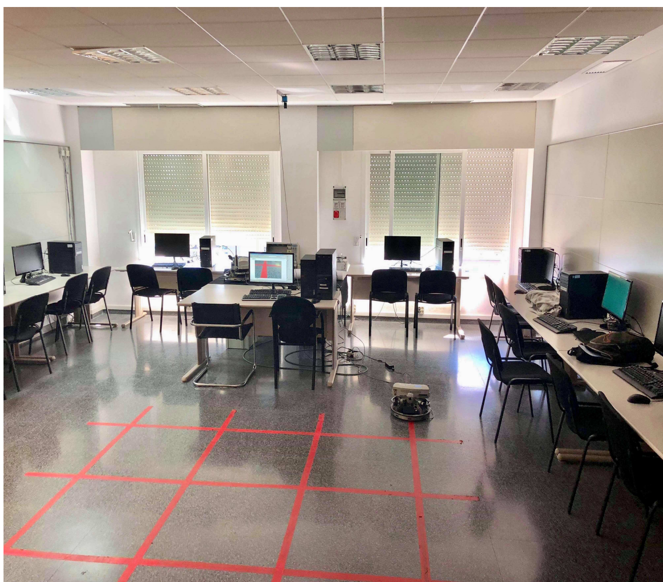
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa



ESTUDI I IMPLEMENTACIÓ MITJANÇANT QNX D'UN ALGORISME DE NAVEGACIÓ SOBRE UNA GRAELLA PER UN ROBOT MÒBIL PROVEÏT D'UNA CÀMERA.

ENGINYERIA INDÚSTRIAL ELECTRÒNICA I AUTOMÀTICA

DIRECTOR: Ramón Sarrate Estruch

ESTUDIANT: Esiaka Minteh Jaiteh

DATA: Octubre 2018

AGRAÏMENTS

Aprofito aquest apartat per expressar el meu agraïment a una sèrie de persones que han fet que aquest petit projecte fos possible.

En primer lloc m'agradaria donar les gràcies al meu tutor de treball de fi de grau, Ramón Sarrate, per la seva exigència, el constant suport, l'orientació i la confiança d'acompanyar-me durant la trajectòria del projecte. Gràcies a ell aquest treball ha anat agafant forma i consistència dia rere dia.

En segon lloc, a tots aquells professors que m'han ajudat encara que sigui amb suggeriments o per un dubte simple, ja que per mi sol no hauria estat possible resoldre'l.

També m'agradaria agrair a un antic professor de secundària, Antonio Díaz, amant de la poesia i de les ciències, per alguns dels consells que hem va donar quan encara era jove. A dia d'avui encara me'n recordo d'alguns d'ells, o dels poemes de Béquer o Pedro Calderón de la Barca que hem va fer copiar una i altra vegada com a càstig per mal comportament. Un que hem va agradar molt va ser la part final del 'Soniloquio de Segismundo', que normalment me'n recordo i la recito quan m'estresso:

` ... Yo sueño que estoy aquí
destas prisiones cargado,
y soñé que en otro estado
más lisonjero me vi.
¿Qué es la vida? Un frenesí.
¿Qué es la vida? Una ilusión,
una sombra, una ficción,
y el mayor bien es pequeño:
que toda la vida es sueño,
y los sueños, sueños son.'
- Pedro Calderón de la Barca
(1600-1681)

També m'agradaria agrair a tots aquells amics i companys per tots els ànims i suport que m'han donat, antics companys per tota la ajuda i complicitat que hem tingut any rere any fins a acabar la carrera; 'Ha sido una batalla dura, pero lo conseguimos!'.

I, per últim i no menys important, a la família; Pares, tiets i molt especialment als meus germans per milers i milers de raons que no podria citar en aquest document.

ÍNDEX

1 INTRODUCCIÓ........ 1
1.1 ANTECEDENTS........2
1.1.1 FONTS PRIMARIES........2
1.1.2 FONTS SECUNDARIES........3
1.1.3 OBJECTIUS........5

2 ROBOTINO........6
2.1 ¿QUÉ ÉS?........6
2.2 COMPONENTS........7
2.2.1 XÀSSIS / PONT DE COMANDAMENT........7
2.2.2 MODULS D'ACCIONAMENT........8
2.2.3 BATERIES / CARREGADOR........9
2.2.4 UNITAT DE CONTROL........9
2.2.4.1 Placa de control d'entrades i sortides *(EA09)*........10
2.2.5 PUNT D'ACCES LAN SENSE FILS........10
2.2.6 SENSORS........12
2.2.6.1 Sensors d'infratoijos........12
2.2.6.2 Para-xocs (Bumper)........12
2.2.6.3 Ampliacions........12
2.2.7 CÀMERA........13

3 SISTEMES OPERATIUS DE TEMPS REAL (SOTR)........14
3.1 DEFINICIÓ........14
3.2 CARACTERÍSTIQUES GENERALS DELS SOTR........15
3.3 EL SOTR QNX........16
3.3.1 HISTORIA........16
3.3.2 PRINCIPIS BASICS........17
3.3.3 CARACTERÍSTIQUES........18
3.3.4 VARIANTS DE QNX........18
3.4 EL NOU IDE 19

4 **Càmera AXIS 207W........21**
4.1 CARACTERISTIQUES........21
4.2 VISIÓ GENERAL........22
4.3 INSTAL·LACIÓ I CONFIGURACIÓ INICIAL........23
4.3.1 ASSIGNACIÓ IP........23
4.4 INTERFICIE WEB........25
4.4.1 ACCES A LA CAMERA........25
4.4.2 LIVE VIEW........26
4.4.3 SETUP........26
4.4.4 CONNEXIÓ SENSE FILS........28
4.5 VAPIX........30

5 **API'S...32**
5.1 API DEL ROBOTINO PER QNX...32
5.2 LLIBRERIA PER A LA CAMERA: APICAMERA...35
5.2.1 GETJPG()...35
5.2.2 GETBMP()...35
5.2.3 ALTRES FUNCIONS...35
5.3 LLIBRERIA PER PROCESSAT D'IMATGES: APIIMATGE...36
5.3.1 FUNCIONS DE PROJECTISTES ANTERIORS...37
5.3.1.1 Binaritzar imatge *(im2bw)*...37
5.3.1.2 Conversió de rgb a ycbcr *(rgb2ycbcr)*...38
5.3.1.3 Separar capes ycbcr *(capaY, capaCb, capaCr)*...39
5.3.1.4 Canviar contrast *(contrast)*...40
5.3.1.5 Separar capes *(capaR, capaG, capaB)*...40
5.3.1.6 Erosionar imatge *(erode)*...41
5.3.1.7 Dilatar imatge *(dilate)*...42
5.3.1.8 Detecció de la linea d'una imatge bineritzada *(detectSegment..)*...42
5.3.2 FUNCIONS PRÒPIES...43
5.3.2.1 Detecció de línia a partir de dos punts *(detectPuntos)*...43
5.3.2.2 Transformació de coordenades píxel a robot *(PimgToRobot)*...45

6 **CALIBRATGE DE LA CÀMERA...46**
6.1 QUE ES LA PERSPECTIVA...46
6.2 QUE ES EL CALIBRATGE...47
6.2.1 PARÀMETRES INTRÍNSECS DE LA CÀMERA...49
6.2.2 PARÀMETRES EXTRÍNSECS DE LA CÀMERA...50
6.3 . PASSOS PER FER EL CALIBRATGE...52
6.3.1 COORDENADES ROBOT A COORDENADES DE CALIBRATGE...52
6.3.2 COORDENADES ROBOT A COORDENADES DE LA CÀMERA...53
6.3.3 COORDENADES ROBOT A COORDENADES DE LA IMATGE...54
6.3.4 COORDENADES IMATGE A COORDENADES ROBOT...55

7 **ALGORISMES DE PATHGFINDING...56**
7.1 DADES NECESSÀRIES...56
7.2 DIFERENTS ALGORISMES DE PATHFINDIG...57
7.2.1 GREEDY...59
7.2.2 DIJKSTRA...61
7.2.3 BFS...63
7.2.4 DFS...64
7.2.5 A* (A-STAR)...67

8 **APLICACIONS DESENVOLUPADES.....68**
8.1 RESUM DEL FUNIONAMENT.....69
8.2 CALIBRATGE.....71
8.3 VALORS A EXPORTAR A QNX.....79
8.3.1 DECLARACIÓ DE LES 4 MATRIUSDE ROTACIÓ I TRANSLACIÓ EN COORDENADES HOMOGÈNIES.....79
8.3.2 CÀLCUL DE LA MATRIU M.....79
8.3.3 CALCUL DE LANDA(λ).....80
8.4 CONNEXIO AMB LA CAMERA I OBTENCIÓ DE LA IMATGE.....81
8.4.1 CONDICIONS LUMÍNIQUES.....83
8.5 TRACTAMENT DE LA IMATGE.....84
8.6 SEGUIMENT DE LINIA.....85
8.6.1 OBTENCIO DE PUNTS.....85
8.6.2 PUNTS IMATGE (PÍXEL) A PUNTS COORDENADES ROBOT.....86
8.6.3 CALCUL DE L'ANGLE I DISTANCIES.....88
8.6.4 PURE-PURSUIT I COMPOSICIÓ DE LES VELOCITATS.....90
8.7 Llibreria de PATHFINDING.....92
8.7.1 EMPLENAMENT DE LA MATRIU DE PESOS.....93
8.7.2 GREEDY.....96
8.7.3 DIJKSTRA.....98
8.8 **PROGRAMA PRINCIPAL.....101**
8.8.1 TASCA SUPERVISOR.....102
8.8.2 TASCA CONTROLLER.....105
8.8.3 TASCA MONITOR.....*108*
8.8.4 GUI_*Input, Show*GUI() *I ShowADVANCED()*.....111
8.8.5 TASCA PRINCIPAL (*Main*).....113

9 **CONCLUSIONS I PROPOSTES DE MILLORA.....116**
10 **BIBLIOGRAFIA.....118**
11 **ANNEXOS.....120**

- A: GUIA DE INSTAL·LACIÓ IDE 5.0
- B: CREACIÓ DEL TARGET ROBOTINO I DEL PROJECTE a l'IDE 5.0
- C: TRANSFERENCIA I EXECUCIÓ DE CODI EN EL ROBOTINO
- D: CODI DE MATLAB PER OBTENIR ELS PARAMETRES NECESSARIS.
- E: GUIA PER AL CALIBRATGE DE LA CÀMERA.
- F: OLLERO
- G: CODI DE LA LLIBRERIA PATHFINDING
- H: CODI DEL PROGRAMA PRINCIPAL
- I: FUNCIÓ DE PATHFINDING MODIFICADA PER MATLAB
- J: ACTIVACIÓ DEL TELNET

1. INTRODUCCIÓ

Anys enrere, quan es parlava de robòtica tothom pensava en essers semblants als humans però sense anima ni sentiments que podrien acabar amenaçant la existència i seguretat de les persones. Però, a l'actualitat, sabem que no es així sinó tot el contrari. El món de la robòtica poc a poc s'ha anat fent un lloc a la industria i ara, cada cop més, son més habituals els seus productes a la nostra vida quotidiana: des del robot Roomba que ens neteja la casa fins a l'automòbil Tesla que condueix autònomament, sempre tenint en compte com a prioritat principal la seguretat de l'usuari.

El repte d'avui dia no només es aconseguir que els robots facin les tasques que se li encomanin sinó aconseguir que ho facin amb un grau d'autonomia i seguretat cada vegada més alt.

En el cas de la industria, tradicionalment, les aplicacions de la robòtica estaven centrades en els sectors manufacturers més desenvolupats per a la producció massiva: indústria de l'automòbil, del metall, industria química, etc. En les ultimes dècades el pes de la industria manufacturera ha disminuït, i això ha fet que la robòtica s'hagués d'adaptar a altres àmbits. Ara, a la industria, es comú veure robots que realitzen soldadures, pinten o mouen grans pesos, col·laboren en laboratoris farmacèutics o quiròfans, o en activitats varies que normalment es feien amb esforç humà.

Per fer tot això possible ha estat necessari un continu avanç tant en la mecànica com en la programació d'aquests robots. Nous algorismes, nous llenguatges de programació, implementació de nous hardwares com càmeres, diferents tipus de sensors, etc. tot això enfocant cap a una robòtica més autònoma, eficaç i segura. Més autonomia implica una millora, però també implica haver d'aplicar més sistemes de seguretat i més complexitat en la estructura.

Per a que un robot mòbil pugui realitzar tasques de forma independent, es essencial que sigui capaç de moure's per si sòl i, si amés és capaç d'analitzar i de planejar tot sol els seus moviments, encara millor. D'aquí surt el propòsit principal d'aquest projecte: Aconseguir que el robot mòbil Robotino sigui capaç de moure's autònomament per visió utilitzant algorismes de pathfinding sobre una estructura ja creada.

1.1 ANTECEDENTS

Bàsicament, la informació útil que s'ha utilitzat per aquest projecte ha provingut de dues fonts. La primera font son els coneixements propis obtinguts al llarg de la meva especialització durant el grau i la segona font es provinent de publicacions d'altres autors (incloent projectistes anteriors i professorat).

1.1.1 FONTS PRIMÀRIES

Els coneixements previs que tenia per elaborar aquest treball, provenen d'assignatures que he anat cursant en aquest darrers anys.
Aquestes assignatures que m'han ajudat són :

- Fonaments d'Informàtica.
- Informàtica Industrial.
- Control i Guiatge de Robots Mòbils (CGRM).
- Programació de Sistemes de Control en Temps Real (PSCTR).
- Planning, Simulation and Supervision of Processes (PSSP).

Aquestes assignatures han sigut fonamentals per entendre les diferents eines i estructures de programació necessàries per dur a terme aquest projecte. Amés m'han proporcionat la formació necessària per entendre també la comunicació i interacció amb dispositius remots, per treballar amb diferents programes utilitzats en el projecte com per exemple Matlab i QNX, i per entendre el funcionament dels algorismes de pathfinding. Per això, la considero la font primària d'informació.

Lògicament, aquestes no han sigut les úniques assignatures que m'han aportat alguna eina o coneixement útil pel desenvolupament del projecte, però son les que més pes han tingut.

1.1.2 FONTS SECUNDÀRIES

El meu projecte de final de grau te com a base els projectes anteriors de tres ex-alumnes de l'antiga EET.

- El primer: ESTUDI D'ADAPTACIÓ A QNX DE LA PLATAFORMA DIDÀCTICA DE ROBÒTICA MÒBIL ROBOTINO[1] (2011).
- El segon: ESTUDI D'INTEGRACIÓ D'UNA CÀMERA DE XARXA A UN ROBOT MÒBIL (2013)[2]
- I el tercer: ESTUDI D'ESTRATÈGIES DE SEGUIMENT DE LÍNEA PER AL ROBOT MÒBIL ROBOTINO (2016)[3]. Tots tres es poden trobar a la web upcommons de la escola.

El primer projecte tenia per objectiu crear una interfície que fos capaç de permetre la comunicació entre el sistema operatiu QNX i el Robotino. Per poder realitzar aquesta interfície, va dissenyar la API Robotino implementada per dos fitxers: robotinoapi1 i robotinoapi2. Aquesta API es va millorar pel seu ús a l'assignatura PSCTR[7].

El segon projecte tracta d'integrar la càmera web el robot Robotino de forma que pugui capturar imatges mitjançant peticions a la pròpia càmera. Per a fer-ho possible va haver de crear unes API's per treballar amb QNX: *apicamera* per poder fer les peticions d'imatge (en format JPG o BMP) i *apiimatge* per al possible posterior tractament d'imatge amb diferents tipus de filtres o mascares. Amés, encara que la comunicació amb la càmera es per WI-FI, va integrar la càmera al xassís del robot. Per fer-ho va haver de dissenyar i crear un convertidor de voltatge DC-DC de 24v a 5v que es connecta directament al port d'E/S del Robotino.

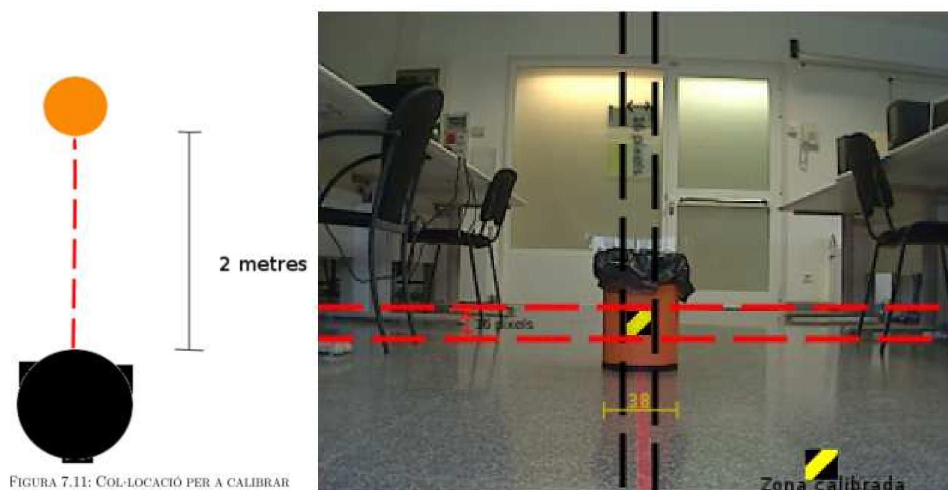


Figura 1.1: Imatges del programa d'exemple del 2n projectista.

Finalment va crear un petit programa d'exemple per mostrar com funcionaven les llibreries desenvolupades. La seqüencia consistia en primer fer un calibratge (Figura 1.1) de la càmera, que es trobava de forma perpendicular al terra, després fer una petició d'imatge a la càmera i un posterior tractament d'imatge amb el filtre que es volia provar.

El tercer projecte va partir d'aquí i va intentar crear un algorisme tal que permetés al robot avançar seguint una línia recta pintada al terra. El problema principal que va tenir era que el temps d'adquisició de la imatge era excessiu i això comprometia la velocitat d'avanç del Robotino fins al punt de que havia d'avançar molt lentament. Llavors, va tenir la idea de canviar la càmera per una altre de la mateixa marca però que respongués més ràpidament.

Figura 1.2: càmera antiga/nova

A la figura 1.2. es poden apreciar els dos models de càmeres. El model de la primera càmera és la AXIS M1011-W (esquerra) i el de la segona és AXIS 207W (dreta).

També va experimentar problemes amb les API's i amb el calibratge de la càmera, fet que va solucionar modificant les API's i posant la càmera de forma paral·lela al terra per tal d'evitar el calibratge.

Apart d'aquestes dos projectes, hi han hagut més fonts provinents d'altres autors, com poden ser:

- API reference de QNX.
- Informació extreta a la xarxa.
- Article d'Ollero sobre el seguiment d'una línia per un robot mòbil[F].
- Guia per al calibratge de la càmera[G].
- Informació vària proporcionada per diferents Professors de l'escola.

1.2 OBJECTIUS

Així doncs, arribat aquí, la meva intenció principal amb aquest projecte consisteix en desenvolupar una aplicació per al sistema operatiu de QNX que permeti al robot Robotino planificar una ruta, utilitzant diferents tipus d'algorismes de pathfinding, i recórrer aquesta ruta sobre una graella amb diferents nodes pintada al terra del laboratori de l'Escola utilitzant la visió per una càmera.

Per tal d'aconseguir l'objectiu principal, abans s'han d'aconseguir uns objectius secundaris que son els següents:

- Primer de tot, observar com funcionen els projectes anteriors, començant amb el primer. Per poder provar-ho s'haurà d'utilitzar un PC que contingui QNX.
- Revisar les API's i, en cas necessari, crear noves funcions.
- Fer un programa propi que sigui capaç de fer peticions d'imatge, després processar la imatge, després que sigui capaç de fer al robot seguir la línia i també que sigui capaç de girar els graus que se li indiqui i quan se li requereixi.
- Utilitzar la càmera amb perspectiva. Això implicarà haver de recuperar el calibratge de la càmera del segon projecte citat[2] (o trobar una nova forma de fer-ho).
- Treballar amb el nou IDE 5.0 de QNX i exportar-hi tot allò que hem sigui útil (llibreries, API, etc).
- Adaptar diferents algorismes de Pathfinding en llenguatge C.
- Crear un codi amb diferents modes de treball que permeti al robot navegar seguint unes indicacions.
- Integrar a tot això un nivell superior encarregat del càlcul de rutes i que enviï les indicacions de moviment al nivell inferior.
- Crear un "nivell intermedi" (que finalment he integrat al superior) que faci de traductor entre el superior, que ens donarà una successió de punts per els que ha de passar el robot fins a arribar al destí, i l'inferior que nomes sap seguir la línia o girar.
- Finalment crear una GUI per poder supervisar el procés des del PC Host.

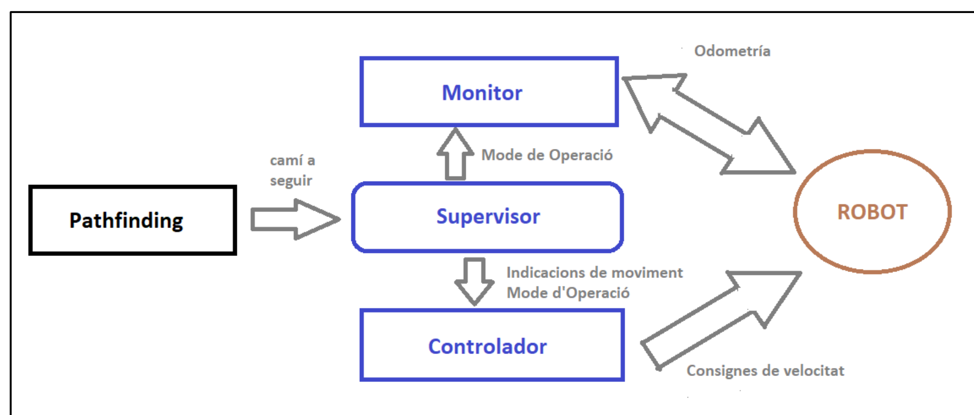


Figura 1.3: Esquema bàsic de funcionament del programa.

NOTA: En els dos últims projectes es van utilitzar càmeres diferents. En el meu projecte, com la 2a càmera és més eficient, utilitzaré aquesta: AXIS 207W.

2 ROBOTINO

2.1 ¿QUE ÉS?

El robot Robotino es un robot mòbil de la empresa alemanya d'automatismes FESTO i va ser creat principalment per a la formació i la recerca en diversos àmbits, entre els quals es troba l'àmbit de la automatització i la tecnologia.

El robot es classifica com un robot mòbil terrestre i una de les seves principals característiques es que es mou a partir d'accionaments omnidireccionals. Això vol dir que les rodes que té el permeten moure's cap endavant, endarrere i lateralment, amés de poder girar sobre un punt. També està equipat amb una web cam (encara que per al projecte s'ha fer servir una altre càmera) i varis tipus de sensors analògics per mesurar distàncies, sensors binaris per protecció de col·lisions (que anomenem Bumper) i encoders per detectar la velocitat real o la posició entre altres.

Figura 2.1: Vista frontal del robot ROBOTINO

El fet de posseir un processador intern permet al sistema posar-se en marxa immediatament sense la necessitat de connectar-lo a un PC, ja que les aplicacions de demostració que té a la targeta Compact flaix poden ser executades des del seu panell de control.

En definitiva, Robotino és autònom, conté nombrosos sensors, una càmera i un controlador d'altes prestacions. Es pot accedir al controlador directament mitjançant LAN sense fils (WLAN) i quan està correctament programat, Robotino realitza de forma autònoma les tasques assignades. Finalment, també es poden connectar motors i sensors addicionals mitjançant el port d'entrades i sortides que té incorporat.

2.2 COMPONENTS

2.2.1 XASSÍS / PONT DE COMANDAMENT

Estructuralment, el Robotino consta d'un xassís d'acer inoxidable circular que suporta tots els seus components i disposa de dues nanses. Aquest xassís té les següents característiques:

- Diàmetre: 370 mm
- Altura, incloent carcassa: 210 mm
- Pes total: aprox. 11 kg

Sobre el xassís trobem les bateries recarregables, les unitats d'accionament i la càmera. També es troben distribuïts al seu perímetre els sensors de mesura per distancies i el para-xocs (bumper). A més a més, el xassís ofereix un espai addicional i opcions de muntatge per afegir altres sensors i/o motors.
Els components sensibles del sistema, tals com el controlador, el mòdul de E/S i les interfícies, es troben situats en el pont de comandament. El pont de comandament està connectat a la restà de mòduls del sistema per mitjà d'un connector.

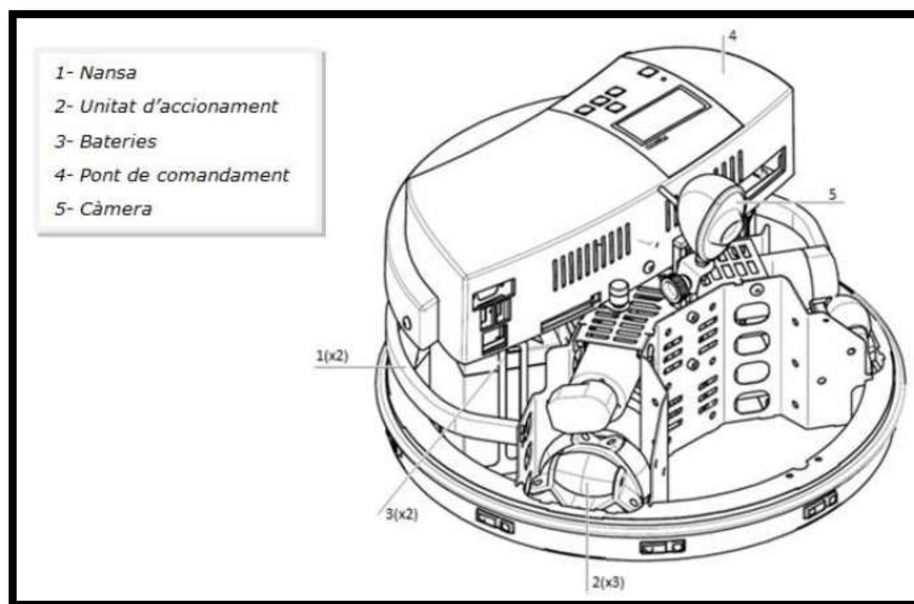


Figura 2.2: Parts del Robotino.

2.2.2 *MÒDULS D'ACCIONAMENT*

Per moure's, el Robotino consta de 3 unitats d'accionament omnidireccionals i independents que es troben muntades formant un angle de 120º entre sí. Cada una de les 3 unitats consten dels següents components:

1. Motor DC
2. Encoder incremental
3. Rodets omnidireccionals
4. Reductor amb relació 16:1
5. Corretja dentada

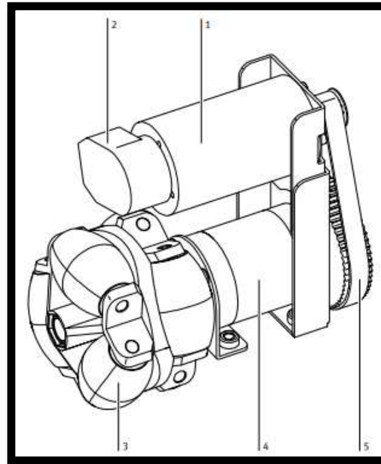


Figura 2.3: Mòdul d'accionament

Tots els components individuals estan fixats a la brida de muntatge en la part posterior. Conjuntament amb la brida frontal, per assegurar el correcte posicionat de les unitats d'accionament entre sí, cadascuna d'elles està subjectada al xassís amb cargols.

La velocitat del motor està controlada mitjançant un llaç de control intern. Es tracta d'un controlador de tipus PID per a cada un dels motors, i els seus paràmetres poden ser definits per l'usuari o es poden utilitzar els valors per defecte. Les velocitats reals dels motors s'obtenen mitjançant uns encoders incrementals.

El rodet omnidireccional es posa en moviment en una determinada direcció per mitja dels seus eixos d'accionament i és capaç de desplaçar-se en qualsevol direcció si es veu forçat per altres accionaments en direccions diferents. Com a resultat de la interacció amb les altres dos unitats d'accionament, és possible obtenir un recorregut en una direcció que difereix de la direcció de cadascun dels respectius accionaments.
A part d'aquestes tres unitats d'accionament, és possible instal·lar-hi una més ja que Robotino disposa de l'espai necessari per a la instal·lació. A més, cal remarcar que aquesta permet un control de posició i una d'intensitat. Apart, el propi fabricant proporciona un actuador opcional anomenat pinça elèctrica.

2.2.3 *BATERIES / CARREGADOR*

La alimentació elèctrica és subministrada per dues bateries recarregables de 12V amb una capacitat de 4 Ah. Ambdues bateries recarregables estan muntades en el xassís com s'ha explicat en l'apartat anterior. A més a més, el Robotino inclou dues bateries addicionals i un carregador de bateries. Així, mentre dues bateries estan en funcionament, les altres dues poden estar en procés de carrega. No obstant, també existeix la possibilitat de treballar amb el Robotino connectat al carregador tot i que això limita molt la mobilitat d'aquest i, a la llarga, afecta notablement a la vida útil de les bateries.

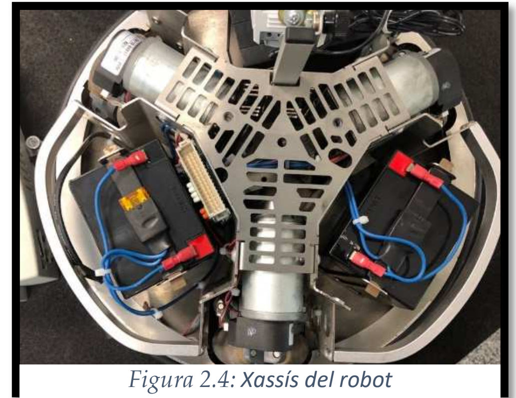


Figura 2.4: Xassís del robot

2.2.4 *UNITAT DE CONTROL*

A aquesta unitat és on es connecten els diversos sensors, els motors, les entrades digitals, entrades analògiques, etc.

El robot Robotino disposa d'un controlador que consisteix en varis components:

- Un processador PC 104 (MOPSlcdVE) amb memòria SDRAM de 1 GB, 400 MHz, i sistema operatiu Linux amb kernel en temps real.

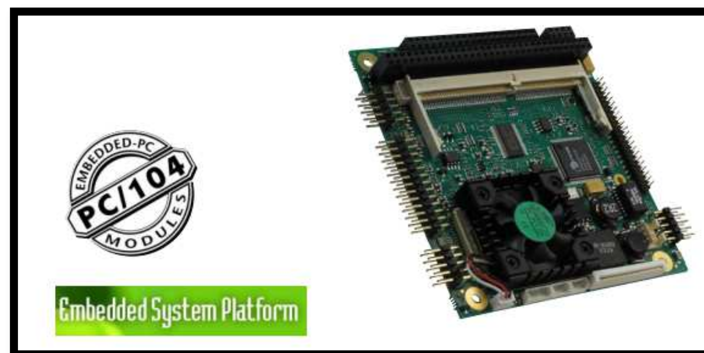


Figura 2.5: PC 104 MOPSlcdVE.

- Una targeta Compact Flash

La targeta te 1MB en la qual hi han instal·lades varies aplicacions de demostració i el sistema operatiu que, en aquest cas, ens permet treballar tant en Linux com amb QNX.

Per Linux es fa servir el Kernel en temps real (RTAI) i s'hi poden trobar les llibreries necessàries per poder executar aplicacions.

Per QNX es fa servir QNX Neutrino que és un sistema operatiu de temps real (RTOS: Real-Time Operating Systems) desenvolupat per QNX Software Systems Ltd. Està constituït per un autèntic micro-kernel rodejat d'un conjunt de mòduls opcionals (resource managers) que poden ser inclosos segons les necessitats de l'usuari.

- Un punt d'accés LAN sense fils (WLAN). Gràcies a això, es possible comunicar-se des de un PC amb el robot sense tenir la necessitat de connectar-se amb fils.

També se'ns faciliten API's Linux i C++ per a la programació del Robot. Això es pot fer amb diverses plataformes, per exemple, amb Matlab, ROS, RobotinoView, QNX, etc.

2.2.4.1 Placa de control d'entrades i sortides (EA09)

La placa de control d'entrades i sortides (a partir d'ara I/O-Board o EA09) proporciona l'accés als motors, sensors, entrades analògiques i digitals, sortides digitals i relés del Robotino. És connecta amb el PC/104 gràcies a una de les seves dues interfícies sèrie (RS-232). Aquesta comunicació és extremadament crítica, per això qualsevol retard en la comunicació té efectes dramàtics en el comportament del Robotino. Per aquesta raó no està implementat ni el control de flux per hardware ni el de software.

Un tret important d'Aquesta placa és que permet realitzar un llaç de control de velocitat sobre cadascun dels actuadors amb una freqüència de 1kHz. També incorpora un timer per saber el temps de funcionament del motor i inclús un comptador de posició per cada encoder.

2.2.5 PUNT D'ACCÉS LAN SENSE FILS

Figura 2.6: Accés point del Robotino

Cal saber que el Robotino té el seu propi punt d'accés (Access Point) sense fils. Esta inclòs dins de la "closca" superior del robot, a sota de la placa i es troba subjectat amb cargols per a que no es mogui.

Pel que fa als PC, aquests es poden connectar a l'Access Point extern via cable (Ethernet) o WLAN. En el nostre cas esta connectada al PC del propi robot per mitjà d'un mini-cable ethernet i, per enviar/executar programes i demés, utilitzem la connexió sense fils al PC *Host*.
Els ordinadors del laboratori estan connectats via cable (Ethernet) amb un mòdem sense fils i aquest es l'encarregat d'establir la connexió via WLAN amb el nostre Robotino. Per defecte, la direcció de la xarxa del robot és 172.26.1.1 però es totalment modificable. En el nostre cas és **192.168.1.118**.

Aquest Access Point es de la marca Level One i te les següents característiques:

- Compleix amb els estàndards IEEE 802.11g i 802.11b.
- Es caracteritza per el seu baix consum de corrent. És possible una alimentació per mitja del port USB.
- Permet velocitats de transmissió de fins 54 Mb per segon per 802.11g, i 11 Mb per segons per 802.11b amb un ample rang de les transmissions (fins 100 m dintre d'edificis).
- Permet establir una xarxa segura amb encriptació WEP i funció WPA-PSK.
- És ràpida i simple de configurar per mitja de la utilitat de gestió de la web.

Te tres modes de treball: AP(Acces Point), Client i "Wireless RT" però només ens interessen els dos primers. En el nostre cas està configurat per treballar en **mode Client**. En aquest mode, l'Access Point funciona exactament com ho faria un mòdem sense fils WI-FI, és a dir, aquest intenta buscar un Access Point extern amb unes determinades especificacions.

2.2.6 *SENSORS*

El Robotino consta de diversos sensors, alguns d'ells ja estan incorporats i d'altres són opcionals. Molts d'ells els proporciona el propi fabricant però també hi han fabricats per tercers.

2.2.6.1 *Sensors d'infrarojos*

El robot està equipat amb nou sensors de mesura de distància per infrarojos de la marca SHARP (model GP2D12) i es troben muntats en el xassís formant un angle de 40º entre sí. Amb aquests sensors el Robotino pot detectar objectes en les zones circumdants.
Cadascun d'aquests sensors pot ser interrogat individualment per mitja de la placa de circuit de E/S. Es poden fer servit per evitar obstacles, mantenir distancies i/o adoptar proteccions enfront a un determinat objectiu, etc.

Els sensors són capaços de mesurar distancies amb precisió relatives a objectes de forma analògica. Fent conversions dels valors obtinguts en forma de Voltatge es poden obtenir valors en cm (entre 4 i 30 cm). La connexió del sensor és especialment senzilla, només es necessita un cable per al senyal analògic i un altre per a la alimentació.

2.2.6.2 *Para-xocs (Bumper)*

El para-xocs està format per una banda de detecció fixada al voltant d'una anella que rodeja el xassís. La estructura consta d'una càmera de commutació que es trobat dintre de la banda de plàstic. Dintre d'aquesta càmera hi han dues superfícies conductores mantenint una determinada distància entre sí i entren en contacte quan s'aplica un mínima pressió a la banda.
Amb això, una senyal perfectament coneguda és transmesa a la unitat de control. Un exemple podria ser quan es detecta qualsevol col·lisió amb un objecte en qualsevol punt del cos, això provocaria la detenció immediata del Robotino per tal de no danyar la seva estructura.

2.2.6.3 Ampliacions

El proveïdor de Robotino, FESTO, ens ofereix una llarga llista de possibles ampliacions pel que fa als actuadors i sensors del Robot. L'avantatge d'utilitzar els sensors/actuadors que ens ofereix FESTO és que ja hi ha funcions implementades per el seu ús. En el cas de voler utilitzar un sensor/actuador fabricat per tercers s'hauria de programar des de zero.

Les principals ampliacions disponibles són:

- Sensor de proximitat inductiu: Serveix per detectar objectes metàl·lics en el terra i s'utilitza per el control filo guiat. (adquirit per la EET).
- Sensor NorthStar: és un sistema de localització per infrarojos. Es basa en un sensor d'infrarojos que determina la seva posició i orientació a partir de la posició de dos punts de llum detectats.
- Làser RangeFinder: Permet posicionar el robot en la zona de treball.

- Pinça elèctrica (Gripper): La seva principal finalitat és poder recollir diferent tipus de peces gràcies al corrent que se li aplica al motor ubicat a l'extrem. La presència d'una peça en la pinça es detecta mitjançant un sensor de barrera de llum.
- Gyroscope sensor: sensor que ens permet mesurar l'angle de gir del robot, ens permet corregir errors de gir en cas de que les rodes rellisquessin, fet que la odometría no pot contemplar de cap manera.

2.2.7 CÀMERA

Com s'ha dit anteriorment, el robot ve equipat amb una càmera de sèrie però per aquest projecte s'ha fet servir una altre.

Aquesta càmera es de la marca Creative i la seva posició és bastant flexible. Es pot ajustar la altura i la inclinació i permet visualitzar imatges en directe amb l'ajuda del Robotino®View. Aquesta interfície ens ofereix diverses opcions de processament d'imatges per a qualsevol aplicació posterior. Els resultats es poden utilitzar per senyalar objectes amb precisió, així com per el seguiment de recorreguts i objectes.

A continuació es mostren a la *figura 2.7.* les especificacions tècniques que porta de fabrica:

Sensor d'imatges	Color VGA CMOS
Profunditat de color	24 Bit
Connexió a PC	USB 1.1
Resolució de Vídeo	160 x 120, 30 fps (SQCGA) 176 x 144, 30 fps (QCIF) 320 x 240, 30 fps (QVGA) 352 x 288, 30 fps (CIF) 640 x 480, 15 fps (VGA)
Resolució d'imatge parada	160 x 120 (SQCGA) 176 x 144 (QCIF) 320 x 240 (QVGA) 352 x 288 (CIF) 640 x 480 (VGA) 1024 x 768 (SVGA)
Format de captura d'imatge	BMP, JPG

Figura 2.7.

3 SISTEMES OPERATIUS DE TEMPS REAL (SOTR)

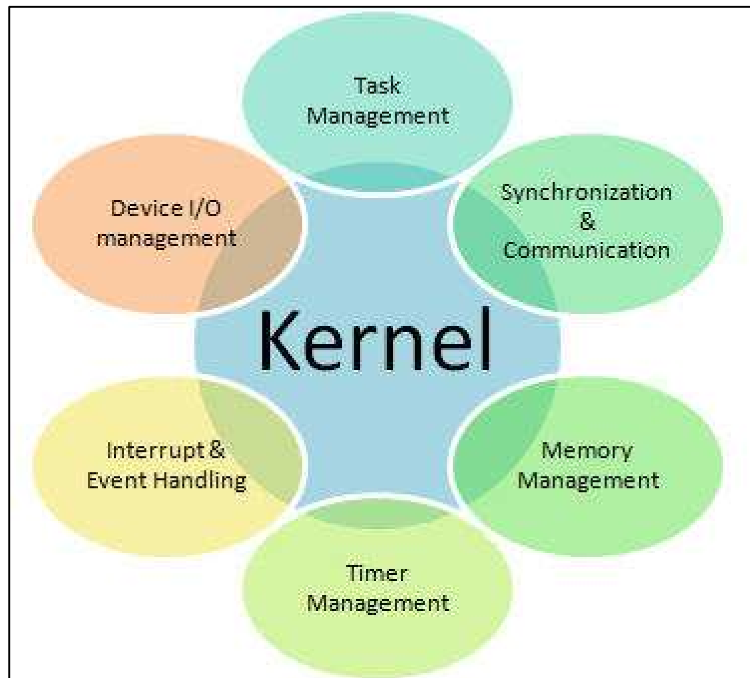


Figura 3.1: Estructura d'un STOR).

3.1 DEFINCIÓ

Un sistema operatiu de temps real (o SOTR) es un sistema informàtic amb la capacitat d'interactuar amb el seu entorn en intervals de temps preestablerts i normalment breus. Es un tipus de SO que ha sigut desenvolupat per aplicacions que requereixen ser tractades en temps real, es a dir, que exigeix no només que les accions empreses siguin lògicament correctes sinó que també es produeixin dintre d'un interval de temps especificat. Per garantit el comportament correcte en el temps requerit es necessita que el sistema sigui predictible.

Es classifiquen bàsicament en tres grups:

- **STR crític:** Un sistema de temps real es considera crític quan es imprescindible complir amb les especificacions temporals.
- **STR acrític:** Quan les especificacions temporals es poden violar ocasionalment (per exemple control de temperatura, control de nivell).
- **STR estricte:** Sistema crític amb especificacions temporals molt curtes o breus.

3.2 CARACTERÍSTIQUES GENERALS DELS SOTR

Per poder considerar un sistema operatiu om un SOTR ha de complir una sèrie de condicions:

- Possibilitat d'execució multi-tasca.
- Possibilitat d'assignar prioritats a les tasques definides.
- Possibilitat de comunicació entre les tasques i mitjans propis per fer-ho possible.
- Comportament temporal conegut.
- Ha de ser del tipus determinista, és a dir, que es pot saber amb una gran probabilitat quin serà el temps que tarda una tasca en iniciar-se.
- Responsivitat: el temps que tarda una tasca en executar-se. Les variables que hem de saber són:
    - Temps d'iniciació
    - Temps necessari per executar la tasca
    - Efectes de la interrupció (altres efectes que poden interferir …)
- Confiabilitat: el sistema no pot presentar errors dintre d'un límit especificat ja que ha d'assegurar el correcte funcionament.

Altres característiques generals son:

- No utilitza molta memòria.
- Qualsevol esdeveniment en el suport físic pot fer que s'executi una tasca.
- Multi-arquitectura (codi portable a qualsevol tipus de CPU).
- Molts tenen temps de resposta predictible per esdeveniments electrònics.

Molts sistemes de temps real tenen un planificador (scheduler) que es una espècie de controlador que minimitza els períodes en els que les interrupcions estan inhabilitades. Aquests períodes són un temps finit conegut de la duració de les interrupcions calculat per el pitjor dels casos. Altres inclouen formes especials de gestió de memòria que limiten la possibilitat de fragmentació de la memòria i asseguren un límit superior mínim per als temps d'assignació i de retir de la memòria assignada.

3.3 EL SOTR QNX

QNX es un sistema operatiu de temps real que ha sigut desenvolupat principalment per treballar amb sistemes encastats. Un sistema encastat ("embedded") es aquell en el que s'executa una aplicació de forma remota i va integrat dins d'un sistema major al qual ha de controlar i supervisar.
Per exemple, el sistema de control i supervisió d'un rentadora va integrat dins d'ella. Aquest tipus de sistemes han d'interactuar en temps real amb el sistema major, complint unes especificacions temporals. Per això, típicament els sistemes encastats s'han de dissenyar com a sistemes en temps real.

QNX ens dona una gran flexibilitat a l'hora de programar ja que es fàcilment personalitzable i ens permet configurar-lo de manera que nomes utilitzem els recursos necessaris per a la nostre tasca. Amés, segueix la norma POSIX (Portable Operating System Interface Unix) per tal de mantenir la seva compatibilitat amb altres aplicacions. POSIX és una norma que defineix una interfície, un entorn i una estructura pel sistema operatiu.

3.3.1 HISTÒRIA

A principis dels anys vuitanta els Canadencs Gordon Bell y Dan Dodge van decidir crear un sistema operatiu amb una interfície similar a UNIX, basat en el intercanvi de missatges. Van començar utilitzant-lo per els seus propis processadors 6809 i 8088. Poc després el van adaptar per el IBM PC, anomenant el sistema QUNIX (Quick Unix) i van crear la companyia QSSL (Quantum Software Systems Limited, actualment QNX Software Systems Limited).

A partir d'aquell moment tan la companyia com el seu S.O van anar evolucionant. El sistema va adaptar-se als successius processadors de INTEL, exprimint les avantatges que cadascun oferia. En l'actualitat QNX es comercialitza en més de cent països, i ha estat utilitzat per empreses i institucions tan importants i variades com INDRA, Honda, Philips, Texaco o la NASA, a part d'innumerables universitats.

Figura 3.2: LOGO DE QNX

3.3.2 *PRINCIPIS BÀSICS*

QNX es basa en una configuració tipus "Model de procés universal" (UPM). Aquesta disposició utilitza un microkernel que realitza dos funcions fonamentals:

- **Intercanvi de missatges** assignant les diferents rutes entre tots els processos al llarg del programa.
- Una part del microkernel anomenada scheduler es la encarregada de la **Planificació**. Aquesta part s'invoca sempre que un procés canvia d'estat.

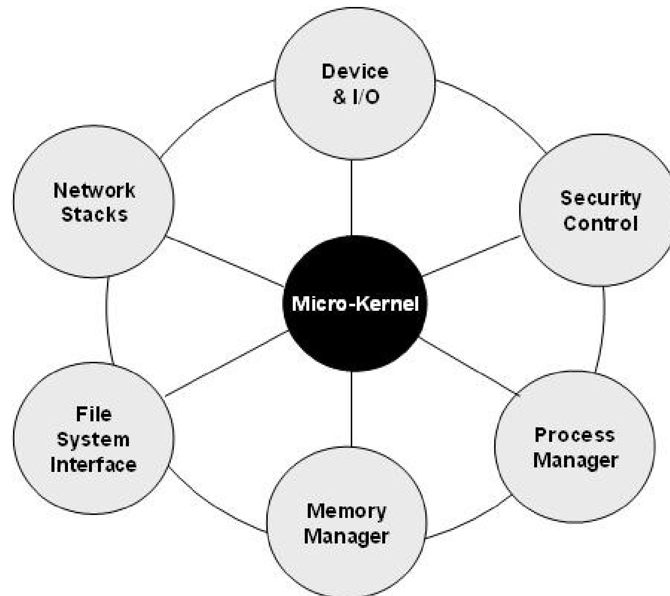


Figura 3.3: Microkernel encarregant-se de diversos processos cooperatius

NOTA: El microkernel no participa en l'encuament. A ell només s'hi arriba a través de crides del kernel ja siguin des d'un procés o d'una interrupció de hardware.

Gràcies a la utilització del microkernel i el tipus de comunicació del que disposa entre processos, QNX es capaç d'obtenir un grau d'eficàcia, modularitat i simplicitat. Aquesta comunicació entre processos es pot fer de 3 formes diferents:

- **Missatges:** Comunicació síncrona proporcionada pel microkernel entre processos cooperatius on el procés que envia el missatge requereix una contestació al missatge.

- **Proxies:** És una forma especial d'enviar un missatge. Estan preparats per la notificació d'un possible esdeveniment, on el procés que ho envia no necessita interactuar amb el receptor.

- **Senyals:** utilitzats entre processos de forma asíncrona.

La restà d'operacions les realitzen els diferents processos, cadascú amb el seu propi espai de direccionament de memòria, protegits entre si. La principal avantatge d'aquesta configuració és que la falla d'una tasca no afecta a les altres. Els processos poden ser parats o reiniciats en cas de falla. D'aquesta manera s'obté un sistema molt més segur i fiable que amb altres configuracions, com l'executiu de temps real o la arquitectura monolítica. De fet QNX s'utilitza en alguns serveis com braços de naus especials o maquines de poliments de vidre, on les falles de software són inacceptables.

3.3.3 *CARACTERÍSTIQUES*

Les principals característiques d'aquest sistema operatiu son, entre d'altres, l'espai de memòria protegit per a cada procés (de SO i d'usuari), la comunicació mitjançant pas de missatge i la gestió multitasca basada en prioritats (256 nivells). També disposa d'una arquitectura modular al voltant d'un micronucli composat per:

- Nucli de 64 Kbytes: gestió de processos i tasques, mecanismes de sincronisme i de pas de missatges, temporitzadors...
- Processos de SO opcionals:
    - gestió de fitxers (discos locals i distribuïts).
    - gestió de la interfície d'usuari.
    - gestió de les comunicacions (TCP/IP, QNET).
    - gestió de dispositius (Flash, UART, USB).

Es diu que aquest micronucli es instrumentat perquè permet rastrejar els esdeveniments del sistema (interrupcions, missatges, estats de les tasques, canvis de context, sincronisme) per facilitar la detecció d'errors de programació en temps d'execució. També disposa de les següents facilitats per a sistemes encastats:

- Processadors: MIPS, PowerPC, x86, SH-4, ARM, StrongARM, XScale.
- Eines: compilació creuada, descàrrega d'imatges i depuració via RS-232 i Ethernet.
- SOTR modular.
- Interfície gràfica d'usuari encastable.

Amés, compleix l'estàndard POSIX (Portable Operating System Interface based on UNIX): API estàndard portable. Té latències de commutació optimitzades (canvi de context [8s], servei interrupció [4s]), suporta SMP (Symmetric Multiprocessing) i permet accés transparent a recursos distribuïts en xarxa.

3.3.4 *VARIANTS DE QNX*

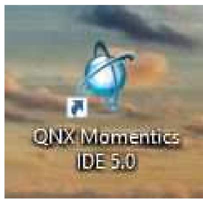
El microkernel de QNX, anomenat Neutrino, està implementat en 4 variants que desenvolupa i comercialitza la companyia:

- **QNX Neutrino RTOS:** Aquesta versió es la més completa i robusta pensada per complir els requeriments de sistemes encastats. Es un microkernel real d'arquitectura modular.

- **QNX OS for Safety:** Està dissenyada per complir amb les normes ISO 26262 en ASIL D i les normes IEC 61508 en SIL3. Proveeix d'un sistema dissenyat sobre una base segura, per implementar en sistemes crítics com automòbils, trens i automatització industrial.

- **QNX OS for Medical:** Compleix les normes IEC 62304 i està dissenyat per reduir l'esforç en el desenvolupament de dispositius mèdics que requereixen aprovacions regulatives.
- **QNX OS for Security:** Es un RTOS de característiques completes, certificat en norma ISO/IEC 15408 EAL 4+.

3.4 EL NOU IDE 5.0



Momentics tool suite proveeix un entorn únic, complert, consistent i integrat de desenvolupament per QNX Neutrino independentment de la plataforma que s'utilitza al PC HOST (Windows o Linux). Permet instal·lar i treballar amb múltiples versions de QNX Neutrino (A partir de la 6.2.1) en sistemes encastats.

Fins ara, QNX funcionava com qualsevol altre sistema operatiu; s'instal·lava al PC i es feia carregar al menú de SO's inicial on podríem trobar altres SO's com ara Windows o Linux. Això vol dir que abans QNX tenia el seu propi entorn i la seva pròpia forma d'interactuar amb altres dispositius.

Però, a partir de la versió 5.0, QNX ja no corre més com un sistema operatiu independent sinó com una aplicació més de Windows que serveix com entorn per desenvolupar aplicacions per a sistemes encastats basats en QNX. Això aporta millores en compatibilitat i simplicitat ja que no fa falta incorporar-hi tot allò que ja tenim a Windows. L'IDE esta integrat amb utilitats de QNX que realitzen un nombre de funcions, incloent la creació, compilació i la depuració de projectes amés de proveir comunicació entre el *Host* i el *Target*.

NOTA: Anomenem ***host*** al PC on resideix l'IDE (per exemple Windows, Linux), i ***target*** al PC on corren tant el QNX Neutrino com el programa creat.

Per veure com s'instal·la podem veure la guia a l'annex A. L'IDE presenta varies formes de visualització i treball amb tots els components del sistema a través d'una sèrie de finestres relacionades. En termes de les tasques que es poden realitzar, els toolsets permeten:

- Organitzar els recursos (projectes, carpetes, arxius).
- Editar recursos.
- Cooperar en projectes amb un equip.
- Compilar, executar i depurar programes.
- Creat imatges flash o del SO per sistemes encastats.
- Analitzar i afinar el funcionament del sistema.

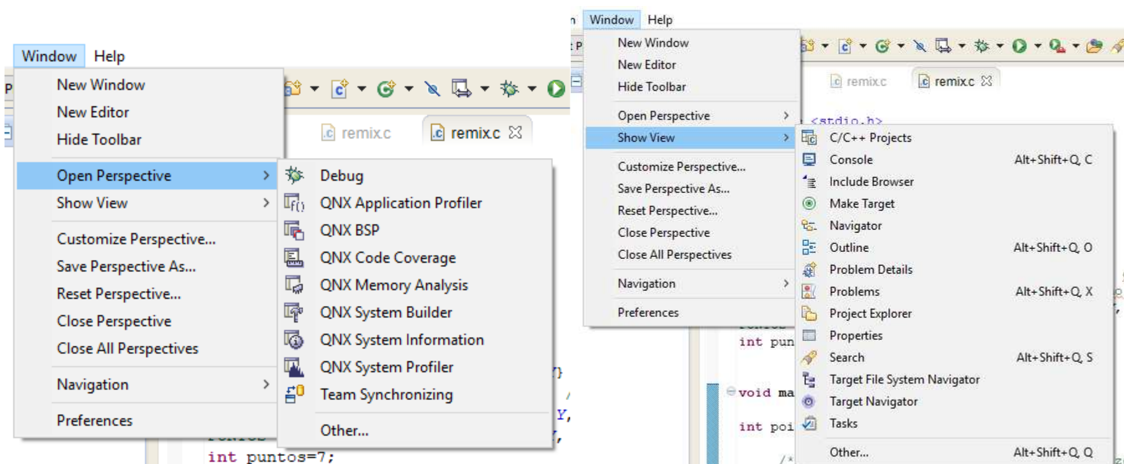


Figura 3.3: Menú per triar diferents finestres i àrees de treball a l'IDE 5.0

Indiferentment de si es vol connectar a un *target* local o remot, s'ha de preparar el *target* de forma que l'IDE pugui interactuar amb la seva imatge de QNX Neutrino (QNX Target System Project.) Per fer-ho veure l'annexB. Una vegada configurat ens apareixerà a la finestra '*Target Navigator*'.

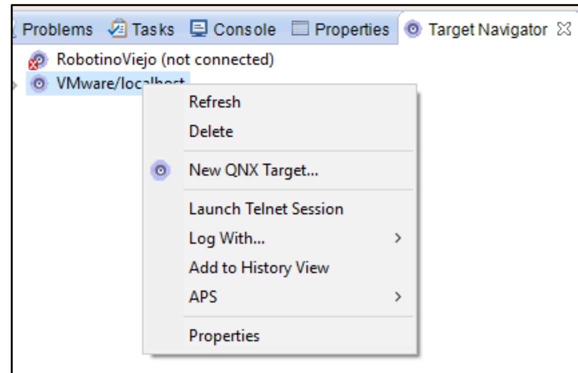


Figura 3.4.

VMware es la maquina virtual que simula el *target* i és útil per aplicacions que no necessiten un *target* real. Hi ha una finestra amb el nom semblant '*Target File System Navigator*' però té una finalitat totalment diferent. Aquesta finestra de l'IDE permet moure fitxers entre diferents dispositius amb memòria com ara entre el nostre PC host i les carpetes d'arxius del *target* utilitzant copy-paste o arrossegant i deixar anar.

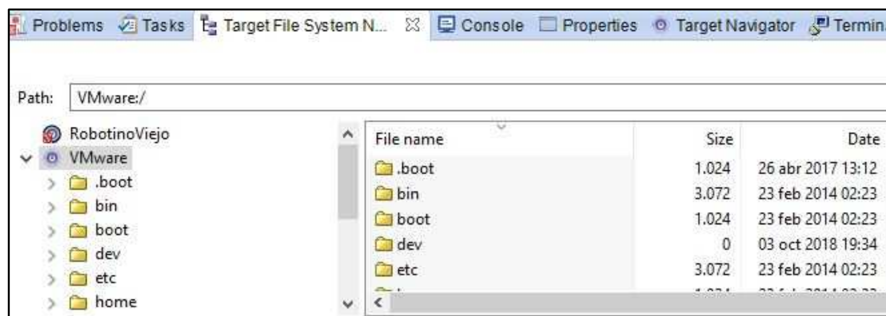


Figura 3.5.

Per hosts de Windows i Linux, l'IDE suporta comunicacions *host-target* utilitzant tant direcció IP com connexió sèrie, ambdós son recomanats: Si només es disposa de connexió sèrie es possible depurar el programa però es necessària la connexió IP per poder utilitzar qualsevol eina avançada de diagnosis de l'IDE. Això si, tant el host com el *target* han d'estar connectats a la mateixa xarxa, i es requereix que aquesta sigui TCP/IP.

Per fer possible la connexió també es requereix executar en el *target* el ***qconn*** ja sigui des de la línia de comandes o des de el boot script (seqüencia d'arrencada) del target. De forma resumida, el qconn serveix per proveir diferents serveis de suport a components remots de l'IDE en funció del que se li demani des del host. Permet executar aplicacions en el target a qualsevol usuari, fet que compromet bastant la seguretat del sistema. Per això es recomana utilitzar-lo en elements privats i que no estiguin de cara al públic.

4 CÀMERA AXIS 207W

Axis és una empresa sueca fundada al 1984. És líder en el mercat internacional de vídeo en xarxa i vídeo vigilància, i els seus productes solen ser utilitzats en cadenes de muntatge, aeroports, trens, autopistes, presons i casinos.
Va ser la primera empresa en treure al mercat una càmera de xarxa en el 1996 i des de llavors no han deixat de treure nous productes i d'innovar en el vídeo en xarxa i vídeo vigilància.
Les càmeres de xarxa d'Axis es basen en estàndards oberts per poder connectar-se a qualsevol xarxa IP, incloent Internet, i permeten la visualització i gravació remota des de qualsevol lloc del món. Ofereixen també altres característiques com la detecció de moviment, detecció d'àudio i alarma anti-manipulació.
Produeixen una gran varietat de tipus de càmeres en xarxa i per a diferents propòsits:

- Càmeres de xarxa fixes, és un disseny de càmera tradicional, la direcció de captura és fixa una vegada muntada.
- Càmeres de xarxa domo fixes, consta de una càmera de petites dimensions amb una carcassa de forma ovulada el seu disseny és discret i passa desapercebuda.
- Càmeres de xarxa PTZ1,es poden controlar el seu moviment remotament amb un ordinador connectat a la xarxa.
- Càmeres de xarxa ocultes, càmeres de disseny funcional i discret pensades especialment per a la vigilància.
- Càmeres de xarxa amb resolució megapíxel/HDTV2,utilitzades on els detalls de les imatges són importants, com per exemple bona visualització de matricules de cotxe.
- Càmeres de xarxa tèrmiques, proporcionen imatges basades en el calor que emeten els objectes capturats.

Per a la integració en el Robotino s'ha escollit una càmera de xarxa fixa, ja que són de dimensions petites. El seu suport es pot acoblar fàcilment al Robotino, poden utilitzar WI-FI i la transmissió d'imatge és ràpida, a part són totalment compatibles amb la VAPIX (api que proporciona AXIS per al desenvolupament d'aplicacions). Dins d'aquesta classe s'ha escollit la càmera 207W.

4.1 CARACTERÍSTIQUES

La següent taula resumeix les característiques principals de a càmera:

AXIS 207W: resolució VGA, interfície de cable i sense fils.	
Sensor d'imatge	CMOS d'escombrat progressiu de 1/4'',VGA i RGB
Objectiu	4,0 mm: visió de 55°*, F2.0,iris fix
Sensibilitat lumínica	1-10000 lux, F2.0
Vel. De obturació	1/10000 s a 1/2 s
Compressió de vídeo	MPEG-4 Part 2 (ISO/IEC 14496-2) MOTION JPEG
Resolució	Màx, 640x480
Vel. D'imatge	30 imatges per segon
Interfície sense fils	IEEE 802.11b/g
Seguretat de xarxa	WEP 64/128 bit, WPA/WPA2-PSK, WPA, WPA2 Enterprise
Protocols compatibles	IPv6, HTTPS, SNMPv1/v2c/v3 (MIB-II)
Memòria	32 MB de RAM, 8 MB de Flash
Alimentació	4,9 - 5,1 V CC, màxim 3,5W
Connectors	RJ-45 Ethernet 10BASE-T/100BASE-TX, Auto-MDIX

4.2 VISIÓ GENERAL

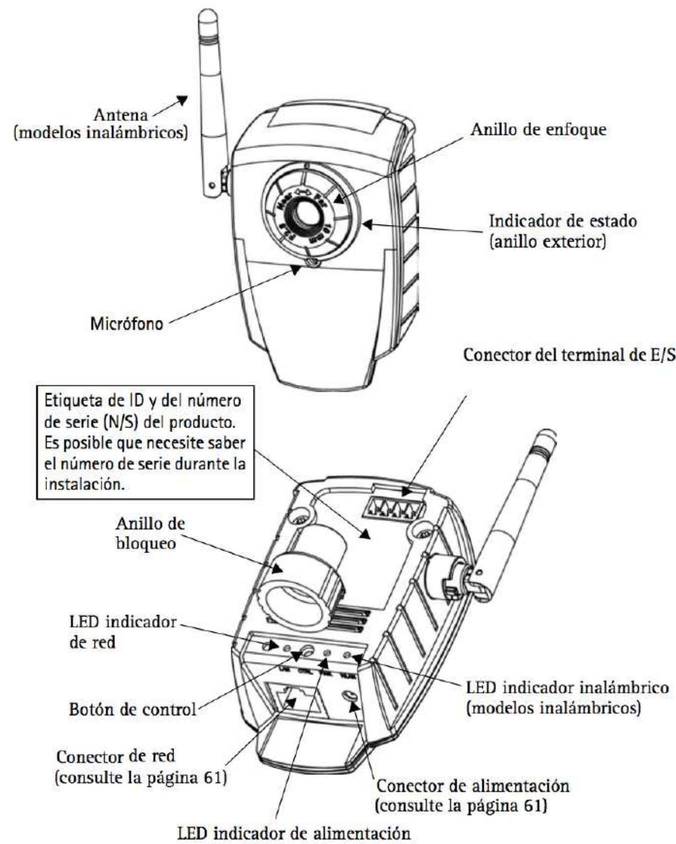


Figura 4.1: Visió general de la càmera de xarxa

- El connector de la alimentació: Per a la connexió de l'adaptador d'alimentació. (4.9-5.1V DC, Max 3,2A.)
- Etiqueta ID: Es el Nombre de Sèrie que es demana durant la instal·lació.
- Botó de control: Per restaurar la configuració per defecte.

LED	COLOR	SIGNIFICAT
Indicador de xarxa	Verd	Connexió a xarxa funcionant a 100 Mbits/s
	Taronja	Connexió a xarxa funcionant a 10 Mbits/s
	Apagat	Desconnectada de la xarxa per cable
Indicador d'Estat	Verd	Opera amb normalitat
	Taronja	Càmera arrancant
	Apagat	Càmera no arrenca correctament
Alimentació	Verd	Opera amb normalitat
	Taronja	Actualitzant firmware
Connexió sense fils	Verd	Connectat a xarxa sense fils
	Taronja	Desconnectat de la xarxa
	Apagat	Connectat a la xarxa per cablejat

4.3 INSTAL·LACIÓ I CONFIGURACIÓ INICIAL

El primer pas per poder treballar amb la càmera serà connectar-la a la mateixa xarxa on tenim l'ordinador supervisor. La xarxa pot tenir connexió a l'exterior mitjançant Internet per si es vol controlar la càmera des de qualsevol punt remot. En el nostre cas, només ha estat connectada a la xarxa local del laboratori d'informàtica industrial de ESAII.

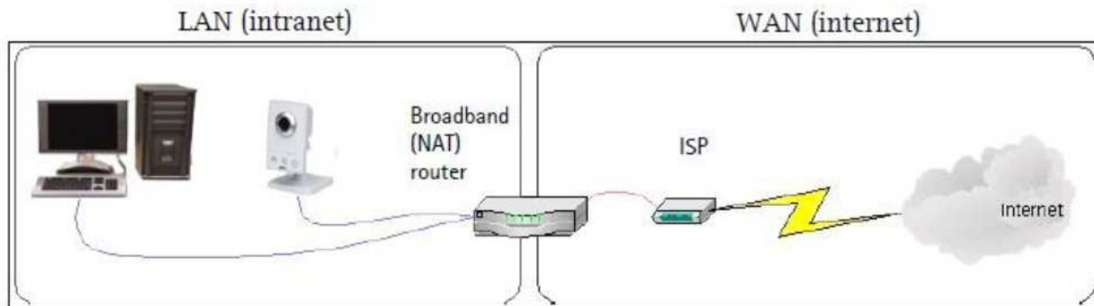


Figura 4.2: Connexió a la xarxa de la càmera

4.3.1 *ASSIGNACIÓ DE IP*

Axis ens proporciona des de la seva web una aplicació gratuïta per a l'assignació de direccions de xarxa a productes de la seva companyia. El software s'anomena IP Utility, només és funcional amb sistemes operatius Windows i s'explica a continuació.

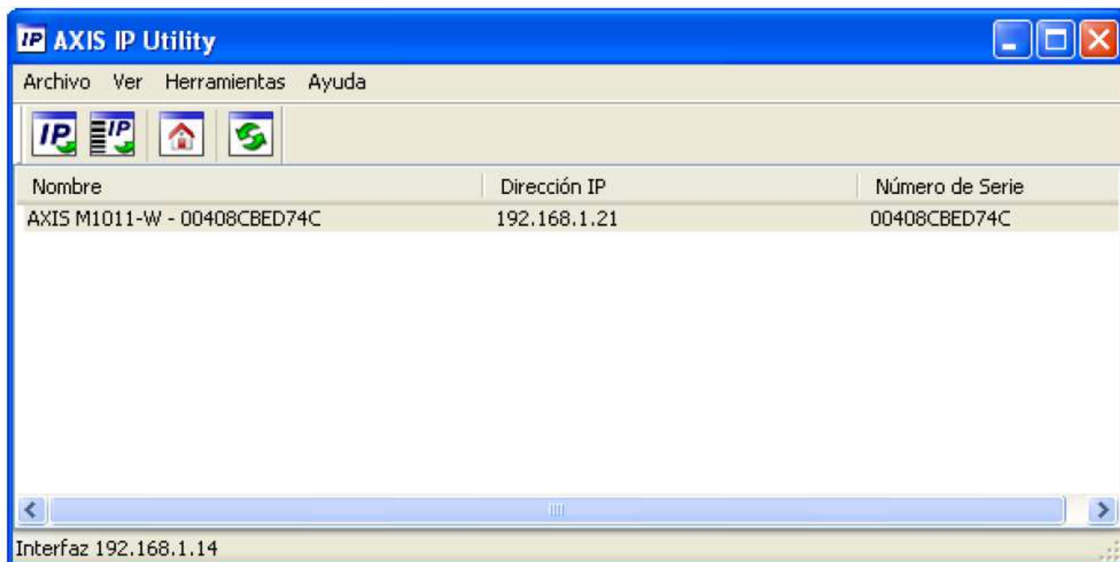


Figura 4.3: Aplicació IP Utility facilitada per AXIS

Al executar l'aplicació, es fa un rastreig automàtic de tots els dispositius connectats a la xarxa i mostra per pantalla els que són de la marca AXIS amb les següents propietats de cada un d'ells:

NOM	DIRECCIÓ IP	NOMBRE DE SERIE
Model-MAC	Direcció IP assignada per el DHCP	MAC/Nº Sèrie

Un cop trobada la nostra càmera dins de la llista hem de fer clic al botó "Configuració de la direcció IP con el número de sèrie" ubicat a la barra d'eines o fent clic dret sobre la càmera detectada. Invocarem la següent pantalla:

Figura 4.4: Assignació de la IP de la càmera de xarxa

On haurem d'assignar la IP estàtica que volem per a la nostre càmera. Hem d'assegurar-nos de que no esta ocupada per cap altre dispositiu de la xarxa.
després d'introduir-la cal clicar 'assignar' i esperar a que el programa acabi de treballar (pot suposar uns quants minuts).
La càmera amb la qual treballem en aquest projecte tenia assignada la direcció: **192.168.1.21**, però, després d'una reestructuració de les adreces dels dispositius del laboratori, se li ha assignat la **192.168.1.114**
Per a altres S.O, Linux o MAC, l'alternativa per a l'assignació de IP es el ARP/Ping instal·lat per defecte en tots els S.O, ja siguin Linux, Mac o Windows. S'executa el terminal i escrivim les següents sentencies:

Sintaxis en Linux/Mac
arp -s <Direcció IP desitjada> <Nºsèrie/MAC> temp ping -s 408 <Direcció IP>

Una vegada fet això la nostre càmera ja es accessible des de la xarxa amb la nova direcció IP.

4.4 INTERFÍCIE WEB

4.4.1 *ACCÉS A LA CÀMERA*

Per accedir a la càmera s'ha d'obrir el navegador i introduir la següent direcció: http://192.168.1.114 . La primera vegada que s'accedeix la càmera ens demana configurar la contrasenya mitjançant un diàleg web:

Figura 4.5: Diàleg web al accedir per primera vegada.

Una vegada configurada, observarem que quan accedim a la càmera, a través del explorador, l'accés esta restringit i ens demana un nom d'usuari i una contrasenya. Per la càmera amb la que estem treballant haurem de posar el següent:

- User name : root.
- Password: ROBOTINO (tot majúscules).

La contrasenya en principi només se'ns demanarà si volem accedir al LIVE VIEW o al setup, si no està configurada d'una altre manera. La visualització de la imatge capturada per a la càmera serà oberta. A partir d'aquí la càmera ja està totalment configurada per accedir a l'aplicació web.

4.4.2 *LIVE VIEW*

És la pagina d'inici que ens apareix una vegada hem accedit a la càmera per web. Se'ns mostra un panell amb tres elements amb els que podem interactuar. Al centre un requadre on es reprodueix la imatge capturada en temps real i a sota hi han els controls d'imatge bàsics.

Figura 4.6: Live View AXIS

A la part esquerra - superior trobem un menú desplegable que ens permet escollir entre els modes de codificació d'imatge disponibles, MPEG-4 i Motion JPEG.
A la dreta trobarem tres enllaços que ens adrecen als tres apartats principals d'aquest panell: Live View, Setup i Help.

4.4.3 *SETUP*

o Basic Setup:

Aquí només s'entra la primera vegada per gestionar els aspectes sobre la connexió i demés.

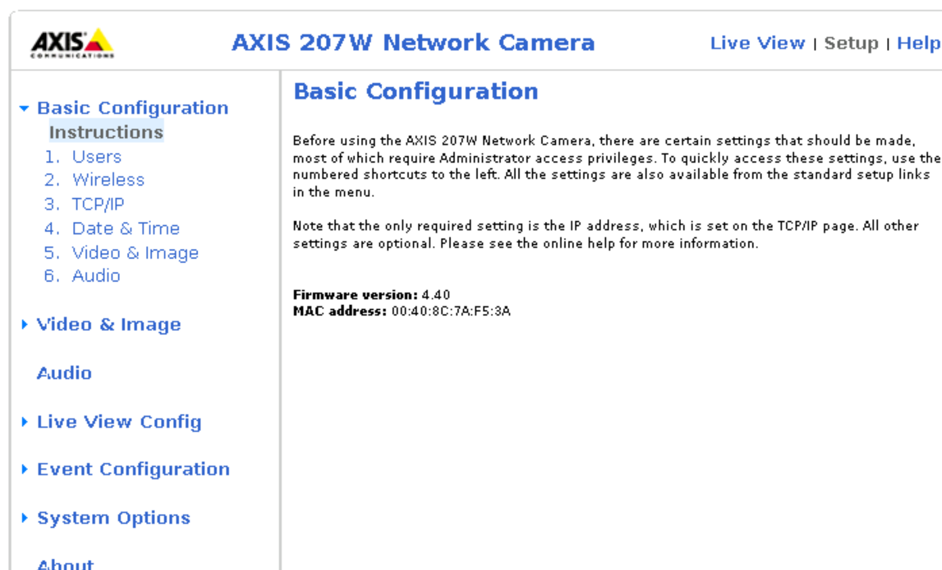
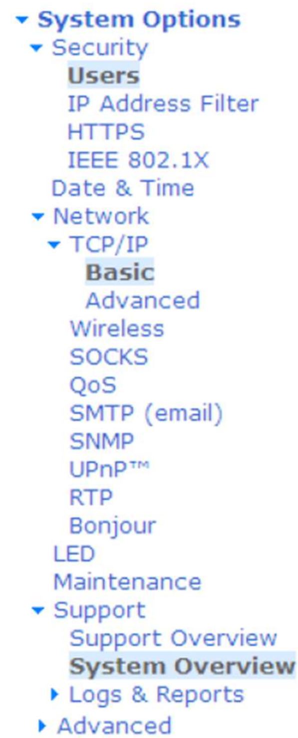


Figura 4.7: Configuració bàsica de la càmera

o System options:

- Security:
    - **Users**: Llistat de diferents usuaris amb diferents privilegis.
    - **IP Addres Filter**: Llistat de IP's per prohibir l'accés.
    - **HTTPS4**: Opció per utilitzar HTTPS.
    - **IEEE 802.1**: Opcions per accedir a una xarxa amb protecció EAPOL5.



- Date & Time: Configuració bàsica de la data i hora.

- Network:
    - **TCP/IP**:
        - Basic: Configuració de IPv4 o IPv6 per la integració a la xarxa. Es pot configurar com a DHCP o estàtica.
        - Advanced: Configuració per connectar la càmera directament a Internet amb les DNS.
    - **Wireless**: Mostra una llista amb les xarxes WIFI detectades i opcions per connectar-te a una d'elles. Les xarxes a les que es pot connectar són les encriptades amb WEP/WPA/WPA2.
    - **SOCKS**: Configuració si volem que la càmera utilitzi SOCKS.
    - **QoS**: Configuració de l'activitat que tindrà la càmera dins d'una xarxa per garantir la bona circulació de dades i funcionament de tràfic a la xarxa.
    - **SMTP (email):** Configuració per enviar esdeveniments o alarmes a un servidor de correu electrònic.
    - **SNMP**: Configuració que permet el seguiment de dispositius des de una xarxa remota.
    - **UPnP**: Paràmetres per si volem que el nostre dispositius apareix-hi en "Mis sitios de red" en un S.O. Windows.
    - **RTP**: Configuració dels ports que volem utilitzar.
    - **Bonjour**: Suport per si volem utilitzar-lo amb productes d'Apple.
- LED: Opcions per a la configuració de l'accionament dels LED's.

- Maintenance: Accions que es poden activar per fer un manteniment preventiu del dispositiu.

- Support:
    - **Support Overview:** Guia de suport per si tenim conflictes amb el dispositiu.
    - **System Overview:** Mostra com està configurat el dispositiu.
- Logs & Reports: Llistat de les ultimes accions que s'han produït al dispositiu.

- Advanced: Opcions per actualitzar el firmware, pujar arxius al dispositiu o "scripts".

4.4.4 CONNEXIÓ SENSE FILS

Una part important d'aquest projecte es basa en la connexió sense fils ja sigui de la càmera o del propi Robotino. Es important perquè la càmera s'ha d'acoblar al Robotino i la seva àrea de treball no pot estar limitada per la longitud d'un cable Ethernet.

A continuació s'explica el procés per a configurar la càmera amb WIFI:

El primer pas és accedir al SETUP > System Options > Network Basic. En aquest apartat se'ns mostren configuracions bàsiques per integrar la càmera a una xarxa, pel que fa a la connexió sense fils el apartat que més ens importa és el "Network Interface Mode".

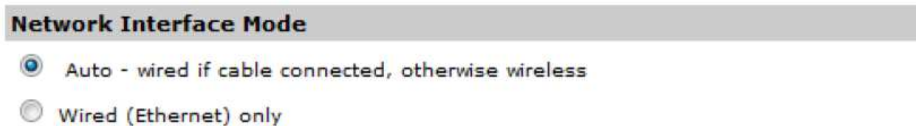


Figura 4.8: Modes de treball de la càmera a la xarxa

Com s'observa a la captura de l'apartat es poden escollir dos opcions: Auto o Wired. Si volem configurar la càmera per que treballi amb WIFI hem d'escollir l'opció Auto. Aquesta opció fa que quan tinguem desconnectat el cable Ethernet la càmera es connecti automàticament a la xarxa sense fils que haguem configurat. En canvi, si connectem el cable Ethernet la càmera treballarà per el cablejat. Aquesta opció és important ja que si configurem la càmera amb WIFI i no la configurem bé o hi ha algun problema sempre som capaços de tornar enrere i tornar a configurar-la connectant-la per Ethernet.

El següent pas és accedir a la pestanya: SETUP > System Options > Network > Wireless. En aquesta nova finestra haurem de configurar la connexió a la xarxa Wifi a la que volem connectar-nos.

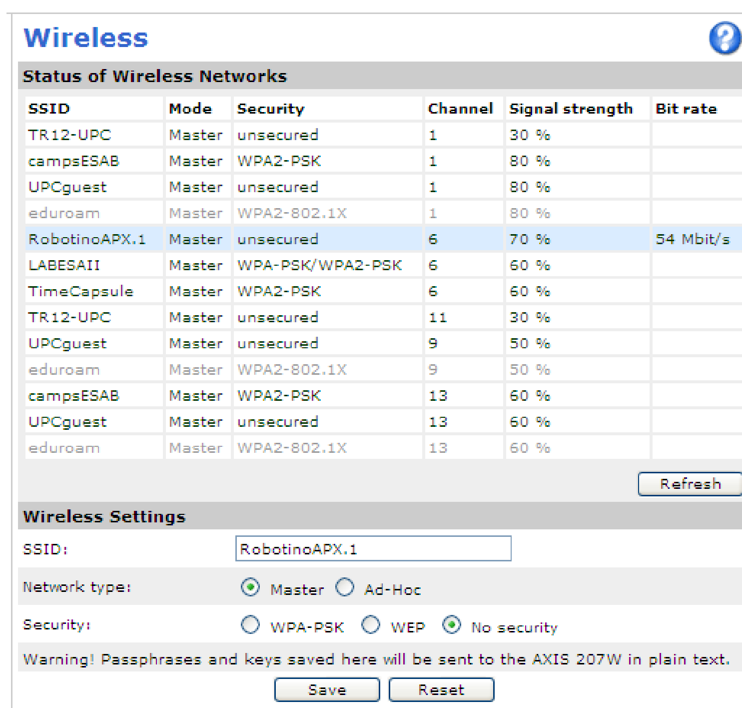


Figura 4.9: Connexió de la càmera a una xarxa WI-FI

En el nostre cas, com volem integrar la càmera dins la xarxa on està connectat el Robotino, escollirem la SSID RobotinoAPX.1. Aquesta xarxa demana una contrasenya d'accés però te filtrat de MAC així que haurem d'autoritzar l'accés de la nostre càmera a la xarxa.
El següent pas serà escollir entre Master o Ad-Hoc. Master s'escollirà quan ens vulguem connectar a un dispositiu enrutador. Aquest mode també és conegut com a mode Acces Point. En canvi Ad-Hoc (també conegut com p2p) s'utilitzarà quan es vulgui connectar la càmera directament a un dispositiu no enrutador, com per exemple un portàtil. En el nostre cas, com utilitzem un Router per a la xarxa haurem d'escollir l'opció **Master**.
En el cas de connectar-nos a una xarxa amb contrasenya, al escollir la xarxa des de la llista, la pestanya 'Security' ens mostrarà automàticament quina codificació té i al costat una textbox on se'ns demanarà la contrasenya.

4.5 VAPIX

Tots els productes de vídeo en xarxa d'Axis tenen una interfície de programació d'aplicacions basades en HTTP anomenada VAPIX ®. Aquesta interfície permet sol·licitar imatges, control de funcions de la càmera de xarxa (PTZ, relés), configurar els valors dels paràmetres interns i molt més. L'objectiu de l'API és fer que sigui més fàcil per als desenvolupadors crear aplicacions que suporten els productes de vídeo d'Axis.

VAPIX ® es compon de:

- Axis API HTTP.
- Axis Parameter Specification.
- Axis RTSP API (pel control de fluxos MPEG-4).

En el nostre cas, hem utilitzat la API HTTP. Aquesta API especifica la interfície de programació d'aplicacions basades en HTTP per a les càmeres i codificadors de vídeo Axis amb la versió de firmware 5.00 o superior. Per saber la versió del firmware del nostre dispositiu s'ha d'accedir a la pestanya *About* del SETUP de la Interfície WEB.
En el nostre cas **tenim la versió firmware 5.20.2**. En el cas de que sigui inferior a 5.00, poc probable, s'haurà d'actualitzar.

La interfície de vídeo basada en HTTP proporciona la funcionalitat per sol·licitar imatges d'una o diverses parts i per obtenir i establir els valors dels paràmetres interns. La imatge i les peticions CGI són manipulades pel servidor web del dispositiu.

El funcionament de la API es senzilla. En l'annex de la memòria del 3r projecte[3] s'adjunta un arxiu anomenat "VAPIX_3_HTTP_API_3_00"on s'expliquen totes les funcions disponibles, el seu funcionament i la seva sintaxis URL per executar-la des de un navegador web.

En la sintaxi d'URL i en les descripcions dels arguments CGI, el text en cursiva i entre parèntesis denotarà contingut que ha de ser reemplaçat per un valor o una cadena. Al substituir la cadena de text els claudàtors angulars també han de ser reemplaçats. Per exemple, <servername> se substitueix per la IP de la càmera.

La resposta de la càmera la rebrem directament a la mateixa finestra del navegador on s'ha enviat la URL de petició. A continuació es mostra un exemple amb la funció Image Size, que serveix per saber quina és la resolució predeterminada de les imatges:

Sintaxi:

```
http://<servername>/axis-cgi/imagesize.cgi?
<argument>=<value>[&<argument>=<value>...]
```

Observant la API, els valors dels arguments i valors s'han de canviar per "càmera" i el valor de la càmera amb la que volem treballar, en el nostre cas, com només en tenim una serà "1". La petició queda de la següent forma:

URL Final:

http://192.168.1.21/axis-cgi/imagesize.cgi?camera=1

Resposta de la càmera:

HTTP Code: 200 OK Content-Type: text/plain Body:
image width = 176 image height = 144

Les primeres línies formen part de la capçalera i no s'observen físicament en el navegador, sinó que les utilitza ell mateix per processar el contingut "body" de la resposta. En aquest cas, el navegador mostrarà les dues ultimes línies de l'exemple: la image (width i height).

La VAPIX proporciona la sintaxis URL per enviar les següents comandes:

- **Parameter management:** La majoria de les característiques dels productes de vídeo en xarxa d'Axis es poden configurar mitjançant paràmetres de la càmera.
- **Agregar, modificar i esborrar usuaris:** Afegir un nou usuari amb contrasenya i la pertinença a grups, modificar la informació i eliminar un usuari.
- **Factory default:** Tots els paràmetres, excepte Network.BootProto, Network.IPAddress, Network.SubnetMask, Network.Broadcast i Network.DefaultRouter s'estableixen en els seus valors per defecte de fàbrica.
- **Hard factory default:** Tots els paràmetres s'ajusten al seu valor predeterminat de fàbrica.
- **Firmware upgrade:** Actualitza la versió del firmware.
- **Restart server:** Reiniciar la càmera.
- **Server report:** Aquesta petició CGI genera i retorna un fitxer d'informe. L'informe inclou informació del producte, configuració dels paràmetres i els registres del sistema.
- **Logs:** Recuperar la informació de registre del sistema.
- **System date and time:** Peticions per rebre l'actual data i hora de la càmera o per configurar les mateixes.
- **Image size:** Retorna la mida de la imatge predeterminada.
- **Video status:** Comprovar l'estat d'una o més fonts de vídeo.
- **Bitmap:** Suport per a imatges de mapa de bits.
- **JPEG/MJPG:** Suport per a imatges JPEG i codificació MJPEG.
- **Dynamic text overlay:** Configurar superposició de text dinàmic a la imatge.
- **PTZ driver update:** Instal·lació i eliminació de controladors PTZ.
- **PTZ administration:** Associar ports series amb controls PTZ de la càmera.
- **PTZ control**: Controlar el comportament de gir, inclinació i zoom d'una unitat PTZ.
- **PTZ configuration:** Establir i configurar posicions predefinides PTZ.
- **Set PTZ parameters:** Establir els paràmetres del PTZ.
- **PTZ control queue:** Sol·licituds referents a la cua de control PTZ.
- **Motion Detection Level:** Retorna el nivell actual en el que la detecció de moviments esdevé e un esdeveniment.
- **I/O ports:** Recuperar informació sobre l'estat dels ports.
- **Serial port control:** Control del port sèrie.
- **Open serial port:** Obrir el port sèrie utilitzant el protocol HTTP.
- **IP address filter administration:** Permetre o denegar les adreces IP de la llista per accedir al dispositiu d'Axis.
- **Audio data request:** Suport per a la sol·licitud de gravacions d'àudio.

5 API's

API es una abreviatura en angles que vol dir Application Programming Interfaces (Interfície de programació d'aplicacions) i es un conjunt de comandes, funcions i protocols informàtics que permeten crear programes específics per a certs sistemes operatius. Les API's simplifiquen en gran mesura el treball d'un creador de programes ja que no s'ha de picar codi des de cero. Aquestes permeten utilitzar funcions predefinides per interactuar amb el sistema operatiu o amb altres programes o dispositius.

Les tres API's anteriors que s'han aprofitat per desenvolupar aquest projecte són:

- API Robotino implementat per Robotinoapi1 i Robotinoapi2.
- Apiimatge
- Apicamera

API Robotino és de l'assignatura PSCTR[7] i les dues segones del segon[2]. S'ha creat una més per als algorismes de pathfinding.

5.1 API DEL ROBOTINO PER QNX.

La API del Robotino està dissenyada per treballar amb el sistema operatiu de temps real QNX i ens proporciona dues maneres de rebre informació del robot, que són les següents:

- **Funcions**: Són les que permetran a l'usuari interactuar amb els diversos components del robot (sensors de distancia, entrades analògiques, bumper , etc.) i així realitzar gran quantitat d'aplicacions diferents.

- **Esdeveniments**: Són enviats automàticament a la cua generada per la pròpia API. És un mecanisme opcional.

A continuació s'expliquen breument les funcions que disposa aquesta API. Com ja s'ha aclarit en els antecedents, les funcions van ser creades en el projecte "Estudi d'adaptació a QNX de la plataforma didàctica de robòtica mòbil Robotino" del primer projectista. Per a més informació, es pot consultar la seva memòria on s'adjunta una descripció més detallada.
Les funcions s'han estructurat en 3 blocs per tal de seguir un ordre en la seva declaració. Aquests blocs són:

- Funcions per controlar el sistema:

    - ***int initSystem*** **(char** * nomFitxer**, float** periodeL**, struct var_esdev** * init**)**: S'encarrega d'inicialitzar tot el sistema a partir dels paràmetres de inicialització introduïdes per l'usuari.
    - ***int closeSystem*** **():** Aquesta funció s'encarregarà de tancar tot el sistema.

- Funcions de consulta/modificació d'algun component del robot:

  - ***int getFirmwareVersion* (unsigned char** *valor**, unsigned char** *valor_1**, unsigned char** *valor_2**)**: Permet obtenir informació sobre el firmware de la placa de control de entrades i sortides.

  - **int *getAnalogInputValue* (int** canal**, int** *valor**)**: Permet obtenir el valor actual, en V, adquirit pel canal d'entrada analògica especificat.

  - ***int getBumperValue* (BOOL** *valor**)**: Permet obtenir l'estat del para-xocs.

  - ***int getDigitalInputValue* (int** canal**, int** *valor**)**: Permet obtenir l'estat actual en binari del canal d'entrada digital especificat.

  - ***int getDistanceSensorValue* (int** sensor**, float** *valor**)**: Permet obtenir el voltatge actual del sensor de proximitat especificat en V.

  - ***int getMotorVelocity* (int** motor**, float** *valor**)**: Permet obtenir la velocitat actual del motor especificat en revolucions per minut (rpm).

  - ***int getMotorPosition* (int** motor**, int** *valor**)**: Permet obtenir la posició actual, en binari (4 bytes), del motor especificat.

  - ***int getMotorCurrent* (int** motor**, float** *valor**)**: Permet obtenir el corrent, en A, que consumeix el motor especificat.

  - ***int setMotorVelocity* (int** motor**, float** speed**)**: Permet establir la consigna de velocitat, en revolucions per minut (rpm), pels motor del robot especificat.

  - ***int resetMotorPosition* (int** motor**)**: Reinicialitza el comptador de posició del motor especificat.

  - ***int resetMotorTimer* (int** motor**)**: Reinicialitza el comptador de temps del motor especificat.

  - ***int setPIDMotor* (int** motor**, unsigned char** kp_i**, unsigned char** ki_i**, unsigned char** kd_i**)**: Estableix els paràmetres Kp , Ki i Kd del controlador PID del motor especificat.

  - ***int getMotorRawCurrent* (int** motor**, unsigned short** *valor**)**: Permet obtenir el corrent, en format binari (10 bits), que consumeix el motor especificat.

  - ***int getBatteryCurrent* (float** *valor**)**: Permet obtenir el corrent actual, en A , que consumeixen les bateries.

  - ***int getBatteryVoltage* (float** *valor**)**: Permet obtenir el voltatge actual, en V , de les bateries.

- ***int setDigitalOutputValue*** **(int** canal**, int** valor**)**: Permet establir l'estat binari desitjat del canal de sortida digital especificat.

- ***int setRelayValue*** **(int** rele**, int** valor**)**: Permet establir l'estat binari desitjat del relé especificat.

- Funcions de consulta/modificació de l'estat del robot:

  - ***int getRobotVelocity*** **(float** *vx**, float** *vy**, float** *omega**)**: Permet obtenir la velocitat del robot, en mm/s i °/s, referenciada al sistema de coordenades robot.

  - ***void setRobotVelocity*** **(float** vx, **float** vy, **float** omega**)**: Permet establir la consigna de velocitat del robot, en mm/s i °/s, referenciada al sistema de coordenades robot.

  - ***void projectVelocityRobot*** **(float** *m0**, float** *m1**, float** *m2**, float** vx**, float** vy**, float** omega**)** : Permet transformar les velocitats del robot referenciades al sistema de coordenades robot a les velocitats angulars dels motors en revolucions per minut (rpm).

  - ***void unprojectVelocityRobot*** **(float** *vx**, float** *vy**, float** *omega**, float** m0**, float** m1**, float** m2**)** : Permet transformar les velocitats angulars dels motors (rpm) en les velocitats del robot referenciades al sistema de coordenades robot.

  - ***int getOdometry*** **(float** *posX**, float** *posY**, float** *phi**)**: Permet determinar la posició actual del robot, en mm i ° , referenciada al sistema de coordenades món.

  - ***void setOdometry*** **(float** posX**, float** posY**, float** phi**)**: Permet inicialitzar el sistema de coordenades món imposant la odometría actual del robot, en mm i ° .

5.2 LLIBRERIA PER A LA CAMERA: APICAMERA

Tot i què teòricament aquest apartat era una recuperació de les llibreries realitzades pel segon projectista, el tercer va tenir que fer modificacions en algunes funcions i altres las va tenir que re-fer ja que al utilitzar-les es va donar compte que donaven errors .
Apart d'aquests errors de compilació, el programa es va tenir que adaptar a una nova càmera perquè la anterior tenia un temps de captura d'imatge alt i condicionava a l'hora de fer el processat i el seguiment de línia.
Aquesta llibreria implementa la comunicació amb la càmera fent servir el protocol HTTP. Per obtenir més informació relacionada amb les funcions de la llibreria per la càmera es pot revisar la memòria del 3r projectista (Alejandro)[3] i també al seu annex on s'adjunten les taules elaborades pel 2r projectista (David)[2] i s'explica cada una d'elles de forma molt intuïtiva i les modificacions fetes.

5.2.1 GETJPG()

```
int getJPG(char ip[],imgmem_t *cammem,imageRes_t resolucio){}
```

Aquesta funció serveix per a capturar una imatge actual de la càmera en format JPG i guardar-la en la memòria. El funcionament es senzill: primer s'estableix la comunicació amb la càmera amb la IP proporcionada, després es fa la petició d'imatge segons la resolució indicada al cridar la funció i finalment emmagatzema les dades en la variable de tipus imgmem_t. La resolució de la imatge pot ser:

- LOW: 160x120
- MEDIUM: 320x240
- HIGH: 640x480

Per a les dues primeres resolucions son suficients 5 Bytes per guardar la informació, però amb resolució alta fan falta 6 Bytes.

5.2.2 GETBMP()

Fa lo mateix que la funció anterior però amb la imatge en format BMP:

```
int getBMP(char ip[],imgmem_t *cammem,imageRes_t resolució){}
```

5.2.3 *ALTRES FUNCIONS*

La resta de funcions son complementaries, es a dir, no s'han arribat a utilitzar per la aplicació però son funcions útils. Totes elles van ser creades pel segon projectista i verificades pel tercer:

- RestartCamera():Funció per a reiniciar la càmera.
- SetDate():Funció per a configurar la data i hora de la càmera segons uns valors estipulats.
- GetDate(): Funció per adquirir la data i hora actual de la càmera.
- CameraON(): funció per l'obertura del canal de comunicació entre el programa I la càmera.

5.3 LLIBRERIA PER PROCESSAT D'IMATGES: APIIMATGE

Aquest apartat tracta bàsicament sobre com manipular les imatges rebudes de la càmera per tal de que el programa sigui capaç d'interpretar cert tipus d'informació i pugui utilitzar-la per a que el robot sigui capaç d'interactuar amb el món real.

Per tal de poder manipular les imatges primer hem d'entendre com s'emmagatzema la informació segons el format d'aquesta i ser conscients de que moltes vegades no obtindrem la imatge tal i com la necessitem (o volem). Ja sigui per la qualitat de la càmera, la intensitat de la llum, o milers de causes possibles existeix un soroll que pot causar una imatge difusa i, per això, es necessari fer el posterior processat de la imatge.

Una imatge digitalitzada es pot considerar monocromàtica, es a dir, que es susceptible de ser tractada com una funció f(x,y) on 'x' i 'y' denotaran coordenades especials. Per poder tractar-la primer hem d'identificar el format amb el que estem treballat. En el cas de la nostra càmera tenim els següents:

- **BMP**: O més conegut com Bitmap, és un dels mes simples i va ser desenvolupat per Microsoft i IBM. Es un arxiu de mapa de bits, es a dir, un arxiu amb píxels emmagatzemats en forma de taula de punts que administra els colors. La seva estructura consta d'una capçalera i el cos de la imatge. Els primers 53 bits corresponen a la capçalera i la imatge comença a llegir-se des de baix cap a dalt (píxel inferior esquerre fins superior dret).

- **JPG**: És el format d'imatge mes comú utilitzat per càmeres digitals i altres dispositius fotogràfics. Utilitza un algoritme de compressió amb pèrdua per reduir la mida dels arxius lo que fa que al re-obrir l'arxiu no s'obté exactament la mateixa imatge que abans de comprimir-la. Hi han variacions de l'estàndard que comprimeixen la imatge sense pèrdua de dades com '*JPEG 2000*' o '*JPEG-LS*'.

Tot i què es va buscar informació de com tractar les imatges JPG, es va arribar a la conclusió de que era molt difícil i es va decidir treballar amb BMP ja que es un format molt fàcil d'entendre i per conseqüent d'interpretar.
Les imatges en JPG s'utilitzaran per fer la calibratge ja que dona millors resultats que amb BMP.

Per obtenir informació mes detallada d'aquest formats es pot revisar les pàgines 44-45 de la memòria del 3r projecte[3].

5.3.1 FUNCIONS DE PROJECTISTES ANTERIORS

Com diu el títol, aquesta serà una recopilació de funcions de processat d'imatge heretades d'anteriors projectistes. Només es citaran les utilitzades o més interesants. Per veure les demés es pot revisar ambdós projectes anteriors, però es recomana el tercer[3] ja que en el segon[2] hi havien errors i es van fer modificacions.

Per mostrar els diferents efectes de les diferents funcions de processat utilitzarem la mateixa imatge:

Figura 5.1: Imatge capturada per la càmera AXIS en format .bmp

Per capturar-la s'ha invocat la funció getBMP() però també hi ha la possibilitat de fer-ho des de el navegador web amb la següent adreça:
http://192.168.1.114/axis-cgi/bitmap/image.cgi?resolution=480x360

5.3.1.1 Binaritzar imatge *(im2bw)*.

La funció im2bw serveix per a binaritzar tant una imatge RGB o YCbCr. EL seu funcionament es senzill ja que l'únic que fa es agafar un llindar (establert quan s'invoca la funció) i transforma els píxels amb valor superior en blanc, i en negre els que tenen valor inferior. La nova imatge pot ser emmagatzemada com a nova o reemplaçant la original.

Figura 5.2: Resultat de la funció im2bw amb llindar de 70 vs fent capaCR abans

5.3.1.2 Conversió de rgb a ycbcr *(rgb2ycbcr)*.

El YCbCr es una família d'espais de colors utilitzada en sistemes de vídeo i fotografia digital, i és una forma de codificar informació RGB. Les components que formen aquest format d'imatge són:

- Y: Component de lluminositat.
- C_B: Croma blau.
- C_R: Croma vermell

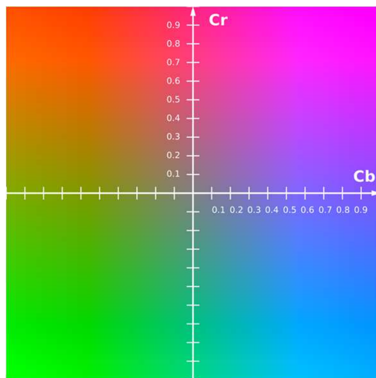


Figura 5.3: Pla $_{CB-CR}$ amb luminància constant de Y'=0.5

Figura 5.4: Funció aplicada a la imatge original.

El color que es mostra depèn de la combinació de colors primaris RGB utilitzats per mostrar la senyal. Per tant, un valor expressat com Y'CBCR és predictible només si s'utilitza la cromaticitat dels colors del estàndard RGB. Normalment, per a processats d'imatge, el que es fa és separar aquestes capes i seleccionar les convenients descartant les altres.

5.3.1.3 Separar capes ycbcr *(capaY, capaCb, capaCr)*.

El procediment és el mateix en les tres funcions però canviant la posició del píxel en que esta dipositat cada un dels colors. Bàsicament el que es fa es fer la conversió de RGB a YCbCr i després separar les capes conservant només una de les tres.

- capaY:

Figura 5.5: Comparació fotografia original vs amb filtre capaY.

- capaCb:

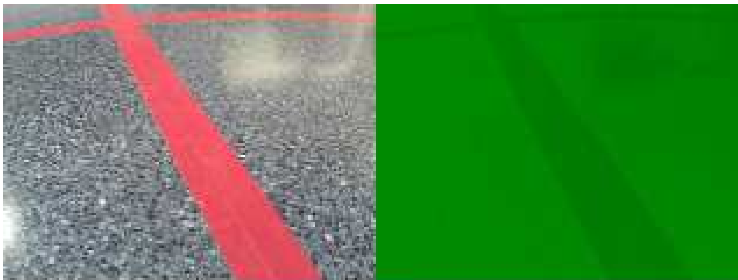


Figura 5.6: Fotografia original vs amb filtre capaCB.

- capaCr:

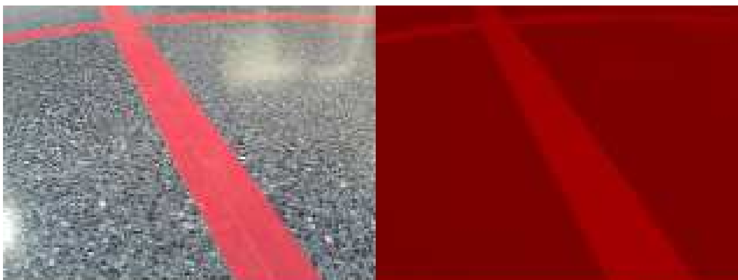


Figura 5.7: Fotografia original vs amb filtre capaCR.

5.3.1.4 Canviar contrast *(contrast).*

Modificar el contrast d'una imatge consisteix en augmentar o disminuir la pendent que formen els valors dels píxels de la imatge, vigilant sempre de no sobrepassar els límits (0 i 255).

Figura 5.8: Foto original i contrastada amb valor 60.

5.3.1.5 Separar capes *(capaR, capaG, capaB).*

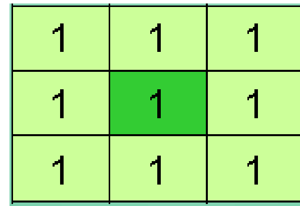
Es tracta de lo mateix que separar les capes YCbCr però sense fer la conversió des de RGB.

Figura 5.9.

Aquí podem apreciar bé que la capa que millors defineix la línia es la de roigs i no es casualitat ja que la línia es vermella.

5.3.1.6 Erosionar imatge *(erode)*.

L'erosió és una de les dues operacions morfològiques bàsiques del processat d'imatges i es basa en la convolució d'una màscara amb tots els píxels d'una imatge.



1	1	1
1	1	1
1	1	1

Figura 5.10: Il·lustració mascara 3x3

En aquesta API la màscara escollida es de 3x3: el píxel central es compara amb tots els que té al voltant i, si tots els píxels són negres, aquest es deixa negre. En canvi, si hi ha algun píxel que és blanc el píxel central també ho serà. Podríem dir que és una condició AND entre els píxel que entren a la màscara.

Partint d'una imatge bineritzada per la funció del punt 5.3.1.1., al aplicar el filtre d'erosió obtenim això:

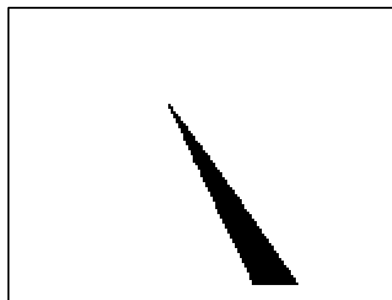


Figura 5.11.

Si ens fixem veiem que ha desaparegut la línia horitzontal i la vertical s'ha aprimat. Això es per que aquesta funció utilitza una mascara per fer la erosió verticalment. Si volguéssim conservat la línia horitzontal s'hauria de canviar la mascara per tal de fer la erosió horitzontal.

5.3.1.7 Dilatar imatge *(dilate).*

Podem dir que es la operació contraria a la anterior. També treballa amb mascara i per a aquesta api utilitza també una 3x3.

El píxel el qual es vol dilatar es compara amb tots els que te al voltant. Si hi ha algun píxel que sigui negre el píxel central serà negre, si no serà blanc. Podríem dir que es una condició OR entre els píxel que entren a la màscara.

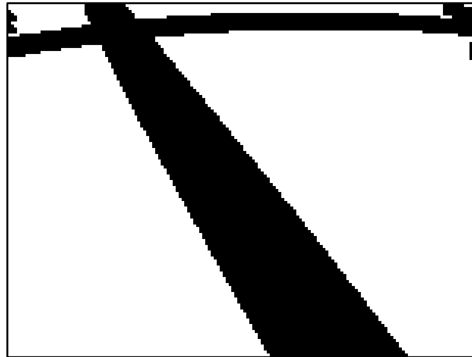


Figura 5.12.

5.3.1.8 Detecció de la línia d'una imatge bineritzada *(detectSegment).*

Amb aquesta funció el que es pretenia era obtenir la posició de la línia situada en la imatge identificant dos punts d'ella. Per fer-ho, es mesurava el primer píxel a la altura 0, es a dir, la primera fila de píxels de la imatge i la segona a la altura 20 però, en comptes de passat directament a la segona altura, la funció recorria tota la imatge, píxel per píxel, fins arribar-hi. Creiem que això comportava una pèrdua de temps innecessària i molt valuosa i que es podria reduir el temps d'execució modificant-la. Això es el que fem en la funció següent.

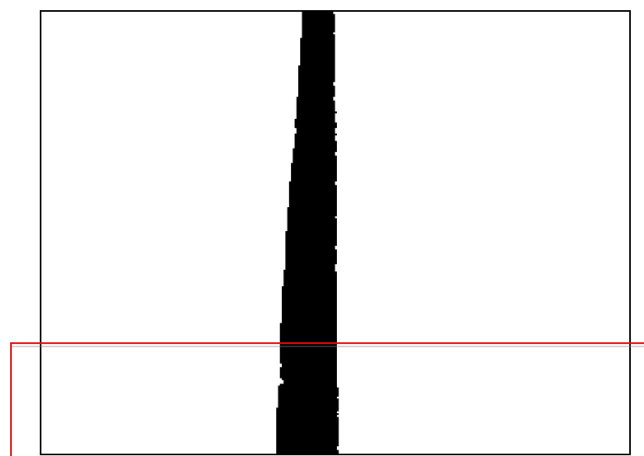


Figura 5.13: Requadre inspeccionat per obtenir la posició dels dos píxels.

5.3.2 FUNCIONS PRÒPIES

5.3.2.1 Detecció de línia a partir de dos punts *(detectPuntos).*

Bàsicament la funcionalitat es la mateixa que la de la funció *detectSegment* de l'anterior projectista però, com aquesta no ens donava els valors que esperàvem ni ens convencia el seu mode de funcionament, s'havia de modificar quasi completament. Així que es va decidir crear una pròpia i deixar-la tal qual per si algun futur projectista la vol utilitzar o aprofitar com a base per a crear una altra funció.

La principal diferencia esta en que en la nostra funció es pot decidir a quina altura fer la mesura del punt i també que compta tots els píxels seguits del mateix color per agafar realment el punt central. En la funció *detectSegment* es compten 20 i es selecciona el numero 10, sense saber si la recta mesura més o menys. Això era acceptable per l'anterior projecte ja que la càmera estava paral·lela al terra i la línia havia de mesurar teòricament el mateix en tota la imatge però, per aquest projecte com es veu amb perspectiva, aquesta generalització no en val.

Es parteix d'una imatge bineritzada en la qual, després de ser tractada, se'ns queda la línia ben definida sense 'brutícia' al costat. Com treballarem amb imatges bmp, aquesta funció esta creada per treballar amb aquest format.

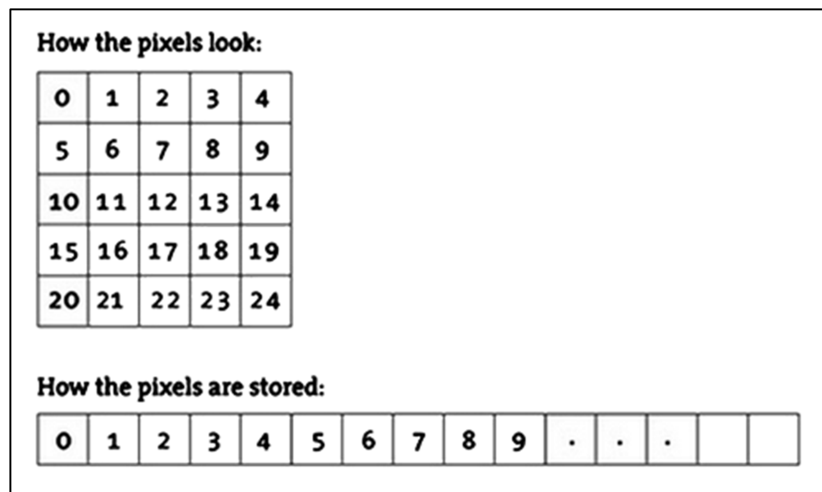


Figura 5.14.

Una imatge bmp emmagatzema la informació en forma de bytes. Els 54 primers (0-53) formen la capçalera. A partir del 54 tenim, de 3 en 3, els bytes amb la informació de color de cada píxel. Els tres bytes estan ordenats com B, G i R.

Normalment els píxels estan guardats de baix a dalt, començant en el cantó inferior esquerre, d'esquerre a dreta, fila per fila, en ordre creixent. Però també es possible emmagatzemar-los de dalt a baix quan el valor de la altura de la imatge es negativa. A la *figura 5.14* tenim representat aquest cas.

La resolució amb la que treballem es la LOW (160x120). 120 en eix Y 160 en eix X (coordenades imatge).

- Funcionament:

Per utilitzar-la primer s'invoca la funció passant els paràmetres necessaris:

- Imatge bineritzada.
- Altures a les que volem que es mesurin els punts (y_ini, y_fin).
- Variables en les que volem guardar els punts obtinguts (x1,y1) i (x2,y2).

```
int detectPuntos(imgmem_t *imgmem,int y_ini, int y_fin,int *x1,int *y1, int *x2, int *y2)
{
```

Desprès de la declaració de variables tenim dos bucles *for*, un per a cada punt:

```
for ( x =0; x <160; x++) {
    z= 54+ 3*(y*160+x); //160 son el num de pixeles en el eje x y se multiplica por 3 porque cada pixel tiene 3 bytes
                        // la imatge comença al bit 54. tot lo anterior forma part de la capçalera
    if (imgmem->buffer[z] == color){ // detecta el primer pixel del color

        contador=0;
        h=0;
        do{
            contador ++; // conta el n° de pixels seguits del mateix color

            imgmem->buffer[z+h+2]=0xFF;
            imgmem->buffer[z+h+1]=0xA5;
            imgmem->buffer[z+h]=0x00;

            h=h+3;
        } while ( imgmem->buffer[z+h] == color);

        *x1=x + contador/2; // punto medio de la linea.
        *y1 = y;
        break; // per a que surti del bucle una vegada trobat el punt
    }

}
```

La condició del *for* es per fer el punter recórrer la imatge de forma horitzontal, des del primer píxel fins a l'últim a la primera altura introduïda per l'usuari (y_ini).

Com s'ha dit abans, la informació de cada píxel esta emmagatzemada en 3 bytes i, com la imatge esta bineritzada, nomes trobarem valors per píxels negres (000000) o blancs (FFFFFF). Llavors, comprovant només un dels bytes es suficient. Hem escollit comprovar el 3r que es el de major pes. D'aquesta forma, quan el byte seleccionat valgui un 0, es tractarà d'un píxel negre i si es un 1 serà un blanc. Per recórrer els bytes utilitzem la variable '*z*'.

Així doncs, es fa augmentar el valor de la variable *z* per recórrer la imatge en sentit horitzontal fins a trobar el primer píxel del color desitjat (en aquest cas negre). Una vegada s'ha trobat, es valida la condició del *if* i la funció comença a contar quants píxels negres consecutius hi ha fins a trobar el primer píxel blanc.

Per a poder observar a la imatge guardada la altura del punt seleccionat, es canvia el color de tots els píxels comptats. Això no es necessari però serveix de gran ajuda. Els tres bytes estan ordenats com B, G i R, i si volem posar un píxel en taronja (FFA500) haurem de posar B1=0x00, B2=0xA5 i B3=0xFF.

Quan s'acaba de comptar, es divideix el valor obtingut entre dos i es suma al valor de la posició en X del primer píxel trobat per seleccionar el punt central de la línia. Finalment es guarden aquests valors X i Y en les variables seleccionades per l'usuari i es procedeix a fer lo mateix per al segon punt.

```
    if (*y1==0 || *y2==0) ret=0;

    return ret;
}
```

Les variables y1 i y2 s'inicialitzen amb valor 0. Si, pel que sigui, no es detecta cap píxel negre i, per conseqüent, no obtenim un píxel central, aquestes variables romanen amb el mateix valor. Al detectar-se això en la condició del *if* final es retorna un 0 per indicar un error.

5.3.2.2 Transformació de coordenades píxel a robot *(PimgToRobot).*

Una vegada tenim els dos punts per identificar la recta, s'ha de calcular la seva equació. Però, per fer-ho s'han de traduir les seves posicions a coordenades robot i així saber les seves posicions reals respecte al centre del robot. Aquí es on entra en joc el paper del calibratge, que s'explicarà en el següent capítol.

Les dades d'entrada son les coordenades píxel (x,y) obtingudes de la funció anterior i les de sortida (xr,yr) referenciades al sistema de coordenades robot. La transformació es fa només per un punt així que s'haurà d'invocar la funció dos cops.

```
int PimgToRobot(int *x, int *y,int *xr,int *yr)
{
```

Per entendre bé el procediment, llegir el capítol 8.2 on s'explica detalladament el seu funcionament.

6 CALIBRATGE DE LA CÀMERA

6.1 QUE ES LA PERSPECTIVA

La perspectiva es un fenomen que afecta a la visió d'objectes o espais tridimensionals. Quan una persona mira per exemple un carrer, encara que aquest sigui perfectament recte, si mira al fons el veurà inclinat cap al centre i cada vegada més estret. Si es fixa en els objectes que hi han, com més lluny es troben més petits es veuen. Això pot portar a confusions varies com per exemple no saber quin objecte esta més a prop o quin és mes gran o petit.

De la mateixa manera, quan una càmera fa una foto, també la fa mantenint aquesta perspectiva. Això vol dir que la distancia entre dos píxels no es tradueix mai de la mateixa forma al mon real i depèn de varis factors, entre els quals es troben la distancia focal de la càmera o la altura de la imatge a la que es faci la mesura.

Figura 6.1: Imatges amb perspectiva

Per això, es necessari calibrar la càmera per poder mesurar les proporcions reals entre el píxels i la representació real d'aquests en el món real i així poder discernir les distancies i grandàries reals dels objectes capturats.

6.2 QUE ÉS EL CALIBRATGE

De forma resumida, quan es calibra el que es fa es establir unes relacions entre dades conegudes i les incerteses de valors que volem conèixer.
En el nostre cas, seguint el mètode utilitzat en la assignatura de Control i Guiatge de Robots Mòbils[E], utilitzarem els paràmetres intrínsecs, extrínsecs i possibles distorsions de la càmera per poder establir aquestes relacions. Els valors intrínsecs son valors que reflecteixen característiques pròpies de la càmera com la distancia focal, error de píxel, etc. Els extrínsecs son els que tenen a veure amb la posició de la càmera respecte a la imatge que veu i al món real. Després de calibrar només es farà ús dels paràmetres extrínsecs per poder establir relacions entre diferents sistemes de referència però, com aquests no es poden calcular sense els intrínsecs, primer s'hauran de calcular aquests últims.

En primer lloc, abans de fer el calibratge és important conèixer quines transformacions (rotacions i translacions) s'han de realitzar per tal de poder passar d'un punt expressat en coordenades del sistema de referència del robot o món a expressar-lo en coordenades de la imatge. A continuació, s'ha de resoldre el problema invers; saber quines transformacions s'han de realitzar per saber quin punt referenciat a l'espai del Robotino representa un píxel de la imatge.
Durant el procés de calibratge del robot, tenen lloc cinc sistemes diferents de referència:

1. Sistema de referencia coordenades del robot.
2. Sistema de referencia coordenades món
3. Sistema de referència de calibratge.
4. Sistema de referencia de la càmera .
5. Sistema de referencia de la imatge 2D.

D'aquest 5 només s'utilitzen 4 per fer el calibratge, depenent de si al final volem passar de coordenades píxels a coordenades món o robot.

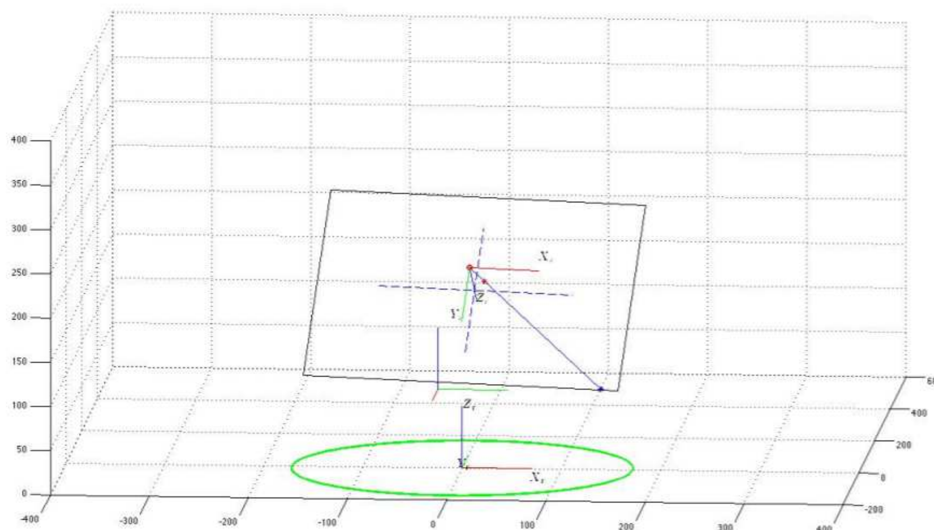


Figura 6.2: Representació dels diferents sistemes de referencia del Robotino. El 5é (c.c.món) no apareix però es com el del robot rotat 90º respecte Z.

Per calibrar la càmera es necessita un patró o escaquer extern de 2D per calcular els paràmetres intrínsecs i la distorsió de la lent. Per fer els càlculs s'utilitzarà la toolbox de Matlab per calibrar.

Albert Masip-Àlvarez

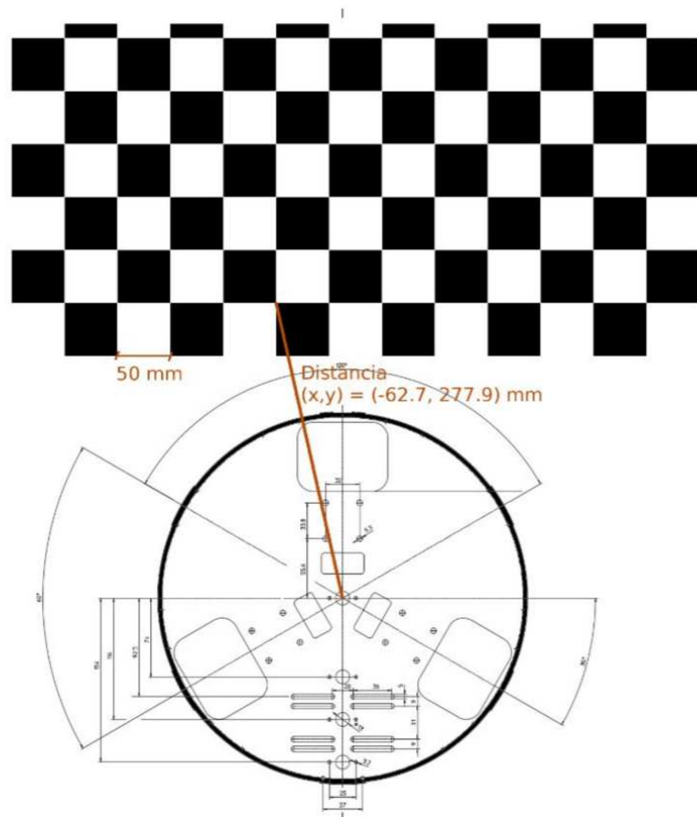


Figura 6.3: Patró per calibrar la càmera del Robotino.

6.2.1 PARÀMETRES INTRÍNSECS DE LA CÀMERA

Com s'ha dit abans, els paràmetres intrínsecs son aquells que defineixen la geometrıa interna i la òptica de la càmera. Aquest paràmetres son sempre constants mentre que no es produeixi un canvi en la orientació del centre òptic de la càmera i venen expressats en la matriu de paràmetres intrínsecs que rep el nom de KK.

$$KK = \begin{bmatrix} fc(1) & \alpha c * fc(1) & cc \\ 0 & fc(2) & cc \\ 0 & 0 & 1 \end{bmatrix}$$

NOTA: Aquesta matriu quadrada és ortogonal així que el producte d'ella mateixa per la seva inversa dona la matriu identitat.

On:

- f_c és la longitud o distància focal expressada en píxels i emmagatzemada en un vector de dues components (x,y).
- c_c són les coordenades x,y del centre òptic o punt principal.
- α_c és el coeficient d'inclinació que correspon a l'angle format entre els eixos x i y dels píxels.

Per obtenir aquests paràmetres es necessari fer ús de l'escaquer mencionat abans i capturar imatges amb la càmera del Robotino en diferents posicions. Es imprescindible conèixer les mides dels quadrats i la distancia a la que es troben del centre de coordenades del Robot (i/o de la càmera).

A la hora de carregar la imatge es pot fer de varies formes; fent la petició URL a la càmera, com s'ha explicat en un apartat anterior, o fent la captura directament des de Matlab. Al fer-ho des de Matlab es important utilitzar la funció imwrite/imread per evitar veure els eixos generats de forma automàtica a la hora de guardar la imatge.

6.2.2 PARÀMETRES EXTRÍNSECS DE LA CÀMERA

Els paràmetres extrínsecs de la càmera son aquells que relacionen els sistemes de referencia del mon real i de la càmera, descrivint la posició i orientació de la càmera en el sistema de referència de coordenades del món real. També es treballa amb l'escaquer per obtenir aquestes dades.

- Vector de Translació **T**: Aquest vector serveix per determinar la ubicació del centre òptic de la càmera respecte als eixos de coordenades del món real (x,y,z). Al fer el calibratge s'obté com a Tc_ext.

- Matriu de rotació **R**: Es una matriu que relaciona la rotació de la posició de la càmera respecte als eixos de coordenades del món real. S'obté amb el nom de Rc_ext.

Generalment, a l'hora de fer el calibratge s'obtenen més dades com per exemple l'error de píxel o el vector de rotació. Aquest últim no ens és útil, però l'error de píxel sí, ja que ens interessa que el seu valor sigui el mínim possible. D'aquesta forma, l'utilitzem com a indicador de la qualitat del calibratge. Si veiem que tenim un error de píxel superior a 1, es recomanable tornar a fer el calibratge i fer la tria de punts més acuradament per tal de reduir aquest valor.

Després de la obtenció d'aquests valors, Matlab ens ofereix la possibilitat de representar la posició de la càmera respecte a l'escaquer per veure si realment es correspon a la realitat. En el nostre cas el resultat ha sigut bastant acurat.

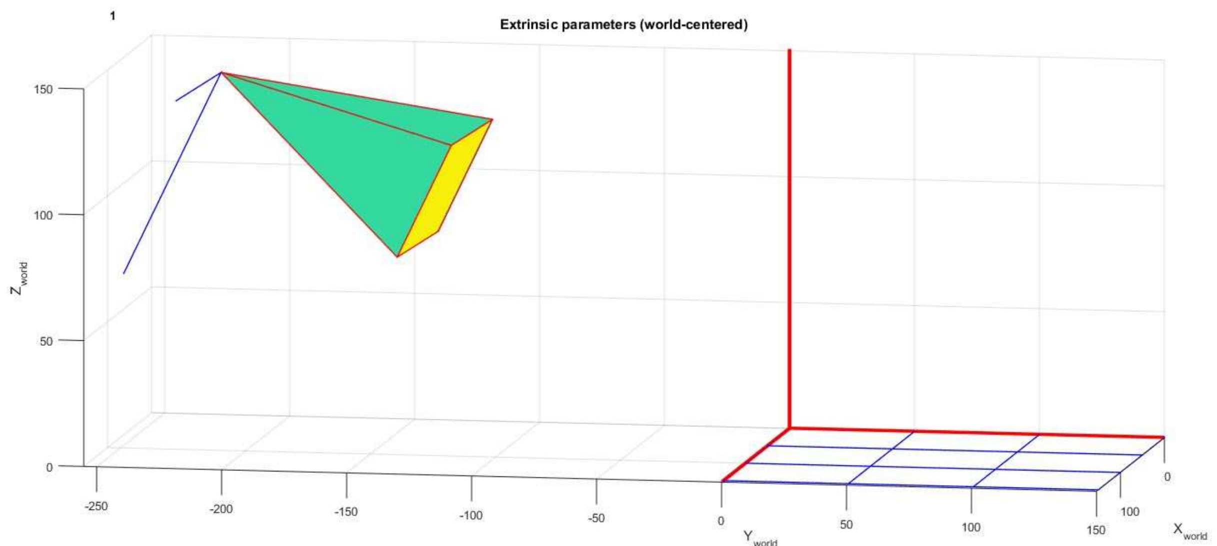


Figura 6.4: Representació de la posició de la càmera segons els paràmetres extrínsecs obtinguts en el calibratge.

NOTA: Aquest calibratge es va fer sense rotació d'eixos i assignant com a centre de coordenades de calibratge el punt (-62.7,277.9) referenciat al centre del robot.

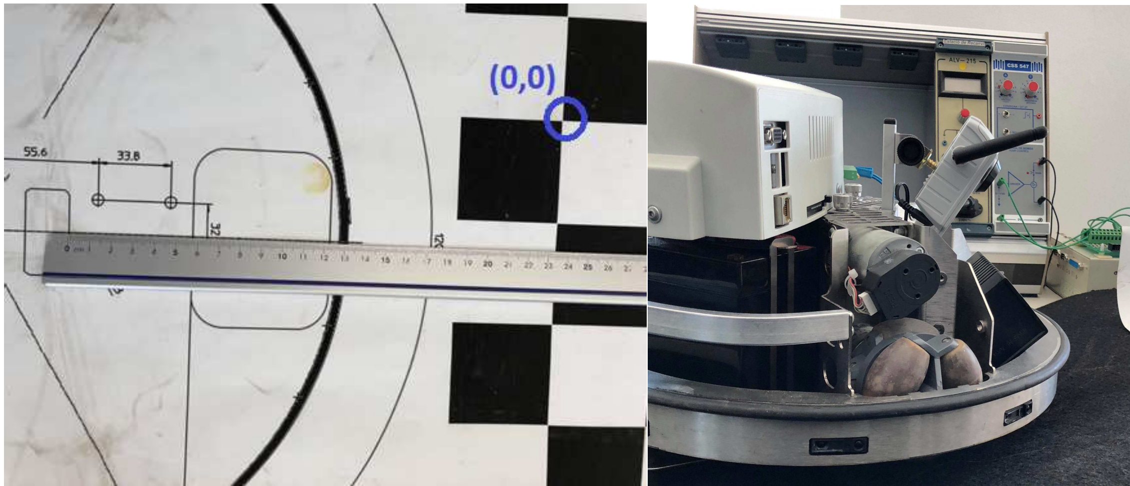


Figura 6.5: Mesura de la distancia del suport de la càmera a l'origen indicat a l'escaquer.

6.3 PASSOS PER FER EL CALIBRATGE

Sabem que els punts de l'espai els podem representar en diferents sistemes de coordenades i tots ells tenen una relació entre ells. Aquesta relació es pot descriure en forma de rotació i/o translació.

Principalment el que ens interessa es poder identificar a quin punt referenciat al sistema de coordenades robot equival un punt seleccionat de la imatge. Però, per poder fer-ho, primer hem d'aconseguir fer el pas invers.

6.3.1 Coordenades robot a coordenades de calibratge.

Per poder passar punts de coordenades robot a coordenades de calibratge es necessita fer una rotació i translació d'eixos respecte a l'eix Z. Per fer-ho existeixen matrius de translació i de rotació, i en coordenades homogènies són les següents:

$$T = \begin{bmatrix} 1 & 0 & 0 & -tx \\ 0 & 1 & 0 & -ty \\ 0 & 0 & 1 & -tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Per poder fer la operació cal que totes les matrius i vectors estiguin expressats en coordenades homogènies. Així les dimensions seran les mateixes i evitarem problemes de incompatibilitat.

Primer de tot s'ha de decidir quin serà el nostre centre de coordenades per al calibratge i mesurar la distancia al centre del Robotino. Es recomana que sigui un punt fàcil de veure, com per exemple el cantó d'algun quadrat, ja que després s'haurà d'identificar al fer el calibratge per Matlab.

Per al patró mostrat anteriorment, es va seleccionar com centre de coordenades el punt (x,y) = (-62.7,277.9) mm, però a l'assignatura '*Control i Guiatge de Robots Mòbils*' es recomanava el punt (x,y) = (-62.7,477.9) mm i rotat 90º ja que el sistema de coordenades robot també es troba rotat 90º respecte l'eix Z. Jo he seleccionat un punt més proper, perquè es veu més clar a l'hora de fer el calibratge (x,y) = (-62.7,427.9) mantenint la rotació de 90º.

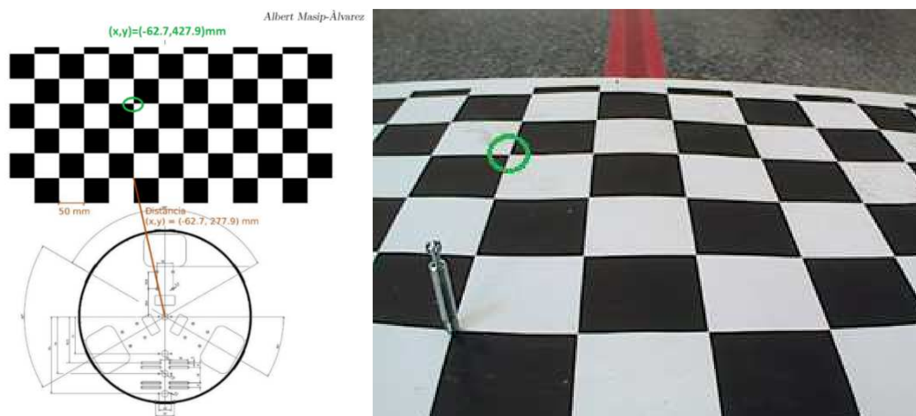


Figura 6.6: Centre de coordenades Calibratge escollit.

Quan es trasllada un punt, els signes de la matriu de translació es mantenen però, quan es traslladen els eixos, s'inverteixen els signes. Així doncs, les matrius queden de la següent forma:

$$T_{rob\text{-}calib} = \begin{bmatrix} 1 & 0 & 0 & -(-62.7) \\ 0 & 1 & 0 & -(427.9) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{rob\text{-}calib} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Caldrà aplicar primer la translació i després la rotació degut a que s'han determinat per fer-ho d'aquesta manera.

6.3.1 Coordenades robot a coordenades de la càmera

Per fer-ho primer es passa el punt seleccionat de coordenades robot a coordenades de calibratge, i després es passa a coordenades de la càmera.

La relació queda definida per la següent equació:

$$P_{cam} = T_{calib\text{-}cam} * R_{calib\text{-}cam} * R_{rob\text{-}calib} * T_{rob\text{-}calib} * P_{rob}$$

On

- P_{cam} és un punt 3D expressat en coordenades homogènies en el sistema de referència de la càmera.
- P_{rob} és un punt 3D expressat en coordenades homogènies en el sistema de referència del robot.
- $T_{calib\text{-}cam}$ és la matriu de translació Tc_ext en coordenades homogènies que se'ns proporciona al fer el calibratge.
- $R_{calib\text{-}cam}$ és la matriu de rotació Rc_ext en coordenades homogènies que se'ns proporciona al fer el calibratge.
- $T_{rob\text{-}calib}$ és la matriu de translació definida a l'apartat anterior.
- $R_{rob\text{-}calib}$ és la matriu de rotació definida a l'apartat anterior.

Les matrius necessàries per fer el canvi de coordenades Robot a calibratge ja estan definides de l'apartat anterior i es calculen manualment. Les altres dues s'obtenen després de fer el calibratge. Fent-ho hem obtingut:

```
Tc_ext =[ -75.5118;
           -26.3008;
          395.5676];

Rc_ext =  [0.0267     0.9996      0.0098;
          0.4036     -0.0018     -0.9150;
          -0.9146    0.0284      -0.4034;]
```

NOTA: Els passos per obtenir aquest valors es detallaran més endavant.

Llavors, les altres matrius queden com:

$$T_{calib\text{-}cam} = \begin{bmatrix} 1 & 0 & 0 & -75.5118 \\ 0 & 1 & 0 & -26.3008 \\ 0 & 0 & 1 & 395.5676 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{calib\text{-}cam} = \begin{bmatrix} 0.0267 & 0.9996 & 0.0098 & 0 \\ 0.4036 & -0.0018 & -0.9150 & 0 \\ -0.9146 & 0.0284 & -0.4034 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6.3.2 Coordenades robot a coordenades imatge

Per aquesta operació s'inclou al pas anterior la matriu KK, que com ja s'ha explicat abans, conté paràmetres intrínsecs, que estan relacionats amb aspectes de la lent. D'aquesta forma s'aconsegueix passar de punts referenciats a la càmera a punts referenciats a la imatge (píxels). La equació és la següent:

$$\lambda * P_{imatge} = [KK, [0, 0, 0]^T] * T_{calib\text{-}cam} * R_{calib\text{-}cam} * R_{rob\text{-}calib} * T_{rob\text{-}calib} * P_{rob}$$

D'aquí s'obté el punt 2D en píxels equivalent al punt seleccionat. Si ens fixem, a l'esquerra de la equació tenim el punt de la imatge multiplicat per una constant anomenada landa. Per obtenir el punt en coordenades imatge, cal dividir aquest valor obtingut pel valor de landa, que coincideix amb el valor de la seva tercera component.

L'origen d'eixos de la imatge correspon al vèrtex superior esquerra de la pantalla, l'eix x de la imatge és l'horitzontal (apuntant cap a la dreta en ordre creixent començant per 1) i el de les y el vertical apuntant cap avall en ordre creixent).

6.3.3 Coordenades imatge a coordenades robot

Aquesta es la operació que de veritat ens interessa. Com s'ha avançat, es tracta del pas invers a l'anterior i ens servirà per identificar a quin punts 3D de l'espai real referenciats en coordenades robot pertany un punt 2D seleccionat a la imatge.

Per simplificar el problema s'agrupen les matrius que contenen valors constants i independents del punt a explorar:

$$M = T_{calib\text{-}cam} * R_{calib\text{-}am} * R_{rob\text{-}calib} * T_{rob\text{-}calib}$$

$$M = \begin{bmatrix} R_{3\times3} & t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

Llavors podem expressar la equació de l'apartat anterior com:

$$\lambda * P_{imatge} = [KK, [0, 0, 0]T\ ] * M * P_{robot}$$

Com el que ens interessa es P_{robot}, hem d'aïllar-lo. Però, com es tracta d'una equació que conté matrius, no es pot fer de qualsevol manera. Cal multiplicar ambdós costats per la inversa de les dues matrius que volem eliminar del costat dret. La equació queda de la següent manera:

$$M^{-1} *[KK]^{-1} * \lambda * P_{imatge} = P_{robot}$$

Recordem que l'ordre de les operacions de la matriu M és fer primer una translació i després una rotació d'eixos. Si es fa el càlcul de la inversa directament, es mantindrà aquest ordre, i el que ens interessa es que també s'inverteixi. Volem que primer es faci la rotació i després la translació. Això s'aconsegueix transposant primer la part de la matriu original corresponent a la rotació i després canviant de signe la part de la matriu original corresponent al vector de translació, alhora que es pre-multiplica per la trasposta (o inversa) de la part de la matriu original corresponent a la rotació.

La nova matriu M queda com:

$$M = \begin{bmatrix} R_{3\times3} & R_{3\times3} * t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

Llavors la inversa queda com:

$$\rightarrow M^{-1} = \begin{bmatrix} R^T_{3\times3} & -R^T_{3\times3} * R_{3\times3} * t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} R^T_{3\times3} & -t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

Tornant a la equació de pas de robot a imatge i després de diversos càlculs s'obté:

$$P_{robot} = R^T * K^{-1} * \lambda * P_{imatge} - R^T * t$$

On, aïllant, landa val la 3a component del resultat de:

$$\lambda = \frac{[R^T * t]}{[R^T * K^{-1} * P_{imatge}]} \quad (3)$$

Es recorda que el punt 3D del robot no resultarà en coordenades homogènies $P_{robot} = [x_{robot}, y_{robot}, z_{robot}]^T$ però el punt 2D de la imatge sí que ho està $P_{imatge} = [x_{imatge}, y_{imatge}, 1]^T$.

7 ALGORISMES DE PATHGFINDING

La Intel·ligència artificial (IA o AI en anglès) es una branca de la informàtica que estudia procediments automatitzats per aconseguir que un autòmat pugi realitzar tasques com un esser intel·ligent. Aquests intenten imitar diverses àrees del comportament humà amb el fi d'apropar-se o fins i tot millorar a una persona normal i corrent en, per exemple, presa de decisions o diverses tasques que un sistema computacional pot fer millor o mes ràpid. Es en aquest context que la intel·ligència artificial es fa present en la robòtica, i un dels comportaments basics de l'esser humà es el desplaçament autònom d'un punt a un altre. Per resoldre aquesta problemàtica, existeix un àrea de la intel·ligència artificial anomenada Pathfinding.

El pathfinding es un terme en anglès composat per dues paraules: path i finding, i es pot traduir literalment com "busca-camins". Així doncs, com es pot preveure, el pathfinding es tracta d'un algorisme computacional que serveix per a detectar el camí mes curt o òptim entre dos punts predefinits. Els algorismes de pathfinding es fan servir per a trobar camins en els GPS o altres aplicacions de transport, i també en videojocs d'estratègia com per exemple el Pac-Man.

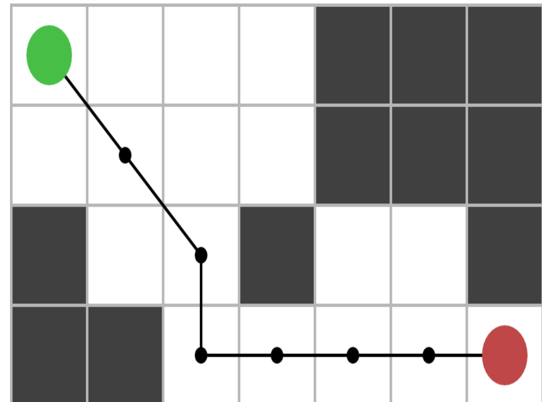


Figura 7.1: Exemple de circuit.

Hi han diferents tipus d'algorismes que s'explicaran mes endavant, però el gran repte en aquest àmbit es trobar l'algorisme més òptim en relació resultat/temps de còmput depenent de cada cas. Això es per que potser ens interessa un resultat mes precís encara que el càlcul sigui lent o, pel contrari, potser el que mes ens interessa es que el temps de còmput sigui el mínim possible encara que el camí seleccionat no sigui realment el més òptim.

7.1 DADES NECESSÀRIES

Hi han varis tipus d'algorismes però tots funcionen bàsicament igual: Es necessita tenir uns punts, uns camins i definir la relació entre ells. No hi poden haver punts sense camí, ni camins que no portin a un punt i a cada camí se li ha d'assignar un pes. Els pesos dels camins no han de ser necessàriament igual per ambdós sentits. Això es lògic ja que no costa lo mateix anar d'un punt "A" a un punt "B" per una pujada que fer el camí invers de baixada.

Una vegada definit això, s'ha de transformar aquesta informació a un llenguatge de programació de forma que sigui fàcil de processar per al programa o aplicació en la que es faci servir. Generalment es fa en forma de matrius i vectors.

7.2 DIFERENTS ALGORISMES DE PATHFINDIG

A continuació, aquí s'explicaran alguns dels algorismes de pathfinding tractats en la assignatura anterior de PSSP i altres trobats a la xarxa[4]. Els algorismes de recerca cega com el BFD (Breadth-first search) o DFS (Depth-First Search) examinen totes i cadascuna de les possibilitats: començant pel node donat, iteren sobre tots els possibles camins fins arribar al node destí (TARGET). Aquests algoritmes tenen un temps d'execució lineal i es proporcional al nombre de nodes del circuit.

Sigui com sigui, no es necessari examinar totes les possibilitats, per això algorismes com el Greedy, Dijkstra o el A* eliminen camins en la recerca del camí òptim. D'aquesta forma, aquests algorismes son capaços de disminuir significativament el temps de còmput.

Per a que sigui mes fàcil d'entendre, a continuació posaré el problema del projecte com exemple i hi aplicaré cadascun dels diferents mètodes explicats per veure el seu funcionament:

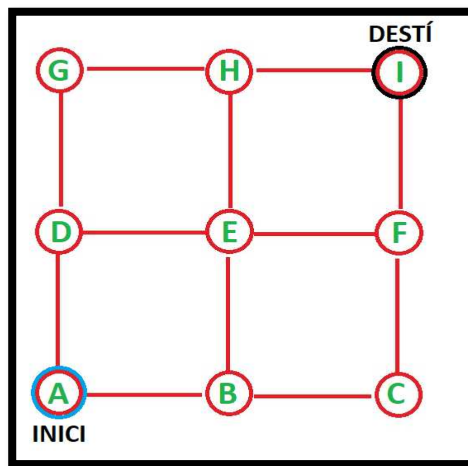


Figura 7.2: Representació del circuit

Al terra del laboratori tenim una matriu pintada com la de la imatge. Cada encreuament conté un punt (o node) i les línies entre mig son els camins. Així dons, tenim 9 punts i 12 camins diferents, que tenint tots 2 sentits cadascun ens donaran un total de 24 pesos o weightpoints.

A continuació tenim una taula amb el pes dels camins entre els nodes. En comptes de donar-li noms als camins he preferit descriure'ls en funció del punt en que comencen i el punt en el que acaben. Per a descartar les connexions inexistents entre nodes i que a l'hora de calcular la ruta l'algorisme no els tingui en compte, els hem de donar un pes amb valor infinit ja que a l'hora de comparar alternatives l'algorisme es queda amb el camí de menor pes.

Aquesta taula ens servirà per a tots els tipus d'algorismes que es tractaran però, per a l'Algorisme anomenat A Star (A*) serà insuficient ja que apart d'això també es necessita tenir l'heurística dels punts. Això s'explicarà detalladament quan es parli d'aquest algorisme.

DE/A	A	B	C	D	E	F	G	H	I
A	∞	3	∞	4	∞	∞	∞	∞	∞
B	9	∞	3	∞	4	∞	∞	∞	∞
C	∞	9	∞	∞	∞	4	∞	∞	∞
D	9	∞	∞	∞	1	∞	2	∞	∞
E	∞	9	∞	9	∞	2	∞	3	∞
F	∞	∞	9	∞	9	∞	∞	∞	2
G	∞	∞	∞	9	∞	∞	∞	1	∞
H	∞	∞	∞	∞	9	∞	9	∞	2
I	∞	∞	∞	∞	∞	9	∞	9	∞

NOTA: COM S'HA EXPLICAT ABANS, ELS PESSOS DELS CAMINS ENTRE DOS PUNTS NO HAN DE SER NECESSARIAMENT IGUALS PER A AMBDOS SENTITS.

Les caselles verdes son els pesos dels camins d'anada, les taronges de tornada i les altres son connexions inexistents.

Al ser un circuit amb nodes equidistants i pla, els costos dels camins en realitat son iguals i la taula hauria d'estar emplenada amb el mateix valor en totes les caselles verdes. Però, si li donem la informació així a l'algorisme, depenent de quin sigui, es podrà quedar estancat. Així doncs, per evitar això, per fer el problema més realista i per a poder implementar l'algorisme simularem que no és així assignant valors de pes aleatòriament però amb seny: Per a la aplicació s'ha utilitzat una altra taula emplenada seguint el criteri de 'com més girs tingui una ruta més alt ha de ser el pes total del recorregut' i està explicat detalladament en l'apartat 8.7.1: EMPLENAMENT DE LA MATRIU DE PESOS.

Es defineix el node A com a node ORIGEN, i el node I com el node TARGET per als exemples que es faran.

7.2.1 GREEDY

Un algorisme voraç (també conegut com Greedy) es un algorisme d'estratègia de cerca que consisteix en escollir la opció optima en cada pas local, sense fixar-se en els passos anteriors, amb la esperança d'arribar a una solució general optima. Aquest esquema algorítmic es el que menys dificultats planteja a la hora de dissenyar i comprovar el seu funcionament, però no garanteix donar la solució optima global. Normalment s'aplica a problemes d'optimització.

- Funcionament:

Primer es declara un conjunt de candidats, que son tots els nodes possibles i els pesos dels camins. Després, es declara un node de partida i un node objectiu.

Començant pel node de partida, l'algorisme escull el millor candidat possible per ser el següent node segons el pes del camí que hi porta. Això ho fa avaluant un per un els candidats comparant el pes del camí amb el mínim registrat. Si el seu valor es menor al mínim, es registra el node i el pes del seu camí i, si no es així, es descarta. Després es passa al següent element del conjunt de candidats per seguir amb la cerca del mínim.

Una vegada comprovats tots els nodes, es passen les dades del node mínim detectat al conjunt que formarà la solució.

Tot això s'itera en un bucle fins que l'últim node detectat coincideixi amb el node objectiu.

- Aplicació a l'exemple del projecte:

Comencem pel punt inicial A. Tenim dues opcions, anar a D o a B. Comparant els pesos, el camí amb pes mes baix es el camí que porta a B, llavors D queda descartat i passem a B:

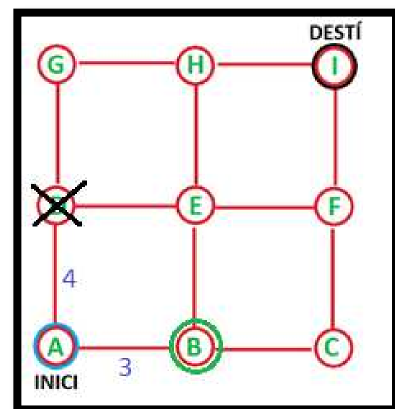


Figura 7.3.

De la mateixa manera, s'analitzen els camins que surten del node B. Tenim tres opcions, entre elles la opció de tornar a A. Però, per evitar que el programa es quedi en un bucle infinit A-B-A-B, s'ha assignat un valor alt al camí de tornada. Llavors, realment nomes tenim dues opcions: anar a C o a E. Comparant els valors dels camins, E queda descartat i passem a C:

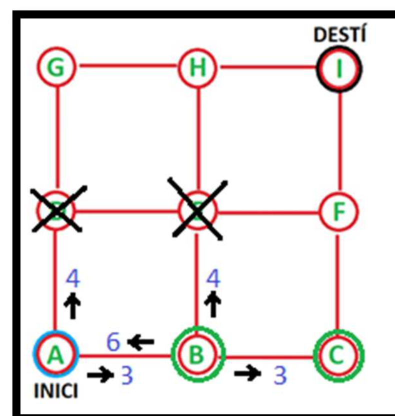


Figura 7.4.

NOTA: Els punts només queden descartats en la elecció actual i tornen a esta habilitats al pas següent. Només s'inhabiliten els nodes visitats.

Fem lo mateix per als seguents nodes fins a arribar al node Target (en aquest cas, el node I):

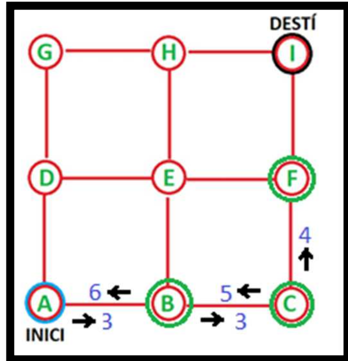


Figura 7.5.

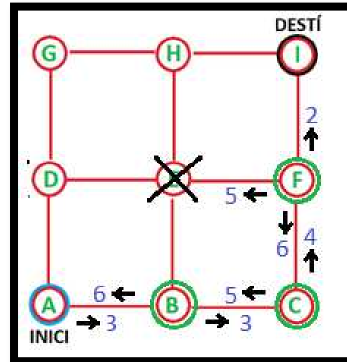


Figura 7.6.

El resultat ha sigut ABCFI amb un pes total de 12.

7.2.2 DIJKSTRA

L'algorisme de Dijkstra, també conegut com algoritme de camins mínims, es un algorisme per a la determinació del camí més curt, donat un node origen, cap a la resta dels vèrtexs en un graf amb pesos en cada aresta. El seu nom ve del científic de la computació Edsger Dijkstra, que el va descriure per primer cop al 1959.

- Funcionament:

La idea d'aquest algorisme consisteix en anar explorant tots els camins més curts des del node origen i que porten a la resta de nodes. A diferencia del Greedy, aquí si que es tornen a revisar nodes descartats per veure si existeix alguna combinació possible amb cost total menor. Quan s'obté la ruta més curta del node origen al node objectiu, l'algorisme es deté.

Una de les seves aplicacions més importants resideix al camp de la telemàtica on s'utilitza per resoldre grafs amb molts nodes, trobant així les rutes mes curtes entre un origen i tots els destins d'una xarxa.

- Aplicació a l'exemple del projecte:

Comencem al node A. El camí que porta a B te pes de 3 i el que porta a D pesa 4.El camí amb menor pes es el que porta a B. Registrem el pes total (a partir d'ara, PT)= 3.

De B, el camí amb menor pes es el que porta a C amb pes de 3. Guardem PT=6. Com la branca que anava a D te menor pes, canviem i la explorem:

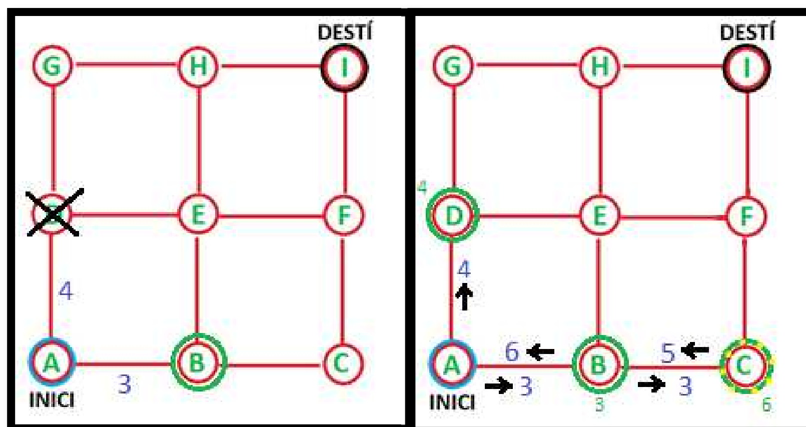


Figura 7.7.

A-D te pes de 4. Sortint de D tenim pes de 1 a E i 2 a G. Anem a E amb pes total de 5. Segueix sent menor que el pes de la branca A-B-C així que seguim.

De E els camins lògics son cap a F amb pes de 2 o H amb pes de 3. Anem a F amb pes total de 7. Es major que el pes de la branca A-B-C llavors canviem i la seguim desenvolupant.

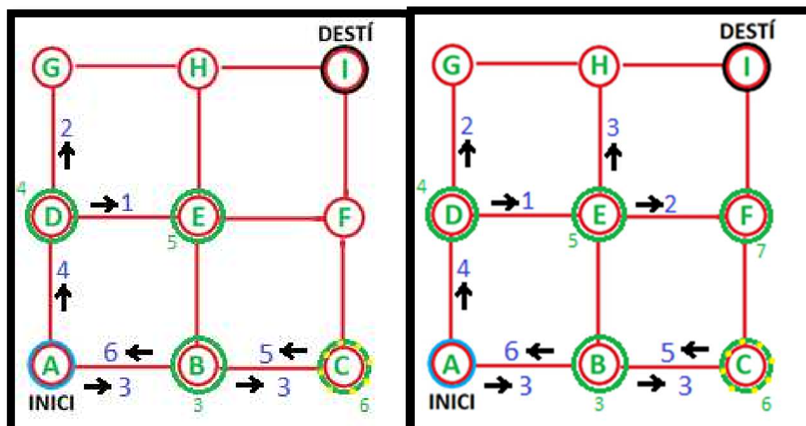


Figura 7.8.

De C podem anar a F amb 4 o a B amb 5. Anem a F amb pes total de 10. La branca A-D-E-F tenia un PT de 7 i com es menor, aquesta queda TOTALMENT DESCARTADA.

Recordem que la branca ADG tenia un PT=6 que es menor a l'actual (7) així que canviem de branca per desenvolupar-la.

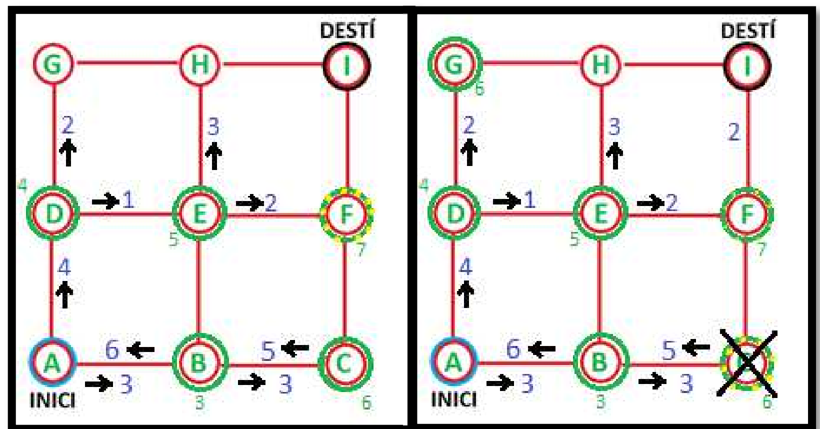


Figura 7.9.

Arribem a H amb pes de 7 i continuem a I (*target*) amb pes total de 9.

Ja hem arribat al node destí però tenim encara una branca amb potencial de ser el millor camí així que TORNEM A DESENVOLUPAR-LA.

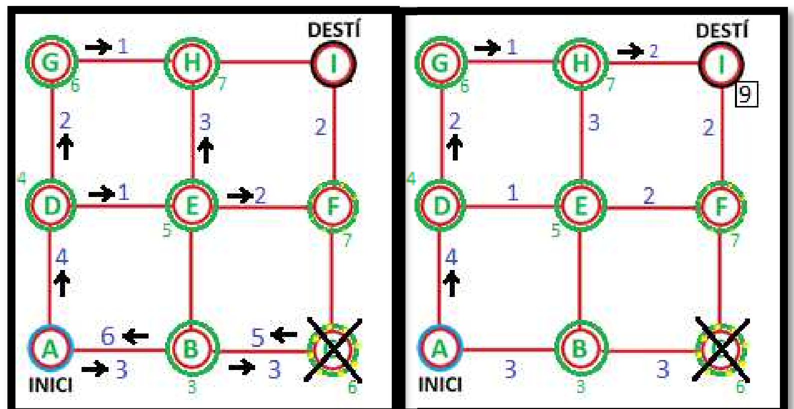


Figura 7.10.

A-D-E-F=7. De F anem a I amb pes total de 9. 'Casualment' ambdós camins tenen el mateix valor així que qualsevol dels dos seria vàlid.

El resultat ha sigut ADEFI o ADGHI amb un pes total de 9.

Comparant amb el Greedy (PT=12) , veiem que aquest algorisme ha detectat camins més òptims però el nombre d'operacions ha sigut major.

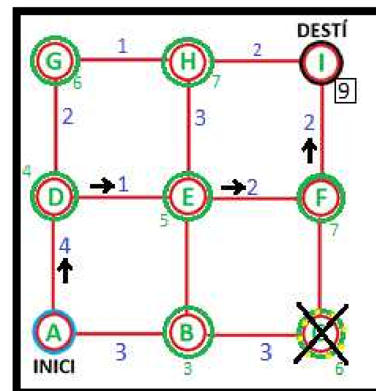


Figura 7.11.

7.2.3 BREADTH-FIRST SEARCH (BFS)

Aquest es un algorisme per cercar informació en forma 'd'arbres' o grafs. Comença al node identificat com arrel (o un node arbitrari en cas de no indicar-se) i explora tots els nodes veïns. Una vegada acabat, passa a analitzar els nodes del següent nivell i així iterativament fins arribar al node destí o a l'últim en cas de no haver-se designat. L'algorisme utilitza una cua on emmagatzema tots els nodes pendents de visitar o que encara no han estat descartats.

- Aplicació a l'exemple del projecte:

Recordem que el node inicial es el A. Els nodes del primer nivell, es a dir, els mes propers son B i D.

Ara toca analitzar els dels següent nivell. Per fer-ho més simple contarem el nombre de camins, que en aquest cas són 2: C, E, G.

Al node E se li arriba per dos camins diferents: ADE amb pes 5 i ABE amb pes de 7. ABE queda descartat.

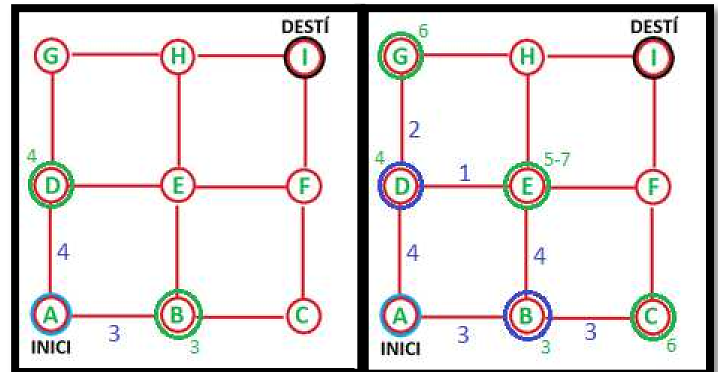


Figura 7.12.

EL següent nivell es a 3 camins de distancia: F,H.

- H: s'arriba amb dos camins, ADGH=7 i ADEH=8. També es podria amb ABEH=10 però el pes es molt major així que ja hem fet be al descartar-lo.
- F: també hi porten dos camins diferents: ABCF=10 i ADEF=7. ABCF queda descartat.

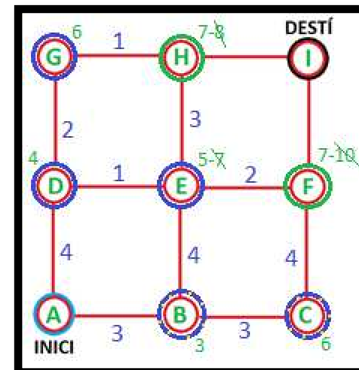


Figura 7.15.

Finalment arribem a l'últim node , que coincideix amb el node destí. Tenim dos possibles camins ADEFI i ADGHI ambdós amb un cost total de 9. Per tant, en aquest cas qualsevol dels dos camins seria vàlid.

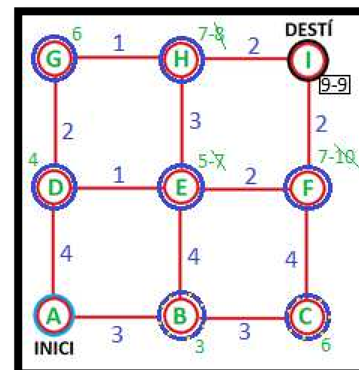


Figura 7.16.

7.2.4 DEPHT-FIRST SEARCH (DFS).

La seva estratègia de recerca es similar a la de l'anterior mètode de pathfinding però aquest el que fa es anar expandint tots i cadascun dels nodes que va localitzant, de forma recurrent, en un camí concret. Quan ja no queden més nodes per visitar en el camí, torna (Backtracking) de forma que repeteix el mateix procediment per cada un dels nodes veïns del node ja processat.

- Aplicació a l'exemple del projecte:

Comencem pel node A. Podem anar al node B o D, però el node més proper es el B. Del B podem anar a E o C, anem a C. De C a F. De F podem anar a E o a I, I es el més proper i coincideix amb el node destí. Hem trobat un camí amb PT=12.

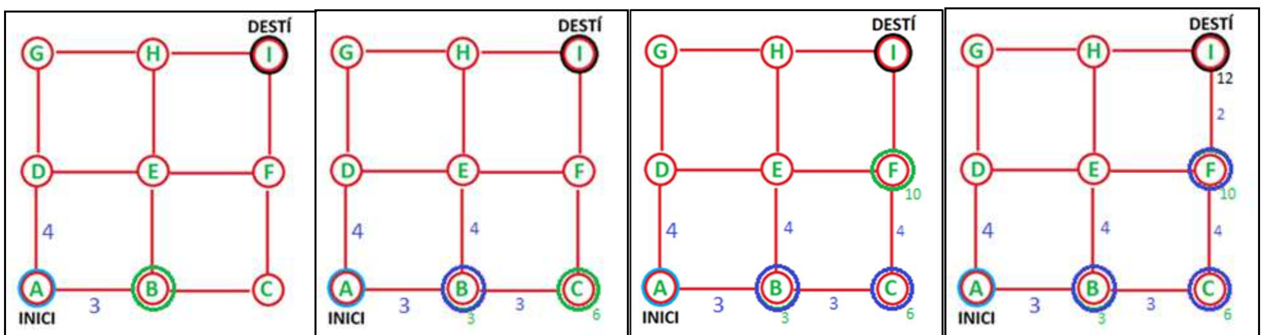


Figura 7.17. i 7.18.

Tornem enrere, al punt F i intentem desenvolupar la branca que va cap a E. Veiem que el pes total anant a E augmenta fins a 19 i és més que el PT del camí més proper al destí així que anul·lem la branca i intentem desenvolupar l'altra branca que surt de B, arribant a E amb PT=7.

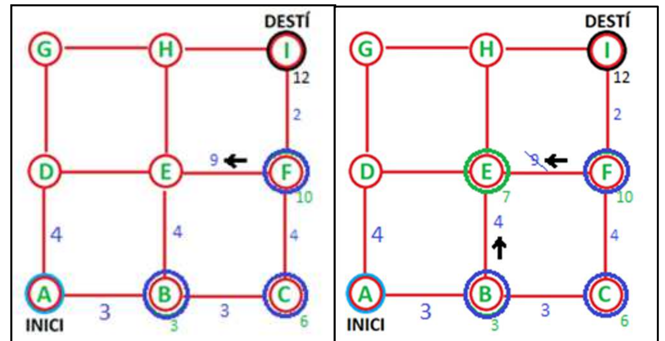


Figura 7.19.

De E podem anar a D,H i F encara que D no és un destí lògic. Per ara anem a F amb PT=9 i arribem a I amb PT=11. Com es menor al PT del camí ABCFI, aquest camí quedarà descartat.

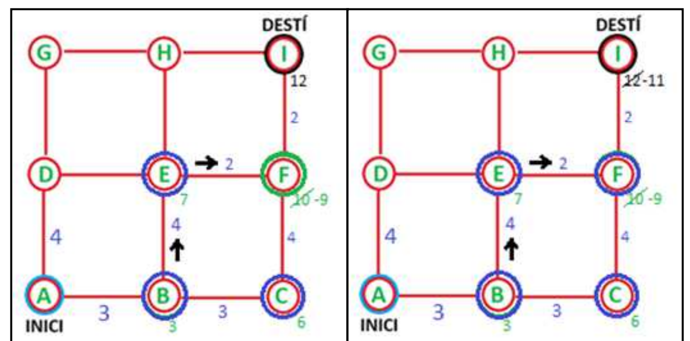


Figura 7.20.

Ara expandim la branca que anava a H i arribem al punt destí amb PT=12. Es major que el mínim registrar així que també queda descartat.

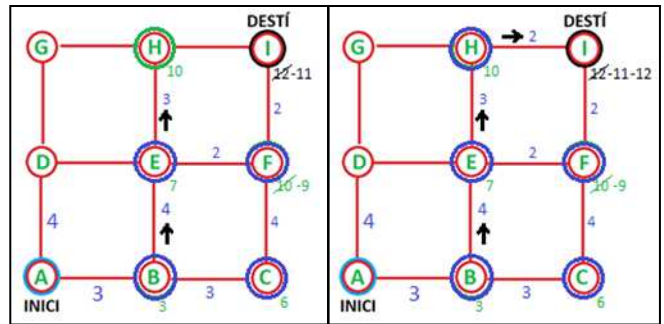


Figura 7.21.

Tornem enrere i, encara que no sigui lògic, intentem expandir la branca que anava a D. Veiem que el PT incrementa a 17 que es molt més que el PT del camí més òptim registrat fins ara així que descartem desenvolupar-la.

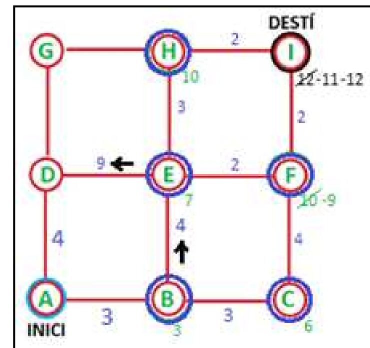


Figura 7.22.

Anem més enrere i desenvolupem la branca A-D. Podem anar a G però per ara anem a E. Arribem amb PT=7, fet que, de moment, fa preveure el descart dels altres camins que passaven pel node E.

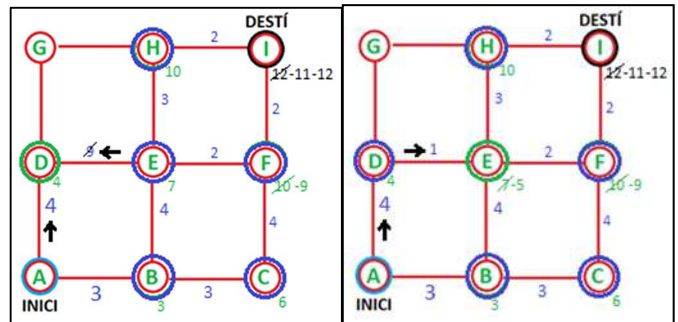


Figura 7.23.

Passem al node F i finalment al node destí amb PT=9 descartant tots els camins anteriors.

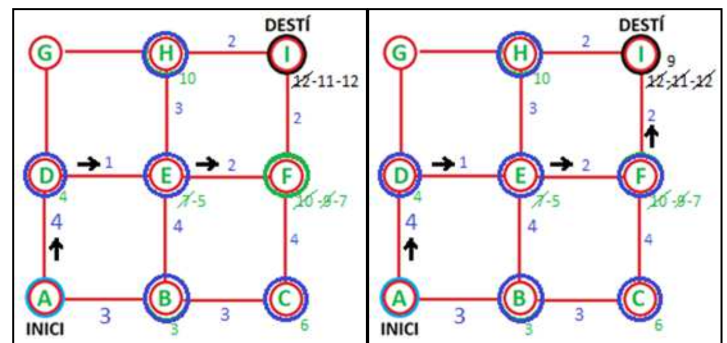


Figura 7.24.

També teníem la possibilitat d'arribar-hi passant per H, però en aquest cas arribem amb un PT= 10. Es un bon camí però no es millor que l'anterior així que també queda descartat.

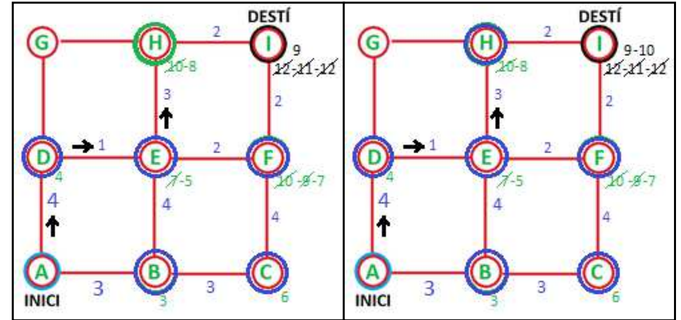


Figura 7.26.

Tornem enrere i desenvolupem la branca AD anant a G. Passem per H i arribem a I amb PT=9. Es igual al mínim trobat així que tornem a obtenir el mateix resultat que amb l'anterior algorisme (BFS).

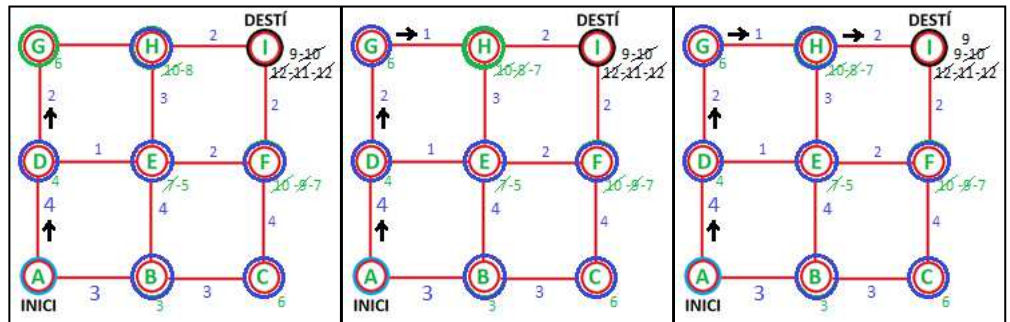


Figura 7.27.

7.2.5 A STAR (A*)

Aquest algoritme de pathfinding va ser publicar al 1968 per recercadors del projecte Shakey que consistia en crear un robot mòbil que pugues planejar les seves pròpies accions. Es pot interpretar com una extensió de l'algorisme de Dijkstra però amb millors resultats gràcies a la utilització de la heurística per guiar la recerca. Pot ser utilitzat per trobar camins òptims per a qualsevol problema satisfent les condicions de cost en forma algebraica.

- Funcionament:

El funcionament d'aquest algorisme és bàsicament igual al Dijkstra però la funció de cost inclou la heurística. Basant-se en la suma del cost del camí al node i del cost estimat, l'algorisme ha de determinar a cada iteració quin dels camins estendre. Això queda representat en una funció on es selecciona el camí que minimitza el seu valor.

$$f(n) = g(n) + h(n)$$

On n es el node a avaluar, $g(n)$ es el cost del camí des del node inicial fins al node n i $h(n)$ és la heurística que estima el cost del camí més curt des del node n fins al node final. La recerca acaba quan el camí que s'està expandint va del node inicial al final o no hi ha més camins a escollir.

- Aplicació a l'exemple del projecte:

No s'ha fet exemple aplicant aquest algorisme perquè tant el procés com el resultat serà igual que amb el Dijkstra, la única diferencia la marcarà el pes de la heurística dels punts.

D'aquests 4 algorismes, DFS I BFS són els que més temps d'execució necessitarien, ja que exploren tots i cadascunes dels possibles camins entre el node *origen* i el *target*, però garanteixen obtenir la solució optima global. Com el que ens interessa es tenir un temps d'execució lo més petit possible, queden descartats. A* ve a ser lo mateix que Dijkstra però afegint la heurística, fet que complica més el càlcul i per al nostre circuit, al ser bastant simple, no fa falta. Així que ens quedem amb el Greedy i el Dijkstra.

8 APLICACIONS DESENVOLUPADES

Com s'ha explicat al principi de la memòria, el meu projecte consisteix en fer que el Robotino sigui capaç de moure's de forma autònoma entre dos punts del circuit guiant-se per la visió i creant la seva pròpia ruta.

Tota la aplicació gira en torn d'aquest programa. És aquí on es fa la utilització de les API's i les seves funcions. Com és d'esperar, el programa comença en la tasca principal *main* on es creen els fils d'execució i les tasques amb les seves respectives prioritats. Es demana per pantalla la interacció de l'usuari per poder fer el càlcul de ruta i es procedeix a fer-ho amb les funcions de pathfinding. Una vegada calculada, es comença a executar el programa contínuament en forma de bucle infinit fins que es finalitza la seqüencia.

Per facilitar la comprensió del funcionament, a continuació tenim el diagrama d'estats del programa principal amb la informació que es comparteix entre les diferents tasques (*figura 8.1*).

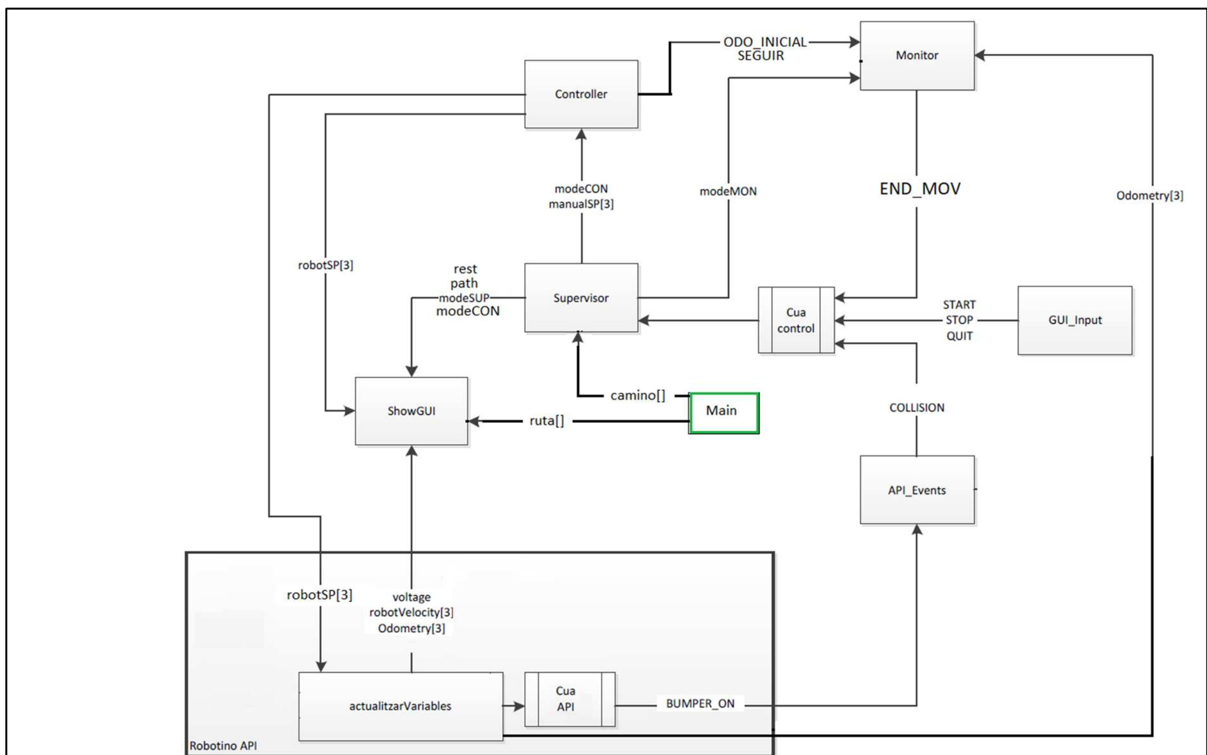


Figura 8.1. Diagrama d'estats del programa.

8.1 RESÚM DEL FUNCIONAMENT

A l'inici del programa es demana a l'usuari que introdueixi per teclat la lletra que identifica el node inicial, el final, i un nombre assignat al tipus d'algorisme de pathfinding. Si es un 1 vol dir Greedy, si es un 2 vol dir Dijkstra. Les lletres s'han d'introduir en majúscules.

Amb aquestes dades es cridarà la funció corresponent de la llibreria creada de Pathfinding, on es calcularà la ruta i ens la retornarà en un vector. Una vegada tenim per pantalla la ruta, el programa es quedarà en espera de la indicació de l'usuari per començar la seqüencia i execució del programa principal. Això es fa introduint un 1 que es tradueix com a START. Un 2 es un STOP, per pausar tant el programa com el robot durant la execució i un 3 es un QUIT, per avortar la execució del programa.

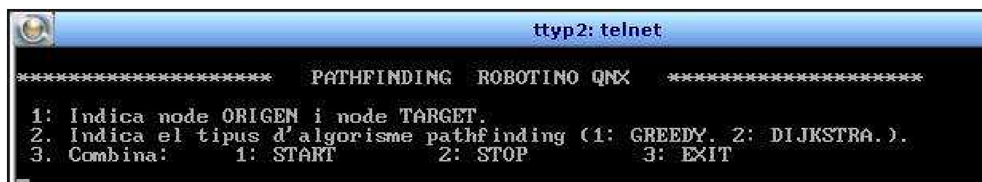


Figura 8.2: Menú inicial del programa.

El programa principal consta de varis subprogrames però com a màxim es desenvoluparan tres tasques simultàniament:

- Per una banda la obtenció de la imatge i posterior processat.

La major part d'aquesta tasca s'ha heretat de projectes anteriors, però s'han fet algunes modificacions. La IP de la càmera ja no es .21 sinó .114 i com la càmera ara no està situada paral·lela al terra i les condicions d'il·luminació son diferents, els paràmetres i llindars per a aplicar els filtres s'han hagut de canviar. També s'han utilitzat diferents filtres.

La seqüencia d'aquesta part es la següent:

- Petició i obtenció de la imatge a la càmera
- Processat de la imatge
- Identificació de la línia
- Selecció dels punts per trobar la equació de la recta.
- Càlcul de la distancia i angle del robot respecte de la recta.
- Càlcul de la inversa del radi de gir (Formula de Ollero).
- Càlcul de la velocitat angular a partir de la inversa del radi de gir.

- Per altre banda el moviment físic del robot.

A partir de la seqüencia prèviament guardada en memòria i que conté els nodes que formen el camí, la tasca Supervisora dictarà si el robot ha d'avançar, girar o quedar-se quiet. Per a cadascuna d'aquestes accions hi ha un estat per a la tasca Controladora i només podrà executar-se un cada vegada. Es tracta d'un switch case i a cada case s'envien les consignes adequades per al moviment desitjat.

L'encarregada de controlar que el robot es mogui la distancia requerida és la tasca Monitor. Aquesta tasca utilitza els encoders de les rodes per mesurar la odometría i comparar-la amb uns valors fixats. Aquest valors son els que faran saber a la tasca Supervisora quan s'ha completat un moviment.

Tot el programa s'executa amb un únic processador, fet que ens obliga a pausar les tasques quan no s'estan utilitzant i posar intervals d'execució a les tasques actives.

- Finalment la mostra simultània per pantalla de les dades sobre el moviment.

Després tenim dues tasques més; una encarregada de recol·lectar la informació que volem mostrar pel terminal (ShowGUI) i la altra encarregada de mostrar-la (ShowAdvanced).

La informació que es mostra es variada, però s'ha seleccionat tot allò que en principi serà útil per a l'usuari:

- Lay-OUT del circuit amb la cadena de punts que formen el camí a realitzar.
- Consignes de velocitat i velocitat real del robot
- Modes de funcionament del Supervisor, Controlador i Monitor.
- Valors de la odometría.
- Diversos avisos sobre el estat de funcionament.

NOTA: La comunicació entre tasques es fa amb esdeveniments i compartint valors en variables globals.

8.2 CALIBRATGE

Anteriorment ja s'ha explicat tot el que es necessita per fer el calibratge. Aquí es detallaran els passos per fer-ho. Aquest

Primer de tot, s'ha de posicionar el robot en el cercle marcat al patró de calibratge que hi ha al laboratori.

1. Petició de imatge a la càmera:

Abans de fer la petició s'han de tenir en compte les condicions lumíniques (veure l'apartat 8.4.1) . Després de tenir-ho tot llest s'ha d'establir la connexió amb la càmera. Aquí s'explica com fer-ho per HTTP però també es pot fer amb comandes de Matlab.

Per HTTP només cal obrir una finestra de l'Explorer (o qualsevol altre navegador) i escriure la direcció IP de la càmera. Si estem a la mateixa xarxa, la detectarà i se'ns apareixerà una finestra en la que se'ns demanarà el nombre d'usuari i contrasenya.

- Usuari: root
- Contrasenya: ROBOTINO (en majúscules).

Figura 8.3: Finestra d'identificació per accedir a la càmera.

S'ha d'anar en compte de no cometre el mateix error que cometia al principi. En el programa principal es treballa amb una resolució petita (160x120) per agilitzar el temps de petició i processat de la imatge. Però, a la hora de fer el calibratge, com interessava tenir una resolució alta per poder veure la imatge clarament i poder discernir be on comencen i on acaben les graelles, ho feia amb una resolució més gran (480x360). El problema venia quan intentava expressar un píxel de la imatge capturada en el programa principal a coordenades robot. Hem donava valors erronis perquè els valors dels píxels no tenien la mateixa relació. Per això, s'ha d'anar amb compte de fer el calibratge amb la mateixa resolució que es treballarà després o establir uns paràmetres que facin mantenir la mateixa relació de píxels per a diferents resolucions.

Jo he optat per la opció més simple, que es treballar amb la mateixa resolució que faig el calibratge.

La comanda URL per fer la petició d'imatge en bitmap i amb resolució 160x120 es la següent:

http://192.168.1.21/axis-cgi/bitmap/image.bmp?resolution=160x120

Si es vol treballar amb imatges '.jpg'. La URL és la següent:

http://192.168.1.21/axis-cgi/jpg/image.cgi?resolution=160x120

2. Obrim Matlab i ens assegurem que estem a la carpeta de treball. Hem de tenir instal·lada la toolbox de calibratge de Matlab. Sinó, es pot descarregar de la pagina oficial de Caltech [5].

3. Set path:

Aquesta funció serveix per afegir la carpeta de la toolbox i la carpeta de treball (En el nostre cas estan juntes).

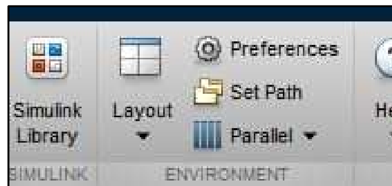


Figura 8.4.

Fem clic a 'Add with subfolders' i seleccionem la carpeta principal:

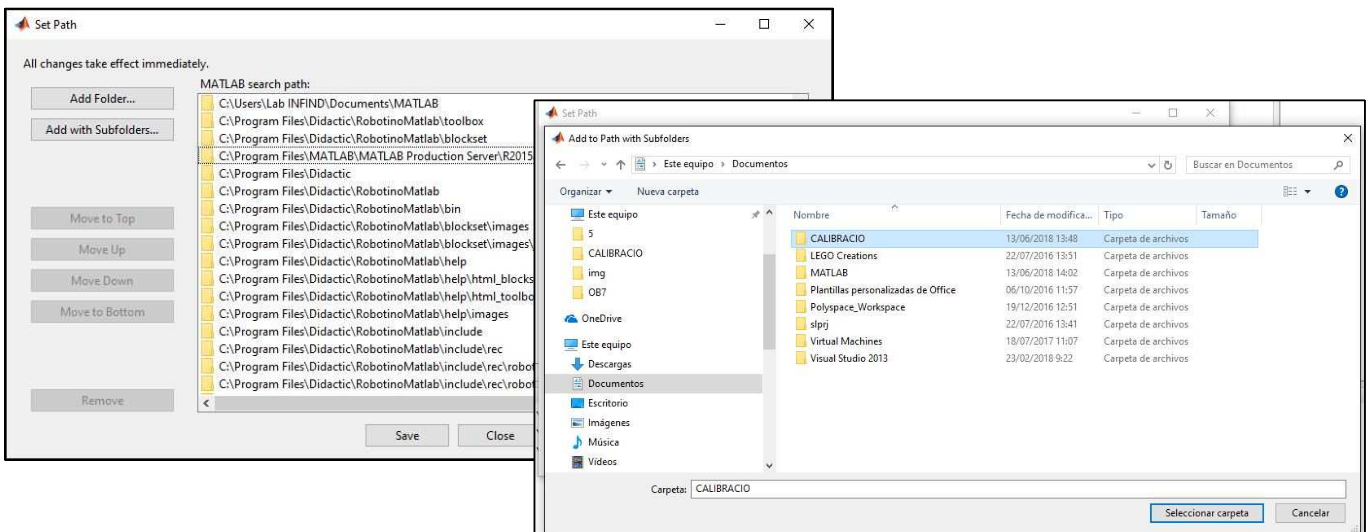


Figura 8.5: Set Path.

Ens assegurem de que surten totes les subcarpetes, cliquem 'Save' i després 'Close'.

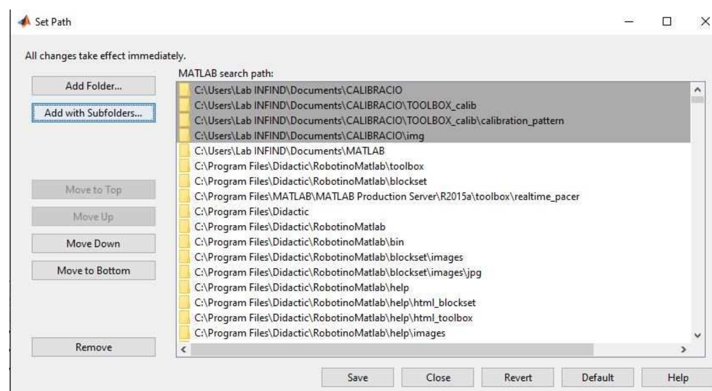


Figura 8.6.

Aquest es un pas previ al calibratge que només s'ha de fer la primera vegada i no fa falta tornar a fer-lo a menys que es canviï la carpeta de treball. A partir d'aquí comença el calibratge:

4. Inici de la toolbox i càrrega de les imatges:

Executem l'arxiu calib.m escrivint 'calib' al terminal. S'obrirà la GUI de calibratge i seleccionem 'standard':

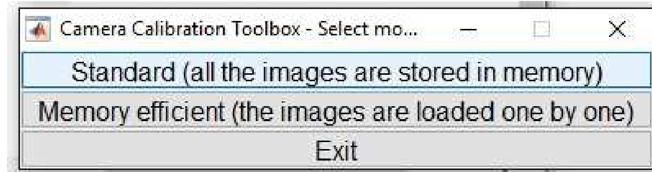


Figura 8.7.

El primer cop que calibrem tindrem que seleccionar 'read images' per a carregar totes les imatges de la carpeta de treball. Després, si no ens agrada el resultat i volem tornar a calibrar amb la mateixa imatge, no farà falta fer-ho. Haurem de passar directament al pas següent.

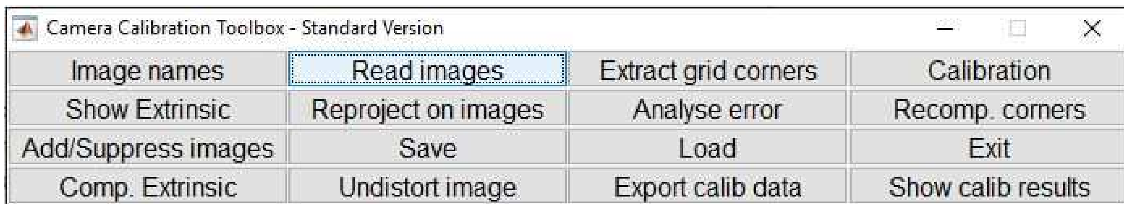


Figura 8.8: Menú de la toolbox de calibratge.

Després de llegir les imatges ens sortirà al terminal els noms de tots els arxius i ens demanarà el nom de la imatge per calibrar. Cal posar el nom sense extensió i sense números. Desprès, en funció de la extensió, s'haurà d'introduir una lletra al teclat. En el nostre cas, al ser una imatge .jpg, haurem d'introduir una 'j' minúscula.

```
>> calib

.                          TOOLBOX_calib              img
..                         act3_de_prueba.m           para C
Actividad2.m               act3_de_prueba1.m          pixelApunt.m
Actividad3.m               calib jpg                  rodrigues.m
Calib_Results.m            calib_data.mat             rotacion_camara_a_calibracion.m
Calib_Results.mat          calib_tfg.mat              rotacion_camara_a_calibracion222.m
Calib_Results_old0.m       extract_grid.m             rotation.m
Calib_Results_old0.mat     extrinsic_computation.m
PATHDEF.m                  foto2.jpg


Basename camera calibration images (without number nor suffix): foto
Image format: ([]='r'='ras', 'b'='bmp', 't'='tif', 'p'='pgm', 'j'='jpg', 'm'='ppm') j
Loading image 1...
done
>> |
```

Figura 8.9.

Una vegada carregada la imatge, se'ns apareixerà per pantalla.

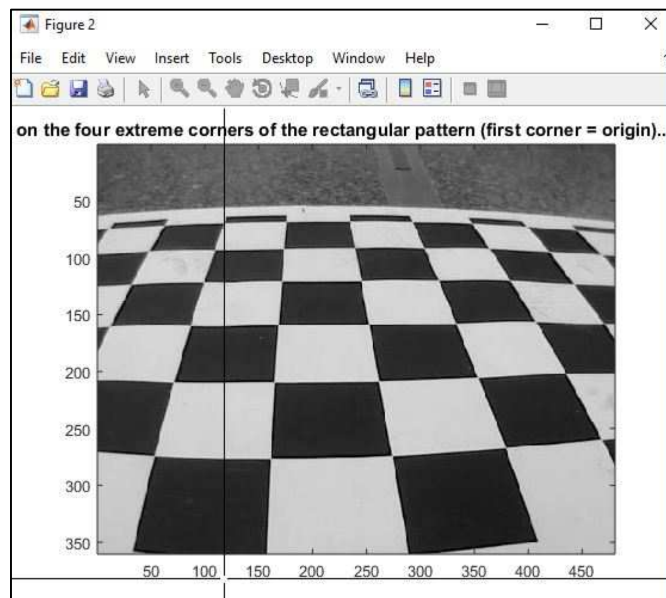


Figura 8.10.

5. Extracció dels cantons dels quadrats.

Ara hem de tornar a la GUI i seleccionar 'extract grid corners'. Es demana quantes imatges es volen processar. Com només es una, posem un 1.

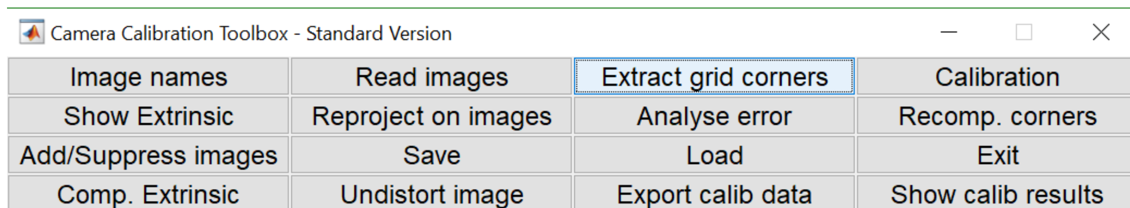


Figura 8.11.

NOTA: Per obtenir millors resultats es recomana fer el calibratge amb varies imatges, cadascuna amb una perspectiva diferent. Però, per el nostre cas, amb una en tenim prou.

Les dues següents preguntes tenen relació amb la mida de la finestra per trobar les cantonades dels quadrats. Deixant-les en blanc (polsant enter directament sense escriure res) estem seleccionant els valors per defecte, que depenent de la versió pot ser **wintx=winty=5** que dona una finestra efectiva de mida **11x11** píxels o, com en el nostre cas, **wintx=winty=4** amb mida **9x9**.

Després es pregunta si volem utilitzar el mètode automàtic de contar quadrats o volem fer-ho manualment. Posem un 1 que significa manualment. La funció automàtica va bé quan es treballa amb varies imatges.

```
Extraction of the grid corners on the images
Number(s) of image(s) to process ([] = all images) = 1
Window size for corner finder (wintx and winty):
wintx ([] = 4) =
winty ([] = 4) =
Window size = 9x9
Do you want to use the automatic square counting mechanism (0=[]=default)
  or do you always want to enter the number of squares manually (1,other)? 1

Processing image 1...
Using (wintx,winty)=(4,4) - Window size = 9x9      (Note: To reset the window size, run script clearwin)
Click on the four extreme corners of the rectangular complete pattern (the first clicked corner is the origin)...
```

Figura 8.12.

Ara ens sortirà una altra finestra amb la imatge. Haurem de seleccionar molt acuradament les vores d'una certa quantitat de quadrats. Intentant ser lo més precisos possible. Tenim dues opcions:

- Mantenir els eixos de coordenades món: S'ha de començar per el cantó inferior esquerre i continuar en el sentit contrari a les agulles del rellotge.
- Eixos de coordenades del Robot: SI, pel contrari, volem rotar-los 90º per tal de fer-los coincidir amb els eixos de coordenades del robot Robotino, s'ha de començar pel cantó superior esquerra i continuar en el sentit de les agulles del rellotge. Aquesta rotació ha de quedar reflectida en la posterior matriu de rotació.

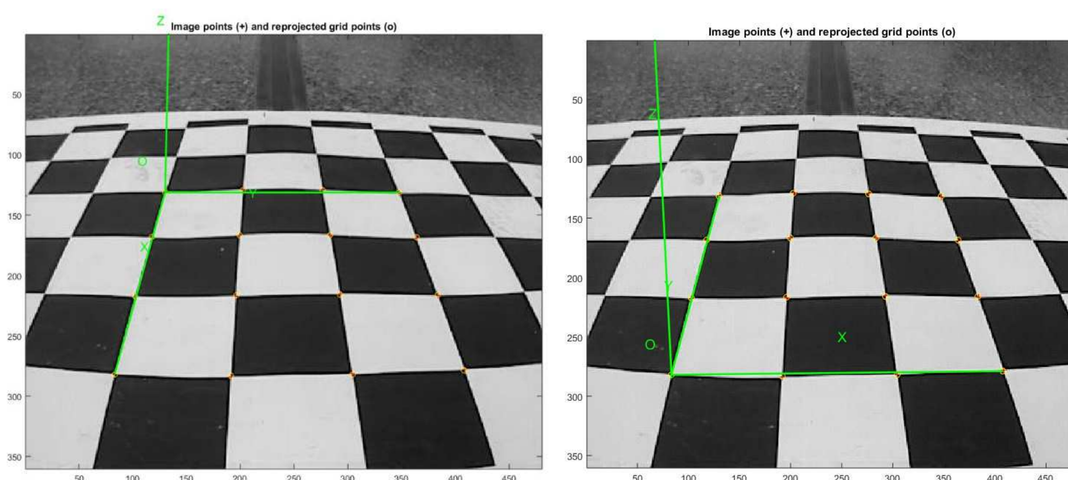


Figura 8.13: Rotació dels eixos de coordenades vs mantenint-los.

Ara ens demanarà el nombre de quadrats que hem seleccionat en l'eix X, en l'eix Y, i les dimensions dels costats de cada quadrat. Tots els quadrats son iguals i tenen arestes de 5cm (50mm). En el nostre cas, s'han seleccionat 3 en la direcció de l'eix X i 3 en l'eix Y:

```
Processing image 1...
Using (wintx,winty)=(4,4) - Window size = 9x9      (Note: To reset the window size, run script clearwin)
Click on the four extreme corners of the rectangular complete pattern (the first clicked corner is the origin)...
Number of squares along the X direction ([]=10) = 3
Number of squares along the Y direction ([]=10) = 3
Size dX of each square along the X direction ([]=100mm) = 50
Size dY of each square along the Y direction ([]=100mm) = 50
If the guessed grid corners (red crosses on the image) are not close to the actual corners,
it is necessary to enter an initial guess for the radial distortion factor kc (useful for subpixel detection)
Need of an initial guess for distortion? ([]=no, other=yes)
```

Figura 8.14.

Després tornarà a sortir una imatge en una finestra. Hem d'observar que els punts roigs coincideixen amb els cantons dels quadrats. En cas de que no ho sigui, haurem de contestar '*yes*' i introduir diferents valors per a la distorsió fins a quedar satisfets amb el resultat (normalment es inferior a 1). Si, després de provar amb varis valors no es troba un resultat adequat, es recomana sortir de la toolbox, borar el workspace i tornar a començar.

Fent enter diem que no, i ens torna a sortir una finestra amb la imatge però ara amb els punts vermells acompanyats d'un requadre blau. Al terminal es pot llegir que posa DONE.

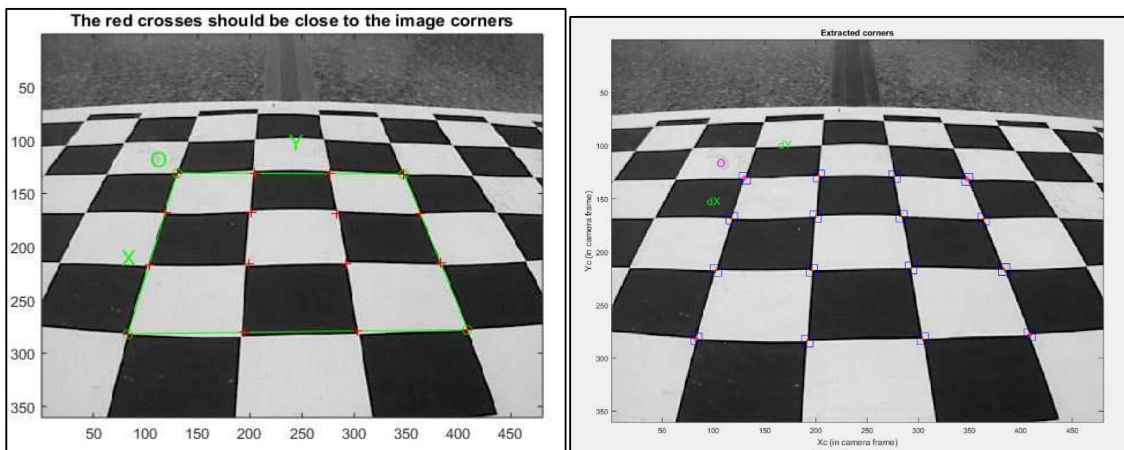


Figura 8.15.

6. Determinació dels paràmetres intrínsecs:

Tornem a la GUI. Aquest cop fem clic sobre 'calibration' i, després de fer càlculs, sortiran varis paràmetres de la càmera pel terminal com per exemple la distancia focal, els errors de píxel etc. Tots aquests valors son els components intrínsecs de la càmera. Si ens fixem, podem veure fins i tot l'angle que presenta l'eix dels píxels (imatge) respecte als eixos del món real.

```
Main calibration optimization procedure - Number of images: 1
Gradient descent iterations: 1...2...3...4...5...6...7...8...9...10...11...12...13...14...15...16...17...18...19...20...21...22...23...2
Estimation of uncertainties...done

Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 589.22020   718.60098 ] +/- [ 71.05321   533.50738 ]
Principal point:       cc = [ 239.50000   179.50000 ] +/- [ 0.00000   0.00000 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000  ]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc = [ -0.69844   0.92583   0.03838   -0.00286  0.00000 ] +/- [ 0.45490   3.69120   0.02450   0.00670  0.00000 ]
Pixel error:          err = [ 0.24762   0.63754 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Figura 8.16: Resultats dels paràmetres intrínsecs.

7. Determinació dels paràmetres extrínsecs:

Tornem un altre cop a la GUI i seleccionem '*comp.extrinsic*'. Ja és l'últim pas!

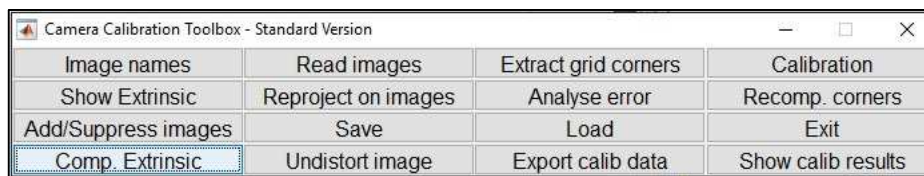


Figura 8.17.

Es demana, igual que per a 'extract grid corners', el nom de la imatge i el format. La diferencia es que aquest cop si que hem d'introduir el nom amb els índexs que tingui, si en té.

Es torna a seleccionar els quadrats a la imatge. Seguint l'ordre que ho hem fet abans.

```
Extrinsic parameters:

Translation vector: Tc_ext = [ -75.511772     -26.300847      395.567571 ]
Rotation vector:   omc_ext = [ 1.517715       1.487202     -0.958941 ]
Rotation matrix:    Rc_ext = [ 0.026707       0.999595     0.009823
                               0.403553      -0.001791     -0.914954
                              -0.914566       0.028400     -0.403438 ]
Pixel error:           err = [ 0.24281    0.64263 ]


>> KK
KK =
  589.2279         0  239.5000
         0  718.6472  179.5000
         0         0    1.0000
```

Figura 8.18.

Després de fer el calibratge hem imposat en el programa dos punts alineats però amb diferent altura. (x1=120,y1=39; x2=120,y2=89). Hem transformat aquests punts píxel a punts en coordenades robot, i en principi, al ser dos punts alineats, el seu componen x hauria de coincidir també, però no. Ens surt que estan distanciats 2cm en l'eix Y del robot (X mon). Això lo que vol dir es que la càmera esta una mica torta, per lo tant, encara que per a nosaltres siguin dos punts alineats per a la càmera no ho son.

NOTA: després es va modificar la posició de la càmera per intentar posar-la lo més dreta possible.

Com ja s'ha dit abans, al principi el format de la imatge que utilitzava era en bmp amb resolució alta (640x480). El principal problema que tenia era que l'error de píxel que donava era bastant gran (normalment entre 1 i 2). Després de varis intents vaig decidir tractar amb un altre fotografia i un altre format (jpg) que dóna millors resultats; els errors de píxel eren menors a 1. Després de detectar l'error provinent de treballar amb diferents resolucions, tenia dos opcions: treballar al programa principal amb resolucions més grans o fer el calibratge amb menor resolució. La primera opció implicava tenir que canviar un altre cop la estructura de la funció per detectar punts, i la segona potser complicava el calibratge a l'hora de fer la selecció dels cantons acuradament.

Desprès de varies proves vaig veure que treballar amb major resolució al programa principal també implicava un major temps de captura de la imatge i de processat, i fer el calibratge amb menor resolució ajudava a reduir l'error de píxel. Encara que seguia sent una mica major que amb jpg, ara el valor era també menor que 1.

captura	processat	diftotal
223.965728	17.997246	241.962974
213.967258	17.997246	231.964504
221.966034	17.997246	239.963280
242.962821	17.997246	260.960067
213.967258	17.997246	231.964504
247.962056	17.997246	265.959302
219.966340	17.997246	237.963586
248.961903	17.997246	266.959149
463.929008	17.997246	481.926254
427.934516	17.997246	445.931762
209.967870	17.997246	227.965116
215.966952	18.997093	234.964045
253.961138	17.997246	271.958384
245.962362	17.997246	263.959608
209.967870	17.997246	227.965116

captura	processat	diftotal
3221.507034	12.998011	3234.505045
3163.515908	11.998164	3175.514072
3050.533197	11.998164	3062.531361
3159.516520	11.998164	3171.514684
3090.527077	11.998164	3102.525241
3134.520345	11.998164	3146.518509
3214.508105	12.998011	3227.506116
3099.525700	11.998164	3111.523864
3321.491734	11.998164	3333.489898
3273.499078	11.998164	3285.497242
3145.518662	12.998011	3158.516673
3168.515143	11.998164	3180.513307
3118.522793	11.998164	3130.520957
3291.496324	12.998011	3304.494335
3250.502597	11.998164	3262.500761

Figura 8.19: Proves de temps: Resolució baixa vs alta.

El temps de captura mitja amb resolució baixa es de 257,427 ms amb pics de fins aproximadament mig segon. Amb resolució alta, el valor mínim és de 3 segons. 12 vegades major!! I sis vegades major al temps màxim detectat. Veient això podem afirmar que la captura de la imatge es el punt més crític, a efectes d'optimitzar el temps d'execució del programa i no te gaire sentit augmentar la resolució de treball. Llavors, la única opció lògica que ens queda es fer el calibratge amb la resolució petita.
Aquesta proba ens servirà també mes endavant a l'hora d'establir els intervals de temps de les tasques del programa.

8.3 VALORS A EXPORTAR A QNX

Ara tenim els valors intrínsecs, extrínsecs i la distorsió de la càmera. Dins d'aquests tenim varis paràmetres, però no tots ens interessen. Tornant al capítol 7.3.4 on s'explica com passar un punt referenciat en coordenades de la imatge a coordenades robot ens fixem en les equacions finals:

$$P_{robot} = R^T * K^{-1} * \lambda * P_{imatge} - R^T * t$$

$$\lambda = \frac{[R^T * t]}{[R^T * K^{-1} * P_{imatge}]} \quad (3)$$

Aquí veiem la matriu de rotació, de translació i la inversa de KK que conté els paràmetres intrínsecs de la càmera. Tots ells en coordenades homogènies. Podríem passar aquest valors a QNX i fer el posterior càlcul de matrius però, per simplificar el codi, he preferit calcular primer tot allò que sigui constant a Matlab. Aquest arxiu es troba a l'annex[D].

8.3.1 DECLARACIÓ DE LES 4 MATRIUS DE ROTACIÓ I TRANSLACIÓ EN COORDENADES HOMOGÈNIES:

```
R_r_c = [0 -1 0 0;
         1 0 0 0;
         0 0 1 0;
         0 0 0 1];
T_r_c = [1 0 0   62.7;
         0 1 0 -427.9;
         0 0 1      0;
         0 0 0      1];
T_c_c = [1 0 0 Tc_ext(1,1);
         0 1 0 Tc_ext(2,1);
         0 0 1 Tc_ext(3,1);
         0 0 0 1];
R_c_c = [Rc_ext(1,1) Rc_ext(1,2) Rc_ext(1,3) 0;
         Rc_ext(2,1) Rc_ext(2,2) Rc_ext(2,3) 0;
         Rc_ext(3,1) Rc_ext(3,2) Rc_ext(3,3) 0;
         0           0           0          1;]
```

Figura 8.20: Declaració de les matrius de Rotació i Translació en coordenades Homogènies.

Per poder començar, primer s'han de tenir declarades en coordenades homogènies les matrius de rotació i translació obtingudes del calibratge i del càlcul manual (figura 8.20).

8.3.2 CÀLCUL DE MATRIU M

Necessitem calcular la matriu M per poder fer la seva inversa adequadament.

```
M=T_c_c * R_c_c * R_r_c * T_r_c;
KK_inv= inv(KK);
```

Observant la matriu M veiem que tenim la nova matriu de rotació 3x3 en les 3 primeres columnes, i en la 4a columna tenim el vector de translació corresponent.

```
M =
    0.9996   -0.0266    0.0107   -1.4196
   -0.0019   -0.4344   -0.9007  157.4786
    0.0287    0.9003   -0.4343    7.2588
         0         0         0    1.0000
```

$$M = \begin{bmatrix} R_{3\times3} & R_{3\times3} * t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

```
R_3_3= [M(1,1) M(1,2) M(1,3);
        M(2,1) M(2,2) M(2,3);
        M(3,1) M(3,2) M(3,3)];
t=[M(1,4) ; M(2,4); M(3,4)];
```

$$M^{-1} = \begin{bmatrix} R^T_{3\times3} & -t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

7.1.1 CÀLCUL DE LANDA (λ)

Tenim indicada la equació per calcular la landa (λ) a l'apartat anterior. Observant veiem que el denominador depèn del punt de la imatge que s'està tractant, llavors separarem numerador i denominador i es deixarà com a bloc tot allò que sigui constant.

```
% P_img=[xi;yi;1];
%
%
 Nom = R_3_3'*t; %cte
 Denom = R_3_3'* KK_inv *P_img;  %cte * punto
 Landa_2= Nom(3)/Denom(3);  %lo puedo calcular en c
```

Figura 8.21.

Els valors que he anomenat A i B són els que s'han de passar a la funció '*PimgToRobot*' de la *apiimatge.* Aquests són els valors necessaris per poder calcular després la landa(λ) i el valor del punt de la imatge en coordenades del robot.

```
%P_rob = (R_3_3' * KK_inv * Landa_2* P_img) - R_3_3'*t %cte*punto - cte
A = R_3_3'* KK_inv  %lo llamare (A)
B = R_3_3'*t        %lo llamare (B)
% llavors: P_rob = (A*Landa_2*P_img)-B
% Xr=P_rob(1);
% Yr=P_rob(2);
```

Figura 8.22.

Ara només queda calcular a la funció els punts en coordenades robots implementant la següent equació:

$$P_{robot} = (A * \lambda * P_{imatge}) - B$$

7.2 CONNEXIÓ AMB LA CAMERA I OBTENCIÓ DE LA IMATGE

Per obtenir la imatge utilitzem les funcions apropiades de la llibreria *apicamera* del Robotino.

Al principi del programa principal[H] es declaren totes les variables globals i constants. Per connectar amb la càmera necessitem el numero del port i la IP de la càmera, que anomenem SERVIDOR. Aquests dos valors són constants així que també han d'estar declarats al principi:

```
#define PORT 80
#define SERVIDOR "192.168.1.114"
```

Si volem guardar la imatge capturada, s'ha d'obrir el fitxer que anomenem fp, però no es necessari. Jo ho fet només per poder comprovar, quan vulgui, què veu el robot. Després es declara la imatge com a variable de tipus *imgmem* i es fa la petició utilitzant una de les dues funcions de la *apicamera*. En el nostre cas ho fem en format bmp i resolució baixa.

Per escollir la resolució hem agafat com a factor basic la optimització del temps. No ens importa tant la qualitat de la imatge, sempre i quan es pugui detectar la línia vermella, però si ens interessa tenir un temps de captura lo més petit possible per a que no rellenteixi el funcionament i el moviment del robot.

```
void main( void )
{
    FILE *fp;
    imgmem_t img;

    getBMP(SERVIDOR ,&img,LOW);
```

Figura 8.23.

Fent varies probes amb les tres diferents resolucions veiem com puja significativament el temps de captura i de processat com més alta es la resolució.

captura	processat	diftotal
223.965728	17.997246	241.962974
213.967258	17.997246	231.964504
221.966034	17.997246	239.963280
242.962821	17.997246	260.960067
213.967258	17.997246	231.964504
247.962056	17.997246	265.959302
219.966340	17.997246	237.963586
248.961903	17.997246	266.959149
463.929008	17.997246	481.926254
427.934516	17.997246	445.931762
209.967870	17.997246	227.965116
215.966952	18.997093	234.964045
253.961138	17.997246	271.958384
245.962362	17.997246	263.959608
209.967870	17.997246	227.965116

captura	processat	diftotal
875.865972	71.988984	947.854956
832.872551	71.988984	904.861535
833.872398	71.988984	905.861382
851.869644	71.988984	923.858628
832.872551	72.988831	905.861382
857.868726	71.988984	929.857710
824.873775	72.988831	897.862606
626.904069	72.988831	699.892900
889.863830	71.988984	961.852814
875.865972	71.988984	947.854956
832.872551	71.988984	904.861535
833.872398	71.988984	905.861382
824.873775	72.988831	897.862606
824.873775	72.988831	897.862606
842.871021	71.988984	914.860005

Figura 8.24: Temps de captura resolució LOW vs MEDIUM.

Comparem amb qualitat mitja perquè queda bastant clar que la gran es molt ineficient.

anem fins a un retard de quasi un segon només per fer la captura i el processat. Això significa que el robot corregirà un segon més tard l'error de posició. Depenent de la velocitat a la que es mogui, es pot preveure que aquesta correcció no serà gens eficaç. En canvi, amb resolució baixa es tarda aproximadament un quart de segon, 4 vegades més ràpid. Això optimitza moltíssim el temps de la tasca i ens permet poder moure el robot més ràpidament.

Finalment, si volem guardar l'arxiu, primer s'ha de crear aquest utilitzant la funció '*fopen*' seguit del nom que li volem donar i una '*w*' de write per indicar que hi volem escriure. Després passem les dades de la imatge del buffer a l'arxiu utilitzant '*fwrite*' i tanquem.

```
fp = fopen( "img.bmp", "w" );
if( fp == NULL ) {
    return;
}

 /* write image in a file */
 i = fwrite( img.buffer, sizeof( unsigned char ), img.tamany, fp );

 fclose( fp );
```

Figura 8.25.

7.2.1 CONDICIONS LUMÍNIQUES

Encara que no ho sembli, la llum es un factor molt important en aquest projecte i pot influir greument en el funcionament del robot, ja que aquest es guia per visió. El terra del laboratori és de terratzo micrograà i està format per rajoles de 40x40cm on s'hi aprecien els colors blanc, negre i gris de diferents tonalitats. Apart, es un terra molt brillant i reflecteix la llum incident. Per conseqüent, s'ha d'anar amb molta cura amb quines llums s'encenen. I no només això, a les imatges capturades es detecten petites variacions com per exemple tenir les persianes pujades o baixades, o encendre els focus centrals en comptes dels laterals.

Tot això afecta a la imatge capturada i més encara al posterior processat. Es podria intentar fer un processat d'imatge que s'adaptés a totes les possibles condicions lumíniques però no s'obtindria un resultat òptim i s'obtindria un codi més complex, fet que acabaria afectant al temps de processat. Per això, s'ha decidit establir unes condicions lumíniques concretes per intentar treure el millor processat d'imatge possible.

Després de varies probes s'han establert les següents condicions lumíniques:

- Persianes baixades però no del tot (una mica per sobre de la taula). Per deixar entrar una mica de llum.
- Llums apagades. Encara que es pot encendre les llums laterals esquerres, ja que no influeixen gaire, però les centrals i les laterals dretes es reflecteixen molt a la imatge.

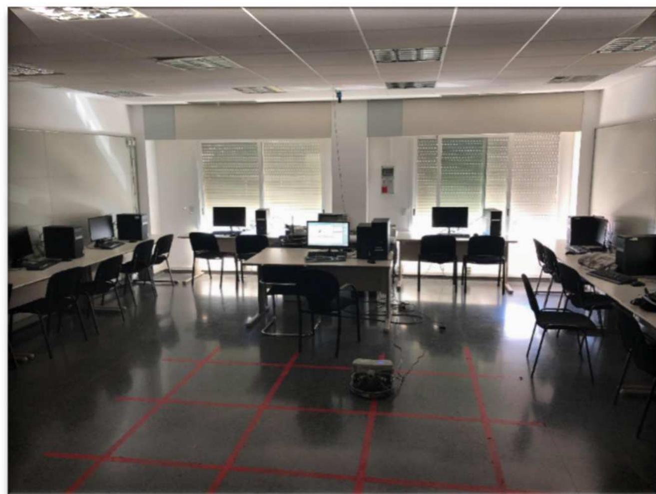


Figura 8.26: Condicions lumíniques del laboratori.

Encara tenint en compte aquestes precaucions, a vegades es detecten reflexes provinents dels fluorescents, de la llum solar o fins i tot de la llum del passadís que hi ha fora del laboratori.

7.3 TRACTAMENT DE LA IMATGE

El tractament de la imatge es pot fer indiferentment per als dos formats, i hi ha multituds de formes de fer-ho i totes son valides, sempre i quan quedi la recta ben definida.

En particular, jo he fet el següent procés:

```
getBMP(SERVIDOR ,&img,LOW);

capaCR(&img,&img2);
im2bw(&img2,&img3,46);
erode(&img3,&img4,7.5);
dilate(&img4,&img5,3);
```

Figura 8.27.

Després de capturar la imatge s'extreu la capa Cr del format YCbCr. Com s'ha explicat en l'apartat de les funcions de la llibreria '*apiimatge*', aquesta es la capa de rojos. Això es perquè la línia del terra és de color vermell.

Després es fa un bineritzat amb llindar 46. Aquest valor s'ha agafat després de fer nombroses probes i sempre fixant-me en el resultat final.

Per últim s'aplica una operació anomenada OBERTURA. Es tracta de fer una erosió seguida d'una dilatació. A priori, son dues operacions contraries per lo que hauria de donar el mateix resultat (com fer 1+1-1) però no és així. Les línies queden millor definides i al fer la erosió de forma vertical, les línies horitzontals desapareixen. Això es especialment bo per a quan el robot comença a capturar una cruïlla.

El resultat final és el següent:

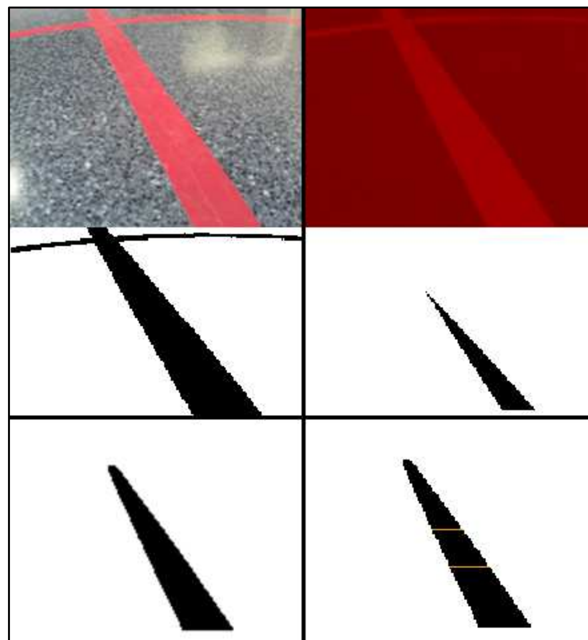


Figura 8.28: Comparació imatge original vs processada.

Encara que a la part superior de la imatge s'ha perdut part de la línia vertical, a la part inferior es conserva quasi perfectament. Amb això n'hi ha prou per poder obtenir els punts i distingir si estem a sobre de la línia o no, i quin angle de desviació té.

7.4 SEGUIMENT DE LINIA

Aquest es un dels punts més importants del programa. En principi, el projectista anterior[3] va crear un algorisme per a que el robot fos capaç de fer-ho, però després de provar-ho varis cops i implementar-ho al meu programa principal vam veure que no funcionava tant bé com esperàvem.

Aquest algorisme consistia en primer detectar la recta a la imatge amb la funció '*detectSegment*' creada per ell mateix. Aquesta funció torna les coordenades píxel (X,Y) de dos punts de la recta. Després, amb aquest punts, es calculava directament l'angle d'inclinació i amb un factor de proporcions passava la distancia de píxels a cm. A aquesta distancia li afegia un offset, que corresponia a la distancia del centre del robot al primer píxel de la imatge i amb un control proporcional controlava que la distancia del robot a la línia fos 0.

Al tractar-se d'un control proporcional, la velocitat estava compromesa amb aquest valor; si era una velocitat alta el robot es movia fent zig-zags i a vegades perdia la orientació ja que perdia completament de vista la línia vermella que seguia i es fixava en la línia transversal més pròxima.

Per evitar que això pugui passar amb el nostre algorisme, ens hem assegurat de deixar la imatge processada sense cap línia horitzontal gracies a la erosió i, per si de cas, d'agafar els dos punts a la part inferior de la imatge.

El seguiment de línia es bastant complex, i consta de varies parts:

7.4.1 OBTENCIÓ DELS PUNTS:

Com s'ha dit abans, per obtenir els dos punts necessaris per trobar la equació de la recta podríem haver utilitzat la funció *detectSegment* de l'anterior projectista, però no era gaire eficient a efectes de temps i no ens donava exactament el que buscàvem: que trobi dos punts centrals de la línia que surt a la imatge. Així doncs, hem procedit a crear la nostra pròpia funció: *detecPuntos*. Per veure més informació, anar al capítol 5.3.2.1.

A la funció se li han de subministrar les dues altures a les que volem que busqui els punts, la imatge bineritzada i les variables on volem que es guardin els valors de les coordenades dels punts en coordenades píxel (o imatge). Es preferible que siguin punts no molt propers i que es trobin en zones no conflictives (marges de la imatge).

Observant varies fotos i després de fer varies probes s'han escollit les altures 36 i 56 (35 i 55 tenint en compte que la primera altura es 0).

```
detectPuntos(&img5,35,55, &x1, &y1, &x2, &y2); // en coordenades imatge
```

Figura 8.29.

7.4.2 PUNTS IMATGE (PÍXELS) A PUNTS COORDENADES ROBOT

Per realitzar aquesta operació invoquem la funció *PimgToRobot* de la *apiimatge*, indicant les coordenades dels píxels com a la figura 8.30:

```
PimgToRobot(&x1,&y1, &x1r,&y1r);
PimgToRobot(&x2,&y2, &x2r,&y2r);
```

Figura 8.30.

A l'apartat 5.3.2.2 s'ha fet una introducció al seu funcionament però s'explicarà aquí amb detall ja que a l'apartat de la API encara no s'havien explicat varis conceptes clau que tenien a veure amb aquesta funció i que s'havien d'aclarir primer.

A l'apartat 9.3.3 s'ha explicat el procediment per obtenir aquest valors del calibratge. Aquí fem la continuació:

```
double A[3][3]= {   // R_3_3'* KK_inv

            { 0.0048,     0.0000,    -0.3531 },
            {-0.0001,    -0.0013,     1.0110},
            { 0.0001,    -0.0032,    -0.2001}};

double B [3]= { -4.4819,-33.7352,  -134.8877
};
```

Figura 8.31.

Partint de la introducció de les dades obtingudes del calibratge a la funció de la *apiimatge* (apartat 9.3.3), ens disposem a implementar la formula per passar els punts de coordenades imatge a coordenades píxels. Recordem que s'havia simplificat la formula fins tenir:

$$P_{rob} = (R_3_3' * KK_inv * \lambda * P_{img}) - R_3_3' * t$$

On després hem anomenat **A=R_3_3' *KK_inv** i **B= R_3_3'*t**. Llavors, la formula ens queda com:

$$P_{rob} = (A * \lambda * P_{img}) - B$$

A i B ja les tenim i, sempre i quan no es mogui la càmera, valdran el mateix. Per contra, s'ha de calcular Landa per a cada punt que es vulgui transformar. Així doncs, s'ha de deixar escrita una equació en la que el seu valor estigui en funció del punt introduït.

```
int P_img[3]={xi,yi,1};

  //calculo del denominador  denom=A*P_img;
 int  i,j;
double temporal,Denom[3];

for (i = 0 ; i < 3 ; i++ ) //i para las filas de la matriz resultante
{
  // como solo hay 1 columna no hace falta for para las columnas de la matriz resultante
        temporal=0;
        for (j=0;j<3;j++){

            temporal=temporal + (A[i][j] * P_img[j]);
        }
        Denom[i] = temporal;
}
```

Figura 8.32.

Totes les matrius i vectors han de tenir una correspondència de dimensions per a que sigui possible fer operacions entre elles .El nombre de columnes del primer element ha de coincidir amb el nombre de files del segon. La matriu A és (3x3) i el vector B és (3x1). Si, per exemple, volem fer una operació entre la matriu A i el vector P_{img}, aquest últim haurà de complir aquesta condició tenint 3 files (3xN).

Així doncs, guardem els punts del píxel en un vector de 3 elements on el 3r te valor unitari per no alterar la posició, i procedim al càlcul de Landa (λ).

Tornant al capítol 7.3.4, recuperem la equació per calcular-la:

$$\lambda = \frac{[R^T * t]}{[R^T * K^{-1} * p^{imatge}]} \quad (3)$$

És tracta de una divisió entre matrius i, com a QNX tenim llibreria matemàtica però no disposa de funcions per fer operacions entre matrius, les hem de programar nosaltres mateixos a mà. Per facilitar el càlcul separarem numerador i denominador, farem el càlcul i finalment es farà la divisió.

Primer procedim al càlcul del denominador: $A*P_{img}$. Com es tracta d'una matriu 3x3 per un vector 3x1 , fem un bucle for extern que servirà per calcular els tres elements del vector resultant. Desprès, internament, fem un altre bucle for que s'encarrega d'anar fent la multiplicació i la posterior suma dels elements que es guardaran en cada fila del vector resultant.

Una vegada tenim el denominador procedim directament al càlcul de landa ja que el numerador no fa falta. Es tracta del vector B obtingut al calibratge. Ho guardem en una matriu *double* ja que lo més probable es que s'obtingui un resultat amb decimals.

```
double landa;

landa= B[2]/Denom[2];
```

Figura 8.33.

L'índex dels vectors es 2 perquè en llenguatge C els índexs comencen en 0. Llavors, 2 correspon al 3r element. Ja tenim A,B i Landa (λ).Ara procedim a fer el càlcul del punt en coordenades robot:

```
//calculo del punto: P_rob= (A*landa*P_img)-B = Denom*landa-B

double P_rob[3];

for (i=0;i<3;i++){
    P_rob[i]= Denom[i]*landa - B[i];
}


*xr= P_rob[0];
*yr=P_rob[1];
```

Figura 8.34.

També es declara com a vector de 3 components. Es programa la operació en un bucle for perquè es tracta de vectors i només es una fila. Si fossin matrius faria falta un segon for, com s'ha fet abans.

Després d'acabar el càlcul retornem les coordenades obtingudes en les variables indicades per l'usuari al invocar la funció.

7.4.3 Càlcul de l'angle i distancies.

Es podria haver fet una funció per aquest càlcul però hem preferit fer-ho directament al programa principal. Necessitem l'angle de desviació i la distancia a la recta (figura 8.35) per poder aplicar l'algorisme de seguiment de linia Pure-Pursuit explicat en el pròxim apartat.

D'assignatures de matemàtiques sabem que la tangent d'una recta ens serveix per relacionar l'angle que forma aquesta amb la vertical i la distancia entre dos punts qualsevols que la formen:

$$\tan(\alpha) = \frac{\Delta x}{\Delta y}, \quad llavors \quad \propto = \tan^{-1}\frac{\Delta x}{\Delta y}$$

Apliquem aquesta equació al codi:

```
// calcul de l'angle de la recta respecte el robot
    int Ax,Ay;
    Ay= y2r - y1r;
    Ax= x2r - x1r;
    phi= atan((double)Ax/(double) Ay)*180/3,14159;
```

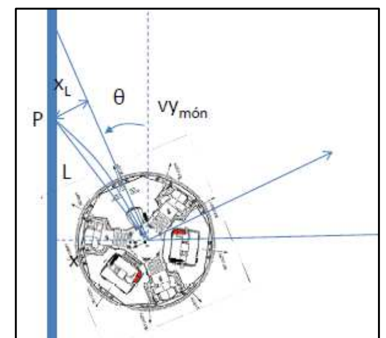


Figura 8.35.

Obtenim l'angle en radians així que fem conversió per obtenir-ho en graus. Una vegada tenim l'angle, ara falta determinar la equació de la recta per poder implantar la equació que ens dona la distancia entre un punt i una recta.

Per obtenir la equació de la recta a partir de dos punts, s'ha de fer el següent càlcul però només ens interessa la equació final:

$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{y - y_1}{x - x_1} \rightarrow (x - x_1) * (y_2 - y_1) = (y - y_1) * (x_2 - x_1)$$

$$x * y_2 - x * y_1 - x_1 * y_2 + x_1 * y_1 = y * x_2 - y * x_1 - y_1 * x_2 + y_1 * x_1$$

Els paràmetres destacats en vermell es simplifiquen. Agrupant per variables i traient factor comú obtenim:

$$x(y_2 - y_1) + y(x_1 - x_2) + [y_1 * x_2 - x_1 * y_2] = 0$$

$$Bx + Ay + C = 0$$

Ja tenim la equació de la recta! On:

- A=$(x_1 - x_2)$
- B=$(y_2 - y_1)$
- C=$[y_1 * x_2 - x_1 * y_2]$

A i B estan invertits perquè estem fent càlculs amb coordenades robots i els eixos X i Y també ho estan. A continuació tenim la implementació en el codi:

```
//  paremtros equacion de la recta Ax + By + C
A=(double)(x1r-x2r);
B=(double)(y2r-y1r);
C=(double)(y1r*x2r - y2r*x1r);
```

Figura 8.36.

Ara que tenim totes les dades necessàries, implementem la equació que ens dona la distancia entre un punt qualsevol i la recta.

$$distancia = \frac{\text{Ax} + \text{By} + \text{C}}{\sqrt{A^2 + B^2}}$$

Agafem com punt a mesurar el centre del Robotino, que coincideix amb l'origen de coordenades (0,0). Llavors queda com:

$$distancia = \frac{\text{C}}{\sqrt{A^2 + B^2}}$$

```
// eq. distancia de punto a una recta. punto es origen asi que Ax=By=0.
    a2=pow(A,2);
    b2=pow(B,2);
    dist=C/sqrt(a2+b2);
```

Figura 8.37.

7.4.4 PURE-PURSUIT I COMPOSICIÓ DE LES VELOCITATS

Ara ja només ens falta el control de trajectòria. Per fer-ho, hem implementat una estratègia de seguiment de línia anomenada Pure-Pursuit. EL seu funcionament esta explicat en detall en l'article escrit per Aníbal Ollero[F].

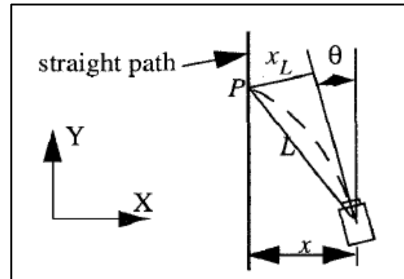


Figura 8.38.

A la figura 8.37. podem apreciar totes les variables que necessitem per poder implementar aquesta estratègia de seguiment de línia, que es resum en una simple equació:

$$\gamma_R = \frac{2}{L^2} * [x * \cos(\theta) - \sqrt{L^2 - x^2} * \sin(\theta)]$$

Es tracta de calculat el radi de curvatura (γ_R) que es la inversa del radi de la circumferència amb centre desconegut que descriu el robot al corregir la seva posició i apropar-se a la recta.

Aquesta equació depèn de la distancia del centre del robot a la línia a l'eix X (eix Y per nosaltres), l'angle de desviació (θ) i L. Aquest últim paràmetre es la distancia que volem que el robot mantingui amb la recta a l'hora de calcular el radi de curvatura. Desprès de fer un estudi d'estabilitat es va concloure que el paràmetre L ha de ser major o igual que 1.

```
    //FORMULA OLLERO
dist2=pow(dist,2);
inv_r=(dist*cos(phi)-sin(phi)*sqrt(L*L-dist2))*2/(L*L);
```

Figura 8.39.

Ara que ja tenim la inversa del radi de curvatura, només ens queda obtenir les consignes de velocitat:

També sabem que la velocitat angular es troba a partir de la relació entre la velocitat linear i el radi que es vol descriure. Com hem dit abans, γ_R és la inversa del radi de curvatura, per tant:

$$\omega = \frac{v}{r} = v * \gamma_R$$

Ens falta escollit una velocitat lineal d'avanç. Després de fer varies probes s'ha decidit posar una velocitat de 2 *cm/segon*. Les consignes de velocitat s'apliquen al robot en *mm/s* per a les components lineals i *graus/s* per a la velocitat angular. S'haurà de fer una conversió ja que amb la formula anterior s'obté *radiants/s*.

```
w=20*inv_r;

        //Composicio de les consignes de velocitat del robot
        robotSP0[0]=20;
        robotSP0[1]= 0;
        robotSP0[2]= w*180/pi;
```

Figura 8.40.

7.5 LLIBRERIA DE PATHFINDING

Aquesta llibreria ha sigut creada per agrupar totes les funcions de Pathfinding i poder invocar-les sempre que es vulgui durant la execució del programa. En el nostre cas, només les invoquem al principi però, si algun futur projectista les vol utilitzar per fer un re-càlcul de ruta al detectar un obstacle, les podrà tornar a invocar en qualsevol moment.

```
typedef enum {A,B,C,D,E,F,G,H,I,Z,Y} PUNTOS;
```

Figura 8.41.

Per a questa llibreria s'ha creat un nou tipus de variable anomenat PUNTOS. Es tracta d'una enumeració on s'assigna a cada lletra un valor que coincideix amb el seu índex. La primera lletra (A) tindrà el valor de 0, B=1,C=2, etc. A la *figura 8.41* tenim la declaració del tipus.

Les variables globals que s'utilitzen a totes les funcions o es poden compartir, les definim al principi de la llibreria. A la *figura 8.42* veiem que s'ha definit *camino[]* amb un valor de 15 posicions. S'ha escollit aquest nombre per assegurar que no hi hauran suficients indicacions per omplir-lo. *Ruta[]* s'ha declarat amb 9 posicions per que el màxim de punts pels que podria passar per fer una ruta son nou. Seria el cas en que passes per tots els punts, però això no passarà mai.

```
PUNTOS origin,target,poss;
PUNTOS camino[15];
PUNTOS ruta[9];
```

Figura 8.42.

Aquesta llibreria consta de 3 funcions:

- Greedy: 1ª funció de pathfinding.
- Dijkstra: 2º funció, més optima.
- puntosAcamino: Funció per passar de punts (nodes) a indicacions de camí.

7.5.1 EMPLENAMENT DE LA MATRIU DE PESOS.

Aquest mètode d'emplenament de la matriu de pesos pels algorismes de pathfinding va ser dissenyat després d'observar algunes incoherències en els pesos de la matriu original utilitzada per els exemples en l'apartat 7.2. Llavors es va decidir fer-ho seguint un criteri vàlid.

Normalment al llarg de la memòria s'ha definit sempre el node A com a inici i el node I com a destí, però no té perquè ser així. Només es fa per comoditat i es pot escollir qualsevol altre combinació de punts.

Entre el punt origen A i el punt objectiu I tenim múltiples rutes possibles, però les úniques lògiques són:

- ABCFI
- ABEFI
- ABEHI
- ADEFI
- ADEHI
- ADGHI

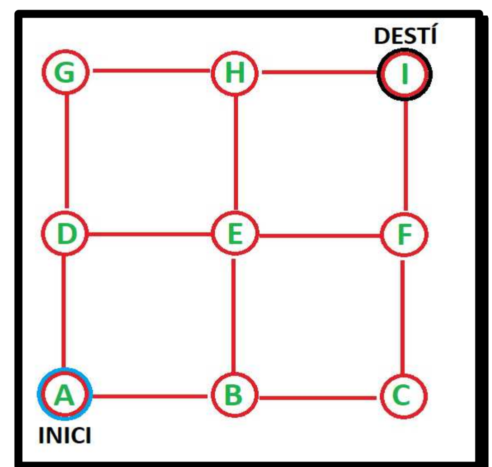


Figura 8.43: Representació del circuit

Per evitar qualsevol de les combinacions il·lògiques, aplicarem pesos alts als camins que els farien possibles.

Per poder representar aquests camins lògics en la matriu de pesos s'intentarà reflectir la rapidesa amb la que el robot podrà arribar al punt *target*. La distancia total recorreguda acabarà sent la mateixa per a tots els recorreguts, però es evident que el temps que es trigarà en girar també influeix en el temps total. Per això, identificarem com a camins ràpids aquells en els que s'hagi de fer un menor nombre de girs, i camins lents en els que s'hagin de fer més.

Així doncs:

- ABCFI: Té 2 girs a la ruta.
- ABEFI: Té 4 girs.
- ABEHI: Té 3 girs a la ruta més un al node final per a orientar-se.
- ADEFI: Té dos girs.
- ADEHI: Té 3 girs a la ruta més un al node final per a orientar-se.
- ADGHI: Té 1 gir a la ruta més un al node final per a orientar-se.

Aparentment, les rutes més optimes son ABCFI, ADGHI i ABEFI. De entre aquestes tres considero ABEFI la perquè conté un gir en mig de l'avanç i això farà retardar al robot en el seu desplaçament.

Desprès d'assignar valors en mode Sudoku per tal de que segons l'algorisme que s'apliqui s'esculli un camí o un altre, obtenim la següent relació de pesos:

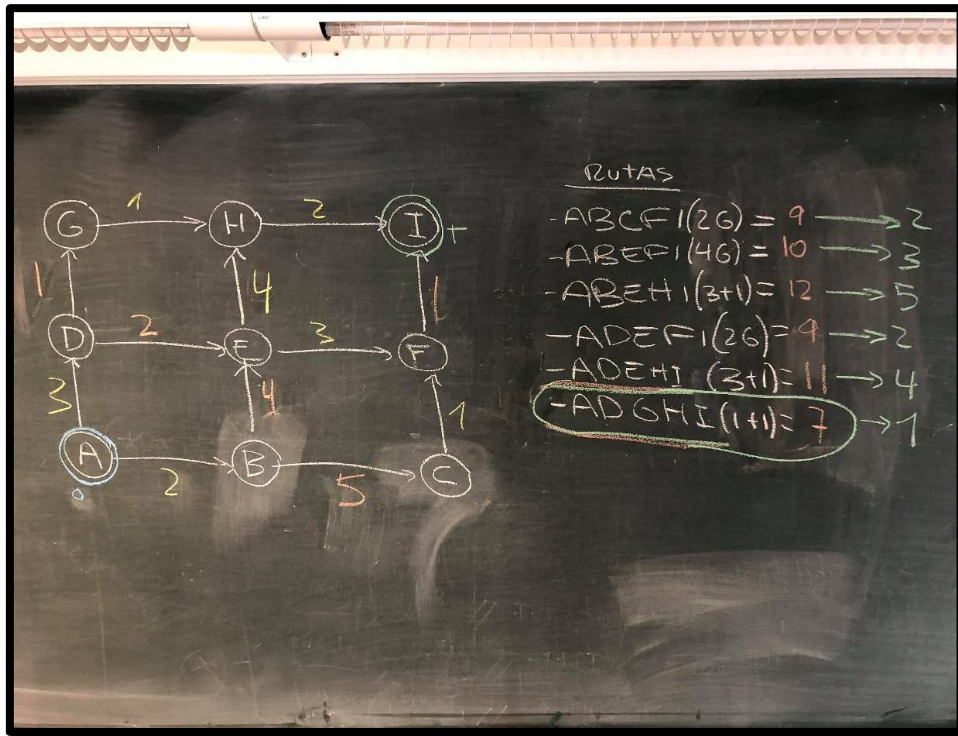


Figura 8.44: Representació a la pissarra del resultat obtingut.

ADGHI serà la ruta més optima. Això vol dir que aquesta hauria de ser el resultat de l'algorisme Dijkstra.

Encara que ABEFI es el 4t millor camí (o el 3r pitjor), com l'algorisme Greedy sempre es queda amb el camí que 'veu primer' , aquesta serà la seva solució.

DE/A	A	B	C	D	E	F	G	H	I
A	∞	2	∞	3	∞	∞	∞	∞	∞
B		∞	5	∞	4	∞	∞	∞	∞
C	∞		∞	∞	∞	1	∞	∞	∞
D		∞	∞	∞	2	∞	1	∞	∞
E	∞		∞		∞	3	∞	4	∞
F	∞	∞		∞		∞	∞	∞	1
G	∞	∞	∞		∞	∞	∞	1	∞
H	∞	∞	∞	∞		∞		∞	2
I	∞	∞	∞	∞	∞		∞		∞

Ara queda emplenat les caselles de color marró. En aquesta aplicació les emplenarem amb valors alts perquè son els camins que ens porten a rutes il·lògiques que no ens interessen. Escullo el valor ramdom 9.

Finalment ens quedarà així:

DE/A	A	B	C	D	E	F	G	H	I
A	∞	2	∞	3	∞	∞	∞	∞	∞
B	9	∞	5	∞	4	∞	∞	∞	∞
C	∞	9	∞	∞	∞	1	∞	∞	∞
D	9	∞	∞	∞	2	∞	1	∞	∞
E	∞	9	∞	9	∞	3	∞	4	∞
F	∞	∞	9	∞	9	∞	∞	∞	1
G	∞	∞	∞	9	∞	∞	∞	1	∞
H	∞	∞	∞	∞	9	∞	9	∞	2
I	∞	∞	∞	∞	∞	9	∞	9	∞

7.5.2 Greedy

A l'hora d'invocar la funció se li ha de passar tres paràmetres del tipus PUNTOS: El punt inicial (*origin*), el punt final o destí on volem que vagi el robot (*target*) i el vector on volem que es guardi la ruta, es a dir, la successió de nodes pels que el robot haurà de passar per anar de l'origen al destí.

```
void  Greedy (PUNTOS origin, PUNTOS target, PUNTOS * ruta)
{
```

Figura 8.45.

Després, ens trobem amb la declaració de la matriu de weightpoints. Lo normal hauria sigut declarar-la junt amb les variables globals ja que es una variable comuna per a ambdós algorismes de pathfinding però, no se perquè, al declarar-la així la funció no la reconeixia. Així que s'ha agut de declarar dos cops, un dins de cada algorisme (*Figura 8.46*) :

```
                        /*      A  B  C  D  E  F   G  H  I   */

int WP[9][9] = {              {0, 2, 0, 3, 0, 0,  0, 0, 0},  // A
                              {9, 0, 5, 0, 4, 0,  0, 0, 0},  // B
                              {0, 9, 0, 0, 0, 1,  0, 0, 0},  // C
                              {9, 0, 0, 0, 2, 0,  1, 0, 0},  // D
                              {0, 8, 0, 9, 0, 3,  0, 4, 0},  // E
                              {0, 0, 9, 0, 9, 0,  0, 0, 1},  // F
                              {0, 0, 0, 9, 0, 0,  0, 1, 0},  // G
                              {0, 0, 0, 0, 9, 0,  1, 0, 2},  // H
                              {0, 0, 0, 0, 0, 8,  0, 7, 0}} ; // I
 /*****************************************************************/
```

Figura 8.46.

Per no utilitzar infinits, s'han posat zeros a totes les connexions inexistents. Això ens ha donat com a resultat una matriu amb diagonal de zeros ja que no es pot anar a un punt des del mateix. Els camins estan definits de forma que el node origen esta en vertical i el destí en horitzontal. Així docs, si volem saber el pes del camí que va de E a F, hem de buscar E en la vertical, i la F en la horitzontal, trobant el pes que val 3.

A la *figura 8.45* tenim la inicialització dels paràmetres principals:

- S'inicialitza el pes total del camí a 0.
- Registrem la posició actual.
- Inicialitzem l'índex per emplenar el vector de la ruta a 1 perquè ja tenim el punt inicial guardat. Així ens assegurem que no es sobreescriurà al calcular el proper punt.

```
int totalweight=0;

ruta[0]=origin;
poss=origin;
int j=1;
```

Figura 8.47.

Dins del bucle for principal, primer s'inicialitza l'índex per recórrer la matriu de weightpoint de forma horitzontal en busca del camí amb el mínim pes des de l'últim node registrat. *K* es el valor que es vol minimitzar, llavors s'inicialitza amb un valor més alt que el màxim que podem trobar a la matriu.
Per a cada posició de la matriu, es compara el seu valor amb *K* i només s'accepta el punt si el seu weightpoint es menor que aquest i major a 0 per descartar connexions inexistents.

```
int i,f=0, k=1000;
    for (i=0;i<9;i++){

        if ((WP[poss][i]<k) && (WP[poss][i]>0)) {
        k = WP[poss][i];
        f=i;
        }
```

Figura 8.48.

Cada vegada que es detecta un camí amb valor mínim, es guarda el seu weightpoint en *K* per actualitzar-la i la seva posició en *f* : *Figura 8.48.*

Una vegada inspeccionades les 9 posicions horitzontals, es guarda l'índex del camí seleccionat, que coincideix amb l'índex del node, i sumem el seu pes al pes total registrat fins ara. Després s'actualitza l'índex de la posició actual i del vector *ruta[]* (*Figura 8.49):*

```
ruta[j]=f;
totalweight=totalweight+k;
j=j+1;
poss=f;
```

Figura 8.49.

Tot això esta dintre d'un bucle while que esta operatiu mentre que la posició actual es diferent a la posició final que busquem:

```
while (poss!= target){
```

Figura 8.50.

Una vegada la posició actual coincideix amb el node *target*, se surt del bucle i s'assigna a la ultima posició una Z, que ens serveix com a indicador de final de ruta:

```
ruta[j]=Z;
```

Figura 8.51.

7.5.3 Dijkstra

Aquesta funció es una adaptació de la funció creada per Jorge Barrera[6] per ser utilitzada amb Matlab. A Matlab, es disposa de moltes funcions de càlcul matemàtiques que simplifiquen molt el treball i que per treballar amb C a QNX no tenim. Per això, més que una adaptació diria que m'ha servit de base per a crear la meva funció.

Abans d'explicar el codi, faré un petit resum del funcionament perquè es una mica complex i costa d'entendre:

Recordem que bàsicament, el Dijkstra treballa igual que el Greedy però amb la diferencia de que en comptes d'observar només el pes de cada camí, observa el pes total des del node inicial al node en qüestió. Així doncs, el que es fa en aquesta funció es crear dos vectors que ens guarden la informació del millor node anterior per a cada node (*prev[]*) i el pes total mínim del node inicial a aquests nodes (*dist[]*). Per emplenar-los, el que es fa es analitzar, node per node, a quins altres nodes li es el millor node anterior. Una vegada emplenats els vectors, es procedeix a omplir el vector de *ruta[]* des del node final fins al inicial. S'emplena al revés.

```
void Dijkstra (PUNTOS origin, PUNTOS target, PUNTOS * ruta)
{
```

Figura 8.52.

A la hora de cridar la funció, els paràmetres necessaris son els mateixos que per la funció de l'algorisme Greedy.

Dins de la funció, trobem declarada la matriu de weightpoints (*Figura 8.46.*) seguida dels següents paràmetres (*Figura 8.53.*):

```
int n=9;  //n=size(WP,1);

int S[n];
int dist[n];
int prev[n];
int h;
```

Figura 8.53.

- *n* és un índex que s'inicialitza amb el valor de la dimensió de la matriu de weightpoints, i serveix per establir les dimensions de les demès variables.
- *S[]* és un vector que guarda la informació dels nodes visitats.
- *dist[]* és un vector que emmagatzema la distancia més curta (menor pes) entre el node *origin* i qualsevol altre node.
- *prev[]* és un vector que emmagatzema el millor node previ conegut per anar a cada node de la xarxa.
- *h* s'utilitza com a índex en alguns bucles *for*'s.

```
for (h=0;h<n;h++){

    S[h] = 0;
    dist[h]= 1000;
    prev[h] = n+1;
}
```

Figura 8.54.

Aquest primer bucle for (*Figura 8.54.*) serveix per inicialitzar els valors de les variables abans declarades. Després (*Figura 8.55.*) , es posa a zero la posició del node inicial al vector de distàncies per mostrar que es on estem, i es declara un altre punter pel bucle for i una variable que ens indicarà quants nodes hem visitat.

```
dist[origin] = 0;
int i,nodvisit=0;
```

Figura 8.55.

Aquí comença el bucle *do-while* amb condició al final.

```
}while (nodvisit!=n);
```

Figura 8.56.

A dins trobem tres operacions diferents i s'executen mentre que el node visitat no coincideixi amb el node destí:

- 1r: Es declara el vector de candidats. Cada vegada que es canviï de node, aquest vector es reseteja. Després tenim el primer bucle *for* : si l'índex correspon a un node no visitat prèviament, es a dir, S[i]=0, es guarda el valor de la seva distancia en el vector de candidats. Si no, se li assigna el valor de 1000, que simbolitza l'infinit, per mostrar que no es troba entre els nodes candidats.

```
int  candidate[n];

  for (i=0;i<n;i++){
      if (S[i]==0) candidate[i]=dist[i];
      else candidate[i]=1000;

  }
```

Figura 8.57.

- 2n: En aquest segon bucle el que es fa es obtenir l'índex de posició de l'últim node seleccionat. Per a la primera vegada es absurd ja que coincideix amb el node origen, però per després és útil.

```
int u,ind=1000;
   for (h=0;h<n;h++){
    if (candidate[h] < ind) {
        u = h;
        ind = candidate[h];
      }
   }
```

Figura 8.58.

- 3r: El funcionament d'aquest bucle es semblant al bucle *for* de l'algorisme Greedy, amb la diferencia de que aquí s'agafen els valors dels camins possibles des del node en qüestió i es va sobreescrivint fins a obtenir els valors mínims.

```
for (h=0;h<n;h++){
    if(((dist[u]+WP[u][h])<dist[h])&&(WP[u][h]>0)){

        dist[h]=dist[u]+WP[u][h];
        prev[h]=u;
    }
}
```

Figura 8.59.

Aquí finalitza el while. De forma simplificada, entre els dos últims bucles el que es fa es analitzar per a cada node, a quin altre node li és la millor opció com a node anterior. Això es guarda en el vector *prev[]* i el seu pes a *dist[]*, que es van omplint poc a poc fins a acabar de visitar tots els nodes.

Una vegada tenim els vectors emplenats, només falta veure quin camí s'escull. Es comença assignant una 'Z' a la ultima posició, que ens serveix com indicador de final de ruta. Després es comença a emplenar pel node final (*target*) i es va seleccionant, un per un, tots els millors nodes anteriors del vector *prev[]* fins a arribar al node origen.

```
ruta[n]=Z;  //rang 0-9.
ruta[n-1]=target; //10 posicions


do{
    if (prev[ruta[n-g]]<=n)
        ruta[n-g-1]=prev[ruta[n-g]];
        g++;
}while (ruta[n-g]!= origin);


int totalweight = dist[target];
```

Figura 8.60.

Executant la funció per calcular la ruta entre el node A i el I obtindríem:

```
>> prev
prev =
     0     1     2     1     4     5     4     7     6
>> dist
dist =
     0     3     6     4     5     7     6     7     9
```

Figura 8.61.

S'ha de tenir en compte que ha estat calculat amb Matlab i els índexs dels nodes van de 1-9 en comptes de 0-8). Llavors la seqüencia seria la següent: comencem al node I (posició 9 del vector) i veiem que el millor node anterior es el 6(F) amb un pes total de 9. Anem a la posició 6 del vector i veiem que el millor anterior es el 5(E) amb pes total de (7). Així successivament fins arribar al node A, que no te precedents perquè es l'inicial.

Finalment obtindríem el camí ADEFI amb pes total de 9, com a l'exemple de l'apartat 7.2.2.

7.6 PROGRAMA PRINCIPAL

Aquest programa necessita que es facin varies activitats simultàniament, per això s'han implementat diverses tasques que col·laboren i comparteixen informació entre elles per tal de poder fer al robot Robotino fer el que se li demana.

La comunicació entre les tasques es fa per mitjà d'esdeveniments enviats a cues i compartint informació en variables globals. A la hora d'utilitzar recursos compartits, existeix el perill de que una tasca vulgui llegir la informació d'una variable quan aquesta esta sent manipulada per una altra tasca, llavors es llegiria una informació errònia. Per evitar això, es protegeixen els recursos compartits amb un *mutex*, que serveix per bloquejar el accés a la CPU de qualsevol altre tasca fins que la tasca que l'ha bloquejat el desbloquegi.

Els tipus d'esdeveniments que hi ha son els següents, i es poden veure a la *Figura 8.62*:

```
// Tipus d'esdeveniments
typedef enum {START,STOP,QUIT,COLLISION,END_MOV} EVENT_CONTROL;
```

Figura 8.62.

- START,STOP I QUIT: Son esdeveniments creats per la interacció de l'usuari amb el programa.
- COLLISION: Generat quan es detecta una col·lisió al bumper que rodeja el robot Robotino.
- END_MOV: Generat per la tasca Monitor sempre que es finalitza un moviment.

Al principi del programa, tenim la declaració dels paràmetres constants, de les prioritats de cada tasca i dels estats de cadascuna d'elles. Hi ha 256 nivells de prioritats i com més alt sigui el nombre, més alta és la prioritat.

```
#define PRIO_SUP 9
#define PRIO_EVE 8
#define PRIO_MON 7
#define PRIO_CON 6
#define PRIO_GUI 5
#define PRIO_SHO 4
```

Figura 8.63: Prioritats de les tasques.

Els modes de treball també es comparteixen en variables globals ja que és la tasca supervisora la que dicta en quin mode han de treballar les altres tasques. Una altre mesura que s'utilitza per protegir les variables globals es copiar els seus valors en variables locals, treballar amb aquestes i, si en algun punt de la tasca es vol manipular el valor de la variable global, es bloqueja el *mutex* i es fa el traspàs.

7.6.1 Tasca Supervisor

Dins d'aquesta tasca primer es fa la recepció dels esdeveniments i la obtenció del mode en que ha d'operar. Aquesta es la tasca amb major prioritat d'accés als recursos i es dicta a si mateix quan ha de canviar de mode i a quin mode. Com el mode també es una variable global la hem de modificar bloquejant el *mutex* quan la llegim, però realment no fa falta ja que només es modifica dins d'aquesta tasca.

```
mq_receive(cua_control, (char *)&eventControl, sizeof(EVENT_CONTROL), NULL);

pthread_mutex_lock(&m);
modeSUP0=modeSUP;
pthread_mutex_lock(&m);
```

Figura 8.64.

Una vegada llegit el mode de treball, tenim un switch case. Depenent del mode llegit i l'esdeveniment rebut, les altres tasques funcionaran d'una forma o una altra.

```
switch (modeSUP0) {
```

Figura 8.65: Switch case.

Per a la tasca Supervisor tenim els següents modes:

```
//Estats de la tasca Supervisor
typedef enum {IDLE,MOVEMENT,PAUSED,SHUTDOWN} MODESUP_T;
volatile MODESUP_T modeSUP = IDLE;
```

Figura 8.66: Estats de la tasca Supervisor.

- IDLE:

Aquest estat només s'utilitza per inicialitzar. Una vegada començat el moviment no es torna mai més. Depenent de l'esdeveniment rebut, s'inicialitza el moviment o s'interromp directament la execució del programa. *Camino[]* és el vector que conté les indicacions del camí a realitzar per arribar al punt *target* i es necessari llegir-lo per començar.

```
case IDLE:
    if (eventControl==START){
        path=1;
        rest=1;
        modeSUP0=MOVEMENT;
        modeMON0=MONITOR_ON;
        getOdometry(&ODO_INICIAL0[0],&ODO_INICIAL0[1],&ODO_INICIAL0[2]);

        if (camino[j]=='S'){
            modeCON0=FORWARD;
        }
        else if (camino[j]=='R'){
            modeCON0=TURN_RIGHT;
        }
        else if (camino[j]=='L'){
            modeCON0=TURN_LEFT;
        }
    } else if (eventControl==QUIT) {        // SI SE DETECTA COLISION, SE
        pthread_mutex_lock(&m);
        modeCON=OFF;
        modeSUP=SHUTDOWN;
        modeMON=MONITOR_OFF;
        pthread_mutex_unlock(&m);
        pthread_exit(NULL);
    }

break;
```

Figura 8.67.

- MOVEMENT:

Aquest mode esta actiu sempre que el robot es trobi en moviment. La primera activació es fa a través del mode IDLE amb la intervenció de l'usuari al introduir un START(1) per pantalla. Una vegada s'ha acabat el primer moviment i es rep un END_MOV, el switch case entra en aquest mode. Dins tenim diferents 'sub-modes' que depenen del valor del vector *camino[]* i es van canviant fins a completar el recorregut.

```
case MOVEMENT:
    if (eventControl==STOP){
        modeSUP0=PAUSED;
        modeACT=modeCON0;
        modeCON0=OFF;
        modeMON0=MONITOR_OFF;
    } if (eventControl==END_MOV){

        getOdometry(&ODO_INICIAL0[0],&ODO_INICIAL0[1],&ODO_INICIAL0[2]);
        j=j+1; // atualització del punter
        if (camino[j]=='S'){
            modeCON0=FORWARD;
            modeMON0=MONITOR_ON;
        }
        else if (camino[j]=='R'){
            modeCON0=TURN_RIGHT;

        }
        else if (camino[j]=='L'){
            modeCON0=TURN_LEFT;

        }
        else if (camino[j]=='Z'){
            pthread_mutex_lock(&m);
                modeCON=OFF;
                modeSUP=SHUTDOWN;
                modeMON=MONITOR_OFF;
            pthread_mutex_unlock(&m);

            // PASSAR A SHOWGUI
            //MISSATGE PER PANTALLA INDICANT FI DE LA SEQUENCIA:
            printf("\033[24;20f SECUENCIA ACABADA. \033[25;15f  ROBOT EN EL DESTINO INDICADO. ");
            printf("\033[23;15f ******************************* \033[26;15f ******************************* ");

            pthread_exit(NULL);

        }
```

Figura 8.68.

Finalment tenim l'últim sub-mode que només entra en acció si es detecta una col·lisió en el bumper.

```
} else if (eventControl==COLLISION) { // SI SE DETECTA COLISION
    pthread_mutex_lock(&m);
    modeCON=OFF;
    modeSUP=SHUTDOWN;
    modeMON=MONITOR_OFF;
    pthread_mutex_unlock(&m);
    pthread_exit(NULL);
```

Figura 8.69.

NOTA: L'usuari no pot enviar directament un QUIT. Si vol avortar el programa primer ha de fer un STOP per parar-ho tot i després fer el QUIT.

- PAUSED:

Aquest mode també entra en acció per la interacció de l'usuari enviant un STOP (2), i pausa tant la execució del programa com el moviment del robot. Roman actiu fins que l'usuari torna a enviar un START o avorta l'execució del programa quan rep un QUIT.

Utilitza la variable modeACT de tipus modeSUP per retornar les condicions d'execució que hi havien abans de fer la pausa.

```
case PAUSED:
    if (eventControl==START){
        modeSUP0=MOVEMENT;
        modeMON0=MONITOR_ON;
        modeCON0=modeACT; //modeACT = FORWARD/TURN_RIGHT/TURN_LEFT
    } else if (eventControl==QUIT) { // SI SE METE POR PANTALLA UN 3
        pthread_mutex_lock(&m);
        modeCON=OFF;
        modeSUP=SHUTDOWN;
        modeMON=MONITOR_OFF;
        pthread_mutex_unlock(&m);
        pthread_exit(NULL);
    }
break;
```

Figura 8.70.

- SHUTDOWN:

Aquest mode pot ser activat per diverses causes i resulta en la suspensió de l'execució del programa. Pot ser activat per l'usuari al enviar un QUIT (3) després de fer un STOP, al finalitzar la seqüencia de moviment o al detectar una col·lisió en el bumper. Aquest mode no te representació en el switch case ja que no es fa res més després d'activar-lo. Es trenca directament el thread i es tanca el programa.

Una vegada s'ha seleccionat el mode de treball del Supervisor i s'han dictat els modes de treball de les demés tasques, es passa la informació a les variables globals. També es passa la odometria mesurada a l'inici de cada moviment per fer les comparacions pertinents a la tasca Monitor.

```
pthread_mutex_lock(&m);
    ODO_INICIAL[0]=ODO_INICIAL0[0];
    ODO_INICIAL[1]=ODO_INICIAL0[1];
    ODO_INICIAL[2]=ODO_INICIAL0[2];
    modeMON=modeMON0;
    modeCON=modeCON0;
    modeSUP=modeSUP0;
pthread_mutex_unlock(&m);
```

Figura 8.71.

Tota la tasca es troba dintre d'un bucle *while(1)* que s'hauria d'executar infinitament però dins s'han posat intervals de temps per deixar que les altres tasques puguin accedit a la CPU per aplicar les comandes fetes.

7.6.2 Tasca Controller:

Per aquesta tasca tenim els següents modes de treball:

```
//Estats de la tasca Controller
typedef enum {FORWARD,TURN_RIGHT,TURN_LEFT,OFF} MODECON_T;
volatile MODECON_T modeCON = OFF;
```

Figura 8.72: Estats de la tasca Controller.

- OFF:

Esta actiu quan el Robotino es troba parat, a la espera d'una ordre o si, per la raó que sigui, es demana que es pari. Consisteix en posat totes les consignes de velocitat a zero.

```
case OFF:
    robotSP0[0]=0.0;
    robotSP0[1]=0.0;
    robotSP0[2]=0.0;
break;
```

Figura 8.73.

- FORWARD (S):

Per a activar aquest mode la tasca Supervisor ha de llegir una "S" a l'array de les indicacions de camí i que vol dir 'Straight'. Llavors, el Robotino ha d'avançar recte seguint la línia fins que es validi la condició d'odometria.

D'assignatures anteriors sabem que per a que el Robotino avanci recte frontalment s'han d'enviar les següents consignes a les rodes: R0=-V, R1=0, R2=+V, però aquestes consignes no inclouen la correcció de posició respecte de la línia. Per poder aplicar-les s'ha de fer el ús de la funció *setMotorVelocity()* tenint en compte la reducció 1:16, i s'ha de dissenyar un controlador realimentat. Per incloure-hi la correcció de posició es pot fer igual que fem en aquest projecte.

Nosaltres el que fem és invocar la funció que assigna la velocitat al robot directament (*setRobotVelocity()*) per lo que no ens hem de preocupar de dissenyar el controlador ni calcular la velocitat per a cada motor. Només s'ha d'assignar el valor per a la component de velocitat de l'eix que es vulgui fer el moviment i el robot aplica el control KPI que té per defecte.

Per al càlcul d'aquest valor s'implementa la estratègia de seguiment de línia del capítol 8.6.

```
case FORWARD: // s means stright,forward

    getBMP(SERVIDOR ,&img,LOW); // obtenim "buffer" que indica on comencen els bites i "tamany"

    capaCR(&img,&img2);
    im2bw(&img2,&img3,46);
    erode(&img3,&img4,7.5);
    dilate(&img4,&img5,3);
    erode(&img5,&img5,2);

    detectPuntos(&img5,35,55, &x1, &y1, &x2, &y2); // en coordenades imatge

    if (detectPuntos=0){    // SUSPENSIÓ DEL PROGRAMA SI NO ES DETECTEN PUNTS A LA IMATGE
        eventC=QUIT;
        mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
    }
    PimgToRobot(&x1,&y1, &x1r,&y1r);
    PimgToRobot(&x2,&y2, &x2r,&y2r);

        // calcul de l'angle de la recta respecte el robot
    int Ax,Ay;
    Ay= y2r - y1r;
    Ax=x2r - x1r;
    phi= atan((double)Ax/(double) Ay)*180/pi;

        //  paremtros equacion de la recta Ax + By + C
    A=(double)(x1r-x2r);
    B=(double)(y2r-y1r);
    C=(double)(y1r*x2r - y2r*x1r);

        // eq. distancia de punto a una recta. punto es origen asi que Ax=By=0.
    a2=pow(A,2);
    b2=pow(B,2);
    dist=C/sqrt(a2+b2);

        //FORMULA OLLERO
    dist2=pow(dist,2);
    inv_r=(dist*cos(phi)-sin(phi)*sqrt(L*L-dist2))*2/(L*L);
        // velocitat angular=vel.lineal/r=v*inv_r   volem velocitat lineal de 20mm/s llavors:

    w=20*inv_r;

            //Composicio de les consignes de velocitat del robot
            robotSP0[0]=20;
            robotSP0[1]= 0;
            robotSP0[2]= w*180/pi;


    break;
```

Figura 8.74.

- TURN_LEFT (L):

S'activa al llegir una 'L' a l'array de les indicacions de camí i es tradueix com 'Left'. Fa rotar al robot Robotino sobre si mateix a una velocitat constant definida a l'inici del programa. Per a girs sobre si mateix només cal assignar valors a la tercera component de velocitat, i això es tradueix com la mateixa consigna per a totes 3 rodes.

```
case TURN_LEFT:
    robotSP0[0]=0.0;
    robotSP0[1]=0.0;
    robotSP0[2]=ROTATE_SP;
break;
```

Figura 8.75.

- TURN_RIGHT (R):

S'activa llegint una "R" a l'array *camino[]* i es tradueix com 'Right'. Té un funcionament idèntic a TURN_LEFT però amb sentit contrari.

```
case TURN_RIGHT:
    robotSP0[0]=0.0;
    robotSP0[1]=0.0;
    robotSP0[2]=- ROTATE_SP;     //-45 rad/seg
break;
```

Figura 8.76.

A continuació tenim un exemple de possible successió d'estats de la tasca controller durant el moviment del robot:

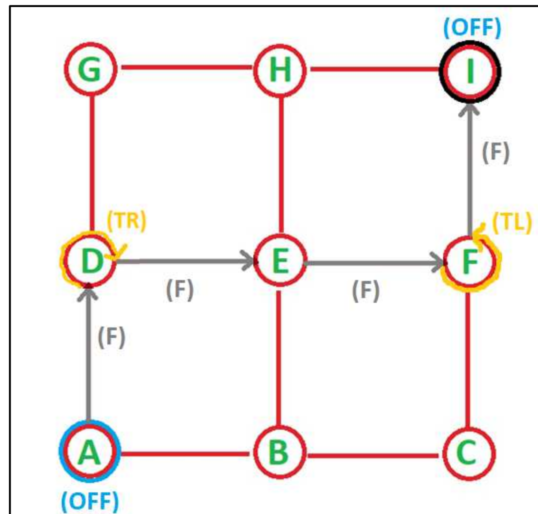


Figura 8.77.

El esquema de transicions de la tasca Controller administrada per la tasca Supervisor és la següent:

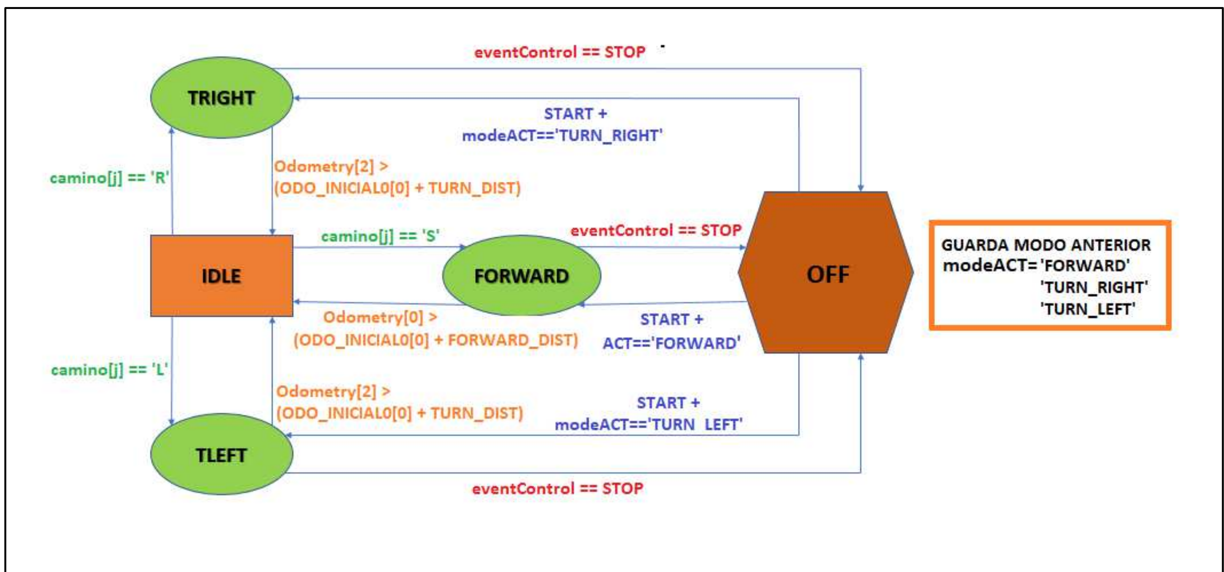


Figura 8.78: Esquema de transicions d'estat de la tasca Controller.

7.6.3 Tasca Monitor

La finalitat d'aquesta tasca es detectar quan s'ha completat un moviment dictat per la tasca Supervisor. Per saber-ho, es fa ús de la odometria.

La odometria és l'estudi de la estimació de la posició de vehicles amb rodes durant la navegació. Per fer aquesta estimació s'utilitzen uns encoders que mesuren les revolucions produïdes en un eix, en aquest cas en el de les rodes, en un interval de temps donat. Les posicions estimades no tenen per que coincidir amb les coordenades món ja que són relatives al punt d'inici, però podem fer que coincideixin al posicionar el robot abans d'iniciar el primer moviment.

Per a la tasca monitor els estats són:

```
//Estats de la tasca Monitor
typedef enum {MONITOR_OFF,MONITOR_ON} MODEMON_T;
volatile MODEMON_T modeMON = MONITOR_OFF;
```

Figura 8.79: Estats de la tasca Monitor.

- Monitor_OFF:

Aquest estat es útil per quan ens interessa tenir el monitor apagat, es a dir, quan no s'està fent cap moviment. Això serà quan s'inicialitzi el programa però encara no s'ha començat el moviment, en les transicions d'estat de la tasca Supervisor o quan es pausa la execució.

```
switch(modeMON0) {
    case MONITOR_OFF:

    break;
```

Figura 8.80.

- Monitor_ON:

Per contra, aquest estat s'activa sempre que s'està fent un moviment i serveix per monitoritzar la odometria. Quan la odometria llegida es igual o superior a uns llindars establerts, s'envia l'esdeveniment END_MOV a la cua d'esdeveniments per a que la rebi la tasca Supervisor.

Durant la monitorització de les odometries es comproven totes tres components però com són excloents entre elles, només es podrà validar una per cop. El funcionament es simple; abans d'iniciar-se un moviment, s'emmagatzema la odometría inicial en la tasca Supervisor i es traspassa a la variable global. Al activar el mode Monitor_ON, es passa aquesta informació a variables locals i es comença a mesurar la odometria actual per comparar-la de forma continuada.

```
pthread_mutex_lock(&m);
ODO_INICIAL0[0]=ODO_INICIAL[0];
ODO_INICIAL0[1]=ODO_INICIAL[1];
ODO_INICIAL0[2]=ODO_INICIAL[2];
pthread_mutex_unlock(&m);

getOdometry(&Odometry[0],&Odometry[1],&Odometry[2]);
```

Figura 8.81.

El principal problema es que les odometries son mesurades utilitzant els eixos de les coordenades món. Per poder operar amb elles lo ideal hauria sigut crear un conversor de coordenades o un inversor d'eixos per fer-los coincidir amb els del Robotino, però probablement ens donaria valors nuls ja que el robot no es mou respecte el seu centre.

Així doncs, s'ha agut de fer varies condicions jugant amb els eixos i separant per quadrants:

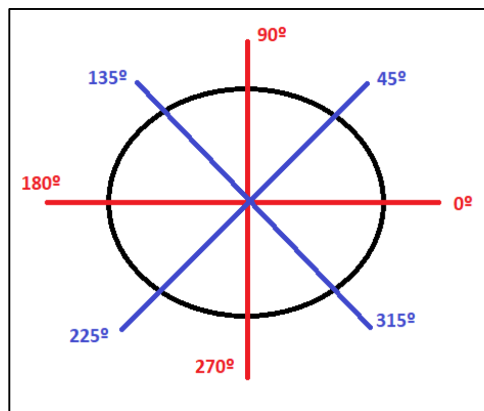


Figura 8.82.

A la hora de fer les condicions s'ha tingut en compte que la tercera component de la odometria no serà exacta. Segurament tindrà alguna desviació. Llavors, en comptes de posar la condició dels graus exacta (vermell) s'ha posat un llindar que inclou els angles compresos entre els angles en blau per identificar en quin sentit s'està fent el moviment.

Per monitoritzar la odometria en cas d'avanç el que es fa, de forma resumida, és comprovar en quina direcció i sentit s'està fent el moviment. Una vegada validada aquesta condició es procedeix a tractar la component corresponent de la odometria.

```
// ODOMETRIA EN FORWARD
if ((Odometry[2]<45) && (Odometry[2]>-45)){

        if (Odometry[0]>(ODO_INICIAL0[0]+FORWARD_DIST)){ //odometria > odometria inicial +1000mm

            eventC=END_MOV;
            mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
        }

}
```

Figura 8.83.

En la Figura anterior tenim el tractament per al sentit positiu de l'eix X en coordenades món.

```
if ((Odometry[2]>135) && (Odometry[2]<-135)){

    if (Odometry[0]<(ODO_INICIAL0[0]-FORWARD_DIST)){
```

Figura 8.84.

En la *Figura 8.84.* tenim lo mateix però per al sentit negatiu. Com les dues condicions anteriors són per quan el moviment es fa en l'eix X de coordenades món, es monitoritza la primera component [0] de la odometría. Per als pròxims dos és en l'eix Y així que s'haurà d'observar la component [1] de la odometría. La distancia d'avanç es 1m perquè els camins entre els nodes mesuren tots 1m±5cm.

```
if ((Odometry[2]<-45) && (Odometry[2]>-135)){

    if (Odometry[1]<(ODO_INICIAL0[1]-FORWARD_DIST)){
```

```
if ((Odometry[2]>45) && (Odometry[2]<135)){

    if (Odometry[1]>(ODO_INICIAL0[1]+FORWARD_DIST)){
```

Figura 8.85.

Per als angles de gir és més fàcil ja que els angles de gir i els convenis de signes segueixen sent els mateixos:

```
// ODOMETRIA EN GIR

        if (Odometry[2]<(ODO_INICIAL0[2]- TURN_DIST)){
            setRobotVelocity(0.0,0.0,0.0); // o poner controlador a off
            if (g==1) g=0;
            else if (g==0) g=1;

            eventC=END_MOV;
            mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
        }

        if (Odometry[2]>(ODO_INICIAL0[2]+ TURN_DIST)){
            setRobotVelocity(0.0,0.0,0.0); // o poner controlador a off
            if (g==1) g=0;
            else if (g==0) g=1;

            eventC=END_MOV;
            mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
        }
```

Figura 8.86.

7.6.4 GUI_Input , ShowGUI() i ShowAdvanced()

La finalitat d'aquestes tasques es obtenir i mostrar informació entre l'usuari i el programa. Per fer-ho possible, una s'encarrega de recol·lectar la informació contínuament, una altra de mostrar-la i la tercera esta pendent de les interaccions per part de l'usuari.

La funció auxiliar *ShowAdvanced()* serveix per mostrar per pantalla informació sobre el robot i el seu moviment. Aquesta informació és:

- Menú d'opcions per l'usuari: START,STOP,QUIT.
- El mode en que esta treballant el Supervisor i el Monitor.
- L'algorisme de pathfinding utilitzat
- Un lay-out del circuit amb la seqüencia de nodes pels que ha passar el Robotino.
- L'estat de les bateries i el Voltatge.
- Les consignes de velocitat en les components Vx, Vy i Ω.
- Les velocitats mesurades en les components.
- La posició actual del robot referenciada per la odometria mesurada.

Per obtenir aquesta informació contínuament s'utilitza la funció *ShowGUI()* i al final s'invoca la *ShowAdvanced()* per passar-se-la *:*

```
do {

    pthread_mutex_lock(&m);
    modeSUP0=modeSUP;
    modeCON0=modeCON;
    robotSP0[0]=robotSP[0];
    robotSP0[1]=robotSP[1];
    robotSP0[2]=robotSP[2];

    ODO_INICIAL1[0]=ODO_INICIAL[0];
    ODO_INICIAL1[1]=ODO_INICIAL[1];
    ODO_INICIAL1[2]=ODO_INICIAL[2];

    pthread_mutex_unlock(&m);

    //POSICIÓ ROBOT (odometria)
    getOdometry(&Odometry[0],&Odometry[1],&Odometry[2]);

    //VELOCITAT ROBOT
    getRobotVelocity(&robotVelocity[0],&robotVelocity[1],&robotVelocity[2]);


    //NIVELL BATERIA
    getBatteryVoltage(&battery);

//  ShowAdvanced(ruta, battery, robotVelocity, Odometry, modeSUP0,robotSP0);
    ShowAdvanced(ruta, battery, robotVelocity, Odometry, modeSUP0,modeCON0, ODO_INICIAL1);
    clock_nanosleep(CLOCK_REALTIME,TIMER_ABSTIME,&ts,NULL);
    timespecPlusTimeInterval(&ts,&ts,&interval);
} while (1);
```

Figura 8.87.

Per últim, la tasca *GUI_Input* serveix per obtenir informació de l'usuari durant la execució del programa i envia un esdeveniment a la cua quan en rep. A l'inici del programa també es demana la introducció dels nodes inici i final del recorregut i el tipus d'algorisme de pathfinding, però com no es una informació que s'esperi continuadament sinó que només es demana un cop, s'ha implementat en la tasca principal *main*.

La informació que s'espera recol·lectar de l'usuari és:

- Un 1, que es tradueix com l'esdeveniment anomenat START.
- Un 2, creant l'esdeveniment STOP.
- Un 3, per crear l'esdeveniment QUIT.

7.6.5 TASCA PRINCIPAL (*Main*).

Gràcies a que s'han dividit les diferents operacions en diferents tasques, el *main* ha quedat bastant reduït.

Al principi el que es fa es crear la estructura dels fils d'execució i l'esdeveniment relacionat amb la col·lisió del bumper que ve implementat en la API del Robotino. També hi ha la possibilitat de crear l'esdeveniment relacionat amb la detecció d'obstacles a partir dels sensors de distancies que es troben repartits al voltant del xassís del robot. Per aquest projecte no s'han fet servir però pot ser bastant útil en cas de implementar-hi el re-càlcul de ruta per evitar obstacles. Finalment es crea la cua per la que s'enviarà l'esdeveniment.

La variable *aux* serveix per detectar si hi ha hagut algun error.

```
int main(void)
{
    pthread_t tidSUP;
    pthread_attr_t attr;
    struct mq_attr qattrC;
    struct sched_param param;
    struct var_esdev init;
    int aux;

    init.tipusEsdeveniments = ESDE_BUMPER;
    aux = initSystem ("/net/*********/home/alumne/errorsRobotino",T_API, &init);
    if (aux==ERROR)
        return;

    cua_api = init.cua;
```

Figura 8.88.

Després es crea la segona cua que hem anomenat *cua_control*, i que ens servirà per enviar els demés esdeveniments. Es crea amb una mida determinada per la constant *CAPACITAT_CUA* a l'hora que s'habilita per poder llegir i escriure-hi fent ús de la funció *mq_open*.

```
// Cua: cua_control
qattrC.mq_maxmsg=CAPACITAT_CUA;
qattrC.mq_msgsize=sizeof(EVENT_CONTROL);
cua_control=mq_open("/event_cua_control", O_RDWR|O_CREAT, S_IRUSR|S_IWUSR, &qattrC);
```

Figura 8.89.

Abans de començar el programa fem la seqüencia per obtenir el recorregut. Primer es declaren les variables *origen* i *target* del tipus *PUNTOS* i el mode de treball. Es treu per pantalla el primer menú i s'espera a que l'usuari introdueixi les dades.

```
//pathfinding
    PUNTOS origin,target;
    int modo;
    printf("%c\n*******************   PATHFINDING  ROBOTINO QNX   *******************",0x0C);
    printf("\n\n 1: Indica node ORIGEN i node TARGET.\n");
    printf(" 2. Indica el tipus d'algorisme pathfinding (1: GREEDY. 2: DIJKSTRA.).\n");

scanf("%i",&origin);
scanf("%i",&target);
scanf("%i",&modo); // indicara el tipo de pathfinding
```

Figura 8.90.

Depenent del mode introduït, es calcula la ruta amb un mètode de pathfinding o amb un altre i després es procedeix a traduir aquest camí a indicacions (anar recte, girar...). Ho podem veure a la Figura 8.91:

```
if (modo==1) Greedy(origin,target,ruta);
else if (modo==2) Dijkstra(origin,target,ruta);

puntosAcamino(ruta,camino);
```

Figura 8.91.

Es crea el mutex i es fa la primera captura de temps per poder calcular posteriorment els intervals de temps de les tasques:

```
pthread_mutex_init(&m,NULL);
clock_gettime(CLOCK_REALTIME, &t0);
```

Figura 8.92.

Després creem el thread principal amb el mètode de planificació de la CPU escollit:

```
pthread_attr_init(&attr);
pthread_attr_setinheritsched(&attr,PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&attr,SCHED_RR);
```

Figura 8.93.

Hi han varis mètodes de planificació de la CPU, però els més usuals són el Round robin i FIFO:

- En el mètode FIFO se li assigna el processador al primer procés que entra a la cua de processos preparats i l'executa fins que acaba. Els demés processos de la cua han d'esperar en ordre d'arribada fins que els hi toqui el seu torn.

- Per al mètode Round Robin s'executen varis processos diferents de forma concurrent. Per la utilització equitativa dels recursos s'assigna un temps limitat d'ús per cada procés i després se suspèn el procés per donar la oportunitat a un altre procés, i així successivament.

També existeix una variant anomenada Round Robin amb Prioritats i és el que s'utilitza en aquest programa. En aquesta variant, se li assigna la CPU al procés de la cua de processos llestos amb prioritat més alta. Per a processos amb igual prioritat es reparteix la CPU com es faria en el mètode de Round Robin original.

Després de crear el thread principal es creen els threads secundaris a partir d'aquest, utilitzant les prioritats pre-definides al principi del programa, on els paràmetres constants.

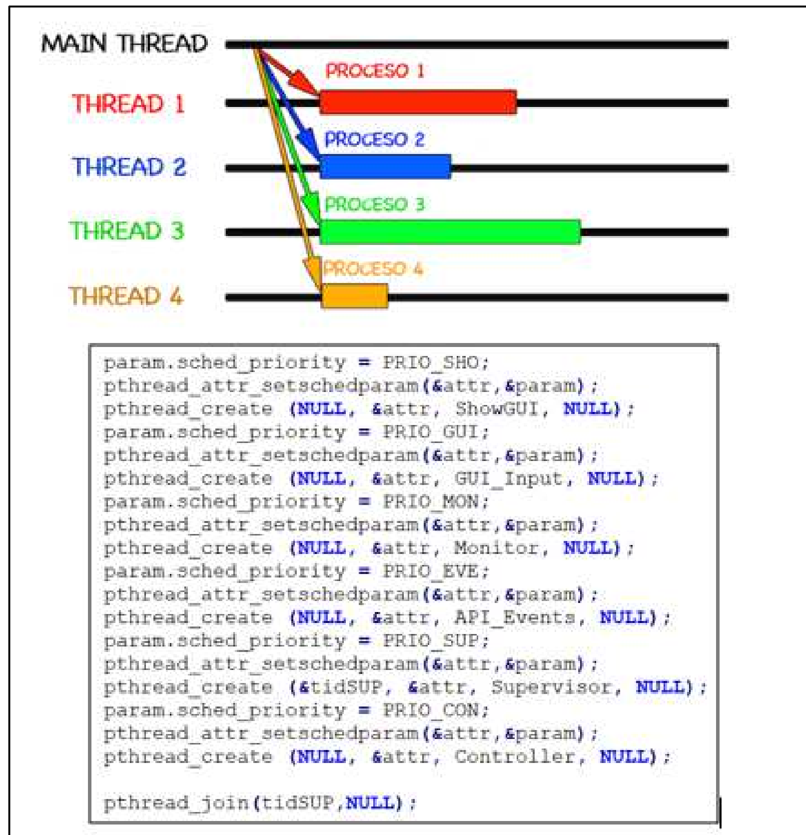


Figura 8.94.

Finalment fem un *thread_join()* per a que el thread principal esperi a que finalitzi el del Supervisor. Una vegada finalitzat, es tanca la connexió, les cues i es trenca el fil principal d'execució (figura 8.95.)

```
    aux=closeSystem();
    if (aux==ERROR)
        return;

    mq_close(cua_control);
    mq_unlink("/event_cua_control");
}
```

Figura 8.95.

9. CONCLUSIONS I PROPOSTES DE MILLORA

Després d'acabar aquest treball de fi de grau i de reflexionar sobre tot el procés he arribat a varies conclusions, però tractaré de ser lo més concís possible.

Observant tot el treball realitzat hem dono compte de les nombroses eines i competències adquirides al llarg de tots aquests anys del grau. Hem resulta paradoxal que de totes aquestes eines, a vegades el punt clau per resoldre un problema, que a priori sembla molt complexa, pot arribar a dependre de coneixements tant simples com el càlcul de l'angle a partir d'una tangent o la conversió dels angles de radiants a segons.

També hem dono compte de la veracitat d'un gran dit que solia dir la meva mare, i que traduït al català ve a ser: 'De vegades, qui té pressa arriba tant ràpid com qui no la té'. I és cert perquè quan es corre la possibilitat d'entrebancar-se i caure és alta, i caure et rellenteix. L'analogia d'això es pot veure clarament en l'àmbit de la programació. Quan apareix un error i s'intenta solucionar ràpidament, normalment es desencadenen més errors. Però quan es para, observa i reflexiona sobre cada punt és quan s'avança amb passos firmes cap a la solució.

Deixant de banda les conclusions personals i centrant-me en la temàtica del treball, aquest projecte tenia uns objectius inicials que s'han hagut d'anar reajustant a mesura que s'avançava. Un dels objectius inicials era poder acabar implementant la detecció d'obstacles i el posterior re-càlcul de rutes, però no ha estat possible. Per això se'n parla tant del tema al llarg de la memòria i és la meva principal proposta de millora.

El propòsit principal s'ha assolit: El robot Robotino és capaç de fer la seqüencia calculada a partir dels punts rebuts amb l'algorisme de seguiment de línia de l'anterior projectista[3] i també ho fa utilitzant l'algorisme de Pure-Pursuit. No ha sigut gens fàcil però finalment s'ha aconseguit. També hem sigut capaços de fer-ho funcionar tant amb l'IDE 5.0 com amb el SO de QNX.

Després de fer les probes amb els algorismes de pathfinding implementats i comparar-los, concloc que no hi ha cap algorisme millor que un altre, sinó més bé que cada algorisme s'adapta millor o pitjor que els demés depenent de la situació en la que s'apliqui i el seu resultat depèn molt de la matriu de pesos o Weightpoints.

Hem observat com l'algorisme Greedy ha passat d'obtenir un dels camins òptims a l'exemple del tema 7, on s'utilitza la primera matriu creada, a obtenir com a resultat el quart millor camí amb la segona matriu de pesos. Però tot i això, per a la nostra aplicació es podria considerar vàlida si el que ens interessa només es que el robot faci la trajectòria entre els nodes donats i ens és irrellevant el temps que es tardi o l'energia consumida.

Per últim, m'agradaria fer una altra proposta de millora: L'algorisme de Pathfinding Dijkstra té limitacions ja que en cas d'empat només és capaç d'obtenir una de les solucions. Parlant amb un professor es va tenir la idea de fer alguna cosa al respecte, així que vaig modificar la funció per a que calculi sempre les dues millors rutes per detectar empats i, en cas afirmatiu , que torni ambdós camins. Aquesta funció esta modificada en MATLAB però no per QNX, així que es pot utilitzar com a base i es troba a l'annex.

10 BIBLIOGRAFÍA:

1: ESTUDI D'ADAPTACIÓ A QNX DE LA PLATAFORMA DIDÀCTICA DE ROBÒTICA MÒBIL ROBOTINO (2011).

2: ESTUDI D'INTEGRACIÓ D'UNA CÀMERA DE XARXA A UN ROBOT MÒBIL (2013):

https://upcommons.upc.edu/handle/2099.1/21951

3: ESTUDI D'ESTRATÈGIES DE SEGUIMENT DE LÍNEA PER AL ROBOT MÒBIL ROBOTINO (2016):

https://upcommons.upc.edu/handle/2117/107423

4: Algorismes de pathfinding trobats a la xarxa: BFS i DFS

https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40

5: descarregar TOOLBOX CAMERA CALIBRATION

http://www.vision.caltech.edu/bouguetj/calib_doc/download/index.html

6: DIKSTRA PER MATLAB, Jorge barrera, ABRIL 2007:

https://es.mathworks.com/matlabcentral/fileexchange/14661-dijkstra-very-simple

7: Documentació API Robotino (pràctica 3) de la assignatura PSCTR.

8: Instal·lador de l'IDE 5.0:

http://blackberry.qnx.com/en/products/tools/qnx-momentics

Manual càmera vapix:

https://www.axis.com/files/manuals/HTTP_API_VAPIX_2.pdf

codi de colors RGB:

http://www.pagaelpato.com/tecno/colores.htm

Imatge Bitmap (bmp):

https://es.wikipedia.org/wiki/Windows_bitmap

Access point del Robotino (LEVEL 1):

http://doc.openrobotino.org/download/manuals/LEVEL1_WLAN_AP_WAP-0004_Manual.pdf

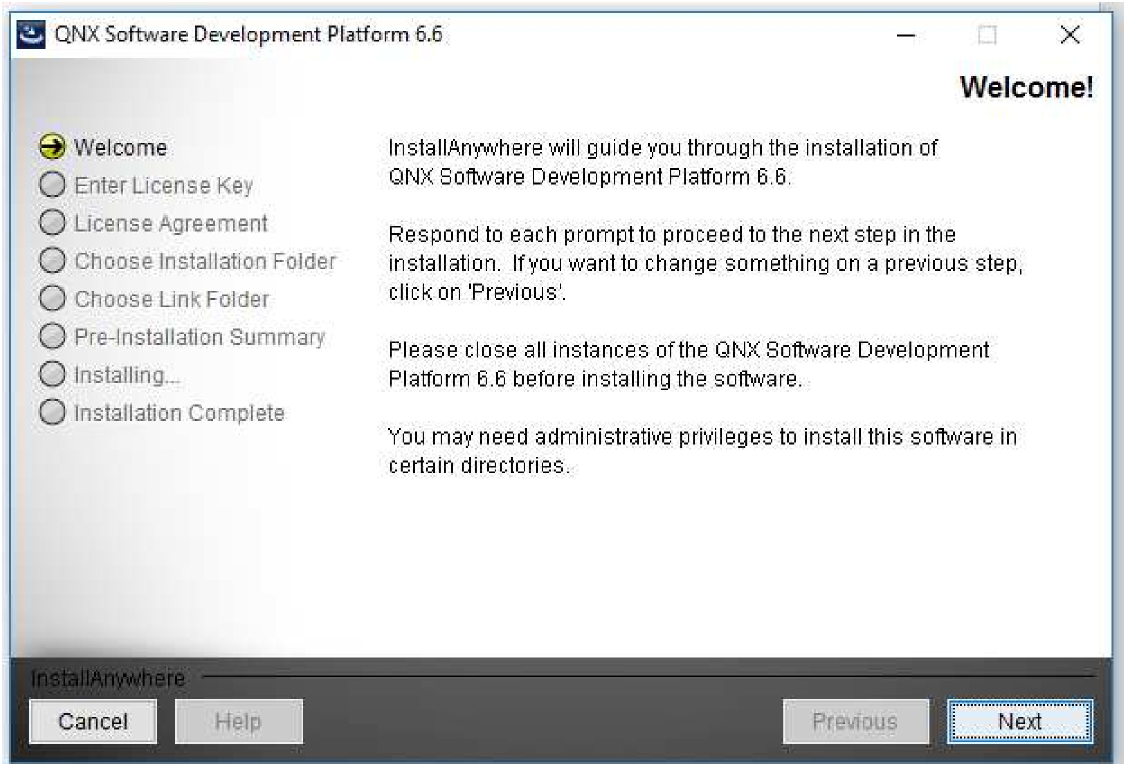
algorismes de pathfinding:

- Greedy http://elvex.ugr.es/decsai/algorithms/slides/4%20Greedy.pdf https://es.wikipedia.org/wiki/Algoritmo_voraz
- BUSQUEDA EN ANCHURA (BFS): https://es.wikipedia.org/wiki/Búsqueda_en_anchura

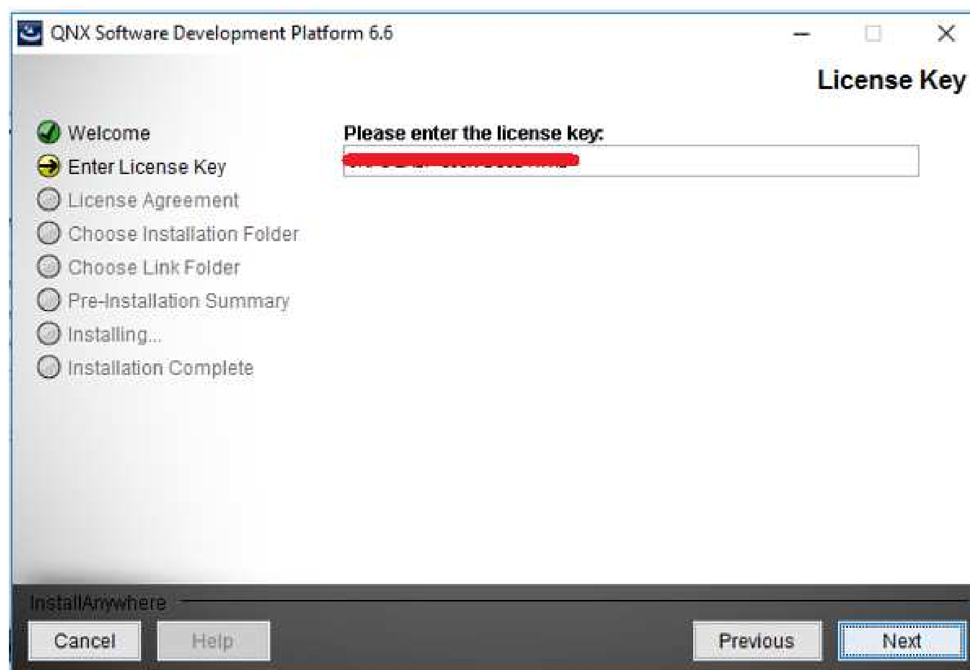
11 Annexos:

A: GUÍA D'INSTAL·LACIÓ DE L'IDE 5.0

Primer s'ha de descarregar l'instal·lador de la pàgina oficial de QNX[8]. Una vegada descarregat, s'ha d'obrir fent doble clic i ens apareixerà per pantalla el següent menú:

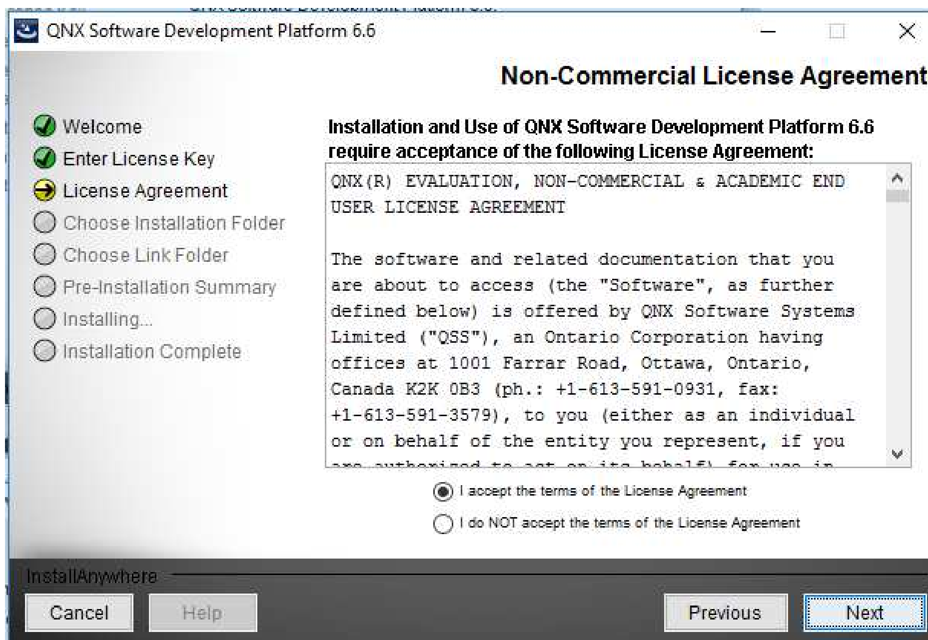


Clic a NEXT.

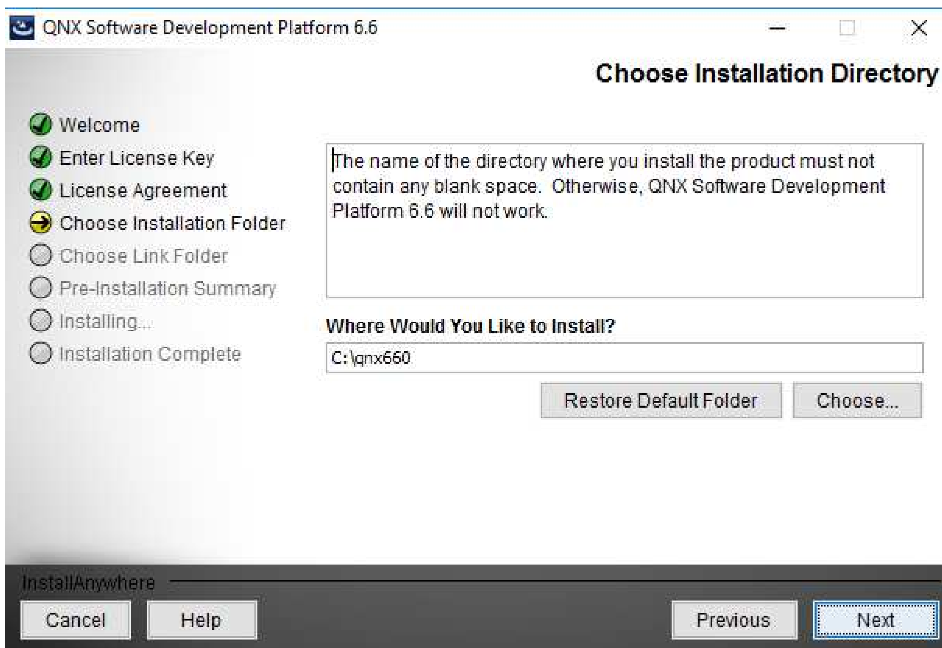


En aquesta finestra se'ns demana introduir el numero de llicencia. En cas de no tenir, posar-se en contacte amb el professor responsable.

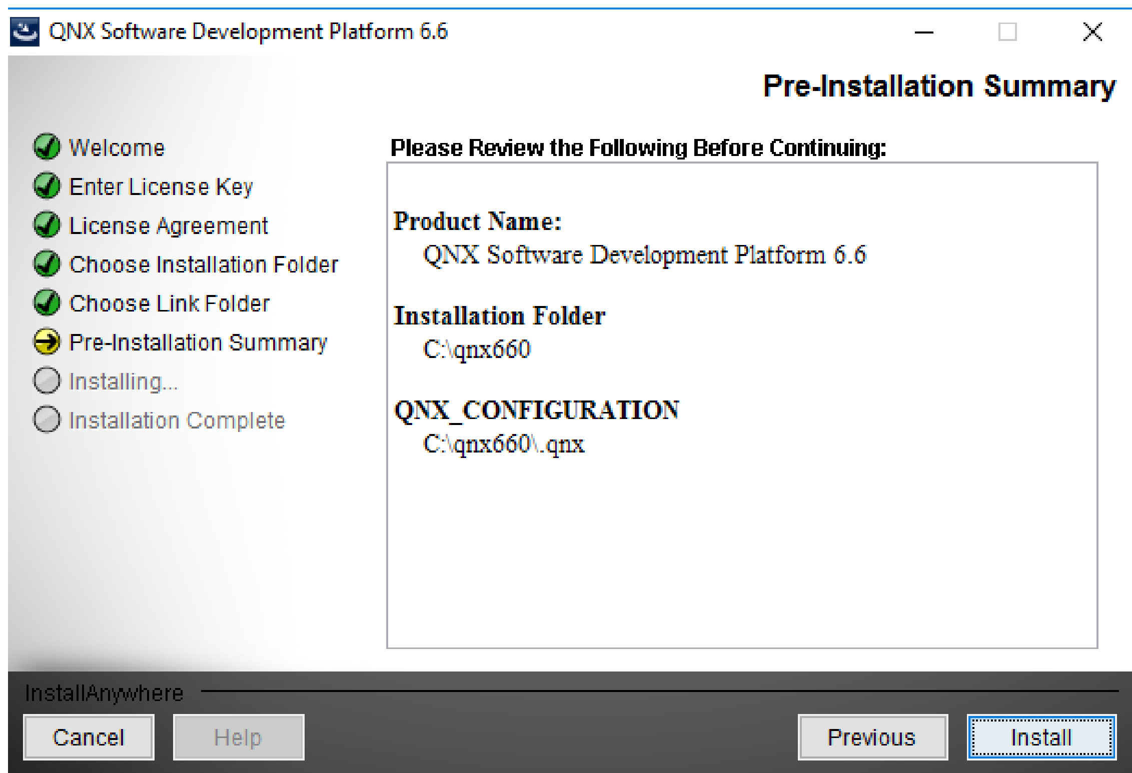
Ara s'ha de Fer clic a NEXT, acceptar les condicions i tornar a clicar NEXT:



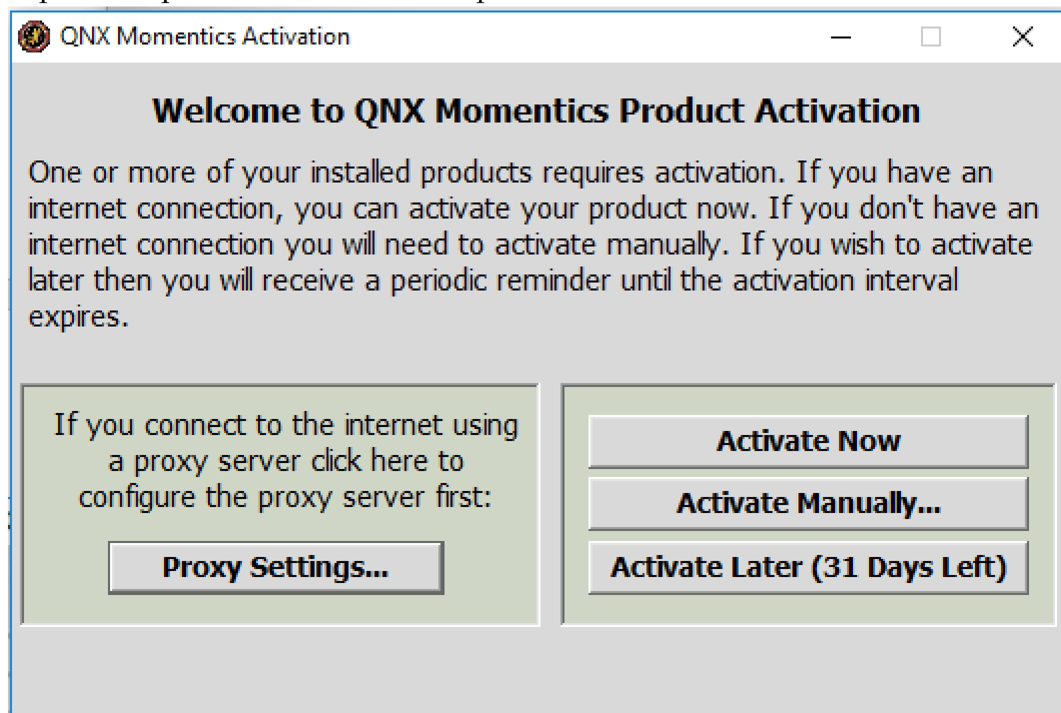
Ens demana on volem instal·lar el programa. Es pot posar a qualsevol carpeta però es recomana que sigui una ubicació no accessible per a usuaris sense privilegi, com ara l'arrel de la memòria:



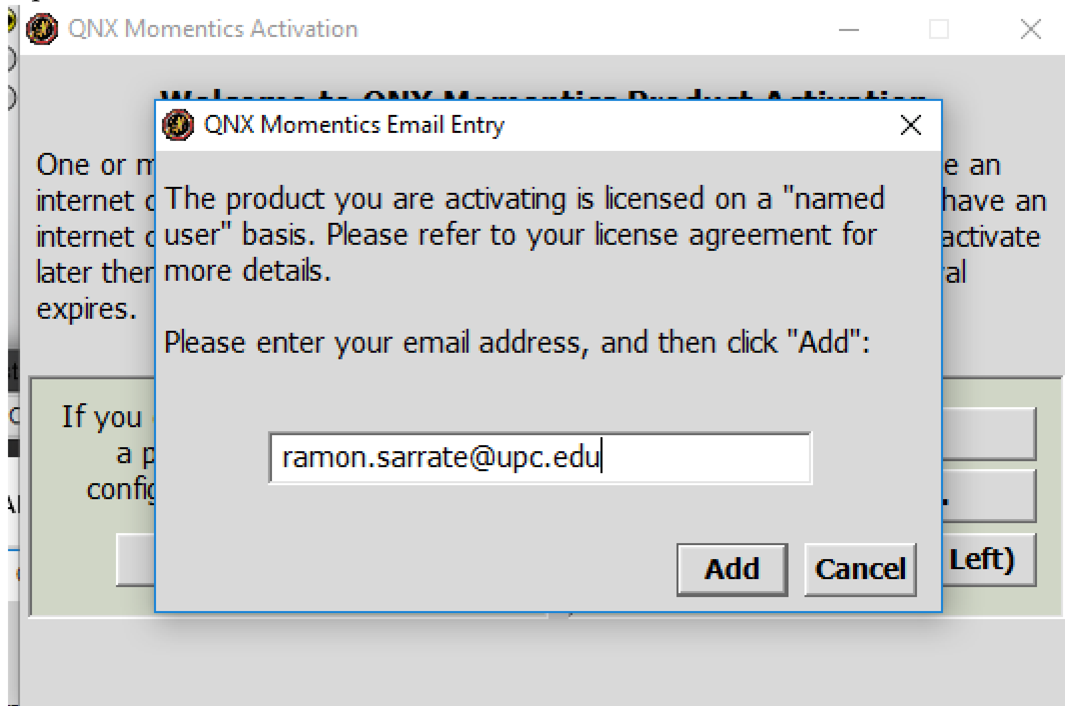
Clic a NEXT i INSTALL:



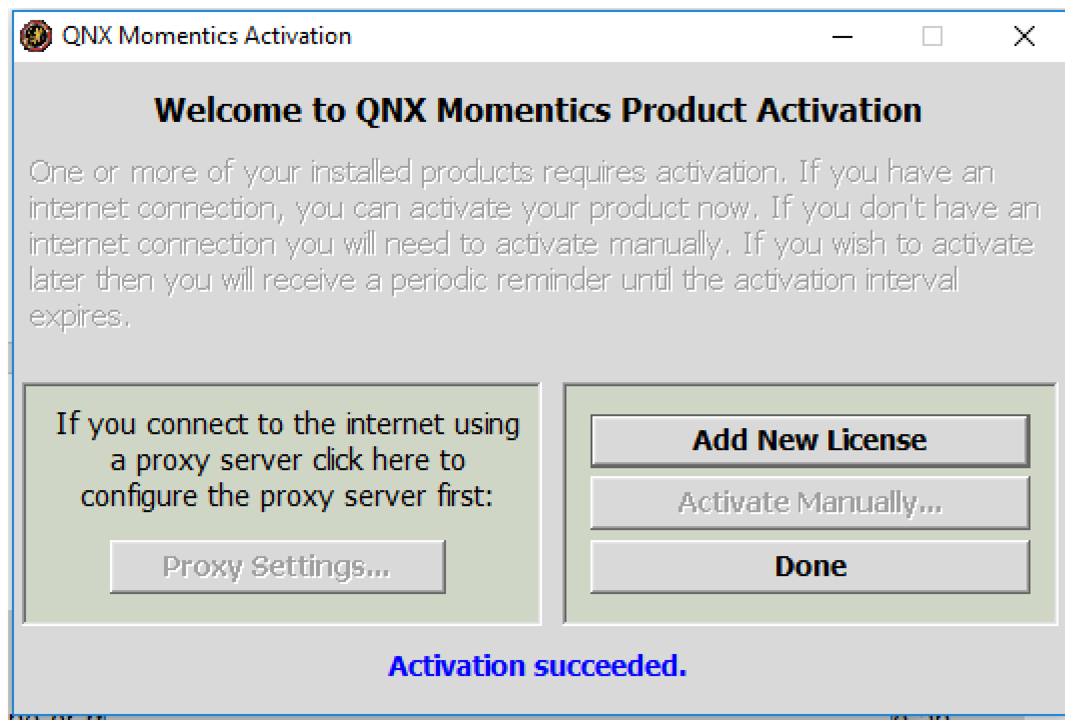
Esperem, i quan acabi sortirà això a pantalla:



Seleccionem ACTIVATE NOW i introduïm l'e-mail associat a la clau de llicencia, en aquest cas del tutor del TFG:



Una vegada validat ens ha de sortir en blau el missatge: ACTIVATION SUCCEEDED.



Cliquem DONE a ambdues finestres i ja esta!!

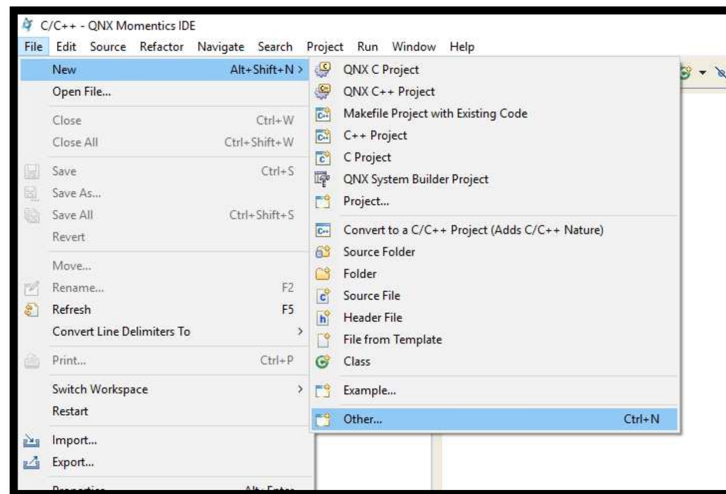
Aquest es la icona que apareixerà a l'Escriptori:

B: CREACIÓ DEL *TARGET* ROBOTINO I DEL PROJECTE A L'IDE 5.0

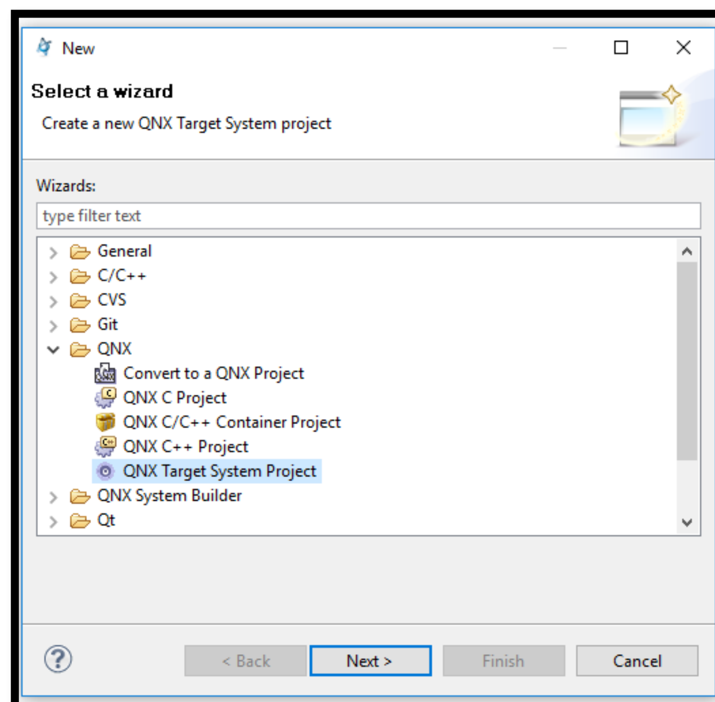
Primer de tot, per poder connectar el nostre PC (host) amb el *target*(Robotino) hem de configurar aquest últim a l'IDE. Per fer-ho necessitem saber la seva direcció IP, que és 192.168.1.118

Ara hem de crear un *QNX Target System Project*:

i. Al menú cliquem *File* > *New* > *Other*



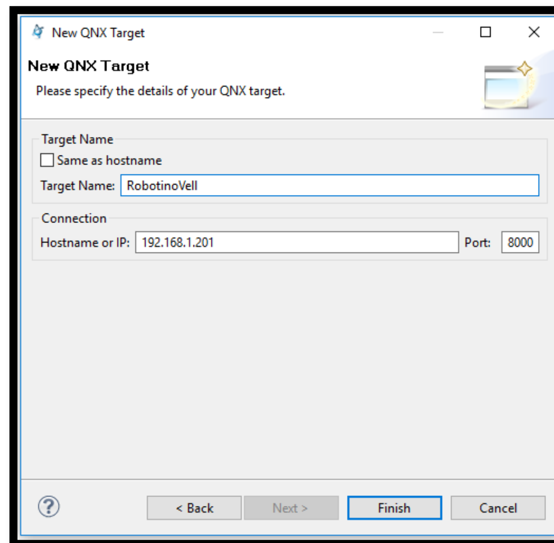
ii. A la llista expandim la carpeta QNX.

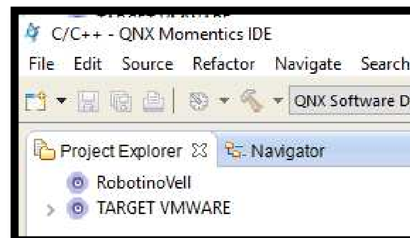


iii. Seleccionem **QNX *Target* System Project**, NEXT i s'obrirà la finestra New QNX Target.

iv. Assignem nom i IP al target. Tenim l'opció d'anomenar-lo per la seva direcció IP.

v. Cliquem FINISH.



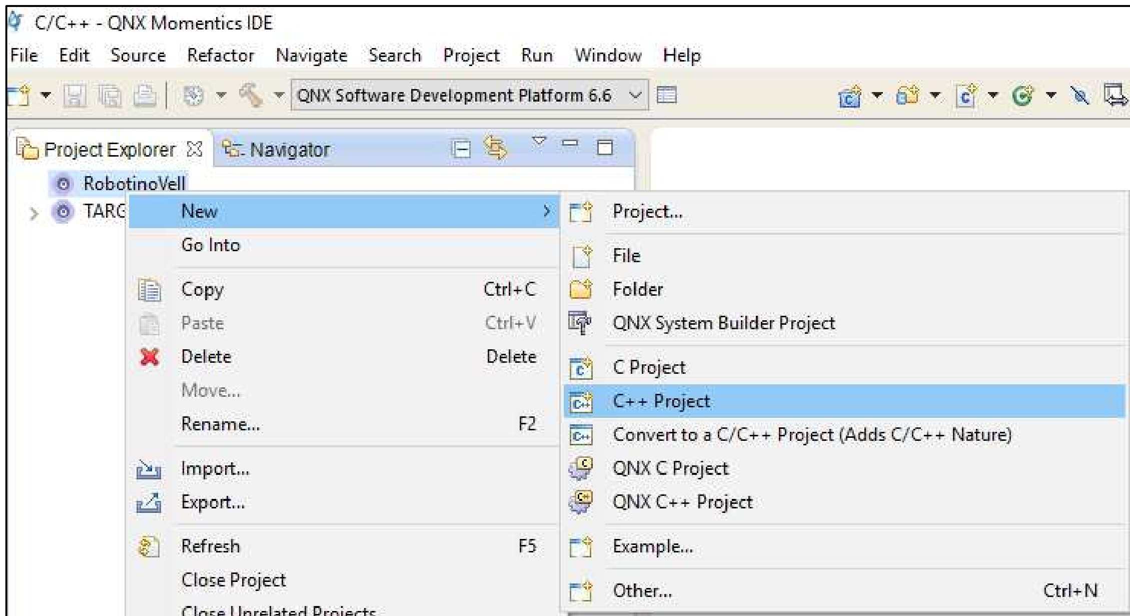


I ja tenim el target creat:

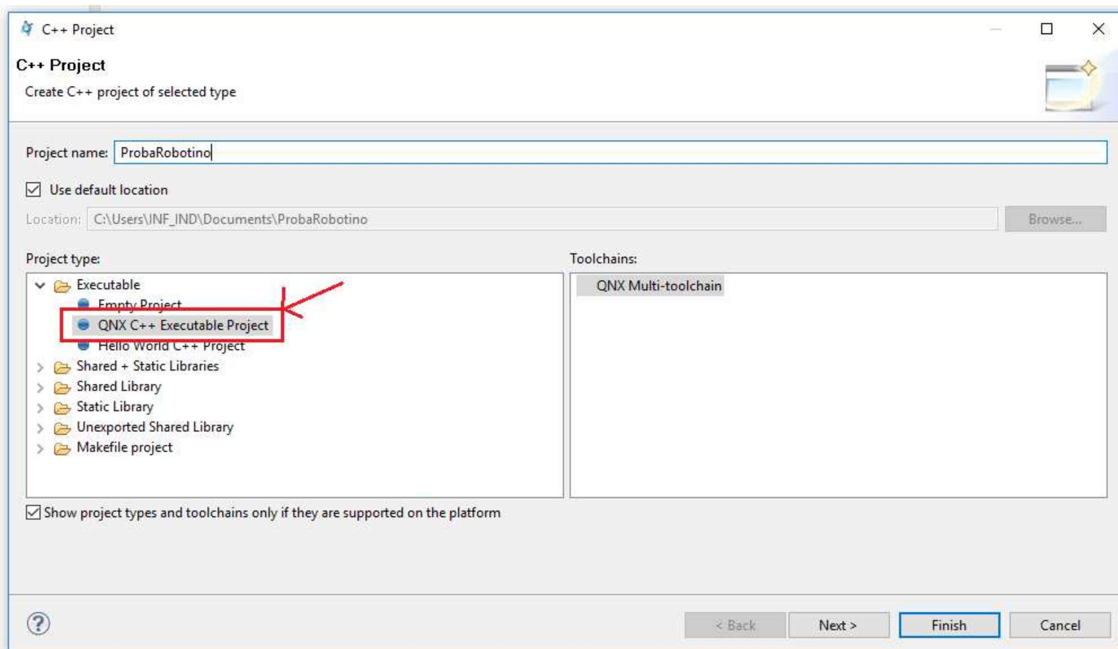
- CREACIÓ DEL PROJECTE

Després de crear el *target* a l'IDE, fem clic dret sobre la seva icona que apareix a la part esquerre del programa i ens sortirà un menú desplegable:

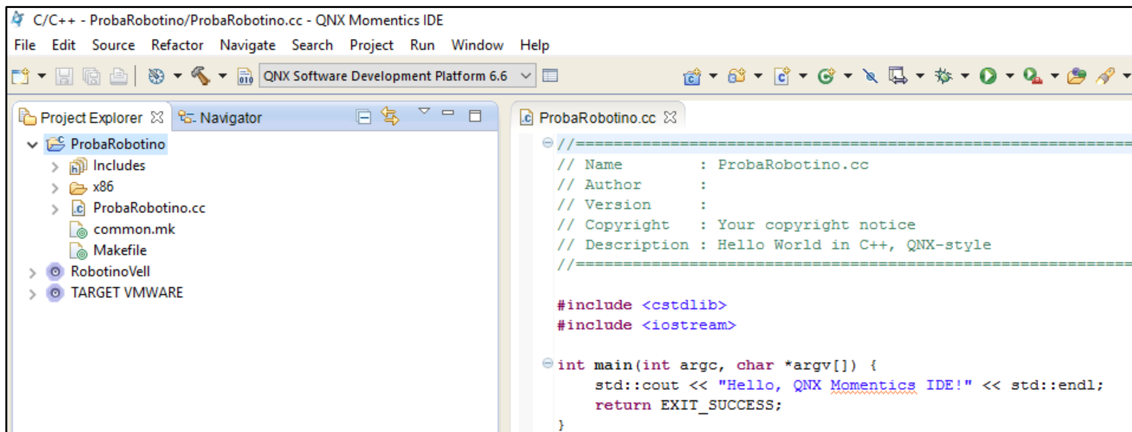
Fem *New* > *C++ Project:*



I ens apareixerà la següent finestra on seleccionem primer el tipus (QNX C++ EXECUTABLE PROJECT) i després li posem el nom al projecte:

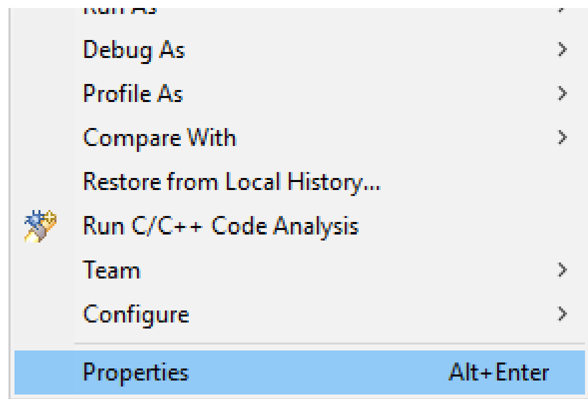


Cliquem a *finish* i ens sortirà la finestra amb un programa d'exemple que podem modificar o substituir pel nostre programa.

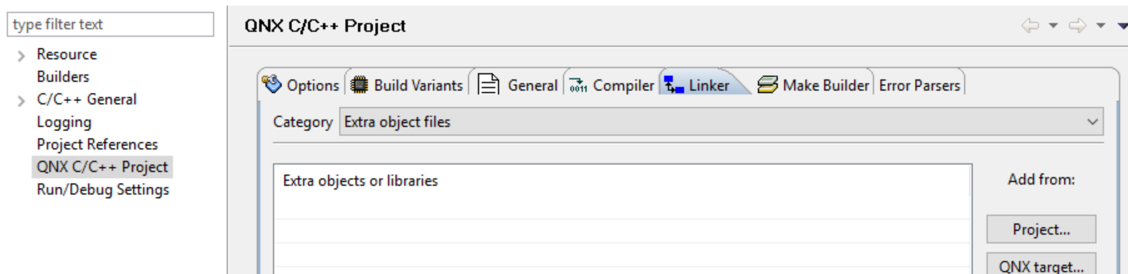

C/C++ - ProbaRobotino/ProbaRobotino.cc - QNX Momentics IDE
File Edit Source Refactor Navigate Search Project Run Window Help
QNX Software Development Platform 6.6
Project Explorer
Navigator
ProbaRobotino
Includes
x86
ProbaRobotino.cc
common.mk
Makefile
RobotinoVell
TARGET VMWARE
ProbaRobotino.cc
// Name   : ProbaRobotino.cc
// Author   :
// Version   :
// Copyright   : Your copyright notice
// Description : Hello World in C++, QNX-style
#include <cstdlib>
#include <iostream>
int main(int argc, char *argv[]) {
std::cout << "Hello, QNX Momentics IDE!" << std::endl;
return EXIT_SUCCESS;
}

C: TRANSFERENCIA I EXECUCIÓ DE CODI EN EL ROBOTINO

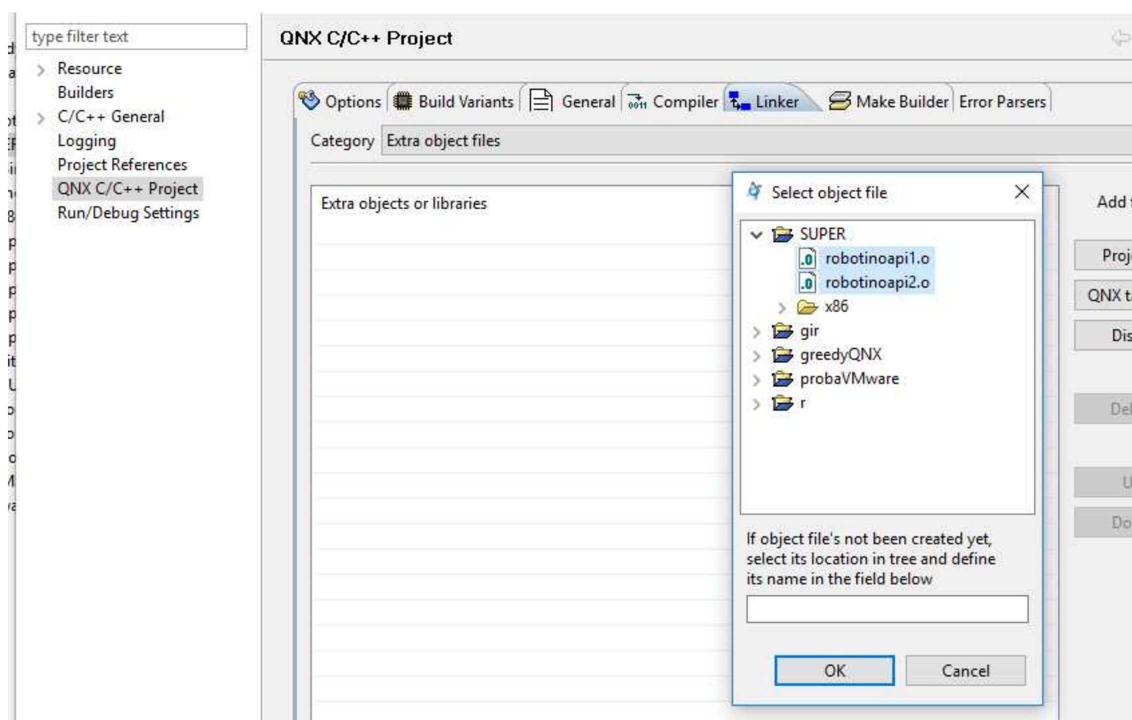
Una vegada tenim creat el *target* i el projecte, fem clic dret sobre ell i seleccionem la opció *Properties:*

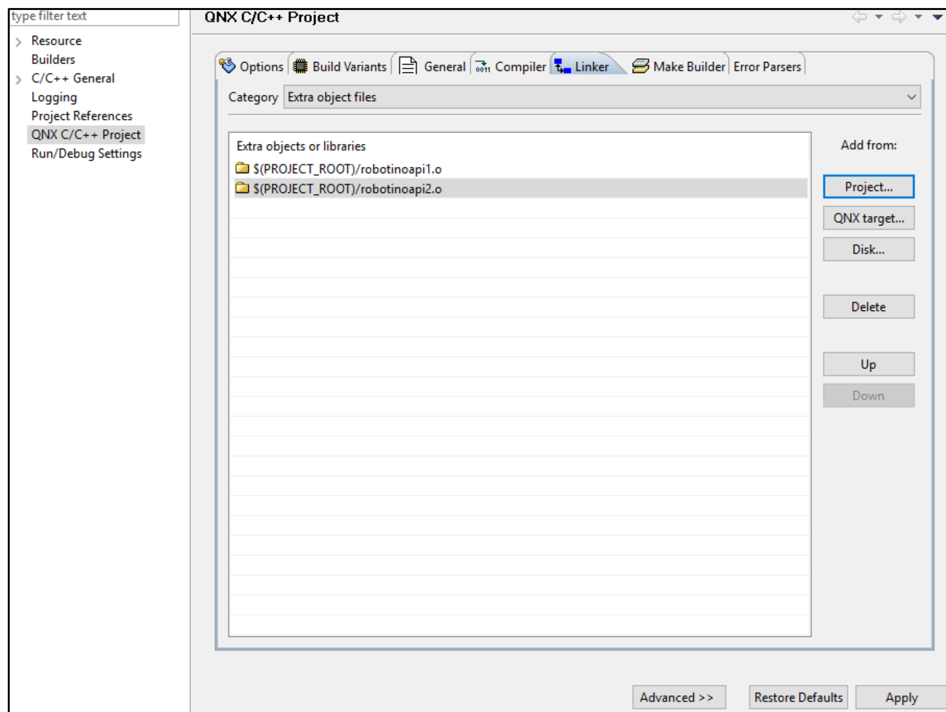


Anem a la pestanya *QNX C/C++ project* i anem a la finestra Linker. Fem clic a la pestanya *category* i seleccionem la opció '*extra object files*'.



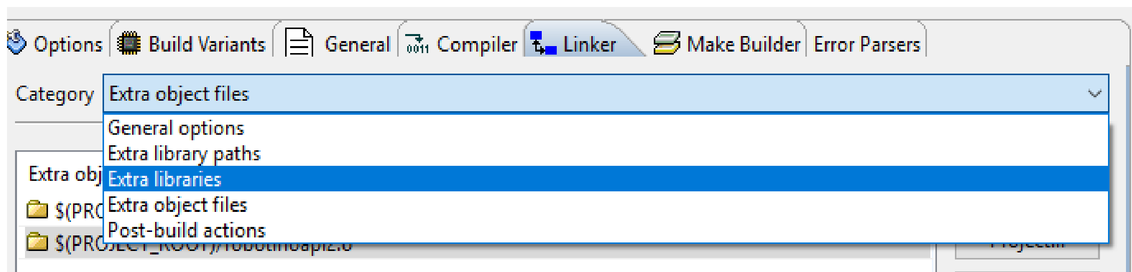
Anem a *project* i busquem els arxius a afegir. En el nostre cas seran totes les API's i la llibreria de Pathfinding, però per al programa de l'exemple només es necessiten els dos arxius de la API Robotino.



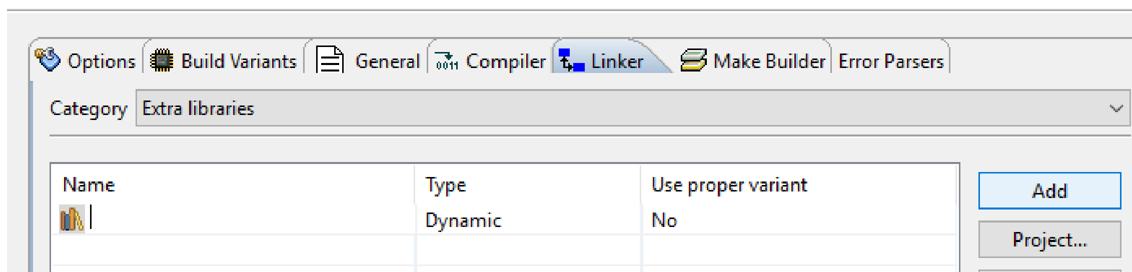


I ara que ens surten li donem *apply*.

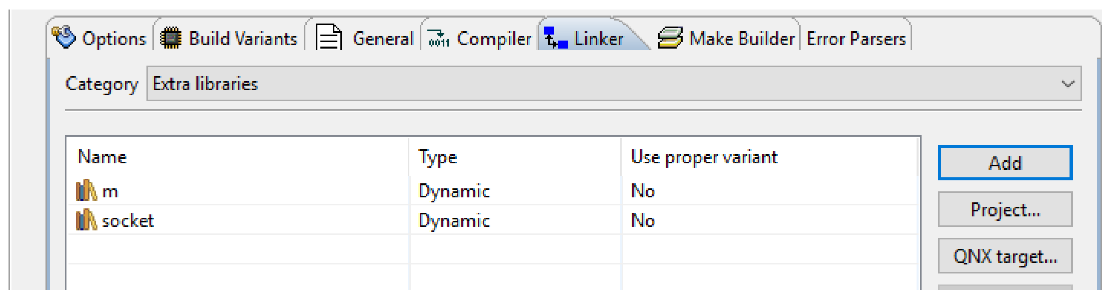
Ara canviem la finestra a extra llibràries



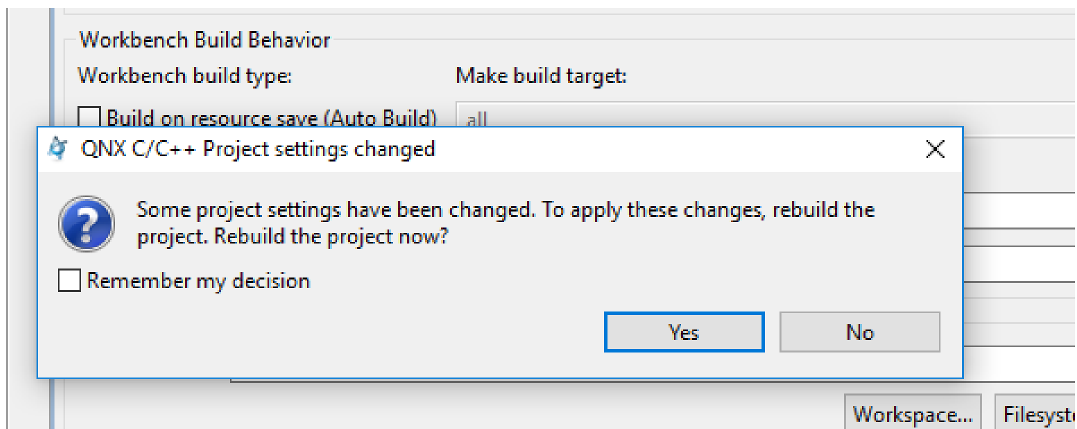
Cliquem a *add* i posem les que necessitem:



En el nostre cas es m (matemàtica) i la del socket.



Tornem a posar *apply* i tanquem.



Cliquem *yes* i es compilarà el programa directament.

D: CODI DE MATLAB PER OBTENIR ELS PARAMETRES NECESSARIS.

FULLCALIB.m

```
%%%%%%%%%%%% RELLENAR %%%%%%%%%%%%%%%%%%%
Tc_ext =[ -75.5118;
-26.3008;
395.5676];

Rc_ext =  [0.0267    0.9996      0.0098;
          0.4036    -0.0018     -0.9150;
          -0.9146    0.0284      -0.4034;]

KK =        [528.5307          0  239.5000;
                   0   520.4079  179.5000;
                   0          0    1.0000;]

%%%%%%%%%%%%%%%%%%%%%% EJECUTAR TAL CUAL %%%%%%%%%%%%
R_r_c = [0 -1 0 0;
          1 0 0 0;
          0 0 1 0;
          0 0 0 1];
T_r_c = [1 0 0   62.7;
          0 1 0 -277.9;
          0 0 1      0;
          0 0 0      1];
T_c_c = [1 0 0 Tc_ext(1,1);
         0 1 0 Tc_ext(2,1);
         0 0 1 Tc_ext(3,1);
         0 0 0           1];
R_c_c = [ Rc_ext(1,1) Rc_ext(1,2) Rc_ext(1,3) 0;
          Rc_ext(2,1) Rc_ext(2,2) Rc_ext(2,3) 0;
          Rc_ext(3,1) Rc_ext(3,2) Rc_ext(3,3) 0;
          0              0              0       1;]

M=T_c_c * R_c_c * R_r_c * T_r_c;
KK_inv= inv(KK);
R_3_3= [  M(1,1) M(1,2) M(1,3);
          M(2,1) M(2,2) M(2,3);
          M(3,1) M(3,2) M(3,3)];
t=[M(1,4) ; M(2,4); M(3,4)];

% %%%%%%%%%%%% lo que necesito %%%%%%%%%%%%%%%%%%%%%
% P_img=[xi;yi;1];
%
%
% Nom = R_3_3'*t; %cte
% Denom = R_3_3'* KK_inv *P_img;  %cte * punto
% Landa_2= Nom(3)/Denom(3);  %lo puedo calcular en c

%%%% nos quedamos con esta equacion:
```

```
%     P_rob = (R_3_3' * KK_inv * Landa_2* P_img) - R_3_3'*t
               %cte*punto - cte
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% P_rob =   (A*Landa_2*P_img)-B
% Xr=P_rob(1);
% Yr=P_rob(2);
%
% exportamos solo A y B:


%calculo denominador
A = R_3_3'* KK_inv %lo llamaré (A). falta multiplicar por
P_img para tenerlo:
     -0.0001   -0.0010    1.0589
     -0.0019    0.0000    0.3809
     0.0000   -0.0016   -0.2405

% calculo numerador
B = R_3_3'*t %lo llamare (B) :
     228.9466
     -210.4838
     -163.1246
```

E: GUÍA PER AL CALIBRATGE DE LA CÀMERA.

Transformacions i rotacions d'eixos. Aplicació pràctica sobre el sistema de visió del Robotino

Transformacions i rotacions d'eixos de coordenades necessaris per passar d'un píxel de la imatge capturada a un punt extern en coordenades robot

Albert Masip-Àlvarez*

15 de desembre de 2017

Departament d'Enginyeria de Sistemes,
Automàtica i Informàtica Industrial
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Aquest document aborda el problema de les transformacions (rotacions i traslacions) existents entre els diferents sistemes de referència que apareixen en el sistema Robotino. Per fer-ho, primer aborda el problema d'efectuar rotacions i traslacions a l'espai 2D i 3D per passar després a treballar sobre l'objecte d'aplicació.

1 Transformacions en coordenades homogènies

En l'aplicació del Robotino es treballarà modelant geometria per després transformar-la: traslladant-la a una altra posició o bé rotant-la respecte a un eix. S'operarà usant matrius, de manera que per efectuar aquestes transformacions, les matrius s'han de modificar lleugerament per dos motius:

- Per a que no alterin de forma igual un vector i un punt, cosa que seria incorrecta.
- Per a poder efectuar algunes transformacions afins com la translació, impossibles d'efectuar multiplicant matrius si no s'usen coordenades homogènies.

*Basat en informació facilitada pel Professor Bernardo Morcego Seix.

És molt senzill convertir un punt cartesià a la seva representació homogènia. De fet, només cal afegir una nova coordenada a les típiques X, Y i Z. S'afegeix la component W d'aquesta manera: El punt $P1 = (x1, y1, z1)$ en cartesianes esdevé $P1 = (x1, y1, z1, w1)$ en homogènies.
El valor típic per a la nova component és $W = 1$. D'aquesta manera, el cas anterior queda modificat així: El punt $P1 = (x1, y1, z1, 1)$ en homogènies.
Aquest conveni permet operar transformant un punt en un altre punt sense haver d'efectuar una operació diferent en cada cas.
En el cas que W sigui diferent de 1, caldrà efectuar una senzilla operació per a transformar un punt homogeni a cartesià. Si es té el punt:

$$P1 = (x1, y1, z1, w1)$$

en coordenades homogènies, aleshores en cartesianes el punt és:

$$P1 = (x1/w1, y1/w1, z1/w1).$$

És a dir, que es normalitza cadascuna de les components XYZ del punt per la seva component W. En el cas de que $W = 1$ no s'ha de fer res perquè la divisió és òbvia, però pot passar que interessi variar W i aleshores no es poden usar les components XYZ correctament fins que no s'hagin normalitzat seguint el procés anteriorment esmentat.

Per tant, es pot afirmar que:

$$P = (1, 2, 3, 1) = (2, 4, 6, 2) = (5, 10, 15, 5)$$

Amb tot, el sistema de coordenades homogènies assigna a cada punt una tupla de $n+1$ components denotada sovint $[x0, x1, ..., xn]$ que estan definides excepte multiplicació per escalars $[\lambda x0, \lambda x1, ..., \lambda xn]$, $\lambda \neq 0$, és a dir, que dues $n+1$-ples representen el mateix punt en tant que es conservi la proporció entre les seves components. Aquest tipus de coordenades són molt usades en **geometria projectiva**, on representen els vectors de l'espai vectorial usat per a definir cada espai projectiu. També es fan servir, per exemple, al denotar les rectes del pla com a equacions en forma general $Ax + By + C = 0$. L'ordre amb que s'aplicaran les translacions i rotacions sobre un punt en coordenades homogènies o bé sobre un sistema de coordenades serà important; aquestes operacions no són commutatives.

1.1 Translacions

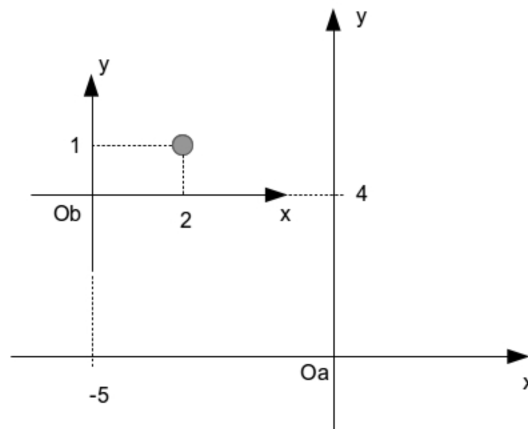
En coordenades homogènies, una translació d'un punt (a l'espai bidimensional, al pla 2D) es realitza mitjançant una matriu de translació amb la forma:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

essent t_x, t_y les components de la translació en els eixos x, y respectivament. Observi's la matriu identitat d'ordre 2 a la part superior esquerra de la matriu de translació.
Quan aquesta translació es realitza per traslladar els eixos de coordenades (per expressar un punt estàtic del pla en diferents sistemes de referència) el vector de translació cal posar-lo canviat de signe:

$$\begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Vegis, com a exemple, el punt de la figura adjunta. Expressat (en coordenades homogènies) en el sistema de referència $O_aX_aY_a$ aquest punt és $[-3, 5, 1]^T$ (es pot deduir fàcilment a partir del dibuix).
Vegi's que el sistema de referència $O_bX_bY_b$ està traslladat amb un vector (en



dues dimensions) de translació $[-5, 4]$ respecte el sistema $O_aX_aY_a$. Si es vol expressar el punt estàtic en el nou sistema de referència $O_bX_bY_b$ només caldrà multiplicar el punt (expressat en coordenades homogènies) en el sistema $O_aX_aY_a$ per la matriu de translació dels eixos (observi's que s'han canviat els signes de les components del vector de translació pel fet de traslladar eixos):

$$\begin{bmatrix} 1 & 0 & -(-5) \\ 0 & 1 & -(4) \\ 0 & 0 & 1 \end{bmatrix} \cdot [-3, 5, 1]^T = [2, 1, 1]^T$$

On $[2, 1, 1]^T$ correspon amb el punt expressat en el sistema $O_bX_bY_b$.
En tres dimensions (x, y, z) la translació d'eixos no difereix massa del que s'ha fet en dues dimensions:

$$\begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Observi's la matriu identitat d'ordre 3 a la part superior esquerra de la matriu de translació.
Seguint l'exemple anterior, però tractat en tres dimensions, la seva solució pel punt expressat originalment en el sistema de coordenades $O_aX_aY_aZ_a$ esdevé ara:

$$\begin{bmatrix} 1 & 0 & 0 & -(-5) \\ 0 & 1 & 0 & -(4) \\ 0 & 0 & 1 & -(0) \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot [-3, 5, 0, 1]^T = [2, 1, 0, 1]^T$$

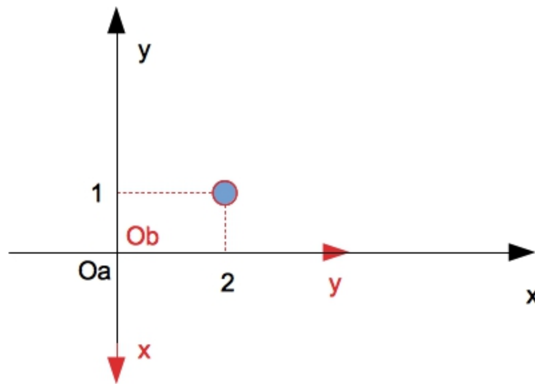
que correspon amb el punt expressat en coordenades $O_bX_bY_bZ_b$.

1.2 Rotacions

Per efectuar la rotació en un angle θ d'un punt en coordenades homogènies sobre el pla (en 2D) cal utilitzar la matriu:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La part pròpia d'aquesta matriu que correspon a la rotació (on apareixen les funcions trigonomètriques) és una matriu ortonormal, fet que vol dir que el seu determinant és unitari i que la seva inversa és igual a la seva trasposta, alhora que aquesta equival a rotar el mateix angle però canviat de signe (l'angle oposat).
En el cas que un mateix punt fix es vulgui expressar en un sistema de coordenades diferent caldrà canviar el signe de l'angle rotat o aplicar la trasposta o bé la inversa de la matriu de rotació.
Serveixi d'exemple una rotació de -90º com la de la figura: Es pot observar



que l'angle de rotació existent entre el sistema de referència a respecte el b és de -90º. Malgrat això, com que el que es vol és expressar el punt donat en el sistema de referència b llavors caldrà rotar l'angle canviat de signe, és a dir

$+90^o$, ja que es roten els eixos. Considerant el punt $P_a = [2, 1, 1]^T$, s'aplica la rotació d'eixos:

$$P_b = \begin{bmatrix} \cos(90^o) & -\sin(90^o) & 0 \\ \sin(90^o) & \cos(90^o) & 0 \\ 0 & 0 & 1 \end{bmatrix} P_a$$

d'on es desprèn que:

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix}$$

que correspon, precisament, amb el punt expressat en coordenades b.
La rotació en tres dimensions esdevé una mica més complexa. Cal, en primer lloc, establir l'eix de rotació entorn del qual s'efectuarà el gir; la forma de fer-ho és a través d'un vector de mòdul unitari denominat $\hat{n}$ i que té tres components:

$$\hat{n} = [\hat{n}_x, \hat{n}_y, \hat{n}_z]$$

Per poder realitzar el producte vectorial cal expressar aquest vector en forma matricial amb la finalitat d'obtenir la matriu de rotació en coordenades homogènies i poder-la aplicar al punt expressat en 3D:

$$[\hat{n}]_\times = \begin{bmatrix} 0 & -\hat{n}_z & \hat{n}_y \\ \hat{n}_z & 0 & -\hat{n}_x \\ -\hat{n}_y & \hat{n}_x & 0 \end{bmatrix}$$

Amb tot, la matriu de rotació es pot obtenir mitjançant la fórmula de Rodrigues[1]:

$$R(\hat{n}, \theta) = I + \sin(\theta)[\hat{n}]_\times + (1 - \cos(\theta))[\hat{n}]^2_\times$$

Aplicant aquestes expressions a l'exemple de rotació anterior:

$$\hat{n} = [0, 0, 1]$$

degut a que la rotació dels eixos x, y es realitza sobre l'eix z.

$$[\hat{n}]_\times = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

S'aplica la fórmula de Rodrigues per obtenir la matriu de rotació:

$$R(\hat{n}, 90^o) = I + \sin(90^o)[\hat{n}]_\times + (1 - \cos(90^o))[\hat{n}]^2_\times = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

[1] A la teoria de rotació tridimensional, la fórmula de rotació de Rodrigues (provinent del nom d'Olinde Rodrigues, matemàtic francès del segle XIX) és una forma eficient per a determinar la rotació d'un punt (o sistema de referència) a l'espai, donat un eix i l'angle de rotació.

que, expressada en coordenades homogènies, resulta ésser:

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

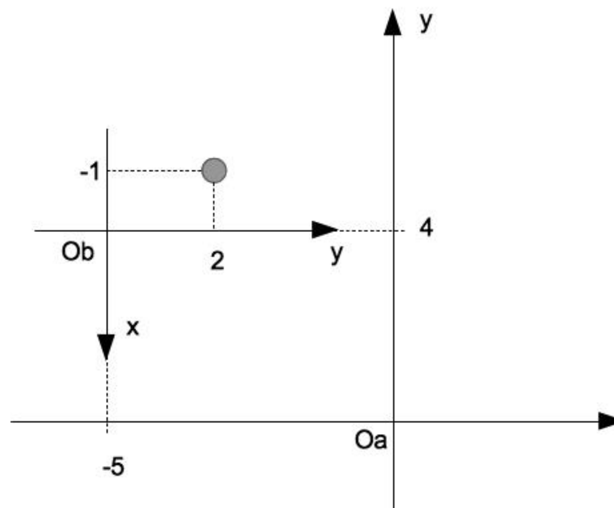
Si s'aplica aquesta matriu sobre el punt expressat en 3D sobre el sistema de referència a

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 0 \\ 1 \end{bmatrix}$$

s'obté el punt expressat en el sistema b.

1.3 Translacions i rotacions

Consideri's, alternativament, la modificació de l'exercici anterior següent: a banda de la translació dels eixos, s'hi ha efectuat, també una rotació de -90º (90º en sentit de les agulles del rellotge).
És ben sabut que l'ordre amb què s'efectuïn les transformacions (primer la



translació i després la rotació o a l'inrevés) és cabdal, no s'arriba al mateix resultat fent-ho d'una manera o una altra. Es proposa començar per la translació de $[-5, +4]$ per passar després a la rotació de -90º. Com que el punt a l'espai queda fix i el que es transformen són els eixos caldrà posar el vector de translació canviat de signe i trasposar els elements de la matriu de rotació (o bé invertir la matriu de rotació, o bé canviar de signe l'angle a

rotar). Es pot escriure, doncs, que:

$$P_b = \begin{bmatrix} \cos(90^o) & -\sin(90^o) & 0 \\ \sin(90^o) & \cos(90^o) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -(-5) \\ 0 & 1 & -(4) \\ 0 & 0 & 1 \end{bmatrix} P_a$$

essent P_a, P_b els punts expressats en els diferents sistemes de referència. Observi's el canvi de signe de l'angle de rotació degut a que es roten els eixos.

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -(-5) \\ 0 & 1 & -(4) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -3 \\ 5 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix}$$

Si es trasllada el mateix exemple però ara en coordenades 3D, les matrius de transformacions en coordenades homogènies esdevenen d'ordre 4:

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -(-5) \\ 0 & 1 & 0 & -(4) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -3 \\ 5 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 0 \\ 1 \end{bmatrix}$$

que correspon amb el punt expressat en 3D i en coordenades homogènies en el sistema de referència b.

1.4 Translacions i rotacions en una sola matriu

Les translacions i rotacions d'eixos que s'han efectuat per etapes a l'apartat anterior es poden realitzar en una sola matriu, en un sol pas. Aquesta matriu de rotació-translació (o translació-rotació) prendrà una o altra forma depenent de quin sigui l'ordre a l'hora d'efectuar les operacions, i.e. depenent de si la translació es fa primer o bé en segon lloc.
Quan la rotació es fa en primer lloc i la translació es fa després la matriu de rotació-translació es pot observar a l'operació:

$$P_b = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} * P_a$$

on es permet expressar un punt P_a en un nou sistema de referència b. A l'expressió es considera que R, t ja es troben amb **el signe de l'angle de rotació i de la translació canviats** pel fet que es mouen eixos i el punt és fix.
Es comprovarà considerant les matrius de translació de $[-5, 4, 0]^T$ i rotació de -90° d'eixos de l'exemple anterior:

$$\begin{bmatrix} 1 & 0 & 0 & -(-5) \\ 0 & 1 & 0 & -(4) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 5 \\ 1 & 0 & 0 & -4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Observi's que a l'operació s'ha posat la rotació a la dreta perquè és la primera transformació que es farà al punt i la translació a l'esquerra pel fet d'aplicar-se en segon lloc.
Per una altra banda, si es vol resumir en una sola matriu una translació feta en primer lloc i la posterior rotació, la matriu de translació-rotació es pot observar a l'operació:

$$P_b = \begin{bmatrix} R & R * t \\ 0 & 1 \end{bmatrix} * P_a$$

on es permet expressar un punt P_a en un nou sistema de referència b. A l'expressió es considera que R, t ja es troben amb **el signe de l'angle de rotació i de la translació canviats** pel fet que es mouen eixos i el punt és fix.
Es comprovarà considerant les matrius de translació de $[-5, 4, 0]^T$ i rotació de -90° d'eixos de l'exemple anterior:

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -(-5) \\ 0 & 1 & 0 & -(4) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 4 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

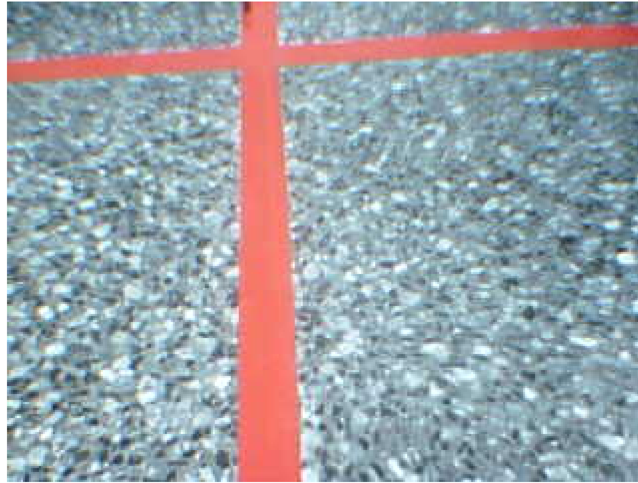
Observi's que a l'operació s'ha posat la translació a la dreta perquè és la primera transformació que es farà al punt i la rotació a l'esquerra pel fet d'aplicar-se en segon lloc. Observi's, per damunt de tot, com el vector de rotació es veu afectat per la rotació de 90°, fent que una translació d'eixos de $[-5, 4, 0]^T$ es converteixi en una translació rotada de $[-4, -5, 0]^T$. A la matriu resultant, aquest vector es troba canviat de signe pel fet de traslladar eixos, com ja se sap.

2 Procediment a seguir en el Robotino

2.1 Dispositiu de visió al Robotino

El Robotino disposa d'una càmera que captura imatges en color de 320x240 píxels a un màxim de 30fps ("frames"per segon). La seva funció serà la de realitzar captures de l'entorn (frontal) del Robotino amb la finalitat de recollir la cinta adhesiva de color vermell enganxada al terra del laboratori i que serà la que ell hagi de seguir. La càmera es pot orientar i, per aquesta aplicació, caldrà fixar-la de manera que reculli el terra, davant seu. Aquest terra (de terratzo micrograà) està format per rajoles de 40x40cm; s'hi aprecien els colors blanc, negre i gris de diferents tonalitats. Per aquest motiu, la "pista"per la que ha de circular el Robotino no serà ni negra, ni blanca ni grisa; s'ha triat la vermella perquè s'ha vist que dóna millors resultats en el seu processat d'imatge per aquest terra de fons donat.

La idea general és que les pistes per les que ha de circular el Robotino es

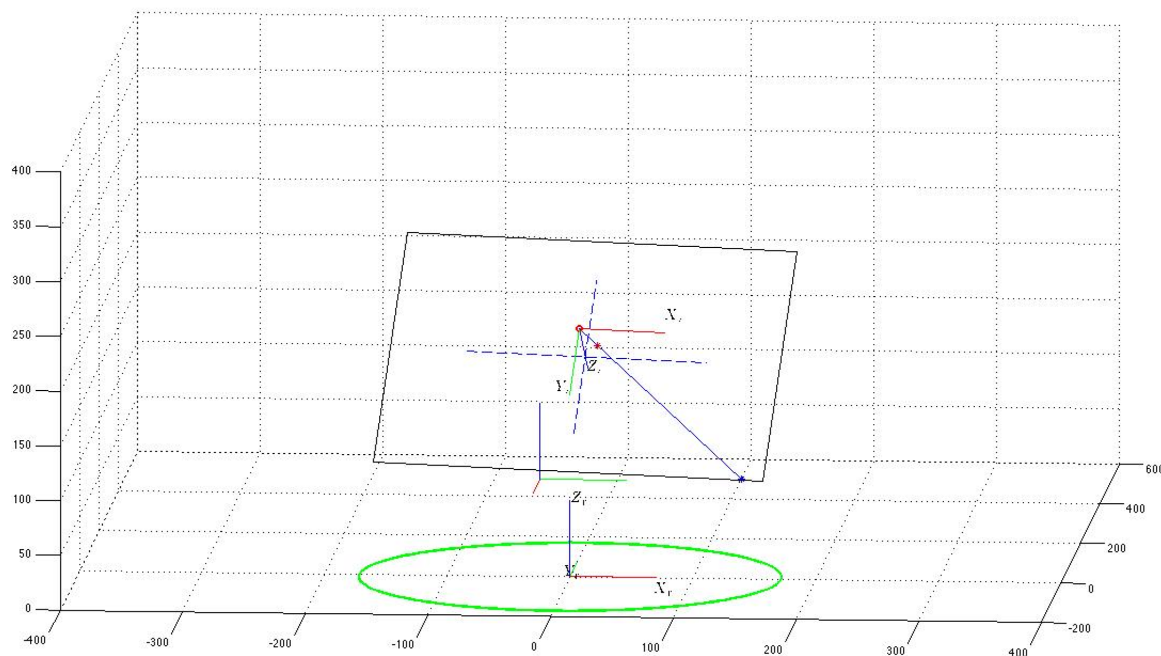


creuen perpendicularment amb d'altres formant un entramat de camins possibles. L'amplada de la cinta vermella és de 19 mm. El que es vol aconseguir és guiar el robot al llarg dels camins tot seguint la línia del terra fent alhora que sigui capaç de canviar de direcció en una cruïlla.
En resum, i sense considerar tots els passos intermitjos que conformen el procediment, el problema consisteix a capturar una imatge i conèixer l'angle i la distància que separa el robot del seguiment perfecte de la línia traçada a terra.
Per començar, el problema que cal resoldre és conèixer quines transformacions (rotacions i translacions) cal efectuar per passar un punt expressat en coordenades del sistema de referència del robot a expressar-lo en coordenades de la imatge (píxel). Finalment, el problema de major interès és precisament l'invers: conèixer les transformacions que s'han de dur a terme per saber quin punt a l'espai del Robotino representa un píxel determinat de la imatge capturada.

2.2 Sistemes de referència existents a l'aplicació

La figura adjunta il·lustra els quatre sistemes de referència que tenen lloc durant el procés de calibratge del Robotino. S'hi pot veure el sistema de referència del propi robot, el sistema de callibratge (rotat -90º i desplaçat sobre el mateix pla), el de la càmera (aixecat en z) i el de la imatge 2D capturada.

Caldrà determinar les relacions entre els diferents sistemes de referència, i.e. conèixer les transformacions existents entre ells.



2.3 Interpretació i maneig dels paràmetres intrínsecs i extrínsecs de la càmera

La font d'informació d'aquest apartat es pot trobar a la URL `http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/parameters.html`.
El punt de partida del problema de seguiment d'una línia marcada a terra té lloc quan es disposa dels paràmetres intrínsecs fc, cc, α_c, kc, o bé KK (com a aglutinació de tots els anteriors), i extrínsecs Tc_{ext}, Rc_{ext} de la càmera utilitzada. A partir d'aquests es podran determinar les transformacions necessàries per saber on es troba, en coordenades del robot, un punt determinat capturat a la imatge de la càmera i poder decidir sobre la navegació del robot.

2.4 Paràmetres intrínsecs de la càmera

Els paràmetres intrínsecs contenen informació pròpiament de la càmera utilitzada i de la seva òptica: la seva distància focal, l'eix principal, etc... Aquests paràmetres es poden recollir tots ells en una matriu denominada **matriu de paràmetres intrínsecs de la càmera** i que porta per nom KK. Aquesta matriu permet relacionar, mitjançant una equació lineal, el vector de coordenades de píxel (subíndexos p) amb el vector de coordenades normalitzat

(distorsionat, subíndexos d):

$$\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = KK * \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}$$

Cal explicar que l'equació anterior és vàlida per tots els punts de la recta en el sentit z ergo cal dividir el vector resultant per z_d per tenir el punt sobre el pla.
La matriu KK ve definida per:

$$\begin{bmatrix} fc(1) & \alpha_c * fc(1) & cc(1) \\ 0 & fc(2) & cc(2) \\ 0 & 0 & 1 \end{bmatrix}$$

essent

- fc. La longitud o distància focal, expressada en píxels i emmagatzemada en un vector de dues components x, y, que es corresponen amb notacions de l'estil f_x, f_y o bé $fc(1)$,$fc(2)$.
- *cc*. Les coordenades x, y del punt principal o centre òptic. Les càmeres solen tenir l'origen de coordenades centrat mentre que les imatges solen tenir l'origen dels eixos coordenats a la cantonada superior esquerra; cal, doncs, fer aquesta translació.
- α_c. El coeficient d'inclinació, que correspon a l'angle format entre els eixos x i y dels píxels. Val 0 si els píxels són quadrats i diferent de zero quan són romboides.

Aquesta matriu quadrada de paràmetres intrínsecs de la càmera és ortogonal, i.e. el producte d'ella mateixa per la seva transformada resulta en la identitat:

$$KK * KK^{-1} = I$$

Malgrat aquest fet, la matriu KK no és ortonormal perquè el seu determinant no val 1. A l'entorn Matlab, aquesta matriu s'emmagatzema en una variable anomenada KK un cop finalitzat el procediment de calibratge. Cal observar que les dues components del vector fc de la distància focal (un valor únic en mm) expressat en unitats de píxels horitzontals i verticals són, habitualment, molt similars. La ratio $fc(2)/fc(1)$, sovint anomenada "aspect ratio", és diferent d'1 si el píxel de la matriu CCD no és quadrat. D'aquesta manera el model de la càmera sosté, naturalment, píxels no quadrats. A més, el coeficient α_c recull l'angle entre els eixos x, y del sensor, de forma que es contempla que els píxels no siguin rectangulars.
A banda de computar els valors estimats pels paràmetres intrínsecs suara desenvolupats, l'eina de calibratge de paràmetres intrínsecs també proporciona valors estimats per a les incerteses sobre aquests paràmetres. Les

variables s'anomenen **fc_error, cc_error, kc_error, alpha_c_error**. Aquests vectors corresponen, aproximadament, a tres cops la desviació estàndard dels errors d'estimació.
En el problema que ens ocupa, aquest calibratge ja es va efectuar en el seu moment i els resultats es guardaren en un arxiu. Cal, doncs, carregar els paràmetres intrínsecs de la pròpia càmera del Robotino; aquests es troben a l'arxiu `Calib_Results.mat` tot fent a l'espai de treball de Matlab: `load Calib_Results.mat`.
Aquest arxiu conté, a banda d'altres variables, la matriu KK (matriu de la càmera)

$$KK = \begin{bmatrix} 404.8412 & 0 & 143.9544 \\ 0 & 408.8882 & 108.0107 \\ 0 & 0 & 1.0000 \end{bmatrix}$$

i els errors (incerteses) resultants del calibratge dels paràmetres intrínsecs.

2.5 Paràmetres extrínsecs de la càmera. Pas de coordenades de calibratge a coordenades de càmera

Seguidament es procedeix a calibrar els paràmetres extrínsecs de la càmera, i.e. determinar les transformacions (rotacions i translacions) necessàries per passar un punt de les coordenades de calibració (coordenades món) a coordenades de la pròpia càmera. Per aquest motiu, el programa de calibratge necessitarà una imatge de calibratge, un escaquer pla del qual es coneixen les mides dels quadres. S'executa, doncs, el programa principal al workspace:
`»extrinsic_computation`
El programa respon dient: `Computation of the extrinsic parameters from an image of a pattern. The intrinsic camera parameters are assumed to be known (previously computed)`
i cal donar el nom (sense extensió) de l'arxiu imatge per a la calibratge (en el cas d'estudi és **calib01**):

```
Image name (full name without extension): calib01
Image format: ([]='r'='ras', 'b'='bmp', 't'='tif', 'p'='pgm', 'j'='jpg',
'm'='ppm') b
```

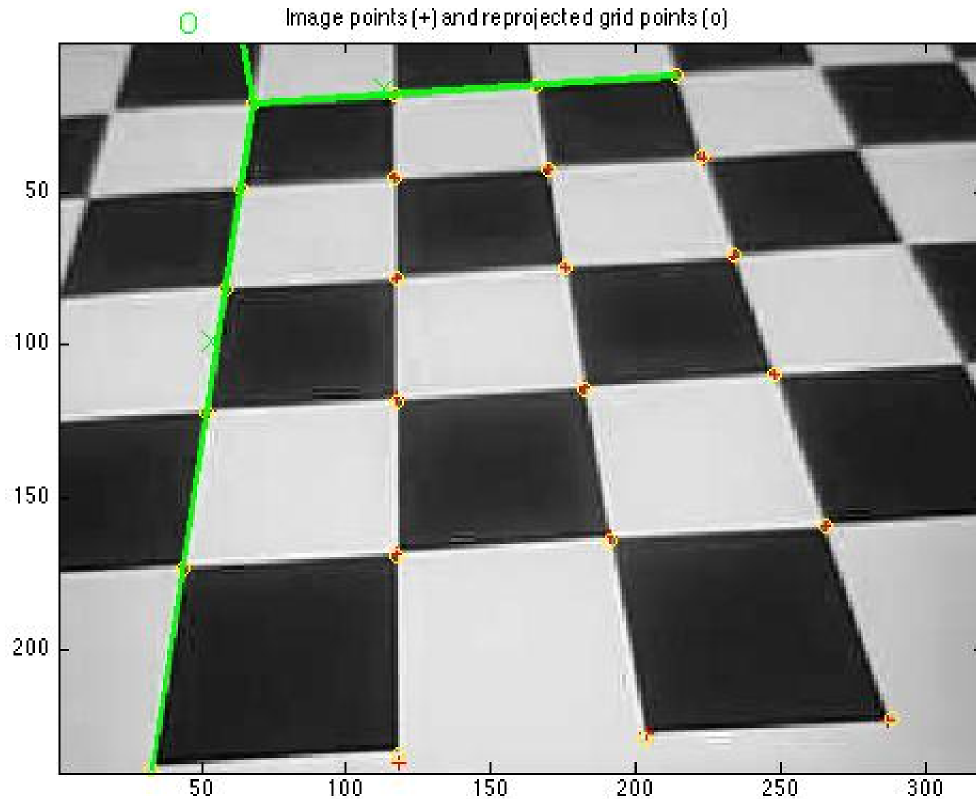
I es carrega la imatge. A continuació el programa demana la longitud (en píxels) de la finestra per a l'extracció dels cantons dels quadrats (per defecte, 5 píxels en x i y):

```
Extraction of the grid corners on the image
Window size for corner finder (wintx and winty):
wintx ([] = 5) =
winty ([] = 5) =
Window size = 11x11
```

Seguidament cal marcar els vèrtexos d'una regió de l'escaquer formant un rectangle que contingui quadrats blancs i negres sencers. Caldria que aquesta regió sigui prou generosa i val a dir que sempre **cal començar pel cantó superior esquerre del rectangle que es vulgui dibuixar**. El programa ho indica dient:

```
Click on the four extreme corners of the rectangular complete pattern
(the first clicked corner is the origin)...
```

A voltes, el programa no és capaç de detectar quants quadrats es troben



continguts dins la regió en forma de rectangle que l'usuari ha marcat i ho demana manualment:

```
Could not count the number of squares in the grid.  Enter manually.
Number of squares along the X direction ([]=10) = 5
Number of squares along the Y direction ([]=10) = 3
```

Finalment, demana la longitud (en mm) dels quadrats en cada direcció x, y:

```
Size dX of each square along the X direction ([]=30mm) = 50
Size dY of each square along the Y direction ([]=30mm) = 50
```

El programa procedeix a l'extracció de cantonades (vèrtexos dels quadrats) i, en acabat, mostra a la pantalla els paràmetres extrínsecs resultants del calibratge efectuat:

```
Corner extraction...
Extrinsic parameters:
Translation vector:  Tc_ext = [ -79.045193 -89.029045 420.618406
]
Rotation vector:  omc_ext = [ 1.783581 1.664442 -0.785025 ]
Rotation matrix:  Rc_ext = [ 0.052710 0.997948 -0.036359
0.662811 -0.062195 -0.746199
-0.746929 0.015233 -0.664729 ]
Pixel error:  err = [ 0.37647 0.89018 ]
```

L'error de píxel és important perquè cal que sigui petit; si no és així, caldria repetir el procediment fins que sigui prou baix. El que s'ha mostrat és acceptablement baix.

Els programadors de l'eina de calibratge informen que cal fer servir els resultats de la següent forma:

$$P_{cam} = Rc_ext * P_{cal} + Tc_ext$$

De manera que s'expressa un punt, que originalment es troba en coordenades del sistema de calibratge P_{cal}, en coordenades de la càmera P_{cam}, i que cal no confondre amb coordenades imatge en aquest punt. Cal observar que es fa primer la rotació i després la translació.

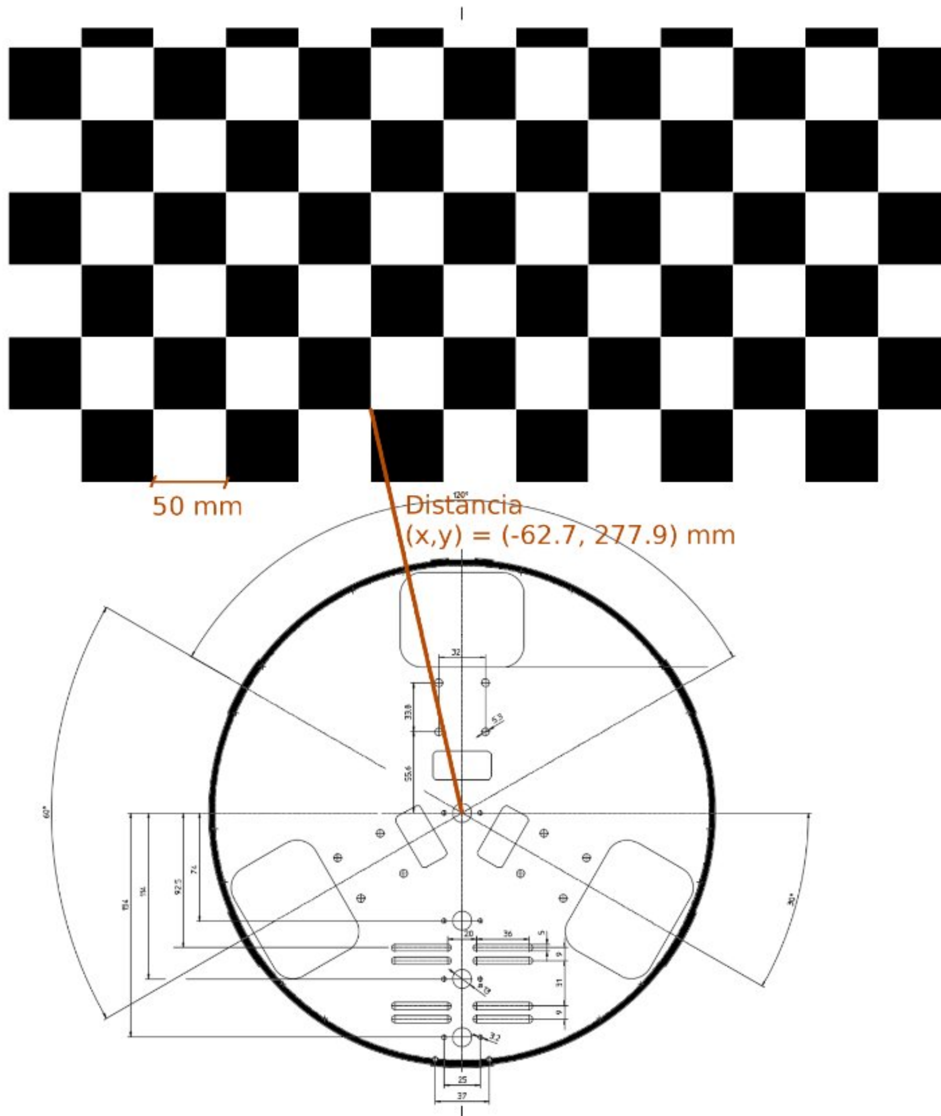
2.6 Pas de coordenades robot a coordenades de calibratge

La figura de la pàgina següent (que es subministra amb el guió de la pràctica) mostra la distància existent entre un punt conegut de l'escaquer i el centre del robot. A partir d'ella i de la imatge resultant del calibratge es pot determinar quina és la translació i la posterior rotació (o a l'inrevés) que cal efectuar sobre el sistema de coordenades del robot per passar a sistema de coordenades del calibratge. Es desprèn fàcilment que la translació és de $[-62.7, 477.9, 0]^T$ i la rotació és de -90º. Com que el que es transformen són els eixos (només es vol reinterpretar un punt de l'espai del robot en l'espai de calibratge) caldrà prendre la translació canviada de signe i l'angle de 90º, en lloc de -90º. Les matrius, doncs, esdevenen:

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

per a la rotació, i

$$\begin{bmatrix} 1 & 0 & 0 & -(-62.7) \\ 0 & 1 & 0 & -(477.9) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



per a la translació.
Donades aquestes matrius, caldrà aplicar primer la translació i després la rotació degut a que s'han determinat per fer-ho d'aquesta manera. Es comprovarà amb un exemple que aquestes transformacions estan correctament determinades; prengui's el punt $[-62.7, 277.9, 0, 1]^T$ en coordenades Robotino. Aquest punt correspon al punt assenyalat a la figura proporcionada en el guió de la pràctica de calibratge. S'espera que quan es transformi aquest punt als eixos de calibratge aquest esdevindrà $[200, 0, 0, 1]^T$ (es pot deduir fàcilment perquè es troba 4 caselles en sentit positiu sobre l'eix x de

coordenades de calibratge).

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -(-62.7) \\ 0 & 1 & 0 & -(477.9) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (-62.7) \\ (277.9) \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 200 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Com era d'esperar. Si, alternativament, es prengués el vèrtex immediat de la seva dreta (en coordenades robot $[-12.7, 277.9, 0, 1]^T$) el resultat seria:

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -(-62.7) \\ 0 & 1 & 0 & -(477.9) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (-12.7) \\ (277.9) \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 200 \\ 50 \\ 0 \\ 1 \end{bmatrix}$$

I es comprova que, en eixos de calibratge, aquest nou vèrtex té un desplaçament de 50mm en l'eix y a diferència de l'anterior exemple.

2.7 Pas de coordenades robot a coordenades de la càmera

Es farà en dos passos: primer es passarà el punt fix de coordenades robot a coordenades de calibratge (primer la translació i després la rotació, perquè les matrius s'han definit d'aquesta manera) i, en segon lloc, es passarà el punt de coordenades calibratge a coordenades de la càmera (primer la rotació i després la translació, perquè així ho indica l'ajuda de la toolbox de calibratge extern). En definitiva, caldrà realitzar la següent operació:

$$P_{cam} = T_{cal \to cam} * R_{cal \to cam} * R_{rob \to cal} * T_{rob \to cal} * P_{rob}$$

Essent P_{cam}, P_{rob} els punts 3D expressats en coordenades homogenies i en el sistema de coordenades de la camera (encara no de la imatge) i del robot, respectivament. Les matrius venen definides per:

$$T_{cal \to cam} = \begin{bmatrix} 1 & 0 & 0 & -79.045193 \\ 0 & 1 & 0 & -89.029045 \\ 0 & 0 & 1 & 420.618406 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{cal \to cam} = \begin{bmatrix} 0.052710 & 0.997948 & -0.036359 & 0 \\ 0.662811 & -0.062195 & -0.746199 & 0 \\ -0.746929 & 0.015233 & -0.664729 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{rob \to cal} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{rob\to cal} = \begin{bmatrix} 1 & 0 & 0 & -(-62.7) \\ 0 & 1 & 0 & -(477.9) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Observi's que $R_{cal\to cam}$ correspon a la matriu Rc_ext proporcionada per la toolbox de calibratge extern però expressada ara en coordenades homogènies. No s'ha trasposat ni invertit la matriu de rotació resultant pel fet que serveixi per fer una rotació d'eixos perquè cal tenir present que el programador de l'eina ja s'ha encarregat que aquesta matriu de rotació serveixi, precisament, per expressar un punt fix en diferents sistemes de coordenades (rotació d'eixos). Recordi's que:

$$P_{cam} = Rc_ext * P_{cal} + Tc_ext$$

Observi's també que $T_{cal\to cam}$ correspon al vector Tc_ext proporcionat per la toolbox de calibratge extern però en forma de matriu en coordenades homogènies. No s'ha canviat de signe el vector de translació perquè (novament) cal tenir present que el programador de l'eina ja s'ha encarregat que aquest vector de translació serveixi, precisament, per expressar un punt fix en diferents sistemes de coordenades (translació d'eixos).
Serveixi d'exemple el punt anteriorment considerat del guió de la pràctica $P_{rob} = [-62.7, 277.9, 0, 1]^T$. En aplicar les transformacions definides anteriorment:

$$P_{cam} = T_{cal\to cam} * R_{cal\to cam} * R_{rob\to cal} * T_{rob\to cal} * \begin{bmatrix} -62.7 \\ 277.9 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -68.5 \\ (43.53) \\ 271.23 \\ 1 \end{bmatrix}$$

El punt es troba ara expressat en coordenades de la càmera; aquest prendrà el seu significat quan s'expressi en coordenades de la imatge, a l'apartat següent.

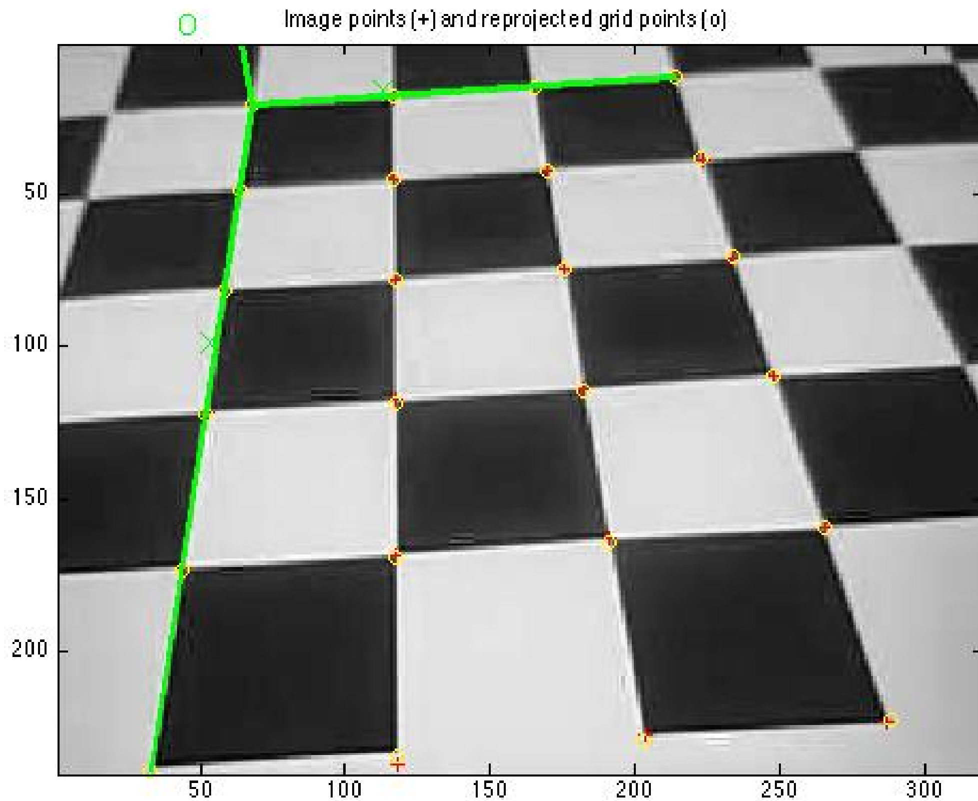
2.8 Pas de coordenades robot a coordenades de la imatge

A l'apartat anterior s'ha vist com passar un punt expressat originalment en coordenades robot a un punt en coordenades de la càmera. Si aquesta transformació es premultiplica per la matriu KK de paràmetres intrínsecs de la càmera s'acaba obtenint el punt 2D sobre la imatge de 320x240 píxels.

$$\lambda * P_{imatge} = [KK, [0, 0, 0]^T] * T_{cal\to cam} * R_{cal\to cam} * R_{rob\to cal} * T_{rob\to cal} * P_{rob}$$

Si s'aplica aquesta operació al complet sobre el punt $P_{rob} = [-62.7, 277.9, 0, 1]^T$ el resultat que s'obté és $\lambda * P_{imatge} = [11312, 47096, 271]^T$, d'on es desprèn que $\lambda = 271$ pel fet que el punt 2D obtingut P_{imatge} es troba expressat en

coordenades homogènies i la seva tercera component cal que sigui 1. De fet, λ és un paràmetre que té a veure amb la profunditat i la distància focal. En dividir les tres components del vector per λ el resultat és que $P_{imatge} = [41.7068, 173.6379, 1.0000]^T$, que coincideix, com es pot comprovar, amb el punt original però traduït (arrodonint) a coordenades de la imatge $P_{imatge} = [42, 174, 1]^T$ (en píxels): L'origen d'eixos de la imatge correspon al vèrtex su-



perior esquerra de la pantalla, l'eix x de la imatge és l'horitzontal (apuntant cap a la dreta, va de 1 a 320) i el de les y el vertical apuntant cap avall (i que va de 1 a 240).

2.9 Inversió del pas: quin punt real (en coordenades robot) representa un píxel de la imatge?

El problema a resoldre en aquest punt és l'invers a l'anterior. De fet consisteix en el que definitivament serà de gran interès per a la navegació del robot: **quines coordenades 3D en el sistema de referència del Robotino té un determinat píxel 2D de la imatge**?

Es resoldrà el problema partint del resultat anterior, però ara invertint els passos.
Sigui M la matriu que permet expressar un punt 3D $P_{robot} = [x_{robot}, y_{robot}, z_{robot}, 1]^T$ en coordenades robot a un punt 3D $P_{camera} = [x_{camera}, y_{camera}, z_{camera}, 1]^T$ expressat en eixos de la càmera:

$$M = T_{cal\to cam} * R_{cal\to cam} * R_{rob\to cal} * T_{rob\to cal}$$

Com s'ha vist a l'apartat anterior, per expressar un punt 3D $P_{robot} = [x_{robot}, y_{robot}, z_{robot}, 1]^T$ en coordenades robot a un punt 2D $P_{imatge} = [x_{imatge}, y_{imatge}, 1]^T$ expressat en píxels de la imatge, es pot escriure que:

$$\lambda * P_{imatge} = [KK, [0,0,0]^T] * T_{cal\to cam} * R_{cal\to cam} * R_{rob\to cal} * T_{rob\to cal} * P_{robot}$$

$$\lambda * P_{imatge} = [KK, [0,0,0]^T] * M * P_{robot}$$

Caldrà, doncs, invertir (raonadament) les matrius que apareixen a l'expressió anterior per poder passar de coordenades de la imatge a coordenades del Robotino:

$$M^{-1}[KK]^{-1}\lambda * P_{imatge} = P_{robot}$$

Cal anar especialment alerta amb la inversió de la matriu M. Recordi's que aquesta matriu incorpora, sigui quin sigui el seu ordre, una translació i una rotació d'eixos. El càlcul de la seva inversa M^{-1} no es pot fer directament sinó que caldrà efectuar les següents operacions:

- Trasposar (o invertir) la part de la matriu original corresponent a la rotació.
- Canviar de signe la part de la matriu original corresponent al vector de translació, alhora que es premultiplica per la trasposta (o inversa) de la part de la matriu original corresponent a la rotació.

$$p_2 = M * p_1 \to p_1 = M^{-1} * p_2$$

essent

$$M = \begin{bmatrix} R_{3\times3} & t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix} \to M^{-1} = \begin{bmatrix} R_{3\times3}^T & -R_{3\times3}^T * t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

Cal fixar-se que, a l'expressió anterior, s'ha considerat que la matriu M original aplica primer una rotació i després una translació (veure secció 1.4 d'aquest document). Quan es determini la inversa de la matriu aquests passos quedaran canviats d'ordre (a la matriu M^{-1} primer es trasllada i després es rota). Més precisament, aquestes operacions consisteixen en traslladar primer els eixos a través del vector de translació original canviat de signe (per això es troba multiplicat per la trasposta/inversa de la matriu de rotació) per passar després a efectuar la rotació de l'angle original però canviat de

signe (per això es fa la trasposta o inversa de la part de la matriu original corresponent a la rotació), i.e. aquestes operacions estan basades en invertir els passos realitzats amb el pas directe.
Consideri's ara, per no deixar de banda aquesta situació, que la matriu M original recull una translació primer i després una rotació d'eixos:

$$M = \begin{bmatrix} R_{3\times3} & R_{3\times3} * t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix} \rightarrow$$

$$\rightarrow M^{-1} = \begin{bmatrix} R^T_{3\times3} & -R^T_{3\times3} * R_{3\times3} * t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} R^T_{3\times3} & -t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

Cal fixar-se que, a l'expressió anterior, s'ha considerat que la matriu M original aplica primer una translació i després una rotació (veure secció 1.4 d'aquest document). Quan es determini la inversa de la matriu aquests passos quedaran canviats d'ordre (a la matriu M^{-1} primer es rota i després es trasllada). Més precisament, aquestes operacions consisteixen en rotar primer els eixos a través de la rotació de l'angle original però canviat de signe (per això es fa la trasposta o inversa de la part de la matriu original corresponent a la rotació) per passar després a efectuar la translació del vector de translació original canviat de signe (per això es troba multiplicat per la trasposta/inversa de la matriu de rotació i acaba quedant la identitat) , i.e. aquestes operacions estan basades en invertir els passos realitzats amb el pas directe.
Com a conclusió final, val a dir que és indiferent quin és l'ordre entre translació/rotació que recull la matriu M original; per efectuar la seva inversa només cal trasposar/invertir la part de la matriu corresponent a la rotació i canviar de signe la part corresponent al vector de translació premultiplicat per la inversa/trasposta de la matriu de rotació.

Per realitzar aquest pas d'inversió descrit anteriorment de forma pràctica (per a poder-lo programar) es considerarà la matriu M de tal forma com si primer realitzés una rotació i després una translació:

$$M = \begin{bmatrix} R_{3\times3} & t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

D'aquesta manera, es pot escriure el pas directe (passar un punt de coordenades robot a coordenades imatge) així:

$$\lambda P_{imatge} = K * R * P_{robot} + K * t$$

essent K la matriu de paràmetres intrínsecs de la càmera, R la part de la matriu M corresponent a la rotació i t la part de la matriu M corresponent

a la translació. El punt 3D del robot no ve expressat en coordenades homogènies $P_{robot} = [x_{robot}, y_{robot}, z_{robot}]^T$ però el punt 2D de la imatge sí que ho està $P_{imatge} = [x_{imatge}, y_{imatge}, 1]^T$.
Precisament, és el pas invers el que interessa: a partir d'un píxel de la imatge es vol conèixer on es troba aquest punt en coordenades del Robotino. Es premultipliquen els termes dels dos membres de la igualtat anterior per la inversa de K i després per la inversa de R (o la seva trasposta) i en resulta:

$$R^T * K^{-1} * \lambda * P_{imatge} - R^T * t = P_{robot}$$

Com que la profunditat λ es desconeix cal prendre la tercera component del vector resultant de la igualtat anterior i igualar-la a zero (el robot es mou per la superfície plana del terra del laboratori):

$$[R^T * K^{-1} * P_{imatge}]_z * \lambda - [R^T * t]_z = 0$$

d'on es pot obtenir el valor de λ:

$$\lambda = \frac{[R^T * t]_z}{[R^T * K^{-1} * P_{imatge}]_z}$$

Amb la profunditat ara ja coneguda, es pot utilitzar l'expressió anteriorment deduïda:

$$P_{robot} = R^T * K^{-1} * \lambda * P_{imatge} - R^T * t$$

Es recorda que el punt 3D del robot no resultarà en coordenades homogènies $P_{robot} = [x_{robot}, y_{robot}, z_{robot}]^T$ però el punt 2D de la imatge sí que ho està $P_{imatge} = [x_{imatge}, y_{imatge}, 1]^T$.

F: PURE PURSUIT EXPLICAT PER ANÍBAL OLLERO.

Stability Analysis of Mobile Robot Path Tracking†

Aníbal Ollero* Guillermo Heredia**

* Dept. Ingeniería de Sistemas y Automática, Escuela Superior de Ingenieros, Universidad de Sevilla
Avenida Reina Mercedes, 41012 - Sevilla, Spain. Email: aollero@etsii.us.es

** The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA
On leave from *. Email: guiller@frc.ri.cmu.edu, guiller@obelix.cica.es

Abstract

This paper presents a new approach that analyzes the stability of a general class of path tracking algorithms taking into account the pure delay in the control loop. The analysis has been done for straight paths and paths of constant curvature. This has sufficient generality since most usual paths can be decomposed in pieces of constant curvature. The method has been applied to the pure pursuit path tracking algorithm, one of the most widely used. The experiments performed with a computer controlled HMMWV show good agreement with the theoretical predictions of the proposed method.

1 Introduction

Path tracking is a basic function of most mobile robot or autonomous vehicle control systems. The objective of path tracking is to generate control commands for the vehicle to follow a previously defined path by taking into account the actual position and the constraints imposed by the vehicle and its lower level motion controllers.

The estimation of the mobile robot position can be performed using several sources of data including the perception system (image, range data, sonars) as well as internal sensors (inclinometers, gyroscopes, accelerometers) and dead reckoning techniques. Navigation in structured environments is usually based on the tracking of visual features such as lines. In this case the position estimation with respect to the path to follow is given by the perception system. Thus, there are path tracking formulations [1] which respond to a controller incorporating real time image processing to estimate the vehicle position.

In more complex situation the path to follow is previously computed by a path planner. The objective of a robot path planner is to find a path from a start position to a goal position with no collisions while minimizing a cost measure. Furthermore, there are also path generation methods which interpolate between way points computed by a global path planner [2]. In these situations the path tracker input is the path defined by the path planning/generation system.

Finally, there are also architectures in which the perception and path planning are integrated with path tracking and low level control in order to provide fast response for real time obstacle avoidance [3].

Path tracking is directly related with the lateral vehicle motion and steering control. Vehicle´s control also involves speed control. Obviously, both are coupled problems. However, path tracking has been usually studied for constant velocity. Thus, the path tracking algorithm implements a steering control law by using the error between the current estimated vehicle position/orientation and the path to follow. The inputs of the path tracker are variables defining the state of the vehicle with respect to the path, and the output is the steering command to be executed by the low-level motion controllers.

From the point of view of the control law, linear proportional feedback of errors [4][5], and PI control algorithms [6], have been implemented by using error coordinate systems. These control laws are derived using kinematic relations.

The most significant dynamic effects in the vehicle´s control come from the vehicle-terrain interaction and from the dynamic of the actuation systems. The former is hard to consider in real time due to the complex relations involved and the difficulties in sensing. However, it can be shown from experimentation that the steering actuation dynamics is relatively easy to model and has a significant impact in the path tracking performance of several vehicles [7].

On the other hand, for short intervals, the kinematic equations can be linearized for constant velocity using vehicle´s fixed coordinates or path dependent coordinates. Thus, the path tracking problem can be formulated for every value of the vehicle speed as a linear control problem and the dynamic of the steering actuation system can be considered as additional linear equations. Thus, linear control methods can be applied for the vehicle automated steering [8][9]. Recently, Generalized Predictive Control (GPC) theory has

†This research has been partially supported by the CICYT projects TAP93-0581 and TAP94-1159-E, and the grant program for stays in foreign countries from the University of Seville.

0-8186-7108-4/95 $4.00 © 1995 IEEE

Authorized licensed use limited to: UNIVERSITAT POLITÈCNICA DE CATALUNYA. Downloaded on March 23, 2009 at 11:14 from IEEE Xplore. Restrictions apply.

been applied obtaining good results [10] when the vehicle is close the path in position and heading, and there are not significant perturbations. However if, due to measuring errors or perturbations, the vehicles separates from the path, the linearization conditions are violated and the tracking deteriorates.

Furthermore, fuzzy control methods have been also applied with good results both for automatic tuning of other methods [11] or to generate directly the steering command by using heuristic rules [12]. This strategy leads to a nonlinear control law on the lateral error, heading and distance from the vehicle to a goal point in the path. The only problem pointed out is the difficulty for designing and tuning the fuzzy controller.

Most existing path tracking implementations are pure pursuit strategies. Due to this reason, this paper considers pure pursuit as the basic technique. However, the conclusions can be extended to other algorithms.

The results of all these control laws greatly depend on parameter tuning. Thus, oscillations or even instability can arise for some values of the controller parameters. These instability conditions are related with both the vehicle´s characteristics and navigation conditions including speed, path to follow and terrain. This paper investigates the dynamic behavior of path tracking. Particularly stability conditions of the pure pursuit algorithm are studied.

The problem is not easy due to the non linear nature of the feedback tracking system. Thus, only few results have been presented in this significant problem. In [14] a non-linear control law is presented and the stability is proved. However, the method requires the assumption of perfect velocity tracking which is unrealistic in many vehicles, particularly in outdoor navigation. In [15], a nonlinear feedback is obtained, and asymptotic input-output stability and Lagrange stability of the overall system for straight line and circular paths are proven.

Furthermore, notice that, in addition of kinematic equations and the dynamic of the steering system, path tracking involves a pure delay in the control loop. This delay is due mainly to the mobile robot position estimation, particularly when involving environment perception, as well as to other computing and communication delays. The consideration of the pure delay in the path tracking loop introduces an additional complexity in the analysis.

Most path tracking methods have a parameter related with the selection of the goal point in the path to follow. This parameter has a significant effect on the tracking performance. In the pure pursuit method this critical parameter is the lookahead distance. In this paper we study the stability of the tracking system in terms of the lookahead.

In the existing stability criteria for time-delay systems, mainly two ways of approach have been adopted. Namely, one direction is to obtain stability conditions which do not include information on the delay. This approach generally provides nice algebraic conditions [16][17]. However, not using information on the delay necessarily causes conservativeness of the criteria. Reported delay-dependent criteria consider explicitly the delay [18][19], and correspond to the trade-off between simplicity and sharpness. In this paper, stability criteria are derived directly from the trascendental equations, giving the exact stability regions.

The remaining of the paper is organized as follows. Section 2 presents the formulation of the path tracking problem. The basic stability analysis is presented in section 3. Section 4 and 5 present the stability analysis with time delays of straight and circular paths. The experiments validating the proposed method are included in Section 6. Sections 7, 8 and 9 are for the Conclusions, Acknowledgments and References.

2 Path tracking

Let (x,y,z) be the vehicle´s coordinates, (ψ, ϕ, θ) the orientation angles, and $\gamma = d\theta/ds$ the vehicle's radius of curvature. For 2D navigation the posture of the vehicle is given by (x,y,θ). Then, the control problem can be formulated in terms of driving the vehicle from its actual configurations (x,y,θ) to the desired or target configurations (x^d, y^d,θ^d).

Let the vehicle´s motion equation be given by:

$$dx = -\sin\theta ds \quad (1)$$

$$dy = \cos\theta ds \quad (2)$$

$$d\theta = \gamma ds \quad (3)$$

where γ is the vehicle´s curvature, $d\theta$ is the increment in vehicle's heading and ds is the distance travelled.The motion equations in world coordinates can also be expressed as:

$$\dot{x}_w = -V\sin\theta \quad (4)$$

$$\dot{y}_w = V\cos\theta \quad (5)$$

$$\dot{\theta} = V\gamma \quad (6)$$

where V is the longitudinal velocity, or vehicle speed, and γ is the angular velocity. These velocities can be considered as the control variables in (4)-(6). However, in some vehicles, it has been shown that the dynamic behavior of the mechanisms to apply these velocities from the control signal is very significant for the vehicle´s driving and then, should be considered for the vehicle´s control.

Let the dynamic of the steering actuation system be represented by means of the following first order model:

$$\frac{d\gamma}{dt} = -\frac{1}{T}(\gamma - \gamma_R) \quad (7)$$

where T is the time constant.

Thus, the steering control system can be represented by means of equations (4),(6) and (7) where x,θ and γ are the state variables and γ_R is the control variable computed by the steering control algorithm.

Authorized licensed use limited to: UNIVERSITAT POLITÈCNICA DE CATALUNYA. Downloaded on March 23, 2009 at 11:14 from IEEE Xplore. Restrictions apply.

In order to simplify the equations and decrease the number of parameters in the model, consider the following nondimensional variables:

$$t' = \frac{t}{T} \qquad x_w' = \frac{x_w}{VT} \qquad \theta' = \theta \qquad \gamma' = VT\gamma \quad (8)$$

and the nondimensional lookahead is:

$$L' = \frac{L}{VT} \quad (9)$$

In this paper, the nondimensional form of the equations is used, and the $'$ in variables and parameters is skipped for clarity. The equations are:

$$\begin{aligned} \dot{x} &= -\sin\theta \\ \dot{\theta} &= \gamma \\ \dot{\gamma} &= -\gamma + \gamma_R \end{aligned} \quad (10)$$

where γ_R is the non-dimensional form of the control law.

2.1 Pure pursuit path tracking

The pure pursuit strategy [4] is based on very simple geometric considerations. The path is tracked by repeatedly fitting circular arcs to different goal points on the path as the vehicle moves forward. The steering command is the requested new curvature ($\gamma_R=1/r$) which can be computed by:

$$\gamma_R = \frac{2x}{L^2} \quad (11)$$

where x is the x displacement of the goal point in vehicle coordinates (lateral displacement) and L is the lookahead distance. Thus, the pure pursuit method is a proportional controller of the steering angle using the lookahead for the gain and the local coordinate x as the error. It is necessary to choose a good goal point in the map according with the current navigation conditions. In fact, if the goal point is too far, the vehicle may cut corners and if too near, oscillations may result. As we will see in the next section there is a minimum value of L to maintain the stability of the system. Observe how for $L \to \infty$ in (11) the controller gain tends to zero and no steering corrections at all are introduced. However, for short lookahead the gains is high.

To conclude this section note that the conventional pure pursuit strategy does not consider the vehicle's dynamics; however from experimentation it can be shown that the steering dynamics (7) has a significant influence on the behavior of the vehicle. Thus, this dynamics will be considered in the following sections.

3 Closed Loop Basic Stability Analysis

Local stability of an equilibrium state of a nonlinear system may be examined by stability of the linearized system around the equilibrium state. This analysis apply only in the neighborhood of the equilibrium state. In the path tracking problem, the vehicle is trying to follow the path, so the vehicle's state will be in the vicinity of the equilibrium state, and therefore the above assumption is acceptable.

Consider the motion equations in non-dimensional form (10). The linearized system around the origin is:

$$\begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \dot{\gamma} \end{bmatrix} = J \begin{bmatrix} x \\ \theta \\ \gamma \end{bmatrix} \quad (12)$$

J is the Jacobian of the nonlinear system:

$$J = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ \varphi_x & \varphi_\theta & (\varphi_\gamma - 1) \end{bmatrix} \quad (13)$$

where $\varphi_x = \left.\frac{\partial \gamma_R}{\partial x}\right|_0$, $\varphi_\theta = \left.\frac{\partial \gamma_R}{\partial \theta}\right|_0$ and $\varphi_\gamma = \left.\frac{\partial \gamma_R}{\partial \gamma}\right|_0$ are the partial derivatives of γ_R at the origin.

Stability problems for linear systems can be reduced to the analysis of roots of the characteristic equations. A given system is said to be stable if all the roots of the characteristic equation have negative real part, and unstable if there exist at least one root with positive real part.

Let $P(s)$ be the characteristic polynomial of the linearized system:

$$P(s) = |sI - J| = s^3 + s^2 - s^2\varphi_\gamma - s\varphi_\theta + \varphi_x \quad (14)$$

The loss of stability will occur when one root crosses the imaginary axis. This condition can be obtained from the equation $P(j\omega) = 0$, where $j = \sqrt{-1}$. Applying this condition for the real and imaginary parts:

$$\begin{aligned} -\omega^2 + \omega^2\varphi_\gamma + \varphi_x &= 0 \\ -\omega^3 - \omega\varphi_\theta &= 0 \end{aligned} \quad (15)$$

and eliminating ω, the stability is given by the equation:

$$\varphi_\theta(1 - \varphi_\gamma) + \varphi_x \geq 0 \quad (16)$$

3.1 Pure Pursuit Basic Stability

Let us consider the tracking of a straight line as shown in Figure 1. In this case, the steering command γ_R is:

$$\gamma_R = \frac{2}{L^2}\left[x\cos\theta - \sqrt{L^2 - x^2}\sin\theta\right] \quad (17)$$

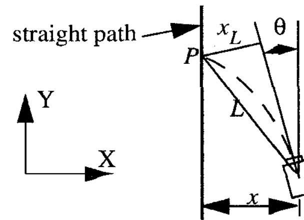


Figure 1 Pure pursuit straight path tracking

Authorized licensed use limited to: UNIVERSITAT POLITÈCNICA DE CATALUNYA. Downloaded on March 23, 2009 at 11:14 from IEEE Xplore. Restrictions apply.

and

$$\varphi_\theta = \frac{-2}{L} \qquad \varphi_x = \frac{2}{L^2} \qquad \varphi_\gamma = 0 \tag{18}$$

So, the stability condition is:

$$L \geq 1 \tag{19}$$

4 Tracking constant-curvature paths

The expressions that have been obtained in Section 3 refers to the tracking of a straight line. Consider now the tracking of a constant curvature path. Tracking these paths, is more convenient to express the equations in polar coordinates (see Figure 2).

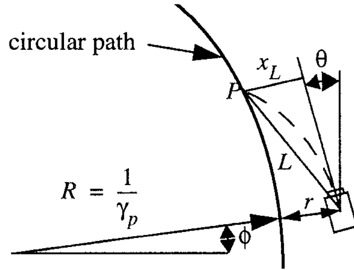


Figure 2 Pure pursuit circular path tracking

Variables are defined as follows: r is the radial distance from the path, θ is the angle between the vehicle heading and the normal to the $\vec{r}$ vector and the curvature γ is $\gamma = \gamma_1 - \gamma_p$, where γ_1 is the absolute curvature, and $\gamma_p = 1/R$ is the constant path curvature.

In this coordinate system, the nondimensional equations of motion are:

$$\begin{aligned} \dot{r} &= -\sin\theta \\ \dot{\theta} &= \gamma + \gamma_p - \frac{\gamma_p \cos\theta}{1 + \gamma_p r} \\ \dot{\gamma} &= -(\gamma + \gamma_p) + \gamma_R \end{aligned} \tag{20}$$

Tracking of straight paths is a particular case with $\gamma_p = 0$.

5 Stability analysis with time delays

Let consider now there is a pure delay (τ) in the feedback control loop, due to computing and communication delays. Now the steering command γ_R is delayed τ with respect to the other variables. To obtain the stability conditions, the system is locally linearized around the origin:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{\theta}(t) \\ \dot{\gamma}(t) \end{bmatrix} = J \begin{bmatrix} x(t) \\ \theta(t) \\ \gamma(t) \end{bmatrix} + J_\tau \begin{bmatrix} x(t-\tau) \\ \theta(t-\tau) \\ \gamma(t-\tau) \end{bmatrix} \tag{21}$$

The Jacobian matrices are:

$$J = \begin{bmatrix} 0 & -1 & 0 \\ \gamma_p^2 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \qquad J_\tau = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \varphi_r & \varphi_\theta & \varphi_\gamma \end{bmatrix} \tag{22}$$

where J_τ is the Jacobian with respect to the delayed variables.

Let $Q(s)$ *be* the characteristic quasi-polynomial of the linearized system. $Q(s)$ is defined as:

$$Q(s) = det[sI - J - J_\tau e^{-s\tau}] \tag{23}$$

It can be shown that the condition for the asymptotic stability of solutions of linear equations with delayed arguments is that the real parts of all roots of the characteristic quasi-polynomial be negative. So, the stability condition is $Q(j\omega) = 0$. For system (20), $Q(s)$ is:

$$\begin{aligned} Q(s) &= s^3 + s^2 + \gamma_p^2 s + \gamma_p^2 + \\ &\left[-s^2\varphi_\gamma - s\varphi_\theta + \left(\varphi_r - \gamma_p^2\varphi_\gamma\right)\right]e^{-s\tau} \end{aligned} \tag{24}$$

Provided that $e^{-j\omega\tau} = \cos(\tau\omega) - j\sin(\tau\omega)$, we obtain two conditions for the real and imaginary parts:

$$\begin{aligned} &-\omega^2 + \left(\omega\varphi_\gamma + \varphi_r - \gamma_p^2\varphi_\gamma\right)\cos(\tau\omega) - \\ &\qquad \omega\varphi_\theta \sin(\tau\omega) + \gamma_p^2 = 0 \\ &-\omega^3 + \gamma_p^2\omega - \omega\varphi_\theta\cos(\tau\omega) + \\ &\qquad \left(-\omega^2\varphi_\gamma - \varphi_r + \gamma_p^2\varphi_\gamma\right)\sin(\tau\omega) = 0 \end{aligned} \tag{25}$$

These equations define the stability conditions of (20).

5.1 Pure pursuit

Consider the tracking of the curvature constant path shown in Figure 2. For pure pursuit path tracking, γ_R is:

$$\gamma_R = \left[\frac{2}{L}\frac{\gamma_p(r^2 + L^2) + 2r}{2L(1 + \gamma_p r)}\cos\theta - \sqrt{1 - \left(\frac{\gamma_p(r^2 + L^2) + 2r}{2L(1 + \gamma_p r)}\right)^2}\sin\theta\right] \tag{26}$$

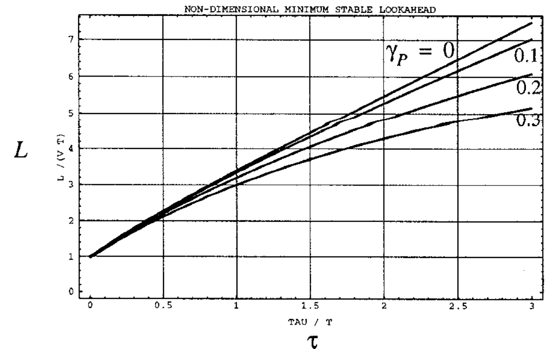


Figure 3 Circular path stable lookahead limits

Authorized licensed use limited to: UNIVERSITAT POLITÈCNICA DE CATALUNYA. Downloaded on March 23, 2009 at 11:14 from IEEE Xplore. Restrictions apply.

and

$$\varphi_r = \frac{2}{L^2}\left(1 - \frac{\gamma_p^2 L^2}{2}\right) \quad \varphi_\theta = -\frac{2}{L^2}\sqrt{1 - \frac{\gamma_p^2 L^2}{4}} \quad \varphi_\gamma = 0 \quad (27)$$

Substituting (27) in (25), the stability conditions are:

$$-\omega^2 + \frac{2}{L^2}\left(1 - \frac{\gamma_p^2 L^2}{2}\right)\cos(\tau\omega) + \frac{2}{L}\sqrt{1 - \frac{\gamma_p^2 L^2}{4}}\,\omega\sin(\tau\omega) + \gamma_p^2 = 0$$

$$-\omega^3 + \gamma_p^2\omega + \frac{2}{L}\sqrt{1 - \frac{\gamma_p^2 L^2}{4}}\,\omega\cos(\tau\omega) - \frac{2}{L^2}\left(1 - \frac{\gamma_p^2 L^2}{2}\right)\sin(\tau\omega) = 0 \quad (28)$$

The stable limit values of the nondimensional lookahead L are shown in Figure 3 for several path curvatures.

Figure 4 RedZone's HMMWV

6 Experimental Results

A series of test runs were performed in order to determine the validity of the analysis. The testbed used was a computer controlled HMMWV (High Mobility Multi-purpose Wheeled Vehicle) from RedZone Robotics, Inc.

The tests were carried out in a 500 m. long and 10 m. width road. This road is almost straight, with a curved section of small curvature. Therefore, the theoretical limits for $\gamma_p = 0$ have been used. The estimated time delay for the test vehicle is $\tau = 0.55s$. This includes computing, communication and actuator delays. The time constant of the steering system is estimated to be $T = 1.3s$.

The objective of the tests was to determine the value of the lookahead that was the limit between stable and unstable motion. In each test, the vehicle was commanded to follow the center path of the road, starting at the same point at the beginning of the path. When the motion was unstable (see Figure 5), small deviations from the path were amplified quickly, and the supervisor had to take control of the vehicle to avoid hitting obstacles on the sides of the road. When the motion was stable (see Figure 6), the vehicle tracked the road path with considerable oscillation around it. This is due to the fact that we used lookaheads only slightly higher than the ones that made the trajectories unstable.

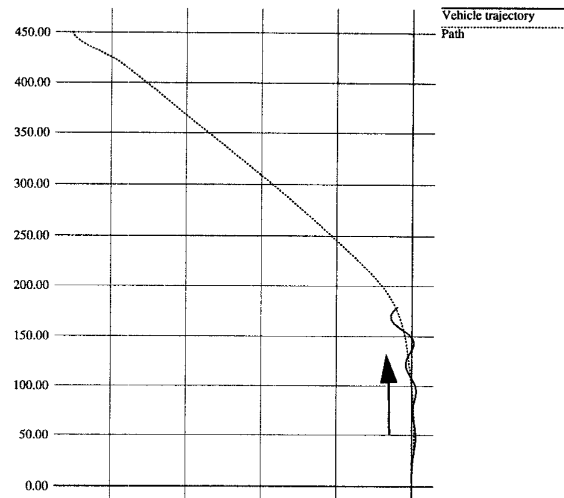


Figure 5 Unstable trajectory

Several runs were performed for three vehicle velocities (3, 6 and 9 m/s). For each velocity, the minimum lookahead that made the motion stable and the maximum lookahead that made it unstable were determined. The stability limit will be between these two values.

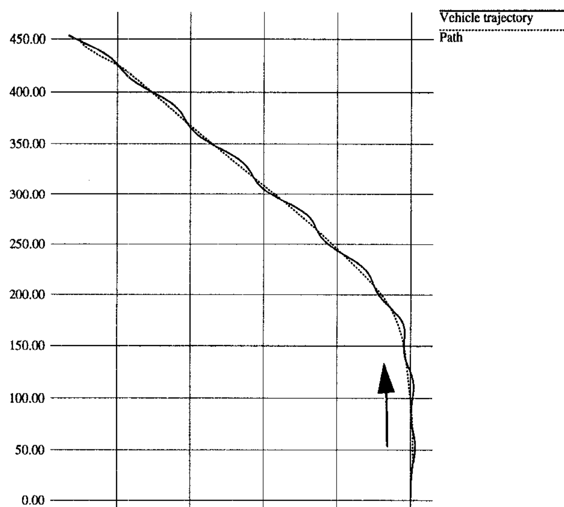


Figure 6 Stable trajectory

The results are presented in Figure 7. The two lines show the stability limits, considering the time delay and without doing it. The bars show the experimental stability limits: the bottom point is the maximum unstable lookahead and the top point is the minimum stable lookahead. The obtained values are close to the limit lookaheads predicted with the stability analysis considering the delay, and are

Authorized licensed use limited to: UNIVERSITAT POLITÈCNICA DE CATALUNYA. Downloaded on March 23, 2009 at 11:14 from IEEE Xplore. Restrictions apply.

much higher than the theoretical limits without considering the time delay. The discrepancies can be due to the nonlinear phenomena not modelled in the equations (like tire slippage) and the estimation of the parameters.

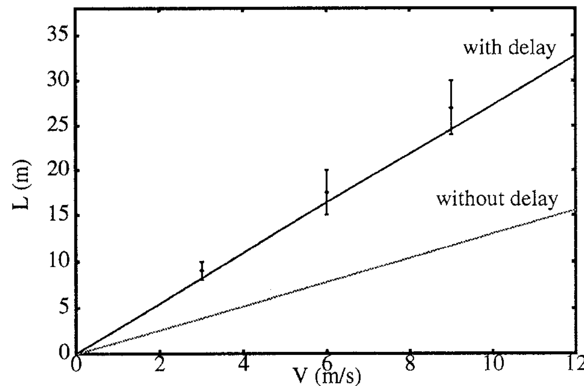


Figure 7 Experimental and theoretical stability limits

7 Conclusions

Path tracking implementations tipically involves a pure delay in the control loop that can affect significantly its stability.

In this paper, we have presented a new approach that analyzes the stability of a general class of path tracking algorithms taking into account the time delay. The analysis has been done for straight paths and paths of constant curvature. This has sufficient generality since most usual paths can be decomposed in pieces of constant curvature.

The method has been applied to the pure pursuit path tracking algorithm, one of the most widely used. The experiments performed with a computer controlled HMMWV show good agreement with the theoretical predictions of the proposed method.

This technique can be used in developing path tracking algorithms for real time control and supervision, and in adaptive lookahead strategies. Future work include a more accurate determination of parameters, more testing with different time delays, the analysis of another path tracking algorithms and the inclusion of more terms of the dynamics in the equations.

8 Acknowledgments

We gratefully acknowledge assistance from Omead Amidi at the VASC (Robotics Institute, CMU), and Ben Motazed, Jeff Callen and Demetri Patukas from RedZone Robotics, Inc. (Pittsburgh, PA, USA)

9 References

[1] E. DickNmans and A. Zapp. Autonomous high-speed road vehicle guidance by computer vision. In Proc. 10 IFAC World Congress, Munich, 1987.

[2] V. F. Muñoz , J. L. Martínez and A. Ollero. New Continuous-Curvature Local Path Generators for Mobile Robots. *Intelligent Components and Instruments for Control Applications.* A. Ollero and E.F.Camacho editors. *Proceedings of the SICICA'92 IFAC Symposium. Pergamon Press.*

[3] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. I. J. Robotic Research, Vol. 5, No 1, pp 90-98, 1986.

[4] O. Amidi. Integrated mobile robot control, Carnegie Mellon Univ. Robotics Institute. Technical Report CMU-RI-TR-90-17, 1990.

[5] W. Nelson and I. Cox. Local path control for an autonomous vehicle. Proc. IEEE Conference on Robotics and Automation, pp 1315-1317, 1988.

[6] Y. Kanayama, A. Nilipour, and C. Lelm. A locomotion control method for autonomous vehicles. Proc. IEEE Conference on Robotics and Automation, pp 1315-1317, 1988.

[7] D. Shin. High performance tracking of explicit paths by road-worthy mobile robots. Ph. D. Dissertation. Carnegie Mellon University, 1990.

[8] R. Fenton, G. Melocik, K. Olson. On the steering of automated vehicles: theory and experiment. IEEE Trans. AC, pp 306-315, 1976.

[9] W. H. Cormier and R. Fenton. On the steering of automated vehicles- a velocity-adaptive controller. IEEE Trans. AC, pp 375-385, 1980.

[10]A. Ollero, and O. Amidi (1991). Predictive Path Tracking of Mobile Robots. Applications to the CMU Navlab. *Proc. of the Fifth International Conference on Advanced Robotics*, Pisa, Vol. II, pp. 1081-1086.

[11]A.Ollero, A.G.Cerezo and J.Martinez. Fuzzy supervisory path tracking of autonomous vehicles. Control Engineering Practice. Vol. 2, No. 2, pp. 313-319, 1994.

[12]J.Martinez, A.Ollero and A.G.Cerezo. Fuzzy strategies for path tracking of autonomous vehicles.Proceedings of the EUFIT Congress. Vol. I. Aachen, 1993.

[13]R. Wallace et al. First Results in Robot Road-Following. Report within CMU-RI-TR 86-4.

[14]Y. Kanayama, Y. Kimura, T. Noguchi and F. Miyazaki. A stable control method for an autonomous mobile robot. Proc. IEEE Conf. on Robotics and Automation, pp 384-389, 1990.

[15]N. Sarkar, X. Yun and V. Kumar. Control of mechanical systems with rolling constraints: application to dynamic control of mobile robots.I. J. Robotic Research, Vol. 13, No 1, pp 55-69, 1994

[16]E. W. Kamen. On the relationship between zero criteria for two-variable polynomials and asymptotic stability of delay differential equations. IEEE Trans. Automat. Contr., vol. AC-25, pp. 983-984, 1980.

[17]G. Gu and E. B. Lee. Stability testing of time delay systems. Automatica, vol. 25, No. 5, pp. 777-780, 1989.

[18]T. Mori. Criteria for asymptotic stability of linear time-delay systems. IEEE Trans. Automat. Contr., vol. AC-30, pp. 158-161, 1985.

[19]T. Mori and H. Kokame. Stability of $\dot{x}(t) = Ax(t) + Bx(t-\tau)$. IEEE Trans. Automat. Contr., vol. AC-34, pp. 460-462, 1989.

Authorized licensed use limited to: UNIVERSITAT POLITÈCNICA DE CATALUNYA. Downloaded on March 23, 2009 at 11:14 from IEEE Xplore. Restrictions apply.

G: CODI DE LA LLIBRERIA PATHFINDING.

Pathfinding.c

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>


PUNTOS origin,target,poss;

PUNTOS camino[15]; // Y simbolizara el valor nulo !=0.

PUNTOS ruta[9]; // el maximo de puntos por los que va a pasar son 9


/****************************** GREEDY **********************************/

void  Greedy (PUNTOS origin, PUNTOS target, PUNTOS * ruta)

{

PUNTOS poss;

    /* MATRIZ DE WEIGHTPOINTS. ES DIAGONAL DE zeros  */


           /*                    A B C  D  E   F   G  H  I                                  */

int WP[9][9] = {            {0, 2, 0, 3, 0, 0,  0, 0, 0},  // A

                            {9, 0, 5, 0, 4, 0,  0, 0, 0},  // B

                            {0, 9, 0, 0, 0, 1,  0, 0, 0},  // C

                            {9, 0, 0, 0, 2, 0,  1, 0, 0},  // D

                            {0, 8, 0, 9, 0, 3,  0, 4, 0},  // E

                            {0, 0, 9, 0, 9, 0,  0, 0, 1},  // F

                            {0, 0, 0, 9, 0, 0,  0, 1, 0},  // G

                            {0, 0, 0, 0, 9, 0,  1, 0, 2},  // H

                            {0, 0, 0, 0, 0, 8,  0, 7, 0}} ; // I

/**********************************************************/

int totalweight=0;

ruta[0]=origin; //registramos el primer punto

poss=origin;

int j=1;
```

```
while (poss!= target){  // mientras la posicion en la que estemos sea diferente al objetivo..
int i,f=0, k=1000;
        for (i=0;i<9;i++){
                if ((WP[poss][i]<k) && (WP[poss][i]>0)) {   // aceptamos el punto si el WP > 0.
                k = WP[poss][i];  // lo actualizamos y nos quedamos con la posición
                 f=i;    //  posicion del weightpoing que nos dara el siguiente punto
                 } //if
        }  // for
        ruta[j]=f;
        totalweight=totalweight+k;
         j=j+1;
        poss=f;
        } //while
  ruta[j]=Z; // indicador de final de vector
   }

/**********************************************************************/
```

```
/************************** DIJKSTRA ************************************/

void Dijkstra (PUNTOS origin, PUNTOS target, PUNTOS * ruta)

{

    /* MATRIZ DE WEIGHTPOINTS. ES DIAGONAL DE zeros  */

        /*              A B C  D  E   F   G  H  I                                */

int WP[9][9] = {        {0, 2, 0, 3, 0, 0,  0, 0, 0},  // A

                        {9, 0, 5, 0, 4, 0,  0, 0, 0},  // B

                        {0, 9, 0, 0, 0, 1,  0, 0, 0},  // C

                        {9, 0, 0, 0, 2, 0,  1, 0, 0},  // D

                        {0, 8, 0, 9, 0, 3,  0, 4, 0},  // E

                        {0, 0, 9, 0, 9, 0,  0, 0, 1},  // F

                        {0, 0, 0, 9, 0, 0,  0, 1, 0},  // G

                        {0, 0, 0, 0, 9, 0,  1, 0, 2},  // H

                        {0, 0, 0, 0, 0, 8,  0, 7, 0}} ; // I

 /********************************************************/

int n=9;  //n=size(WP,1);

int S[n];      // s, vector, set of visited vectors

int dist[n];   // it stores the shortest distance between the source node and any other node;

int prev[n];  // Previous node, informs about the best previous node known to reach each
network node

int h;

int  candidate[n];

for (h=0;h<n;h++){

   S[h] = 0;

   dist[h]= 1000;

   prev[h] = n+1;

      if (S[i]==0) candidate[i]=dist[i];

      else candidate[i]=1000;

}
```

```
dist[origin] = 0;
int i,nodvisit=0;
do{
            int u,ind=1000;
            for (h=0;h<n;h++){
                        if (candidate[h] < ind) {
                                    u = h;
                                    ind = candidate[h];
                                    }
                        }
            S[u]=1;
            nodvisit++;
            for (h=0;h<n;h++){
                        if(((dist[u]+WP[u][h])<dist[h])&&(WP[u][h]>0)){
                        dist[h]=dist[u]+WP[u][h];
                        prev[h]=u;
                        }
            }
}while (nodvisit!=n);
//  DIJKSTRA RELLENA LA RUTA AL REVES. DESDE EL TARGET HASTA EL ORIGINT
ruta[n]=Z;  //rang 0-9.
ruta[n-1]=target; //10 posicions
int g=1;
do{
   if (prev[ruta[n-g]]<=n)
      ruta[n-g-1]=prev[ruta[n-g]];
                        g++;
}while (ruta[n-g]!= origin);
int totalweight = dist[target];
}
/************************************************************************/
```

```
/************************ PUNTOS A CAMINO ****************************/

 void  puntosAcamino (PUNTOS * ruta, char * camino) // recive ruta y devuelve camino.

{

int i=0;

int o=0;

int v=0;

  do {
             printf( "%i ", ruta[v]);
             v++;
 } while(ruta[v]!=Z);

do{

            printf("%d\n",i);


            switch (ruta[i]){

       case A:

       if (ruta[i+1]== B) {
          camino[o]='R';   //R=turn right, L= turn left, S=straight
          o++;
          camino[o]='S';
          o++;
                      if (ruta[i+2]==E){
           camino[o]='L';
          o++;
                      }

            }

       else if (ruta [i+1]==D){
            camino[o]='S';
            o++;
            if (ruta [i+2]==E){
            camino[o]='R';
            o++;

            }}

        break;
```

```
  case B:

        if ((ruta [i+1]==C )||(ruta [i+1]==A )){
             camino[o]='S';
             o++;
             if ((ruta [i+2]==F)||(ruta [i+2]==D)){
                          camino[o]='L';
                          o++;
                          }
           }
        else if (ruta [i+1]==E){
              camino[o]='S';
              o=o+1;
              if (ruta [i+2]==F){
                          camino[o]='R';
                                      o++;
                                      }
              else if (ruta [i+2]==D){
                          camino[o]='L';
                                      o++;
              }}
break;

       case C:

        if ((ruta [i+1]==F)||(ruta [i+1]==B)){
           camino[o]='S';
           o++;
           if (ruta [i+2]==E ){
              camino[o]='L';
              o++;
           }}
break;

       case D:

        if (ruta [i+1]==G){
           camino[o]='S';
           o++;
           if (ruta [i+2]==H){
              camino[o]='R';
              o++;


           }}
        else if ((ruta [i+1]==E)||(ruta [i+1]==A)){
              camino[o]='S';
              o++;
              if ((ruta [i+2]==H)||(ruta [i+2]==B)){
                          camino[o]='L';
                          o++;
                          }
```

```
            else if (ruta [i+2]==B){
                    camino[o]='R';
                    o++;
            }}

break;

        case E:

        if (ruta [i+1]==H){
          camino[o]='S';
          o++;
          if (ruta [i+2]==I){
            camino[o]='R';
            o++;

            }
          else if (ruta [i+2]==G){
            camino[o]='L';
            o++;
                    }
           }
        else if (ruta [i+1]==F){
           camino[o]='S';
           o++;
           if (ruta [i+2]==C){
            camino[o]='R';
            o++;
            }
           else if (ruta [i+2]==I){
            camino[o]='L';
            o++;
         }}
              if (ruta [i+1]==B){
          camino[o]='S';
          o++;
          if (ruta [i+2]==A){
            camino[o]='R';
            o++;

            }
          else if (ruta [i+2]==C){
            camino[o]='L';
            o++;
                    }
           }
       break;
```

```
case F:

        if (ruta [i+1]==I){
          camino[o]='S';
          o++;
          if (ruta [i+2]==H){
             camino[o]='L';
             o++;
            }
          else if (ruta [i+2]==Z){
//fin de ruta hacemos giro de 180º
             camino[o]='R';
             o++;
             camino[o]='R';
             o++;

            }
          }
        else if (ruta [i+1]==E){
             camino[o]='S';
             o++;
             if (ruta [i+2]==H){
                      camino[o]='R';
                      o++;
                      }
             else if (ruta [i+2]==B){
                      camino[o]='L';
                      o++;
             }}
break;


case G:

        if (ruta [i+1]==H ){
          camino[o]='S';
          o++;
          if (ruta [i+2]==E){
             camino[o]='R';
             o++;
            }

          }

        else if (ruta [i+1]==D){
             camino[o]='S';
             o++;

             if (ruta [i+2]==E){
            camino[o]='L'; //seria R pero viene en sentido contrario. pues es la izquierda del robot
```

```
                    o++;

            }}

  break;

 case H:

       if (ruta [i+1]==I){
          camino[o]='S';
          o++;
            if ((ruta [i+2]==Z)||(ruta [i+2]==F)) { //fin de ruta o vamos a F: hacemos giro de 90º.
relamente es redundante. siempre acabaremos con esta configuracion.

            camino[o]='R';
            o++;
           }
         }
       else if (ruta [i+1]==E){
            camino[o]='S';
            o++;
            if (ruta [i+2]==F){
                    camino[o]='R';
                    o++;
                    }
            else if (ruta [i+2]==D){
                    camino[o]='L';
                    o++;
            }}

                     else if (ruta [i+1]==G){
         camino[o]='S';

         if (ruta [i+2]==D){
            camino[o]='L';
            o++;
           }
         }
       else if (ruta [i+1]==E){
            camino[o]='S';
            o=o+1;
            if (ruta [i+2]==F){
                    camino[o]='L';
                              o++;
                              }
            else if (ruta [i+2]==D){
                    camino[o]='R';
                              o++;
            }}
                              o++;
     break;
```

```
 case I:
                        if (ruta[i+1]== H) {
          camino[o]='R';   //R=turn right, L= turn left, S=straight
          o++;
          camino[o]='S';
          o++;
                        if (ruta[i+2]==E){
                         camino[o]='L';
                        o++;
                        }
            }
       else if (ruta [i+1]==F){
              camino[o]='S';
              o++;
              if (ruta [i+2]==E){
              camino[o]='R';
              o++;
                }}
            o++;
      break;

  } //case
  i++;

 } while (ruta[i]!=Z);  // do-while

 camino[o]='Z';
}

/*****************************************************************************
*************************/
```

H: CODI DEL PROGRMA PRINCIPAL.

Super11pp.c

```
 #include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <time.h>
#include <mqueue.h>
#include <math.h>
#include <string.h>
#include "sitr.h"
#include "api.h"

#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define ROTATE_SP 45.0 // Consigna de rotacio del robot a ROTATING (º/s) per a que fagi 90º
en 2seg.
#define FORWARD_DIST 1050
#define TURN_DIST 90 // OFFSET DE GIR 90º. depenent del sentit sera positiu o negatiu
#define BATTERY_OK 18.0 // Nivell minim admissible de carrega de la bateria


#define pi 3,14159

/* *****      camera       */
#define PORT 80
#define SERVIDOR "192.168.1.114"
#include "apicamera.h"
#include "apiimatge.h"

#define T_CON 600                   //30.0
#define T_GUI_SHOW 100.0
#define T_MONITOR 25.0
#define T_API 20.0

#define PRIO_SUP 9      // 9
#define PRIO_EVE 8  // 8
#define PRIO_MON 7  // 7
#define PRIO_CON 6  // 6
#define PRIO_GUI 5  // 5
#define PRIO_SHO 4   //  4

#define CAPACITAT_CUA 30    //modificat de 10 a 30

struct timespec t0;
```

```
//CUES
mqd_t cua_api, cua_control;

//Estats de la tasca Supervisor
typedef enum {IDLE,MOVEMENT,PAUSED,SHUTDOWN} MODESUP_T;
volatile MODESUP_T modeSUP = IDLE;              // IDLE S'UTILITZA NOMES PER INICIALITZAR

//Estats de la tasca Controller
typedef enum {FORWARD,TURN_RIGHT,TURN_LEFT,OFF} MODECON_T;
volatile MODECON_T modeCON = OFF;

//Estats de la tasca Monitor
typedef enum {MONITOR_OFF,MONITOR_ON} MODEMON_T; //
volatile MODEMON_T modeMON = MONITOR_OFF;

//MUTEX
pthread_mutex_t m;

// Tipus d'esdeveniments
typedef enum {START,STOP,QUIT,COLLISION,END_MOV} EVENT_CONTROL; //END_MOV
QUAN HAGI RECORREGUT LA DIST O QUAN HAGI GIRAT ELS +-90º

//variables globals
volatile int path=0; // para sacar parte de la info de pantalla solo despues de calcular ruta
pathfinding
volatile int rest=0;
volatile float manualSP[NUM_COMP]={0.0, 0.0, 0.0};
volatile float robotSP[NUM_COMP];
volatile float ODO_INICIAL[NUM_COMP];

PUNTOS ruta[10];
char camino[10];

// Mostra informacio del Robotino per pantalla

void ShowAdvanced(PUNTOS ruta[], float battery, float robotVelocity[], float Odometry[],
MODESUP_T modeSUP0,MODECON_T modeCON0,  float ODO_INICIAL[NUM_COMP] /* float
robotSP0[]*/);
```

```
void * Supervisor(void * none)
{
	EVENT_CONTROL eventControl;
	MODEMON_T modeMON0;
	MODECON_T modeCON0;
	MODECON_T modeACT=OFF;
//variable que ens ajudara a discernir de quin estat veniem al fer la pausa per tornar-hi.
	MODESUP_T modeSUP0;
	float ODO_INICIAL0[NUM_COMP]={0.0, 0.0, 0.0};
	int j=0;

	do {

	mq_receive(cua_control, (char *)&eventControl, sizeof(EVENT_CONTROL), NULL);

		pthread_mutex_lock(&m);
		modeSUP0=modeSUP;
		pthread_mutex_lock(&m);

		switch (modeSUP0) {
			case IDLE:
				if (eventControl==START){
					path=1;
					rest=1;
					modeSUP0=MOVEMENT;
					modeMON0=MONITOR_ON;

	getOdometry(&ODO_INICIAL0[0],&ODO_INICIAL0[1],&ODO_INICIAL0[2]);

					if (camino[j]=='S'){
						modeCON0=FORWARD;
					}
					else if (camino[j]=='R'){
						modeCON0=TURN_RIGHT;
					}
					else if (camino[j]=='L'){
						modeCON0=TURN_LEFT;
					}
				} else if (eventControl==QUIT) {			// SI SE
DETECTA COLISION, SE METE POR PANTALLA UN 3 o no se calcula bien los puntos de imagen
					pthread_mutex_lock(&m);
					modeCON=OFF;
					modeSUP=SHUTDOWN;
					modeMON=MONITOR_OFF;
					pthread_mutex_unlock(&m);
					pthread_exit(NULL);
				}
			break;
```

```
				case MOVEMENT:
					if (eventControl==STOP){
						modeSUP0=PAUSED;
						modeACT=modeCON0;
						modeCON0=OFF;
						modeMON0=MONITOR_OFF;
					} if (eventControl==END_MOV){

	getOdometry(&ODO_INICIAL0[0],&ODO_INICIAL0[1],&ODO_INICIAL0[2]);

						j=j+1; // atualització del punter

						if (camino[j]=='S'){
							modeCON0=FORWARD;
//							modeMON0=MONITOR_ON;
						}
						else if (camino[j]=='R'){
							modeCON0=TURN_RIGHT;

						}
						else if (camino[j]=='L'){
							modeCON0=TURN_LEFT;

						}
						else if (camino[j]=='Z'){
							pthread_mutex_lock(&m);
								modeCON=OFF;
								modeSUP=SHUTDOWN;
								modeMON=MONITOR_OFF;
							pthread_mutex_unlock(&m);


							//MISSATGE PER PANTALLA INDICANT FI DE LA
SEQUENCIA:
							printf("\033[24;20f SECUENCIA ACABADA.
\033[25;15f  ROBOT EN EL DESTINO INDICADO. ");
							printf("\033[23;15f
****************************** \033[26;15f ******************************* ");


							pthread_exit(NULL);


						}
					} else if (eventControl==COLLISION) { // SE DETECTA COLISION
						pthread_mutex_lock(&m);
						modeCON=OFF;
						modeSUP=SHUTDOWN;
```

```
					modeMON=MONITOR_OFF;
					pthread_mutex_unlock(&m);
					pthread_exit(NULL);
			}
			break;

			case PAUSED:
				if (eventControl==START){
					modeSUP0=MOVEMENT;
					modeMON0=MONITOR_ON;
					modeCON0=modeACT; //modeACT =
FORWARD/TURN_RIGHT/TURN_LEFT
				} else if (eventControl==QUIT) { // SE METE ‘3’ POR PANTALLA
					pthread_mutex_lock(&m);
					modeCON=OFF;
					modeSUP=SHUTDOWN;
					modeMON=MONITOR_OFF;
					pthread_mutex_unlock(&m);
					pthread_exit(NULL);
				}
			break;
		}

		pthread_mutex_lock(&m);
			ODO_INICIAL[0]=ODO_INICIAL0[0];
			ODO_INICIAL[1]=ODO_INICIAL0[1];
			ODO_INICIAL[2]=ODO_INICIAL0[2];
			modeMON=modeMON0;
			modeCON=modeCON0;
			modeSUP=modeSUP0;
		pthread_mutex_unlock(&m);


	} while (1);
}
```

```
void * Controller (void * none)
{
        struct timespec ts, interval,interval2;
        MODECON_T modeCON0;
        float robotSP0[NUM_COMP];
        EVENT_API eventApi;
        EVENT_CONTROL eventC;


                        //IMATGE
  int i,x1,x2,y1,y2,x1r,y1r,x2r,y2r;
        imgmem_t img,img2,img3,img4,img5;
        double phi;

        int B,b2;
        int L=50; // condicio estabilitat L>=1.
        double A,C,dist,dist2,a2,inv_r,w;
            /************************************/

        nsec2timespec(&interval,T_CON*1000000LL);
        timespecPlusTimeInterval(&ts, &t0, &interval);

do {
                        pthread_mutex_lock(&m);
                modeCON0=modeCON;
                        pthread_mutex_unlock(&m);


switch (modeCON0) {

                case FORWARD: // s means stright,forward

                        getBMP(SERVIDOR ,&img,LOW); // obtenim "buffer" que indica on
comencen els bites i "tamany"

                        capaCR(&img,&img2);
                        im2bw(&img2,&img3,46);
                        erode(&img3,&img4,7.5);
                        dilate(&img4,&img5,3);
                        erode(&img5,&img5,2);
                        detectPuntos(&img5,35,55, &x1, &y1, &x2, &y2); // en coord. imatge

                        if (detectPuntos=0){    // SUSPENSIÓ DEL PROGRAMA SI NO ES
DETECTEN PUNTS A LA IMATGE
                                eventC=QUIT;
                                mq_send(cua_control, (char *)&eventC,
sizeof(EVENT_CONTROL), NULL);
                        }
```

```
PimgToRobot(&x1,&y1, &x1r,&y1r);
PimgToRobot(&x2,&y2, &x2r,&y2r);

        // calcul de l'angle de la recta respecte el robot
int Ax,Ay;
Ay= y2r - y1r;
Ax=x2r - x1r;
phi= atan((double)Ax/(double) Ay)*180/pi;

        //      paremtros equacion de la recta Ax + By + C
A=(double)(x1r-x2r);
B=(double)(y2r-y1r);
C=(double)(y1r*x2r - y2r*x1r);

        // eq. distancia de punto a una recta. punto es origen Ax=By=0.
a2=pow(A,2);
b2=pow(B,2);
dist=C/sqrt(a2+b2);

        //FORMULA OLLERO
dist2=pow(dist,2);
inv_r=(dist*cos(phi)-sin(phi)*sqrt(L*L-dist2))*2/(L*L);
        // velocitat angular=vel.lineal/r=v*inv_r   volem velocitat lineal
de 20mm/s llavors:

w=20*inv_r;

                //Composicio de les consignes de velocitat del robot
                robotSP0[0]=20;
                robotSP0[1]= 0;
                robotSP0[2]= w*180/pi;



break;

case TURN_RIGHT:
        robotSP0[0]=0.0;
        robotSP0[1]=0.0;
        robotSP0[2]=- ROTATE_SP;      //-45 rad/seg cuadrant dret es
negatiu
break;

case TURN_LEFT:
        robotSP0[0]=0.0;
        robotSP0[1]=0.0;
        robotSP0[2]=ROTATE_SP;

break;
```

```
				case OFF:
					robotSP0[0]=0.0;
					robotSP0[1]=0.0;
					robotSP0[2]=0.0;
				break;
			}
					pthread_mutex_lock(&m);
					robotSP[0]=robotSP0[0];
					robotSP[1]=robotSP0[1];
					robotSP[2]=robotSP0[2];
					pthread_mutex_unlock(&m);

				setRobotVelocity(robotSP0[0],robotSP0[1],robotSP0[2]);
			clock_nanosleep(CLOCK_REALTIME,TIMER_ABSTIME,&ts,NULL);
			timespecPlusTimeInterval(&ts,&ts,&interval);
		} while (1);
}


void * API_Events(void * none)
{
		EVENT_API eventApi;
		EVENT_CONTROL eventC;

		do {

			mq_receive(cua_api, (char *)&eventApi, sizeof(EVENT_API), NULL);

			// Generacio de l'esdeveniment COLLISION
			if(eventApi==BUMPER_ON){
				eventC=COLLISION;
				mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL),
NULL);
			}

		} while (1);
}
```

```
void * Monitor(void * none)
{
	struct timespec ts, interval;
	EVENT_CONTROL eventC;
	float Odometry[NUM_COMP];
	MODEMON_T modeMON0;
	float ODO_INICIAL0[NUM_COMP];
	int g=0; // cuando el robot empieza, la x del robot es la y del mundo. y cuando gira a la
derecha o a la izq coincide con la x del mundo

	nsec2timespec(&interval,T_MONITOR*1000000LL);
	timespecPlusTimeInterval(&ts, &t0, &interval);

	do {
		pthread_mutex_lock(&m);
		modeMON0=modeMON;
		pthread_mutex_unlock(&m);

		switch(modeMON0) {
			case MONITOR_OFF:

			break;

			case MONITOR_ON: //COMPROBA TOTES 3 PERO COM SON
EXCLUSIVES, NOMES ES PODRA VALIDAR UNA PER COP

				pthread_mutex_lock(&m);
				ODO_INICIAL0[0]=ODO_INICIAL[0];
				ODO_INICIAL0[1]=ODO_INICIAL[1];
				ODO_INICIAL0[2]=ODO_INICIAL[2];
				pthread_mutex_unlock(&m);

				getOdometry(&Odometry[0],&Odometry[1],&Odometry[2]);

				// ODOMETRIA EN FORWARD
				if ((Odometry[2]<45) && (Odometry[2]>-45)){

					if(Odometry[0]>(ODO_INICIAL0[0]+FORWARD_DIST)){
					//odometria > odometria inicial +1000mm
					//printf("+Y mundo\n");
					eventC=END_MOV;
		mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
						}

				}
```

```
if ((Odometry[2]>135) && (Odometry[2]<-135)){

        if (Odometry[0]<(ODO_INICIAL0[0]-FORWARD_DIST)){
                //odometria < odometria inicial -1000mm

                                        // VA EN SENTIT NEGATIU      DE Y EN C.MÓN
                //printf("-Y mundo\n");
                eventC=END_MOV;
                mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
                                        }
                                }

if ((Odometry[2]<-45) && (Odometry[2]>-135)){

        if (Odometry[1]<(ODO_INICIAL0[1]-FORWARD_DIST)){
                //odometria > odometria inicial +1000mm
                //printf("X mundo\n");
                eventC=END_MOV;
                mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
                                        }
                                }

if ((Odometry[2]>45) && (Odometry[2]<135)){
        if (Odometry[1]>(ODO_INICIAL0[1]+FORWARD_DIST)){
                //odometria > odometria inicial +1000mm
                //printf("-X mundo\n");
                eventC=END_MOV;
                mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
                }
        }
                        // ODOMETRIA EN GIR

if (Odometry[2]<(ODO_INICIAL0[2]- TURN_DIST)){
        setRobotVelocity(0.0,0.0,0.0); // o poner controlador a off
        if (g==1) g=0;
        else if (g==0) g=1;
        eventC=END_MOV;
        mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
        }
if (Odometry[2]>(ODO_INICIAL0[2]+ TURN_DIST)){
        setRobotVelocity(0.0,0.0,0.0); // o poner controlador a off
        if (g==1) g=0;
        else if (g==0) g=1;
        eventC=END_MOV;
        mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
                                        }
                        break;
        }
```

```
		pthread_mutex_lock(&m);
		modeMON =modeMON0;
		pthread_mutex_unlock(&m);

		clock_nanosleep(CLOCK_REALTIME,TIMER_ABSTIME,&ts,NULL);
		timespecPlusTimeInterval(&ts,&ts,&interval);

	} while (1);

}



void * ShowGUI (void *none)
{
	struct timespec ts, interval;
	float Odometry[NUM_COMP], battery;
	float robotVelocity[NUM_COMP];
	MODESUP_T modeSUP0;
	MODECON_T modeCON0;
	float robotSP0[NUM_COMP];
	float ODO_INICIAL1[NUM_COMP];

	nsec2timespec(&interval,T_GUI_SHOW*1000000LL);
	timespecPlusTimeInterval(&ts, &t0, &interval);

	do {

		pthread_mutex_lock(&m);
		modeSUP0=modeSUP;
		modeCON0=modeCON;
		robotSP0[0]=robotSP[0];
		robotSP0[1]=robotSP[1];
		robotSP0[2]=robotSP[2];

		ODO_INICIAL1[0]=ODO_INICIAL[0];
		ODO_INICIAL1[1]=ODO_INICIAL[1];
		ODO_INICIAL1[2]=ODO_INICIAL[2];

		pthread_mutex_unlock(&m);

		//POSICIÓ ROBOT (odometria)
		getOdometry(&Odometry[0],&Odometry[1],&Odometry[2]);

		//VELOCITAT ROBOT
		getRobotVelocity(&robotVelocity[0],&robotVelocity[1],&robotVelocity[2]);
```

```
		//NIVELL BATERIA
		getBatteryVoltage(&battery);

	//	ShowAdvanced(ruta, battery, robotVelocity, Odometry, modeSUP0,robotSP0);
		ShowAdvanced(ruta, battery, robotVelocity, Odometry,
modeSUP0,modeCON0, ODO_INICIAL1);
		clock_nanosleep(CLOCK_REALTIME,TIMER_ABSTIME,&ts,NULL);
		timespecPlusTimeInterval(&ts,&ts,&interval);
	} while (1);
}

void * GUI_Input (void *none)
{
	char text[3];
	EVENT_CONTROL eventC;

	do {

		fgets(text, 3, stdin);
		if (text[1]!=10) continue;

		switch (text[0]) {
			case '1':
		eventC=START;
		mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
			break;
			case '2':
		eventC=STOP;
		mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
			break;
			case '3':
		eventC=QUIT;
		mq_send(cua_control, (char *)&eventC, sizeof(EVENT_CONTROL), NULL);
			break;
		}

	} while (1);
}

int main(void)
{
	pthread_t tidSUP;
	pthread_attr_t attr;
	struct mq_attr qattrC;
	struct sched_param param;
	struct var_esdev init;
	int aux;
```

```
	init.tipusEsdeveniments = ESDE_BUMPER;
	aux = initSystem ("/net/dionysus/home/alumne/errorsRobotino",T_API, &init);
//canviar segons ordinador
	if (aux==ERROR)
		return;

	cua_api = init.cua;

	// Cua: cua_control
	qattrC.mq_maxmsg=CAPACITAT_CUA;
	qattrC.mq_msgsize=sizeof(EVENT_CONTROL);
	cua_control=mq_open("/event_cua_control", O_RDWR|O_CREAT, S_IRUSR|S_IWUSR,
&qattrC);

	//pathfinding
		PUNTOS origin,target;
		int modo;
		printf("%c\n********************   PATHFINDING  ROBOTINO QNX
********************",0x0C);
		printf("\n\n 1: Indica node ORIGEN i node TARGET.\n");
		printf(" 2. Indica el tipus d'algorisme pathfinding (1: GREEDY. 2:
DIJKSTRA.).\n");

	scanf("%i",&origin);
	scanf("%i",&target);
	scanf("%i",&modo); // indicara el tipo de pathfinding


	if (modo==1) Greedy(origin,target,ruta);				// nos devolvera el
vector ruta y creamos el camino
	else if (modo==2) Dijkstra(origin,target,ruta);

	puntosAcamino(ruta,camino);

	pthread_mutex_init(&m,NULL);
	clock_gettime(CLOCK_REALTIME, &t0);

	pthread_attr_init(&attr);
	pthread_attr_setinheritsched(&attr,PTHREAD_EXPLICIT_SCHED);
	pthread_attr_setschedpolicy(&attr,SCHED_RR);

	param.sched_priority = PRIO_SHO;
	pthread_attr_setschedparam(&attr,&param);
	pthread_create (NULL, &attr, ShowGUI, NULL);
	param.sched_priority = PRIO_GUI;
	pthread_attr_setschedparam(&attr,&param);
	pthread_create (NULL, &attr, GUI_Input, NULL);
```

```
        param.sched_priority = PRIO_MON;
        pthread_attr_setschedparam(&attr,&param);
        pthread_create (NULL, &attr, Monitor, NULL);
        param.sched_priority = PRIO_EVE;
        pthread_attr_setschedparam(&attr,&param);
        pthread_create (NULL, &attr, API_Events, NULL);
        param.sched_priority = PRIO_SUP;
        pthread_attr_setschedparam(&attr,&param);
        pthread_create (&tidSUP, &attr, Supervisor, NULL);
        param.sched_priority = PRIO_CON;
        pthread_attr_setschedparam(&attr,&param);
        pthread_create (NULL, &attr, Controller, NULL);

        pthread_join(tidSUP,NULL);

        aux=closeSystem();
        if (aux==ERROR)
                return;

        mq_close(cua_control);
        mq_unlink("/event_cua_control");
}


// Mostra informacio del Robotino per pantalla practiques 4 i 5 PSCTR
void ShowAdvanced(PUNTOS ruta[], float battery, float robotVelocity[], float Odometry[],
MODESUP_T modeSUP0,MODECON_T modeCON0, float ODO_INICIAL1[] /* robotSP0[]*/ )
{
        char text[100];
   char buffer_text[1000];
   static char start=0;

                if (start==0){
                        system("cls");
                        printf("%c\n********************   PATHFINDING  ROBOTINO QNX
********************",0x0C);
                        printf("\n 3. Combina: \t 1: START \t 2: STOP \t 3: EXIT\n");

printf("**********************************************************************
\n",0x0C);


                   start=1;
                }


   if (path==1){   // només apareixerá quan s'hagi escollit la ruta de pathfinding
```

```
                printf("\n MODE SUPERVISOR:  \033[7;45f Battery: \n MODE CONTROLLER:
",0x0C);
                    printf("\n\n\t CIRCUIT \n\n     G ----- H ----- I \n");
                    printf("     |       |       | \n");
                    printf("     D ----- E ----- F \n");
                    printf("     |       |       | \n");
                    printf("     A ----- B ----- C \n\n");
                    printf(" Recorregut a Realitzar:\n\n\t");


/**************************** printf ruta *****************/
            int b=0;
            int a=sizeof(camino);
            char RUTA[a];

      do{

 switch (ruta[b]){
 case A:
      RUTA[b]= 'A';
 break;
 case B:
      RUTA[b]= 'B';
 break;
 case C:
      RUTA[b]= 'C';
 break;
 case D:
      RUTA[b]= 'D';
 break;
 case E:
      RUTA[b]= 'E';
 break;
 case F:
      RUTA[b]= 'F';
 break;
 case G:
      RUTA[b]= 'G';
 break;
 case H:
      RUTA[b]= 'H';
 break;
 case I:
      RUTA[b]= 'I';
 break;
 }
 b++;
```

```
    }
while(b<a+1);


int t=0;
do{
            printf("%c ", RUTA[t]);
t++;
              }
while(t<a+1);

/*****************************************************************/


                              printf("\033[9;35f Odometry X: \033[10;35f Odometry Y: \033[11;35f
Odometry phi:");
                              printf("\033[13;35f Consigna Vel X: \033[14;35f Consigna Vel Y:
\033[15;35f Consigna Vel phi: \n\n");
                              printf("\033[17;35f Vel. Real Robot X: \033[18;35f Vel. Real Robot Y:
\033[19;35f Vel. Real Robot phi: \n\n");

          printf("\033[25;0f*******************************************************
***************\n");
path=0;
 }

 if(rest==1){

                    //MODE SUPERVISOR
                    switch ( modeSUP0 ) {
                              case IDLE:
                                        sprintf(text,"\033[7;19f IDLE    ");
                                        strcpy(buffer_text, text);
                              break;
                              case MOVEMENT:
                                        sprintf(text,"\033[7;19f MOVING  ");
                                        strcpy(buffer_text, text);
                              break;
                              case PAUSED:
                                        sprintf(text,"\033[7;19f PAUSED  ");
                                        strcpy(buffer_text, text);
                              break;

                              case SHUTDOWN:
                                        sprintf(text,"\033[7;19f SHUTDOWN");
                                        strcpy(buffer_text, text);
                              break;
                    }
```

```
	//MODE CONTROLADOR
			switch ( modeCON0 ) {
				case FORWARD:
					sprintf(text,"\033[8;19f FORWARD    ");
					strcat(buffer_text, text);
				break;
				case TURN_RIGHT:
					sprintf(text,"\033[8;19f TURN_RIGHT ");
					strcat(buffer_text, text);
				break;
				case TURN_LEFT:
					sprintf(text,"\033[8;19f TURN_LEFT ");
					strcat(buffer_text, text);
				break;

				case OFF:
					sprintf(text,"\033[8;19f OFF       ");
					strcat(buffer_text, text);
				break;

			}

	//NIVELL BATERIA

		if (battery<BATTERY_OK) {
		sprintf(text,"\033[7;55f% .2lfV (LOW) ",battery);
		} else {
		sprintf(text,"\033[7;55f% .2lfV (OK)  ",battery);
		}

		strcat(buffer_text, text);



	//POSICIÓ ROBOT (odometria)

		sprintf(text,"\033[9;54f% .2lf\033[10;54f% .2lf\033[11;54f%
.2lf",Odometry[0],Odometry[1],Odometry[2]);
		strcat(buffer_text, text);

// odo inicial

	sprintf(text,"\033[13;54f% .2lf\033[14;54f% .2lf\033[15;54f%
.2lf",ODO_INICIAL1[0],ODO_INICIAL1[1],ODO_INICIAL1[2]);
		strcat(buffer_text, text);
/*
	//CONSIGNA VELOCITAT ROBOT
```

```
			sprintf(text,"\033[15;54f% .2lf\033[16;54f% .2lf\033[17;54f%
.2lf",nearOdometry1[0],nearOdometry1[1],nearOdometry1[2]);
			strcat(buffer_text, text);
*/
		//VELOCITAT ROBOT

		sprintf(text,"\033[17;56f% .2lf\033[18;56f% .2lf\033[19;56f%
.2lf",robotVelocity[0],robotVelocity[1],robotVelocity[2]);
			strcat(buffer_text, text);

			printf ("%s\n\n\n\n", buffer_text);
			printf("\033[20;1f%c",0x20);

}}
```

I: FUNCIÓ DE PATHFINDING MODIFICADA PER MATLAB

dijkstraMATLABmod.c

Basicament és igual que l'anterior pero incorporant tot el programa en un bucle for que l'executa dues vegades. Les dues rutes es podrien guardar en una mateixa variable en forma de matriu.

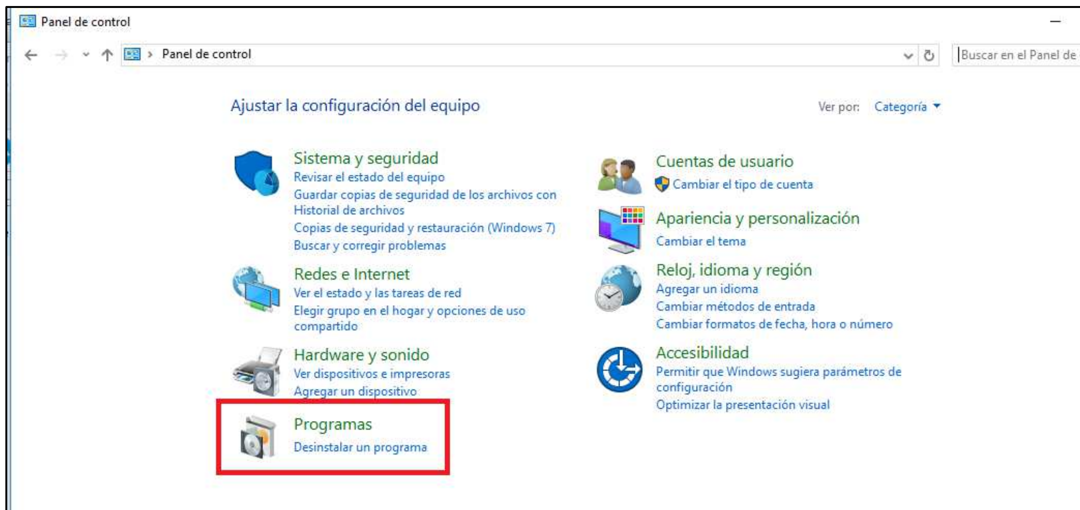
```
s=1; %index de A es 1
d=9; %index de I es 9


%     /* MATRIZ DE WEIGHTPOINTS. ES DIAGONAL DE zeros   */

%                A    B    C    D    E    F    G    H    I     */

matriz_costo= [0     3    0    4    0    0    0    0    0;  % /A
               9     0    3    0    4    0    0    0    0;  % /B
               0     9    0    0    0    4    0    0    0;  % /C
               9     0    0    0    1    0    2    0    0;  % /D
               0     9    0    9    0    2    0    3    0;  % /E
               0     0    9    0    9    0    0    0    2;  % /F
               0     0    0    9    0    0    0    1    0;  % /G
               0     0    0    0    9    0    9    0    2;  % /H
               0     0    0    0    0    9    0    9    0;]; % /I

% /************************************************************/



% inici bucle que torna a calcuar la ruta manipulant el pes de
la ruta mes
% rapida en busca d'un empat. abans de fer el 2n cicle ha d'em-
magatzemar la
% 1a ruta i el pes total.
buc=1; %indicador per saber si estem a la 1r o 2na iteracio
pes1=0;
pes2=0;
for i=1:2

n=size(matriz_costo,1);
S(1:n) = 0;      %s, vector, set of visited vectors
dist(1:n) = inf;    % it stores the shortest distance between the
source node and any other node;
prev(1:n) = n+1;     % Previous node, informs about the best pre-
vious node known to reach each  network node

dist(s) = 0;


while sum(S)~=n
    candidate=[];
    for i=1:n
        if S(i)==0
            candidate=[candidate dist(i)];
        else
            candidate=[candidate inf];
        end
```

```
    end
    [u_index u]=min(candidate);
    S(u)=1;
    for i=1:n
        if((dist(u)+matriz_costo(u,i))<dist(i))&&(ma-
triz_costo(u,i)>0)
            dist(i)=dist(u)+matriz_costo(u,i);
            prev(i)=u;
        end
    end
end


sp = [d];

while sp(1) ~= s
    if prev(sp(1))<=n
        sp=[prev(sp(1)) sp];
    else
        error;
    end
end;
j=length(sp);
spcost = dist(d);

if (buc==1) pes1=spcost;
for p=1:j   %//empezamos en 1 porque la primera posicion ya la
tenemos en forma de letra. recordamos que 'j' tiene la informa-
cion de cuantas posiciones tiene el vector 'camino'

    if sp(p)==1 ruta(p)='A'; %// recordamos que  la 1a posicion
ya esta ocupada con el punto de partida
    elseif sp(p)==2 ruta(p)='B';
    elseif sp(p)==3 ruta(p)='C';
    elseif sp(p)==4 ruta(p)='D';
    elseif sp(p)==5 ruta(p)='E';
    elseif sp(p)==6 ruta(p)='F';
    elseif sp(p)==7 ruta(p)='G';
    elseif sp(p)==8 ruta(p)='H';
    elseif sp(p)==9 ruta(p)='I';
    end
end
ruta(j+1)='Z'; %// indicador de final de vector

elseif (buc==2) pes2=spcost;

    for p=1:j   %//empezamos en 1 porque la primera posicion ya
la tenemos en forma de letra. recordamos que 'j' tiene la infor-
macion de cuantas posiciones tiene el vector 'camino'

    if sp(p)==1 ruta2(p)='A'; %// recordamos que  la 1a posicion
ya esta ocupada con el punto de partida
    elseif sp(p)==2 ruta2(p)='B';
    elseif sp(p)==3 ruta2(p)='C';
    elseif sp(p)==4 ruta2(p)='D';
    elseif sp(p)==5 ruta2(p)='E';
```

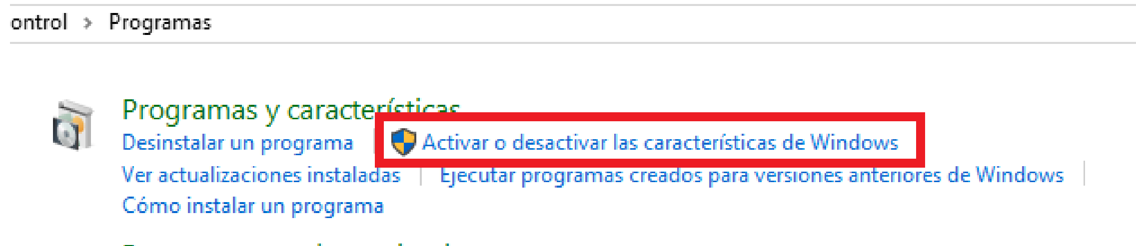
```
    elseif sp(p)==6 ruta2(p)='F';
    elseif sp(p)==7 ruta2(p)='G';
    elseif sp(p)==8 ruta2(p)='H';
    elseif sp(p)==9 ruta2(p)='I';
    end
end
ruta2(j+1)='Z'; %// indicador de final de vector


end

matriz_costo(sp(2),sp(3))=10; % aumentem WP d'un dels camins de
la 1a solucio.
buc=buc+1;
    % el problema sera si resulta que les rutes empatades com-
parteixen
    % aquest cami.
end % end for en busca de empat

%if pes1==pes2 %empate detectado. enviar las dos soluciones
% print empate and print ruta2
% end
pes1
pes2
ruta
ruta2
```

J: ACTIVACIÓ DEL TELNET

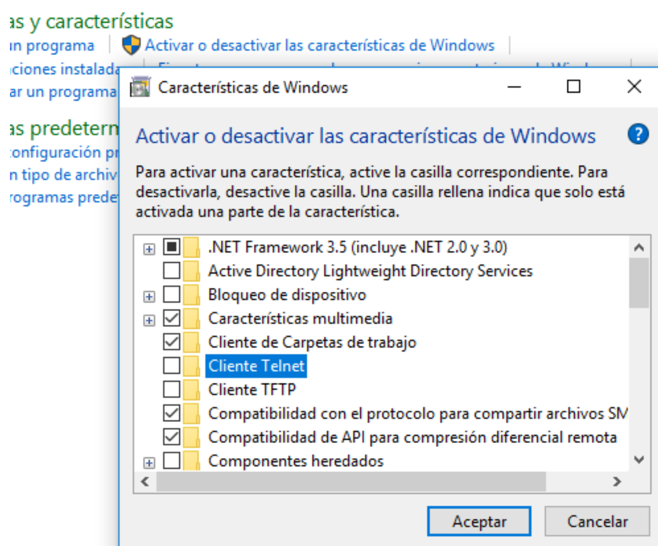
La connexió Telnet es necessària per poder executar remotament les aplicacions en el Robotino. Per activar-lo, primer hem d'anar a **Panell de Control** i seleccionem **PROGRAMAS.**



Després cliquem sobre **Activar o desactivar las característiques de Windows**.

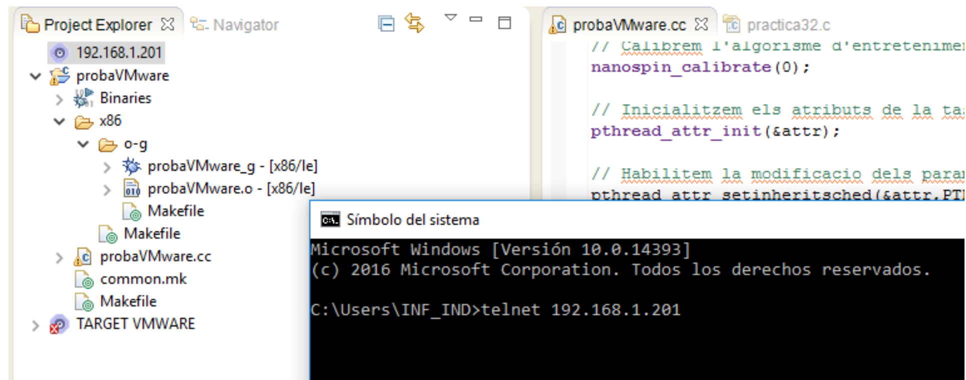


Se'ns obrirà la següent finestra i hem de seleccionar la casella de **Client Telnet** i acceptem**.**

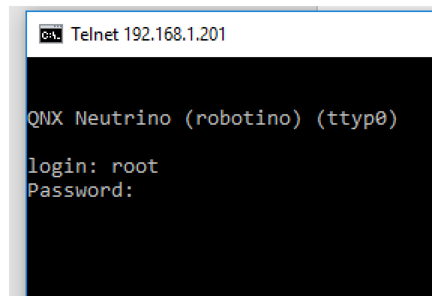


- CONNECTAR AMB ROBOTINO VIA TELNET

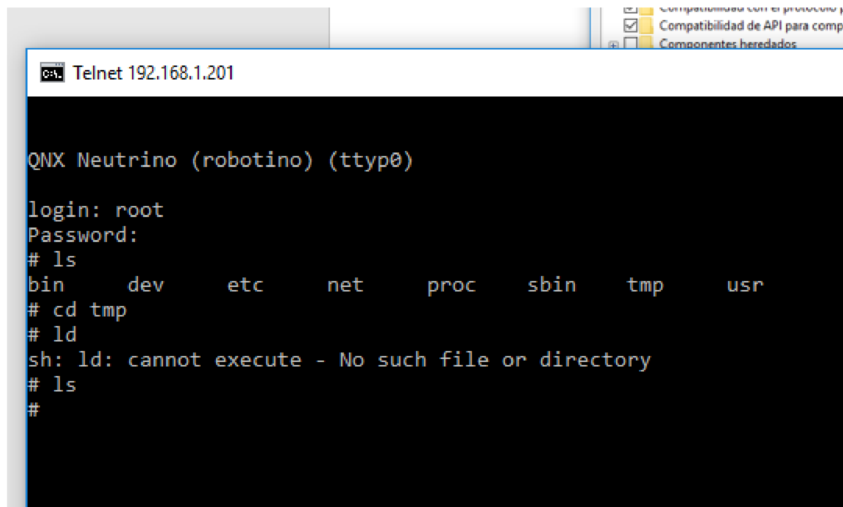
Primer obrim el terminal CMD i hi escrivim *telnet NºIP del target* (en el nostre cas: Telnet 192.168.1.118)



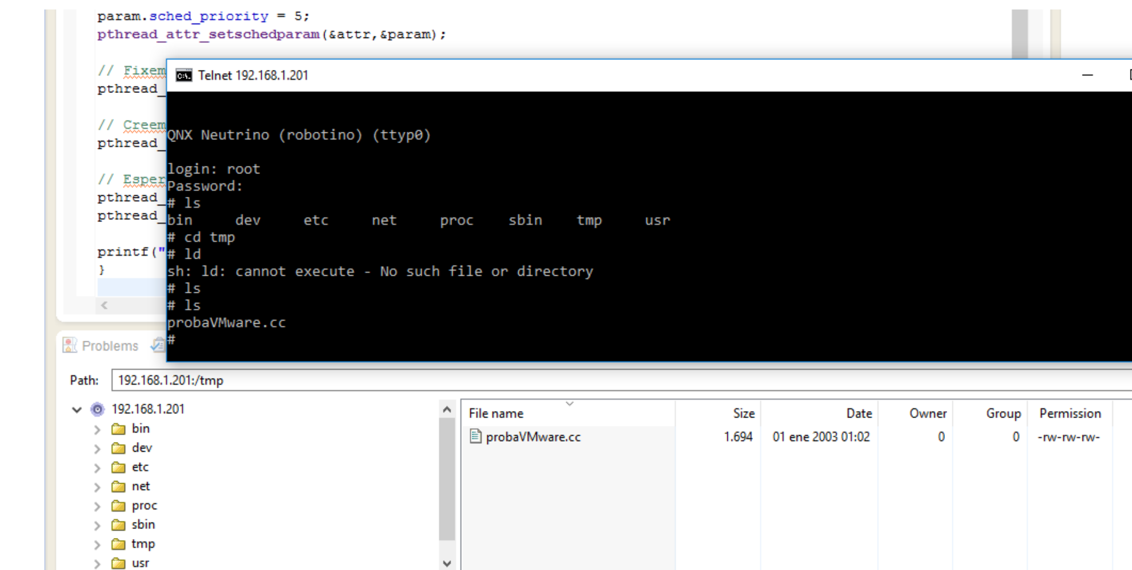
Ens apareixerà el següent missatge. Haurem d'introduir l'usuari (**root**) i la contrasenya (**target**) i ens permetrà establir la connexió i accedir al Robotino.



Una vegada dins, podem navegar per les carpetes que conté la memòria del *target*. Primer de tot he mirat la carpeta */tmp* i com es pot veure estava buida.



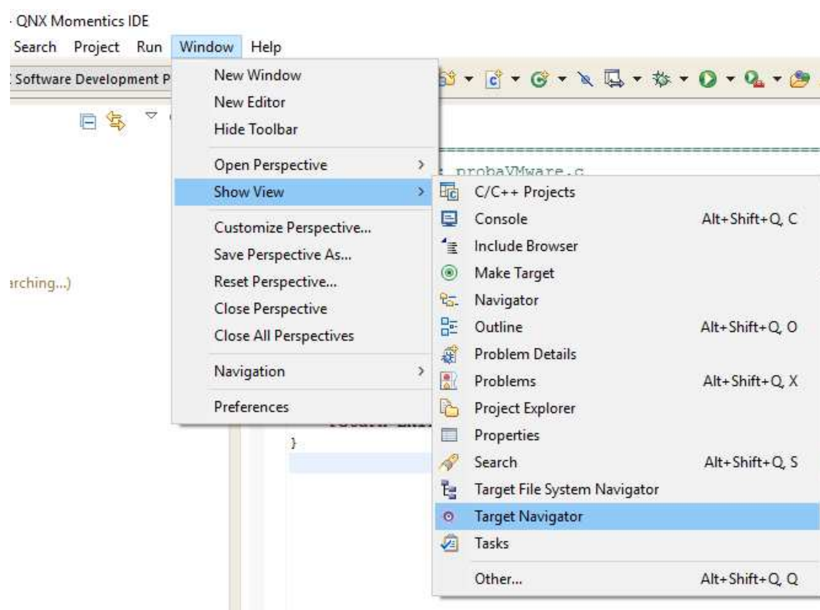
Després de transferir-li el programa amb l'IDE (esta explicat com fer-ho a l'anterior document) he tornat a comprovar la carpeta */tmp* i hi apareix l'arxiu. Ara només faltarà executar-lo com fèiem a l'entorn de QNX.

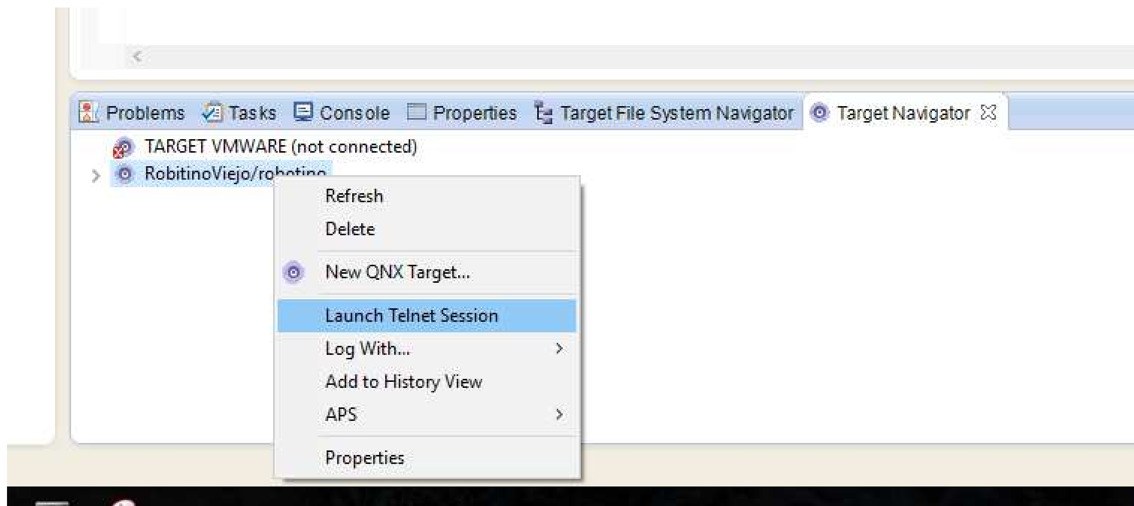


- TELNET DESDE L'IDE

Una altre forma de connectar-se per Telnet amb el Robotino es des de el propi IDE. Aquesta funció es molt útil ja que així no fa falta obrir el terminal CMD i haver d'estar controlant dues (o varies) finestres.

Per fer-ho hem d'anar a la finestra inferior "Target Navigator" Fer clic dret sobre el *target* i seleccionar *Launch Telnet Session:*





Se'ns obrirà el terminal en una altra finestra, iniciem la sessió amb l'usuari *root* i ja podem accedir al Robotino.