Universitatea “Politehnica” din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Sistem de recunoaștere a mișcărilor și redarea acestora folosind un braț robotic

Proiect de diplomă

prezentat ca cerinţă parţială pentru obţinerea titlului de
Inginer în domeniul *Calculatoare şi Tehnologia Informaţiei*
programul de studii de licenţă *Ingineria Informaţiei*

Conducători ştiinţifici
Ş. L. Dr. Ing. Anamaria RĂDOI
Prof. Dr. Ing. Corneliu BURILEANU

Absolvent
Costin-Claudiu ANEGROAEI

București
2019

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul **EAII**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **ANEGROAEI I. Costin-Claudiu , 441A**

1. Titlul temei: Sistem de recunoaștere a mișcărilor și redarea acestora folosind un braț robotic

2. Descrierea contribuţiei originale a studentului (în afara părţii de documentare) și specificații de proiectare:
Lucrarea are drept scop crearea unui sistem de recunoaștere în timp real a mișcărilor efectuate de brațul unui utilizator. Aplicațiile acestui sistem vizează în mod special domeniul medical, fiind conceput pentru a ajuta persoanele cu handicap (de exemplu, persoanele care nu pot scrie în urma unei traume suferite la nivelul degetelor sau a mâinii). De asemenea, aplicațiile se pot extinde și în domeniul educațional.
Sistemul pornește de la o brățară prevăzută cu accelerometru, giroscop și senzori de activitate musculară și care va fi amplasată pe antebrațul utilizatorului. În urma procesării datelor, se va dori recunoașterea unor categorii de gesturi folosind algoritmi de măsurare a similarității între două semnale și algoritmi de clasificare supervizată. Recunoașterea mișcării efectuate de antebraț va fi transpusă într-o mișcare efectuată de un braț robotic (Kinova Jaco robotic arm) programat să execute o serie de mișcări predefinite. Aplicația va fi dezvoltată în limbajele de programare C++ și Python astfel: procesarea datelor colectate de brățară și controlul brațului robotic în C++, iar modulul de recunoaștere a mișcărilor va fi implementat în Python/C++.
În final, se va obține un sistem care va fi capabil să colecteze date generate prin mișcarea brațului unui utilizator, să recunoască gestul pe care utilizatorul l-a efectuat și să redea mișcarea dorită cu ajutorul brațului robotic.

3. Resurse folosite la dezvoltarea proiectului:
Limbaje de programare: C++,Python Biblioteci: numpy, myo,KinovaTypes Dispozitive: Myo Armband, Kinova Jaco

4. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline:
DEPI,POO,RFIA,TOP,Programare Distribuita

5. Proprietatea intelectuală asupra proiectului aparţine: U.P.B.

6. Data înregistrării temei: 2018-11-29 21:34:02

Conducător(i) lucrare,
Ş. L. Dr. Ing. Anamaria RĂDOI
semnătura: .....

Prof. Dr. Ing. Corneliu BURILEANU
semnătura: .....

Student,
semnătura: .....

Director departament,
Prof. dr. ing Sever PAŞCA
semnătura: .....

Decan,
Prof. dr. ing. Cristian NEGRESCU
semnătura: .....

Cod Validare: **355d21af77**

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul *" Sistem de recunoaștere a mișcărilor și redarea acestora folosind un braț robotic"*, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Calculatoare și Tehnologia Informației*, programul de studii de licență *Ingineria Informației* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 12/06/2019

Absolvent Costin-Claudiu ANEGROAEI

(semnătura în original)

Cuprins

Listă de figuri

Listă de Tabele

Listă de abrevieri

DTW – dynamic time warping
OCR – optical character recognition
SDK – Software Development Kit

Introducere

Motivație și aplicabilitate

Recunoașterea mișcărilor mâinii poate fi utilizată într-un domeniu vast de aplicații, oamenii de știință realizând soluții diverse pentru îndeplinirea acestei sarcini. Până la momentul actual, cele mai utilizate metode de recunoaștere ale mișcărilor se bazează pe analiza imaginilor și a fost folosită în sisteme de control la distanță, aplicații interactive sau analize comportamentale a persoanelor.

Ne propunem crearea unui sistem de recunoaștere în timp real a mișcărilor efectuate de brațul unui utilizator. Aplicațiile acestui sistem vizează în mod special domeniul medical, fiind conceput pentru a ajuta persoanele cu handicap (de exemplu, persoanele care au suferit o traumă la nivelul degetelor sau a mâinii). De asemenea, aplicațiile se pot extinde și în domeniul educațional.

Spre deosebire de abordările anterioare, sistemul dezvoltat în acest proiect pornește de la o brățară prevăzută cu accelerometru, giroscop și senzori de activitate musculară și care va fi amplasată pe antebrațul utilizatorului. În urma procesării datelor, se va dori recunoașterea unor categorii de gesturi folosind algoritmi de măsurare a similarității între două semnale și algoritmi de clasificare supervizată. Recunoașterea mișcării efectuate de antebraț va fi transpusă într-o mișcare efectuată de un braț robotic (Kinova Jaco robotic arm) programat să execute o serie de mișcări predefinite. Aplicația va fi dezvoltată în limbajele de programare C++ și Python astfel: procesarea datelor colectate de brățară și controlul brațului robotic în C++, iar modulul de recunoaștere a mișcărilor va fi implementat în Python/C++.

În final, se va obține un sistem care va fi capabil să colecteze date generate prin mișcarea brațului unui utilizator, să recunoască gestul pe care utilizatorul l-a efectuat și să redea mișcarea dorită cu ajutorul brațului robotic.

Categoriile de gesturi vizate de acest proiect sunt reprezentate de literele alfabetului latin. Pentru simplitate, vom folosi doar majusculele. Vom avea în total 26 de gesturi reprezentând desenarea în aer a literelor cu ajutorul mâinii.

După ce s-a realizat recunoașterea literei, ne propunem să desenăm pe o tablă albă litera corespunzătoare. Această etapă se va realiza cu ajutorul unui braț robotic. Aceasta are 6 grade de libertate și permite efectuarea unor acțiuni specifice brațului uman. Am ales un braț robotic deoarce acestea au o utilizare largă în asistența persoanelor cu dizabilități și s-au dovedit a fi foarte performante. Prin intermediul acestui proiect, vom realiza un sistem care va facilita integrarea în societate a persoanelor cu handicap. Aceștia vor putea realiza cu ușurință acțiuni din viața cotidiană pe care nu le puteau îndeplini fără asistența unei alte persoane.

Structură

Capitolul 1 cuprinde o descriere sumară a dispozitivelor hardware utilizate. Vom detalia informații tehnice despre brațul robotic Kinova Jaco^2 cât și despre brățara Myo. De asemenea vom prezenta modul de extragere a quaternionilor și de determinare a unghiurilor Euler pe baza acestora.

Capitolul 2 cuprinde o detaliere a algoritmului Dynamic Time Warping(DTW). Vor fi prezentate modalitățile prezentate pentru calculul distanțelor și determinarea minimului acestora.

Capitolul 3 va prezenta proiecția coordonatelor determinate anterior într-o imagine și apoi folosirea unei aplicații de tipul „Optical Character Recognition”(OCR) pentru recunoașterea literelor trasate. Vom folosi aplicația Tesseract și vom detalia particularitățile acesteia.

Capitolul 4 va sublinia principalele optimizări realizate în cadrul acestui proiect. Printre metodele utilizate putem enumera fereastra Sakoe-Chiba, abandonarea timpurie și divizarea sarcinii de calcul în fire de execuție.

Capitolul 5 va evidenția datele experimentale ale proiectului. Ulterior vom menționa rezultatele obținute și vom analiza performanțele sistemului în funcție de acestea.

Capitolul 1 Dispozitive Hardware

1.1 Brațul robotic Kinova Jaco^2

Compania Kinova Robotics, originară în Canada, inventează și implementează platforme robotice sigure si eficiente în 3 domenii: Industrial, prin intermediul platformelor ce pot executa sarcini diverse înlocuind factorul uman în situații periculoase și sporind eficiența; În domeniul medical proiectând platforme ajutătoare pentru persoane cu handicap pentru a efectua sarcini ce sunt in afara capabilității lor; Cercetare, oferind soluții flexibile pentru o gamă largă de proiecte.

Pentru această lucrare, am utilizat brațul robotic Kinova Jaco^2 lansat in 2010. Acesta prezintă 6 grade de libertate și este prevăzut cu o mână formată din 3 degete flexibile. Acest robot a fost creat cu scopul de a ajuta persoanele cu deficiență de mobilitate. Fiind proiectat să se monteze cu ușurință pe un scaun cu rotile, Jaco^2 se integrează cu succes în rutina zilnică pentru îndeplinirea sarcinilor elementare.

1.1.1 Specificații Kinova Jaco2

- 6 grade de libertate
- 3 degete flexibile
- structura din fibră de carbon
- 2 moduri de control: cartezian și unghiular
- senzori de temperatură, curent, accelerometru,poziție

- poate fi controlat folosind maneta de control sau programat
- conceput să poată fi montat pe aproape orice tip de scaun cu rotile electric

Figura 1. 1 Kinova Jaco²[1]

MECHANICAL	
TOTAL WEIGHT	**5.3 Kg**
MAXIMUM PAYLOAD	**2.5Kg mid-range** **1.5Kg full extension**
MAXIMUM REACH	**90 cm**
GEAR SYSTEM	
Type	**Harmonic Drive™**
Gear ratio	**1:136 (Large Actuators 1 and 3)** **1:160 (Large Actuator 2)** **1:110 (Small Actuators 4, 5 and 6)**
Back drivability	**When shutdown**
MAXIMUM LINEAR ARM SPEED	**20 cm/s**
AMBIANT TEMPERATURE	**-10 °C to 40 °C expected**
WATER RESISTANCE	**IPX2 expected**

Tabel 1. 1 Specificații Kinova Jaco²[1]

1.1.2. Braț

Brațul este format din 7 segmente între care se află 6 motoare de curent continuu fară colector, ultimul segment fiind reprezentat de mână și cele 3 degete. Fiecare segment este fabricat din fibră de carbon ce conferă brațului rezistență și eficiență. Axurile sunt formate din discuri compacte din aluminiu.

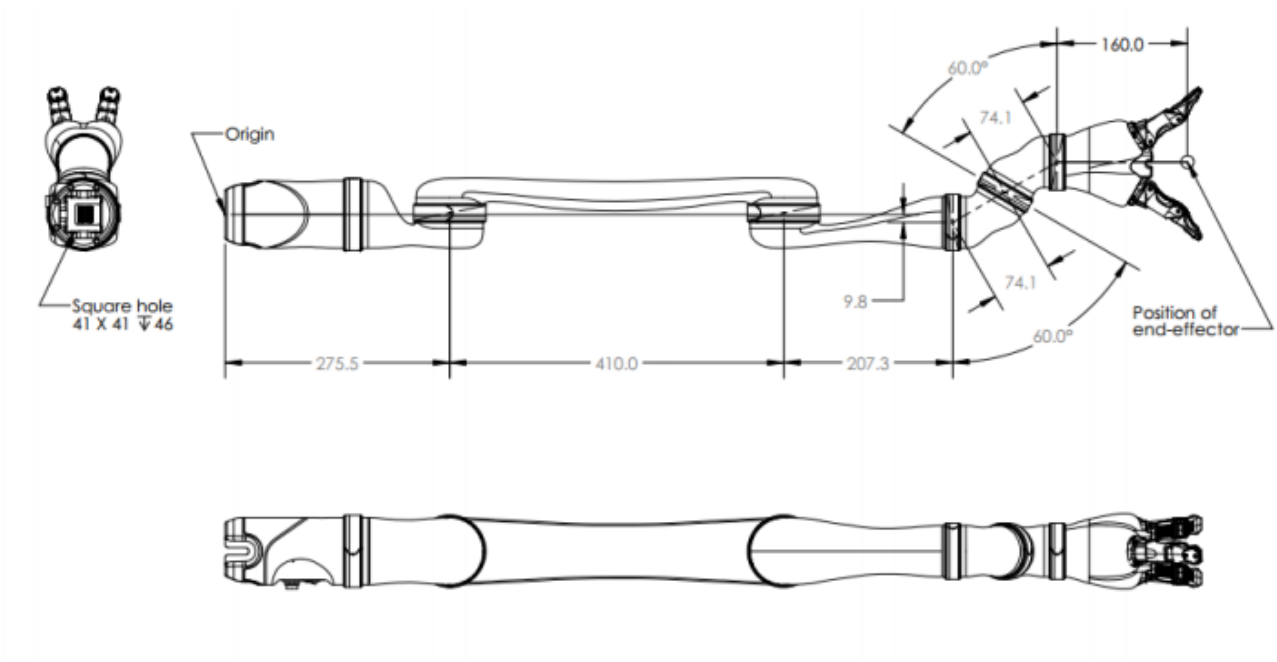

Figura 1. 2 Specificații Braț[1]

Fiecare motor acționează brațul într-o gamă largă de unghiuri, singura limitare fiind reprezentată de modul în care brațul a fost proiectat. Mai jos sunt prezentate intervalele petru fiecare actuator:

Actuator #	Minimum (degrees)	Maximum (degrees)
1	-10 000	10 000
2	50	310
3	19	341
4	-10 000	10 000
5	-10 000	10 000
6	-10 000	10 000

Tabel 1. 2 Valori maximale ale actuatoarelor[1]

1.1.3 Mâna KG-3

Mâna KG-3 este varianta cu 3 degete. Deschiderea și închiderea degetelor este controlată de 3 actuatoare liniare, unul pentru fiecare deget. Actuatoarele se situează în carcasa de fibră de carbon a mâinii. Degetele sunt dotate cu benzi de cauciuc cu aderență ridicată ce îi permit brațului să manevreze obiecte de forme și dimensiuni diferite cu usurință.

Figura 1. 3 Mâna KG-3[1]

	KG-3	KG-2
Fingers quantity	3	2
Total weight	727 g	556 g
Gripping force	40 N	25 N
Minimum object diameter for cylindrical grip	45 mm	55 mm
Maximum object diameter for cylindrical grip	100 mm	
Actuation system	Under-actuated	
Actuators	One per finger	
Actuators sensors	Current Temperature Rotational encoder	
Opening (fingertip)	175 mm	
Minimum object diameter for object-on-the-ground pinch	8 mm	
Opening or closing travel time	1.2 sec	
Operating temperature	-10°C to 40°C	

Tabel 1. 3 Specificații modele KG-3 si KG-2[1]

1.1.4 Moduri de control

Jaco[2] prezintă 2 moduri de control: unghiular si cartezian. Primul dintre acestea permite utilizatorului să controleze fiecare actuator independent.

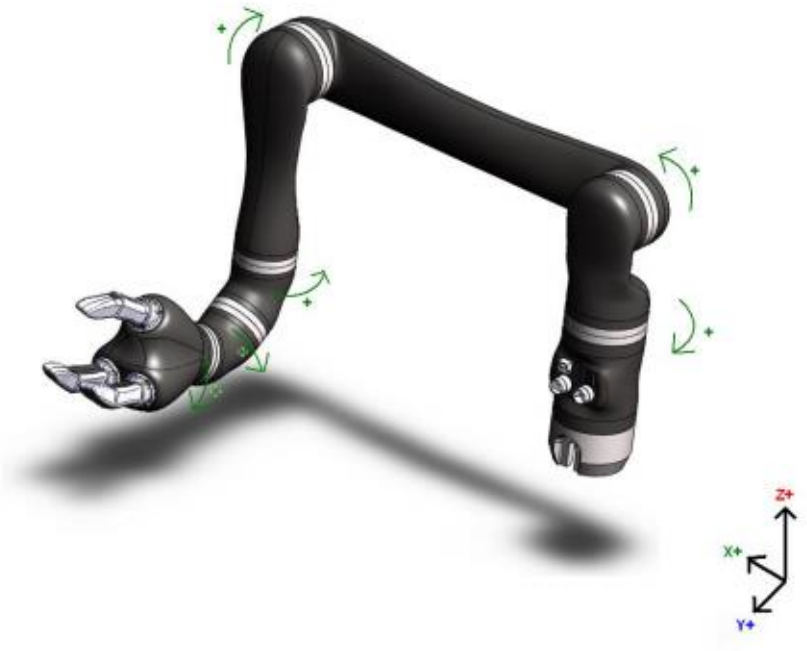

Figura 1. 4 Mod de control unghiular[1]

Modul cartezian este caracterizat de o mișcare complexă a actuatoarelor ce permite brațului să efectueze o deplasare de-a lungul axelor de coordonate. Mișcarea este formată din orientare și translație. La rândul său, modul cartezian poate fi implementat pe 2 sau 3 axe de coordonate. De asemenea, deplasarea de-a lungul axelor de coordonate prezintă atât un mod de control în funcție de poziție cât și în funcție de viteză. Prima metodă constă în specificarea unei poziții în spațiu determinată anterior urmând ca brațul să se deplasese către acea poziție în timp ce a doua metodă constă în translația sau orientarea brațului pe o anumită direcție cu o viteză stabilită pentru un interval de timp sau până la îndeplinirea unei condiții de stop.[2]

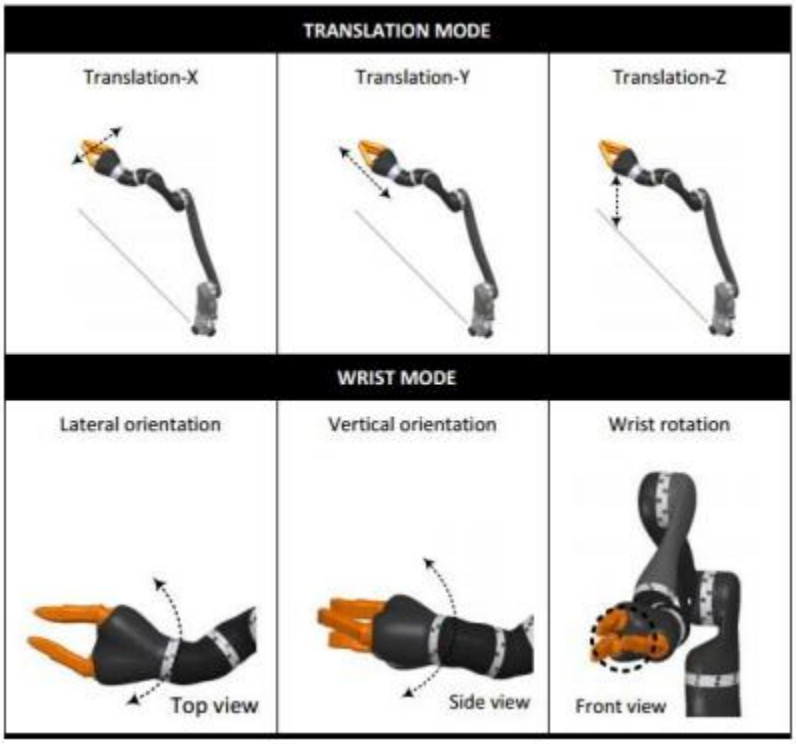

Figura 1. 5 Mod de control cartezian[2]

BLUE LIGHTS		CONTROL MODE
3-Axis	■□□□	Translation (X-Y-Z)
	□■□□	Wrist
	□□■□	Finger
	□■□■	Drinking mode (to use with wrist rotation mode)
	□□□□	Disabled controller
2-Axis	■□□□	Translation (X-Y)
	■■□□	Translation (Z) / Wrist Rotation
	□■□□	Wrist Orientation
	□□■□	Fingers
	□■□■	Drinking mode (to use with wrist rotation mode)
	□□□□	Disabled controller

Tabel 1. 4 Moduri de operare în cartezian[1]

1.1.5 Maneta de control

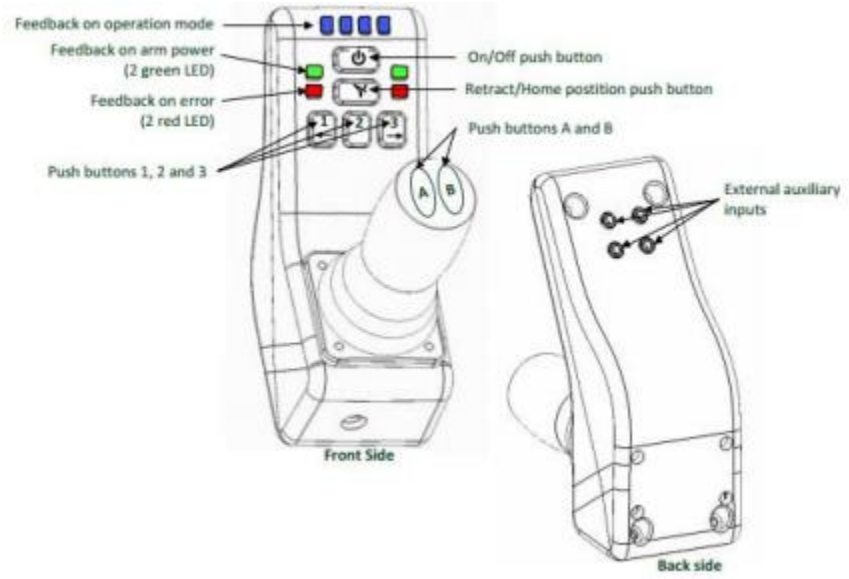

Figura 1. 6 Maneta de control[1]

Maneta de control este montată pe un suport ce prezintă 5 butoane independente, 4 intrari auxiliare si 4 LED-uri ce specifică utilizatorului modul de operare selectat. Comutarea între aceste moduri de operare se efectuează folosind butoanele “A” și “B” plasate în capătul manetei de control. Astfel, în modul de control unghiular, mișcarea manetei în poziție orizontală va controla actuatorul 1, pe vertical va controla actuatorul 2, iar prin răsucire va controla actuatorul 3. Pentru mișcarea celorlalte 3 actuatoare se va schimba modul de control și se va efectua tot prin intermediul manetei.[1]

În modul de control cartezian, acționarea brațului se realizează în 2 meniuri : unul dintre acestea va efectua translatări ale brațului pe 3 axe de coordinate, iar cel de-al doilea meniu va modifica orientarea brațului.

1.2 Brațara Myo

Brațara Myo produsă de compania Thalmic Labs este un dispozitiv complex conceput pentru a crea o interfață interactivă om-mașină. Aceasta permite utilizatorului să controleze diverse dispozitive electronice folosind o diversitate de mișcări ale mâinii putând fi folosită pentru a controla jocuri video, prezentări, muzică și diverse dispozitive inteligente.

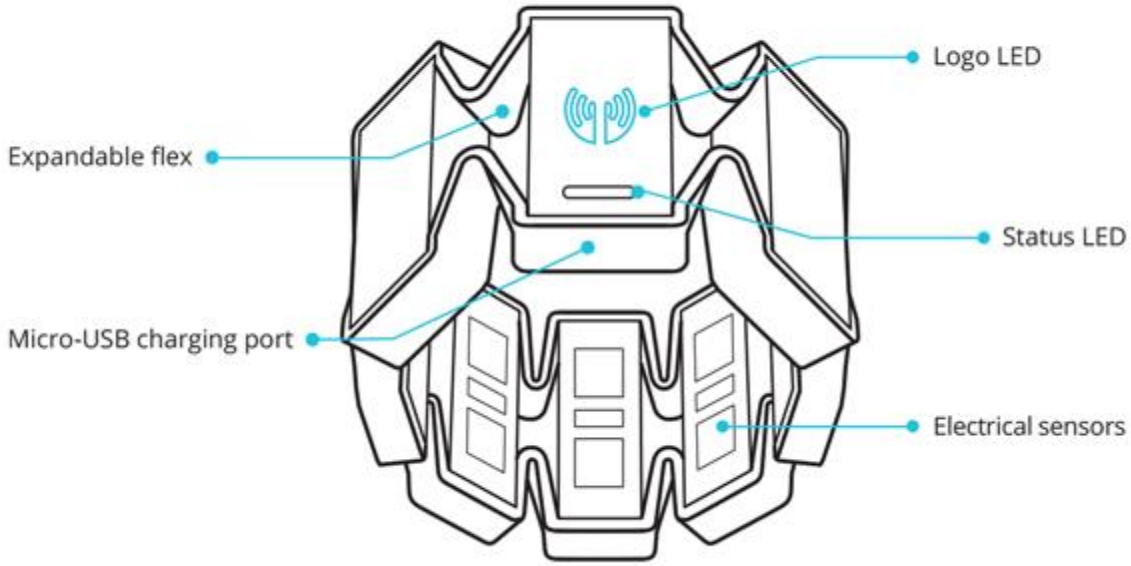

Figura 1. 7 Brațara Myo[4]

Myo este alcatuită din 8 segmente ce conțin senzori elecromiografici, accelerometru, giroscop, un port de încărcare USB și un LED ce indică starea în care brățara se află. Cele 8 segmente sunt interconectate prin material extensibil, permițând brățării să se adapteze în funcție de antebrațul utilizatorului (figura 2.7).

Brățara Myo se poate conecta la un dipozitiv (ex Laptop, Calculator, Telefon) prin intermediul unui modul Bluetooth 4.0 de joasă putere. Comunicarea cu dispozitivul este asigurată de SDK-ul brățării.

La bază, Myo oferă 2 tipuri de informație într-o aplicație: coordonate spațiale și recunoaștere de gesturi. Cei 8 senzori electromiografici sunt capabili să capteze semnalele electrice musculare de la nivelul antebrațului utilizatorului. Brațara este preantrenată pentru a fi capabilă să clasifice 5 gesturi pe care utilizatorul le efectuează cu brațul respectiv analizând formele de undă a celor 8 senzori musculari.[4]

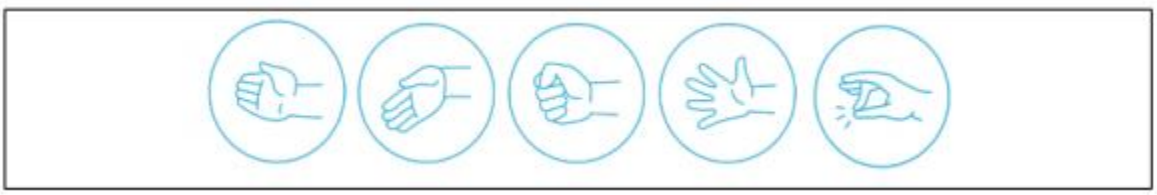

Figura 1. 8 Gesturi predefinite ale brățării Myo

În această aplicație, se vor folosi doar coordonatele spațiale puse la dispoziție de către brațara Myo, folosind accelerometrul și giroscopul integrate, sub formă de quaternioni. În matematică, quaternionii reprezintă un sistem de numere ce extind numerele complexe. Aceștia au fost prima dată descriși de către matematicianul William Rowan Hamilton în 1843. [5] Forma uzuală a acestora este:

$$q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} \quad \textbf{(2.1)}$$

unde w, x, y și z sunt numere reale iar **i**, **j**, **k** sunt unități fundamentale ale quaternionilor.

Quaternionii au aplicații atât în matematica teoretica cât și în cea aplicată, fiind în general utilizați pentru a reprezenta rotații tridimensionale ale obiectelor in spațiu. Caracteristica principală a quaternionilor este dată de formula:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \quad \textbf{(2.2)}$$

ce stă la baza tuturor aplicațiilor ce folosesc acest sistem de numere.

Pentru o reprezentare mai simplă a orientării brațului în spațiu, vom converti quaternionii puși la dispoziție de către brățara Myo [w, x, y, z] în unghiuri Euler [Φ, θ, ψ].

$$\phi = \arctan\frac{2(wx + yz)}{1 - 2(x^2 + y^2)} \quad \textbf{(2.3)}$$

$$\theta = \arcsin(2(wy - xz)) \quad \textbf{(2.4)}$$

$$\psi = \arctan\frac{2(wz + xy)}{1 - 2(y^2 + z^2)} \quad \textbf{(2.5)}$$

Capitolul 2 DTW

DTW este un algoritm ce masoară similaritatea între 2 secvențe temporale, ce pot varia în viteză. De exemplu, similaritățile între modurile în care două persoane distincte se deplasează poate fi detectată folosind DTW chiar dacă o persoană merge mai rapid decât cealaltă sau chiar dacă pe parcursul observației au existat accelerări sau decelerări în modul de deplasare al persoanelor. Această metodă a fost intens studiată într-o gamă largă de aplicații, fiind folosită pentru compararea secvențelor temporale audio si video, clasificare, găsirea secvențelor asemănătoare sau clusterizare. La bază, DTW poate fi folosit în cadrul oricărei aplicații ale cărei date se pot transforma într-o secvență liniară. [8]

Probabil cea mai des utilizată aplicație a DTW este recunoașterea de vorbitor, pentru a determina daca 2 secvențe temporale aparțin aceleiași persoane. Într-o astfel de secvență, durata pronunțării fiecarui sunet și distanța între sunete succesive pot varia, însă forma de undă generală trebuie sa fie similară. Alte domenii sunt reprezentate de recunoaștere de gesturi,minare de date, robotică, medicină, meteorologie, prelucrare de imagini, seismologie, finanțe, etc.

O secvență temporală reprezintă o colecție de observații asupra unui eveniment pe o anumită perioadă de timp. Fiecare observație în parte, în majoritatea cazurilor, nu oferă suficiente informații pentru a descrie un fenomen, însă, prin trasarea graficului a tuturor valorilor înregistrate se pot observa diverse caracteristici ce au o însemnătate în domeniul respectiv. De exemplu, în figura 3.1 este reprezentată o secvență temporală a bătăilor inimii unui individ. Fiecare element din vectorul afișat reprezintă valoarea captată de senzorul optic al aparatului, ce masoară variația volumului de sânge prin organism rezultând o variație a intensității luminoase.

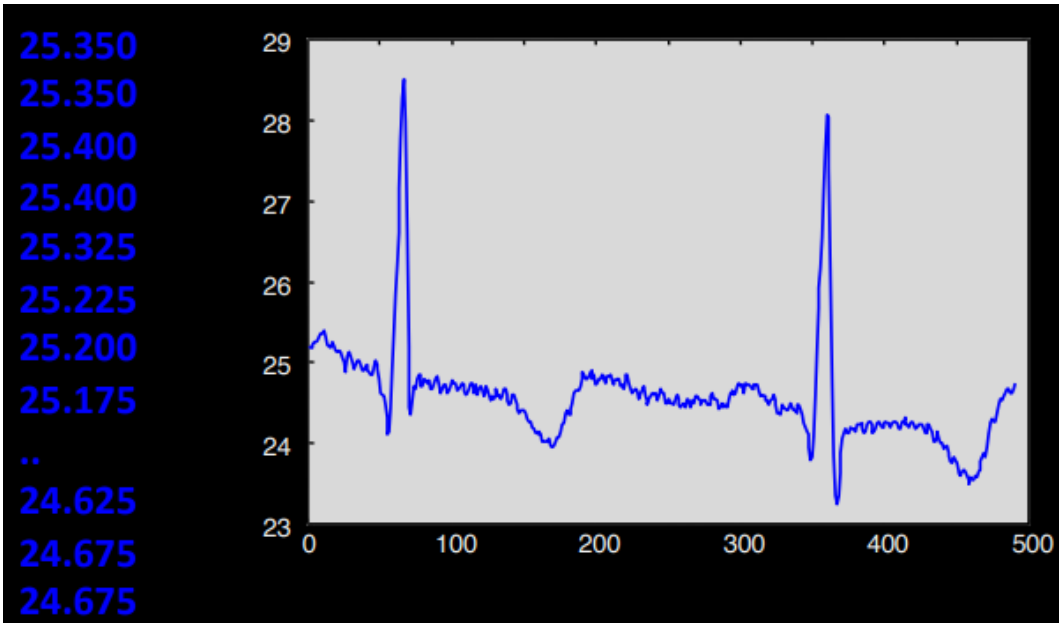

Figura 2. 1 Bătăile inimii [7]

Fiecare valoare în parte nu oferă nici o informație, însă prin plasarea acestora într-un grafic se poate analiza forma de undă și chiar determina eventuale boli pe care pacientul le-ar putea suferi[7].

2.1 Calcul DTW

Pentru a putea compara 2 secvențe temporale, $Q = [Q_0,Q_1,Q_{2,\dots},Q_n]$ și $C = [C_0,C_1,C_2,\dots,C_m]$ se creează o matrice cu laturile egale cu lungimile celor 2 serii ce va conține toate distanțele posibile între 2 perechi de puncte ale celor 2 semnale:

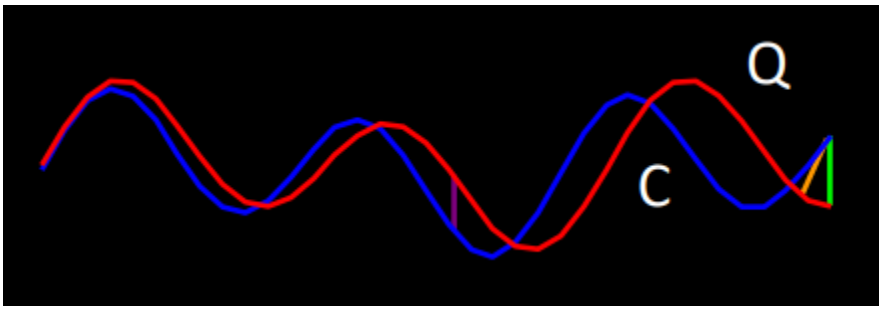

Figura 2. 2 Suprapunerea a 2 semnale [7]

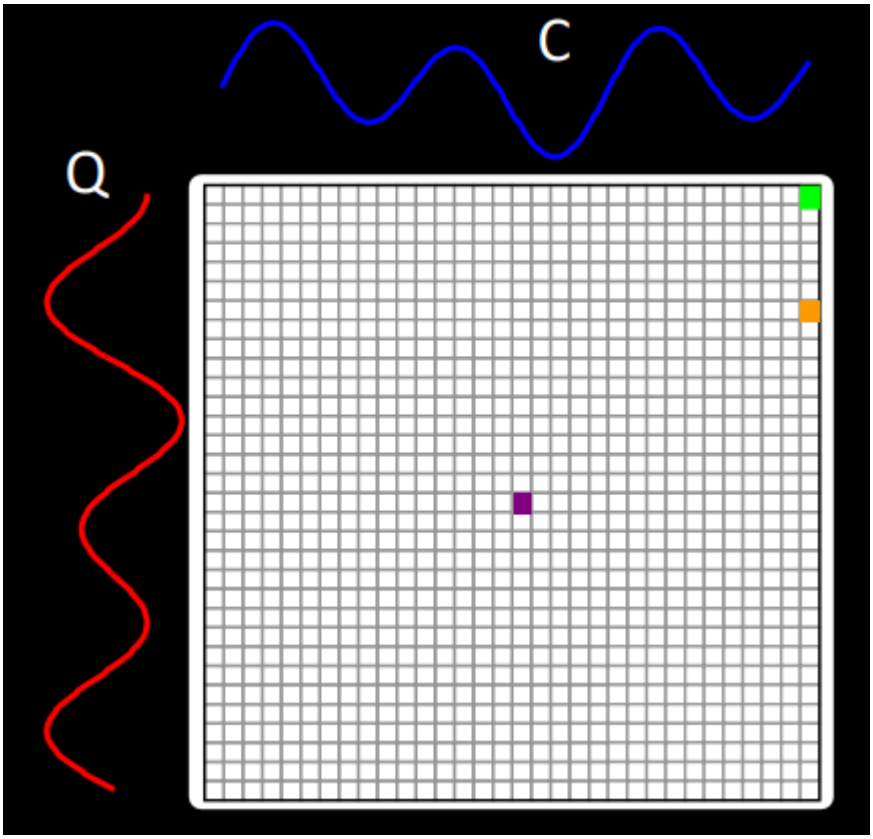

Figura 2. 3 Matricea DTW[7]

Fiecare valoare din matrice se va calcula după următoarea ecuație:

$$\Upsilon(i,j) = d(Q_i,C_j) + \min\{ \Upsilon(i-1, j-1),\ \Upsilon(i-1, j),\ \Upsilon(i, j-1)\} \qquad \textbf{(3.1)}$$

unde Υ reprezintă matricea DTW iar $d(Q_i,C_j)$ reprezintă distanța între cele 2 puncte.

Calculul matricei DTW va începe din punctul de coordonate $\Upsilon(0,0)$ ce va avea valoarea diferenței în modul a primului element din fiecare serie temporală : Q_0 și C_0. Fiecare punct al matricei se va calcula ca suma a 2 componente:

- distanța dintre coordonatele aferente fiecărui vector în punctul respectiv
- minimul dintre punctele vecine ale matricei DTW calculate anterior.

După calculul întregii matrice, corespondența între cele 2 serii temporale este dată de calea ce reprezintă distanța cea mai mica:

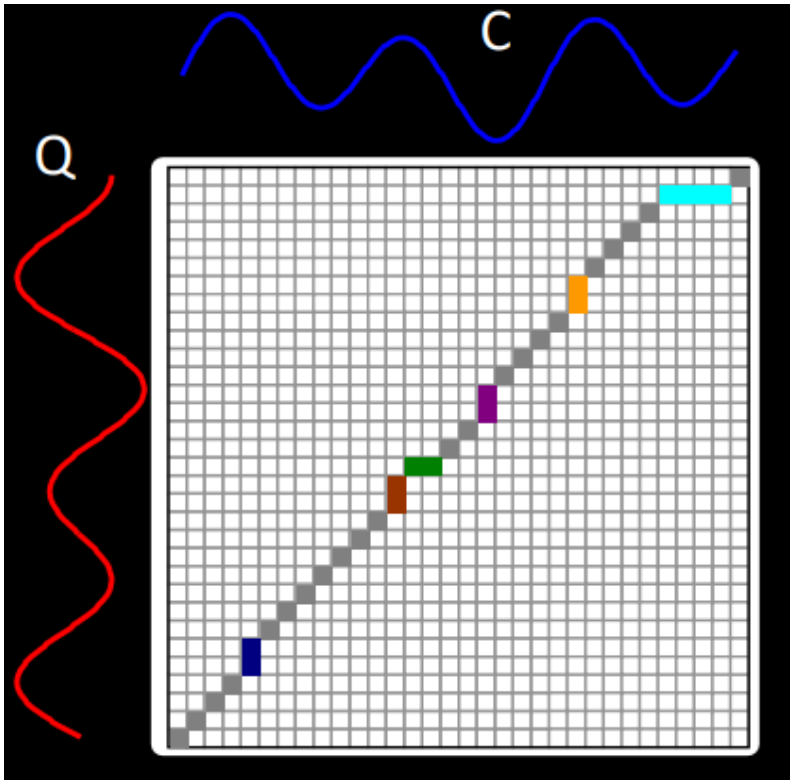

Figura 2. 4 Calcul distanță minimă[7]

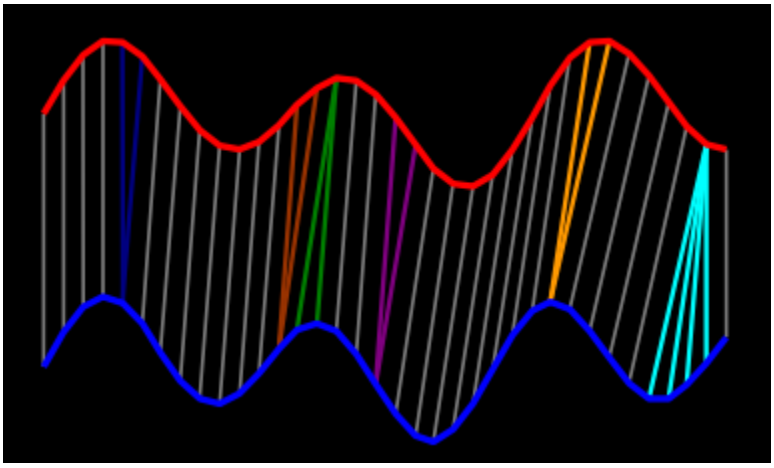

Figura 2. 5 Corespondența DTW[7]

Corespondența între 2 serii temporale poate fi efectuată prin orice cale ce unește colțul din stânga jos al matricii cu cel din dreapta sus, însă acestea nu corespund distanței minime. O reprezentare tridimensională a valorilor matricii DTW și determinarea distanței optime dintre 2 semnale este reprezentată în figura 2.6.

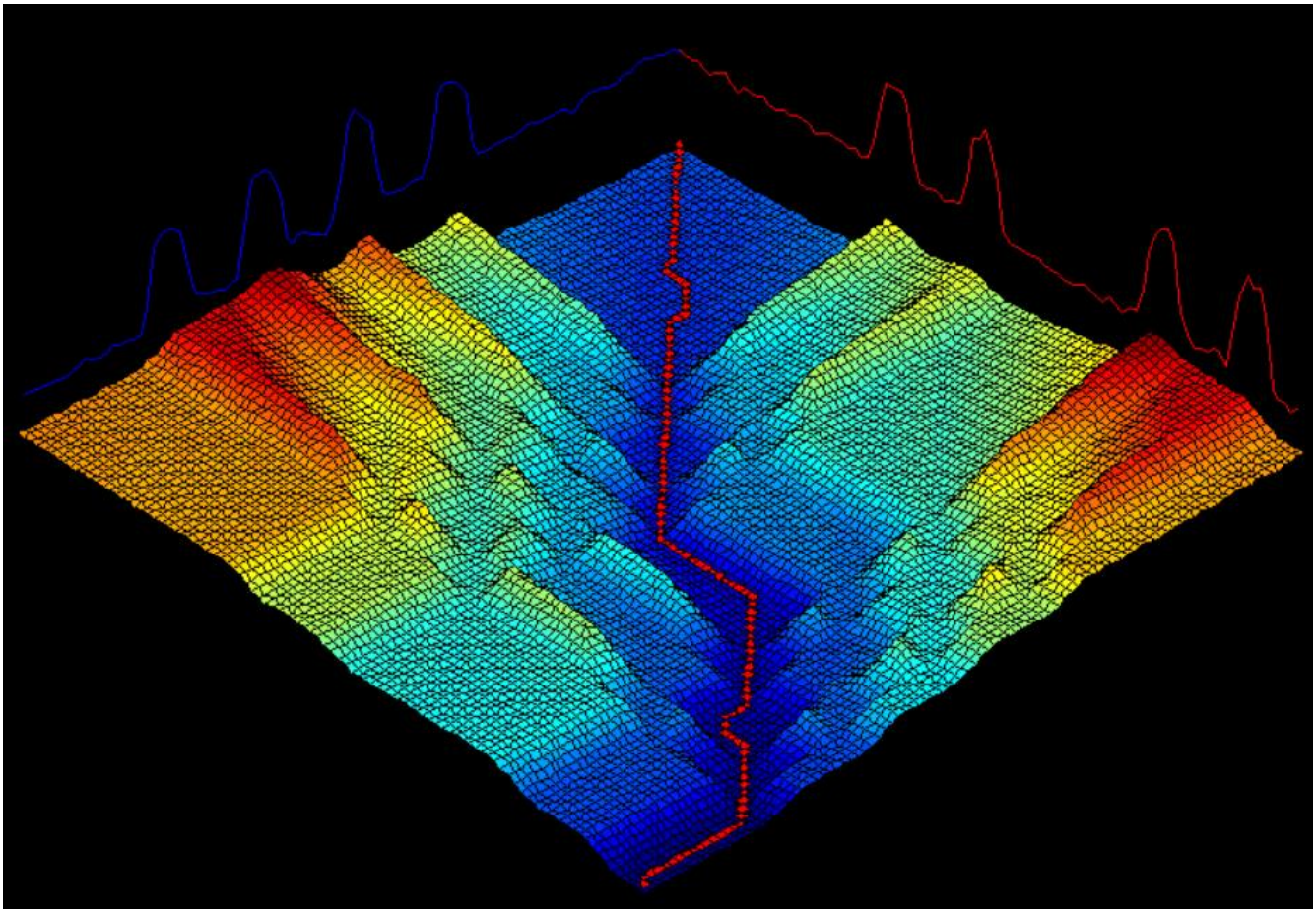

Figura 2. 6 Reprezentare 3D DTW

Este cunoscut faptul că DTW este superior distanței euclidiene pentru clasificarea și clusterizarea seriilor temporale. Un prim argument este faptul că în cazul distanței euclidiene, cele 2 serii temporale trebuie să fie de aceeași lungime. În acest caz, este nevoie de interpolarea unuia dintre semnale până la obținerea lungimii celuilalt semnal. DTW nu prezintă astfel de restricții, fiind capabil să clasifice serii temporale de lungimi diferite. Până de curând, majoritatea proiectelor de cercetare foloseau distanța euclidiană pentru clasificare deoarece reprezenta o metodă mai rapidă. Complexitatea atât în spațiu cât și în timp a algoritmului DTW este $O(n^2)$, fiind astfel un algoritm lent și ce necesită resurse considerabile. Însă, recentele optimizări ale algoritmului au plasat DTW pe prima poziție în comparația seriilor temporale. Vom discuta despre astfel de optimizări în capitolul următor.

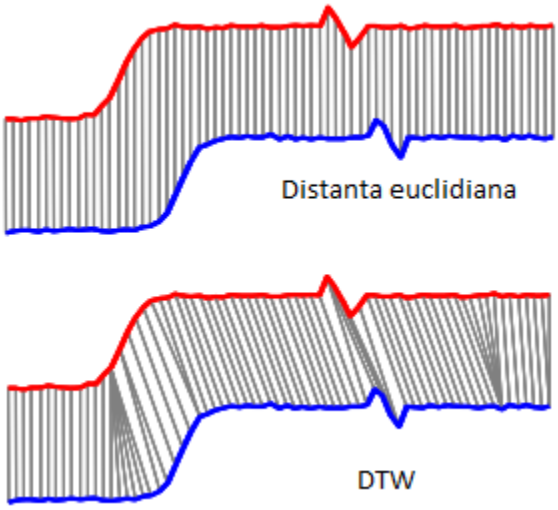

Figura 2. 7 Comparație DTW și Distanța euclidiana

2.2 Implementare DTW

În această aplicație, vom folosi DTW pentru clasificarea literelor pe care utilizatorul brățării Myo le va desena. Myo reprezintă o sursă continuă de informații, transmițând date sub formă de quaternioni [w, x, y, z] urmând apoi calculul unghiurilor Euler [Φ, θ, ψ] prin procedeul prezentat în capitolul anterior.

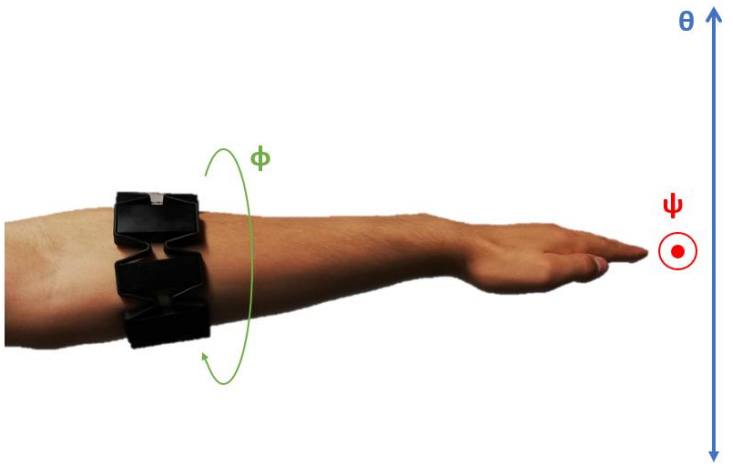

Figura 2. 8 Unghiuri Euler

Pentru reprezentare, valorile fiecărui unghi vor fi scalate în intervalul [-50,50]. Valoarea 50 a unghiului θ va reprezenta o poziție a brațului pe verticală orientat în sus iar -50 va reprezenta valoarea opusă pe aceeași axă. Analog, ψ va reprezenta mișcarea pe orizontală iar Φ reprezenta rotația antebrațului(figura 3.8). Fiecare punct al literei va fi caracterizat doar prin 2 din cele 3 coordonate: θ și ψ.

Captarea unei litere se efectuează prin alcătuirea a 2 vectori la care se vor adăuga valori succesive a celor 2 coordonate menționate într-un interval de timp. Fiecare literă va fi reprezentată de perechi de astfel de vectori, lungimea acestora fiind diferită de la o literă la alta, și chiar și între 2 reprezentări succesive ale aceleiași litere. Astfel, și in cazul acestei aplicații DTW reprezintă o variantă mai bună decât distanța euclidiană.

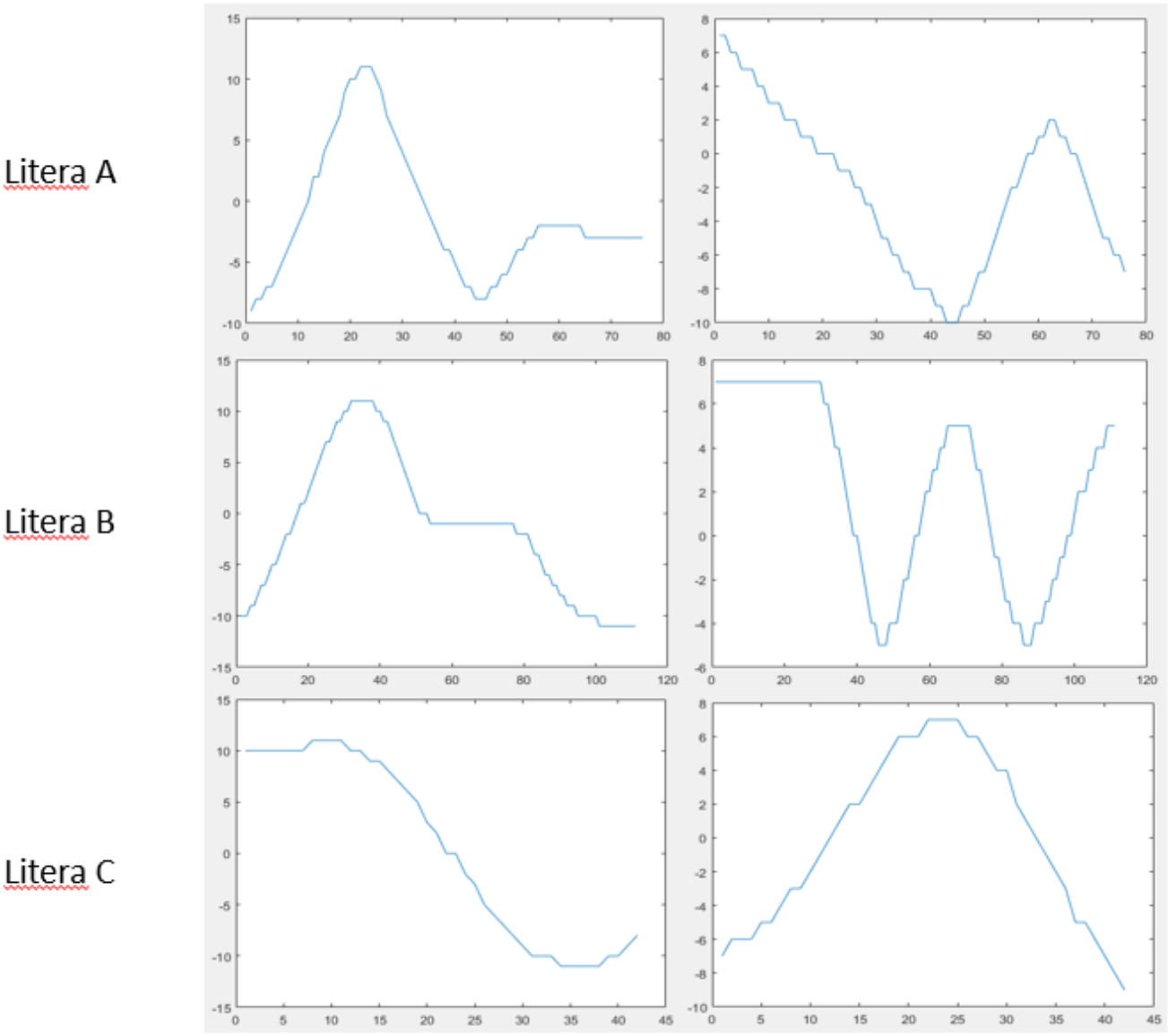

Figura 2. 9 Reprezentare grafică litere

În figura 2.9 sunt reprezentate perechile de formă de undă a literelor A, B si C. Capturile din partea stângă reprezintă deplasarea pe verticală (valoarea unghiului pitch) iar capturile din partea dreaptă reprezintă deplasarea pe orizontală (valoarea unghiului yaw). Se poate observa că secvențele temporale ale fiecărei litere este în general diferită. Se obțin durate secvențiale diferite și între 2 desenări succesive ale aceleiași litere (figura 2.10), durata și forma acestora depinzând de modul în care utilizatorul va desena litera la momentul respectiv.

Distanța dintre 2 litere va fi calculată aplicând DTW pentru fiecare dintre cele 2 axe și adunând distanțele cumulate obținute.

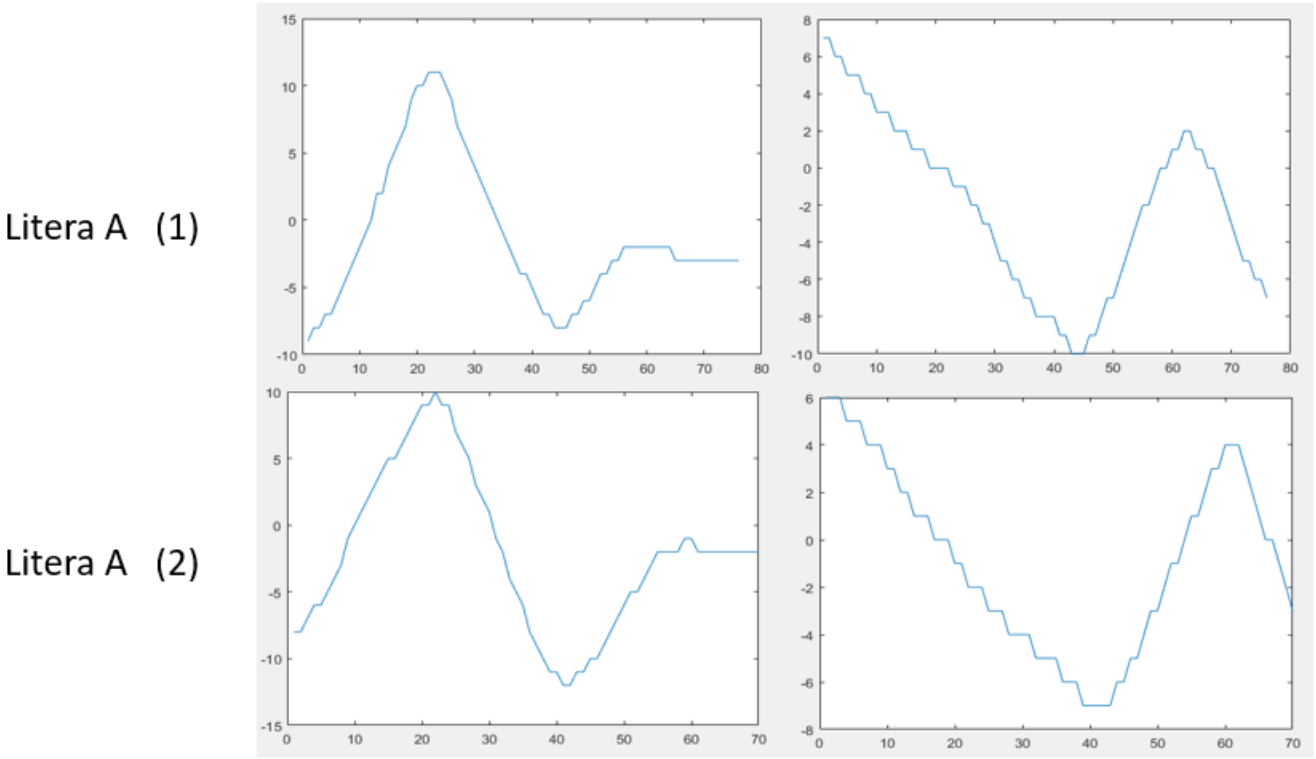

Figura 2. 10 Diferențe în durată și formă

Pentru recunoașterea literei trasate, se calculează distanța DTW de la aceasta la fiecare literă din baza de date. Litera recunoscută va fi reprezentată de acea literă din baza de date către care s-a obținut distanța minimă comparativ cu celelalte.

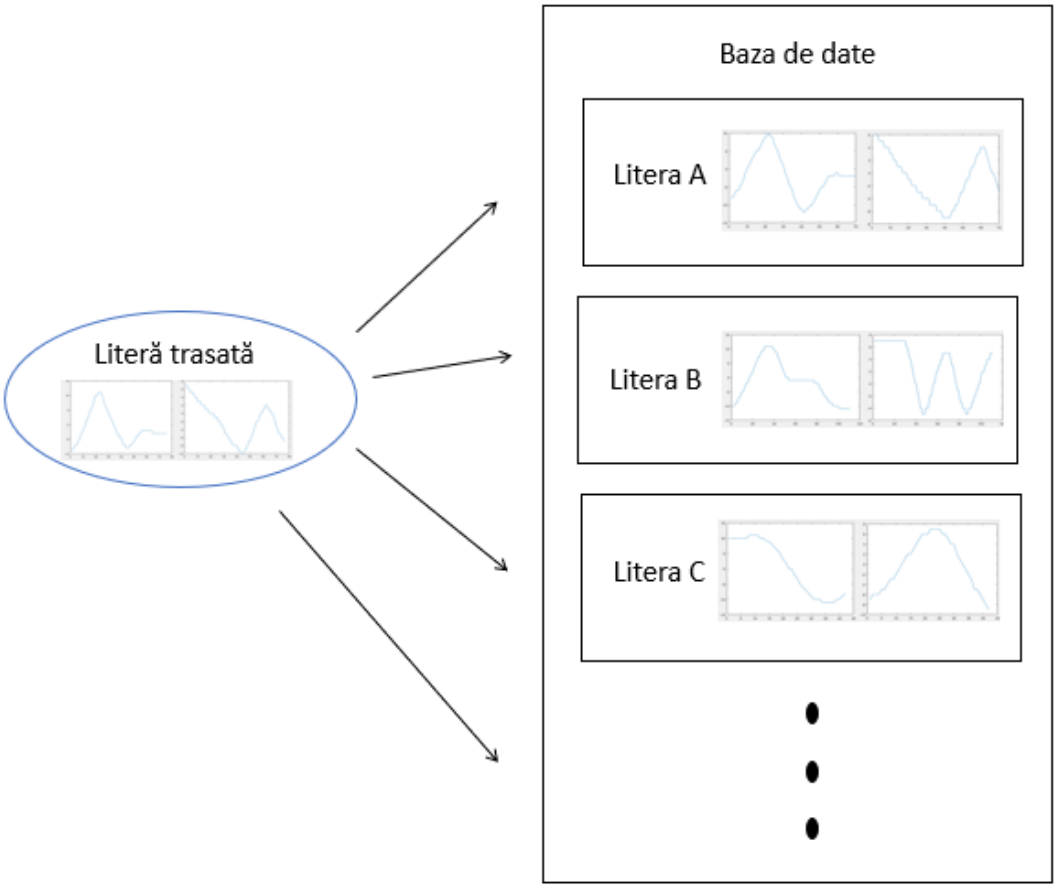

Figura 2. 11 Calcularea distanțelor DTW

Capitoul 3 Recunoaștere pe imagine

A doua metodă de recunoaștere a literelor utilizată constă în proiecția coordonatelor captate de brațara Myo într-o matrice bidimensionala, urmând apoi recunoașterea literei desenate în imaginea nou creată.

3.1 OpenCV

OpenCV este o librărie software complexă cu aplicații diverse în prelucrare de imagini și inteligență artificială. Aceasta conține aproximativ 2500 de algoritmi optimizați ce acoperă o gamă largă de functionalități, de la elemente de bază până la algoritmi de ultimă generatie în materie de prelucrare și analiză de imagini. OpenCV este dezvoltată într-o varietate de limbaje de programare precum C++, Python, Java și Matlab și este compatibil cu Windows, Linux, Android si MacOS.[9]

În această aplicație, OpenCV este utilizat pentru a crea imagini binare în funcție de datele primite de la brățara Myo. Pentru a desena litere în imagine se folosesc proiecțiile a 2 dintre cele 3 unghiuri Euler calculate anterior. Valoarea Φ(pitch) va reprezenta deplasarea pe verticală în timp ce
ψ(yaw) va reprezenta deplasarea pe orizontală, punctul de coordonate (0,0) fiind plasat în centrul imaginii(figura 3.1).

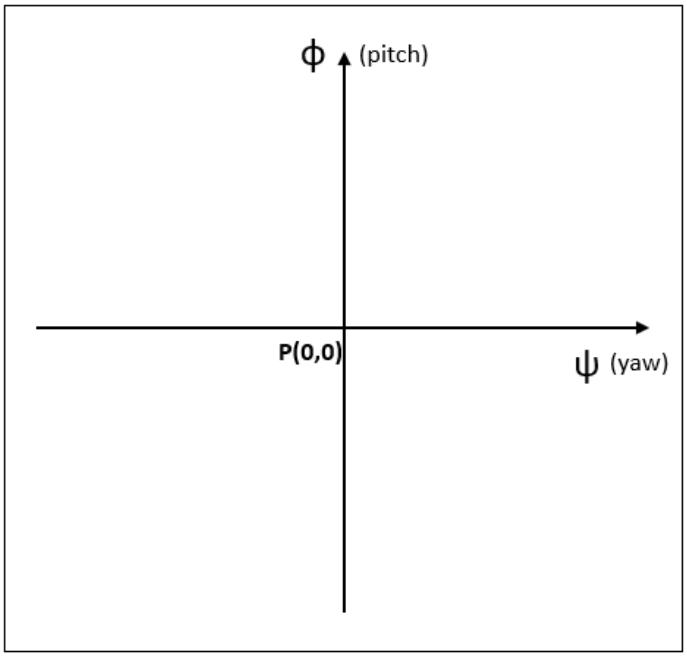

Figura 3. 1 Proiecția coordonatelor în imagine

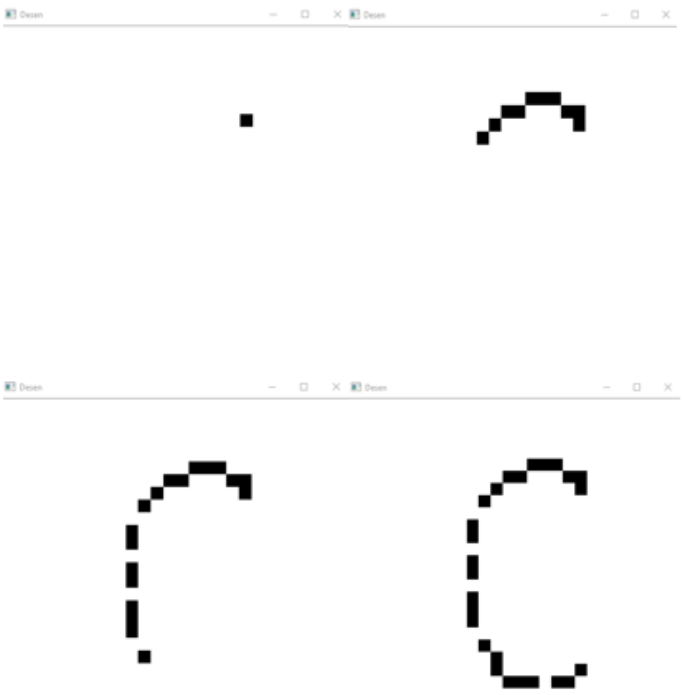

Figura 3. 2 Desenare litere

3.2 Tesseract

OCR(recunoașterea optică a caracterelor) s-a dezvoltat în secolul 20 cu scopul principal de a ajuta persoanele cu deficiențe de vedere, fiind capabil să descifreze text urmând ca acesta să fie redat pe cale auditivă.

Tesseract este o aplicație de recunoaștere optică a caracterelor și a fost lansat sub licența Apache, versiunea 2.0, dezvoltarea acesteia fiind sponsorizată de compania Google începând cu anul 2006 [9] . Aceasta este valabilă pe diverse sisteme de operare precum Linux, Windows si MAC OS X, însă, din cauza resurselor limitate, dezvoltarea aplicației este axată preponderent pe Windows și distribuția Ubuntu a sistemului de operare Linux. Ultima versiune, versiunea 4, poate recunoaste până la 116 limbaje și suportă mai multe tipuri de formate ale imaginilor.

Tesseract poate fi rulat direct din linia de comandă sau se poate integra în aplicații folosind librăria Leptonica. Acesta este capabil să descifreze text linie cu linie folosind o imagine scanată a paginii. Performanța algoritmului scade drastic dacă imaginea de intrare nu este preprocesată pentru a respecta condițiile acestuia: imaginile trebuiesc scalate astfel încât înălțimea literelor să fie de cel puțin 20 de pixeli, orice rotație a literelor trebuie corectată, iar tranzițiile de lumină vor fi eliminate în urma unui proces de binarizare a imaginii. [12]

De asemenea, Tesseract nu este capabil să descifreze text ale carui litere prezintă discontinuități. De aceea, pentru recunoașterea literelor a fost nevoie de interpolarea coordonatelor obținute anterior cu ferestre de 2x2 sau 3x3 pixeli.

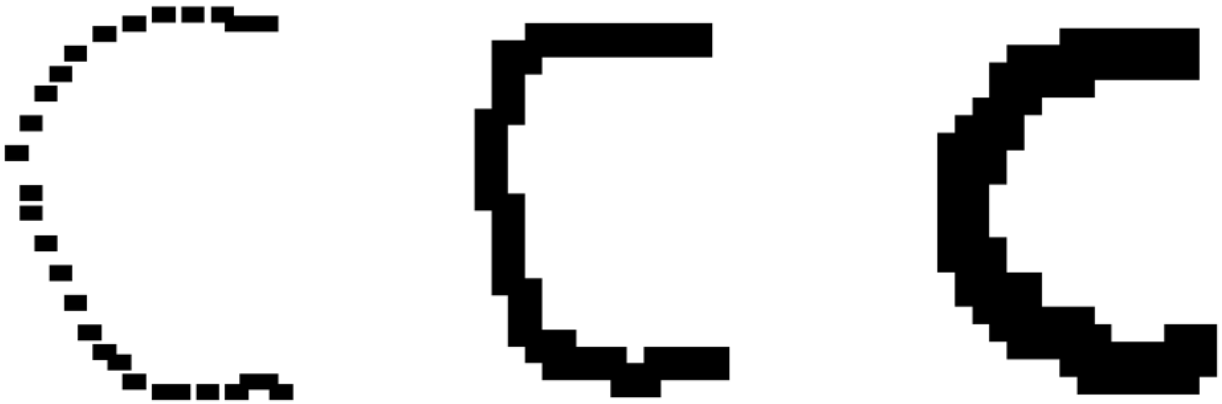

Figura 3. 3 Interpolare valori

Capitolul 4 Optimizări algoritm

4.1 Optimizare DTW

4.1.1 Normare la primul punct

Reprezentarea literelor în planul format de axele **{θ , ψ}** poate varia considerabil,coordonatele fiind reprezentate în coordonate absolute. O aceeași literă desenată succesiv poate conține valori complet diferite față de versiunea precedentă, acest fenomen fiind cauzat de alegerea diferită a punctului inițial în care se începe trasarea literei. Deși variația în timp a coordonatelor coincide, componenta continuă ce separă cele 2 litere va avea o pondere considerabilă in calculul distanței DTW.

Pentru evitarea acestui fenomen, vom norma toate celelalte coordonate ale fiecărei litere la primul punct. Se obține astfel un sistem de reprezentare relativ la prima poziție X[0,0] (figura 4.2).
Prin această metodă, se elimină componenta continuă de separare a celor 2 reprezentări, punându-se accentul numai pe diferențele în variația în timp a celor 2 coordonate. Astfel, distanța DTW între 2 elemente ce aparțin aceleiași clase se micșorează.

Figura 4. 1 Deplasamentul reprezentărilor succesive

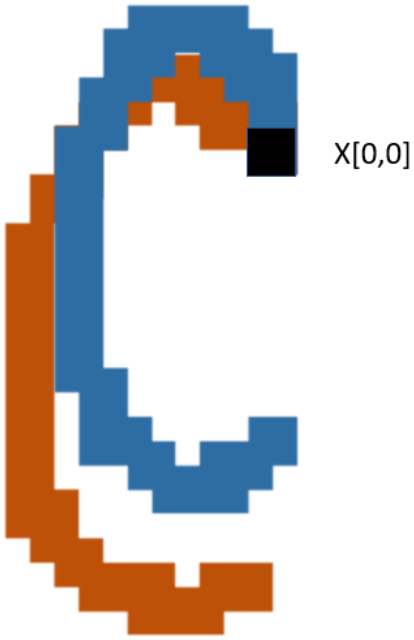

Figura 4. 2 Normarea la primul punct

4.1.2 Împărțirea spațiului în cadrane

O consecință a optimizării anterioare o reprezintă îmbunătățirea separabilității între clase. În urma normării la primul punct, vectorii de coordonate ce aparțin în clase diferite vor fi diferențiați cu o precizie mult mai bună. Se creează o impărțire a spațiului de reprezentare în 4 cadrane.(figura 4.3)
Se poate observa faptul că distanța între 2 vectori ce fac parte din cadrane diferite este considerabil mai mare decât distanța între vectori ce aparțin aceluiași cadran.

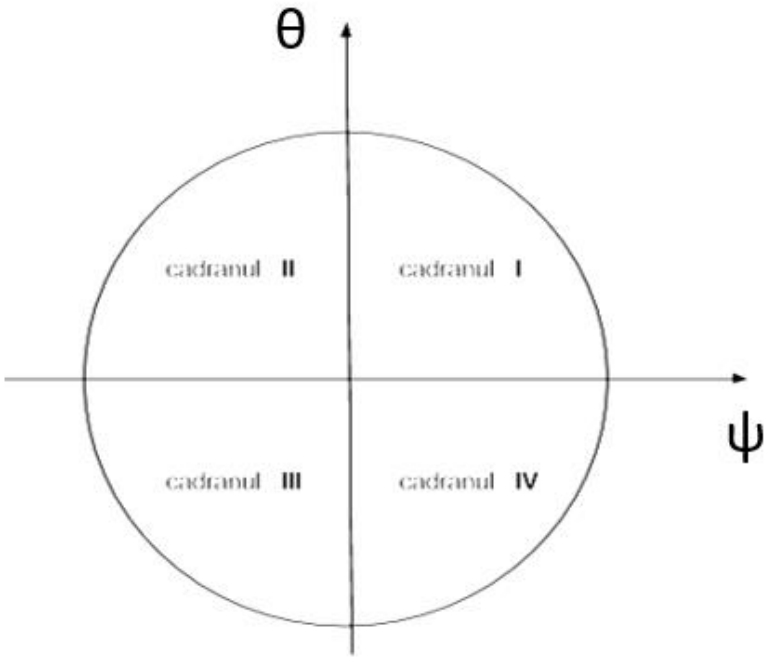

Figura 4. 3 Împărțirea în cadrane a spațiului de reprezentare

În recunoașterea literelor folosind DTW, ordinea de trasare a punctelor caracterului respectiv va influența considerabil secvența temporală a acestuia. Se încearcă plasarea vectorilor cu caracteristici asemănatoare în cadrane diferite. De exemplu, literele ‘D’ și ’O’, în transpunerea într-o imagine, vor avea o formă asemănătoare. Însă prin alegerea unui mod de trasare diferit pentru fiecare literă(figura 4.4), se obține o separabilitate clară între cele 2 (figura 4.5).

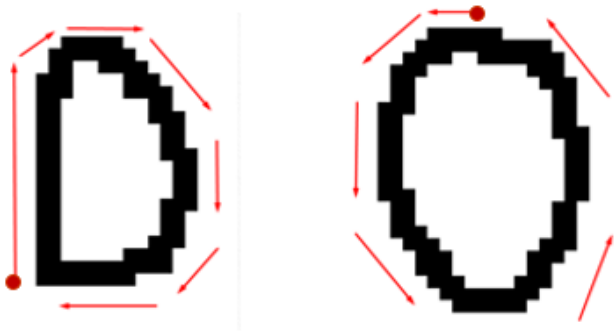

Figura 4. 4 Sensul de trasare

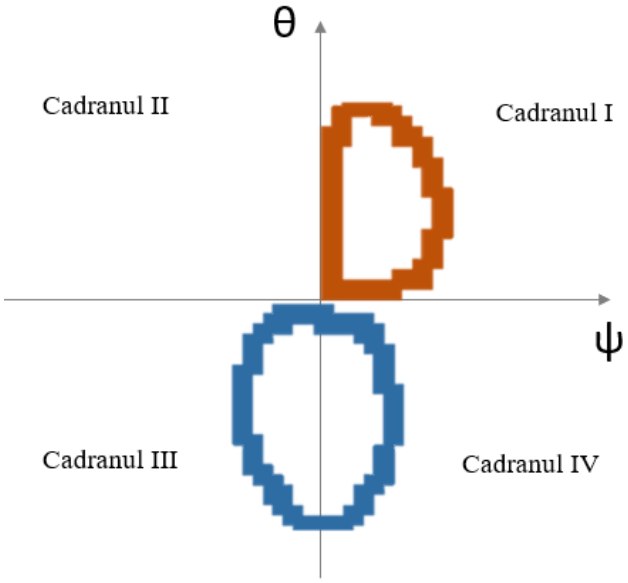

Figura 4. 5 Reprezentarea în planul de coordonate

4.1.3 Optimizarea volumului de calcul: Banda Sakoe-Chiba

Dupa cum am precizat și în capitolul 3, complexitatea de calcul a algoritmului DTW atât în spațiu cât și în timp este $O(n^2)$. În literatura modernă, metodele de îmbunătățire a timpului de execuție se împart în 3 domenii de interes:

- Constrângeri - limitarea numărului de elemente ale matricei DTW ce vor fi evaluate
- Abstractizarea datelor - aplicarea DTW pe o reprezentare redusă a datelor
- Indexare – Folosirea unor funcții de limitare inferioară pentru a reduce numărul de repetări ale algoritmului în timpul clasificării sau clusterizării.

În această aplicație, se va urmări scăderea volumului de calcul prin aplicarea unor constrângeri. Una dintre cele mai cunoscute metode în acest scop este banda Sakoe-Chiba, implementată și studiată de Hiraoki Sakoe și Seibi Chiba în lucrarea "Dynamic Programming Algorithm Optimization for Spoken Word Recognition"[10]. Această metodă constă în evaluarea elementelor din matricea DTW ce se află în vecinătatea diagonalei principale, dimensiunea vecinătății fiind specificată de un parametru variabil.

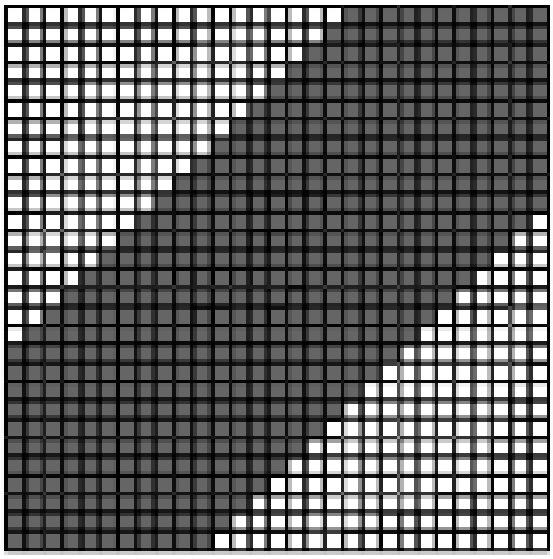

Figura 4. 6 Banda Sakoe-Chiba [11]

La aplicarea algoritmului, se va căuta calea optimă prin secțiunea delimitată de fereastra aleasă, ignorând calculul celorlalte elemente ale matricii.Totuși, calea optimă globală nu va fi găsită dacă nu se află în totalitate în interiorul ferestrei. Această constrângere oferă rezultate foarte bune în domeniile în care se estimează o cale optimă relativ lineară între cele 2 serii temporale, aceasta trecând prin matricea de cost DTW în linie dreaptă. În caz contrar, constrângerea are o performanță foarte scăzută în cazul în care cele 2 serii temporale prezintă o cale minimă ce se depărtează semnificativ de diagonala principală.

În cazul prezentat, aplicarea acestei ferestre în recunoașterea literelor oferă rezultate foarte bune. Distanța dintre desenări succesive ale unei litere rămâne aceeași dacă se respectă ordinea de trasare și dacă se alege o lățime a ferestrei suficient de mare pentru a acoperi variații ale trasării. Calculul distanței între 2 litere diferite se va depărta considerabil de diagonala principală, obținându-se astfel o separabilitate mai bună a claselor.

4.1.4 Optimizarea volumului de calcul: Abandonarea timpurie

La recunoașterea unei noi litere trasate, se calculează distanța DTW de la aceasta la cele 26 de litere din baza de date. Fiecare calcul al distanțelor este un proces iterativ cumulativ. Dacă distanța în curs de procesare de la litera trasată la o literă din baza de date depășește optimul determinat până la momentul respectiv, acest calcul se poate abandona. Orice calcul ulterior este irelevant deoarece este imposibil să se obțină o distanță mai mică, acest calcul fiind unul aditiv.

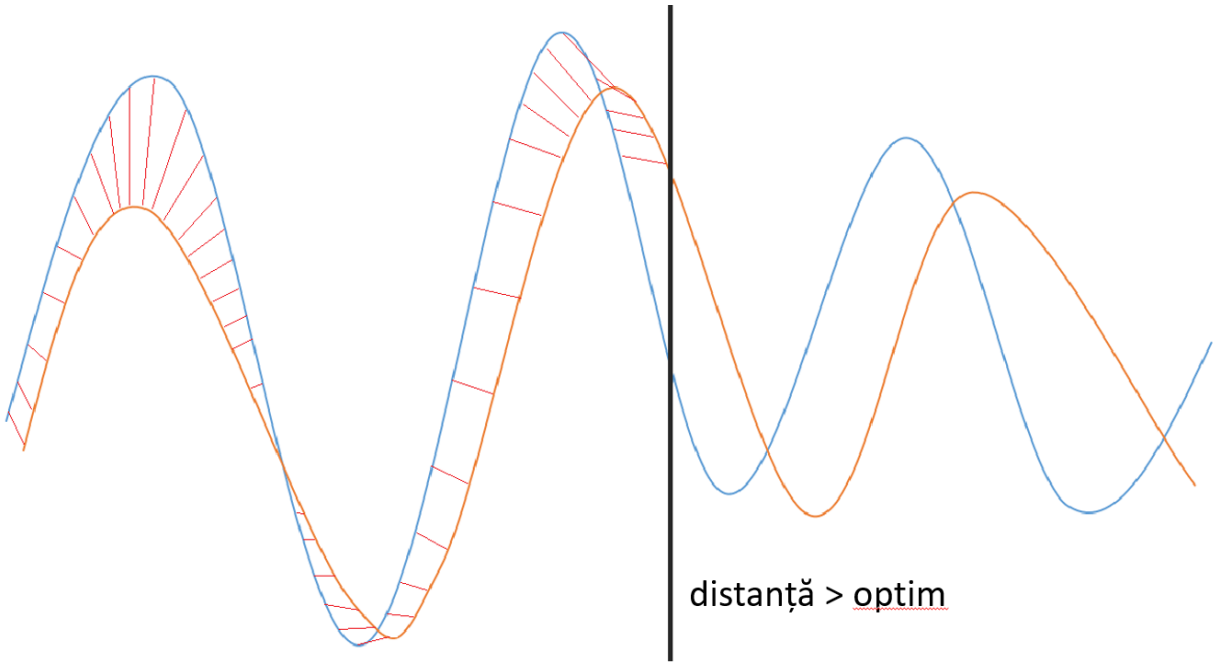

Figura 4. 7 Abandonare timpurie – reprezentare grafică

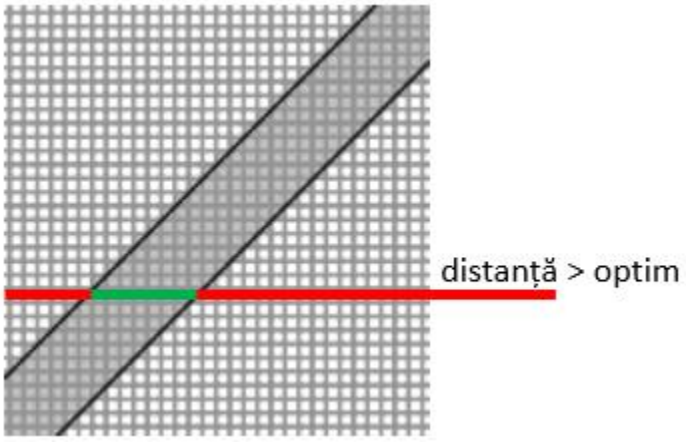

Figura 4. 8 Abandonare timpurie – reprezentare matriceală

4.1.5 Împărțirea sarcinii de calcul în multiple fire de execuție

Calculul celor 26 de distanțe de la noua literă trasată la cele stocate în baza de date reprezintă o operație redundantă. Se încearcă paralelizarea sarcinilor de calcul în mai multe fire de execuție. Se observă faptul că aceste distanțe DTW nu prezintă dependențe de intrare sau antidependențe între ele. Singura constrângere a paralelizării calculului acestor distanțe fiind dependența de ieșire la determinarea optimului global.

Sarcina de calcul va fi împărțită în 4 fire de execuție, fiecare acoperind calculul a 6 respectiv 7 matrice DTW din totalul de 26. Fiecare fir de execuție va determina un optim local, urmând ca acestea să actualizeze optimul global. Pentru a evita posibilitatea pierderii optimului prin suprascrierea acestuia de către alt fir de execuție, modificarea optimului global se va efectua printr-un mutex ce va asigura accesul secvențial la locația acestuia.

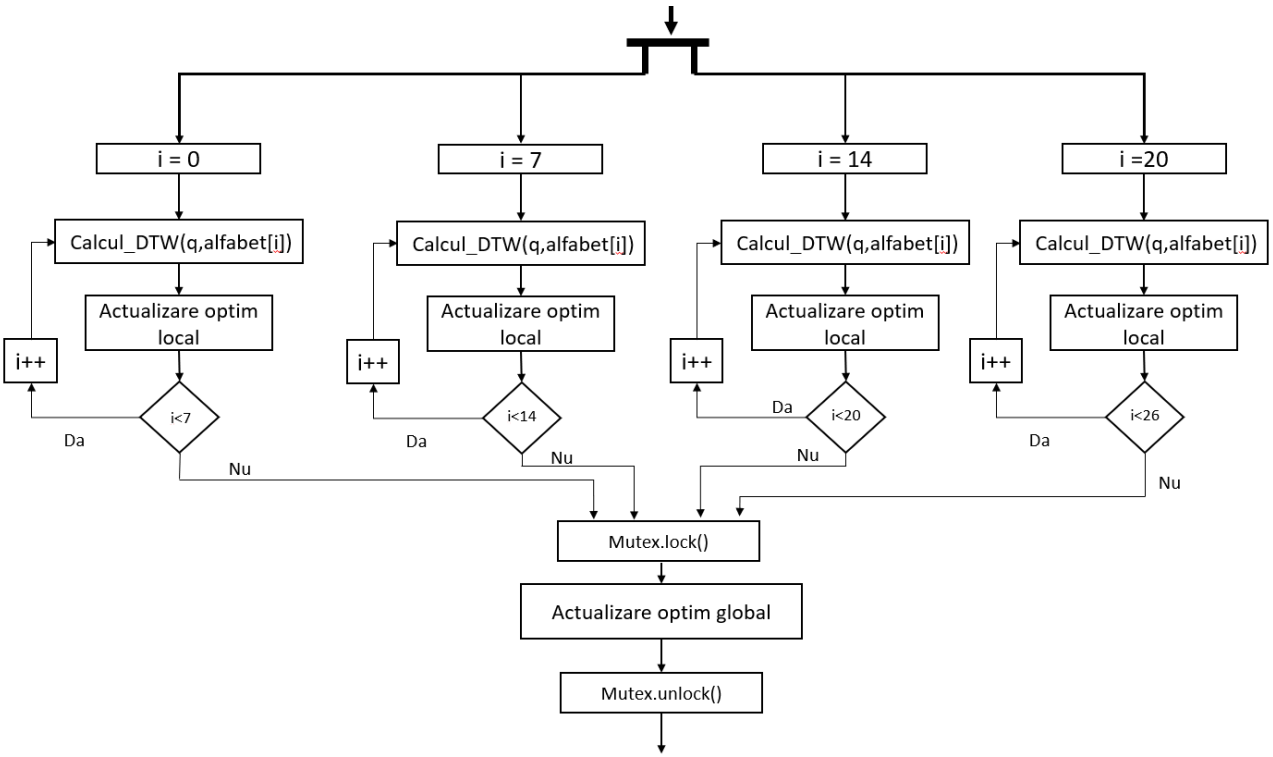

Figura 4. 9 Împărțirea pe fire de execuție

4.2 Optimizare Tesseract

După cum am prezentat și în capitolul anterior, Tesseract recunoaște literele trasate într-o imagine. Însă, valorile celor 2 unghiuri pitch și yaw $\{\theta , \psi\}$ alcătuiesc o secvență temporală fără

discontinuități. Astfel, o literă ce necesită o întrerupere a trasării pentru a se relua dintr-un alt punct trebuie tratată separat.

Pentru implementarea acestui mecanism, se utilizează al treilea unghi Euler: Roll {Φ}. Acesta se modifică prin rotirea brațului.

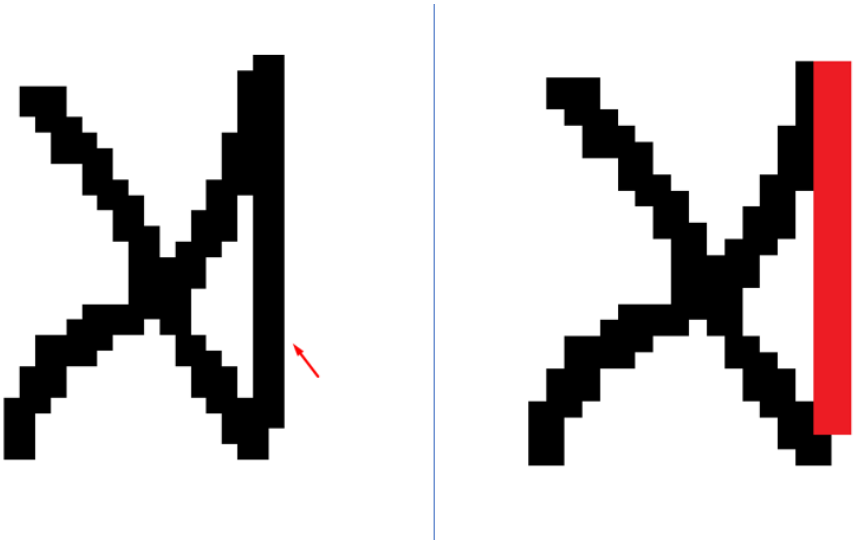

Figura 4. 10 Corectarea literelor

Capitolul 5 Date și rezultate experimentale

DTW

Pentru recunoașterea literelor folosind DTW, primul pas a fost crearea unei baze de date ce va conține reprezentări ale celor 26 de litere.

În acest scop am implementat un algoritm ce captează informații în timp real de la brățara Myo sub formă de quaternioni [w, x, y, z]. În urma prelucrării acestora, se extrag valorile unghiurilor Euler [Φ, θ, ψ] la fiecare moment de timp. Analiza și recunoașterea literelor efectuându-se folosind doar coordonatele { θ , ψ } vom ignora valoarea unghiului { Φ } în timp ce pe celelalte 2 le vom stoca într-un vector. Vectorul va fi structurat astfel încât pe pozițiile impare se vor afla valori succesive ale unghiului ψ iar pe pozițiile pare valori succesive ale unghiului θ. Astfel, la citirea vectorului, citirea a 2 elemente din acesta va corespunde extragerii unui punct descris prin aceste 2 coordonate. Vectorul linie va fi exportat într-un fișier precedat de o literă, aceasta din urmă reprezentând eticheta pe care o vom atribui vectorului respectiv.

La rularea aplicației, se citește întreaga bază de date si se stochează local într-un vector de elemente, fiecare element conținând toate informațiile necesare ce descriu litera: cele 2 secvențe temporale pentru fiecare axă { θ , ψ }, dimensiunile acestora și eticheta după care se va face clasificarea.

Tesseract

Tesseract pune la dispoziție multiple baze de date ce conțin modele a peste 100 de limbaje. Una dintre cele mai utilizate baze de date se numește „tessdata_fast”. Aceasta este foarte rapidă însă acuratețea este redusă. Spre deosebire de aceasta, „tessdata_best” este o bază de date mult mai lentă, însă care poate oferi o acuratețe ridicată. Alt treilea set este „tessdata” și este singura ce suportă și metodele de recunoaștere mai vechi. În acest proiect vom utiliza setul de date tessdata versiunea 4.0.0 din noiembrie 2016.

Acest set de date conține atât majuscule cât și litere mici, cifre , etc. Pentru această aplicație, vom limita numărul de caractere,extrăgând din setul de date doar majusculele.

Rezultate experimentale

Pentru testare, vom trasa folosind brățara Myo litere în aer și vom aplica pe rând atât calculul distanței minime DTW cât și recunoașterea pe imaginea creată folosind Tesseract pentru fiecare reprezentare. Se vor utiliza 10 reprezentări pentru fiecare literă, 260 în total, și vom compara rezultatele celor 2 metode de recunoaștere atât după acuratețe cât și după timp.

Tabel 5. 1 Statistici de performanță a DTW și Tesseract

	DTW		Tesseract	
	Acc(%)	timp(ms)	Acc(%)	timp(ms)
A	100	12	100	16
B	100	16	80	34
C	70	8.5	100	24
D	100	14	100	18
E	90	18	100	20
F	100	22	100	22
G	100	18	100	21
H	100	17	100	7
I	80	16	90	11
J	100	8	100	11
K	100	16	100	21
L	100	9	100	13
M	100	12	100	24
N	100	13	100	19
O	100	10.5	100	17
P	100	12	100	25
Q	80	18	90	21
R	100	22	100	34
S	100	11	100	20
T	100	20	100	28
U	100	9	100	18
V	100	9	100	17
W	100	15	100	19
X	90	17	80	16
Y	100	13	100	15
Z	100	12	100	10

În urma statisticii, s-a bținut o acuratețe a DTW de **96,53%** comparativ cu Tesseract unde acuratețea obținută este de **97,69%**. De asemenea, timpul de recunoaștere în fiecare caz este **14,1ms** pentru DTW și **19,2ms** pentru Tesseract.

Capitolul 6 Reproducerea literelor folosind brațul robotic Kinova

Development Center este aplicația companiei Kinova ce oferă utilizatorului accesul și controlul majorității resurselor pe care brațul Jaco^2 le are în dotare. Aceasta permite atât citirea valorilor fiecărui actuator în parte, a informațiilor despre curentul, forța și temperatura la care acestea sunt supuse cât și comutarea modului de control al brațului(figura 6.1). Kinova pune de asemenea la dispoziție un SDK ce permite utilizatorilor să integreze brațul Jaco^2 în aplicațiile personale la nivel înalt.

Folosind Development Center, am captat pe tablă puncte succesive, echidistante, ce vor fi utilizate ca puncte de referință pentru trasarea literelor. Astfel, fiecare literă va fi incadrată într-un pătrat de dimensiuni fixate, având unul dintre aceste puncte de reper în colțul din stânga jos. Orice mișcare ulterioară se va face relativ la acest punct, obținându-se astfel o modularizare a aplicației.

Exemplu:

Pornind din punctul de reper și cunoscând dimensiunea pe care litera o va avea se calculează puncte intermediare:

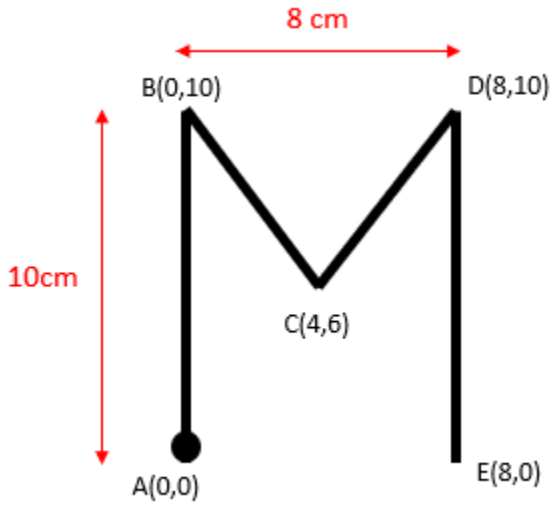

Figura 6. 1 Trasare literă folosind drepte

Prin apelul funcției MySendBasicTrajectory din librăria 'CommandLayer', folosind modul de comandă bazat pe poziții și parsarea unui punct de coordonate acestei funcții, brațul va executa o mișcare către acel punct pe cea mai scurtă traiectorie, trasând astfel în planul tablei o dreapta între punctul curent

și punctul parsat funcției. Singurele valori ce vor fi modificate sunt cele ce corespund axelor ce formează un plan paralel cu planul tablei, celelalte coordonate rămânând neschimbate.

Trasarea buclelor se poate efectua în același mod, însă pentru a trasa o buclă continuă, fără convexități, ar fi necesar un număr considerabil de puncte intermediare. De aceea, modul de control în funcție de poziții nu este optim în acest caz. Vom folosi modul de control în funcție de viteza de deplasare pe fiecare axă.

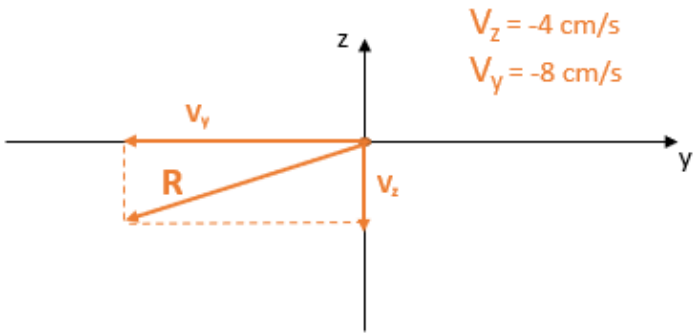

Figura 6. 2 Vector viteză rezultat

În acest mod de control, funcția MySendBasicTrajectory va interpreta valorile punctului descris anterior ca fiind vitezele de deplasare pe axele respective. Pentru a trasa o traiectorie circulară, este necesar ca vectorul de deplasare al cursorului să se schimbe constant. Vom crea un fir de execuție separat ce se va ocupa de transmisia în buclă a comenzii de mișcare în timp ce firul de execuție principal va modifica constant valoarea coordonatelor punctului, acesta din urmă fiind declarat global permițând celor 2 fire de execuție să comunice între ele. Acestea 2 nu au dependențe de ieșire și nu sunt nici antidependente, singura dependență prezentă fiind de succesiune, firul de execuție principal modificând variabila globală în timp ce firul de execuție secundar o accesează. Însă acest lucru nu reprezintă un impediment pentru aplicația de față.

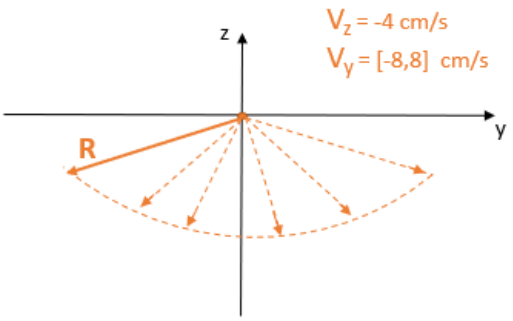

Figura 6. 3 Rotire vector viteza

În figura 6.3, vectorul viteză pe coordonata z are o viteză constantă în timp ce vectorul viteză pe coordonata y va baleia de la -8cm/s la 8cm/s, rezultând astfel o mișcare în spațiu sub forma unui arc de cerc (figura 6.4).

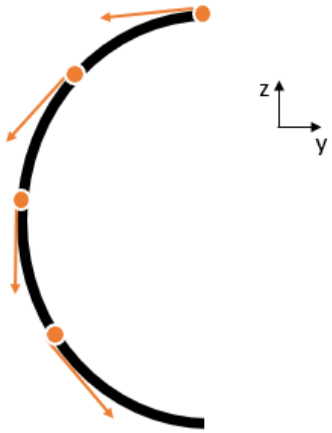

Figura 6. 4 Mișcare rezultată

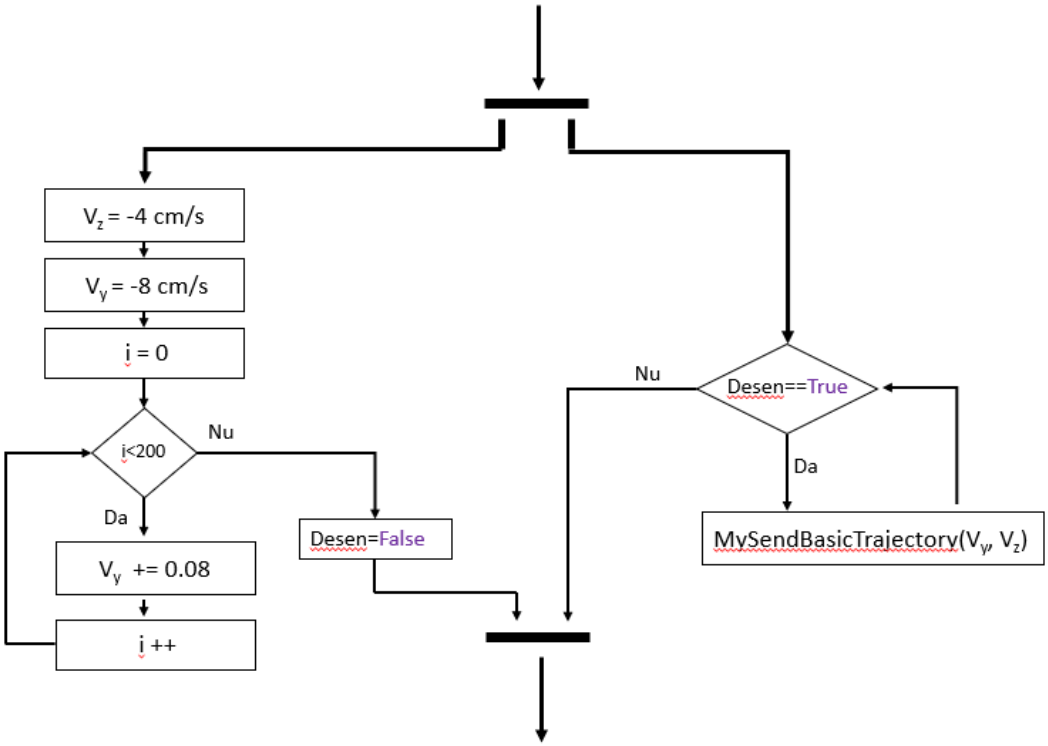

Figura 6. 5 Diagrama firelor de execuție pentru desenarea unei bucle

Se folosesc combinații a celor 2 metode pentru a desena fiecare literă. De exemplu, litera ‘D’ va fi formată dintr-o deplasare în funcție de poziție pentru a desena dreapta verticală și deplasare în funcție de viteză pentru buclă.

Concluzii

Scopul principal al proiectului a fost crearea unui sistem de recunoaștere în timp real a mișcărilor efectuate de brațul unui utilizator și apoi reproducerea acestora cu un braț robotic. În această lucrare, s-a studiat modul de recunoaștere al gesturilor ce trasează în spațiu litere ale alfabetului latin.

Sistemul pornește de la o brățară prevăzută cu accelerometru, giroscop și senzori de activitate musculară și care este amplasată pe antebrațul utilizatorului. În urma procesării datelor, s-a dorit recunoașterea literelor trasate în aer de către persoana purtătoare. În acest scop, s-au dezvoltat 2 metode de recunoaștere: prima se bazează pe determinarea similarității a 2 secvențe temporale, iar cea de-a doua a urmărit proiectarea acestora într-o imagine și efectuând o analiză pe baza imaginilor rezultate. În urma optimizărilor efectuate, s-a urmărit compararea celor 2 metode studiate în funcție de acuratețe și timpul de recunoaștere.

Pentru trasarea literelor pe o tablă albă verticală s-a utilizat brațul robotic Kinova Jaco^2. Pentru determinarea modului de control pe care l-am utilizat, am analizat performanțele și limitările fiecărui mod de control în parte. Brațul Jaco^2 s-a dovedit a fi un dispozitiv complex, capabil să imite mișcarea brațului uman în efectuarea sarcinii atribuite.

Contribuții personale

- Am reușit să captez datele sub formă de quaternioni de la brățara Myo și să le convertesc în unghiuri Euler.
- Am creat o bază de date cu reprezentări ale majusculelor din alfabetul latin, fiecare literă fiind alcătuită din observații succesive a unghiurilor Euler formate în spatiu tridimensional.
- Am implementat un algoritm de calcul al distanței minime DTW de la o nouă literă trasată la cele 26 de litere din baza de date.
- Am implementat un algoritm de proiecție într-o imagine a valorilor unghiurilor Euler normate și am aplicat Tesseract OCR pentru recunoașterea literelor trasate.
- Am optimizat cele 2 metode de recunoaștere utilizate și am comparat performanțele acestora.
- Am implementat un algoritm de control al brațului Kinova Jaco^2 pentru a trasa literele recunoscute pe o tablă verticală.

Bibliografie

[1] „Kinova Jaco^2 specifications”, disponibil la https://www.kinovarobotics.com/

[2]Kinova Jaco^2 User Guide (https://www.kinovarobotics.com/)

[3] Thalmic Labs, „Myo SDK documentation”

[4] Thalmic Labs, „Myo Armband User Manual”

[5] - , „Quaternion”, disponibil la https://en.wikipedia.org/wiki/Quaternion

[6] Ratanamahatana, Chotirat and Eamonn J. Keogh. “Making Time-Series Classification More Accurate Using Learned Constraints.” *SDM* (2004).

[7] Mueen, Abdullah and Eamonn J. Keogh. “Extracting Optimal Performance from Dynamic Time Warping.” *KDD* (2016).

[8] - , „Dynamic Time Warping” diponibil la https://en.wikipedia.org/wiki/Dynamic_time_warping

[9] - , „About OpenCV” diponibil la https://opencv.org/about/

[10] Sakoe, Hiroaki and Seibi Chiba. “Dynamic programming algorithm optimization for spoken word recognition.” (1978).

[11] Salvador, Stan and Philip Chan. “FastDTW : Toward Accurate Dynamic Time Warping in Linear Time and Space.” (2004).

[12] -, „Tesseract” disponibil la https://en.wikipedia.org/wiki/Tesseract_(software)

Anexa 1

- Cod sursă program principal -

```
#include<iostream>
#include <thread>
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include "myo.h"
#include "inc\litere.h"
#include "inc\fastDTW.h"
#include "BratJaco.h"
#include <Windows.h>
#include "conio.h"
#include<ctime>
#include<fstream>
#include "tesseract\baseapi.h"
#include "leptonica\allheaders.h"
#define inf 99999
int firstRoll, firstPitch, firstYaw;
cv::Mat mtx(28, 28, CV_8U);
std::vector<int> tmp;

int contor = 0;
element q;
char winner = 'A';
std::mutex mutex;
double GlobalDistance;
void calculDTW(int first, int last)
{
	double distance;
	double distTmp;
	char localWinner = alfabet[first].litera;
	distance = _dtw_window(q.pitch, q.yaw, alfabet[first].pitch, alfabet[first].yaw, q.size,
alfabet[0].size, 35, inf);

	for (int i = first + 1; i < last; i++)
	{
		distTmp = _dtw_window(q.pitch, q.yaw, alfabet[i].pitch, alfabet[i].yaw, q.size,
alfabet[i].size, 35, distance);
		if (distance > distTmp)
		{
			distance = distTmp;
			localWinner = alfabet[i].litera;
		}
	}
	mutex.lock();
	if (distance < GlobalDistance)
	{
		GlobalDistance = distance;
		winner = localWinner;
	}
	mutex.unlock();
```

```
}
int main(int argc, char** argv)
{


	mtx.setTo(255);
	cv::namedWindow("Desen", cv::WINDOW_FREERATIO);
	cv::resizeWindow("Desen", 500, 500);
	char *outText;

	tesseract::TessBaseAPI *api = new tesseract::TessBaseAPI();
	// Initializare tesseract-ocr cu limba engleza, fara specificarea caii catre tessdata
	if (api->Init("C:\\Program Files\\Tesseract-OCR\\tessdata", "eng",
tesseract::OEM_TESSERACT_ONLY)) {
		fprintf(stderr, "Could not initialize tesseract.\n");
		exit(1);
	}
	//setarea claselor ce vor fi recunoscute
	api->SetVariable("tessedit_char_whitelist", "ABCDEFGHIJKLMNOPQRSTUVWXYZ");

	citire_litere();

	std::vector<element> aux;
	try {

		myo::Hub hub("com.example.hello-myo");

		std::cout << "Attempting to find a Myo..." << std::endl;

		myo::Myo* myo = hub.waitForMyo(10000);

		if (!myo) {
			throw std::runtime_error("Unable to find a Myo!");
		}

		std::cout << "Connected to a Myo armband!" << std::endl << std::endl;

		DataCollector collector;
		hub.addListener(&collector);

		for (int i = 0; i < 150; i++)
		{
			q.pitch[i] = 0;
			q.yaw[i] = 0;
		}

		unsigned int t3;
		unsigned int t2;
		unsigned int t1;
		while (1) {
			// setarea ratei de refresh la 20 de cadre pe secunda
			hub.run(1000 / 20);
			switch (act)
			{
			case START:
				if (z < 8)
				{
					mtx.at<uchar>(x, y) = 0;
					mtx.at<uchar>(x, y - 1) = 0;
```

```
                        mtx.at<uchar>(x - 1, y) = 0;
                        mtx.at<uchar>(x - 1, y - 1) = 0;
                }
                if (0 == contor)
                {
                        firstRoll = collector.roll_w;
                        firstPitch = collector.pitch_w;
                        firstYaw = collector.yaw_w;
                }
                q.pitch[contor] = collector.pitch_w - firstPitch;
                q.yaw[contor] = collector.yaw_w - firstYaw;
                contor++;
                break;


        case REFRESH:
                mtx.setTo(255);
                mtx.at<uchar>(x, y) = 0;
                mtx.at<uchar>(x, y - 1) = 0;
                mtx.at<uchar>(x - 1, y) = 0;

                tmp.clear();
                contor = 0;
                break;

        case CAPTURE:
        {
                winner = 'A';
                q.size = contor;
                double distance;
                double distTmp;

                GlobalDistance = 9999;
                auto t1 = std::chrono::high_resolution_clock::now();

                std::thread thread1(calculDTW, 0, 7);
                std::thread thread2(calculDTW, 7, 14);
                std::thread thread3(calculDTW, 14, 20);
                std::thread thread4(calculDTW, 20, 26);

                thread1.join();
                thread2.join();
                thread3.join();
                thread4.join();
                auto t1elapsed = std::chrono::high_resolution_clock::now() - t1;
                long long microseconds1 =
std::chrono::duration_cast<std::chrono::microseconds>(t1elapsed).count();
                auto t2 = std::chrono::high_resolution_clock::now();

                api->SetImage((uchar*)mtx.data, mtx.size().width,
mtx.size().height, mtx.channels(), mtx.step1());


                // Get OCR result
                outText = api->GetUTF8Text();
                auto t2elapsed = std::chrono::high_resolution_clock::now() - t2;
                long long microseconds2 =
std::chrono::duration_cast<std::chrono::microseconds>(t2elapsed).count();
                printf("\n\nCastigator DTW: %c\n", winner);
                printf("Castigator Tesseract:%s\n\n", outText);
```

```
                        std::cout << "Timp de executie DTW:              " << microseconds1
<< " microsecunde" << std::endl;
                        std::cout << "Timp de executie Tesseract: " << microseconds2 << "
microsecunde" << std::endl;
                        desenareLitera(winner);

                        act = STOP;
#endif
                        break;
                    }
                    case STOP:

                        break;

                    }
                    collector.print();
                    imshow("Desen", mtx);
                    cv::waitKey(1);
                    previous_X = x;
                    previous_Y = y;
                }
        }
        catch (const std::exception& e) {
                std::cerr << "Error: " << e.what() << std::endl;
                std::cerr << "Press enter to continue.";
                std::cin.ignore();
                return 1;
        }
        return 0;
}
```

Anexa 2

- Cod sursă citire bază de date -

```
#include "stdafx.h"
#include "litere.h"
#include<iostream>
#include <vector>
#include <fstream>
#include<sstream>
element alfabet[26];
void citire_litere()
{
	std::ifstream myfile;
	std::string line;
	int contorLitera = 0;
	myfile.open("litere/SlowRate.txt");
	while (std::getline(myfile, line))
	{

		std::istringstream l(line);
		l.ignore(10, ' ');
		char litera;
		l >> litera;
		line.clear();
		getline(myfile, line);
		l.clear();
		std::istringstream v(line);
		int contorLungime = 0;

		while (v>> alfabet[contorLitera].pitch[contorLungime]\
			>> alfabet[contorLitera].yaw[contorLungime])
		{
			alfabet[contorLitera].litera = litera;
			contorLungime++;
		}
		alfabet[contorLitera].size = contorLungime + 1;
		contorLitera++;
		line.clear();
	}
}
```

Anexa 3

- funcții calcul DTW utilizate –

```
-   inline double  distance(int x1, int x2, int x3, int y1, int y2, int y3)
-   {
-          return 0.45*abs(x1 - y1) + 0.45*abs(x2 - y2) + 0.1*abs(x3 - y3);
-   }
-
-
-   inline double distanceW(int x1, int x2, int y1, int y2)
-   {
-          return abs(x1 - y1) + abs(x2 - y2);
-   }
-   /***********************Global Functions************************/
-   static inline double min(double a, double b, double c)
-   {
-          double m = a;
-          if (m > b) m = b;
-          if (m > c) m = c;
-          return m;
-   }
-
-   double _dtw(int* x1, int* x2, int* x3, int* y1, int* y2, int* y3,int sizeX,int sizeY)
-   {
-          double**mat;
-          mat = new double*[sizeX];
-          for (int i = 0; i < sizeX; i++)
-                 mat[i] = new double[sizeY];
-
-          mat[0][0] = distance(x1[0], x2[0], x3[0], y1[0], y2[0], y3[0]);
-          for (int i = 1; i < sizeX; i++)
-                 mat[i][0] = distance(x1[i], x2[i], x3[i], y1[0], y2[0], y3[0]) + mat[i -
    1][0];
-
-          for( int i=1;i< sizeY;i++)
-                 mat[0][i] = distance(x1[0], x2[0], x3[0], y1[i], y2[i], y3[i]) + mat[0][i
    - 1];
-
-          for(int i=1;i<sizeX;i++)
-                 for (int j = 1; j < sizeY; j++)
-                        mat[i][j] = distance(x1[i], x2[i], x3[i], y1[j], y2[j], y3[j]) +
    min(mat[i - 1][j - 1], mat[i - 1][j], mat[i][j - 1]);
-
-          std::cout << std::endl;
-          return mat[sizeX - 1][sizeY - 1];
-   }
-
-
-   double _dtw_window(int* x1, int* x2, int* y1, int* y2, int sizeX, int sizeY, int
    windowSize, double bestSoFar)
-   {
-
-          double*blockMat;
-          blockMat = new double[(sizeX + 1)*(sizeY + 1)];
-          std::fill_n(blockMat, (sizeX + 1)*(sizeY + 1), 2147483647);
-
-          double **mat;
-          mat = new double*[sizeX + 1];
```

```
-          for (int i = 0; i <= sizeX; i++)
-                 mat[i] = &blockMat[i*(sizeY + 1)];
-
-          double slope = (double)sizeY / sizeX;
-          mat[0][0] = 0;
-
-          for (int i = 1; i <= sizeX; i++)
-          {
-                 bool abandonFlag = true;
-                 int cost = 0;
-                 for (int j = std::_Max_value(1, (int)round(i*slope) - windowSize); j <
    std::_Min_value(sizeY + 1, (int)round(i*slope) + windowSize); j++)
-                 {
-                        cost = distanceW(x1[i - 1], x2[i - 1], y1[j - 1], y2[j - 1]);
-                        mat[i][j] = cost + std::_Min_value(mat[i - 1][j],
    std::_Min_value(mat[i][j - 1], mat[i - 1][j - 1]));
-                        if (mat[i][j] < bestSoFar)
-                               abandonFlag = false;
-                 }
-                 if (abandonFlag == true)
-                        return bestSoFar + 1;
-
-          }
-          return mat[sizeX][sizeY];
-   }
```

Anexa 4

- functie de control braț -

```
void desenareLitera(char litera)
{
	CartesianPosition dataPosition;
	switch (litera)
	{
	case 'L':
	{
		CartesianPosition dataPosition;
		MyGetCartesianPosition(dataPosition);
		copyPosition(pointToSend.Position.CartesianPosition, dataPosition.Coordinates);
		pointToSend.Position.CartesianPosition.Z += 0.1339;
		MoveToPoint(0.015);
		DelayBrat(10);

		pointToSend.Position.CartesianPosition.X = -0.5520;
		MoveToPoint(0.015);
		DelayBrat(10);

		pointToSend.Position.CartesianPosition.Z -= 0.1339;
		MoveToPoint(0.015);
		DelayBrat(10);

		pointToSend.Position.CartesianPosition.Y += 0.0515;
		MoveToPoint(0.015);
		DelayBrat(10);

		break;
	}

	case 'M':
	{
		CartesianPosition dataPosition;
		MyGetCartesianPosition(dataPosition);
		copyPosition(pointToSend.Position.CartesianPosition, dataPosition.Coordinates);
		pointToSend.Position.CartesianPosition.X = -0.550;
		MoveToPoint(0.015);
		DelayBrat(30);

		pointToSend.Position.CartesianPosition.Z += 0.0939;
		MoveToPoint(0.015);
		DelayBrat(30);

		pointToSend.Position.CartesianPosition.Z -= 0.0402;
		pointToSend.Position.CartesianPosition.Y += 0.0309;
		MoveToPoint(0.015);
		DelayBrat(30);

		pointToSend.Position.CartesianPosition.Z += 0.0402;
		pointToSend.Position.CartesianPosition.Y += 0.0309;
		MoveToPoint(0.015);
		DelayBrat(90);
```

```
        pointToSend.Position.CartesianPosition.Z -= 0.0939;
        MoveToPoint(0.015);
        DelayBrat(30);

        break;
}
```