotecă. Complexitatea liniară ajută la procesarea extrem de rapidă a fiecărei imagini primite, ducând astfel la o bibliotecă ce poate funcționa în timp real.

3.9 Diferențierea acțiunii mâinii

În această ultimă secțiune vom analiza metoda de recunoaștere a acțiunii de click, motivând totodată alegerea algoritmului folosit. Amintim că gestul recunoscut pentru această acțiune este cel al unei palme având toate degetele întinse și apropiate.

Pentru a diferenția între cele două forme ale mâinii am putea să ne folosim de numărul de degete despărțite ale acesteia: 5 în primul caz și 0 în cel de-al doilea. Numeroase articole științifice despre recunoașterea mâinii [21,22,23] propun metode bazate pe conturul ei pentru a afla numărul degetelor. Astfel, o posibilă abordare ar fi calcularea punctelor de maximă defecțiune convexă, adică acele zone de piele care se abat cel mai mult de la înfășurătoarea convexă. Apoi, filtrând după distanță, am putea calcula numărul de degete despărțite ale palmei.

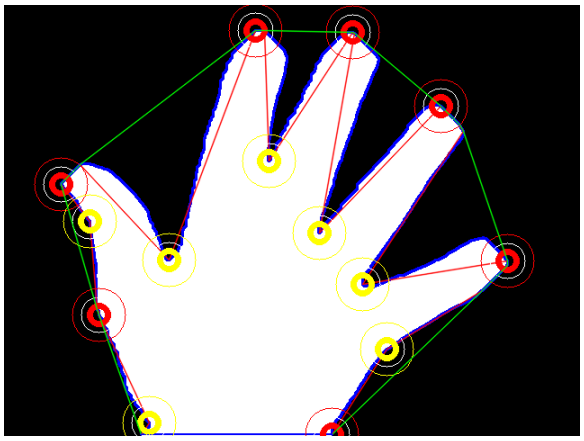

Figura 9 - Înfășurătoarea convexă (verde) și defecțiunile convexe (galben) [25]

O alta modalitate de a număra degetele prezentă în literatura de specialitate se folosește de unghiuri ale dreptelor dintre punctele de pe contur pentru a decide dacă un anumit punct este sau nu în vârful degetului.

Deși aceste metode au in practică rezultate bune, ambele se bazează pe calculul conturului, operație scump computațională. Deși pixelii conturului pot fi repede aflați prin tehnica descrisă în secțiune procesărilor morfologice, ordonarea lor nu este atât de trivială. Deși acest lucru este posibil [24] considerăm timpul computațional investit pentru aceste calcule inutil căci, în situația noastră, nu ne interesează exact numărul de degete ci doar diferențierea dintre cele două gesturi. Așadar analizând proporțiile mâinii [25] am descoperit că, în medie, raportul dintre lungimea celui mai lung deget și lățimea palmei este de 0.97 adică, degetul este aproximativ de două ori mai lung decât jumătate din lățimea palmei. Înlocuind lățimea palmei cu lățimea dintre extremele din stânga și din dreapta ale întregii mâini, putem afirma că aceasta propoziție își menține veridicitatea in cazul celui de-al doilea gest dar nu și în primul caz când, extremitățile laterale sunt extrem de depărtate.

Pe scurt, metoda folosită calculează distanța dintre centrul mâinii și cel mai apropiat, respectiv cel mai îndepărtat punct de pe înfășurătoarea convexă. Apoi, calculăm raportul acesta și dacă observăm că proporția este cel puțin dublă decidem că suntem în cazul celui de-al doilea gest. Cum cunoaștem deja punctele ce formează înfășurătoarea convexă, singura provocare rămasa este calcularea centrului acesteia. Pentru a-i determina poziția am utilizat o metoda de calcul rapida, cu o complexitate liniara, ce se bazează pe

coordonatele punctelor ce compun înfășurătoarea convexă **[Wiki - 27]**. Pseudocodul pentru această metodă este prezentat mai jos. Merită menționat faptul că o notație de tipul *P.x* reprezintă coordonata *x* a punctului *P*.

```
sx < -0;
sy <- 0;
a <- 0;
x <- 0;
y <- 0;
pentru fiecare vârf P al înfășurătoarei convexe
     fie T vârful imediat următor lui P
     s <- (P.x * T.y – T.x * P.y)
     a <- a + s
     sx <- sx +(P.x  + T.x) * s

a <- a / 2
x <- sx / (6 * a)
y <- sy / (6 * a)
```

La finalul algoritmului, în variabilele *x* și *y* vom avea valorile de coordonate pentru cele doua axe ale punctului central. Testând algoritmul pe coordonatele unui pătrat nu am obținut deloc rezultatul așteptat, adică punctul de la intersecția diagonalelor. Cu toate acestea, în practică, algoritmul de mai sus a oferit rezultate bune.

Figura 10 - Cele două gesturi împreună cu înfășurătoare convexă și centrul ei

Revenind la formula propusă mai sus, aceasta nu ne oferă nici o informație despre aranjarea mâinii și a degetelor. Totuși, ea diferențiază corect între cele două gesturi deci reprezintă o metoda bună în cazul folosirii conform specificațiilor, după cum se poate observa și în imaginea anterioară, un culoarea albastra a punctului central reprezintă o mână deschisă în timp ce culoarea roșie gestul închis.

3.10 Optimizări notabile

Crearea și rularea unor algoritmi optimi ca timp de execuție dar și ca spațiu de memorare reprezintă o necesitate pentru acest proiect. În cazul de față chiar și o îmbunătățire mică adusă timpului de execuție poate însemna o îmbunătățire de ordinul cadrelor pe secundă

pentru folosire librăriei. De exemplu, o îmbunătățire a procesului de detecție pentru fiecare cadru cu doar 10 milisecunde, de la 40 la 30, duce la o rată de cadre pe secundă mai bună cu 5 valori. Așadar, în acest capitol vom discuta despre alegerile de implementare făcute pentru a optimiza la maxim această librărie.

În primul rând, putem îmbunătăți viteza de detecție reducând semnificativ spațiul de căutare al mâinii bazând-u-ne pe informații despre poziția acesteia în cadrul precedent. Astfel, dacă presupunerea că utilizatorii vor folosi mișcări fluide și deloc haotice pentru controlul aplicațiilor atunci diferența poziției mâinii in cadre succesive este mică, deci putem restrânge aria de căutare a acesteia la doar o mică porțiune din întreaga imagine. Această porțiune va fi practic un dreptunghi centrat în poziția anterioară dar cu o suprafață mai mare decât a zonei detectate anterior pentru a permite eventualele mișcări a utilizatorului. Figura următoare ilustrează această metoda, chenarul marcat cu albastru fiind locul unde vom căuta mâna în pasul următor.

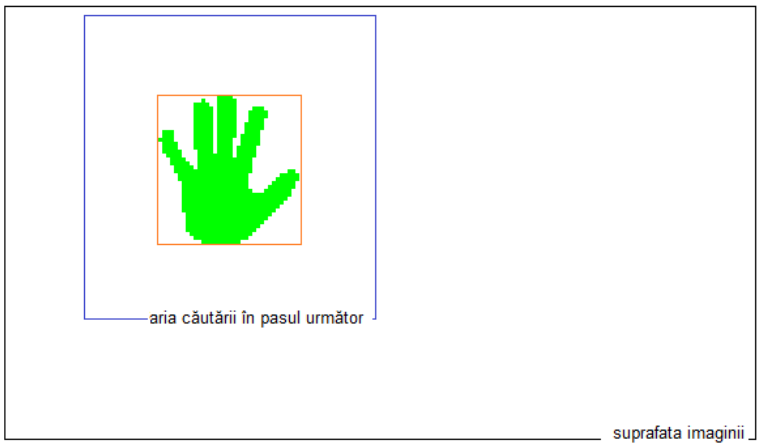

Figura 11- Optimizarea folosind detecția anterioară

Acest procedeu duce la o îmbunătățire semnificativă a timpului de detecție căci aria testată este mult mai mică decât întreaga imagine. În practică timpul mediu pentru realizarea întregului ciclu de detecție s-a îmbunătățit cu peste 50%. Totuși, această metodă nu funcționează în cazul în care utilizatorul își mișcă rapid mâna dintr-o parte în alta a imaginii. În acest caz aria analizată nu va conține mâna și vom fi nevoiți să reanalizăm întreaga imagine, lucru ilustrat și în diagrama următoare:

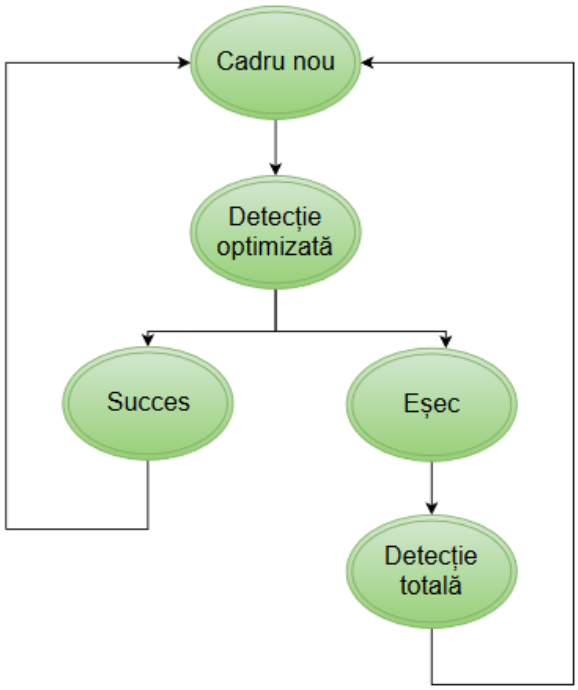

Figura 12- Procesul de optimizare folosind detecția anterioară

În al doilea rând, în urma unei inspecții atente a rezultatelor obținute la fiecare pas de după filtrarea de culoare am observat că doar o mica parte a imaginii conține valori reprezentative pentru piele. În multe cazuri doar palma, mâna, fața și gâtul utilizatorului vor fi singurele părți ale corpului expuse la camera web. Așadar, o noua optimizare a spațiului de stocare dar și a timpului propune stocarea acestor informați în mod similar cu matricele rare, deoarece, la fel ca acestea, conțin un număr mic de elemente nenule.

Consultând literatura de specialitate am observat că există mai multe modalități de a stoca matrice rare, fiecare cu avantajele și dezavantajele ei. Pentru această lucrare am ales formatul compresat pe linii [27], numit în engleză „Compressed Row Format”. Acesta stochează o matrice utilizând 3 structuri de tip vector, una pentru indicii rândurilor, una pentru indicii coloanelor și în final una pentru valorile propriu-zise. În cazul nostru, cum toate valorile sunt binare putem elimina ultima structura, reducând astfel și mai mult spațiul de stocare utilizat.

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Rând = [0 0 2 3]
Coloana = [0 1 2 1],

Figura 13- Exemplu de matrice binară împreună cu reprezentarea ei in format CSR

Cel de-al doilea vector, numit aici vectorul coloana, conține valorile coloanelor elementelor, în ordinea parcurgerii lor de la stânga la dreapta și de sus în jos. Se observă că numărul de elemente al acestui vector este egal cu numărul total de elemente nenule ale imaginii binare.

Primul vector, numit aici vectorul rând, stochează informații despre rândurile elementelor nenule si este completat folosind următoarea recurență:

```
R[0] =0
R[i+1] = R[i] + total(R[0]),
```

unde *total(X)* reprezintă al numărul de elemente nenule ale rândului X.

Astfel, vectorul ce stochează informații despre rânduri va avea același număr de elemente cât numărul total de rânduri al matricei, iar valoarea fiecărei poziții *i* semnifică indexul vectorului coloane începând de unde elementele acelui rând sunt memorate. Pe exemplul din exemplul anterior, dacă vrem să aflăm elementele rândului 2 atunci ne uitam în vectorul coloană începând de la poziția R[2] = 2.

Motivația din spatele acestei alegeri o constituie calcularea rapidă a coordonatelor unui punct fiind dat indexul său în vectorul coloană și invers, adică calcularea indexului său in vectorul coloană fiind date coordonatele sale. Aceste doua operații sunt de interes deoarece algoritmul folosit pentru eliminarea componentelor mici primește la intrare o astfel de structura și se folosește de aceste două operații.

În primul caz se observă imediat că coordonata coloanei este valoarea indexului i din vectorul coloană. În ceea ce privește valoarea rândului, cum vectorul rând este sortat crescător aplicăm o căutare binară pe acesta, oprindu-ne în momentul în care găsim o valoare cel puțin egală cu indexul „i” pentru care valoarea imediat următoare este strict mai mare ca acesta. Cu alte cuvinte, căutăm rândul ce-l conține pe *i* și astfel aflăm coordonatele din spațiul imaginii ale punctului cu indexul *i* în stocarea matricei rare.

În cel de-al doilea caz, cel în care ne dorim calcularea indexului său coloană bazat pe coordonatele sale, putem afla într-un singur pas valoarea vectorului rând, iar apoi folosind un pas iterativ testam valorile corespunzătoare de pe vectorul coloană pentru a găsi indexul elementului. Această operație parcurge, în cel mai rău caz, un număr de pași egal cu numărul de coloane al imaginii.

În concluzie, alegerea aceasta de stocare a imaginilor duce la o economie semnificativă de spațiu și timp. Singurul dezavantaj al acestei structuri este inserarea unui element printre alte elemente, operație ce nu se poate realiza fără rearanjarea tuturor elementelor următoare. Cu toate acestea, în cazul nostru nu avem nevoie de inserare finalul structurii deci putem folosi cu succes această modalitate de stocare.

Capitolul 4
Utilizarea librăriei

4.1 Implementare și utilizare

În aceasta secțiune vom explica, pentru început, modul în care un utilizator obișnuit va integra și utiliza această bibliotecă în propriile proiecte, iar apoi vom analiza în detaliu modalitatea și parametrii de configurare a acesteia.

Codul acestei librării este organizat în două fișiere JavaScript, unul pentru codul de bază numit *pafcal.main.js* și unul pentru codul ce va fi rulat de worker denumit *pafcal.worker.js*. Odată obținute, acest fișiere trebuie incluse în aplicație folosind-u-se elementul *script*:

```
<script src="pafcal.main.js" type="text/JavaScript"></script>
<script src="pafcal.worker.js" type="text/JavaScript"></script>
```

Toate funcțiile expuse de această bibliotecă sunt de fapt proprietăți ale obiectului *pafcal*. Am ales această abordare pentru a avea o uniformitate și claritate în numirea funcțiilor, lucru de mare ajutor pentru dezvoltatorii de aplicații. Astfel, pornirea acestei librarii este la fel de simplă ca apelarea metodei *pafcal.start()* în timp ce oprirea se face apălând metoda *pafcal.stop()*.

Metoda *pafcal.start()* acceptă un singur parametru, de tip obiect, folosit pentru a configura opțiunile inițiale ale aplicației. Acestea sunt doar două la număr: folosirea tehnicii de eliminare de fundal și folosirea detecției feței pentru a îmbunătăți procesul. În cazul in care funcția este apelată fără nici un parametru, ea va folosi parametrii impliciți de start:

```
{
      BACKGROUND_SUBSTRACTION : true,
      FACE_DETECTION:           true
}
```

Deși, în practică, eliminarea detectării de față duce la rezultate foarte slabe, am ales prezența ei ca setare configurabilă pentru posibilele situații în care acest lucru e dorit. În continuare, metoda de start apelează în ordine metodele corespunzătoare fiecărui pas, astfel:

```
pafcal.start() -> pafcal.backSubstraction() -> pafcal.colorThreshold() ->
pafcal.faceDetection() -> pafcal.morpho() -> pafcal.delete() ->
pafcal.convexHull() -> pafcal.decide()
```

Pentru implementarea acestor metode am ales o arhitectură funcțională. Astfel, fiecare funcție lucrează doar cu parametrii primiți la intrare și returnează rezultatul obținut. Această metodă oferă trei avantaje: claritate a codului, ușurință de testare și ușurință de modificare.

Deoarece funcțiile nu operează cu variabile externe duce la o înțelegere rapidă a scopului lor, fără a fi nevoiți să analizăm implementarea. De asemenea, codul este mai previzibil căci scăpăm de situația în care o variabila partajată se modifică neintenționat. În plus, testarea funcțiilor pure este mult mai simplă decât a funcțiilor impure deoarece ele reprezintă o singură funcționalitate unitară și izolată. În al treilea rând, datorită acestei arhitecturi funcționale și înlănțuite putem introduce cu ușurință o funcție nouă, atâta timp cât este scrisă în manieră funcțională și acceptă același tip de parametri. Dezavantajul constă în lipsa de comoditate căci trebuie constant să trimitem și să acceptăm parametri când am putea stoca valoarea aceea într-o variabilă publică.

Analizând funcțiile principale ale librăriei, merită menționat modul în care realizăm acțiunea de click la anumite coordonate. Pe scurt, creăm un obiect de tip eveniment de click căruia îi definim coordonatele drept coordonatele centrului mâinii. Apoi, preluăm elementul *HTML* aflat la acea poziție folosind funcția *elementFromPoint()* și declanșăm evenimentul de click asupra sa. Fragmentul de cod de mai jos ilustrează întreg procedeul:

```
function pafcal.click(x, y) {
    var event = document.createEvent("MouseEvent");
    event.initMouseEvent(
        "click",
        true,
        true,
        window,
        null,
        x, y, 0, 0
    );
    var topElement = document.elementFromPoint(x, y);
    topElement.dispatchEvent(ev);
}
```

În continuare vom vorbi despre sistemul de constante ce stă la baza implementării. Tehnicile de filtrare a culorii împreună cu aplicarea operațiilor morfologice și a eliminării componentelor mici se bazează pe praguri de valori, după cum am descris în capitolul anterior. Toate aceste valori sunt stocate în obiectul numit *pafcal.constants* și denumite folosind doar caractere majuscule și caracterul '_'. Valoarea implicită a acestui obiect este:

```
{
      WIDTH: 640,
      HEIGHT: 480,
      HSV_THRESHOLD: {
            HUE: [
                  {MIN: 0, MAX: 25},
                  {MIN: 230, MAX: 360}
            ],
            SATURATION: [
                  {MIN: 0, MAX: 100}
            ], VALUE: [
                  {MIN: 0, MAX: 100}
            ]
```

```
        },
        NUMBER_OF_BACKGROUND_ITERATIONS: 20,
        TRACKER_SIZE: 20,

}
```

Bineînțeles, orice utilizator poate configura aceste valori fără a modifica direct fișierul librăriei. Astfel, am creat funcția *pafcal.configure()* ce primește ca argument un obiect, îl parsează urmând ca pentru fiecare proprietate a cărei nume se potrivește cu numele unei proprietăți din obiectul de configurare, modifică valoarea ei cu cea primita în parametru. Obiectul trimis funcției de configurare nu trebuie să conțină valori pentru toate constantele ci doar pentru cele ale căror valoare se dorește modificată. De exemplu, un apel cu următorul obiect drept parametru:

```
{
        NUMBER_OF_BACKGROUND_ITERATIONS: 30,
}
```

va modifica numărul de cadre folosite pentru crearea imaginii de referință a fundului la 30. Deși în practică aceste valori pot fi schimbate și în timpul detecției, recomandăm configurarea acestora înainte de începerea ei pentru a asigura rezultate consistente.

În acest ultim paragraf vom prezenta metoda prin care un utilizator poate opri sau pune pauză detecției. Oprirea totală a funcționării librăriei se realizează prin apăsarea simultană a tastelor *SHIFT* și *S*. Dacă se dorește repornirea detecției trebuie repetată procedura descrisă mai sus. De asemenea, am implementat un mecanism de pauză a detecției ce are loc în urma acționării tastelor *SHIFT* și *P*. Dacă utilizatorul acționează aceste taste în momentul în care librăria funcționează rularea ei va trece în stare de pauză, iar daca această combinație e utilizată într-un moment de pauză librăria își va continua desfășurarea. Pentru a detecta aceste combinații ne-am folosit evenimentul *keydown* unde am verificat dacă tastele apăsate sunt cele căutate.

4.2 Feedback-ul utilizatorului

În această secțiune vom prezenta informațiile vizuale și textuale pe care un utilizator le primește în timp real folosind această librărie. Considerăm că menținerea utilizatorilor informați asupra stării în care se află detecția reprezintă un lucru extrem de important pentru crearea unei experiențe plăcute. De exemplu, în momentul în care s-a detectat un click, utilizatorul trebuie informat asupra acestui lucru pentru a ști să deschidă palma în vederea realizării unui click viitor. În cazul în care feedback-ul ar fi inexistent, considerăm că utilizatorii ar deveni confuzi și ar renunța ușor la a folosi această aplicație.

În continuare vom prezenta diferitele informații pe care le primește un utilizator în funcție de acțiunea sau starea în care se află procesul de detecție a gesturilor.

O primă stare a librăriei o reprezintă captura celor 20 de imagini de fundal pentru calcularea imaginii de referință a acestuia. În acest caz, în colțul din dreapta jos al aplicației va apărea un chenar gri cu o numărătoare inversa de 5 secunde reprezentând timpul rămas până la începerea procesului și deci îndemnul către utilizatori să părăsească fundalul. La finalul acestui timp, chenarul va dispărea și va fi înlocuit de un mic cerc roșu pentru a semnaliza că captura de ecran este în progres. Figura 15 ilustrează în ordine, de la stânga la dreapta apariția acestor elemente.

Figura 14- Indici vizuali pentru captura fundalului

În momentul în care întreg procedeul de captură a fundalului a luat sfârșit, orice indice vizual din partea din dreapta jos a browserului va dispărea și va fi înlocuit de una din următoarele trei ilustrații:

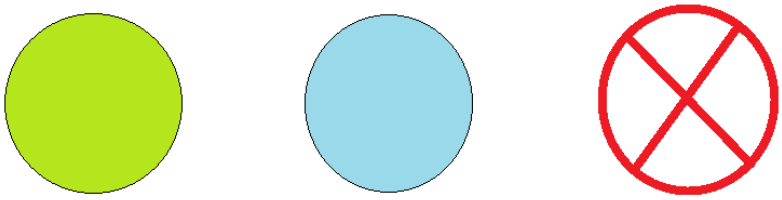

Figura 15- cele 3 elemente vizuale de pe parcursul rulării

Primele două cercuri vor urmări în timp real poziția centrului mâinii utilizatorului făcând posibilă acțiunea de click la locul dorit. Diferența de culoare semnifică tipul gestului identificat pentru mână. Astfel, cel verde din stânga indica detecția unei mâini deschise in timp ce cel albastru indica poziția închisa a mâinii utilizatorului.

Cel de-al treilea element prezentat mai sus va apărea în momentul în care librăria nu detectează nici o componentă. În acel moment, punctul activ de pe ecran va dispărea iar în colțul din dreapta jos a browserului vom arată această iconiță semnalând-ui utilizatorului lipsa vreunei detecții.

Acestea sunt singurele indiciile vizuale pe care utilizatorul le primește în timpul folosirii acestei aplicații. Cu toate acestea, un număr mult mai mare de informații este transmis. Aceste informații sunt textuale, apărând în consola browserului web, și de mare folos pentru

o retrospectiva în detaliu asupra rulării și rezultatelor librăriei. În continuare vom prezenta tipurile de informații ce apar în consolă.

Primul dintre acestea reprezintă alegerea de utilizarea sau nu a tehnicii de eliminare de fundal. Cum am precizat in capitolul dedicat acesteia, în momentul in care fundalul se schimba prea mult intre cele 30 cadre preluate atunci vom renunța la folosirea acestei tehnici considerând-o nepotrivită pentru un fundal dinamic. Consemnarea în consola browserului a acestei decizii se face în felul următor:

```
BACKGROUND_EXTRACTION: ON
BACKGROUND_EXTRACTION: OFF
BACKGROUND_EXTRACTION: DROPPED
```

Primele doua depind doar de alegerea utilizatorului din momentul în care configurează librăria în timp ce ce-a de-a treia va apărea doar daca programul a decis să renunțe la alegerea utilizatorului.

Următoarele informații au rol de statistică asupra întregii rulări a programului și vor apărea atunci când utilizatorul decide să oprească total funcționarea librăriei folosind combinația de taste descrisă anterior. Statistica va prezenta informații despre timpul de rulare în secunde, numărul total de imagini capturate, numărul de mâini detectate, timpul mediu necesar pentru detecția unui mâini cât și numărul de clickuri efectuate. Toate aceste informații vor veni într-un element JSON, astfel:

```
TOTAL_TIME_SECONDS:           123
TOTAL_FRAMES_CAPTURED:        2460
TOTAL_HANDS_DETECTED:         1500
DETECTION_TIME_MILISECONDS:   34
TOTAL_CLICKS:                 22
```

Mai mult decât atât, utilizatorul poate alege să afle informații complete despre fiecare cadru în parte. Aceste informații sunt reprezentative pentru fiecare cadru în parte și cuprind numărul de ordine al cadrului, starea detecției sale, coordonatele centrului detecției, timpul necesar și dacă s-a folosit sau nu cu succes fereastra de optimizare. Toate aceste informații vor veni într-un element JSON astfel:

```
FRAME_NUMBER:                 111
DETECTION_STATUS:             FREE_HAND / CLOSED_HAND / NONE
CENTER_POINT:                 {x: 10, y: 100}
DETECTION_TIME_MILISECONDS:   111
SURFACE_OPTIMIZATION_SUCCESS: TRUE / FALSE
```

Această funcție oferă o privire completa asupra întregului proces de detecție, pentru fiecare cadru în parte. O recomandăm în momentul în care se doresc operații de tip debugging asupra librăriei sau aplicației ce o folosește, căci informațiile apar mult prea rapid pentru a putea fi înțelese într-o rulare normala.

Așadar, această metoda de informare încheie această secțiune a acestei lucrări. Metodele prezentate aici oferă informații valoroase asupra utilizării, fiind un binevenit plus a acestei librării. În continuare vom descrie aplicația creată special pentru a testa această librărie.

4.3 Aplicația web de testare

Pentru a testa această librărie, a ilustra modul ușor de integrare, utilizare cât și procesul constant de feedback pe care utilizatorii îl primesc am creat o aplicație web separată special pentru aceste scopuri. Interfața acesteia nu este deosebit de prietenoasă dar se potrivește pentru testarea acțiunii de click a acestei librării. Ea consta dintr-o serie de butoane de dimensiuni mari, dispuse sub forma unui calculator, ce tipăresc după apăsare, elementul înscris pe ele in chenarul alb de deasupra. Această funcționalitate este îndeajuns pentru a observa comportamentul librăriei în condiții obișnuite.

Figura 16- Interfața aplicației create pentru a testa librăria

Serverul aplicației este unul static și simplist ce servește fișierul *index.html*. Am ales să-l creăm folosind limbajul NodeJS datorită ușurinței și rapidității de implementare. Astfel, folosind pachetele *node connect* si *serve-static* putem crea și porni un astfel de server în doar câteva linii de cod:

```
var connect = require('connect');
var serveStatic = require('serve-static');
connect().use(serveStatic(__dirname)).listen(8080, function(){
      console.log('Server running on 8080...');
});
```

Scenariul de testare implică nu doar evenimentul de click ci si corectitudinea informațiilor vizuale primite de utilizator. Astfel, vom testa mai întâi corecta apariție a informațiilor cu privire la captura de fundal din startul aplicației. Apoi, vom verifica apariția și mișcarea punctului de centru în funcție de mișcările mâinii, urmată de acțiunea de click a librăriei, dorind declanșarea în ordine, de la stânga la dreapta și de sus în jos a tuturor butoanele. În continuare ne vom axa asupra celorlalte informații vizuale oferite.

Vom testa schimbarea culorii punctului de centru in momentul strângerii mâinii cat și informația ce apare în momentul în care mâna nu face parte din câmpul vizual al camerei web. În final vom arunca o privire asupra consolei pentru a urmări în timp real informații detaliate despre rularea librăriei. In momentul in care proiectul de față trece toate aceste teste putem considera implementarea finala și scopul îndeplinit.

4.4 Rezultatele obținute

Rezultatele obținute sunt îmbucurătoare și demonstrează încă o dată posibilitatea unor astfel de moduri de control pentru dispozitivelor de calcul. Biblioteca creată este ușor de integrat și de utilizat în contextul aplicațiilor Web moderne iar modul de personalizare este intuitiv și binevenit în contextul extrem de larg al scenariilor de utilizare.

Testul principal enunțat în secțiunea anterioară a fost trecut fără mari probleme. Singurul inconvenient a fost nevoia de mutare a poziției capului pentru putea efectua evenimentul de click asupra butoanelor din partea stângă. De asemenea, palma trebuie sa fie paralele cu câmpul camerei web pentru a oferi o recunoaștere consistentă. Mai mult decât atât toate informațiile vizuale, începând cu eliminarea fundalului și încheind cu rezultatele în timp real a detecției au funcționat cum era de așteptat și au oferit informații importante.

De partea cealaltă, rezultate slabe au fost obținute în diverse condiții. În primul rând în cazul unei lumini puternici asupra mâinii, camera Web a laptopului folosit pentru testare produce rezultate cu puțină informație de culoare, făcând deci detecția pielii imposibilă. De asemenea, în cazul mâinilor acoperite de mănuși sau tatuaje biblioteca de față nu va oferi rezultatele așteptate.

Capitolul 5

Concluzii

Subiectul tratat în această lucrare este cel al interacțiunii dintre om și calculator. Acesta este un domeniu extrem de variat și complex dar cu mult potențial pentru dezvoltarea de sisteme si aplicații interactive. Scopul acestei lucrări a fost de a crea o bibliotecă ce expune o metodă de control a siturilor web folosind gesturi ale mâinii. Mai exact, ne-am propus sa creăm o bibliotecă JavaScript prin care să-i permitem utilizatorului să realizeze acțiuni de click pe situri web fără control asupra tastaturii sau a mouse-ului.

Rezultatul obținut este mai mult decât încurajator. Am reușit să detectăm și să urmărim mâna utilizatorului, în timp real, folosind camera web a laptopului sau calculatorului. De asemenea, am putut recunoaște și executa operațiunea de click asupra aplicației acestuia, si cel mai important, am reușit să realizam aceste lucruri în timp real, lucru absolut esențial pentru folosirea în practică a acestei librării.

Numeroase tehnici și metode au fost folosite pentru a ne atinge scopul. În continuare le vom prezenta doar cele mai notabile și importante dintre acestea.

În primul rând, utilizând culoarea pielii ca mijloc de detecție am obținut o viteză de rulare ridicată. Mai mult decât atât, folosind o combinație între extragere de fundal și detecție de piele am reușit să scăpăm de eventualele obiecte cu culoare asemănătoare din fundal.

În al doilea rând, modalitatea proprie de a calcula înfășurătoarea convexă ne-a adus o semnificată îmbunătățire față de metodele din literatură, obținând o complexitate medie de O(n).

Ce-a de-a treia tehnica importantă pentru robustețea acestei librării o reprezintă folosirea clasificatorilor Haar pentru detecția feței. Din cauza apariției feței în marea majoritate a scenariilor de utilizare a acestei librării a fost necesar să utilizam o metodă performanta de eliminare a acestor pixeli. Astfel, ne-am folosit de o altă bibliotecă JavaScript, ce oferă diferite funcții pentru procesarea si analiza de imagini, pentru a detecta fața independent a o elimina din imaginea procesată.

În al patrulea rând, merită menționată și tehnica de eliminare a componentelor conexe mici, tehnică folosita pentru curățarea imaginii de potențialul zgomot. Abia după acest procedeu putem afirma că avem o imagine ce conține doar mâna utilizatorului.

În cele din urmă, am adăugat la această bibliotecă o metodă de personalizare a majorității parametrilor folosiți în detecție. Astfel, utilizatorii nu sunt restrânși în a doar folosi librăria, putând să o adapteze în funcție de necesitățile proprii.

În continuare vom discuta despre posibilele direcții de îmbunătățire ale acestei librării. Prima dintre acestea reprezintă auto ajustarea culorilor din cadrele sub/supra expuse. În acest mod, cel puțin din prismă teoretică, vom putea aduce culorile pielii în intervalele detectate de librărie.

Al doilea mod de îmbunătățire se concentrează, la fel ca primul, pe recunoașterea pielii chiar și când culorile imaginii nu corespund cu cele din interval. Acest lucru poate fi realizat dacă pragurile necesare se calculează în funcție de culoarea descoperită în fața utilizatorului. Deși această metoda ar da greș în cazul condițiilor diferite de lumina dintre mână și față, o considerăm promițătoare pentru uzul general.

O ultimă direcție de îmbunătățire, nu doar a librăriei curente ci a tuturor aplicațiilor ce se bazează pe detecție de mâini ar fi antrenarea unui clasificator Haar la un nivel de performanță cel puțin la fel de ridicat ca pentru clasificatoarele existente deja pentru față. Astfel, tehnica de detecție ar fi mult mai rapidă și simplă. Cu toate acestea, mâna prezintă un mai mare grad de libertate decât fața, ducând astfel la imposibilitatea folosirii eventualilor clasificatori Haar pentru a detecta toate gesturile sale.

Capitolul 6
Bibliografie

În acest ultim capitol al lucrării de față vom enumera articolele, lucrările și siturile web folosite drept documentație pentru crearea acestei librării, ordinea acestora fiind dată de titlul lucrării.

Articole:

Xianquan Zhang & Zhenjun Tang & Jinhui Yu & Mingming Guo. A Fast Convex Hull Algorithm for Binary Image. May 28, 2009

Duan Hong & Luo Yang. A Method of Gesture Segmentation Based on Skin Color and Background Difference Method. Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)

Fahad Shahbaz Khan & Rao Muhammad Anwer & Joost van de Weijer & Andrew D. Bagdanov & Maria Vanrell & Antonio M. Lopez. Color Attributes for Object Detection

Ciar´an ´O Conaire & Noel E. O'Connor, & Alan Smeaton. Detector adaptation by maximizing agreement between independent data sources.

Rein-Lien Hus & Mohamed A-M & Anil K. Jain. Face Detection in color Images. IEEE Transactions On Pattern Analysis and Machine Intelligence, vol.24, No.5, pp.696-706, 2002

S. N. Karishma & V. Lathasree. Fusion of skin color detection and background substraction for hand gesture segmentation. International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 13 - 18 (2014), Special Issue of ICETETS 2014 - Held on 24-25 February, 2014 in Malla Reddy Institute of Engineering and Technology, Secunderabad– 14, AP, India

Arpit Mittal & Andrew Zisserman & Philip H. S. Torr. Hand detection using multiple proposals.

Qiu-yu Zhang & Mo-yi Zhang & Jian-qiang Hu. Hand Gesture Contour Tracking Based on Skin Color Probability and State Estimation Model. Journal of Multimedia, Vol. 4, No. 6, December 2009

Ray Lockton. Hand Gesture Recognition Using Computer Vision.

J. Fritsch & S. Lang & M. Kleinehagenbrock & G. A. Fink & G. Sagerer. Improving Adaptive Skin Color Segmentation by Incorporating Results from Face Detection

Gleb V. Tcheslavski. Morphological Image Processing: Basic Algorithms. http://ee.lamar.edu/gleb/dip/index.htm. Spring 2009

Yao Wang. Image Filtering: Noise Removal, Sharpening, Deblurring. EE3414 Multimeti Communication Systems – I

Mathew George1 & C. Lakshmi2. Object Detection using the Canny Edge Detector. International Journal of Science and Research (IJSR), India Online ISSN: 2319-7064

Paul Voila & Michael Jones. Rapid object detection using a boosted cascade of simple features. Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on (Vol. 1, pp. I-511). IEEE.

Fei Zhao & Huan zhang Lu & Zhi yong Zhang. Real-time single-pass connected components analysis algorithm. Zhao et al. EURASIP Journal on Image and Video Processing 2013, 2013:21

Manish Suyal1 &Vijay2. Review of Object Segmentation and Identification Using Edge Detection and Feature Matching Technique. International Journal of Advances in Computer Science and Technology, Volume 3, No.6, June 2014

Nusirwan Anwar bin Abdul Rahman & Kit Chong Wei and John See. RGB-H-CbCr Skin Colour Model for Human Face Detection.

D. G. Bailey & C. T. Johnston. Single Pass Connected Components Analysi. Proceedings of Image and Vision Computing New Zealand 2007, pp. 282–287, Hamilton, New Zealand, December 2007

Jamie Sherrah & Shaogang Gong. Skin Colour Analysis

Situri web:

Arthur C. Clarke. Automatic color adjustments. Titlu sit: http://pippin.gimp.org/image_processing/chapter-automaticadjustments.html

Thomas Royal. A High Level Description of Two Fingertip Tracking Techniques: k-curvature and convexity defects. Titlu sit: http://www.tmroyal.com/a-high-level-description-of-two-fingertip-tracking-techniques-k-curvature-and-convexity-defects.html. August 19, 201

Rob Hawkes. Canvas from Scratch: Pixel Manipulation. Titlu sit: http://code.tutsplus.com/tutorials/canvas-from-scratch-pixel-manipulation--net-20573. 17 Jun 2011

Abeer George Ghuneim. Contour Tracing Algorithms. Titlu sit: http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/alg.html. 2000

*. Contours : More Functions. Titlu sit: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_contours/py_contours_more_functions/py_contours_more_functions.html

Martijn van Mensvoort. Finger Length Proportions & Hand Shapes. Titlu sit: http://www.handresearch.com/diagnostics/finger-length-proportions-elemental-hand-shapes.htm. September 6, 2014

John M. Carroll. Human Computer Interaction. Titlu sit: https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/human-computer-interaction-brief-intro

Juan Mellado. JavaScript Hand Tracking. Titlu sit: https://github.com/jcmellado/js-handtracking

. KeyboardEvent. Titlu sit: https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent.MediaDevices.getUserMedia. Titlu sit: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia

*.Navigator.getUserMedia. Titlu sit: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/getUserMedia

Angelo Marletta. The fastest convex hull algorithm ever.Titlu sit: http://mindthenerd.blogspot.ro/2012/05/fastest-convex-hull-algorithm-ever.html. 21st May 2012

Jeffrey Juday. Understanding JavaScript Web Workers. Titlu sit: http://www.codeguru.com/csharp/csharp/cs_internet/article.php/c19231/Understanding-JavaScript-Web-Workers.htm

*. Using Web Workers Titlu sit: https://developer.mozilla.org/enUS/docs/Web/API/Web_Workers_API/Using_web_workers