

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tènica Superior d'Enginyeria del Disseny

DESENVOLUPAMENT D'UN SISTEMA AUTOMÀTIC
PER A JUGAR A ESCACS MITJANÇANT ROBÒTICA
COL·LABORATIVA I VISIÓ ARTIFICIAL.

TREBALL FINAL DEL

Grau en Enginyeria Electrònica Industrial i Automàtica

REALITZAT PER

Lluna Sanz Montrull

TUTORITZAT PER

Eugenio Ivorra Martínez i Antonio José Sánchez Salmerón

DATA: **València, setembre, 2019**

Índex

1.	Introducció	5
1.1.	Problema a resoldre	5
1.2.	Objectius	5
1.3.	Marc teòric	6
1.3.1.	Protocol TCP/IP	6
1.3.2.	Homografia	7
1.3.3.	Segmentació: llindar (threshold) i filtre Canny	8
1.3.4.	Cinemàtica en robòtica	11
2.	Escenari	14
2.1.	Tauler i peces	14
2.2.	Il·luminació	15
2.3.	Càmera	15
2.4.	Robot UR3	15
2.5.	Chess Master i Stockfish	15
3.	Implementació	17
3.1.	Connexió amb el robot i amb la webcam	17
3.2.	Visió	19
3.3.	Processat de dades en Matlab	25
3.4.	Elaboració d'un efecto final	28
3.5.	Implementació del robot	31
3.6.	Codi principal: <i>joc.m</i>	37
4.	Resultats i conclusions	39
4.1.	Proves en l'adquisició i la homografia	39
4.2.	Proves en el processament de dades de la imatge	39
4.3.	Proves amb el robot	40
4.4.	Problemes trobats i solucions	40
4.4.1.	Imprecisió del robot a l'hora de situar les peces	40
4.4.2.	Caiguda de les peces agafades	41
4.4.3.	El robot s'enrotlla amb els tubs	42
4.4.4.	Incorrecta definició del lloc on el robot deixa les peces capturades	42
4.4.5.	Detecció errònia de les jugades fetes per el jugador	42
4.5.	Temps aproximats d'execució	43
4.6.	Conclusions	43
5.	Referències	45

1. Introducció

1.1. Problema a resoldre

Tenint un tauler d'escacs, es requereix dissenyar un sistema capaç de jugar contra un humà a una partida amb la menor intervenció humana possible per al correcte funcionament del mateix. Per això, es farà ús d'un braç robòtic, una càmera, una intel·ligència artificial capaç de processar els moviments i una intereficie gràfica.

1.2. Objectius

Els objectius per desenvolupar aquesta aplicació seran:

- Establir les condicions òptimes per a obtenir una correcta adquisició del tauler i de les peces. S'inclou el re-disseny de l'entorn en cas de el processament d'imatges no puga aconseguir una segmentació suficient, el robot no puga situar-se en totes les posicions del tauler o la complexitat del processament de la imatge es considere massa elevada.
- Es determinaran unes condicions d'il·luminació uniformes que no es vegen influenciades per qualsevol factor exterior al laboratori com, per exemple, la posició del Sol segons el moment del dia.
- El sistema de visió ha de ser capaç de distingir les distintes variacions de moviment de les peces que tinguen lloc en el tauler, així com identificar el color de cada peça per detectar, correctament, els moviments del jugador.
- Programar el robot col·laboratiu *UR3* de *Universal Robots* per a dirigir-se a les caselles que li indique la intel·ligència artificial *Stockfish* seguint un conjunt de punts determinat. A més, serà necessari el disseny d'un efector final capaç d'agafar qualsevol peça.
- Correcció de les errades que tinguen lloc durant les proves d'execució.
- Verificar el correcte funcionament de tot el conjunt implementat en Matlab.

1.3. Marc teòric

1.3.1. Protocol TCP/IP

El conjunt de protocols TCP/IP es nombrat així per els dos protocols que el conformen: TCP (Transmission Control Protocol) i IP (Internet Protocol). Aquest conjunt té el principal objectiu de fer una interconnexió de de distintes xarxes per a crear hosts i clients.

El que respecta a la direcció IP, aquesta es representa amb 32 bits i s'expressa amb números decimals separats per un punts. Cada número decimal representa 8 bits. Depenent del primer grup de bits, cada IP tindrà una classe o altra. Aquestes son:

- Classe A: empra 7 bits per a la xarxa i 24 per al host, tenint 126 xarxes amb 16.777.214 hosts. Les direccions tenen un interval de 0.0.0.0 a 127.255.255.255.
- Classe B: empra 14 bits per a la xarxa i 16 per al host, tenint 16.382 xarxes amb 65.534 hosts. Les direccions tenen un interval de 128.0.0.0 a 191.255.255.255.
- Classe C: empra 21 bits per a la xarxa i 8 per al host, tenint 2.097.150 xarxes amb 254 hosts. Les direccions tenen un interval de 192.0.0.0 a 233.255.255.255.
- Classe D: direccions reservades per a difusió selectiva o *multicasting* (els paquets son enviats a múltiples xarxes de forma simultània). Les direccions tenen un interval de 224.0.0.0 a 239.255.255.255.
- Classe E: aquestes direccions queden reservades per a l'ús experimental o bé per a una futura aplicació. Les direccions tenen un interval de 240.0.0.0 a 255.255.255.255.

Si, per exemple tenim les direccions IP 158.42.69.139 i 158.42.206.10, es sap que les dues son de classe B i pertanyen a la mateixa xarxa (158.42.xxx.xxx). Per tant, aquests dos hosts podran tindre connexió. En cas de tindre una IP de classe A, aquesta no pot connectar-se a les altres dues.

En aquest protocol, la direcció 0.0.0.0 fa referència al propi host, la que empra per a referir-se a ell mateix.

1.3.2. Homografia

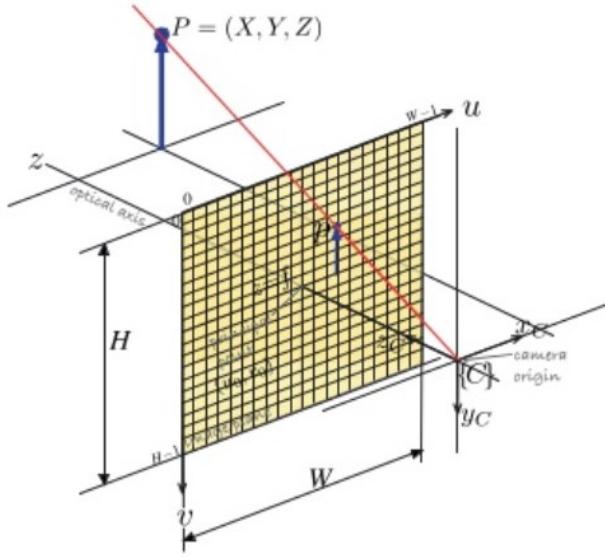
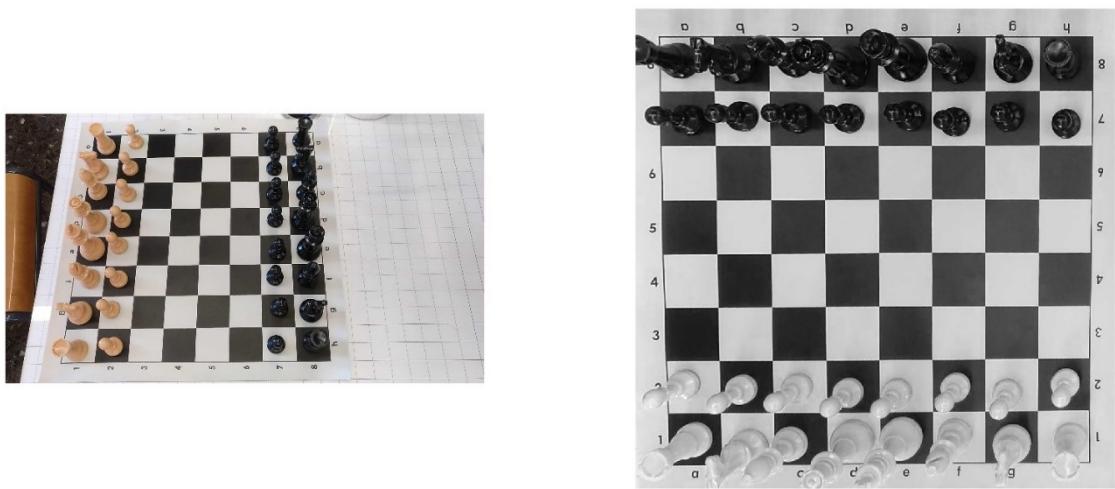


Figura 1: model on es representa la projecció d'un punt P en l'espai (X , Y i Z) en el sensor d'una càmera (H i W en píxels) [2].

Donades dues perspectives distintes (des d'una càmera 1 i una càmera 2) i un conjunt de punts immòbils en l'espai, l'objectiu de la homografia és trobar una transformació (expressada en una matriu H de tres per tres) que ajuste la imatge captada per la càmera 2 a la perspectiva de la càmera 1.

Per a dur a terme aquest procés han de processar-se les dues imatges per a trobar patrons i punts en comú entre les dues imatges. Depenent del mètode d'homografia emprat, es fa servir d'un mètode o altre per a obtenir la matriu de transformació d'una perspectiva d'una imatge a la d'una altra.

La homografia permet situar la càmera en un lloc de treball convenient sense de comprometre a la qualitat del procés d'adquisició. Una altra perspectiva pot donar major facilitats a un braç robòtic per a operar sense tindre en compte que hi ha una càmera dins de la seuà àrea, per exemple.



Figures 2 (esquerra) i 3 (dreta): imatge del tauler sense aplicar la matriu d'homografia i imatge del tauler després d'aplicar la transformació respectivament. El procediment queda detallat en l'apartat 3.2.

D'acord al que es pot observar en les dues figures anteriors, la perspectiva de la número 2 s'ha transformat a una altra que sembla que la càmera estiga captant la imatge des de la part superior del tauler. Malgrat, la imatge captada no seria la mateixa que la transformada ja que, aquesta primera, no presentaria l'efecte on les peces semblen estar inclinades cap a l'esquerra, degut a la informació que conté la imatge original.

1.3.3. Segmentació: llindar (threshold) i filtre Canny

En el processament té l'objectiu d'identificar els elements d'interès de la escena i actuar segons la aplicació que es vulga dur a terme. En el cas de la segmentació, és convenient treballar amb imatges binàries (composta únicament per píxels negres i blancs) perquè permeten un menor càlcul computacional i simplificar el problema a tractar.

La tècnica del llindar (en anglès, *threshold*) és un procés de segmentació on una imatge d'escala de grisos es transforma a una binària seguint el següent criteri, donada una matriu de píxels c de dimensions u i v :

$$c[u, v] = \begin{cases} 0 & I[u, v] < t \\ 1 & I[u, v] \geq t \end{cases} \quad \forall(u, v) \in I$$

Com s'observa en la operació anterior, tots els píxels que tinguen un valor menor a t , seran transformats en píxels de color negre; mentre que els píxels amb un valor major o igual a t seran transformats a píxels blancs. A continuació, es mostra un exemple de com funciona aquesta tècnica.

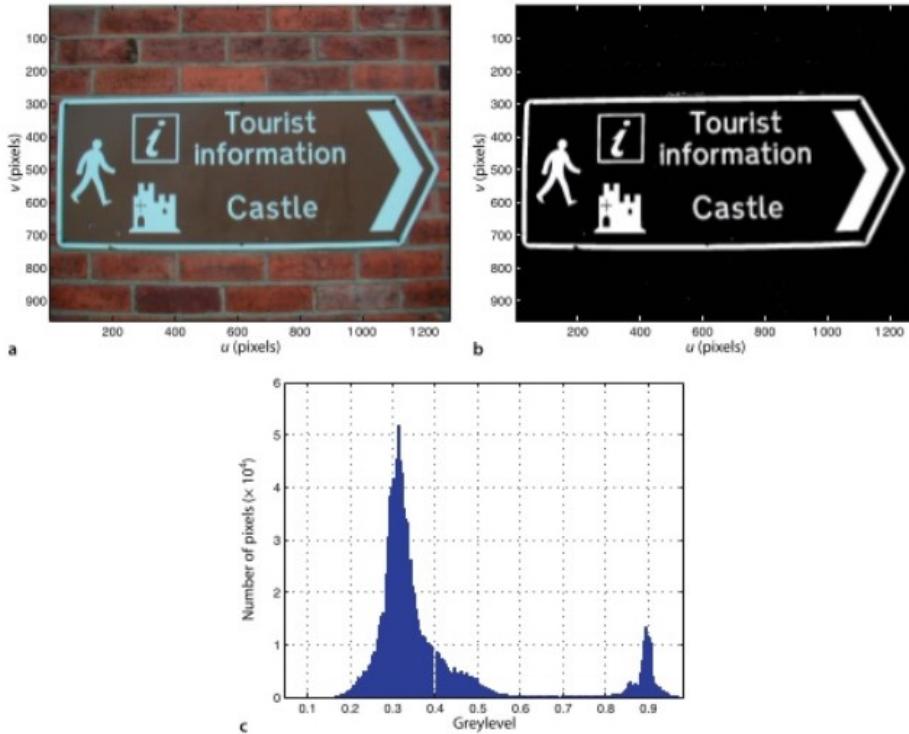


Figura 4: imatge original (a), imatge on s'aplica un llindar de 0.6 (b) i histograma de la imatge original en escala de grisos (c).

En l'exemple anterior, s'ha aconseguit fer una segmentació òptima al establir el valor del llindar en 0.6 separant els elements d'interès de la imatge (en aquest cas, el contingut de la senyal) de la resta.

El que respecta a la detecció de vores en les imatges, es basa principalment en la operació del gradient per detectar els canvis d'intensitat més bruscs entre píxels del voltant. D'aquesta manera, les zones que tinguen una intensitat similar es transformaran en píxels oscurs després d'aplicar el filtre i , d'aquesta manera, es voran més marcades les zones on es produeix un major canvi d'intensitat. Aquesta operació de gradient té la següent forma:

$$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

A continuació, es pot observar la aplicació d'aquest operador en l'exemple següent on es pot notar aquest destacament de vores en les zones on els cabells es situen prop de la pell, per exemple.



Figures 5 i 6: imatge original (esquerra) i imatge després d'aplicar l'operador descrit (dreta).

Per altra part, en el cas del filtre Canny, es busquen els màxims locals de la imatge, calculant el gradient amb la derivada del filtre gaussià. A més, dos llindars son emprats per a trobar vores fortes i febles. Els valor dels llindars son escollits de forma automàtica per Matlab en funció de la imatge. En Matlab, s'executa amb la funció *edge* on es permet definir els següents paràmetres:

- Threshold: es defineix, amb un vector de dos elements, els límits inferiors i superiors amb la assignació d'un número entre el 0 i l'1. També es pot ficar un escalar que delimita el llindar superior i l'inferior es defineix multiplicant el valor del superior per 0.4.
- Direcció de les vores: poden assignar-se en horitzontal, vertical o ambdues.
- Desviació estàndard del filtre (sigma): aquest paràmetre es defineix després del llindar. Si no s'especifica, es farà ús del valor predeterminat de 2.

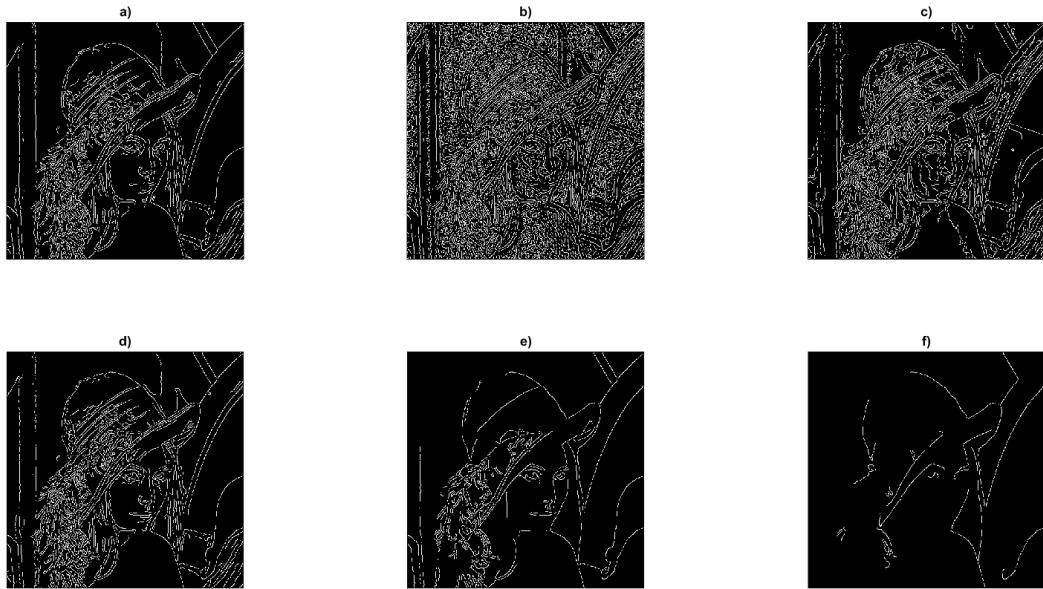


Figura 7: filtre Canny sense indicar un valor de threshold (a), filtre Canny amb un llindar superior de 0.01 (b), filtre Canny amb un llindar superior de 0.05 (c), filtre Canny amb un llindar superior de 0.1 (d), filtre Canny amb un llindar superior de 0.3 (e) i filtre Canny amb un llindar superior de 0.6 (f).

1.3.4. Cinemàtica en robòtica

El model matemàtic per a descriure la cinemàtica dels braços robòtics va ser introduït per Jacques Denavit i Richard Hartenberg en un paper del 1955. En aquest model, donat un braç robòtic amb N articulacions i , per tant, N+1 enllaços (considerats cossos rígids); es poden definir la posició i orientació de l'efector final a partir dels valors d'orientació de les articulacions.

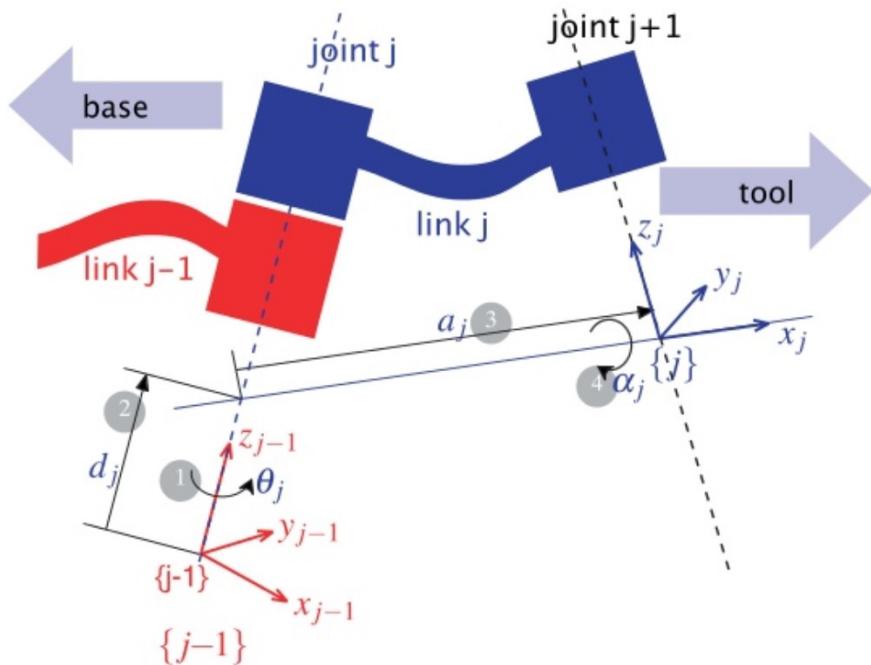


Figura 8: definició dels paràmetres de Denavit i Hartenberg extreta de [2].

Per a passar del sistema de coordenades $j-1$ a j , és necessari fer una transformació de rotació i translació. En primer lloc, es fa una rotació en θ (sobre l'eix z); seguidament una translació en direcció a l'eix z (d) i altra amb la mateixa direcció que l'eix x (a); i, finalment, una rotació en α (sobre l'eix x). Aquesta transformació pot ser expressada en la següent matriu:

$${}_{j-1}A_j = \begin{pmatrix} \cos \theta_j & -\sin \theta_j \cdot \cos \alpha_j & \sin \theta_j \cdot \sin \alpha_j & a_j \cdot \cos \theta_j \\ \sin \theta_j & \cos \theta_j \cdot \cos \alpha_j & -\cos \theta_j \cdot \sin \alpha_j & a_j \cdot \sin \theta_j \\ 0 & \sin \alpha_j & \cos \alpha_j & d_j \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

En el cas del robot de 6 articulacions de revolució (també conegut com RRRRRR), la cinemàtica directa es compon per 6 matrius de Denavit i Hartenberg que descriuen la transformació del sistema de coordenades des de la base fins a l'efector final.

$${}^0A_6 = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6$$

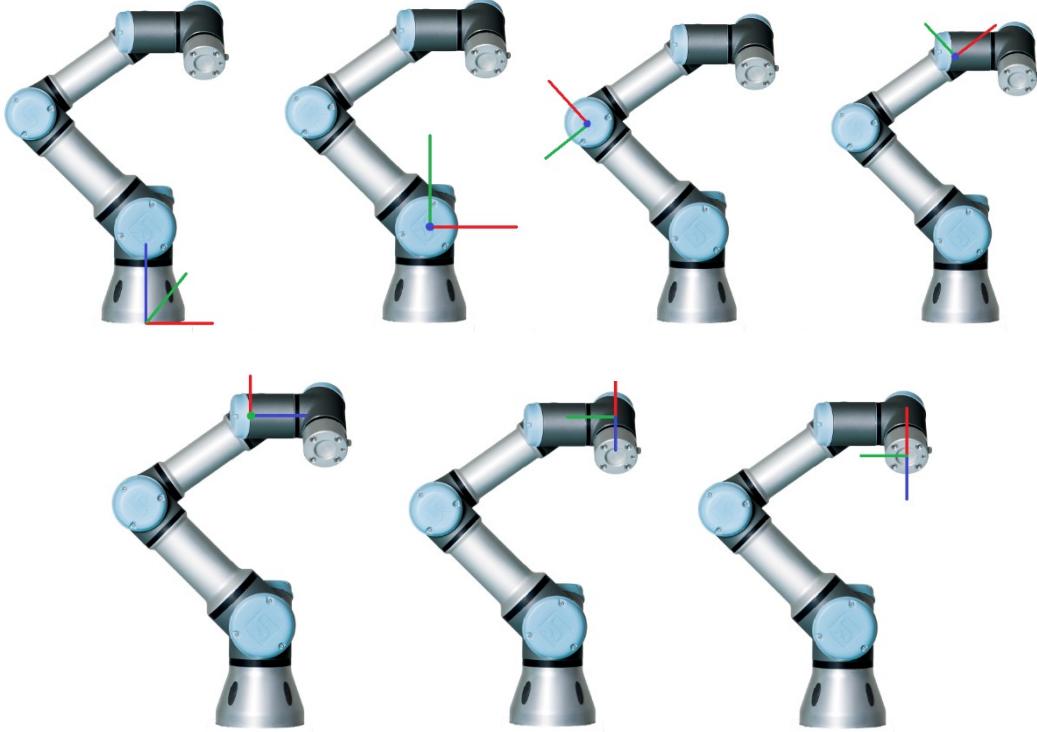


Figura 9: els distints sistemes de referència per al cas dels braços robòtics UR (en concret, l'UR3). Els eixos X, Y i Z es representen en roig, verd i blau respectivament.

Els paràmetres per a cada matriu A es defineixen segons la següent taula. Per a cada transformació, es suposarà que el valor dels angles de totes les articulacions és 0.

Articulació j	θ	d	a	α
1	q_1	d_1	0	$\pi/2$
2	$q_2 + \pi/2$	d_2	a_2	0
3	q_3	$-d_3$	a_3	0
4	q_4	d_4	0	$\pi/2$
5	q_5	d_5	0	$-\pi/2$
6	q_6	d_6	0	0

Tenint aquests paràmetres en compte, es pot saber la posició i orientació de l'efector final a partir de saber els valors dels angles de les articulacions (q_n) i, així, queda resolt el problema de cinemàtica directa.

Per altra part, està el problema de cinemàtica inversa on s'han de calcular el valor de totes les articulacions per a una posició d'efector final donada. A diferència de la cinemàtica directa, la inversa té més d'una solució possible (concretament, infinites) i el càlcul és més complex. Per facilitar els càlculs, es determinen les orientacions de les tres primeres articulacions o, bé, es determinen les de les tres últimes. D'aquesta manera, s'eviten indeterminacions en la computació.

En el procés on s'indica al robot menejar l'efector final d'una posició a altra, es calculen els angles de les articulacions segons la posició actual del robot i la final. La trajectòria pot ser d'una manera o altra segons el tipus de moviment: en J, linear i circular.

2. Escenari



Figura 10: escenari en el que es treballarà.

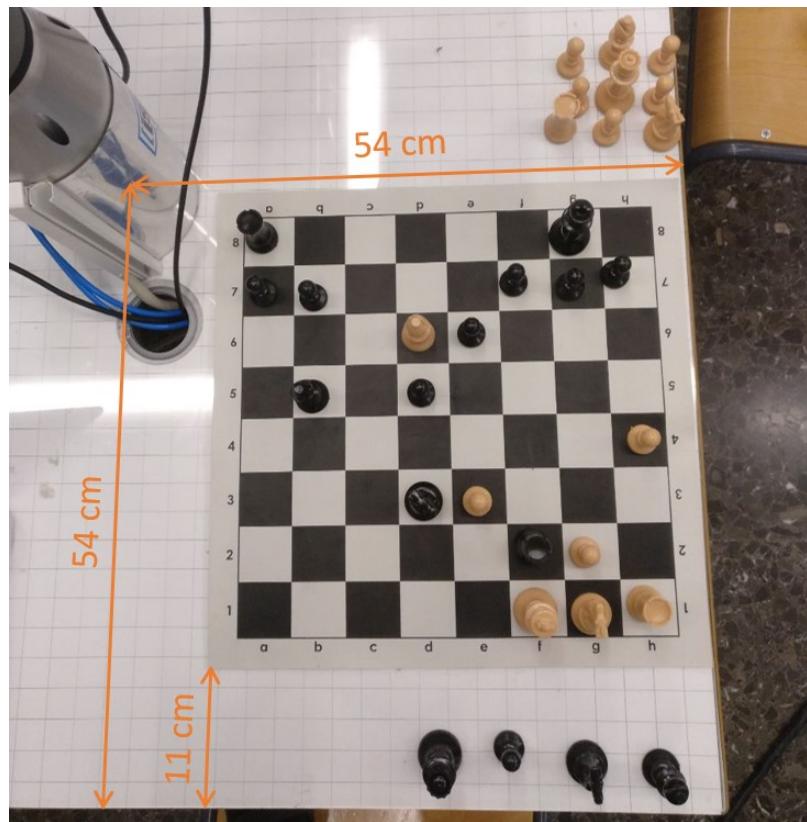


Figura 11: Distàncies del robot i del tauler en l'escenari.

2.1. Tauler i peces

S'utilitza un tauler de 43x43 cm amb caselles de 4.7 cm per costat i amb colors blancs i negres.

S’empren unes peces de plàstic *Stauton 5* de colors crema i negre amb una altura de rei de 9 cm i un diàmetre màxim de la base de 3.5 cm. Aquests tipus de peces son emprades en competicions i en escoles d’escacs.

2.2. Il·luminació

L’escenari compta amb una il·luminació controlada i difusa. Concretament, s’utilitzen tubs fluorescents amb una temperatura de llum de 4000 K i una potència de 36 W. Aquests es situen a dos metres per damunt del tauler.

A més, l’entorn disposa dels elements necessaris (en aquest cas, cortines) que permeten que la influència de l’exterior (com la posició del Sol segons el moment del dia) no tinga influència en la adquisició.

2.3. Càmera

Per a la adquisició d’imatge s’empra la càmera d’un mòbil *BQ Aquaris X*, concretament, una *Sony IMX298* de 16 MP. La seuva apertura és de $f/2.0$ ($1.12 \mu\text{m}/\text{píxel}$). La resolució de vídeo és 1080p, gravant a 60fps.

2.4. Robot UR3

Per a aquesta aplicació, s’empra el robot col·laboratiu UR3 d’Universal Robots. Aquest té una longitud de braç de 50 cm (menor a la diagonal del tauler) i es fa necessari tindre un efector final el suficientment llarg com per a poder arribar a la casella **h1** sense problemes.

En aquest cas, el robot pot treballar en aquestes condicions, però s’han de definir dues orientacions per a la pinça per a que el robot no tinga col·lisions amb ell mateix (com pot passar en la zona propera de la casella **a8**).

2.5. Chess Master i Stockfish

Chess Master serà la interfície gràfica d’usuari que processarà la partida i rebrà les dades mitjançant Matlab que, a la vegada, es comunicarà amb la càmera (detectant i comunicant a la IA dels moviments del jugador) i amb el robot (transmetent els moviments que realitza la IA al tauler de l’escenari).

Per a la IA, s’ha escollit *Stockfish 10* per ser compatible amb *Chess Master* i, a més, és un dels motors d’escacs més potents. Aquest pot ser configurat des de la interfície fent que tarde en calcular la jugada segons el nivell de profunditat o interval de temps.

La connexió en ambdues parts es fa la primera vegada que s'inicia la interfície de Chess Master prement *Engine* → *Add Engine...* i, des d'allí, es selecciona el directori on està la IA d'Stockfish.

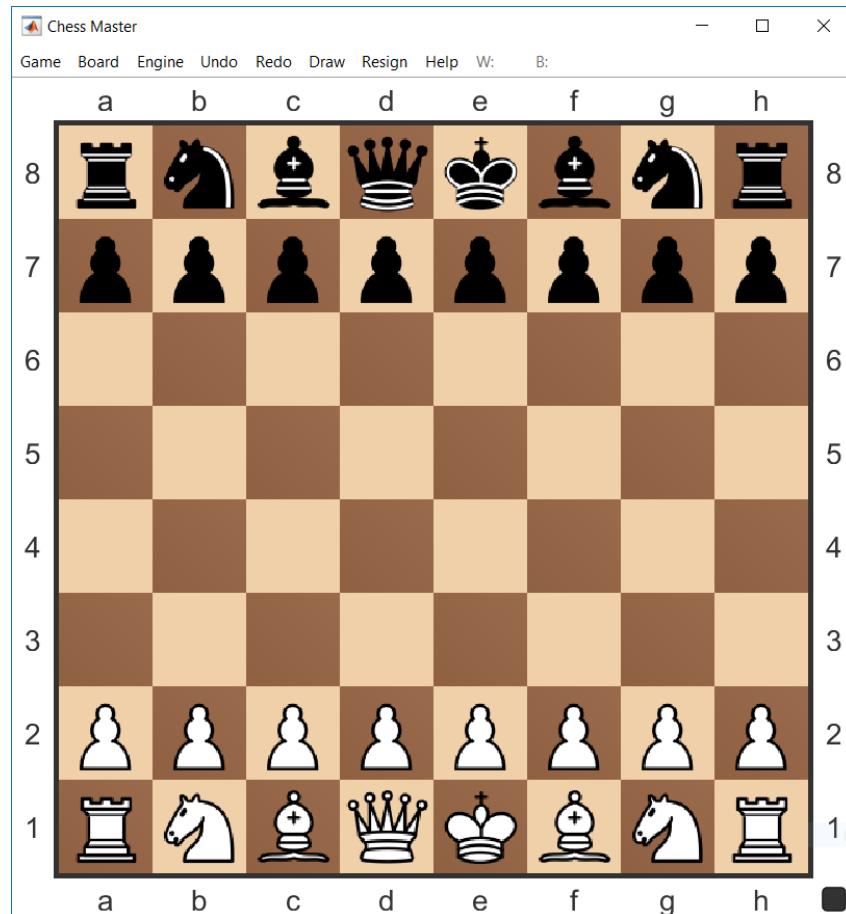


Figura 12: interfície de Chess Master.

3. Implementació

3.1. Connexió amb el robot i amb la webcam

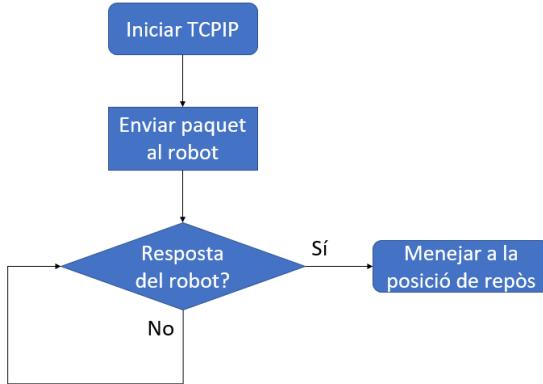


Figura 13: diagrama de flux de la connexió entre el Matlab i el robot.

Per fer possible la connexió entre el robot (client) i l'ordinador (servidor) mitjançant Matlab, serà necessària realitzar una connexió amb el protocol TCPIP. En primer lloc, cal assegurar-se de que el robot i l'ordinador estiguin connectats en la mateixa xarxa (en aquest cas, la xarxa amb la IPv4 158.42.x.x).

Aplicant el comandament de “ipconfig” en el símbol del sistema, es comprova que la IPv4 assignada a l'ordinador és 158.42.143.80 que comparteix la mateixa xarxa que el robot UR3 que té la direcció IPv4 158.42.206.10.

En el Matlab s'ha de crear un objecte TCPIP amb el host local 0.0.0.0 i port 3000 que farà el rol de servidor. Una vegada configurat, s'obri la connexió i s'espera l'arribada d'un paquet.

```
t = tcpip('0.0.0.0', 3000, 'NetworkRole', 'server');
    % Es crea un objecte TCPIP.
    % Client: 158.42.206.10
t.terminator='';
disp("Esperant la connexió del client");
%Espera la conexión
fopen(t);
% Llegir el missatge rebut del del robot
data = fscanf(t);
%data = fread(t, t.BytesAvailable);
disp(data)
```

La connexió del client al servidor es realitzarà mitjançant el programa “ClientTCP.urp” que executarem des del robot.

```

Programa
    BeforeStart
        MoveJ
            Punt_de_pas_1
            open:=socket_open("158.42.143.80", 3000)
        Bucle open!=False
            open:=socket_open("158.42.143.80", 3000)
            targetPos:=p[0,0,0,0,0,0]
            counter:=0
            tipusMov:=0
            obrir:=0
        Programa de robot
            receiveFromServ:=socket_read_ascii_float(8)
            Bucle receiveFromServ[0]≠8
                Esperar: 0.3
                receiveFromServ:=socket_read_ascii_float(8)
                tipusMov:=receiveFromServ[counter+1]
                Bucle counter<6
                    targetPos[counter]:=receiveFromServ[counter+2]
                    counter:=counter+1
                obrir:=receiveFromServ[8]
                If tipusMov!=0
                    MoveJ
                        targetPos
                Else
                    MoveL
                        targetPos
                If obrir!=0
                    Ajustar DO[4]=Apagar
                Else
                    Ajustar DO[4]=Encender
                counter:=0

```

Quan el robot recibeix el primer paquet del servidor, aquest programa roman a l'espera de rebre paquets del servidor que indiquen els moviments a realitzar.

Per a la connexió del mòbil, que funcionarà com a càmera, es fa ús de la aplicació “IP Webcam” (disponible de forma gratuïta en la “Play Store”). Des del mòbil, es configura un usuari i contrasenya d'accés accedint al menú “Local broadcasting”. També es comprovarà que estiga la opció d'IPv4 activada i que no hi haja cap conflicte amb el port predeterminat 8080. Una vegada fet, es prem la opció “Start server”.

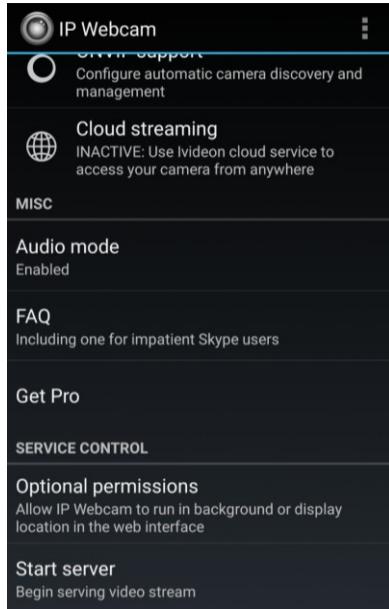


Figura 14: captura de pantalla de la aplicación IP Webcam.

En la pantalla apareixerà dues direccions: una *http* i altra *https*. Aquestes dues es faran servir per a fer la connexió amb el mòbil mitjançant Matlab. Per a això, s'emprarà la funció *ipcam* amb tres paràmetres d'entrada: una de les dues direccions afegint */video* al final, l'usuari i la contrasenya.

```
cam = ipcam('http://158.42.69.139:8080/video', 'ChessCam',
'TFG123')
```

Fent ús de la funció *snapshot*, s'obté una imatge amb una resolució de 1920x1080 píxels. Aquesta és suficient per a poder treballar amb l'escenari plantejat, a més de permetre que qualsevol altre *smartphone* puga ser utilitzat per a l'adquisició.

3.2. Visió

La homografia del tauler s'ha de realitzar sense ninguna peça present en el mateix ja que la funció *cv.findChessboardCorners* (de la llibreria d'Open CV) no pot fer una correcta detecció si hi ha peces que, segons la perspectiva, tapen punts claus. A més, aquesta s'ha de fer tenint la casella **h1** en la part inferior esquerra i la càmera ha d'estar situada a una distància aproximada de 50 cm del tauler, en el cas de no emprar algoritmes OCR per llegir lletres i dígits (identificant la casella **a1** trobant la a més pròxima de l'**1**).

Prèviament a la realització de la homografia, hem de tindre una imatge plana del tauler on el sensor de la càmera siga paral·lel al tauler i estiga situat en la part central del mateix. A més, de que el tauler ocupe tota la imatge de la fotografia per obtindré la màxima

resolució possible. La imatge es tracta per a que siga quadrada i es guarda en la mateixa carpeta del Chess Master. En l'script *joc.m*, emprem l'arxiu *tauler_pla_crop_bo.jpg*.

En les dues imatges, s'empra la funció *cv.findChessboardCorners()*, sent els paràmetres d'entrada aquestes imatges mencionades i un vector [7,7] que indicarà a la funció els punts clau a que tornarà (en aquest cas, 49). Per comprovar que ha funcionat correctament, mostrarem les imatges que s'obtenen amb la funció *cv.drawChessboardCorners()*, sent els paràmetres d'entrada la imatge emprada, el vector [7,7] i els 49 punts obtinguts de la funció anterior.

```
sensePeces = snapshot(cam);
[punts, ok] = cv.findChessboardCorners(sensePeces, [7,7]);
provaCrop_1 = cv.drawChessboardCorners(taulerPla,[7,7],
puntsCrop);
```

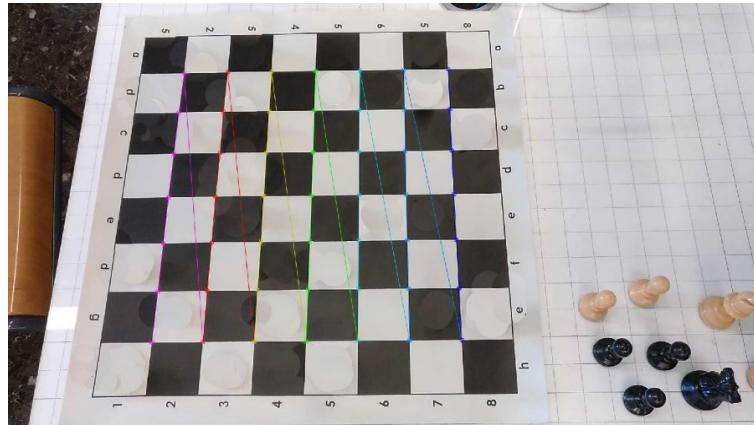


Figura 15: captura de la càmera després d'aplicar la funció *cv.findChessboardCorners()* i *cv.drawChessboardCorners()*.

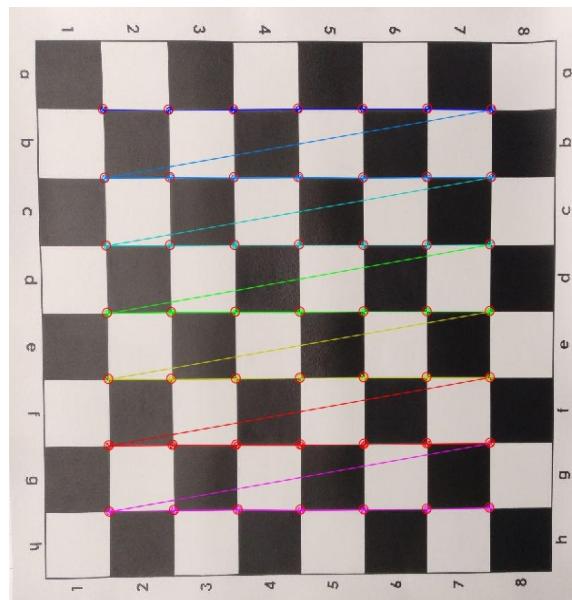


Figura 16: imatge del tauler pla després d'aplicar la funció `cv.findChessboardCorners()` i `cv.drawChessboardCorners()`.

Les dues matrius de cel·les de punts han de ser transformades a matrius ordinàries del tipus de dades subjacents amb la funció `cell2mat()`. D'aquesta manera, podrem treballar amb elles per obtindre la matriu d'homografia entre les dues imatges fent ús de la funció `cv.findHomography()` on el primer paràmetre d'entrada serà la matriu de punts obtinguts de la captura de la càmera i el segon correspondrà a la matriu de punts de la imatge del tauler pla (sense perspectiva).

Amb la matriu d'homografia es podrà transformar la fotografia en perspectiva a una imatge plana en la instrucció `cv.warpPerspective()`. On el primer paràmetre serà qualsevol imatge obtinguda en perspectiva i el segon la matriu d'homografia. Aquesta funció s'emprarà al llarg de tota la partida per l'anàlisi de cada estat del tauler.



Figures 17 (esquerra) i 18 (dreta): imatge del tauler des de la càmera i la imatge obtinguda per la transformació homogràfica respectivament.

Tenint els 49 punts del tauler pla, $puntsC$, es poden calcular els altres 64 punts d'interès que determinaran la posició de cada casella. Per a això, es farà ús de la funció implementada `sortMat()`. A més, els ordenarà des de la casella **a1** fins a la **h1**, després, de la casella **a2** fins a la **h2**; i, així, en totes les files fins obtenir una matriu de 64x2 elements.

La funció calcula la resta de vèrtexs del tauler, sent aquests 81 calculant les distàncies dels punts situats als extrems (part esquerra, inferior, dreta i superior). En primer lloc, es calcula el vèrtex inferior esquerre d'**a1** a partir del punt 43 de la matriu M ($puntsC$), el vèrtex superior dret d'**a1** on es suma la distància en y , dis_down , i es resta la distància en x , dis_left . Afegint els punts que ja es tenien en un principi i calculant el de les vores, ja tenim els punts suficients per calcular el centre de cada casella fent la mitja de cada costat per la part vertical i horitzontal, com mostra el fragment de codi de `sortMat()`.

```

for i=2:72
    if (mod(i-1,9)<0 || mod(i-1,9)>0)
        sQua = [sQua; ((sM(i+8,1)+sM(i,1))/2),
(sM(i+8,2)+sM(i,2))/2];
    end
end

```

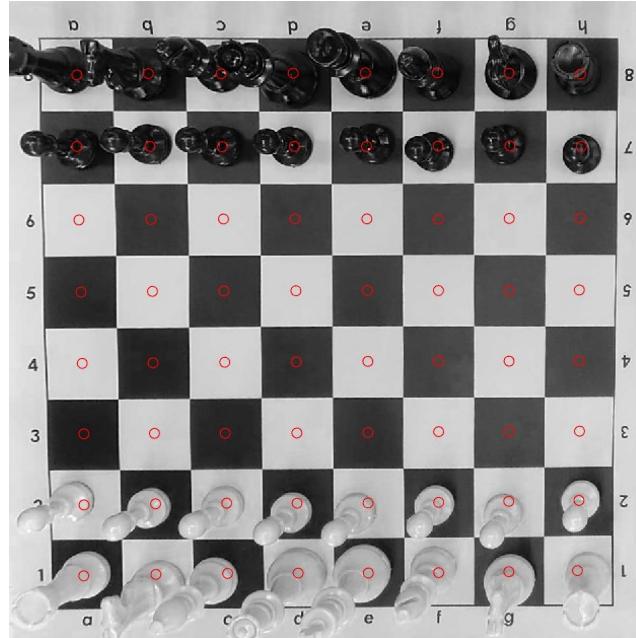


Figura 19: fotografia transformada on es mostren els punts obtinguts per *sortMat()*.

Identificades totes les posicions de les caselles, es pot fer un anàlisi individual de cada una d'elles. Com s'observarà en la figura (número de figura), es farà l'anàlisi d'un fragment de casella que es situe a la part esquerra de la casella. Aquesta és la posició més probable de trobar una peça ocupant una casella i que, a la vegada, la mateixa casella no tinga influència d'una peça veïna (com el cas del rei en d8 que ocupa part de c8) degut al punt de vista de la càmera i a la transformació homogràfica posterior.

Per una banda, s'ha d'identificar si una casella està ocupada o lliure. Per això, s'empra el filtre *Canny* ja que detecta gradients en la imatge i, en les caselles ocupades, estaran representades les peces d'escacs i, en les lliures, com tenen un color llis, no hi haurà cap gradient representant. Per tant, si dins del fragment seleccionat hi ha una quantitat de píxels blancs superior a determinat número (que es trobarà de forma experimental), la casella estarà ocupada. Del contrari, estarà lliure. Açò es comprovarà analitzant els nivells de l'histograma.

En aquest cas, si la quantitat de píxels negres és igual a 3526 (quantitat de píxels totals del quadre) o la quantitat de píxels blancs és menor a 110, el quadre estarà lliure. Si està ocupat, es procedeix a un segon anàlisi, explícitat a continuació.

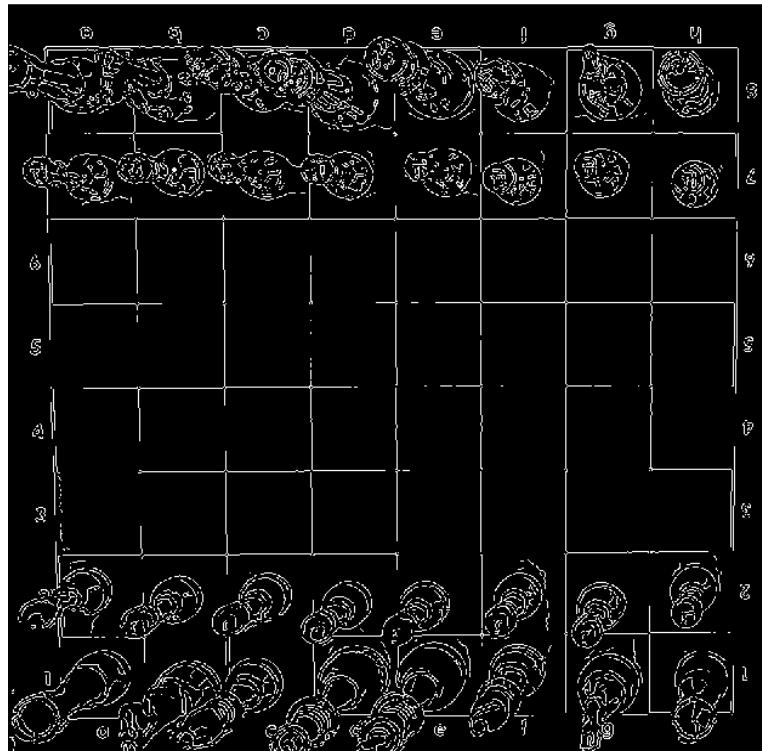


Figura 20: imatge del tauler obtinguda després d'aplicar el filtre canny.

Dins de les caselles ocupades, caldrà distingir el color de la peça que la ocupa. Interessarà fer una segmentació segons els colors de les peces. En aquest cas es fa un llindar en el nivell 25 per transformar la imatge del tauler en una imatge binaria. Com s'observa en la figura posterior, les caselles ocupades per les peces negres tenen píxels negres i, en canvi, aquelles ocupades per peces blanques tenen molts pocs píxels negres. Es farà un histograma de la imatge per decidir-ho.

En aquest cas, si la quantitat de píxels blancs és superior a 205, la casella estarà ocupada per una peça negra. Sinó, estarà ocupada per una peça blanca.

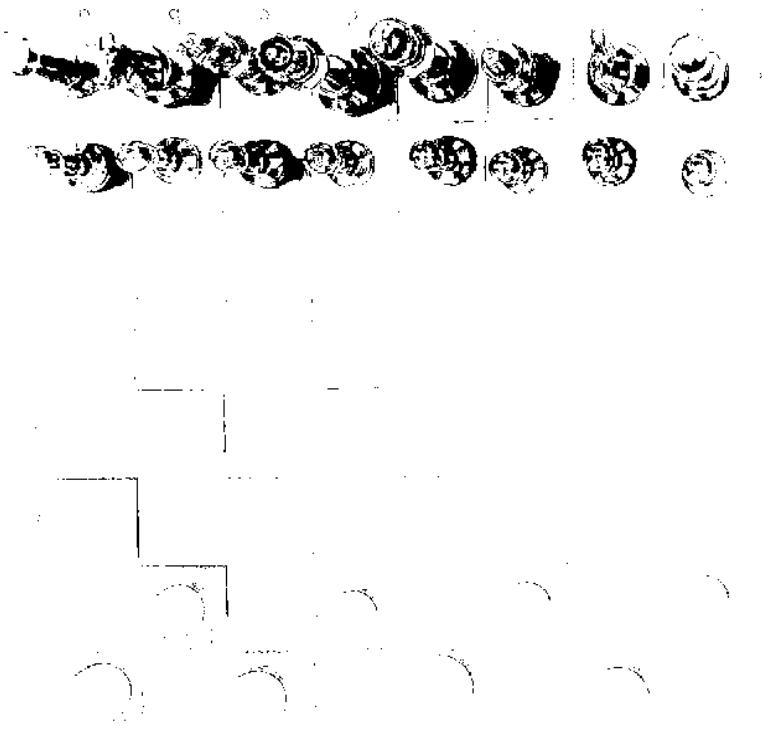


Figura 21: imatge del tauler obtinguda després d'aplicar un llindar (threshold) en el nivell 25.

A continuació, es mostra com es fa l'anàlisi de la casella **a1** per part de la funció *casellesOcupades()*. En primer lloc, es selecciona un fragment de la casella de 86x41 píxels a partir dels punts de cada casella, com es mostra en la figura següent.



Figura 22: fragment de la casella **a1** que s'emprarà per a analitzar el seu estat.



Figura 23: fragment de la casella **a1** quan s'aplica el filtre Canny.

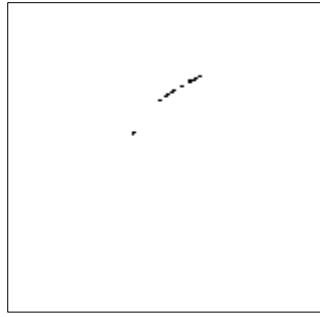


Figura 24: fragment de la casella **a1** quan s'aplica el llindar de nivell 25.

Aplicant els filtres corresponents, s'observa que, efectivament, la casella **a1** està ocupada i la peça és blanca, com figura en la imatge original.

3.3. Processat de dades en Matlab

Cal analitzar les dues fotos del tauler: la foto obtinguda abans de que el jugador faça una jugada i la que s'obté després de fer-la. Es faran servir els 64 punts obtinguts que es defineixen en la variable *punts_q*.

El tractament de la imatge es realitza en la funció de *casellesOcupades.m* on ha d'identificar l'estat de cada casella. El vector que torna aquesta funció, de 64 elements, tindrà un número o altra segons l'estat:

- 0: casella buida.
- 1: casella ocupada per una peça negra.
- 2: casella ocupada per una peça blanca.

La funció mencionada servirà per poder comparar l'estat actual del tauler (variable *c_Pos*), quan el jugador ja ha fet la jugada, i l'estat previ (variable *c_Prev*), quan el jugador encara no ha fet la jugada. La diferència entre *c_Pos* i *d_Prev* dóna lloc a la variable *c_Dif* que indica les variacions que han hagut en el tauler. Es pot saber que ha passat segons els valors dels elements de *c_Dif*:

- Si hi ha un element de valor -2 i un altre de valor 2: moviment d'una peça blanca.
- Si hi ha un element de valor -1 i un altre de valor 1: moviment d'una peça negra.
- Si hi ha un element de valor -2 i altre de valor 1: captura per part de blanques.
- Si hi ha dos elements de valor -1: captura per part de negres.
- Si hi ha quatre elements consecutius -2, 2, 2 i -2: enroc curt de blanques.

- Si hi ha quatre elements consecutius -1, 1, 1 i -1: enroc curt de blanques.
- Si hi ha cinc elements consecutius -2, 0, 2, 2 i -2: enroc llarg de blanques.
- Si hi ha cinc elements consecutius -1, 0, 1, 1 i -1: enroc llarg de negres.

En els dos primers casos anteriors, el valor negatiu representa la posició anterior d'una peça de determinat color mentre que el positiu fa referència a la casella a la que s'ha menejat la peça blanca anterior. Amb els possibles resultats que es donen lloc en C_Dif , es pot identificar el moviment realitzat per el jugador a partir d'aquest anterior vector i de l'estat actual c_Pos . Aquestes dues variables seran les de entrada de $detectarMov.m$.

La funció mencionada anteriorment estudiarà, en primer lloc, si s'ha produït un enroc. Com aquest moviment només és possible en uns determinats quadres, s'estudiarà si aquests tenen les variacions descrites en la enumeració on figuren els distints escenaris per a c_Dif .

- Enroc llarg de blanques: comprovació des de l'element 1 fins al 5.
- Enroc curt de blanques: comprovació des de l'element 5 fins al 8.
- Enroc llarg de negres: comprovació des de l'element 57 fins al 61.
- Enroc curt de negres: comprovació des de l'element 61 fins al 64.

Si no s'ha donat cap d'aquests escenaris, s'inicia un bucle *for* on s'analitza cada element de c_Dif per buscar la casella on anteriorment es situava la peça. Per això, ha de detectar un valor distint de zero en c_Dif i que, en la mateixa posició del vector, en c_Pos tinga un valor nul.

Per a passar de les coordenades en la matriu de la posició anterior de la peça a la nomenclatura LAN (Long Algebraic Notation), es fa servir un bucle *for* on s'identifica en quina columna del tauler es troba la peça. En aquest cas, es compleix la propietat de que les posicions del vector que comparteixen un mateix residu quan al número se li suma 7 i es divideix entre 8. Així, es té que, per exemple, totes les posicions a les que se li apliquen les operacions anteriors tenen un residu de 0 corresponen a la **columna a**; les que tenen un residu de valor 1 corresponen a la **columna b**; i, així, fins que totes les posicions que tenen un residu de 7 corresponen a la **columna h**.

Abans de procedir amb la explicació, cal entendre que la notació LAN especifica cada moviment indicant la casella d'origen de la peça i la casella de destí. En el cas d'un peó de blanques que va de **e2** a **e4**, el seu moviment en LAN serà '**e2e4**'. En canvi, si la reina de negres (partint de la casella **d8**) capturen a un peó de blanques que està en **d5**, el moviment serà indicat com '**d8d5**'.

Una vegada es té la columna, es procedeix a assignar la fila corresponent. La propietat que es compleix ací és que cada sèrie de 8 números es relacionen a una fila. Per exemple, les posicions del vector que van de l'1 al 8 es refereixen a la **fila 1**; les que van de la posició 9 a la 16 estan situades en la **fila 2**; i, seguidament, les posicions compreses entre 57 i 64 estan en la **fila 8**.

Per trobar la posició de destí de la peça, es troba quina posició del vector comparteixen c_Dif i c_Pos on tenen un valor distint de 0, procedint de manera similar que en el cas de la detecció de la posició anterior. També s'empra el mateix mètode per a afegir en l'string de notació LAN la posició de destí de la peça moguda per el jugador, tenint el moviment del jugador detectat per complet.



Figura 25: moviment del peó blanc i els vectors c_Prev , c_Pos i c_Dif .

Nota: els valors dels vectors han sigut ordenats d'esquerra a dreta i de baix fins amunt per un millor entendiment de l'exemple ficat.



1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	0	0	-1	0	0	0	0
1	1	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	2	0	0	2	2	2	2	2	2	2	2	2	2	2	0	0	2	2	2	0
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	0	0	0

Figura 26: moviment de captura de la reina negra i els vectors c_Prev , c_Pos i c_Dif .

Per tot açò, coneixent la situació inicial de joc i coneixent les variacions de moviment, son suficients per tindre l'estat actual del tauler i les posicions de cada peça del mateix.

3.4. Elaboració d'un efector final

La pinça que té el robot de sèrie no és suficient com per a recollir les peces d'escacs, a més de no ser versàtil amb totes les peces (quan està tancada, la distància és de 15 mm i, oberta, de 27 mm). És per això que es fa necessari el disseny d'un complement per a aquest efector final. Per al disseny, cal tindre en compte que la secció més estreta a agafar per part de la pinça elaborada és de 16 mm (sen la part superior del peó); i, per altra part, el diàmetre més gran que s'agafarà és de 26 mm (sent les parts més prominents de les parts superiors del rei i la reina).

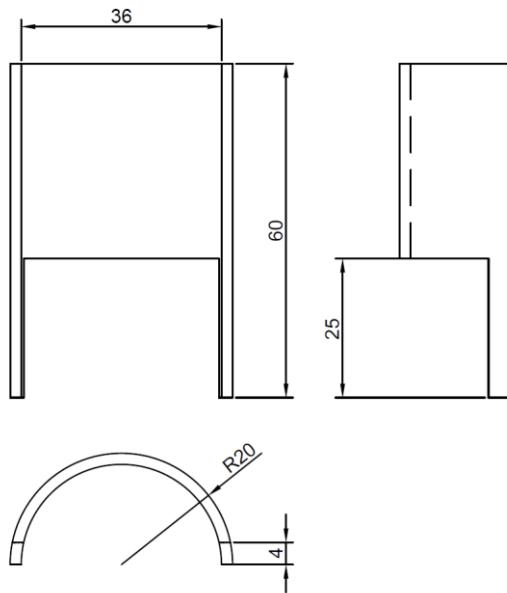


Figura 27: tub de PVC resultant despès de fer els talls necessaris (unitats en mm).

Es requereix d'un tub PVC de 40 mil·límetres de diàmetre on caldrà tallar-lo per la mitat i fer els talls corresponents per obtenir una peça com la de la figura anterior. D'aquesta manera, s'evitarà que el tub col·lisione amb les peces veïnes quan agafe una determinada peça.

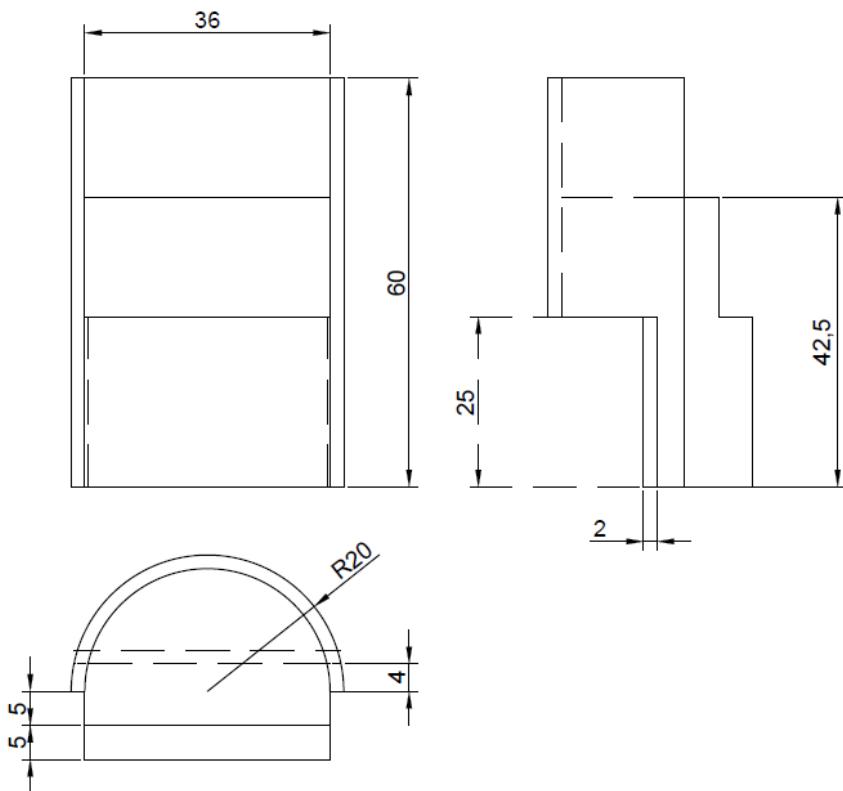
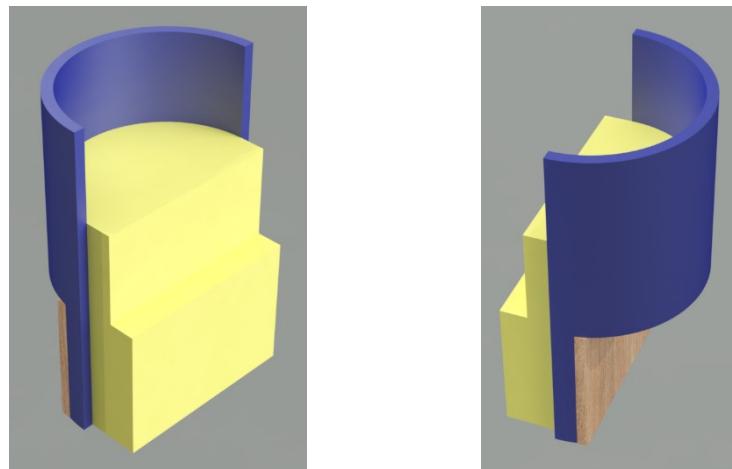


Figura 28: tub de PVC retallat amb el fragment d'esponja pegat (unitats en mm)..

Com es veu en la figura 28, es pegarà una esponja que siga més gran que les dimensions especificades en el plànol. L'excedent, en aquest cas, s'ha anat retallant fins que les dimensions de la esponja han permès que es puga agafar qualsevol peça i, a més, ninguna es quede enganxada amb la esponja quan s'obri. La esponja, com es detalla en la figura anterior, sobresurt 5 mm i altes 5 mm en la part inferior.

A més, s'ha de ficar un reforç en la part inferior del tub tallat per a que la esponja no es llisque cap darrere quan la pinça es tanque. Per a aquest cas, s'ha emprat una xapa de fusta de 2 mil·límetres de grossor.



Figures 29 i 30: peça dissenyada.

Amb dues peces com la anterior, s'enganxaran a la pinça del robot aprofitant l'espai lliure on no hi ha esponja. En aquest cas, s'ha fet ús de cinta aïllant per fixar-les en el lloc per ser la opció més simple i barata. A més a més, l'efector final en conjunt té una massa de 46 grams aproximadament i no afecta a les prestacions del robot.



Figura 29: detall de la pinça amb les peces fixades.

3.5. Implementació del robot

La intervenció de la intel·ligència artificial en el tauler es farà mitjançant el robot *UR3* d'*Universal Robots* (com apareix en la figura 10). Abans caldrà identificar els punts clau del tauler (de posició fixa) de forma manual. Per a aquest cas, s'ha hagut d'emprar dues orientacions distintes degut a les limitacions de moviment del propi robot com, per exemple, la col·lisió amb ell mateix quan es situa en zones properes de la casella **a8**.

Les dues disposicions del canell robot es defineixen en *ori_1*, on les rotacions en x, y i z son 0.866, 3.4821, i -0.2418 respectivament; i *ori_0*, on les rotacions en x, y i z son 3.0665, -0.6946, i 0.7061 respectivament.

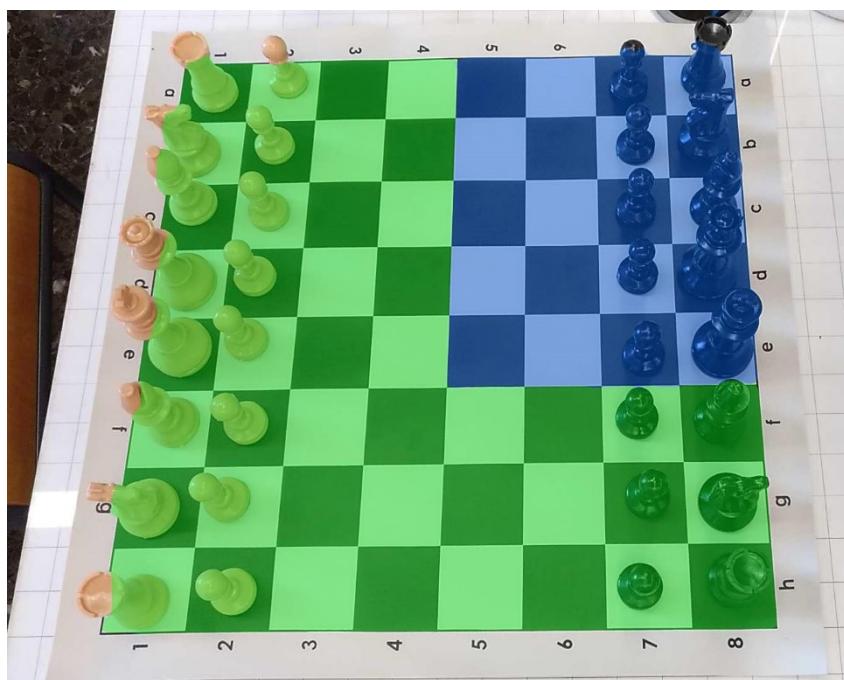


Figura 30: en verd, la zona del tauler on la orientació correspon a *ori_1* i, en blau, la zona del tauler on la orientació correspon a *ori_0*.

Donada aquesta situació, s'ha de trobar, manualment, els punts claus del tauler per definir la resta de posicions del robot en el tauler. Per al cas de la orientació *ori_1*, serà suficient amb **a1**, **h1**, i **a4**. Per a *ori_0*, es necessitarà conèixer **a5**, **e5** i **a8**. A partir de les caselles anteriors, es poden deduir les coordenades de les restants calculant la diferència entre distàncies. En les columnes de la zona verda, la distància de casella a casella serà la diferència de les coordenades en y d'**a4** i **a1** entre tres; i, la distància en x d'entre quadre i quadre serà la diferència entre **h1** i **a1** dividida entre set. Es procedeix de manera similar en la zona blava. Com es vorà en el codi mostrat a continuació, s'han redefinit, experimentalment, els valors de les coordenades en y de *dif_M_h* i *dif_M_h_1* a 0 per a

millorar la precisió del robot. De la mateixa manera, s'ha procedit en les coordenades en x de $dif_M_v_1$.

Tot aquest procés es fa mitjançant la funció *puntsRobot.m*.

```
% Distància entre casella i casella en les files de ori_1
dif_M_h = (M(2,1:2)-M(1,1:2))/7;
% S'anula la component en y per minimitzar les impresicions.
dif_M_h(2) = 0;
% Distància entre casella i casella en les columnes de ori_1
dif_M_v = (M(3,1:2)-M(1,1:2))/3;
puntsR = [];
% c és una variable que determina la posició de la que es parteix per a
% anar determinant les coordenades de cada fila. Quan es completa una, se
% li suma la distància per casella de les columnes i ompli la fila.
c = M(1,1:2);
% Variable que contarà la posició en la que es situa en cada fila.
d = 0;
% Bucle per a completar mig tauler (d'a1 fins a h4).
for i=1:32
    if mod(i-1,8)==0 && i>1
        c = M(1,1:2)+((i-1)/8)*dif_M_v;
        d = 0;
        puntsR = [puntsR; c+d*dif_M_h];
    elseif i == 1
        puntsR = M(1,1:2);
    else
        d = d+1;
        puntsR = [puntsR; c+d*dif_M_h];
    end
end
puntsR = [puntsR; M(5,1:2)];
% Distàncies entre casella i casella en les files de ori_0.
Es
% procedeix de manera similar.
dif_M_h_1 = (M(6,1:2)-M(5,1:2))/4;
dif_M_h_1(2) = 0;
dif_M_v_1 = (M(7,1:2)-M(5,1:2))/3;
dif_M_v_1(1) = 0;
d_1 = 0;
c_1 = M(5,1:2);
% Bucle per a completar la resta del tauler.
for i=34:64
    if i==64
        puntsR = [puntsR; M(4,1:2)];
    elseif mod(i-1,8)==0
        c_1 = M(5,1:2)+((i-33)/8)*dif_M_v_1;
        d_1 = 0;
        puntsR = [puntsR; c_1+d_1*dif_M_h_1];
    elseif mod(i-1,8)>=1 && mod(i-1,8)<=4
```

```

d_1 = d_1+1;
puntsR = [puntsR; c_1+d_1*dif_M_h_1];
elseif mod(i-1,8)==5
c = M(1,1:2)+((i-6)/8)*dif_M_v;
d = 5;
puntsR = [puntsR; c+d*dif_M_h];
elseif mod(i-1,8)>=6
d = d+1;
puntsR = [puntsR; c+d*dif_M_h];
else
end
end

```

Aquests punts, els de la variable *puntsR*, poden associar-se a cada moviment que fa la IA, podent identificar-se emprant la funció *CM.GetLANstrs()*. Aquesta ens torna una matriu de *cel·les* on, cada una d'elles, té un *string* de l'últim moviment que s'ha realitzat en el tauler en notació LAN (Long Algebraic Notation), siga de la IA o del jugador. Si l'últim moviment ha sigut, per exemple, un enroc curt de blanques, l'últim *string* de *CM.GetLANstrs()* serà '*e1g1*'; i, si s'haguera avançat el peó del rei de les negres dos quadres, l'*string* corresponent a eixe moviment seria '*e7e5*'.

Per altra part, s'ha d'analitzar si en el moviment de la peça ha hagut una captura. Per a això, s'utilitza la funció *CM.GetSANstrs()* que torna un *string* (relacionat en la variable *pos_ult* en *joc.m*) en notació SAN (Standard Algebraic Notation), la més emprada en els escacs. Aquesta, en diferència de la notació LAN, en el propi *string* indica si s'ha fet una captura o no indicant-ho amb una 'x'. El valor numèric de la 'x' és igual a 120. Per tant, si el valor de *pos_ult* és igual a 'c4' (el peó de blanques en **c2** avança a **c4**) i es fa la operació *pos_ult == 120*, tornarà un vector de valor [0,0], sent *sum(pos_ult==120)* igual a 0. Si, per contrari, *pos_ult* és igual a 'Qxd5' (la reina de les negres capturen en d5) i s'aplica la operació anterior, es tornarà un vector de valor [0,1,0,0], sent *sum(pos_ult==120)* igual a 1.

Si hi ha captura de peça, comprovant la condició descrita anteriorment, s'actualitzarà la variable *captura* al valor de 1, indicant al robot en *menejaRobot()* que ha de llevar la peça que capturen en primer lloc i, després, ficar l'altra peça. També intervé altra variable anomenada *captures* per evitar que una peça es fique damunt d'una altra actualitzant la variable de posició de les peces captures.

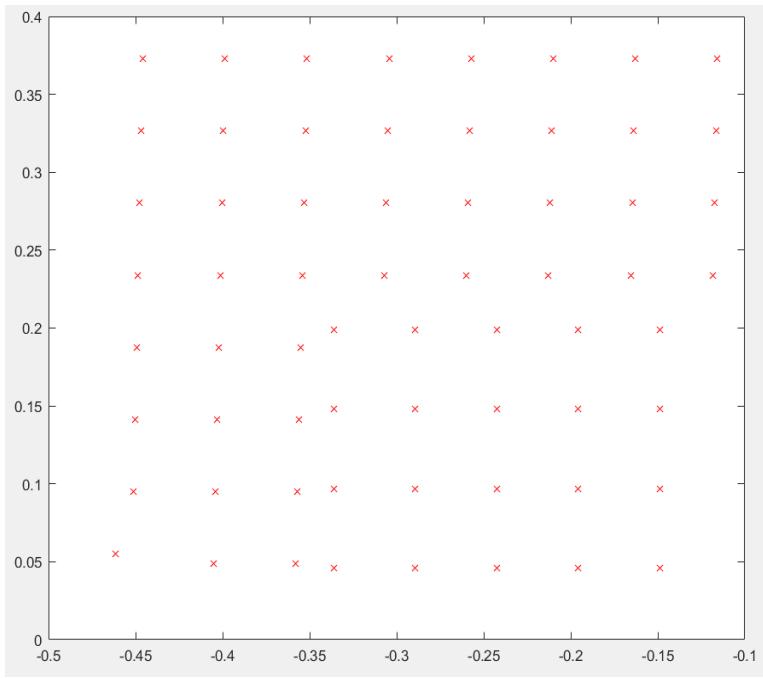


Figura 31: coordenades dels punts del tauler on es dirigirà el robot per agafar les peces.

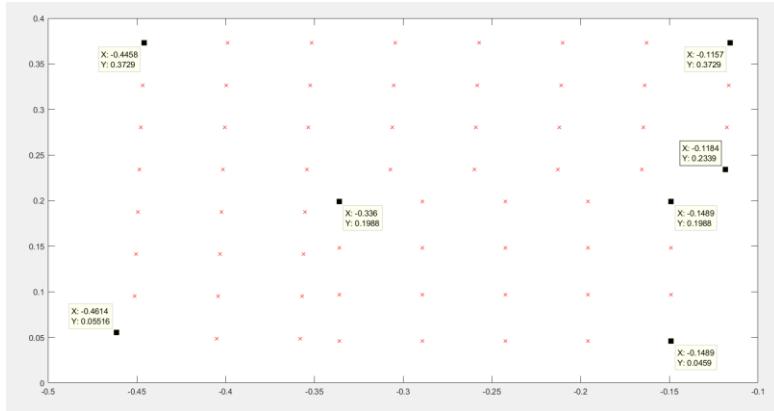


Figura 32: coordenades anteriors marcant els punts **a1, h1, a4, a5, e5, a8 i h8** (ordenats de dreta a esquerra i de dalt cap a baix).

En els moviments del robot per el tauler es duen a terme amb la funció *fprintf()* que rep la variable del tipus TCPIP i un *string* que li indica al robot el tipus de moviment (moviment *joint* i moviment *lineal*); les coordenades de les posicions en x, y i z; les coordenades de rotació en x, y i z; i si la pinça està oberta o tancada (indicant-ho amb un 0 o amb un 1 respectivament).

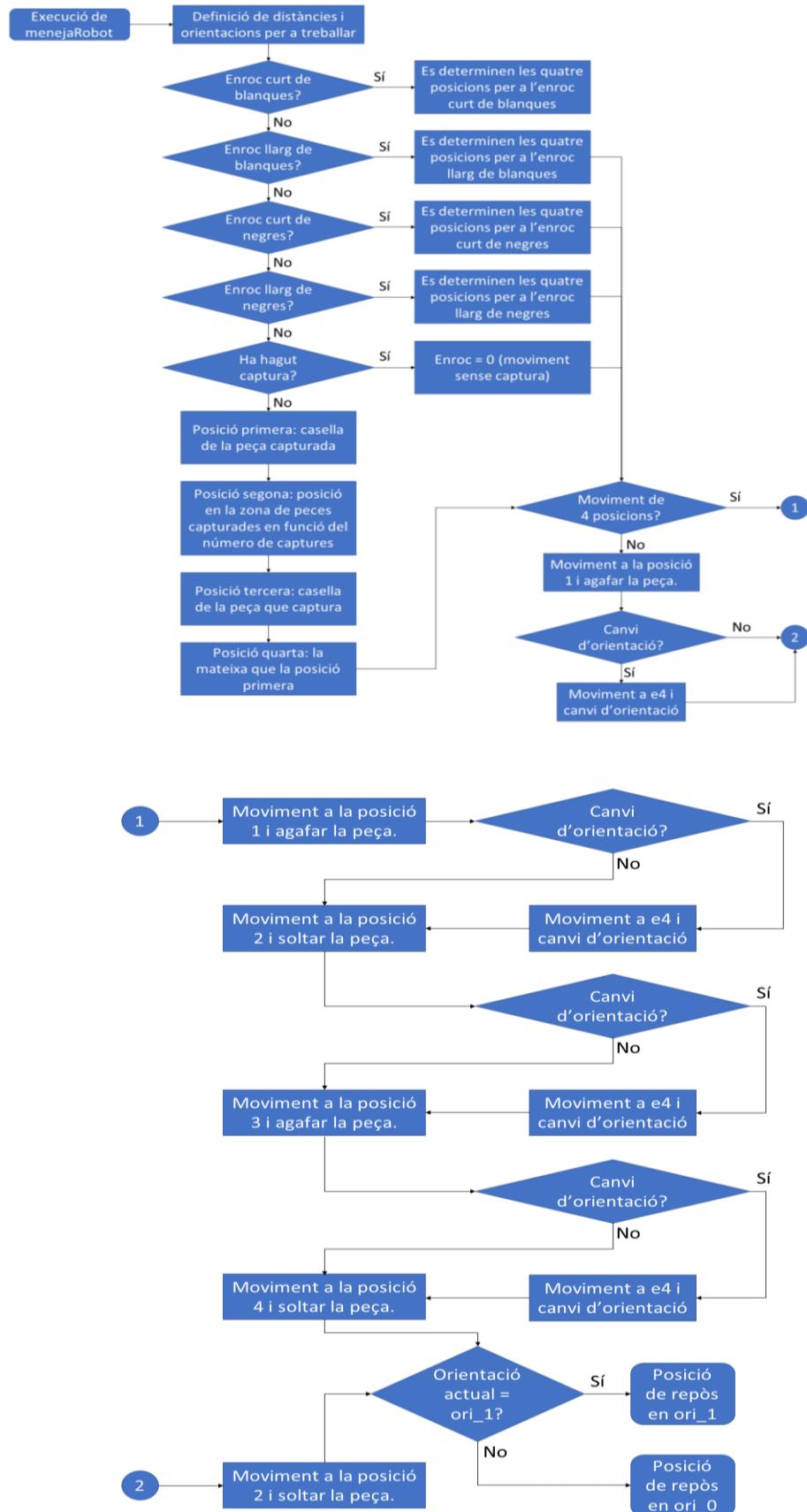


Figura 33: diagrama de flux del funcionament de la funció *menejaRobot()*.

Al principi es defineixen les altures òptimes d'operació on els moviments de tipus J es fan a una distància de 50 mm en les coordenades z respecte al sistema de referència de la base. De la mateixa manera, les peces s'agafen a una altura de -122 mm i es solten a una altura de -116 mm. De seguit, també es defineixen *ori_1* i *ori_0* i la posició de la casella e4 donada per la matriu d'entrada.

Com es pot observar en el diagrama anterior, es defineixen les posicions de destí del robot en les coordenades x i y de cada casella de destí on el robot menejarà les peces. Tant les captures com els enrocs funcionen de la mateixa forma ja que el robot ha de menejar dues peces en els dos casos.

En el cas de les captures, la segona posició es defineix a partir del nombre de captures on les peces capturades s'agrupen en files de tres columnes. Entre elles es deixa la distància suficient per evitar col·lisions i bloquejos del robot.



Figura 34: distribució de les peces capturades per el robot.

Tenint en compte el diagrama de flux de *menejaRobot*, el braç passa de posició de repòs fins situar-se per dalt de la casella de destí (canvia d'orientació si la seu posició de repòs i la de destí tenen orientacions distintes). A continuació, baixa a la casella fent un moviment lineal i tanca la pinça quan aplica a la posició en les coordenades z. Es mou a la posició anterior i va a la casella de destí mantenint la altura. Després baixa a la posició de -116 mm on solta la peça. Finalment, torna a pujar a la altura de 50 mm i va a la posició de repòs que dependrà de la seu última orientació.

3.6. Codi principal: *joc.m*

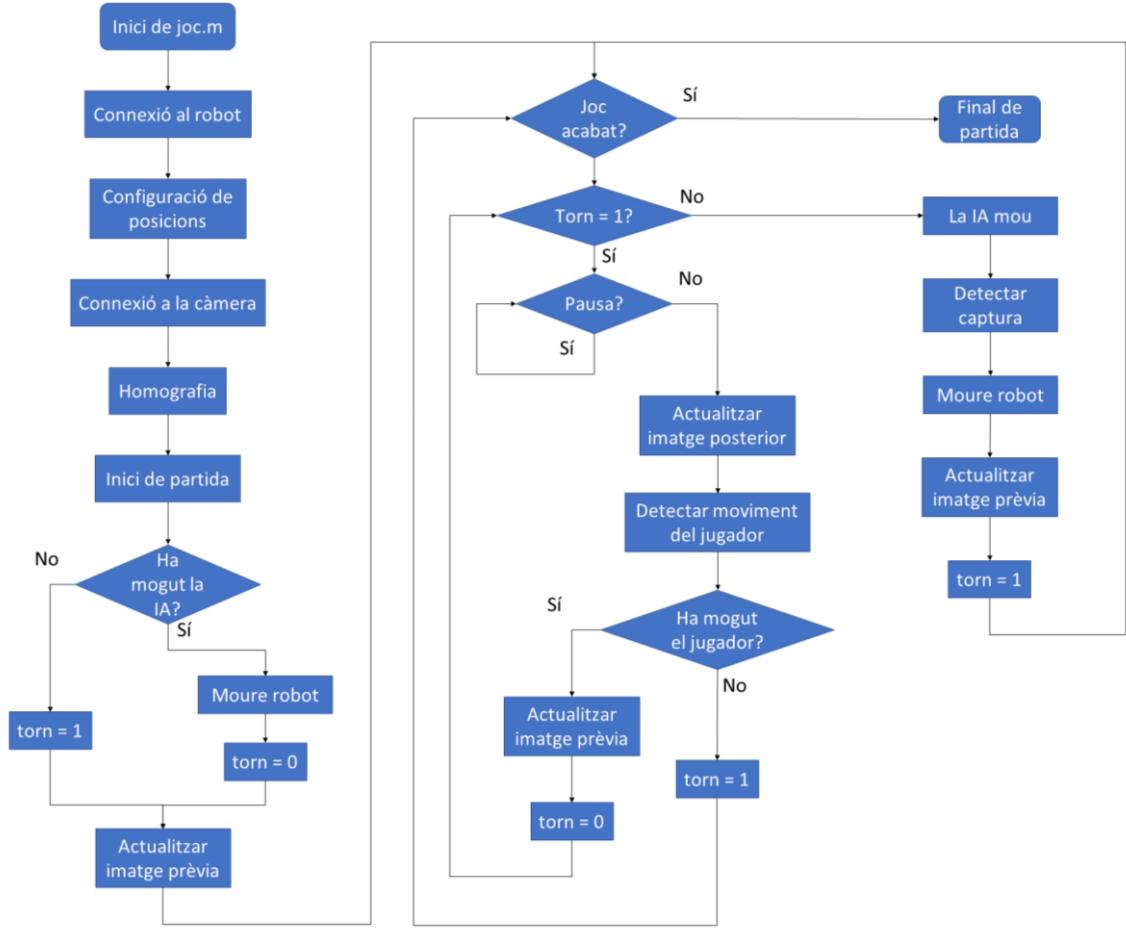


Figura 35: diagrama de flux de *joc.m*

Després de que tota la configuració siga realitzada, incloent la homografia, s'inicia la interfície gràfica d'usuari *Chess Master* igualant *CM* a *ChessMaster()*. D'aquesta manera, *CM* és una classe a la que se li apliquen les funcions de *ChessMaster* i, així, pot interactuar en la interfície i facilitar les dades necessàries per al seu processament.

A continuació, el jugador ha de configurar la IA com es descriu en l'apartat 2.5 i escollint el color al que juga. Una vegada fet això, es desactiva la pausa prement qualsevol tecla de l'ordinador. Aleshores, es tindran dos escenaris:

- El jugador ha escollit jugar en blanques: es defineix que es torn del jugador.
- El jugador ha escollit jugar en negres: en aquest cas, el jugador esperarà a que la IA faça el primer moviment i, seguidament, serà detectat per a que el robot el duga a terme.

Quan acaba el robot o el jugador desactiva la pausa, es fa la captura prèvia al moviment del jugador on apareixerà el tauler en la posició inicial o amb el moviment de la IA.

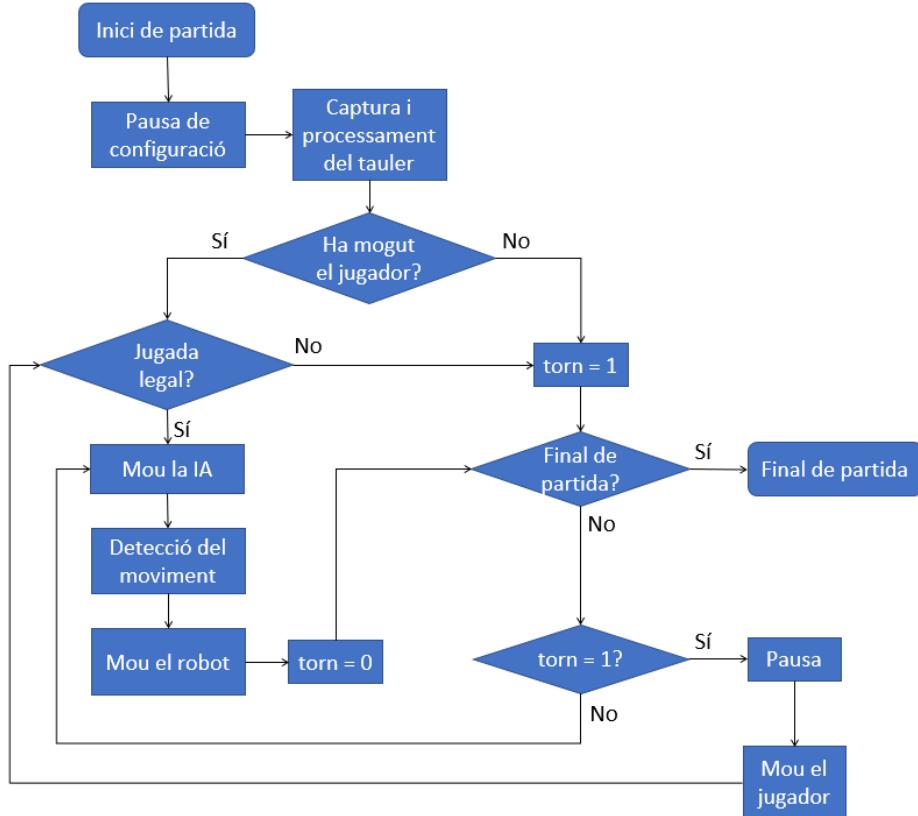


Figura 36: diagrama de flux de la partida.

La resta de la partida ocorrerà dins del bucle que s'inicia en la línia 111 de *joc.m* i es comprova si la partida està acabada, detectant els escenaris d'escac i mat o de tables. La variable que canvia de 0 a 1 en aquestes situacions és *CM.isGameOver*.

De seguit, es comprova si el torn és del jugador o de la IA amb la variable *torn* que té un valor de 1 si li toca moure al jugador o de 0 si li toca moure a la IA. Aquesta variable sempre canvia quan el jugador i el robot acaben de fer la jugada o, bé, si durant el torn del jugador no es detecta cap moviment i es torna al principi del bucle sense canviar el valor de la variable de torn.

En els apartats anteriors queden descrits els processos que es duen a terme per a la comunicació òptima entre la càmera, el robot i l'ordinador. Durant el torn del jugador es fa una captura del tauler i es fa altra quan s'acaba de fer el moviment. Amb l'anàlisi de les dues imatges es rèplica el moviment en la intereficie gràfica. Mentre que, durant el torn del robot, s'extrau el moviment fet per la IA i el robot fa els moviments corresponents.

4. Resultats i conclusions

4.1. Proves en l'adquisició i la homografia

En els mètodes de selecció de tauler, s'ha hagut de determinar les condicions inicials d'homografia amb el tauler sense peces ja que no es podien detectar els quadres amb la presència de les peces (llevant de la detecció de línies).

Entre els distints mètodes per detectar el tauler i fer la homografia, es troben els següents:

- Detecció de les línies del tauler.
- Trobar els punts en comú de dues imatges distintes.
- Emprar la funció *cv.findChessboardCorners()*.

Entre tots els anteriors, s'ha optat per la última opció per ser el més robust en aquestes condicions i, a més a més, perquè torna una matriu de punts de 7x7 per a les dues imatges (la del tauler pla i la del tauler en perspectiva). Per això, aquesta és la solució més simple.

El primer mètode va obtenir resultats decentment sòlids inclús havent peces en el tauler. Malgrat, la seu desqualificació es deguda a la difícil extracció de dades útils per a treballar amb elles.

Per altra part, el segon mètode funcionava de manera semblant que l'últim. Malgrat, té l'inconvenient de presentar una complexitat major d'implementació.

S'ha comprovat la efectivitat de la funció *cv.findChessboardCorners()* en el mateix escenari i situant la càmera en perspectives distintes, mantenint la casella **h1** situada a la esquerra de la imatge. També es té la necessitat de que el tauler ocupe tot el que puga de la imatge per a tindre una major resolució.

4.2. Proves en el processament de dades de la imatge

Per a comprovar el correcte funcionament de les funcions implementades per a la visió, s'han emprant unes 47 captures de tauler on cada una d'elles representa un estat de tauler de la partida d'Aronian contra Anand de gener del 2013. Aquesta fou escollida com a mostra per validar el funcionament de les funcions implementades per ser una partida breu, amb enrocs per part dels dos jugadors i perquè les peces blanques i negres es troben més relativament properes.

En aquest cas, s'han verificat les funcions *casellesOcupades.m* i *detectarMov.m* imatge a imatge (amb un bucle for) de forma manual, comprovant que les matrius resultants

coincideixen amb les posicions de les peces en el tauler i que els valors de les variables de tipus *string* generades per la funció corresponguen a les variacions de moviment d'estat a estat.

Una vegada els filtres estan ajustats correctament amb totes les imatges (amb una sensibilitat del filtre Canny de 0.06 i amb un llindar en el nivell 25), s'estudien les situacions de moviments, captures i enrocs per a ambdós colors amb la funció *detectarMov.m* amb una simulació real. Com aquesta última funció depèn de la correcta detecció de les caselles ocupades i lliures, també es verifica que les dues funcionen.

4.3. Proves amb el robot

En aquest cas, partint dels punts que proporciona la funció *puntsRobot()*, es fa la prova de *menejaRobot()* fent els distints tipus de moviment:

- Moviments i captures en les zones *ori_1* i *ori_0* dins de les mateixes zones.
- Moviments i captures de la zona *ori_1* a la zona *ori_0* i viceversa.
- Enrocs llargs i curts en els distints colors.

En els dos tipus proves, s'ha fet ús de distinta peces per assegurar el correcte funcionament del complement dissenyat per a la pinça. Tanmateix, la varietat de condicions ha servit per a determinar una altura on el robot puga agafar totes les peces sense que caiguen i que, a la vegada, les més altes no puguen vore afectades (col·lisió amb el robot o canvi brusc de la orientació quan es agafada).

També, com es descriu en l'apartat 3.5, s'ha determinat de forma experimental la altura òptima on el robot treballa en *moveJ* sense tindre cap dels problemes que s'han descrit.

Finalment, abans de combinar-ho amb tota la part d'adquisició i processament la imatge, s'han mogut les peces del tauler tant en la interície com en el físic. La correcta execució dels moviments que fa el robot en base de la IA, permeten iniciar la execució de totes les parts en conjunt.

4.4. Problemes trobats i solucions

4.4.1. Imprecisió del robot a l'hora de situar les peces

En la execució de tot el conjunt, es va detectar que el robot no deixava les peces de forma precisa en el tauler ja que les peces no es situaven en una part central del quadre i provocava col·lisions entre la pinça i la peça.

Tanmateix, s'ha detectat que les files i columnes presenten desviacions per la errada acumulada en les operacions de càlcul de les posicions de cada quadre. Per aquest motiu, s'han assignat valors en distin tes components de dif_M_h , dif_M_v , $dif_M_h_1$ i $dif_M_v_1$; explicat en l'apartat 3.5.

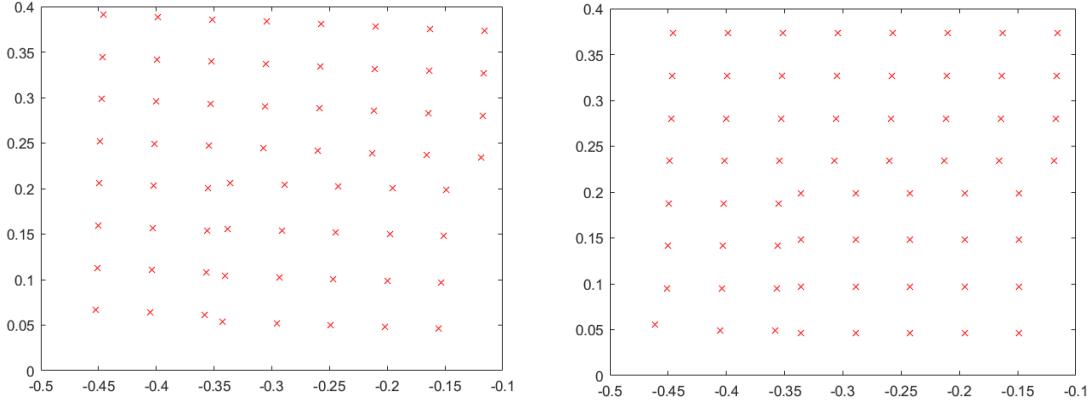


Figura 37: punts de les coordenades del robot sense corregir (esquerra) i corregides (dreta).

4.4.2. Caiguda de les peces agafades

Durant les proves amb el robot i amb la prova general de tot el conjunt, s'ha detectat que algunes peces caien quan eren agafades. Les més freqüents eren els peons i els cavalls degut a que l'efector final tancat no disposava del volum d'esponja suficient.

Com s'indica en la figura (...) de l'apartat 3.4, s'ha fet necessari l'afege ment d'esponja addicional en la part inferior sense comprometre a les peces més grans i grosses com el rei, la reina o la torre. A més del retoc anterior comentat, també s'ha retallat l'excedent suficient per a que, quan s'agafen altres peces, aquestes no romanguen en la pinça quan s'obri.



Figura 38: detall del problema comentat anteriorment. Malgrat que la pinça està oberta, la reina segueix en ella.

4.4.3. El robot s'enrotlla amb els tubs

Durant la execució del codi *joc.m*, s'ha detectat que els canvis d'orientació del robot provoca que els tubs s'enrotllen en cada canvi d'orientació fins que la tensió generada bloqueja el funcionament del braç robòtic.

4.4.4. Incorrecta definició del lloc on el robot deixa les peces capturades.

En les proves generals, s'ha detectat una incorrecta implementació en les posicions del robot quan ha de deixar les peces capturades i distribuir-les de forma òptima. Aquest fet es reflexa en el vídeo adjuntat on s'observa que el robot no deixa les peces en les posicions corresponents.

La solució al problema passa per reajustar les operacions presents entre les línies 70 i 80 de la funció *menejaRobot.m* i verificar-ho experimentalment. Després d'uns quants intents, s'aconsegueix que les peces queden agrupades en tres columnes.



Figura 39: peces capturades després de la correcció.

Una alternativa per a aquest imprevist hauria sigut la col·locació d'una caixa per a que el robot només haja d'anar a una única posició i deixar caure la peça. Però no s'ha optat per aquesta degut a que el jugador no pot vore, a colp de vista, les peces capturades per la IA.

4.4.5. Detecció errònia de les jugades fetes per el jugador.

En aquest cas, si la persona que jugava feia, per exemple, un moviment '*c3d5*' i, anteriorment, la IA havia fet un moviment '*d5d4*', el moviment detectat per el jugador passava a ser '*c3d4d5*' i no es podia continuar la partida amb normalitat. En el cas de que la IA haguera fet el moviment '*e7e6*', el moviment detectat era *c3d5e6*'. Malgrat que el

resultat és incorrecte, la partida podia seguir amb normalitat ja que *detectarMov()* treballa amb els primers quatre valors de l'string, corresponent al moviment del jugador.

Aquesta errada havia de vore en la detecció de peces i les caselles ocupades. De forma manual, es comprovaren els moviments conflictius amb èxit. Així que es va descobrir que la errada estava en la execució de *joc.m*. Quan la funció de *menejaRobot()*, de la línia 154 de *joc.m*, s'iniciava, es feia a la mateixa vegada la actualització de la imatge prèvia abans de que el robot haguera acabat el seu torn. La evidència d'aquest fet es troava amb la mostra del missatge de la línia 164 abans de tornar al principi del bucle *while*.

Per solucionar-ho, s'ha introduït una pausa manual que l'usuari desbloquejaria quan el robot acabara de fer els moviments. Malgrat això, una solució més òptima haguera sigut que el robot envie un missatge quan s'acabe d'executar un moviment complet.

4.5. Temps aproximats d'execució

Tenint en compte les distintes distàncies que pot fer el robot i que aquest ha treballat en una velocitat del 50 %, s'obtenen les següents duracions al que respecta cada tipus de moviment:

- Processament del moviment de jugador i enviament de dades al robot: 2.3 segons.
- Càcul de jugada per part de la IA: 3 segons.
- Moviment sense captura (sense canvis d'orientació): de 18 a 23 segons.
- Moviment sense captura (amb un canvi d'orientació): de 23 a 27 segons.
- Moviment amb captura (sense canvis d'orientació): de 30 a 36 segons.
- Moviment amb captura (amb un canvi d'orientació): de 32 a 28 segons.
- Moviment amb captura (amb dos canvis d'orientació): de 55 a 60 segons.

4.6. Conclusions

Malgrat totes les dificultats, el problema ha quedat resolt de manera eficient en els recursos disponibles, sent tot el conjunt del sistema capaç de dur a terme una partida d'escacs contra una altra persona de forma segura.

La implementació del conjunt del sistema no ha sigut la més òptima degut a la falta de recursos i que, a més, han tingut lloc un conjunt de limitacions que no hagueren estat presents amb la elecció de processos més costosos com:

- El disseny d'un efector final amb impressió 3D.
- La impossibilitat de seleccionar un robot amb un major rang de moviment per a una mateixa orientació del canell.

Per la part del canell, hauria sigut el disseny d'un efector final amb un comportament similar al de una pinça d'estendre. D'aquesta manera, la interferència sobre les peces veïnes quan s'intenta agafar una peça no tindria lloc. A més a més, les peces agafades tindrien una major adherència al robot, permetent una major velocitat de moviment sense que la peça caiga o es canvie d'orientació per la falta de força de fregament.

El robot escollit, malgrat que necessita que el canell canvie d'orientació, ha sigut la millor opció entre tots els recursos disponibles. Encara que el model UR5 siga més gran, és prou més barat (en diners i temps) aplicar un canvi d'orientació que fer una compra d'un model i muntar-lo. Contemplant les prestacions i cost, un robot cartesià haguera sigut la millor opció. Però, aquest últim s'ha descartat per no ser segur perquè no permet la interacció segura amb les persones (a diferència del robot col·laboratiu).

La implementació d'aquest sistema no podria quedar la seu aplicació limitada a jugar partides d'escacs. Encara que no es compte amb una mateixa IA ni en la mateixa interfície d'usuari, aquest sistema pot extrapolar-se en aplicacions en l'àmbit industrial o de l'oci com, per exemple:

- Una càmera que ha d'identificar els objectes depositats en una estanteria i ordenar-los segons uns determinats criteris determinats.
- Reordenar un conjunt d'elements que queden depositats arbitràriament en una bandeja per a, després, ser entregats amb una presentació adequada.
- Implementació d'un braç robòtic de joguina i interactiu dirigit a xiquets per a l'aprenentatge de formes i colors. Tanmateix, pot ser útil per a persones amb infermetats neuro-degeneratives.
- Joc de dames.
- Joc del backgammon.

5. Referències

1. L. Parziale, W. Liu, C. Matthews, N. Rosselot, C. Davis, J. Forrester, D. T. Britt. TCP/IP Tutorial and Technical Overview. IBM Redbookks, 2006.
2. P. Corke. Robotics, Vision and Control: fundamental Algorithms in MATLAB. Berlin, Germany: Springer, 2013.
3. <https://es.mathworks.com/help/instrument/tcpip.html>
4. <https://es.mathworks.com/help/instrument/communicate-using-tcpip-server-sockets.html>
5. <https://www.mathworks.com/help/images/ref/edge.html>
6. Y. Xie, G. Tang, W. Hoff. Geometry-Based Populated Chessboard Recognition.
7. <https://stockfishchess.org/about/>
8. <https://es.mathworks.com/help/supportpkg/usbwebcams/ug/acquire-images-from-webcams.html>
9. <https://www.chess.com/games/view/13459415>
10. <https://www.universal-robots.com/es/productos/robot-ur3/>
11. <https://github.com/kyamagu/mexopencv>
12. Normes d'escacs de la FIDE (Federació Internacional d'Escacs) vigents des de l'1 de gener del 2018.
<https://www.fide.com/fide/handbook.html?id=208&view=article>

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria del Disseny

**DESENVOLUPAMENT D'UN SISTEMA AUTOMÀTIC
PER A JUGAR A ESCACS MITJANÇANT ROBÒTICA
COL·LABORATIVA I VISIÓ ARTIFICIAL.**

ANNEXES

TREBALL FINAL DEL

Grau en Enginyeria Electrònica Industrial i Automàtica

REALITZAT PER

Lluna Sanz Montrull

TUTORITZAT PER

Eugenio Ivorra Martínez i Antonio José Sánchez Salmerón

DATA: **València, setembre, 2019**

Índex

1.Codi emprat i funcions	3
1.1.joc.m.....	3
1.2.sortMat.m.....	6
1.3.casellesOcupades.m.....	7
1.4.detectarMov.m	8
1.5.puntsRobot.m.....	10
1.6.menejaRobot.m	11
2.Diagrames de flux	17

1. Codi emprat i funcions

1.1. joc.m

```
%% Joc
%% Iniciem servidor entre el pc i el braç
t = tcpip('0.0.0.0', 3000, 'NetworkRole', 'server');
    % Es crea un objecte TCPIP.
    % Client: 158.42.206.10
t.terminator="";
disp("Esperant la connexió del client");
%Espera la conexión
fopen(t);
% Llegir el missatge rebut del del robot
data = fscanf(t);
%data = fread(t, t.BytesAvailable);
disp(data)

% Orientació del braç en la zona d'a1.
ori_1 = [0.866,3.4821,-0.2418];
m_str = sprintf("[0,-0.46,-
0.095,0.05,%f,%f,%f,0]",ori_1(1),ori_1(2),ori_1(3));
fprintf(t,m_str);

%% Definició de posicions del robot en el taule
% Punts de a1, h1, a4, h8, a5, e5 i a8.
    % La tercera columna representa la orientació del robot segons en
la
    % zona que es trobe.
punts_r_4 = [-0.1157 0.37293 1; -0.44582 0.39093 1; -0.11843 0.23395
1; -0.4614 0.05516 1; -0.14894 0.19884 0; -0.33598 0.20642 0; -0.15535
0.0459 0];
punts_r = puntsRobot(punts_r_4);
% Es dedueixen la resta de punts a partir dels donats

%% Inicialització de la webcam de l'smartphone
% Es necessari que el tauler no tinga cap peça.
% Es connecta a la webcam (en aquest cas, el mòbil).
cam = ipcam('http://158.42.69.139:8080/video', 'ChessCam', 'TFG123')
% Es fa una captura des de la webcam.
sensePeces = snapshot(cam);
[punts, ok] = cv.findChessboardCorners(sensePeces, [7,7]);
% Es crea una segona imatge on es dibuixen els punts detectats. Si em-
prant
% "imshow" no apareix cap punt en la imatge, voldrà dir que la detec-
ció no
% s'ha pogut realitzar (com en el cas de tindre peces en el tauler).
% sensePeces_1 = cv.drawChessboardCorners(sensePeces,[7,7],punts);

taulerPla = imread('tauler_pla_crop_bo.jpg');
[puntsCrop, ok] = cv.findChessboardCorners(taulerPla, [7,7]);
provaCrop_1 = cv.drawChessboardCorners(taulerPla,[7,7],puntsCrop);

% Punts homografia del tauler en perspectiva
puntsM = [];
for i=1:length(punts)
    puntsTemp = cell2mat(punts(i));
    puntsM = [puntsM; puntsTemp];
end
```

```

% Punts homografia del tauler pla
puntsC = [];
for i=1:length(puntsCrop)
    puntsTemp = cell2mat(puntsCrop(i));
    puntsC = [puntsC; puntsTemp];
end

% Es trau la homografia per a la posició actual del tauler.
H_1 = cv.findHomography(puntsM,puntsC);
tauler = cv.warpPerspective(sensePeces, H_1);
tauler = tauler(1:945,1:945);
punts_q = sortMat(puntsC); % Es trauen els punts d'anàlisi.

%% Iniciem la partida

CM = ChessMaster()
captures = 0; % Captures fetes per el robot.
captura = 0;
% S'ha de ficar l'Auto-Play. Sinó, no funcionarà. Comprobar el color
del
% jugador (es pot fer si ha havut moviment després de tants segons
% després).
% Una vegada iniciada, s'espera a que el jugador menege primer.

% Pausar fins que s'inicie l'engine
pause()
% Si fa jugada il·legal, el programa no procedirà (es detecta que la
FEN
% segueix sent igual).

pos_ini = CM.GetFENstr();
if sum(pos_ini(1:43)) == 3561
    % pos_ini(1:43) == 'rnbqbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR'
    % La persona juga amb blanques.
    torn = 1;
else
    % Espera sobredimensionada fins que la IA calcule el moviment.
    pause(3.5)
    % Obtenció de coordenades de moviment del robot. Es separarà en 2.
    ultim_mov = CM.GetLANstrs();
    ultim_mov = cell2mat(ultim_mov(numel(ultim_mov)));
    pos_ult = CM.GetSANstrs();
    pos_ult = cell2mat(pos_ult(numel(pos_ult)));
    % El robot executa el moviment de la IA.
    menejaRobot(captura,captures,ultim_mov,punts_r,t)
    % S'actualitza el tauler i li toca al jugador.
    torn = 0;
end

% Es fa la foto a la posició prèvia d'abans de que la faç el jugador.
% Igual es convenient ficar-ho en la part de l'if per a evitar redundància
im_prev = snapshot(cam);
im_prev = rgb2gray(im_prev);
tauler_prev = cv.warpPerspective(im_prev, H_1);
tauler_prev = tauler_prev(1:945,1:945);
c_Prev = casellesOcupades(punts_q,tauler_prev)
    % Matriu de caselles lliures i ocupades de la foto prèvia.

%% Resta de la partida

```

```

while CM.isGameOver==0
    % Es fa la partida fins que arriba al final

    if torn          % Li toca al jugador
        % Es fa la pausa fins que el jugador acabe el seu torn prement
        qualsevol tecla.
            pause();
            im_pos = snapshot(cam);
            im_pos = rgb2gray(im_pos);
            tauler_pos = cv.warpPerspective(im_pos, H_1);
            tauler_pos = tauler_pos(1:945,1:945);
            c_Pos = casellesOcupades(punts_q,tauler_pos);
            % Obtenció del moviment realitzat
            c_Dif = c_Pos - c_Prev;
            m_jugador = detectarMov(c_Dif,c_Pos)
            if sum(c_Pos==c_Dif)<64 || size(m_jugador)==4
                CM.MakeMove(m_jugador)           % Actualització del tauler
                im_prev = im_pos;
                tauler_prev = tauler_pos;
                c_Prev = c_Pos;
                torn = 0;
                disp("Torn del robot.");
            else
                torn = 1;
            end
            % Ací la IA ha d'actuar
        else          % Li toca al robot
            pause(3.5);
            % Temps d'espera per a obtindre la última jugada.
            ultim_mov = CM.GetLANstrs();
            ultim_mov = cell2mat(ultim_mov(numel(ultim_mov)));
            pos_ult = CM.GetSANstrs();
            pos_ult = cell2mat(pos_ult(numel(pos_ult)));
            if sum(pos_ult == 120)==0
                % No hi ha captura: ninguna 'x' detectada.
                captura = 0;
            else
                % S'ha detectat una 'x' en l'string.
                captura = 1;
                captures = captures + 1;
            end
            % El robot executa el moviment.
            menejaRobot(captura,captures,ultim_mov,punts_r,t)
            pause() % Pausa per a depurar.
            % Actualització de les imatges prèvies.
            im_prev = snapshot(cam);
            im_prev = rgb2gray(im_prev);
            tauler_prev = cv.warpPerspective(im_prev, H_1);
            tauler_prev = tauler_prev(1:945,1:945);
            c_Prev = casellesOcupades(punts_q,tauler_prev);
            % Matriu de caselles lliures i ocupades de la foto prèvia.
            torn = 1;
            disp("El teu torn.");
        end
    end

```

1.2. sortMat.m

```

function sQua = sortMat(M)
% sortMat -> ordena els punts del tauler i crea els necessaris
% L'input és una matriu de 7x7 i torna una matriu de 9x9 on els
% punts
% queden ordenats de a1 fins a8, de h1 a a2...
% a1 (cantó dret superior) en la posició 43
% Distància a l'esquerra: restar distància de 43 a 44
% Distància avall: sumar distància entre 43 i 36
% Distància dreta: restar 7 menys 6
% Distància amunt: dependrà Restar punt i menys i+7

dis_left = M(44,:)-M(43,:);
dis_down = M(43,:)-M(36,:);
dis_right = M(49,:)-M(48,:);
dis_up = M(8,:)-M(1,:);
a1 = [M(43,1)-0.95*dis_left(1)+0.95*dis_down(1) M(43,2)-
0.95*dis_left(2)+0.95*dis_down(2)];
sM = a1;
sQua = [];
for i=2:81
    % Linia horizontal inferior
    if i<9
        sM = [sM; M(41+i,1)+dis_down(1) M(41+i,2)+dis_down(2)];
    elseif i==9
        sM = [sM; M(49,1)+0.96*dis_right(1)+0.95*dis_down(1)
M(43,2)+0.96*dis_right(2)+0.95*dis_down(2)];

    % Linia vertical esquerra
    elseif mod(i-1,9)==0 && i<66 && i>9
        coef = (i-1)/9;
        coef = -7*coef+50;    % Recta y = mx + n
        dis_left = M(coef+1,:)-M(coef,:);
        sM = [sM; M(coef,1)-dis_left(1) M(coef,2)-dis_left(2)];

    % Linia vertical dreta
    elseif mod(i,9)==0 && i<73 && i>15
        coef = i/9 - 1;
        coef = -7*coef+56;    % Recta y = mx + n
        dis_right = M(coef,:)-M(coef-1,:);
        sM = [sM; M(coef,1)+dis_right(1) M(coef,2)+dis_right(2)];
    elseif i == 73    % dis_right ja està actualitzat.
        % No cal actualitzar dis_up perquè ve d'abans.
        sM = [sM; M(1,1)-0.95*dis_up(1)-0.95*dis_left(1) M(1,2)-
0.95*dis_up(2)-0.95*dis_left(2)];

    elseif i>73 && i<81
        sM = [sM; M(i-73,1)-dis_up(1) M(i-73,2)-dis_up(2)];
    elseif i == 81
        sM = [sM; M(7,1)-dis_up(1)+dis_right(1) M(7,2)-
dis_up(2)+dis_right(2)];
    elseif i>=11 && i<=17
        sM = [sM; M(32+i,1) M(32+i,2)];
    elseif i>=20 && i<=26
        sM = [sM; M(16+i,1) M(16+i,2)];
    elseif i>=29 && i<=35
        sM = [sM; M(i,1) M(i,2)];
    elseif i>=38 && i<=44
        sM = [sM; M(i-16,1) M(i-16,2)];
    elseif i>=47 && i<=53

```

```

    sM = [sM; M(i-32,1) M(i-32,2)];
elseif i>=56 && i<=62
    sM = [sM; M(i-48,1) M(i-48,2)];
elseif i>=65 && i<=71
    sM = [sM; M(i-64,1) M(i-64,2)];
else
end
end

for i=2:72
    if (mod(i-1,9)<0 || mod(i-1,9)>0)
        sQua = [sQua;
((sM(i+8,1)+sM(i,1))/2), (sM(i+8,2)+sM(i,2))/2];
    end
end
end

```

1.3. casellesOcupades.m

```
function cOcupades = casellesOcupades(M,img)
% Aquesta funció determinarà quines caselles estan ocupades i quines
estan
% lliures, agrupant-ho tot en un array de tamany 64.
% La matriu d'entrada son els punts que representen els cantons de
cada
% quadre i la imatge d'entrada serà la del tauler en l'estat actual.
% La funció analitzarà l'histograma de cada quadre del tauler.
% 0 -> quadre lliure
% 1 -> quadre ocupat per una peça blanca
% 2 -> quadre ocupat per una peça negra

cOcupades = [];
% Segona imatge del tauler aplicant un threshold.
tauler_b = cv.threshold(img, 25);
% S'aplica un filtre Canny a la imatge original.
img = edge(img,'Canny', 0.06);
% S'obre una finestra per a que els histogrames no interferisquen en
el
% tauler.
figure

for i=1:64
    quadre = img((M(i,2)-55):(M(i,2)+30),(M(i,1)-45):(M(i,1)-5));
    h_q = histogram(quadre);
    if (h_q.Values(1)==3526)      % No es detecta cap píxel blanc.
        cOcupades = [cOcupades 0];
    elseif (h_q.Values(2)<110)
        cOcupades = [cOcupades 0];  % Pixels blancs insuficients.
    else
        quadre = tauler_b((M(i,2)-45):(M(i,2)+40),(M(i,1)-
45):(M(i,1)-5));
        h_q = histogram(quadre,[0:5:255]);
        if h_q.Values(1)>205      % Peça negra
            cOcupades = [cOcupades 1];
        else
            cOcupades = [cOcupades 2];  % Peça blanca
        end
    end
end
end
```

1.4. detectarMov.m

```

function mov_jugador = detectarMov(M_dif,M_actual)
% Aquesta funció torna el valor, en una string, del moviment realitzat
per
% el jugador. Amb el paràmetre obtès, s'actualitzarà el tauler. Les
% variables d'entrada seran la matriu de variació i la actual.
% La matriu d'entrada serà la de variació de moviments (que apareix
en
% "joc.m") i, per a cada variació, asignarà un tipus d'string o al-
tre.

% En primer lloc, comprobarà si ha hagut algun enroc.
if M_dif(5:8)==[-2,2,2,-2]                                % Enroc curt blanques
    mov_jugador = 'e1g1';
elseif M_dif(1:5)==[-2,0,2,2,-2]                            % Enroc llarg blanques
    mov_jugador = 'e1c1';
elseif M_dif(61:64)==[-1,1,1,-1]                            % Enroc curt negres
    CM.MakeMove('e8g8')
elseif M_dif(57:61)==[-1,0,1,1,-1]                          % Enroc llarg negres
    CM.MakeMove('e8c8')
else
    for i=1:64      % Detectar posició anterior.
        if M_dif(i)~=0
            if M_actual(i)==0 % Casella on estava la posició an-
terior.
                switch mod(i+7,8)
                    case 0
                        mov_jugador = 'a';
                    case 1
                        mov_jugador = 'b';
                    case 2
                        mov_jugador = 'c';
                    case 3
                        mov_jugador = 'd';
                    case 4
                        mov_jugador = 'e';
                    case 5
                        mov_jugador = 'f';
                    case 6
                        mov_jugador = 'g';
                    case 7
                        mov_jugador = 'h';
                    otherwise
                end
                if i<9
                    mov_jugador = [mov_jugador '1'];
                elseif i<17
                    mov_jugador = [mov_jugador '2'];
                elseif i<25
                    mov_jugador = [mov_jugador '3'];
                elseif i<33
                    mov_jugador = [mov_jugador '4'];
                elseif i<41
                    mov_jugador = [mov_jugador '5'];
                elseif i<49
                    mov_jugador = [mov_jugador '6'];
                elseif i<57
                    mov_jugador = [mov_jugador '7'];
                else
                    mov_jugador = [mov_jugador '8'];
                end
            end
        end
    end
end

```

```

                end
            end
        end
    end

    for i=1:64      % Detectar posició actual
        if M_dif(i)~=0
            if M_actual(i)~=0 % Casella on està la posició actual
                switch mod(i+7,8)
                    case 0
                        mov_jugador = [mov_jugador 'a'];
                    case 1
                        mov_jugador = [mov_jugador 'b'];
                    case 2
                        mov_jugador = [mov_jugador 'c'];
                    case 3
                        mov_jugador = [mov_jugador 'd'];
                    case 4
                        mov_jugador = [mov_jugador 'e'];
                    case 5
                        mov_jugador = [mov_jugador 'f'];
                    case 6
                        mov_jugador = [mov_jugador 'g'];
                    case 7
                        mov_jugador = [mov_jugador 'h'];
                    otherwise
                end
                if i<9
                    mov_jugador = [mov_jugador '1'];
                elseif i<17
                    mov_jugador = [mov_jugador '2'];
                elseif i<25
                    mov_jugador = [mov_jugador '3'];
                elseif i<33
                    mov_jugador = [mov_jugador '4'];
                elseif i<41
                    mov_jugador = [mov_jugador '5'];
                elseif i<49
                    mov_jugador = [mov_jugador '6'];
                elseif i<57
                    mov_jugador = [mov_jugador '7'];
                else
                    mov_jugador = [mov_jugador '8'];
                end
            end
        end
    end
end

```

1.5. puntsRobot.m

```
function puntsR = puntsRobot(M)
%puntsRobot és la funció que, a partir de 3 punts del tualer, dedueix
les
%posicions de les altres caselles en X i Y.
% Donada una matriu on els tres primers punts en x i y siguen a1, h1
i a8
% respectivament, farà un càlcul de les posicions del robot que
aquest
% haurà de tindre per a situar-se en una cassella segons les coord-
nades
% x i y.
% Punts de a1, h1, a4, h8, a5, e5 i a8.

% Distància entre casella i casella en les files de ori_1
dif_M_h = (M(2,1:2)-M(1,1:2))/7;
% S'anula la component en y per minimitzar les impresicions.
dif_M_h(2) = 0;
% Distància entre casella i casella en les columnes de ori_1
dif_M_v = (M(3,1:2)-M(1,1:2))/3;
puntsR = [];
% c és una variable que determina la posició de la que es parteix per
a
% anar determinant les coordenades de cada fila. Quan es completa una,
se
% li suma la distància per casella de les columnes i ompli la fila.
c = M(1,1:2);
% Variable que contará la posició en la que es situa en cada fila.
d = 0;
% Bucle per a completar mig tauler (d'a1 fins a h4).
for i=1:32
    if mod(i-1,8)==0 && i>1
        c = M(1,1:2)+((i-1)/8)*dif_M_v;
        d = 0;
        puntsR = [puntsR; c+d*dif_M_h];
    elseif i == 1
        puntsR = M(1,1:2);
    else
        d = d+1;
        puntsR = [puntsR; c+d*dif_M_h];
    end
end
puntsR = [puntsR; M(5,1:2)];
% Distàncies entre casella i casella en les files de ori_0. Es
% procedeix de manera similar.
dif_M_h_1 = (M(6,1:2)-M(5,1:2))/4;
dif_M_h_1(2) = 0;
dif_M_v_1 = (M(7,1:2)-M(5,1:2))/3;
dif_M_v_1(1) = 0;
d_1 = 0;
c_1 = M(5,1:2);
% Bucle per a completar la resta del tauler.
for i=34:64
    if i==64
        puntsR = [puntsR; M(4,1:2)];
    elseif mod(i-1,8)==0
        c_1 = M(5,1:2)+((i-33)/8)*dif_M_v_1;
        d_1 = 0;
        puntsR = [puntsR; c_1+d_1*dif_M_h_1];
    elseif mod(i-1,8)>=1 && mod(i-1,8)<=4
```

```

d_1 = d_1+1;
puntsR = [puntsR; c_1+d_1*dif_M_h_1];
elseif mod(i-1,8)==5
    c = M(1,1:2)+((i-6)/8)*dif_M_v;
    d = 5;
    puntsR = [puntsR; c+d*dif_M_h];
elseif mod(i-1,8)>=6
    d = d+1;
    puntsR = [puntsR; c+d*dif_M_h];
else
end
end

```

1.6. menejaRobot.m

```

function [] = menejaRobot(capt,n_cap,last_mov,M,t)
%Aquesta funció farà que el robot façà el moviment de la IA mitjançant
les
%variacions detectades en el tauler.
% En funció de les condicions que detecte en les matrius de variació
i en
% la de la posició actual (moviment, captura o enroc), el braç es
% comportarà d'una manera o altra.
% En primer lloc, comprobarà si ha hagut algun enroc.
dis_m_J = 0.05;           % Distància en z en la que el robot es meneja
en x i y.
dis_m_pick = -0.122;      % Distància en z en la que s'agafa la peça.
dis_m_place = -0.116;     % Distància en z en la que es deixa la peça.
ori_1 = [0.866,3.4821,-0.2418];    % Orientació del braç en la zona
d'a1.
ori_0 = [3.0665,-0.6946,0.7061];    % Orientació del braç en la zona
d'a8.
canvi_ori = [];            % Ens dirà si caldrà fer canvi d'orientació.
pos_e4 = [M(29,1),M(29,2)];    % Per a canviar la orientació.
temps_pausa = 1;

% Equivalència string a número.
% a = 97; b = 98; c = 99; d = 100; e = 101; f = 102; g = 103; h = 104;
% 1 = 49; 2 = 50; 3 = 51; 4 = 52; 5 = 53; 6 = 54; 7 = 55; 8 = 56;

if last_mov == 'e1g1'          % Enroc curt blanques
    % Moviment de 'e1g1' i 'h1f1'.
    pos_1 = [M(5,1),M(5,2),ori_1];
    pos_2 = [M(7,1),M(7,2),ori_1];
    pos_3 = [M(8,1),M(8,2),ori_1];
    pos_4 = [M(6,1),M(6,2),ori_1];
    enroc = 1;
    canvi_ori = [1 1 1 1];
elseif last_mov == 'e1c1'        % Enroc llarg blanques
    % Moviment de 'e1c1' i 'a1d1'.
    pos_1 = [M(5,1),M(5,2),ori_1];
    pos_2 = [M(3,1),M(3,2),ori_1];
    pos_3 = [M(1,1),M(1,2),ori_1];
    pos_4 = [M(4,1),M(4,2),ori_1];
    enroc = 1;
    canvi_ori = [1 1 1 1];
elseif last_mov == 'e8g8'        % Enroc curt negres
    % Moviment de 'e8g8' i 'h8f8'.
    pos_1 = [M(61,1),M(61,2),ori_0];
    pos_2 = [M(63,1),M(63,2),ori_1];
    pos_3 = [M(64,1),M(64,2),ori_1];
    pos_4 = [M(62,1),M(62,2),ori_1];
    enroc = 1;
    canvi_ori = [0 1 1 1];
elseif last_mov == 'e8c8'        % Enroc llarg negres
    % Moviment de 'e8c8' i 'a8d8'.
    pos_1 = [M(61,1),M(61,2),ori_0];
    pos_2 = [M(59,1),M(59,2),ori_0];
    pos_3 = [M(57,1),M(57,2),ori_0];
    pos_4 = [M(60,1),M(60,2),ori_0];
    enroc = 1;
    canvi_ori = [0 0 0 0];
else                           % Comprobar si ha hagut captura.
    if capt==0                  % No hi ha captura

```

```

        enroc = 0;
    else % Ha hagut captura
        % Es determina la posició de captura de la peça en funció
de la
        % quantitat de captures produïdes.
        c = (last_mov(4)-49)*8+last_mov(3)-96;
        pos_1 = [M(c,1),M(c,2)];
        if mod(c,8)>0 && mod(c,8)<=5 && c>32
            pos_1 = [pos_1,ori_0];
            canvi_ori = [canvi_ori 0];
            enroc = 1;
        else
            pos_1 = [pos_1,ori_1];
            canvi_ori = [canvi_ori 1];
            enroc = 1;
        end
        if n_cap < 4
            pos_2 = [-0.45-0.04*n_cap,-0.03,ori_1];
        elseif n_cap < 7
            pos_2 = [-0.45+0.04*(n_cap-4),-0.09,ori_1];
        elseif n_cap < 10
            pos_2 = [-0.45+0.04*(n_cap-7),-0.15,ori_1];
        elseif n_cap < 13
            pos_2 = [-0.45+0.04*(n_cap-10),-0.21,ori_1];
        else
            pos_2 = [-0.45+0.04*(n_cap-10),-0.33,ori_1];
        end
        canvi_ori = [canvi_ori 1];
        c = (last_mov(2)-49)*8+last_mov(1)-96;
        pos_3 = [M(c,1),M(c,2)];
        if mod(c,8)>0 && mod(c,8)<=5 && c>32
            pos_3 = [pos_3,ori_0];
            canvi_ori = [canvi_ori 0];
            enroc = 1;
        else
            pos_3 = [pos_3,ori_1];
            canvi_ori = [canvi_ori 1];
            enroc = 1;
        end
        pos_4 = pos_1;
        canvi_ori = [canvi_ori canvi_ori(1)];
    end
end

if enroc || capt
    % S'empra el mateix codi tant per a la captura com per a l'en-
roc.
    % Move J per a agafar la peça en pos_1.
    m_str =
sprintf("[0,%f,%f,%f,%f,%f,0]",pos_1(1),pos_1(2),dis_m_J,pos_1(3),p
os_1(4),pos_1(5));
    fprintf(t,m_str);
    % Baixa on està la peça.
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,0]",pos_1(1),pos_1(2),dis_m_pick,pos_1(3
),pos_1(4),pos_1(5));
    fprintf(t,m_str);
    % Tanca la pinça.
    pause(temp_pausa)

```

```

    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_1(1),pos_1(2),dis_m_pick,pos_1(3)
),pos_1(4),pos_1(5));
    fprintf(t,m_str);
    % Puja a la zona de Move J.
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_1(1),pos_1(2),dis_m_J,pos_1(3),p
os_1(4),pos_1(5));
    fprintf(t,m_str);

    if canvi_ori(1)~=canvi_ori(2)
        % Si en la determinació de totes les posicions es detecta
un
        % canvi d'orientació entre la posició 1 a la 2. Anirà a la
        % posició de la casella e4 per a canviar d'orientació.
        if canvi_ori(1)==1      % Pasar de la orientació 1 a 0
            m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_1(1)
,ori_1(2),ori_1(3));
            fprintf(t,m_str);
            m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_0(1)
,ori_0(2),ori_0(3));
            fprintf(t,m_str);
        else                      % Pasar de la orientació 0 a 1
            m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_0(1)
,ori_0(2),ori_0(3));
            fprintf(t,m_str);
            m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_1(1)
,ori_1(2),ori_1(3));
            fprintf(t,m_str);
        end
    end

    % Move J per a agafar la peça en pos_2
    m_str =
sprintf("[0,%f,%f,%f,%f,%f,%f,1]",pos_2(1),pos_2(2),dis_m_J,pos_2(3),p
os_2(4),pos_2(5));
    fprintf(t,m_str);
    % Baixa on està la peça.
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_2(1),pos_2(2),dis_m_place,pos_2(
3),pos_2(4),pos_2(5));
    fprintf(t,m_str);
    % Obri la pinça.
    pause(temps_pausa)
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,0]",pos_2(1),pos_2(2),dis_m_place,pos_2(
3),pos_2(4),pos_2(5));
    fprintf(t,m_str);
    % Puja a la zona de Move J.
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,0]",pos_2(1),pos_2(2),dis_m_J,pos_2(3),p
os_2(4),pos_2(5));
    fprintf(t,m_str);

    if canvi_ori(2)~=canvi_ori(3)
        if canvi_ori(2)==1      % Pasar de la orientació 1 a 0

```

```

    m_str =
sprintf("[1,%f,%f,%f,%f,%f,0]",pos_e4(1),pos_e4(2),dis_m_J,ori_1(1)
,ori_1(2),ori_1(3));
    fprintf(t,m_str);
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,0]",pos_e4(1),pos_e4(2),dis_m_J,ori_0(1)
,ori_0(2),ori_0(3));
    fprintf(t,m_str);
else % Pasar de la orientació 0 a 1
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,0]",pos_e4(1),pos_e4(2),dis_m_J,ori_0(1)
,ori_0(2),ori_0(3));
    fprintf(t,m_str);
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,0]",pos_e4(1),pos_e4(2),dis_m_J,ori_1(1)
,ori_1(2),ori_1(3));
    fprintf(t,m_str);
end

% Move J per a agafar la peça en pos_3.
m_str =
sprintf("[0,%f,%f,%f,%f,%f,%f,0]",pos_3(1),pos_3(2),dis_m_J,pos_3(3),p
os_3(4),pos_3(5));
    fprintf(t,m_str);
    % Baixa on està la peça.
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,0]",pos_3(1),pos_3(2),dis_m_pick,pos_3(3
),pos_3(4),pos_3(5));
    fprintf(t,m_str);
    % Tanca la pinça.
    pause(tempo_pausa)
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_3(1),pos_3(2),dis_m_pick,pos_3(3
),pos_3(4),pos_3(5));
    fprintf(t,m_str);
    % Puja a la zona de Move J.
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_3(1),pos_3(2),dis_m_J,pos_3(3),p
os_3(4),pos_3(5));
    fprintf(t,m_str);

if canvi_ori(3) ~= canvi_ori(4)
    if canvi_ori(3) == 1 % Pasar de la orientació 1 a 0
        m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_1(1)
,ori_1(2),ori_1(3));
        fprintf(t,m_str);
        m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_0(1)
,ori_0(2),ori_0(3));
        fprintf(t,m_str);
    else % Pasar de la orientació 0 a 1
        m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_0(1)
,ori_0(2),ori_0(3));
        fprintf(t,m_str);
        m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_1(1)
,ori_1(2),ori_1(3));
        fprintf(t,m_str);

```

```

        end
    end

    % Move J per a agafar la peça en pos_4.
    m_str =
sprintf("[0,%f,%f,%f,%f,%f,%f,1]",pos_4(1),pos_4(2),dis_m_J,pos_4(3),p
os_4(4),pos_4(5));
    fprintf(t,m_str);
    % Baixa on està la peça.
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_4(1),pos_4(2),dis_m_place,pos_4(
3),pos_4(4),pos_4(5));
    fprintf(t,m_str);
    % Obri la pinça.
    pause(temp_pausa)
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,0]",pos_4(1),pos_4(2),dis_m_place,pos_4(
3),pos_4(4),pos_4(5));
    fprintf(t,m_str);
    % Puja a la zona de Move J.
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,0]",pos_4(1),pos_4(2),dis_m_J,pos_4(3),p
os_4(4),pos_4(5));
    fprintf(t,m_str);

else      % Moviment sense captura
    c = (last_mov(2)-49)*8+last_mov(1)-96;
    pos_1 = [M(c,1),M(c,2)];
    if mod(c,8)>0 && mod(c,8)<=5 && c>32
        pos_1 = [pos_1,ori_0];
        canvi_ori = [canvi_ori 0];
    else
        pos_1 = [pos_1,ori_1];
        canvi_ori = [canvi_ori 1];
    end
    c = (last_mov(4)-49)*8+last_mov(3)-96;
    pos_2 = [M(c,1),M(c,2)];
    if mod(c,8)>0 && mod(c,8)<=5 && c>32
        pos_2 = [pos_2,ori_0];
        canvi_ori = [canvi_ori 0];
    else
        pos_2 = [pos_2,ori_1];
        canvi_ori = [canvi_ori 1];
    end

    % Move J per a agafar la peça en pos_1.
    m_str =
sprintf("[0,%f,%f,%f,%f,%f,%f,0]",pos_1(1),pos_1(2),dis_m_J,pos_1(3),p
os_1(4),pos_1(5));
    fprintf(t,m_str);
    % Baixa on està la peça.
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,0]",pos_1(1),pos_1(2),dis_m_pick,pos_1(3),
),pos_1(4),pos_1(5));
    fprintf(t,m_str);
    % Tanca la pinça.
    pause(temp_pausa)
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_1(1),pos_1(2),dis_m_pick,pos_1(3),
),pos_1(4),pos_1(5));
    fprintf(t,m_str);      % Puja a la zona de Move J.

```

```

    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_1(1),pos_1(2),dis_m_J,pos_1(3),p
os_1(4),pos_1(5));
    fprintf(t,m_str);

    if canvi_ori(1)~=canvi_ori(2)
        if canvi_ori(1)==1           % Pasar de la orientació 1 a 0
            m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_1(1)
,ori_1(2),ori_1(3));
            fprintf(t,m_str);
            m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_0(1)
,ori_0(2),ori_0(3));
            fprintf(t,m_str);
        else                         % Pasar de la orientació 0 a 1
            m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_0(1)
,ori_0(2),ori_0(3));
            fprintf(t,m_str);
            m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_e4(1),pos_e4(2),dis_m_J,ori_1(1)
,ori_1(2),ori_1(3));
            fprintf(t,m_str);
        end
    end

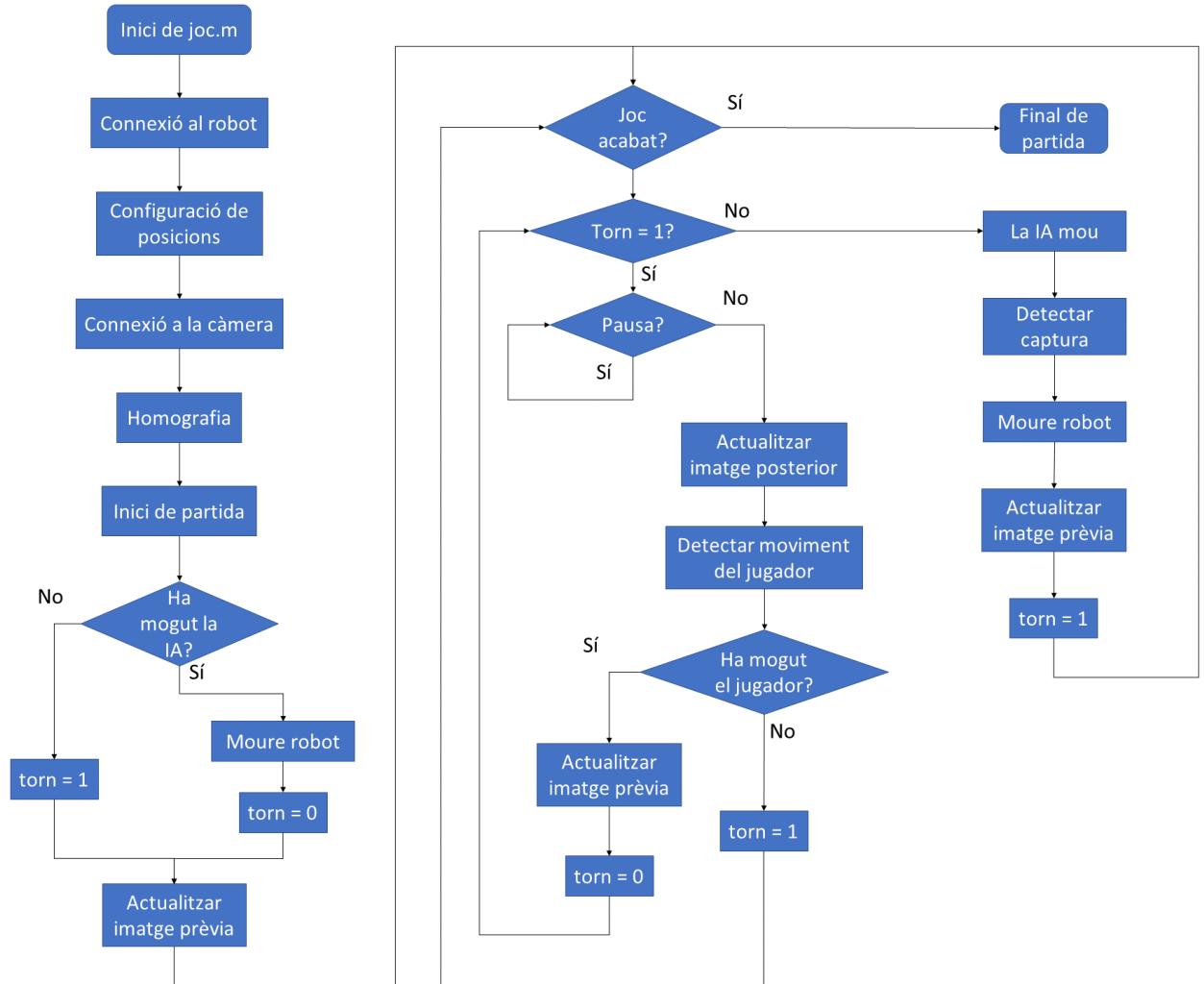
    % Move J per a agafar la peça en pos_2.
    m_str =
sprintf("[0,%f,%f,%f,%f,%f,%f,1]",pos_2(1),pos_2(2),dis_m_J,pos_2(3),p
os_2(4),pos_2(5));
    fprintf(t,m_str);
    % Baixa on està la peça.
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,1]",pos_2(1),pos_2(2),dis_m_place,pos_2(
3),pos_2(4),pos_2(5));
    fprintf(t,m_str);
    % Obri la pinça.
    pause(temps_pausa)
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,0]",pos_2(1),pos_2(2),dis_m_place,pos_2(
3),pos_2(4),pos_2(5));
    fprintf(t,m_str);      % Puja a la zona de Move J.
    m_str =
sprintf("[1,%f,%f,%f,%f,%f,%f,0]",pos_2(1),pos_2(2),dis_m_J,pos_2(3),p
os_2(4),pos_2(5));
    fprintf(t,m_str);
end

if canvi_ori(numel(canvi_ori))==0
    % Posició de repòs amb ori_1.
    m_str = sprintf("[0,-0.216,-
0.111,0.025,%f,%f,%f,0]",ori_0(1),ori_0(2),ori_0(3));
    fprintf(t,m_str);
else
    % Posició de repòs amb ori_0.
    m_str = sprintf("[0,-0.35,-
0.095,0.05,%f,%f,%f,0]",ori_1(1),ori_1(2),ori_1(3));
    fprintf(t,m_str);
end
end

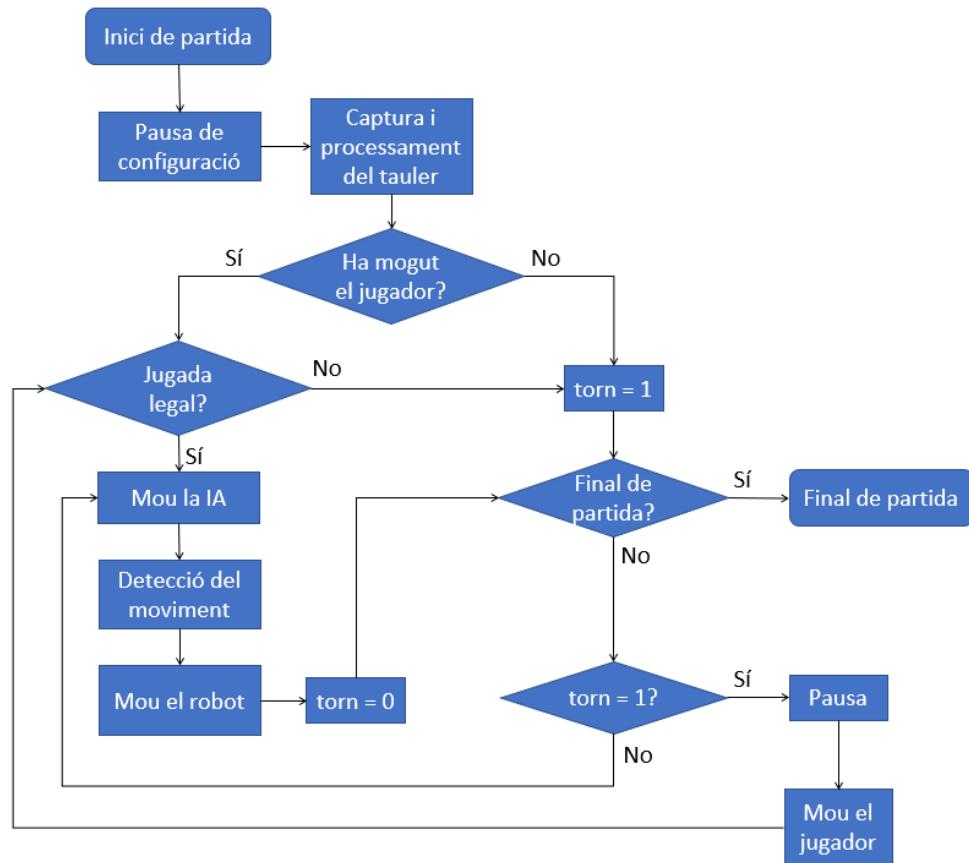
```

2. Diagrames de flux

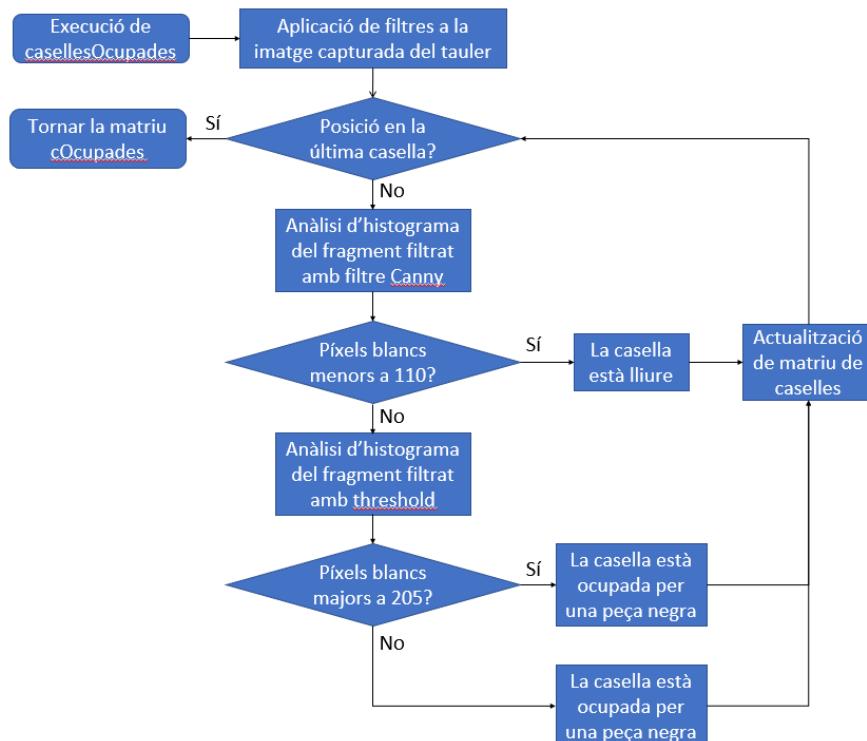
2.1. Diagrama de joc.m



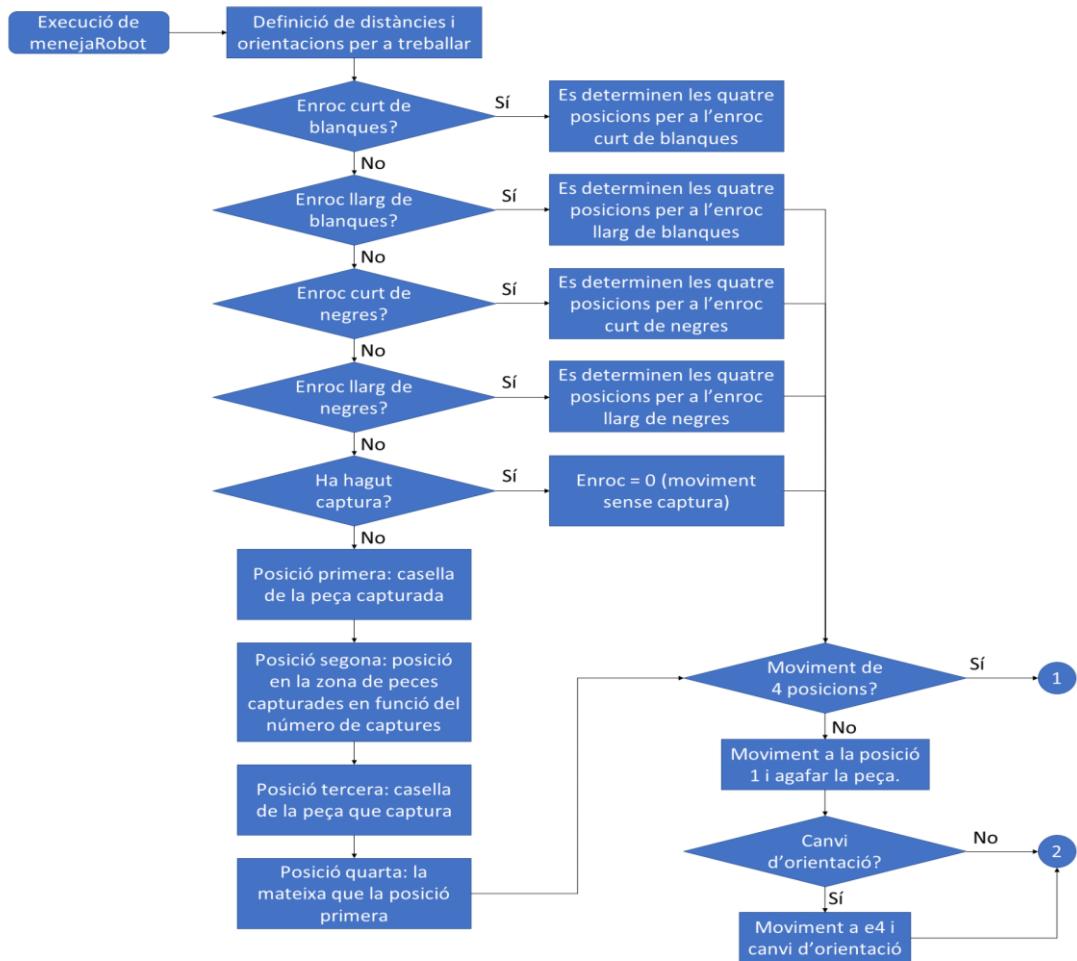
2.2. Diagrama de funcionament d'una partida

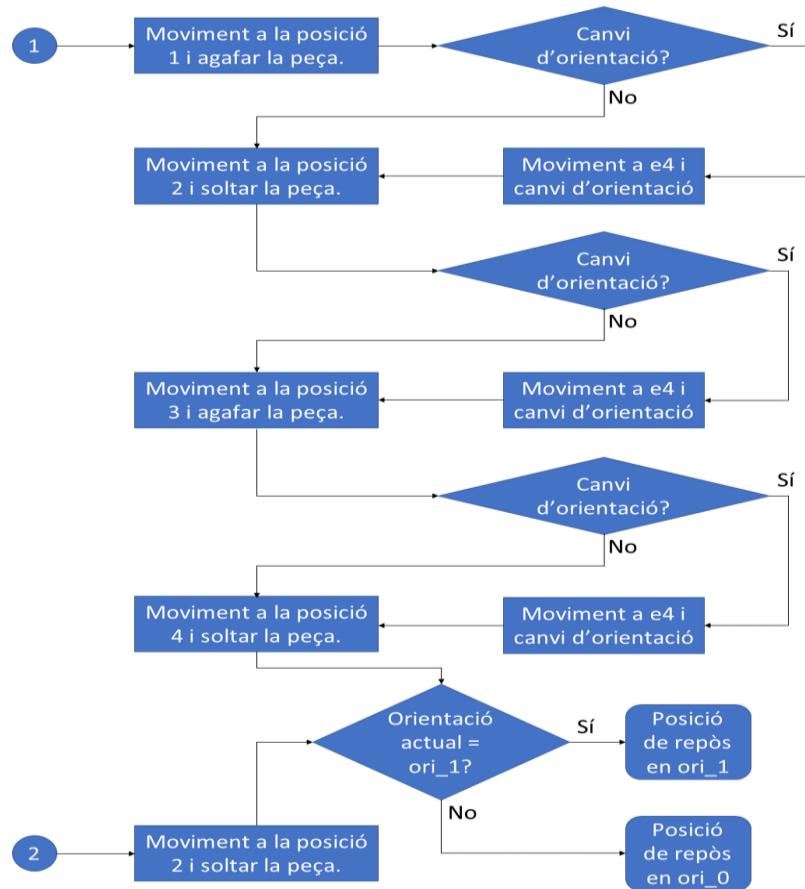


2.3. Diagrama de la funció casellesOcupades.m



2.4. Diagrama de la funció menejaRobot.m





3. Datasheets

3.1. Robot UR3

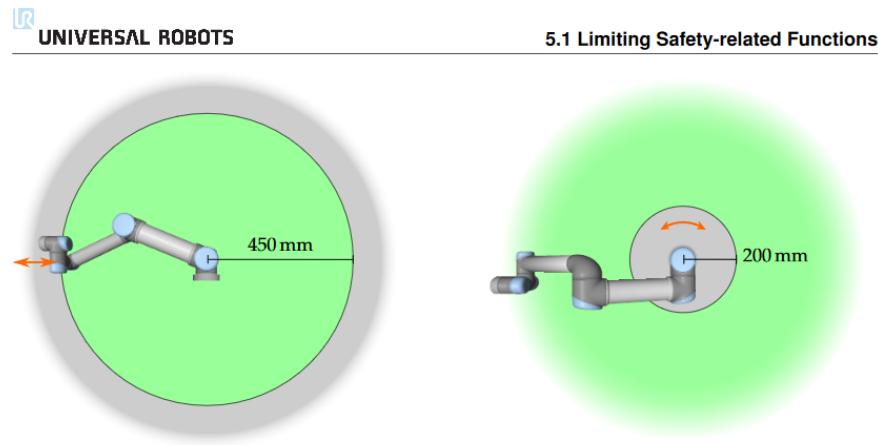


Figure 5.1: Certain areas of the workspace should receive attention regarding pinching hazards, due to the physical properties of the robot arm. One area is defined for radial motions, when the wrist 1 joint is at a distance of at least 450 mm from the base of the robot. The other area is within 200 mm of the base of the robot, when moving in the tangential direction.

Limiting Safety Function	Worst Case			
	Trueness	Detection Time	De-energizing Time	Reaction Time
Joint position	1.15 °	100 ms	1000 ms	1100 ms
Joint speed	1.15 °/s	250 ms	1000 ms	1250 ms
TCP position	20 mm	100 ms	1000 ms	1100 ms
TCP orientation	1.15 °	100 ms	1000 ms	1100 ms
TCP speed	50 mm/s	250 ms	1000 ms	1250 ms
TCP force	25 N	250 ms	1000 ms	1250 ms
Momentum	3 kg m/s	250 ms	1000 ms	1250 ms
Power	10 W	250 ms	1000 ms	1250 ms

The system is considered *de-energized* when the 48 V bus voltage reaches an electrical potential below 7.3 V. The de-energizing time is the time from a detection of an event until the system has been de-energized.



WARNING:

There are two exceptions to the force limiting function that are important to notice when designing the work cell for the robot. These are illustrated in Figure 5.1. As the robot stretches out, the knee-joint effect can give high forces in the radial direction (away from the base), but at the same time, low speeds. Similarly, the short leverage arm, when the tool is close to the base and moving tangential (around) the base, can cause high forces, but also at low speeds. Pinching hazards can be avoided for instance by, removing obstacles in these areas, placing the robot differently, or by using a combination of safety planes and joint limits to remove the hazard by preventing the robot moving into this region of its workspace.

D Technical Specifications

Robot type	UR3
Weight	9.4 kg / 20.7 lb
Payload	3 kg / 6.6 lb
Reach	500 mm / 19.7 in
Joint ranges	Unlimited for Wrist 3, ± 360° for all other joints
Speed	Base, Shoulder and Elbow joints: Max 180 °/s. Wrist 1, 2, 3 joints: Max 360 °/s. Tool: Approx. 1 in/s / Approx. 39.4 in/s.
Repeatability	± 0.1 mm / ± 0.0039 in (4 mils)
Footprint	Ø128 mm / 5.0 in
Degrees of freedom	6 rotating joints
Control box size (W × H × D)	475 mm × 423 mm × 268 mm / 18.7 in × 16.7 in × 10.6 in
Control box I/O ports	16 digital in, 16 digital out, 2 analogue in, 2 analogue out
Tool I/O ports	2 digital in, 2 digital out, 2 analogue in
I/O power supply	24 V 2 A in control box and 12 V/24 V 600 mA in tool
Communication	TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP
Programming	PolyScope graphical user interface on 12" touchscreen with mounting
Noise	Comparatively noiseless
IP classification	IP54
Power consumption	Approx. 100 W using a typical program
Collaboration operation	Collaborative operation according to ISO 10218-1:2011
Temperature	The robot can work in a temperature range of 0-50 °C
Power supply	100-240 VAC, 50-60 Hz
Calculated operating life	35,000 hours
Cabling	Cable between robot and control box (6 m / 236 in) Cable between touchscreen and control box (4.5 m / 177 in)

Copyright © 2009-2015 by Universal Robots A/S. All rights reserved.

3.2. Càmera

CÁMARA

Trasera

Sony IMX298, 16 MP

Apertura de $f/2.0$, 1.12 $\mu\text{m}/\text{pixel}$

6 lentes Largan

Dual Tone Flash

Auto-enfoque detección de fase (PDAF)

Resolución de vídeo 4K@30fps

Estabilización de vídeo (Vidhance)

Slow Motion (720p@120fps)

Fast Motion y Time Lapse

HDR automático

Disparo en formato RAW

Control manual de parámetros (tiempo de exposición, enfoque e ISO)

Modos nocturno, panorámico y ráfaga