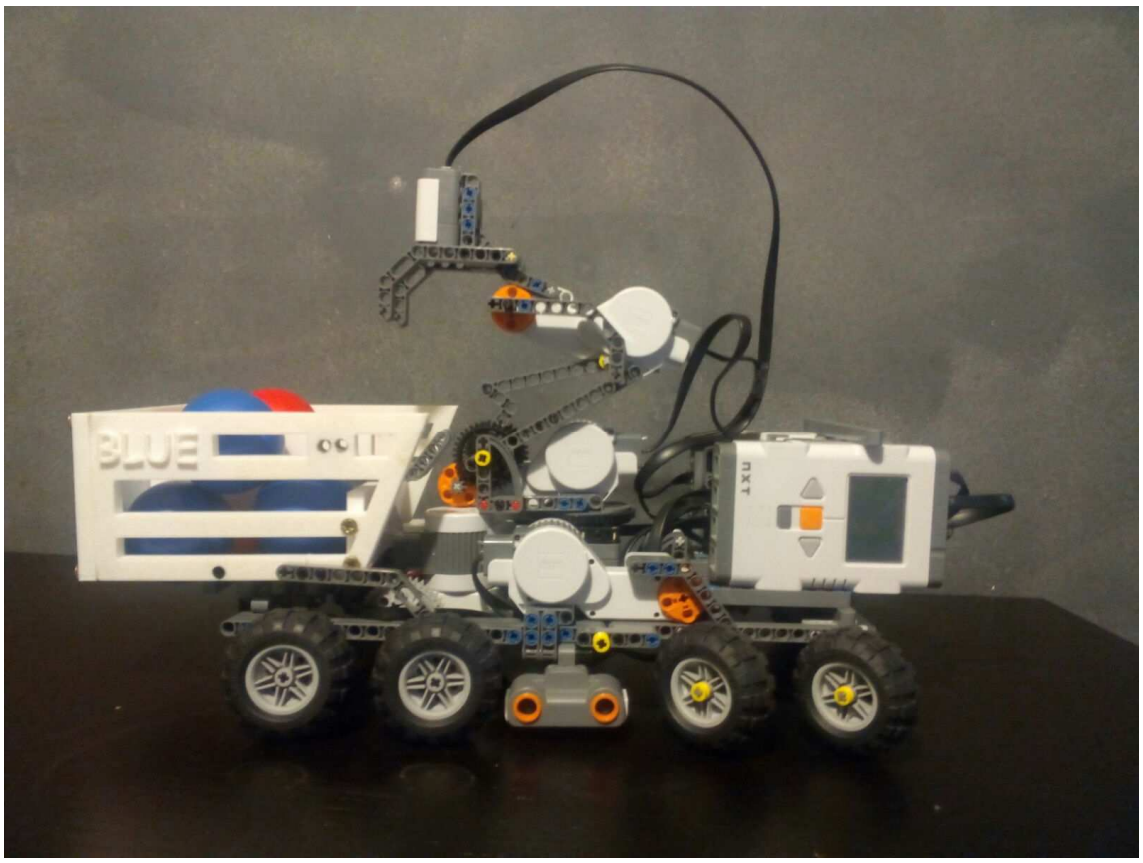
CREACIÓ D'UN ROBOT AMB CONTROL REMOT I FUNCIONS AUTÒMATES



xRedShark15

Dirigit per: M**** D***-G*******

INS Montmeló

2n Batxillerat A

12.01.2016

ÍNDEX

Pàg.

“L'aprendre millor no vindrà d'oferir les millors eines perquè el professor instrueixi, sinó de donar les millors oportunitats als estudiants per construir.”

Seymour Papert.

1. Introducció

Com bé indica la portada, el treball consisteix en l'elaboració d'un robot mitjançant el producte Lego Mindstorms amb la seva segona generació NXT 2.0, aprofundint per tant en la robòtica i la programació.

He decidit realitzar el meu treball sobre aquest àmbit de la tecnologia perquè en un futur estaria interessat a graduar-me en enginyeria mecànica o en enginyeria robòtica. Així, el treball m'ajudarà a conèixer millor el món de la programació i la robòtica de tal manera que, amb els coneixements que obtindré realitzant aquesta recerca, disposaré de la informació necessària per a poder decidir de manera correcta sobre cap a on orientar els meus estudis.

A més, aquest tema conté gran relació amb el temari de tecnologia industrial. Potser no podem establir tanta relació amb assignatures com física o dibuix tècnic pròpies de la vessant tecnològica de batxillerat que estic estudiant, però si manté certa relació amb matemàtiques, doncs als programes s'utilitza molt aquesta disciplina. Com he comentat abans, a part de les relacions que manté amb les matèries que estudio actualment, també està enfocat a començar a conèixer el món al qual em vull dedicar professionalment.

Les raons que m'han impulsat a decidir-me per aquest tema han tingut molta influència en mi durant tota la vida. Ja des de ben petit estava molt interessat en el món del motor (tant automobilisme i motociclisme com construcció de màquines i aparells útils que facilitin certes tasques). Per tant, ja des de ben petit estava molt interessat en el món de la enginyeria en general: sempre estava intentant arreglar objectes espatllats o trencats, fabricant tot tipus d'andròmines... A part, sempre m'han atret molt les joguines Lego: quan em regalaven alguna, jo la muntava segons les instruccions i després la desfeia i feia les meves pròpies creacions i invencions. D'aquesta manera, quan la tutora em va comentar de realitzar el treball amb un robot programable Lego no vaig dubtar ni un instant.

També cal mencionar el concurs de robòtica que vam realitzar a la facultat de matemàtiques de la Universitat de Barcelona, amb el qual em vaig introduir al funcionament del programari amb el que funcionava el robot i les funcions que podia fer aquest. En particular, també m'agradaria mencionar a

l’E*** P******, professor de la universitat que ens va ajudar a l’hora de realitzar el programa per al robot del concurs i em va explicar quin era el seu funcionament. La qualificació final que vam obtenir (quarts de dotze participants) va ser molt satisfactòria i em va motivar a seguir esforçant-me per aconseguir l’objectiu d’aquest treball i treballar dur esforçant-me al màxim.

2. Objectius

A l'hora de plantejar el treball i quines serien les dificultats amb les que em trobaria vaig haver de marcar-me una sèrie d'objectius bastant àmplia per a poder acomplir la tasca que se'm presentava. La construcció d'aquest robot no està limitada a una eina, ja sigui un programa o unes instruccions de muntatge, sinó que és producte d'un conjunt d'elements de diferent caire (com el disseny de l'estructura, una tasca més manual; el disseny de les peces 3D, un procés més gràfic o la programació del brick, més teòric). Així doncs, podem diferenciar els objectius en teòrics i pràctics.

Com a objectius teòrics:

- Conèixer les bases i els fonaments de la robòtica i els sistemes de control.
- Aprendre en certa mesura la metodologia del llenguatge de programació Java per a elaborar el programa.
- Conèixer les característiques dels Lego Mindstorms i el ventall de possibilitats que aquest ofereixen.

Com a objectius pràctics:

- Aprendre a realitzar un programa en llenguatge leJOS.
- Aprendre a programar una senzilla aplicació per a mòbil.
- Conèixer millor el programa Sketchup i agafar destresa amb programes de disseny 3D.
- Ser capaç d'elaborar una grua des de zero superant els problemes que se'm presentin i minimitzant l'error que aquesta pugui tenir, elaborant una estructura el més eficient i funcional possible.

MARC TEÒRIC

3. La robòtica i els sistemes automàtics i de control

3.1. La robòtica

3.1.1. Què és la robòtica?

Abans de començar a desenvolupar quins són els inicis d'aquesta branca de la enginyeria i veure la seva evolució al llarg de la història és precís que definim de manera clara quin és el concepte de robòtica, doncs és un terme molt estès a la societat d'avui en dia però la majoria de persones no som capaços de definir-lo de manera precisa i detallada.

La robòtica és la branca de l'enginyeria mecànica, l'energia elèctrica i l'energia electrònica que s'encarrega del disseny, la construcció, la disposició estructural, la manufactura i l'aplicació dels robots. Així doncs, com podem observar combina diferents disciplines com la mecànica, la informàtica, l'electrònica, la intel·ligència artificial, l'enginyeria de control i la física. Aquest terme és utilitzat per Isaac Asimov per a referir-se a aquesta ciència.

Per a poder completar aquesta definició hem de conèixer quin és el significat del terme robot. La paraula robot és per primera vegada utilitzada en una obra de teatre anomenada "R. U. R."(Els Robots Universals de Rossum) escrita pel dramaturg xec Karel Campek que data del 1917. L'argument de l'obra era senzill: l'home fabrica un robot i després aquest el mata. Partint d'aquesta obra moltes altres obres han seguit aquesta línia i han tingut molt èxit, com per exemple la pel·lícula "Jo robot" dirigida per Alex Proyas i interpretada per Will Smith, entre d'altres.

La paraula "Robota" provinent del txec significa servitud o treballador forçat. D'aquesta manera, quan el terme va ser traduït a l'anglès va adoptar la forma que fins avui en dia manté, robot.

3.1.2. Història

Molts segles enrere l'home ja tenia l'interès de construir màquines que imitessin les parts del cos humà. Ja en l'antiguitat els antics egipcis van unir braços mecànics a les estàtues dels seus déus. Aquest moviment produït per

aquest mecanismes eren aclamats com a inspiració dels déus. A la costa nord del Mediterrani, els grecs van construir estàtues que funcionaven mitjançant sistemes hidràulics. D'aquesta manera s'aconseguia fascinar als adoradors dels temples.

És a partir del segle XVIII amb l'inici de la revolució industrial quan la creació de màquines destinades a imitar els moviments del cos humà adopta un enfocament més pràctic deixant de banda l'oci i la creació artística. En un moment on la producció en cadena està en ple auge es precisen de màquines que substitueixin a les persones. D'aquesta manera es volia aconseguir major eficiència a un menor cost. Les màquines principalment havien de substituir a l'home en les tasques més feixugues, repetitives, difícils o que requerissin un desgast físic molt gran. En aquest context històric el món del que ara coneixem com a robòtica experimenta un creixement exponencial.

Als inicis de la robòtica, aquestes màquines que substituïen l'home no eren pròpiament conegudes com a robots (aquest terme sorgeix després com abans es menciona), sinó que eren denominades simplement màquines o autòmats.

A continuació s'adjunta una taula amb els invents i els successos més rellevants en ordre cronològic fins l'any 2000:

Data	Descripció	Nom	Autor
Segle I a.C. i anteriors	Descripcions de més de 100 màquines i autòmates com per exeple un artefacte amb foc, un òrgan de vent, una màquina operada mitjançant una moneda, una màquina de vapor... al llibre de Pneumàtica i autòmat de Herón de Alexandria.	Autòmat	Ctesibio de Alejandría, Filón de Bizancio, Herón de Alexandria, i altres.
1495	Disseny d'un robot humanoide	Cavaller mecànic	Leonardo da Vinci
1738	Ànec mecànic capaç de menjar, moure les ales i excretar.	Digesting Duck	Jacques de VauCanson
1800	Joguines mecàniques japoneses.	Joguines Karakuri	Hisashige Tanaka
1801	Telar programable per al ordit(1)[1].	-	J. Jaquard

[1] Veure glossari.

1805	Nina mecànica capaç de fer dibuixos.	-	H. Maillardet
1939-1940	S'exhibeix un robot humanoide a l'Exposició Universal.	Elektro	Westinghouse Electric Corporation
1942	La revista Astounding Science Fiction publica "cerce viciós", una història on es donen a conèixer les tres lleis de la robòtica.	SPD-13 (anomenat "Speedy")	Isaac Asimov
1948	Exhibició d'un robot amb comportament biològic simple.	Elsie i Elmer	William Grey Walter
1952	Una màquina prototip de control numèric va ser l'objectiu del MIT[1]. Es va desenvolupar al 1961 un llenguatge de programació de peces denominat ATP (Automatically Programmed Tooling).	-	MIT
1954	Desenvolupament de dissenys per a transferència d'articles programada. Patent emesa a estats units al 1961	-	George Devol
1956	Primer robot comercial basat en la patent de Devol.	Unimate	George Devol
1959	Robot controlat per interruptors de fi de cursa.	-	Planet Corporation
1961	S'instal·la el primer robot industrial a la Ford Motors Company per atendre a una màquina de fundació.	Unimate	George Devol
1966	Construcció i instal·lació d'un robot de pintura per pulverització.	-	Trallfa
1968	Robot dotat de sensors i càmera de visió que podia desplaçar-se pel terra.	Shakey	SRI[2]
1971	Elaboració d'un petit braç de robot de accionament elèctric	Standford Arm	Standford University
1973	Primer robot amb sis eixos electromecànics.	Famulus	KUKA Robot Group
1974	Kawasaki, sota llicència de Unimation, va instal·lar un robot per a soldadura per arc per a estructures de motocicletes.	-	Unimation
1974	Cincinnati Milacron va introduir el robot T3 amb control per ordinador.	T3	Cincinnati Milacron
1975	El robot 'Sigma' 'd'Olivetti s'utilitzà en operacions de muntatge.	Sigma	Olivetti
1975	Braç manipulador programable universal.	-	Unimation
1976	Un dispositiu de Remote Center Compliance (RCC) per a la inserció de peces en la línia de muntatge es va desenvolupar als laboratoris	-	Laboratoris Stark Draper Labs

[2] Standford Research Institute.

	Charles Stark Draper Labs a Estats Units.		
1978	Es va introduir el robot PUMA[3] per tasques de muntatge per Unimation, basant-se en dissenys obtinguts en un estudi de la General Motors.	PUMA	Unimation
1979	Desenvolupament del robot tipus SCARA[4] a la Universitat de Yamanashi al Japó per muntatge. Diversos robots SCAR comercials es van introduir cap al 1981.	SCAR	Universitat de Yamanashi
1980	Un sistema robòtic de captació de recipients va ser objecte de demostració a la Universitat de Rhode Island. Amb l'ús de visió de màquina el sistema era capaç de captar peces en orientacions aleatòries i posicions fora d'un recipient.	-	Universitat de Rhode Island
1981	Es va desenvolupar a la Universitat de Carnegie-Mellon un robot d'impulsió directa. Utilitzava motors elèctrics situats a les articulacions del manipulador sense les transmissions mecàniques habituals emprades en la majoria dels robots.	-	Universitat de Carniege-Mellon
1982	IBM va introduir el robot RS-1 per a muntatge, basat en diversos anys de desenvolupament intern. Es tracta d'un robot d'estructura de caixa que utilitza un braç constituït per tres dispositius de lliscament ortogonals. El llenguatge del robot AML, desenvolupat per IBM, es va introduir també per programar.	RS-1	IBM[5]
1983	Sistema de muntatge programable adaptable (APAS), un projecte pilot per a una línia de muntatge automatitzada flexible amb l'ús de robots. el robot SR-1.	-	Westinghouse Electric Corporation
1984	Robots 8. L'operació típica d'aquests sistemes permet que es desenvoluparan programes de robots utilitzant gràfics interactius en un ordinador personal i després es carregaven al robot.	-	-
2000	Robot Humanoide capaç de desplaçar-se de forma bípeda i interactuar amb les persones.	ASIMO	Honda Motor Co.

[3] Programmable Universal Machine for Assambly.

[4] Selective Compliance Arm for Robotic Assambly.

[5] International Business Machines Corp.

Degut a aquest auge a les darreries de segle XX, a partir de l'any 2000 aquest sector va experimentar un gran creixement encara major i és impossible de mostrar tots els avenços que hi han hagut des de llavors donat a que són innumerables. Tot i així, a continuació trobem alguns dels més importants i amb més impacte a la societat.

2002	Comercialització d'un aspirador amb funcionament automàtic.	Roomba	iRobot
2007/2010	Disseny i construcció d'un robot avançat amb aparença humana capaç de relacionar-se, diferenciar menjars, medicaments...	Aiko	Le Trung

Fig 3.1. Cavaller mecànic.

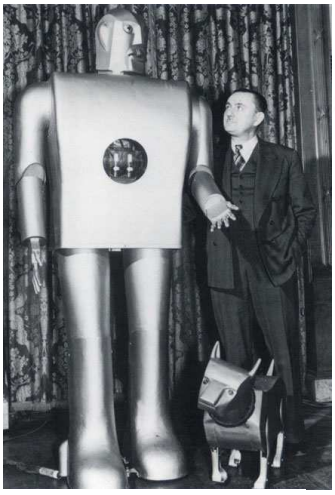


Fig 3.2. Robot Elektro.

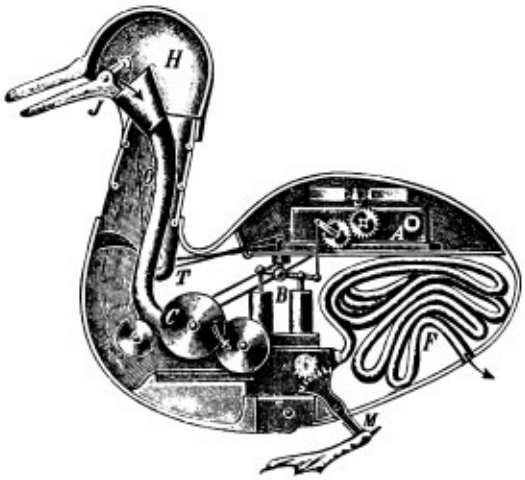


Fig 3.3. Digesting duck.

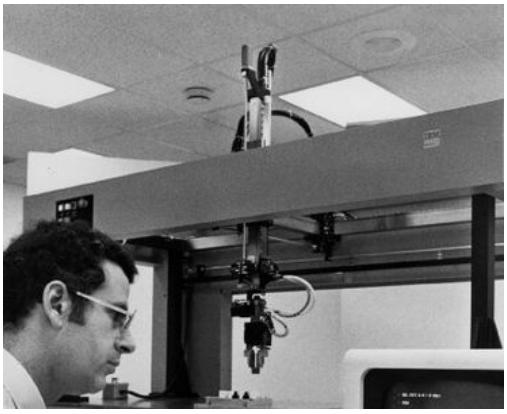


Fig 3.4. RS-1.

Fig 3.5. ASIMO.

Fig 3.6. Roomba.

Fig 3.7. Aiko.

Com podem observar a la taula, a partir de 1800 els projectes creats són de caire més pràctic i busquen facilitar algunes tasques. D'aquesta manera veiem que l'explicació abans realitzada té uns fets reals que la suporten. També salta a la vista que en un principi els dissenys eren elaborats per part de particulars, persones que treballaven soles i normalment no buscaven (com a principal objectiu) un benefici econòmic en els seus projectes, sinó que més aviat s'enfocaven més a la recerca i a la investigació. Podríem afirmar que són persones avançades a la seva època. Ara bé, a partir de 1900 aproximadament la robòtica esdevé un sector amb projecció de futur a nivell econòmic i moltes empreses i grups van apostar per ell. A partir de la dècada dels 80 ja veiem una implicació de les universitats de manera que podem afirmar que la robòtica adquireix un pes important a la nostra societat i s'amplifica la investigació i la recerca en aquest assumpte.

3.1.3. Classificació dels robots

Podem classificar els robots segons diferents criteris.

- Segons la seva cronologia:

- 1ª generació

  Manipuladors. Són sistemes mecànics multifuncionals amb un senzill sistema de control, bon manual, de seqüència física o seqüència variable.

- 2ª generació

  Robots d'aprenentatge. Repeteixen una seqüència de moviments que ha sigut executada prèviament per un operador humà. El mode d'execució es fa a través d'un dispositiu mecànic. L'operador realitza els moviments requerits mentre el robot el segueix i els memoritza.

- 3ª generació

  Robots amb control sensoritzat. El controlador es una computadora que executa les ordres d'un programa i les envia al manipulador per a que realitzi els moviments necessaris.

- 4ª generació

  Robots intel·ligents. Son similars als anteriors, però a més a més disposen de sensors que envien informació a la computadora de control sobre el estat del procés. Això permet que el robot prengui decisions intel·ligents de manera independent i a temps real.

- Segons la seva estructura:

Els podem classificar poliarticulats, mòbils, androides, zoomòrfics i híbrids. Recentment s'ha incorporat un nou concepte al món de la robòtica que amplia la flexibilitat d'operacions i de funcions del robot en qüestió. Aquest concepte, anomenat metamorfisme, es basa en el canvi de la configuració del robot per part d'aquest, és a dir, el robot és capaç de modificar la seva estructura per adaptar-se en cada cas a la tasca que ha de realitzar, de manera que adquireix una flexibilitat funcional molt gran.

El metamorfisme admet diversos nivells, des dels més bàsics i elementals com el canvi d'eines o d'efecte terminal, fins als més complexes com l'alteració d'algun dels seus subsistemes estructurals.

- Poliarticulats

  En aquest grups s’agrupen robot amb una àmplia gama de característiques i particularitats però que tenen un tret en comú: són robots bàsicament sedentaris, és a dir, no es produeix un desplaçament gran del robot sinó que està fixe en una posició. En determinats casos poden ser guiats per a efectuar desplaçaments molt limitats.

  En aquest grup s’inclouen els manipuladors, els robots industrials, els robots cartesians (funcionen mitjançant un sistema de coordenades). s’utilitzen sobretot en cadenes de muntatge o tasques que no requereixin d’un desplaçament autònom o ampli. Un clar exemple seria una grua d’una cadena de muntatge, la qual pot tenir infinitat de característiques diferents segons la seva operació però que és estàtica o amb un moviment molt limitats, normalment guiat per carrils.

- Mòbils

  Al contrari dels anteriors, són robots amb alta mobilitat i capacitat de desplaçament, basats en sistemes locomotors de tipus rodant (sempre depenent del medi on es desplacin). Segueixen el seu camí mitjançant control a distància (dependents) o per la informació que reben dels sensors (independents). Aquests robots s’utilitzen per al transport de peces en una cadena de muntatge normalment. Per a aquestes tasques, el més usual és utilitzar robots independents. aquest estan guiats mitjançant pistes materialitzades a través de radiació electromagnètica de circuits impresos al terra (de manera que no s’obstaculitza el pas a persones degut a que no hi ha carrils físics) o a través de bandes detectades fotoelèctricament (làsers). Poden sortejar objectes i estan dotats d’una intel·ligència artificial elevada, doncs són molt autònoms.

- Androides

  Són robots que intenten reproduir de manera total o de manera parcial la forma i el comportament cinemàtic de l’esser humà. Aquest tipus de robots tenen en l’actualitat cap utilitat pràctica i són bàsicament objecte

d'estudi i d'experimentació, sobretot en el camp de la locomoció bípeda. En aquest cas el principal problema es controlar la dinàmica i la coordinació del robot simultàniament amb l'equilibri del propi.

- Zoomòrfics

  En un cert sentit els zoomòrfics podrien incloure als androides, doncs principalment estan caracteritzats perquè el seu sistema de locomoció imita al dels essers vius. Degut a aquesta característica, i donat a l'ampli ventall de models possibles, és precís classificar els zoomòrfics en caminadors i no caminadors. Actualment els robots no caminadors estan molt poc evolucionats. En canvi, els robots caminadors amb diverses extremitats són molt nombrosos i estan sent la principal via d'investigació per a elaborar vehicles ja siguin pilotats o autònoms capaços de superar superfícies accidentades. La aplicació d'aquest tipus de robots és molt interesant en l'exploració espacial però també terrestre, com per exemple l'exploració de volcans.

3.2. Els sistemes automàtics i de control

Fig 3.8. Cadena de muntatge.

La tecnologia de control abasta tots els procediments i sistemes que permeten automatitzar màquines, aparells i processos de fabricació en general. Ara bé, per a poder entendre millor la naturalesa d'aquesta tecnologia hem de saber amb exactitud el significat del terme automatitzar. Automatitzar és reduir al mínim la intervenció humana en l'accionament de les màquines o aparells en la realització dels processos. Així doncs hi ha màquines que només requereixen d'intervenció humana per posar-les en marxa i per aturar-les, anomenades automàtiques, i d'altres que requereixen de persones en alguna fase del procés o per realitzar certes operacions, anomenades automàtiques.

Els sistemes de control són el conjunt d'elements que actuen de manera coordinada per governar un procés mitjançant la manipulació directa o

indirecta de les magnituds que hi intervenen. El concepte de control és molt ampli: Abarca des de la maneta d'una aixeta amb la qual podem regular el cabal d'aigua que surt fins a el control d'un robot d'una cadena de producció.

3.2.1. Sistemes de control automàtic

Els sistemes de control automàtic tenen com a objectiu aconseguir que una màquina o procés faci les seves funcions amb una intervenció humana mínima. Les tecnologies presents en aquests sistemes es poden dividir en dos grans grups: cablejades i programables.

La tecnologia cablejada s'aplica a dispositius pneumàtics, hidràulics, elèctrics i electrotècnics. Funciona mitjançant la unió física dels elements que constitueixen el sistema. Presenta alguns inconvenients com el volum que ocupa el sistema, la poca flexibilitat davant de modificacions, la baixa adaptabilitat...

La tecnologia programable, però, és molt adaptable i no presenta tota aquesta sèrie d'inconvenients: pot dur a terme diferents funcions sense alterar la configuració física del sistema, simplement canviant el programa que el controla.

3.2.2. Components dels sistemes de control

- Dispositius d'entrada d'ordres: Fan possible que l'operador introdueixi dades i ordres al sistema. Es classifiquen en binaris, numèrics i alfanumèrics:

  Binaris. Són els que permeten l'entrada d'ordres de tipus binari, és a dir, amb dues posicions possibles 0 i 1. Entre ells destaquen els polsadors, els interruptors, els commutadors...

  Numèrics. Són els que permeten l'entrada de nombres, com el teclat d'una calculadora per exemple.

  Alfanumèrics. Són els que permeten introduir tant nombres com lletres com podríem fer amb el teclat d'un ordinador.
- Dispositius d'entrada d'informació: són els denominats com a sensors, elements que prenen dades de la situació del procés o de les variables de sortida i les transmeten a la unitat de control. Es classifiquen segons

el tipus de senyal que proporcionen (binària, numèrica o alfanumèrica) o segons la magnitud que indiquen (temperatura, pressió, cabal...).

- Unitat de control o controlador: És el que conté la funció de transferència per a tractar la informació rebuda.

- Dispositius de sortida d'informació: Són els que transmeten la informació al operador. Es classifiquen en binaris, numèrics i alfanumèrics, tot i que també hi ha d'analògics com els indicadors d'agulla.

- Actuadors: els actuadors són els que s'encarreguen d'operar sobre el procés per aconseguir el resultat. Sovint no estan directament connectats al controlador i requereixen de preactuadors com relés, vàlvules distribuïdores, variadors de tensió...

3.2.3. Tipus de sistemes de control:

3.2.3.1 Sistemes de control de llaç obert

Aquest sistemes es caracteritzen per, un cop activats, executar el procés durant un temps prefixat independentment del resultat obtingut. És a dir, el resultat que s'obté després de realitzar l'operació no influeix en el dispositiu de control. Un exemple seria el funcionament d'un microones: el microones escalfa a una determinada potència durant un cert temps (variables controlades manualment) el que haguem introduït, independentment de si realment s'escalfa a la temperatura desitjada. Tant si el menjar es queda fred com massa calent, el microones farà el procés d'acord amb les variables d'entrada que hem introduït sense tenir em compte el resultat.

Fig 3.9. Esquema d'un sistema de llaç obert.

Està format per dues parts: el controlador i els actuadors. El controlador és el dispositiu que determina i executa el procés per al que està preparat. Sovint disposa d'element d'ajustament o de visualitzadors d'estat. En el cas del

microones, el controlador seria el temporitzador i el control de potència al qual treballa. Els actuadors són els elements que fan l'acció final sobre el dispositiu o procés que es vol controlar. El dispositiu que emet les ones que escalfen el menjar seria l'actuador d'un microones.

3.2.3.2. Sistemes de control de llaç tancat

Aquests sistemes es caracteritzen per l'anàlisi del resultat obtingut al realitzar el procés. Si el resultat no compleix una consigna determinada, el dispositiu de control n'és informat i manté el procés actiu fins assolir el resultat d'acord amb l'establert per la consigna. Posant el mateix exemple que en l'anterior, si el microones funcionés amb un sistema de control de llaç tancat avaluaria la temperatura de l'aliment i la compararia amb la de la consigna. Quan el valor del resultat fos igual que el de la consigna, llavors el procés es finalitzaria.

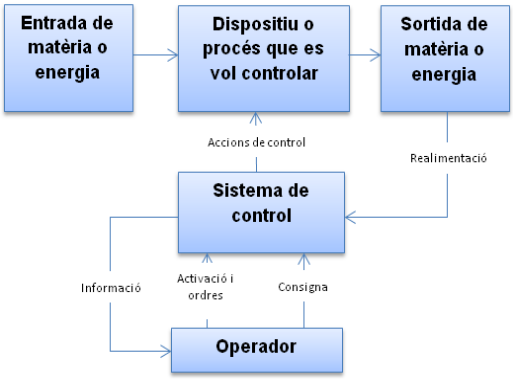


Fig 3.10. Esquema d'un sistema de llaç tancat.

Als ser sistemes més complexos disposen d'una sèrie d'elements diferents dels sistemes de llaç obert. Per a captar les magnituds del procés són necessaris una sèrie d'elements anomenats genèricament sensors o transductors, uns circuits adaptadors anomenats interfícies que ofereixen la informació sobre l'estat del procés i els actuadors, que realitzen les accions finals sobre el procés. També és necessari un element anomenat generador de valor de consigna el qual valora el senyal d'error de la sortida o resultat respecte la consigna mitjançant un comparador.

3.2.4. Funció de transferència

És l'expressió matemàtica que relaciona la variable de sortida amb la d'entrada en un bloc. És independent a la funció de transferència la manera en què estan construïts i amb quina tecnologia funcionen els blocs que duen a terme l'operació: si dos blocs tenen una funció de transferència idèntica es consideren equivalents independentment de si són pneumàtics, elèctrics, mecànics...

3.2.5. Controladors

El controlador és el dispositiu que elabora el senyal corrector que constantment és enviat al element final de regulació del procés amb la finalitat d'aconseguir, restablir o mantenir les condicions de regulació pròximes al valor de la consigna. La seva tasca és comparar el valor de la consigna amb el valor real de la magnitud de sortida del procés i generar el senyal de control més adient per minimitzar els errors i obtenir una resposta tan ràpida com sigui possible davant variacions de consigna o pertorbacions exteriors.

La funció de transferència del controlador normalment obeeixen a uns models bàsics de comportament les quals s'anomenen accions bàsiques de control. Les principals són l'acció proporcional, l'acció integral i l'acció derivativa, la combinació de les tres i finalment el control tot o res:

3.2.5.1. Control proporcional

El control proporcional es caracteritza per la capacitat de correcció exacta (sense desviació permanent) només en una condició específica de funcionament; en la resta persistirà una desviació anomenada offset. L'acció de control C(t) depèn proporcionalment del senyal d'error ε(t), és a dir, es modificarà l'acció de control de manera proporcional al error respecte la consigna i el valor real mesurat.

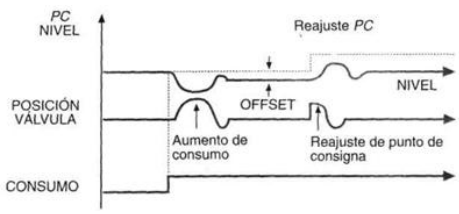


Fig 2.5. Gràfica de funcionament del control proporcional

$$C(t) = K_p \varepsilon(t)$$

K_p = guany o constant proporcional

$C(t)$ = Acció de control

$\varepsilon(t)$ = Senyal d'error

Quan la senyal d'error sigui 0, l'acció de control també serà 0 de manera que no es realitzarà cap modificació sobre el procés. De manera que cal que hi hagi un error perquè es dugui a terme l'acció de control. La funció de transferència G (s) d'aquests controladors és la constant proporcional.

$$G(s) = Kp$$

L'invers de la constant proporcional s'anomena banda proporcional (BP). Aquesta banda està determinada per dos valors, un per sobre de la consigna i l'altre per sota, entre els quals la variable oscil·la. Així doncs podem concloure que l'error del controlador ε(t) és igual a la seva banda proporcional, una característica pròpia de cada controlador.

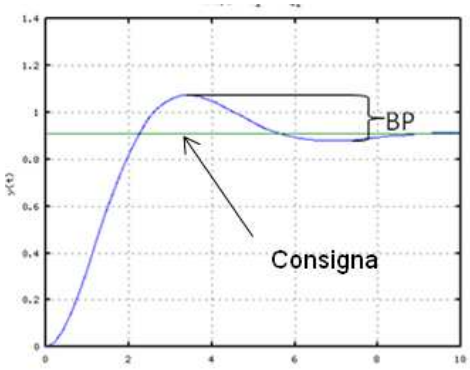


Fig 3.11. Funció de C(t).

$$\varepsilon(t) = \frac{C(t)}{Kp} = BP\ C(t)$$

3.2.5.2. Control integral

Per definició parlem de control integrat quan la velocitat de canvi de la sortida de control és proporcional al senyal d'error d'entrada. L'acció integral permet anul·lar l'error permanent que es produeix al control proporcional.

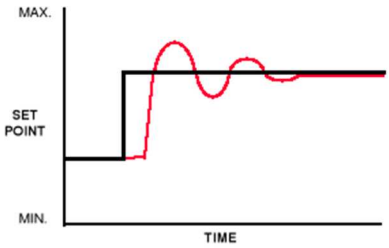


Fig 3.12. Gràfica control integral.

La característica principal d'aquest tipus de control és que mentre hi hagi error es realitzarà l'acció correctora, i serà menys enèrgica com més petita sigui la desviació respecte el valor real i la consigna. L'acció correctora respon tant a la magnitud de l'error com a la durada d'aquest.

L'inconvenient d'aquest sistema és que, al contrari del control proporcional, té una resposta poc enèrgica en els instants immediats després de l'aparició brusca d'un error, ja que té un efecte progressiu. Actua tot seguint la següent expressió matemàtica:

$$C(t) = T_I^{-1} \int_0^t \varepsilon(t)\, dt = K_I \int_0^t \varepsilon(t)\, dt$$

$$\frac{\Delta C(t)}{\Delta t} = T_I^{-1} \varepsilon(t)$$

$C(t)$ = Senyal de control o sortida del controlador.

T_I = Temps d'integració (factor de proporcionalitat invers).

$\varepsilon(t)$ = Error o desviació.

K_I = Constant integral.

Si K_I és un valor elevat, l'error es corregeix ràpidament però es produeix més inestabilitat al sistema. Si K_I és un valor reduït, en canvi, el temps per corregir l'error augmenta però augmenta l'estabilitat del sistema.

Aquest tipus de control no t'utilitza purament, sinó que es combina amb l'acció proporcional. D'aquesta manera aconseguim una actuació ràpida davant l'error i eliminem la desviació permanent. Aquesta combinació s'anomena control proporcional integral (PI), i actua de la següent manera:

$$C(t) = K_p \varepsilon(t) + K_I \int_0^t \varepsilon(t)\, dt$$

3.2.5.3. Control derivatiu

El control derivatiu es caracteritza per què genera un senyal de control proporcional a la velocitat amb la que varia la magnitud d'error amb el temps. Ho podem expressar com que l'acció derivativa s'oposa a les desviacions amb una acció que és proporcional a la rapidesa d'aquestes. El seu comportament ve regit per l'expressió matemàtica següent:

$$C(t) = T_D \frac{d\varepsilon(t)}{dt} = K_D \frac{d\varepsilon(t)}{dt}$$

$$C(t) = T_D \frac{\Delta\varepsilon(t)}{\Delta t}$$

K_D = Constant d'acció derivativa.

T_D = Constant de temps derivativa.

Si la desviació es produeix de manera instantània (temps proper a 0), la velocitat de variació és infinita. Per tant, l'acció derivativa provoca moviments bruscos en la regulació, cosa que no interessa. És per això que sovint es combina amb l'acció integral o la proporcional integral.

La característica principal d'aquest sistema és la capacitat d'anticipació a l'error, doncs va corregint-lo de manera proporcional a la velocitat a la qual augmenta.

3.2.5.4. Control proporcional-integral-derivatiu (PID)

El controlador PID combina els tres tipus de controladors que hem vist amb l'objectiu d'aconseguir obtenir les avantatges de cada un i superar els seus inconvenients. S'expressa mitjançant la següent equació:

$$C(t) = K_p \left\{ \varepsilon(t) + K_I \int_0^t \varepsilon(t)\, dt + K_D \frac{d\varepsilon(t)}{dt} \right\}$$

Mitjançant la modificació de les constants K_p, K_I i K_D aconseguim tractar la resposta de l'acció davant els errors que es presentin. Depenent de com modifiquem cadascuna podem obtenir una resposta més o menys oscil·latòria, més o menys lenta, més o menys precisa...

El controlador òptim llavors és el que aconsegueix que la variable controlada no resulti influenciada per les pertorbacions del sistema i segueixi sense cap retard ni oscil·lació els canvis de la variable de consigna.

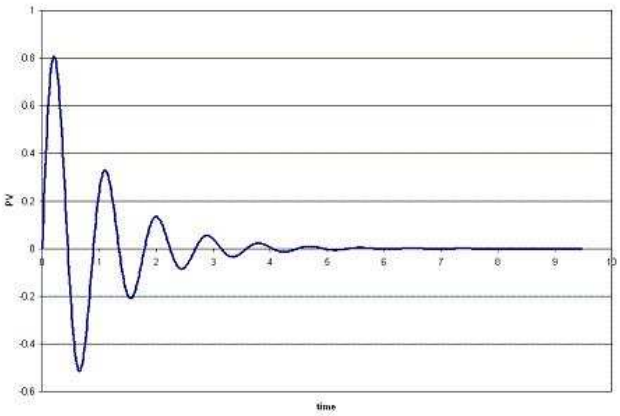


Fig 3.12. Gràfic control PID

3.2.5.5. Control tot o res

El sistema de control tot o res es caracteritza per què la sortida del qual només pot tenir dos estats: connectat i desconnectat, també anomenat màxima i mínima sortida.

Aquest tipus de control és aplicable quan el procés que es vol controlar es comporta amb un retard de temps anomenat de primer ordre, és a dir, amb una constant de temps molt gran, com per exemple els termòstats d'un habitatge.

La seva expressió matemàtica és:

$$\varepsilon(t) = E(t) - R(t)$$

$$C(t) = 1, \qquad si\ E(t) > R(t)$$

$$C(t) = 0, \qquad si\ E(t) < R(t)$$

$E(t)$ = Consigna

$R(t)$ = Realimentació o valor mesurat

$C(t)$ = Senyal de control o sortida del controlador

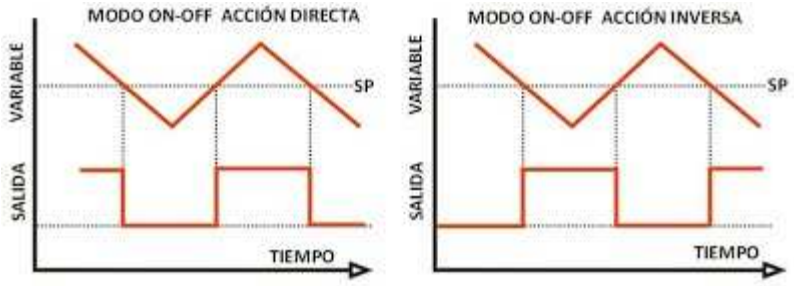


Fig 3.13. Gràfic control tot o res.

3.2.6. Transductors

Els transductors són dispositius que transformen una magnitud física en una altra magnitud física, normalment en senyals elèctrics, entre les quals hi ha una relació determinada.

Estan estructurats en l'element sensor o captador, que converteix les variacions d'una magnitud física en variacions d'una magnitud elèctrica(senyal); el bloc de tractament, que tracta el senyal elaborat pel captador u l'adapta a l'entrada de l'etapa de sortida; i l'etapa de sortida, que adapta el senyal a les necessitats del comparador o el controlador.

Es classifiquen en actius si són els que generen directament el senyal captat o passius si necessiten alimentació externa per captar el senyal. També es classifiquen segons com expressin la magnitud mesurada: analògics, digitals i tot o res. Finalment podem classificar-los segons la magnitud mesurada com es mostra a la taula següent;:

Magnitud	Tipus de transductor	Característica
Posició	Final de cursa	Tot o res
	Microruptor	Tot o res
Proximitat	Inductiu	Tot o res
	Capacitatiu	Tot o res
	Òptic	Tot o res
	Magnètic	Tot o res
Desplaçament lineal o deformacions	Potenciòmetre lineal	Analògic
	Làser	Analògic
	Ultrasònics	Analògic
	Galgues extensomètriques	Analògic
Desplaçament o posició angular	Pontenciòmetres circulars	Analògic
	Codificador absolut i incremental	Digital
	Sincro i resolutor	Analògic
Velocitat lineal o angular	Codificador incremental	Digital
	Dinamo tacomètrica	Analògic
	Detector inductiu o òptic	Digital
Temperatura	Termòstat	Tot o res
	Termoparell	Analògic
	Termoresistència	Analògic
	Piròmetre de radiació	Analògic
Pressió	Mecànic (pressòstat)	Tot o res
	Membrana + detector desplaçament	Analògic
	Piezoelèctrics	Analògic

4. Lego mindstorms

4.1. Història

La línia Lego Mindstorms va néixer en una època difícil per Lego, a partir d'un acord entre Lego i el MIT (Massachusetts Institute of Technology). Segons aquest tracte, Lego finançaria investigacions del grup d'epistemologia i aprenentatge del MIT sobre com aprenen els nens i a canvi obtindria noves idees per als seus productes, que podria comercialitzar sense haver de pagar cap mena de taxa al MIT. Un fruit d'aquesta col·laboració va ser el desenvolupament del MIT Programmable Brick (Maó programable MIT).

El mentor del grup, Seymour Papert, era un matemàtic interessat des de la dècada de 1960 per la relació entre la ciència, l'adquisició del coneixement i el desenvolupament de la ment infantil. De fet, el nom del producte, Mindstorms, prové del títol d'un llibre seu, anomenat *MindStorms: Children, Computers, and Powerful Idees*, en què descriu les seves idees, que defensen l'ús de les computadores com a mètode per al desenvolupament i l'aprenentatge dels nens. Papert és partidari del construccionisme, tesi que sosté que el nen crea el seu coneixement de forma activa, és a dir, que aprèn realitzant petits projectes i superant reptes per mitjà de l'experiència, i que l'educació ha de facilitar eines per a realitzar activitats que impulsin aquesta faceta intel·lectual. La lectura del seu llibre va ser el que va impulsar al president de Lego a contactar en 1985 amb el MIT.

"L'aprendre millor no vindrà d'oferir les millors eines perquè el professor instrueixi, sinó de donar les millors oportunitats als estudiants per construir." – Seymour Papert.

Antecedents i desenvolupament del "Bloc programable"

La línia Mindstorms no va ser el primer fruit la relació entre Lego i el MIT, encara que sí el de més èxit. Amb anterioritat, Lego s'havia interessat pel Llenguatge de programació Logo. D'aquesta manera, va néixer el 1986 Lego TC Logo, creat per Resnick i Steve Ocko. Lego TC Logo era un sistema en què es programava en un ordinador amb llenguatge Logo. Aquest ordinador, que

estava connectat per un cable a una construcció Lego que comptava amb motors, llums i sensors, executava el programa i enviava les ordres finals a l’estructura. Aquest projecte va aconseguir prou èxit entre el públic interessat en aquests nous productes, però els seus impulsors no estaven satisfets amb el resultat, doncs creien que el Lego TC Logo limitava molt als usuaris tant per la part estructural com per la del programari. Així doncs, aquesta línia de desenvolupament continuaria en 1993 amb el llançament de Control Lab, de programari millorat.

Per a solucionar aquests inconvenients ja s’estava estudiant la possibilitat de crear un bloc programable que executés el programa per si sol de manera que no hagués d’estar connectat a un ordinador. El problema principal d’aquest projecte era l’època en el que s’estava plantejant. Cap als inicis de la dècada dels 90 la tecnologia de les computadores era bastant recent i novedosa, i com a conseqüència cara. No era usual tenir un ordinador a casa, ni molt menys un “miniordinador”, que és el que tenien en ment els integrants d’aquest projecte. D’aquesta manera van decidir esperar i donar temps per a que els ordinadors s’estenguessin entre la societat i s’abaratís el seu preu (hem de tindre en compte que al cap i a la fi l’objectiu del projecte era generar beneficis, com qualsevol empresa). Aquest període va tenir una durada de 5 anys.

Llavors, quan les condicions del mercat van ser propícies, es va començar seriosament el desenvolupament del projecte. El resultat final va ser l’anomenat bloc RCX, un bloc de Lego que comptava amb un microcontrolador, i que constitueix el cor del producte Mindstorms. D'aquesta forma les construccions Lego passaven de ser estructures estàtiques a màquines dinàmiques que interactuen amb el món. D'altra banda, mentre que en molts casos els productes Lego proporcionaven les peces necessàries per construir alguna cosa amb un objectiu fix, com un tren o un pont, el que permet "aprendre fent", en el desenvolupament del nou bloc es va seguir en canvi la filosofia de Papert de fomentar el "aprendre dissenyant", i tractar de deixar més obertes les possibilitats.

Les bases que van establir per al desenvolupament del projecte van ser:

- El sistema havia de ser senzill per al nou usuari i alhora havia de permetre realitzar dissenys sofisticats per a l'iniciat.
- La joguina havia de poder-se emprar de moltes maneres diferents.
- Era necessari crear una eina simple.
- El nen havia de tenir a la seva disposició motors, sensors i microcontroladors, però no poder modificar-los o redissenyar. El joc tractaria sobre com combinar-los per fer dissenys, no sobre com dissenyar aquests components.
- Posar èmfasi en l'aprenentatge de la programació. Van elaborar un llenguatge de programació mitjançant imatges ja que van observar que els nens trigaven molt en aprendre el codi escrit, però al veure les limitacions que aquest sistema suposava van decidir fer-ne un híbrid que combinés codi escrit i imatges més intuïtives.
- El bloc havia de tenir un nombre suficient de ports d'entrada / sortida que poguessin connectar amb diferents tipus de sensors: de temperatura, d'amplitud del so o de llum, per a poder donar llibertat alhora de dissenyar les construccions.

Llançament del producte

La primera versió va sortir al mercat amb un preu de 200 dòlars. Incloïa 717 components, entre ells el bloc RCX. Després del seu llançament es van vendre 80.000 unitats en tres mesos. A més, la comunitat d'aficionats a la robòtica, un públic adult, va acollir amb interès aquest nou producte. Aquest interès imprevist del públic adult va fer que les vendes tripliquessin les expectatives.

Programació del bloc

La programació del Lego Mindstorms RCX es realitza mitjançant el programari que s'adjunta en el paquet original, el qual incorpora el firmware del robot i un programa que emula un arbre de decisions, per als quals, l'usuari ha de programar les accions a seguir pel robot. El programari es troba dividit per cada tipus de robot que es pot construir, i que ve recomanat en el paquet original.

Una de les principals característiques d'aquest programari de programació, és el seu entorn visual, el qual emula la construcció per blocs, donant la possibilitat a qualsevol aprenent a acostumar relativament ràpidament a la programació de bloc.

Aquest llenguatge permet les instruccions seqüencials, instruccions de cicles i instruccions de decisions, aquestes últimes, basades en les dades reportats pels sensors que es poden afegir al robot.

4.2. Evolució del brick programable Lego Mindstorms

Al llarg del temps, el brick de la línia Lego Mindstorms ha patit una evolució i una sèrie de millores pròpies de qualsevol producte tecnològic. En un primer moment, el MIT va elaborar un prototip de brick al 1996 tot i que no va se comercialitzat. Va ser amb la sortida del bloc RCX dos anys més tard quan es va iniciar el que es convertiria en la línia de productes Mindstorms de la mà de Lego.

4.2.1. Primera generació: RCX

El bloc RCX és el principal component d'aquest producte. És on es troba la part lògica i electrònica del robot i on s'emmagatzema el programa a executar (fins un total de 5 programes) determinant així les accions a fer pel robot. A més, és on hi ha instal·lat el firmware(1) per al control del bloc.

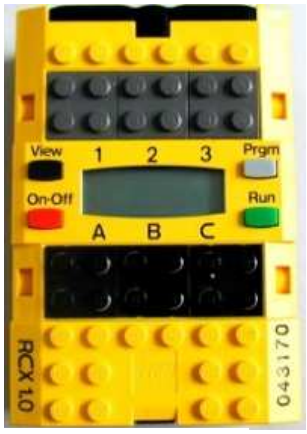


Fig 4.1. Brick RCX

Aquest brick té tres versions diferents: 1.0, 1.5 i 2.0. la modificació que es va fer entre una versió i l'altra és bàsicament de software(2) però són incompatibles entre elles, doncs utilitzen diferents voltatges, tot i que en hardware(3) no varien massa.

Components

El cor del RCX és el seu microcontrolador(4) Hitachi H8, el qual compta amb 32 KB de memòria RAM(5) i 16 de memòria ROM(6), funcionant a una freqüència de 16 MHz i posseeix un descodificador analògic digital que

converteix les senyals elèctriques en bits(7). Una de les principals mancances que ofereix aquest microcontrolador és la incapacitat per a executar dos ordres simultàniament. A ull nuu, sembla que el robot realitzi dos processos al mateix temps, però és una falsa sensació, doncs el temps entre operacions és molt baix i imperceptible, però realment no les realitza alhora.

Com a components externs compta amb la possibilitat de conectar-li 3 sensors i 3 motors. La connexió es realitza mitjançant un brick de construcció Lego amb material conductor a les puntes, fet que permet la transmissió dels impulsos elèctrics. A la part superior, anomenats amb els nombres 1, 2 i 3 i de color gris, trobem els ports de connexió dels sensors. A la part inferior, de color negre i denominats amb les lletres A, B i C, trobem els ports de connexió dels motors, els quals funcionen amb un voltatge de 9V.

El bloc compta amb una pantalla LCD(8) en la que se'ns mostren 3 missatges: a la part superior, la detecció dels sensors connectats i el nivell de càrrega de les bateries; a la part central apareixen valors numèrics que mostren els valors dels sensors, un temporitzador i un comptador; a la part inferior indica el sentit de rotació dels motors i per últim, a la part esquerra indica si hi ha connexió mitjançant rajos infrarojos.

Pel que fa referència a les connexions, compta amb un port infraroig que permet al robot comunicar-se a distància amb la computadora però també amb mòbils. La connexió es realitza a 37 KHz de freqüència (l'equivalent aproximadament a un comandament a distància d'una televisió) i té un abast màxim de 30 cm, sovint reduït per la contaminació lumínica de l'ambient. No poden haver-hi objectes entremig del robot i el comandament, doncs va per rajos infrarojos (llum). També es por connectar via USB.

Està alimentat per 6 piles d'1'5 V connectades en paral·lel que donen energia tan al brick com als seus motors. Això implica que el voltatge total de les bateries sigui de 1'5V però que les bateries durin més temps.

Programació

Els blocs RCX normalment són programats a partir d'un ordinador utilitzant el software que s'inclou a la caixa anomenat RoboLAB. El robot i

l'ordinador es comuniquem mitjançant un tipus d'arxiu proveït per Lego anomenat Spirit.ocx. També és possible utilitzar variants del llenguatge Java i C i fins i tot es pot controlar el brick mitjançant programes directes des del ordinador (no cal compilar-los).

4.2.2. Segona generació: NXT

Aquesta generació de bricks es considera la precursora dels robots LEGO Mindstorms i es van elaborar 4 versions diferents: 1.0, 1.1, 2.0 i 2.1.

Fig 4.2. Brick NXT

Components

El bloc NXT compta amb dos microprocessadors: un ARM7 de 32 bits que inclou 256 Kb de memòria Flash(9) i 64 Kb de memòria RAM externa, i un de 8 bits amb 4 Kb de memòria flash i 512 Kb de RAM, millorant la capacitat del robot d'executar programes i evitant problemes d'execució de tasques de manera simultània.

En quant a entrades i sortides té una millora en la quantitat d'entrades respecte el seu antecessor, doncs compta amb quatre ports de connexió per als sensors en comptes de tres. Pel que fa al nombre de sortides es mantenen en tres. Un detall important és que els actuadors i els sensors de la generació posterior no són compatibles amb els del NXT però existeixen adaptadors per als sensors que fan possible la connexió.

Té una pantalla LCD de 64x100 píxels que mostra la interfície del firmware del robot, i compta amb un altaveu de 8 KHz de freqüència.

Pel que es refereix a comunicacions el robot consta de dos mètodes diferents de comunicació. Principalment per comunicar-se amb la computadora mitjançant un port USB per a transferència de dades, però també és possible (mitjançant una connexió Bluetooth(10)) enllaçar el brick amb un altre brick, que permet realitzar tasques de manera conjunta amb altres robots, o inclús es pot

establir comunicacions amb ordinadors, telèfons mòbils i d'altres aparells que disposin d'aquesta eina. Això obre un món de possibilitats molt gran als usuaris a l'hora de fer els seus projectes, doncs permet la comunicació entre dos bricks i la transmissió de dades a distàncies molt superiors a les del RCX sense cablejat.

El firmware del robot (inclòs a un CD a la caixa del producte) ens permet fer petits programes, testejar els sensors, administrar els arxius emmagatzemats al robot i ajustar paràmetres.

Està alimentat per 6 piles de tipus AA d'1,5V o bé per una bateria de liti recarregable de 10V que ocupa més volum que no pas les piles convencionals, de manera que s'ha de ser curós si es fan proves amb piles però en un futur es vol connectar una bateria.

Programació

Es pot programar en RoboLab però s'ha creat una versió més nova, el RoboLab 2.9 que conté noves opcions. A més, hi ha varis entorns de programació del bloc:

- BricxCC (Brick Comand Center), creat per John Hansen i que també és compatible amb RCX.
- RobotC, un entorn de programació creat per la acadèmia de robòtica de la CMU[6] que utilitza el llenguatge C i treballa amb el RCX i el NXT Robotics Studio de Microsoft.

També hi ha programes per a controlar el brick des del ordinador com:

- BrickTool, un entorn basat en la connexió Bluetooth entre l'ordinador i el NXT creat per John Hansen.
- ICommand, una API de Java que permet controlar el robot remotament creada per Brian Bagnall.
- NXT#, una llibreria de Lego Mindstorms creada per Bram Fokke.
- NXT Perl API, una interfície de control del NXT escrita en Perl(12) creada per Michael Collins.

[6] Carnegie Mellon University

- Ruby-nxt, una llibreria per a controlar el NXT de manera remota en codi Ruby(13).
- LibNXT, una llibreria de comunicació entre l'amfitrió (el que controla, el master) i el brick escrita en C i creada per David Anderson.

4.2.3. Tercera generació: EV3

La tercera i fins al moment última generació dels robots programables Lego Mindstorms és L'EV3 (Evolution 3), que es va comercialitzar al 2013. Compta amb varis models base amb instruccions i també gràcies a l'enorme comunitat d'usuaris podem trobar varis models de robot amb instruccions inclús guies per a construir un model propi.

Fig 4.3. Brick EV3.

Components

El bloc EV3 conté un microprocessador ARM9 que funciona a 300 MHz i té una memòria RAM principal de 64 Mb i una de 64 Kb addicional.

Consta de 4 entrades per a connectar els sensors i 4 sortides per a connectar els actuadors, de manera que el nombre de sortides respecte el seu antecessor és major. D'aquesta manera un brick pot alimentar i manar fins a quatre motors. Una diferència important és que les connexions ja no tenen la pestanya al centre, sinó desviada a un lateral, com podem observar a la imatge està desplaçada a la dreta. D'aquesta manera no es confonen els cables amb els de telèfon, que treballen a voltatges superiors fora dels marges de seguretat i podien ocasionar danys als circuits inters del brick o malmetre els elements externs com motors o sensors.

Té una pantalla LCD mono cròmica (un sol color) de 178x128 píxels on es mostra la interfície del firmware del robot. Compta amb un altaveu per a reproduir sons.

Pel que fa a la comunicació, el robot permet la transmissió de dades mitjançant USB i Bluetooth (com

Fig 4.4. Comandament per infrarojos

l'anterior versió), però també té un connector USB host(11) que permet enllaçar un connector Wi-Fi i poder establir comunicacions tan amb un sol brick com amb diversos, de tal manera que permet crear xarxes Wi-Fi on estan enllaçats diferents robots. També podem donar ordres mitjançant un comandament a distància que transmet la informació per infrarojos. El mode d'alimentació és igual que el de la versió anterior NXT: 6 piles de tipus AA de 1'5V reemplaçables per una bateria de liti recarregable de 10V.

Programació

El brick EV3 es pot programar mitjançant diversos llenguatges:

- .EV3, el programari oficial de Lego que ve inclòs a la caixa de l'EV3, un programari molt visual i intuïtiu.
- RobotC, del qual va sortir una versió per a EV3 al febrer de 2014.
- BricxCC, en la seva nova versió per a EV3 el qual suportava el llenguatge EVC (llenguatge basat en C creat per John Hansen) i el leJos (variant de Java per a mindstorms) entre d'altres.
- Python, un popular llenguatge de programació.

4.2.4. Taula comparativa

	RCX	NXT	EV3
Microprocessador	Hitachi H8 32 KB de memòria RAM 16 de memòria ROM 16 MHz	ARM7 de 32 bits 256 Kb de memòria Flash 64 Kb de memòria RAM externa ARM7 de 8 bits 4 Kb de memòria flash 512 Kb de RAM	ARM9 64 Mb RAM 64 Kb addicional 300 MHz
Entrades/sortides	3 entrades 3 sortides	4 entrades 3 sortides	4 entrades 4 sortides
Pantalla	LCD de tres missatges	LCD 64x100 p	LCD 178x128 p
Altaveu	-	8 Khz	Sí
Connexió	USB (esclau només) Port infrarojos	USB (esclau només) Bluetooth	USB (master i esclau) Bluetooth Xarxa Wi-Fi Port infrarojos
Alimentació	6 piles de 1'5V en paral·lel.	6 piles de 1'5V en paral·lel. Bateria de liti de 10V	6 piles de 1'5V en paral·lel. Bateria de liti de 10V
Programació	- RoboLab - Variants de Java i C	- RoboLab 2.9 - BricxCC - RobotC - BrickTool - ICommand	- .EV3 - RobotC - BricxCC - Python

		- NXT Perl API - Ruby-nxt	

4.3. Programari

4.3.1. Programa de Lego Mindstorms

La programació del Lego Mindstorms es realitza mitjançant el programari que s'adjunta en el paquet original, el qual incorpora el firmware del robot i un programa que emula un arbre de decisions, amb el qual l'usuari ha de programar les accions a seguir pel robot . El programari es troba dividit per cada tipus de robot que es pot construir, és a dir, cada tipus de robot té una estructura en arbre diferent segons el seu disseny.

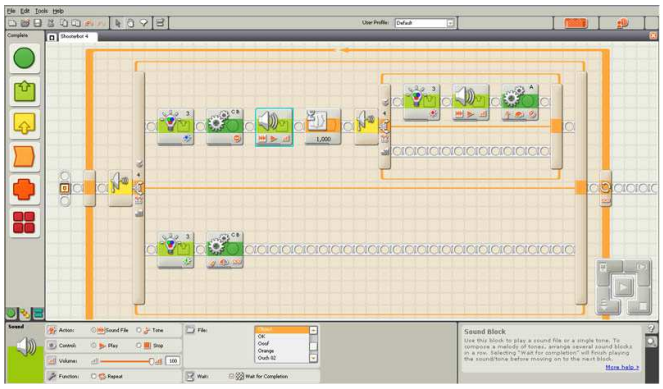


Fig 4.5. Interfície del programa original de Lego Mindstorms.

Una de les principals característiques d'aquest programari, és el seu entorn visual, el qual emula la construcció per blocs, donant la possibilitat a qualsevol aprenent a acostumar-se relativament ràpid a la programació de bloc i convertint-la en una tasca molt senzilla apta per a un ampli públic (inclòs nens).

Aquest llenguatge permet les instruccions seqüencials, instruccions de cicles i instruccions de decisions, aquestes últimes basades en la informació sobre el medi que capta el robot, és a dir, interactuant amb l'entorn, no tan sols efectuant un procés.

4.3.2. Llenguatges alternatius de programació utilitzables amb Lego Mindstorms

El bloc del Lego Mindstorms és un producte de maquinari i programari integrat, però pot ser programat amb diverses interfícies. Això es pot fer utilitzant les eines correctes per poder accedir al firmware bàsic de Lego.

Alguns dels frameworks més coneguts són el BrickOS, LeJos i Not Quite C.

- **BrickOS**

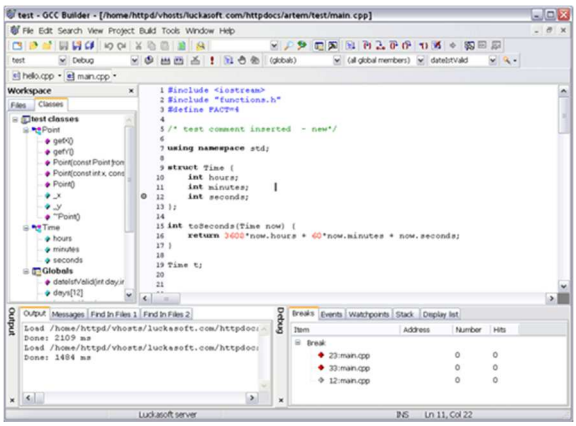


Fig 4.6. Interfície del compilador GCC de Linux.

BrickOS és una llibreria d'instruccions i programes que permeten al programador ingressar de forma directa a la BIOS del bloc i instal·lar-hi un microsistema operatiu (l'equivalent a Windows o Linux a les computadores convencionals) amb el seu respectiu nucli operatiu i llibreries necessàries per enllaçar tots els recursos que disposa el bloc. Per ser instal·lat ha de sobreescriure's l'àrea on es troba el framework original, però amb aquest canvi, el bloc pot ser programat en C, C ++ i assembler.

BrickOS està suportat en la majoria de les distribucions de Linux i en Windows, usant el compilador que porta integrat Linux (gcc o gcc ++), generant el mapa de bytecodes per controlar les accions del bloc.

- **LeJOS**

Fig 4.7. Firmware LeJOS a un NXT 2.0.

A diferència de BrickOS, no instal·la un sistema operatiu en reemplaçament del firmware del bloc RCX, sinó que instal·la una màquina virtual de Java, la qual cosa permet el bloc sigui programable en el llenguatge Java, per la qual cosa no depenen d'un compilador o sistema operatiu per a ser reemplaçat. No obstant això, la transparència de processos per al programador és més baixa a causa de la programació orientada a objectes[7] que restringeix LeJOS, fent que el programa de BrickOS sigui més utilitzat per la transparència de processos tant interns com externs. Aquest programa és molt utilitzat amb els estudiants de primer any de carrera per a programació de màquines.

[7] Veure pàg. 53

- **Not Quite C**

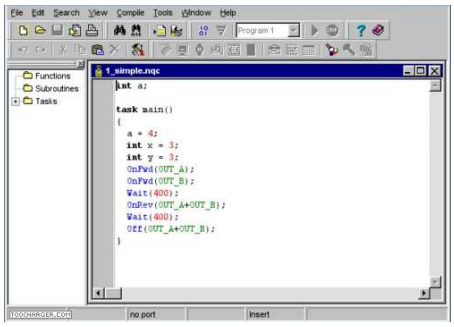


Fig 4.8. Interfície del programa Bricx Command Center per a programar en NXC.

Not Quite C és l'únic conjunt de programes que no reemplaça el framework original del bloc, però això representa un desavantatge, per que ha de coexistir al costat del framework original, per tant emular les seves instruccions, fent que el procés sigui més lent que per la metodologia de reemplaçar el framework. La seva versió per al desenvolupament de programes pels productes de la línia Mindstorms és el Not eXactly C (NXC).

Not Quite C aquesta disponible per a Mac OS i Windows, i utilitza com a llenguatge de programació una versió pròpia de C. A més, és compatible amb la majoria de la línia de Mindstorms, la qual cosa ho fa el més extens de tots els llenguatges alternatius, a causa a les seves capacitats de migrar entre plataformes dels blocs de RCX.

- **Physical Etoys**

Physical Etoys permet la programació del bloc NXT mitjançant un entorn de programació visual. Tot i que comparteix l'aspecte visual amb el programari proveït per Lego, la metodologia de programació de Physical Etoys és molt diferent.

Ofereix dues maneres de programació:

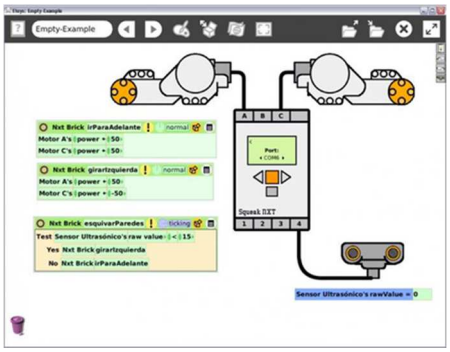


Fig 4.9. Interfície del Physical Etoys.

El mode "directe", en el qual els programes s'executen a l'ordinador de l'usuari i les ordres es transmeten al bloc NXT immediatament a través de la comunicació per Bluetooth o USB. Aquest mode permet modificar els programes i veure els canvis produïts de manera immediata en el comportament del robot (sense necessitat de compilar), la qual cosa facilita

molt la programació, sobretot a l'usuari inexpert. Així mateix, en proveir una comunicació ininterrompuda entre el robot i l'ordinador, permet veure constantment els valors dels sensors i utilitzar el robot, per exemple, com un adquiridor de dades.

El mode "compilat", en el qual els programes es baixen al bloc NXT usant el llenguatge Not eXactly C abans mencionat. El mode "compilat", per la seva banda, elimina el temps necessari per enviar l'ordre des de l'ordinador fins al bloc, el que és preferible pel desenvolupament de tasques autònomes, on la velocitat de resposta del robot ha de ser màxima.

Així doncs, com podem observar cada llenguatge de programació, ja sigui qualsevol d'aquest alternatius o el que ve integrat al paquet original, ens ofereix una sèrie de característiques diferents que afecten al desenvolupament del nostre projecte i a les funcions que podrà realitzar el robot, així com a les limitacions que tindrà. És per això que hem de tenir en compte quina alternativa escollirem alhora d'iniciar el projecte i ser coneixedors de les avantatges i els inconvenients que ens suposa la nostra elecció.

Per exemple, si el nostre projecte està constituït per un model de robot estàtic que té la funció de traslladar un objecte d'una posició a una altra o fer una seqüència de moviments determinada (com podria ser una cadena de muntatge), estarem davant d'un disseny el qual no necessita interactuar amb el medi de manera que no ens caldrà un programa compilat i emmagatzemat al brick. En aquest cas utilitzaríem el Physical Etoys en la seva versió directa perquè les seves característiques són les que més s'adeqüen al nostre projecte.

En canvi, si parlem d'un robot el qual hagi de fer funcions de manera autònoma o desplaçar-se per un espai determinat (delimitat o no), llavors hauríem de fer servir un programari que instal·lés l'arxiu al brick per a que aquest no depengués d'un ordinador central. En aquest cas, utilitzaríem qualsevol de les altres tres opcions alternatives (BrickOS, LeJos o Not Quite C).

També cal atendre al nivell d'experiència que té l'usuari que desenvolupi el projecte, de manera que podríem ordenar aquestes opcions per a programar el nostre brick (dificultat ascendent):

Com a més senzill tindríem el programari que ve inclòs al paquet original, el programa estructurat en arbre i basat en imatges. Aquest sistema és molt intuïtiu i senzill però limita molt al usuari, de manera que és una opció més enfocada a nens.

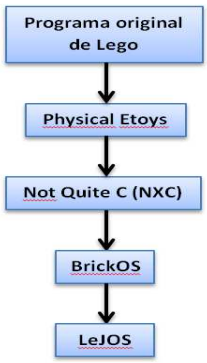


Fig 4.10. Ordre dels programes per raó de dificultat creixent.

A continuació trobaríem el Physical Etoys, que guarda estreta relació amb l'original de Lego però ofereix més possibilitats al programador degut a els seus dos mètodes per a transmetre el programa al robot.

Seguidament tenim el Not Quite C. Els motius pels quals és més senzill que els altres són que no s'ha d'instal·lar un sistema operatiu ni una màquina virtual i que la seva versió NXC per a Mindstorms és bastant senzilla d'utilitzar tot i que guardi una estreta relació amb el llenguatge de programació C.

Ja finalitzant trobaríem el BrickOS, que per les característiques abans mencionades ofereix una àmplia llibertat per a elaborar codi al programador, fet que alhora dificulta el seu ús i aprenentatge.

Per últim tenim el LeJOS, un programa que ofereix moltíssimes opcions i dota al programador d'una àmplia llibertat alhora de determinar el comportament del robot, sobretot quan parlem de robots autònoms que interactuen amb el medi i han de donar respostes al que analitzen de manera independent. El fet que sigui un derivat de Java permet la comunicació amb dispositius que utilitzen aquest llenguatge, com dispositius Android.

4.4. Components Lego Mindstorms NXT

El kit Lego Mindstorms inclou una sèrie de components que permeten construir el robot al nostre gust. A continuació veurem quins són aquest components i els analitzarem amb detall. Podem dividir aquest components en peces de construcció, sensors i actuadors (apart del controlador).

4.4.1. Peces de construcció

Per a poder col·locar els actuadors i els sensors en la disposició que desitgem hem de primer crear una estructura que faci de suport. Les peces de construcció tenen aquesta funció. Per a una major flexibilitat de l'estructura Lego ofereix diferents tipus de peces com les peces de rotació, peces flexibles que permeten crear una articulació i les peces de fixació, que ens permeten unir les diferents peces d'infinites maneres per elaborar les nostres estructures.

En el cas dels robots mòbils, el kit ofereix diferents tipus de rodes que tan es poden fer servir com a tals com es poden utilitzar com a politges. També disposem d'engranatges per transmetre el moviment entre aquestes rodes i politges amb diferent nombre de dents i diàmetre, cosa molt útil a l'hora d'establir sistemes de reducció o acceleració segons les nostres necessitats.

4.4.2. Sensors

Els sensors són components essencials a l'hora de construir robots que siguin capaços de prendre decisions per si sols, doncs proporcionen informació essencial sobre el medi. Els sensors disponibles amb el kit són el sensors d'ultrasons, el de tacte, el de so, el de lluminositat, el de rotació, el de color, el de temperatura... entre d'altres.

- **Ultrasons:** es tracta d'un transductor de desplaçament lineal o deformacions. Determina a quina distància està l'objecte més proper mitjançant rajos ultrasons

Fig 4.6. Sensor d'ultrasons.

imperceptibles per l'oïda humana: emet les ones i calcula el temps que triguen a tornar fins al sensor(principi de l'eco). Proporciona valors analògics entre 0 i 255 cm.

- **Tacte:** És un transductor de final de cursa. Determina si està pressionat o no. Marca dos valors: pressionat i no pressionat (tipus booleà[8] o tot o res).

Fig 4.7. Sensor de tacte.

- **So:** Detecta els decibels mitjançant una escala logarítmica i proporciona un valor en forma de percentatge.

  Les equivalències serien les següents:

  -5%: Casa silenciosa.

  5-10%: Conversa llunyana al sensor

  10-30%: Conversa propera al sensor

  30-100%: Crits o música a volum alt prop del sensor.

Fig 4.8. Sensor de tacte.

- **Lluminositat:** És un transductor de desplaçament lineal o deformacions, concretament un de tipus làser. Calcula el nivell de llum reflectida mitjançant una llum vermella i expressa el valor en forma de percentatge mitjançant la fórmula següent:

$$\boldsymbol{Llum} = \frac{\mathbf{145}-\boldsymbol{RAW}}{\mathbf{7}}\ \%$$

Fig 4.9. Sensor de llum.

- **Rotació:** Equivaldria a un transductor de velocitat angular, concretament una dinamo tacomètrica Està a l'interior del servomotors. Permet mesurar l'angle de rotació per unitat de temps, és a dir, la velocitat angular.

Fig 4.10. Sensor de rotació.

$$\omega = \frac{\Delta\theta}{\Delta t}\ (\text{rad/s})$$

[8] Veure punt 7.1.

- **Color:** Obté la composició del color segons els seus valors de blau, verd i vermell i determina de quin color es tracta segons aquest nivells. Proporciona un valor entre 0 i 255 per a cada color.

Fig 4.11. Sensor de tacte.

- **Temperatura:** Permet calcular el valor aproximat de temperatura mitjançant un termistor que genera un camp magnètic que permet calcular la temperatura a la que està el bloc. Segueix la següent fórmula:

Fig 4.12. Sensor de temperatura.

$$Temp = \frac{785-RAW}{8}(^{o}C)$$

4.4.3. Actuadors

Els actuadors del sistema són servomotors elèctrics. Podem determinar a la velocitat a la que volem que girin, amb quina acceleració angular o quin angle volem que descriguin. A més, com abans he mencionat funcionen com a sensors de rotació. Està compost pels següents elements:

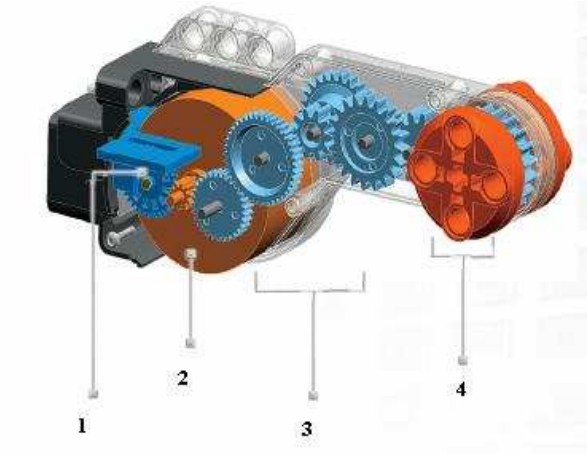


Fig 4.13. Servomotor.

1. Tacòmetre amb sensor de rotació
2. Suport del motor
3. Caixa de canvis
4. Connector de rodes i altres elements

MARC PRÀCTIC

L'elaboració del robot ha estat realitzada mitjançant tres kits de Lego Mindstorms en la seva versió 2.0. Tot i així, degut a les necessitats en quant a entrades i sortides per connectar components com motors i sensors al meu robot he utilitzat dos brick NXT 2.0. Per un correcte funcionament de l'estructura, he dissenyat i imprès peces amb l'impresora 3D. A l'hora d'escriure el codi que regiria les accions a seguir pel robot vaig haver de fer servir Java, ja que és l'únic que em permetia controlar el bloc mitjançant una aplicació Android.

Així doncs, la part pràctica està dividida en tres subapartats:

- Muntatge del robot
- Disseny de peces 3D
- Programació del robot

5. Muntatge del robot

El robot es basa en una imitació d'un camió amb una grua motoritzada que li permet fer la funció al vehicle de, no només transportar els materials, sinó també tenir la capacitat d'emmagatzemar-los per si sol, sense l'ajuda d'un altre tipus de vehicle com podria ser una grua o una excavadora (posant exemples del món de la construcció). D'aquesta manera, el robot està compost per un sistema motriu a la part darrera de l'estructura, un sistema de direcció a la part del davant, una grua motoritzada al mig de l'estructura uns 10 cm més avançada que les rodes posteriors, els dos Brics NXT 2.0 situats a la part davantera a sobre del sistema de direcció i uns petits contenidors (fets a mida mitjançant el programa de disseny en 3D SketchUp 2015 i impresos amb l'impresora 3D) on el robot emmagatzema el que recull classificant-ho per colors mitjançant la grua situats a la part darrera, on es troben les rodes motrius.

5.1. Estructura principal

L'estructura principal es basa en un rectangle de dimensions 30 x 15 cm, construït amb majoritàriament amb peces Lego tot i que consta d'alguns

elements dissenyats amb Sketchup i impresos amb impressora 3D que milloren les prestacions que les peces ja prefabricades de Lego ens ofereixen, de manera que podem crear una peca que s'ajusti més a les nostres necessitats.

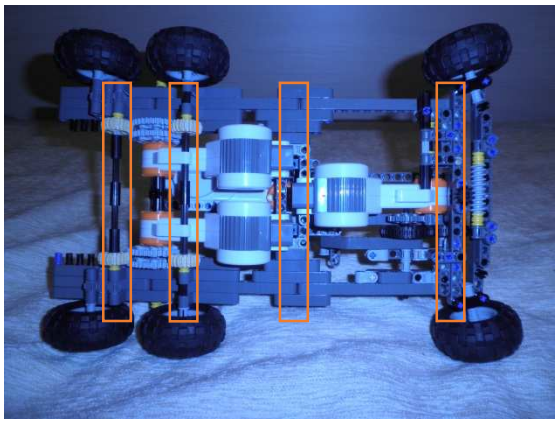


Fig 5.1. Posició dels eixos transversals inicial.

En un principi, l'estructura constava de quatre unions horitzontals, és a dir, quatre unions entre ambdós laterals que formen el xassís i donen rigidesa i força al robot per a suportar el pes dels seus components. Dos d'aquest quatre eixos transversals constituïen els eixos motrius de las rodes posteriors. Els dos restants estaven situats un al sistema de direcció i l'altre tot just a sota de la grua. Amb aquesta disposició dels eixos s'aconsegueix la major resistència de l'estructura amb el menor volum ocupat, de manera que tenim una estructura senzilla, resistent i que ens permet construir els components a damunt sense cap problema d'espai.

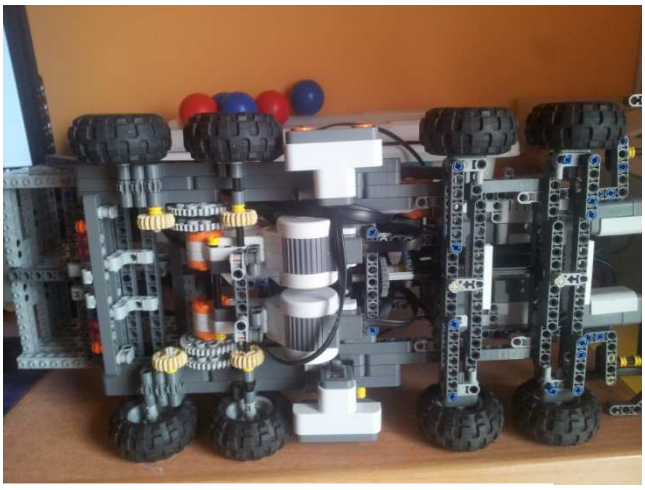


Fig 5.2. Posició dels eixos transversals final.

Degut a certs problemes relacionats amb el gir[9] i d'altres modificacions, aquesta disposició dels eixos ha sigut modificada. El robot final té un total de cinc eixos transversals: dos eixos que formen part del sistema de direcció, l'eix (ja existent en l'estructura anterior) situat a sota de la grua, un que uneix els motors de tracció i un situat a sota de la caixa on el robot diposita el que recull. D'aquesta manera hem guanyat un eix que provoca una millora en la fermesa de l'estructura.

[9] Veure 5.3. Direcció

5.2. Tracció

Inicialment el sistema de tracció estava format per dues rodes darreres amb un sistema d'acceleració mitjançant dos engranatges amb una relació de transmissió de 1:3[10], de manera que la velocitat del motor es triplicava. Al reflexionar sobre el pes que hauria de suportar l'estructura, vaig decidir valorar l'opció de posar dues rodes motrius darreres que poguessin repartir millor el pes del robot. Per a instal·lar la roda necessitava una certa distància per a que no es toquessin una amb l'altre, de manera que, com l'engranatge que transmetia el moviment del motor a les rodes no permetia la distància necessària entre ambdós eixos, vaig haver de fer un petit sistema d'engranatges. Amb la col·locació de tres rodes dentades, vaig aconseguir desplaçar el suficient els eixos i mantenir el sentit de rotació, doncs si col·locava un nombre parell d'engranatges el sentit de rotació canviava i quan una roda avançava, l'altre retrocedia.

Així doncs la col·locació d'una roda més al costat de la que hi havia instal·lada, però canviant la posició de les dues centrant-les una mica més, em va permetre aconseguir solucionar el problema que més endavant se'm podria presentar.

El nou sistema format per dues rodes constava igualment de dos motors com l'anterior però aquest repartien la força en quatre rodes. Pel mateix motiu, el sistema d'acceleració té una relació de transmissió de 1:2, de manera que el parell motor es veu incrementat tot i que la velocitat angular es veu reduïda (la potència del motor no

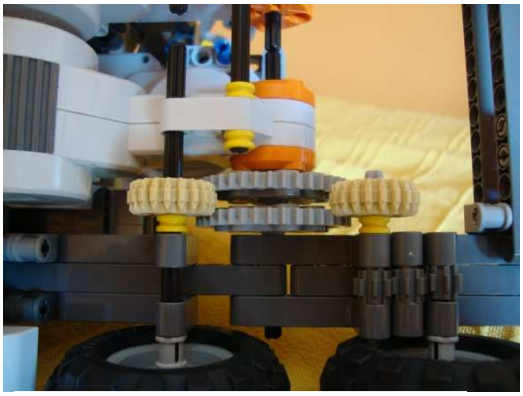


Fig 5.3. Transmissió de la tracció.

[10] Per cada volta del motor, la roda en fa tres.

es veu afectada)[11]. Així aconseguim major força (per tant arrossegar més pes) a l'hora de fer el treball de desplaçament del robot, tot i que recorrerem menys distància en el mateix temps.

Però a l'hora de fer la prova vaig comprovar que per una raó simplement de composició dels materials les dents dels eixos saltaven. Com les peces Lego són de plàstic, les dents dels eixos no suporten la força i salten, de manera que la transmissió no conserva el moviment, el motor no transmet tota la seva rotació. Per tant els eixos no giraven per igual. A més aquest error, com les dents saltaven de manera aleatòria, provocava desigualtats en els moviments dels eixos de dreta i esquerra, per tant el robot es torçava.

La solució la vaig trobar fent una simple pinça amb un altre engranatge al eix del motor, de manera que és impossible que les dents dels eixos saltin, estan bloquejades entre els dos engranatges.

5.3. Direcció

El sistema de direcció imita al de qualsevol vehicle convencional com podria ser un cotxe o un camió.

En un principi, constava d'un eix fixe unit a l'estructura del xassís i d'un segon eix mòbil que permet la variació en la inclinació de les rodes. Sota aquest eix mòbil teníem un cargol sense fi que actuava com a cremallera, i un engranatge simple. Amb aquest sistema canviem l'eix de rotació en 90º i convertim el moviment en lineal. L'eix on es trobava aquest engranatge estava en posició vertical. Situat a la part inicial d'aquest eix vertical hi havia una peça que canviava el sentit de rotació en 90º però continuava essent un moviment circular. Amb aquesta peça traslladàvem el moviment de

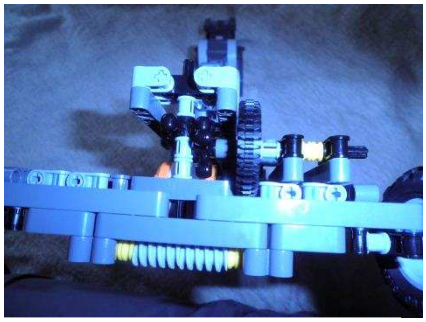


Fig 5.4. Direcció part frontal.

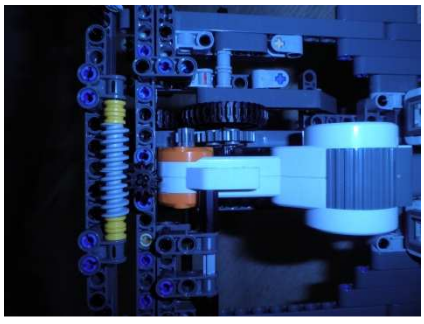


Fig 5.5. Direcció part inferior.

[11] **Potència = Parell motor · velocitat angular.** Si el parell motor augmenta en la mateixa mesura en que la velocitat angular disminueix, la potència es manté degut a que ambdues són directament proporcionals.

manera paral·lela als eixos transversals que formen la direcció però a l'alçada del motor, situat centrat i davant dels motors de la tracció. El motor transmetia el moviment a un eix que a la vegada el transmetia a l'eix paral·lel mencionat amb un sistema de reducció de 3:1. Tot aquest sistema d'eixos constava de petites estructures unides al xassís que donaven rigidesa al sistema i feien que les dents del eixos no saltessin.

Al posar a prova aquest sistema em vaig adornar que a l'hora de efectuar el gir, com l'eix mòbil (on està el cargol sense fi) s'acosta a l'eix fixe (l'ancorat al xassís i que roman immòbil), l'engranatge anava molt forçat i això provocava desajustos continus en el sistema en general i una imprecisió al efectuar el gir.

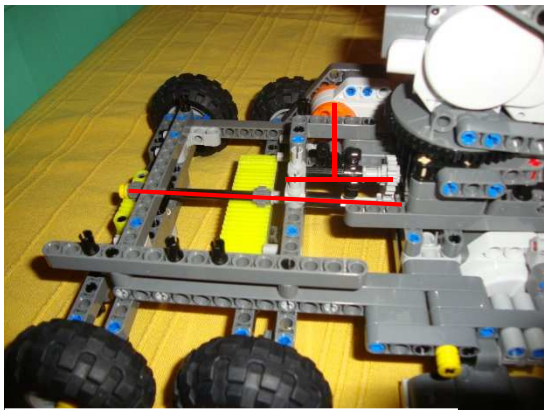


Fig 5.6. Eixos de la direcció.

Per a solucionar el problema vaig idear un sistema totalment nou variant la posició del motor, situant-lo al costat de la base de la grua sense que interferís en el moviment d'aquesta. El motor transmet el moviment rotatori a un eix paral·lel als eixos transversals, que al mateix temps el passa a un altre (situat al centre de l'estructura paral·lel a les rodes i perpendicular a les bigues transversals) en un angle de 90º. Aquest eix el transmet a un altre eix paral·lel situat a sota a l'altura de les bigues de direcció, i mitjançant una cremallera dissenyada amb SketchUp i un engranatge es transforma el moviment rotatori en rectilini.

Fig 5.7. Direcció amb reducció.

A l'hora de programar el robot i testejar-lo em vaig adonar que per les limitacions que tenien els materials en quant a precisió del sistema feia que el robot anés patint una petita desviació. Això no era perceptible quan girava però si quan anava recte, doncs les rodes no tornaven a l'angle inicial i es torçava bastant. Per això vaig decidir incloure una reducció mitjançant engranatges. En

la primera transmissió vaig substituir els engranatges negres (mostrats a la figura 5.6) per uns que oferien una reducció de 2:1. En els altres, els dels eixos paral·lels de color gris clar, van ser canviats per uns altres que oferien una reducció de 3:1. Això suma un total de 6:1 de reducció des del motor fins a la cremallera. Amb aquestes modificacions vaig aconseguir que les imprecisions es reduïssin en bona part, tot i que el sistema continua sent un pèl imprecís. Tot i així, aquestes petites imprecisions no impedeixen el bon funcionament del robot.

5.4. Grua motoritzada

El robot inclou una grua formada per tres motors, un rotor i un braç articulat amb una pinça. Un motor, situat just a sobre dels motors encarregats de la tracció és el que permet ala grua rotar sobre si mateixa però no en un angle de 360º, doncs els bricks NXT ho impedeixen. El segon és el que fa el treball necessari per a estendre el braç articulat. Aquest motor inicialment transmetia el moviment directament al braç, però al veure que no tenia la suficient força per aixecar l'objecte vaig haver d'introduir un sistema de reducció i modificar una mica la seva estructura. Aquesta reducció és de 4:1. Mitjançant un mecanisme de peces Lego la grua es desplega fins a copsar l'alçada del terra, on ha de recollir l'objecte. L'últim motor permet a la pinça fer el moviment d'obertura i tancament per poder recollir els objectes.

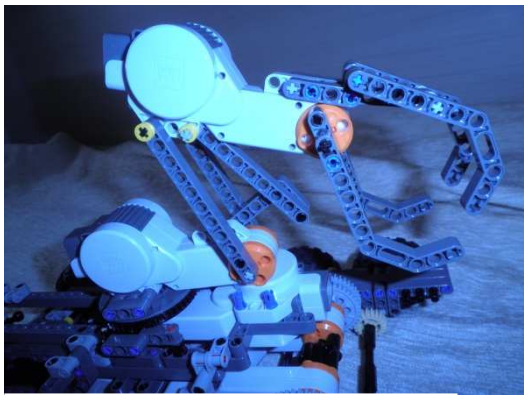


Fig 5.8. Grua motoritzada.

A la part superior de la pinça consta d'un sensor de color que analitza el color de l'objecte. Llavors la grua, segons la informació que rep d'aquest sensor classifica l'objecte dipositant-lo en el seu contenidor corresponent.

Fig 5.9. Grua amb mòdul de reducció.

5.5. Sensors

El robot consta de tres sensors: dos d'ells situats entre les rodes motrius (un a la part esquerra i l'altra a la dreta) són ultrasònics i permeten al robot veure l'objecte a recollir. L'altre és un sensor de color situat a sobre de la pinça de la grua i permet al robot classificar el que recull.

6. Disseny 3D

A l'hora de construir el robot, el kit inclou un nombre limitat de peces que poden tenir diverses utilitats segons com les agrupis i col·loquis. Depenent de les teves necessitats, pots utilitzar unes o altres. Gràcies a que el meu projecte consta de dos kits Lego Mindstorms NXT 2.0 no he tingut masses problemes en quant al nombre de peces per a fer l'estructura, si bé n'he tingut algun no ha sigut molt difícil trobar un altre mètode per construir una estructura que fos capaç de cobrir les necessitats que tenia (per exemple fer un suport per a un eix o fer la base de la grua).

Tot i això, en alguns moments he vist limitat el meu projecte perquè amb les peces que disposava no podia dur a terme la meva idea o simplement no podia crear un sistema que s'ajustés perfectament al que jo volia construir, doncs no feien la funció que havien de fer de manera precisa o s'acabaven desmuntant. Un clar exemple és el sistema de direcció, el qual en un principi era imprecís, molt forçat i no complia la seva funció correctament.

Aquestes imprecisions donades per les limitacions que tenia a l'hora de realitzar el meu projecte podien ser totalment decisives, doncs si el robot, per exemple, ha de col·locar-se a una certa distància de la pilota per a poder agafar-la i el sistema de gir no fa la seva funció de manera correcta, aquesta imprecisió provocaria que no fos capaç de recollir la pilota. Aquest error, si disposés d'un programa molt extens i complicat, podria ser solucionat, però està clar que és més fàcil solucionar el problema des de la part de l'estructura que des de la part del programari.

Per aquest motiu vaig decidir que el disseny de peces en 3D mitjançant l'impresora de l'escola seria la millor solució. Una altra solució possible era comprar les peces necessàries al proveïdor, però tot i que serien peces millor fabricades no m'oferien la llibertat de la que disposava amb l'impresora. També podia intentar fer-les jo manualment, però al ser elements tant petits (estem parlant de mides deumil·limètriques) era impossible fer una peça precisa, l'error era massa gran. Per tant, el fet de poder dissenyar jo mateix la peça ajustant-la exactament a les meves necessitats de manera precisa clarament era la millor de les solucions al meu abast.

Així doncs, amb els programes adequats vaig començar a dissenyar les peces que em calien a mesura que anava donant-li forma al robot.

6.1. Programes per a l'impresió de peces 3D

Els programes que he utilitzat són el SketchUp 2015, el Netbaff Studio i el Cura 14.09. Amb el SketchUp dissenyava la peça, a continuació l'exportava a STL (un format d'arxiu), obria aquest arxiu amb el Netbaff per veure i reparar els errors i per últim l'obria amb el cura per a imprimir-lo.

6.1.1. SketchUp 2015

El SketchUp és un dels programes de modelatge de la mà de la companyia Trimble en tres dimensions més utilitzat per arquitectes, enginyers i professionals del disseny i la construcció en general, però també és molt utilitzat pels entusiastes de l'impresió en impressora 3D .

Aquesta versió publicada a finals de 2014 ha introduït certes millores que fan del SketchUp un programa més intuïtiu, veloç, senzill i fiable per als usuaris tant professionals com aficionats. Una millora destacable és l'eina d'arc mitjançant tres punts molt efectiva per a realitzar trossos de circumferència amb precisió, la qual he utilitzat en els meus dissenys.

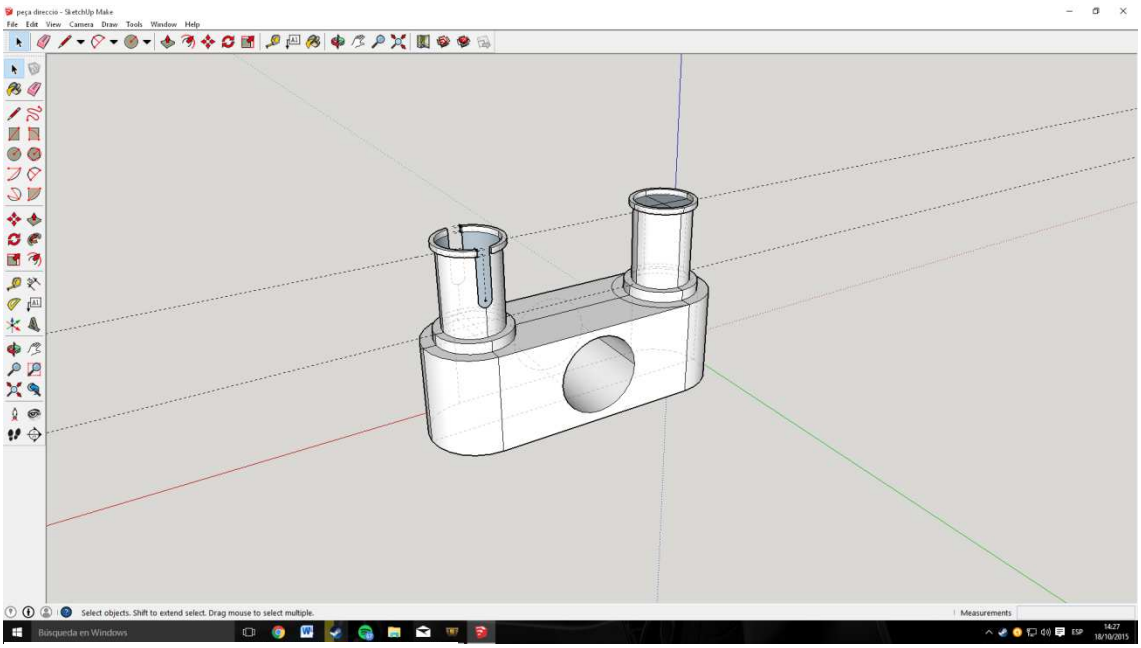


Fig 6.1. Intefície del SketchUp 2015.

6.1.2. Netfabb Studio Basic

El Netfabb Studio Basic és un programa gratuït disponible per a Mac, Windows i Linux de la mà de l'empresa Netfabb. La funció principal per a la qual ha estat elaborat aquest programa és la d'analitzar els dissenys 3D en busca d'errors i reparar-los de manera automàtica. La seva versió Netfabb Studio Professional ens ofereix moltes més eines i millors reparacions, però no és gratuïta. Cal tenir en compte que el Basic no és una "demo" del Professional, i ambdós treballen amb el mateix tipus d'arxiu, l'arxiu STL.

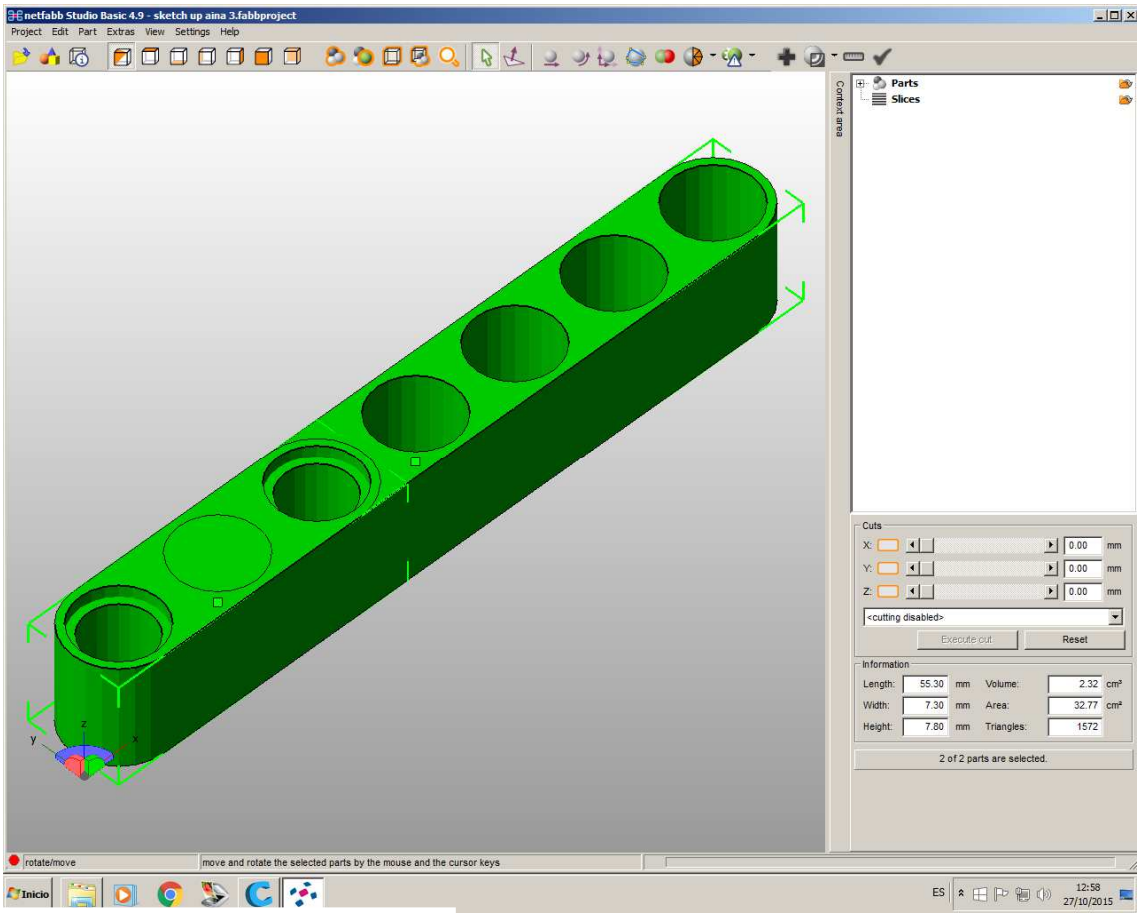


Fig 6.2. Interfície del Netfabb Studio.

6.1.3. Cura 3D

Cura 3D és un programa de l'empresa Ultimaker que ens permet transformar els nostres arxius STL en peces reals. Antigament s'anomenava SkeinPyPy, doncs és fruit de la combinació de dos programes anomenats Skeinforge i PyPy. Skeinforge és un programa que divideix els nostres dissenys en capes per a que la impressora els pugui llegir i imprimir. Aquest programa està fet en Phyton (un llenguatge de programació), de manera que el van combinar amb el PyPy que accelerava el procés de divisió en capes i reduïa en gran mesura el temps de processament (era de més de una hora, depenent de la complexitat del disseny).

Tot i això, la companyia no estava satisfeta amb el resultat per que era un programa amb una interfície bastant dolenta i amb moltes opcions difícils de configurar i que tampoc influïen gaire a l'hora d'imprimir. Per això van elaborar el que ara es coneix com a CURA, un programa basat en aquest dos abans mencionats però amb una interfície i una configuració molt més senzilles.

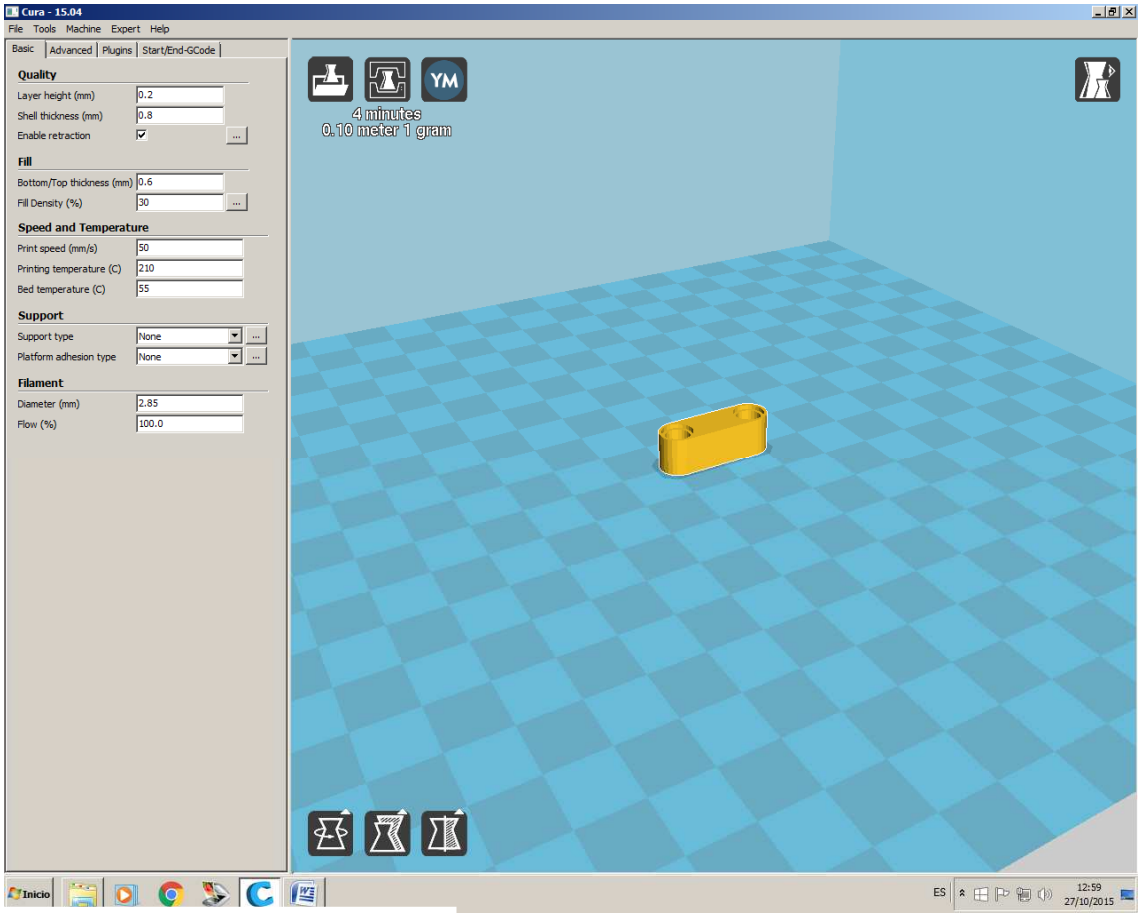


Fig 6.3. Interfície del Cura 3D.

6.2. Peces dissenyades

Durant l'elaboració del robot m'he anat trobant amb situacions que requerien un tipus de peça específica per fer una funció determinada. De totes maneres he elaborat diferents peces per a una mateixa funció, sempre intentant millorar els error de la peça dissenyada precedent fins a aconseguir el resultat desitjat. Per a dissenyar-les, mitjançant un peu de rei, he mesurat les mesures necessàries (forats, amplada, llargària, alçada...) i també he extret informació d'alguna pàgina web on es mostraven les mesures per defecte (per exemple el diàmetre del forat.

A continuació hi han descrites les peces que he anat elaborant al llarg del projecte, quina és la funció que duen a terme i on estan col·locades.

Biga de 5 forats: Inicialment vaig dissenyar una peça simple imitant la de Lego per a comprovar quin era l'error de l'impresora i disminuir-lo. És una peça de mides 39'3x7'3 mil·límetres de base i 7'8 d'alçada. Consta de 5 forats de diàmetre 4'8 mm amb una obertura a la part inferior de 6'2 mm de diàmetre i 0'9 mm de fondària.

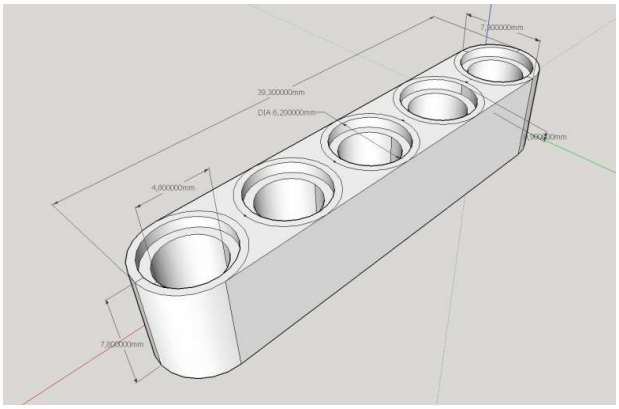


Fig 6.4. Peça de 5 forats inicial.

Més tard, al veure que l'error de l'impresora era massa gran i la peça no encaixava amb les originals vaig decidir eliminar la circumferència interior. Tot i això, la peça seguia sense encaixar, llavors vaig decidir fe una circumferència interior de 5 mm de diàmetre en lloc de quatre. Al veure que la solució del problema no estava en modificar les mides perquè l'impresora feia un error massa gran i mai aconseguia la peça adequada vaig decidir repassar el forat amb una broca del 5 i va encaixar.

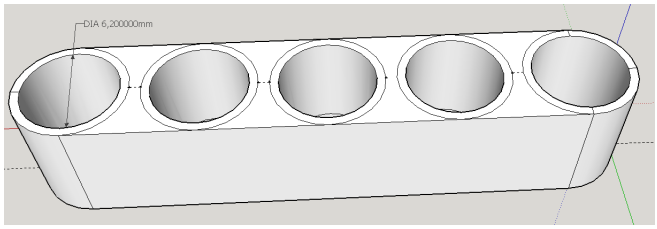


Fig 6.5. Peça 5 forats sense circumferència interior

Fig .6. Peça d'unió lego.

Cremallera: Amb el problema de la direcció del cargol sense fi[x], vaig haver de dissenyar una cremallera per a poder transformar el moviment rotatori en rectilini. Primerament vaig intentar un disseny que bàsicament era una plataforma dentada amb dos cilindres que imiten a les peces d'unió lego, però l'impresora no podia imprimir una cosa tant petita i va imprimir un cilindre directament. A més, no encaixava amb la peça original, així que vaig intentar dissenyar una peça sense aquest cilindres. A part d'això, les dents eren massa petites i no engranaven bé amb l'engranatge Lego.

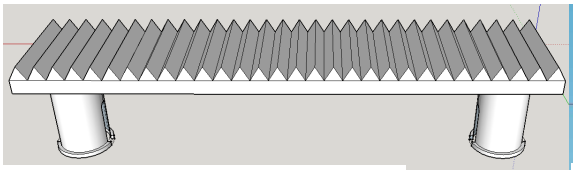


Fig 6.7. Cremallera 1, part superior.

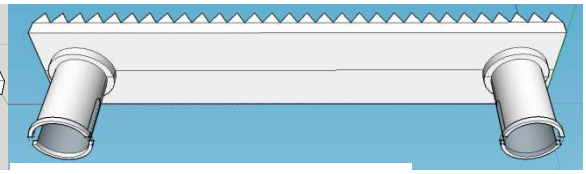


Fig 6.8. Cremallera 1, part inferior.

Per tal de millorar la cremallera anterior (ja que no em solucionava el problema de la direcció), vaig elaborar unes dents noves amb una forma no triangular, sinó que vaig posar una petita base de 0'75 mm d'alçada i un trapezi a sobre de 1'5 mm d'alçada. En conjunt, amb la base de tota la peça, sumava una alçada de 3 mm. Vaig aprofitar la biga de 5 forats que ja tenia dissenyada com a suport per a unir la cremallera al robot. Donat que la impressora no tenia la capacitat de fer aquesta tasca per motius de precisió, vaig haver d'imprimir les peces per separat i unir-les amb cola instantània.

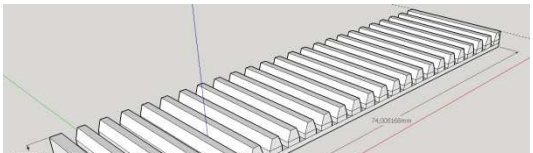


Fig 6.9. Segona cremallera.

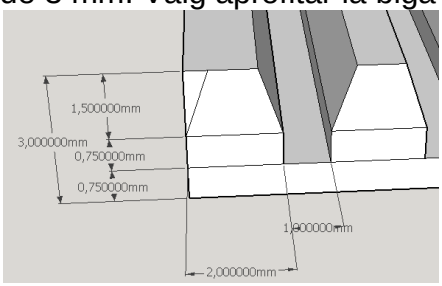


Fig 6.10. Mides de les dents.

Aquest segon model no va ser efectiu perquè donat l'alçada de les dents, l'eix anava molt forçat i saltava, fent que es perdés el contacte entre l'engranatge i la cremallera i per tant fent que no es conservés el moviment.

Amb l'objectiu de solucionar aquest nou inconvenient, vaig dissenyar una peça on la cremallera hi estigués integrada, de manera que no sobrepassaria l'altura i l'eix no saltaria.

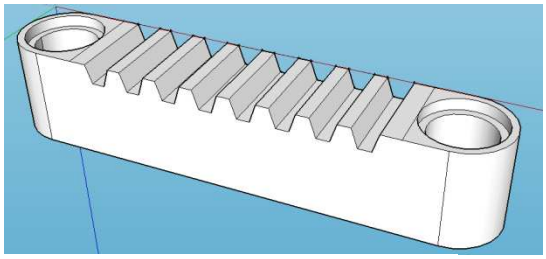


Fig 6.11. Cremallera integrada.

En un principi era de 5 forats, però més tard vaig canviar l'estructura de les bigues de direcció i la vaig fer de 7 per a que tingués més recorregut. A més, vaig ampliar l'àrea de la cremallera perquè al moure's l'eix mòbil s'apropava al fixe, de manera que vaig haver de allargar una mica l'amplada de la peça.

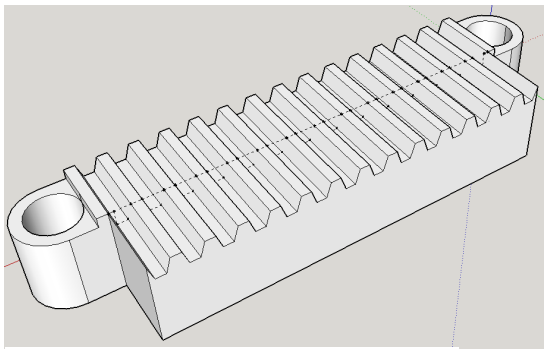


Fig 6.12. Cremallera final.

Caixa: Per al disseny de l'estructura que permetria al robot emmagatzemar i transportar les pilotes recollides vaig fer primerament una estructura amb peces Lego utilitzant les que m'oferia el kit (Figura .13). Vaig procurar que fos el més estable possible i el suficientment gran com per emmagatzemar fins a 6 pilotes (les que disposava). El separador simplement fa la funció de evitar que es barregin les pilotes de colors. D'aquesta manera es veu clarament com el robot les classifica. A continuació, tenint en compte diverses mesures i amb l'ajut d'una pàgina web on es mostraven les mides de les peces que jo vaig utilitzar, vaig dissenyar la caixa en cinc parts diferents: la part frontal (Figura .14), amb dos orificis per a encaixar el separador; la part darrera (Figura .15), amb un petit detall de decoració a mode de matrícula; el lateral esquerra (Figura .16), on hi ha gravat la paraula "RED", indicant el color de les pilotes que s'ubiquen en aquell compartiment; el lateral dret (Figura .17), amb un gravat que posa "BLUE" amb la mateixa funció que l'anterior i finalment el separador (Figura .18).

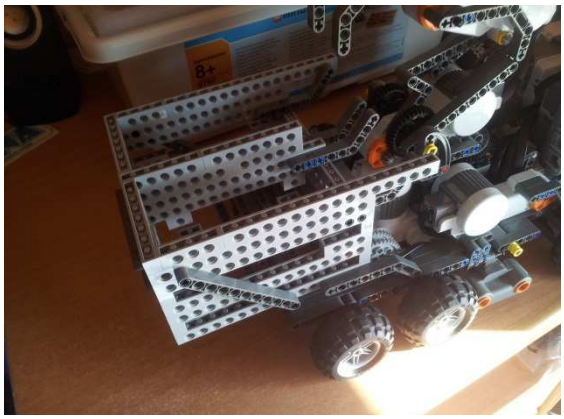


Fig 6.13. Caixa Lego.

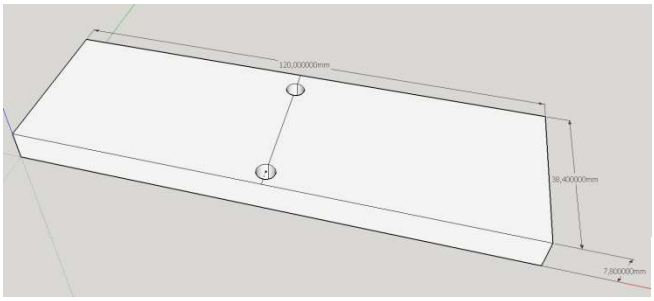


Fig 6.14. Part Frontal.

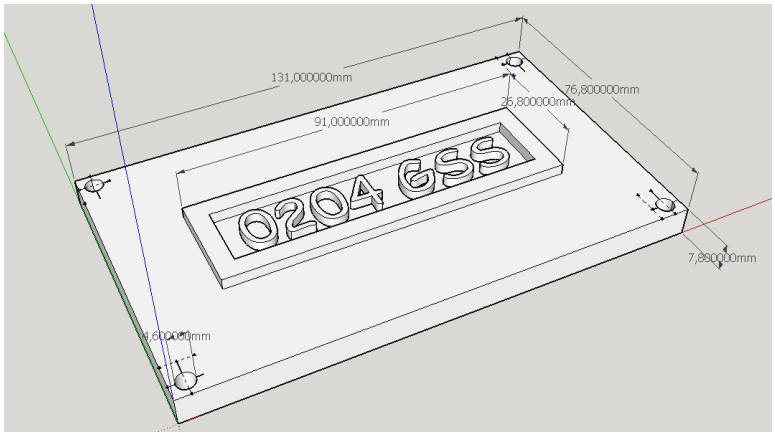


Fig 6.15. Part darrera.

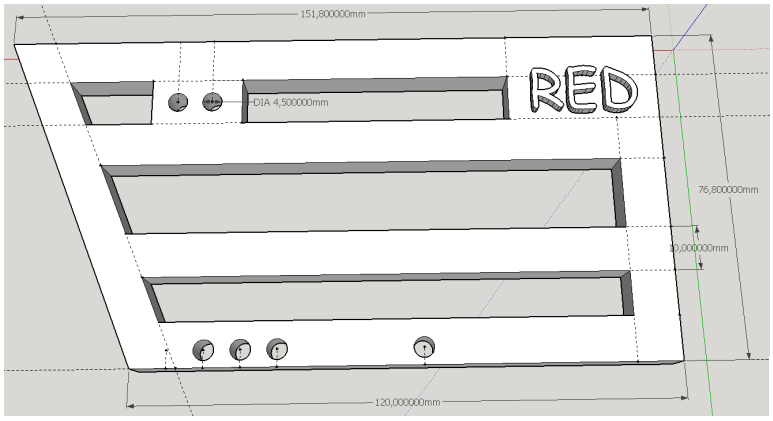


Fig 6.16. Lateral Esquerra.

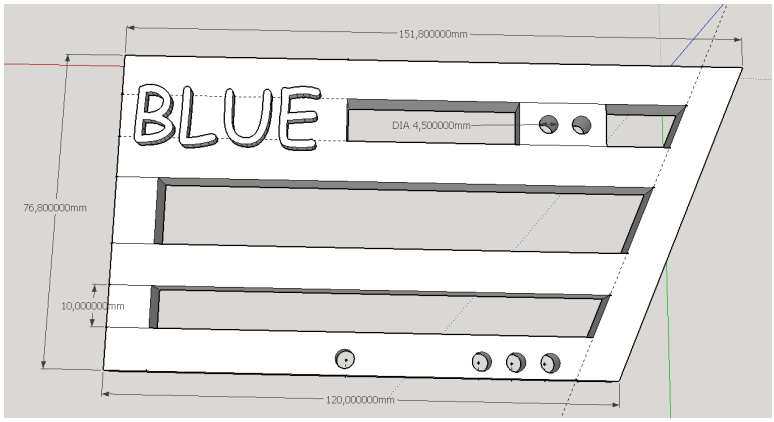


Fig 6.17. Lateral dret.

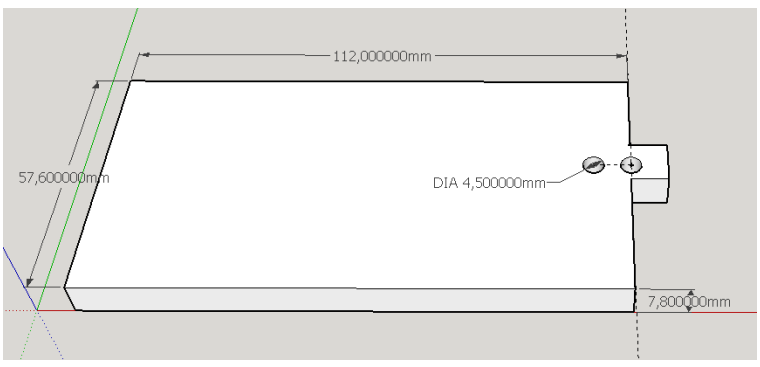


Fig 6.18. Separador.

7. Programació del robot

El plantejament del funcionament del meu robot inclou dos aspectes diferents: el control remot i l'automatisme. L'automatisme en quant a dificultat de codi o limitacions d'aquest no em suposava cap problema. En canvi, el control remot ja limitava les meves possibilitats d'elecció. A més, al tractar-se d'una aplicació Android (el qual funciona amb java), em vaig veure obligat a fer el programa mitjançant aquest llenguatge. Una possibilitat de la que també disposava era fer el comandament a distància mitjançant el tercer brick NXT i poder programar en llenguatge C (del qual ja tenia coneixements previs), però vaig decidir assumir el repte d'aprendre aquest llenguatge nou per a mi, doncs em feia molta curiositat i em feia il·lusió elaborar una aplicació per a mòbil.

7.1. El llenguate Java

Primerament, per a poder compendre millor l'estructura del codi que he elaborat, trobarem l'explicació de manera resumida sense entrar en masses detalls dels principals conceptes en els quals es basa Java i sobretot centrant-nos en el que apareix d'una forma directa al codi que he escrit.

Per entendre el funcionament de Java hem de tenir clars certs conceptes. Els principals són paquet, classe, objecte i mètode:

- Un objecte és un element o entitat existent dins la memòria del ordinador que té una sèrie de propietats o atributs sobre sí mateix i unes operacions disponibles segons la classe on es trobi.
- Una classe és un concepte abstracte que determina quines propietats i operacions disponibles tindrà un objecte.
- Un mètode és un conjunt d'instruccions definides per una classe que realitzen una determinada tasca. Generalment hi ha de dos tipus: el mètodes que executen processos segons unes variables (executen tasques) i els mètodes que realitzen càlculs (calculen variables).
- Un paquet és una manera d'organitzar un grup de classes semblants ja sigui per àmbit en el que operen o per finalitat. Eviten el conflicte entre els noms de les classes (que no tinguin dues classes el mateix nom).

Per a entendre millor aquest conceptes fonamentals sobre el funcionament del llenguatge Java ho exemplificarem per simplificar-ho i fer-ho més entenedor establint equivalències amb situacions a la vida real.

Imaginem, per exemple, que el nostre objecte és una casa. Aquesta casa té una sèrie de propietats pròpies de les cases, com el nombre d'habitacions, l'adreça en la que es troba, el nombre de plantes de les que disposa, la ciutat on es troba... i també podem realitzar una sèrie d'operacions en ella com construir-la, derruir-la, restaurar-la, ampliar-la, canviar-la de carrer, pintar-la...

En canvi, si tenim un pis, no podem determinar la propietat del nombre de plantes, ni el podem derruir (doncs forma part d'una estructura major, l'edifici). Tot i així, tenim altres propietats pròpies dels pisos com nombre de planta i porta (baixos primera per exemple).

D'aquesta manera podríem dir que la classe "vivenda" determinaria les propietats dels objectes "pis" i "casa" i determinaria quines són les operacions disponibles a realitzar sobre aquests objectes, és a dir, quins mètodes els hi podem aplicar. Al seu torn, els objectes porten inclosos una sèrie de propietats sobre si mateixos que venen definides a la classe, però no tots disposen de totes les propietats i operacions que determina la classe, sinó que tenen certs matisos (si no serien el mateix objecte).

Al mateix temps la classe vivenda està classificada dins d'un paquet amb un grup de classes que mantenir certa relació per semblança. Per exemple, en el paquet "edificacions" podríem incloure les classes vivenda, zona esportiva, centre comercial, serveis públics, serveis privats... D'aquesta manera s'engloben les classes en grups diferents.

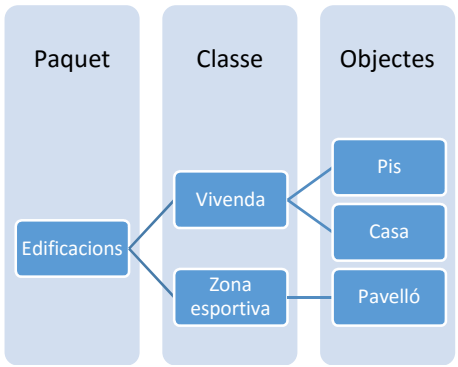


Fig 7.1. Esquema de les bases de Java.

Ara que ja coneixem aquest conceptes bàsics em disposo a explicar algunes de les sentències i els tipus de variables que he utilitzat per programar les decisions del robot i de l'aplicació.

Tipus de variables:

- Booleans(*Boolean*): Són variables que funcionen com el sistema digital, és a dir, tenen dues posicions: true i false.
- Enters(*int*): Són variables formades per nombres enters (-n, ... , -3, -2, -1, 0, 1, 2, 3, ... , n).
- Dobles(*double*): Són variables amb nombres decimals.

Tipus de sentències:

- If/else: La instrucció if / else permet controlar quins processos tenen lloc típicament en funció del valor d'una o diverses variables, d'un valor de càlcul o booleà, o de les decisions de l'usuari. La seva sintaxi és la següent:

```
if (condició){
      Instruccions
}
else{
      Instruccions
}
```

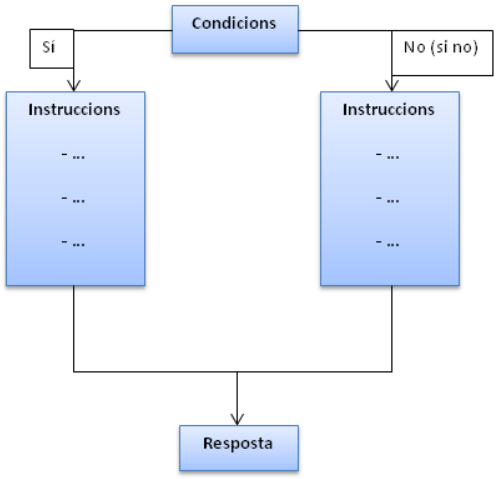


Fig 7.2. Esquema if / else.

En el cas de Java, la clàusula else admet no contenir res, de manera que contemplem el cas però no realitzem res al respecte. També podem establir una cascada mitjançant l'else if, que significa "si no i si". Així podem definir les instruccions a seguir si tenim més d'un cas segons certes condicions.

```
if (condició){
      Instruccions
}
else if(condició){
      Instruccions
}
else{}
```

- While: Aquesta sentència determina que es realitzarà quelcom mentre una condició sigui certa. Aquesta condició pot ser qualsevol tipus de variable, però usualment es fan servir booleans per ser de tipus cert/fals. La seva sintaxi és:

```
while(condició){
      instruccions
}
```

- Switch/case: La sentència switch és equivalent a l'if/else però és útil a l'hora de clarificar el codi i fer-lo més comprensible. La diferència és que només pot assimilar valors enters o expressions que es puguin traduir a valors enters. La seva sintaxi és la següent:

```
switch (expressió){
      case valor1:
            Instruccions
      break;
      case valor2:
            Instruccions
      break;
      case valor3:
            Instruccions
      break;
      .
      .
      .
      .
      case valorN:
            Instruccions
      break;
}
```

- try/catch: És una sentència que té com a funció tractar els errors que es puguin produir al codi durant la seva execució, no del tipus sintàctic (compilació). Hi ha dues maneres de tractar els errors al llenguatge Java. Una d'elles seria intentar capturar l'error i l'altre seria intentar expulsar-lo. Aquesta sentència forma part de les que intenten capturar l'error.

Està dividida en dos blocs, el try i el catch. Dins el try (que vol dir intentar) s'introdueix el codi sobre el que es pot produir un error i es captura en cas que succeeixi, és a dir, s'intenta capturar l'error. Al bloc catch definim el conjunt d'instruccions necessàries per tractar l'error identificat al bloc catch. El bloc catch crea un objecte de tipus Exception. Aquest objecte conté l'informació de l'error detectat al bloc try i és enviat al bloc catch. La seva sintaxi és:

```
try{
      Bloc de codi susceptible de generar un error
}
catch(Exception e){
      Bloc de codi que tracta l'error
}
```

7.2. Programació de l'aplicació Android

7.2.1. Android Studio

Per a programar l'aplicació que funciona de control remot per controlar el robot he utilitzat el programa Android Studio. Aquest programa, elaborat per l'empresa Google, és gratuït i va ser publicat al desembre de 2014 (és un programa bastant nou). Està basat en el software IntelliJ IDEA de JetBrains. Les seves característiques principals són:

- Renderització en temps real
- Consola de desenvolupador: consells d'optimització, ajuda per a la traducció, estadístiques d'ús.
- Suport per construcció basada en Gradle.
- Refactorització especifica d'Android i arranjaments ràpids.
- Eines Lint per detectar problemes de rendiment, usabilitat, compatibilitat de versions, i altres problemes.
- Plantilles per crear dissenys comuns d'Android i altres components.
- Suport per a programar aplicacions per Android Wear.

El codi que determina el comportament de l'aplicació és una barreja entre una plantilla trobada a internet sobre la comunicació Bluetooth i petites tasques estàndar com la de tancar l'aplicació mitjançant el botó "enrere" de qualsevol Smartphone amb el codi que he escrit.

La lògica que he seguit per elaborar aquesta aplicació és la següent: a la pantalla es mostren una sèrie de botons que determinen les accions a seguir pel robot. Quan modifiquem l'estat de qualsevol d'aquests controladors (premem un botó per exemple) s'envia un missatge en forma de byte cap al brick NXT. L'aplicació no ordena que es moguin els motors o que s'activin els sensors, sinó que el que fa és enviar un missatge en forma de byte.

D'aquesta manera el que determina el codi és quin és el missatge a enviar segons quin controlador i de quina manera s'hagi modificat. Així doncs procedeixo a explicar el codi (complet al annex) en profunditat un cop ja explicat quina és la seva base conceptual.

El codi està dividit en dues tasques. Una d'elles, l'extreta d'internet és la tasca BTComm que vol dir Bluetooth Comunication. Com el seu nom indica, és la tasca que estableix la comunicació entre el mòbil i els dos NXT. El mòbil funciona com a màster i els brick NXT com a esclaus. Això vol dir que els brics NXT estan a l'espera de les ordres del mòbil, que és el que mana. L'altre tasca és l'anomenada MainActivity, que determina quin byte correspon a cada modificació dels controladors de la pantalla.

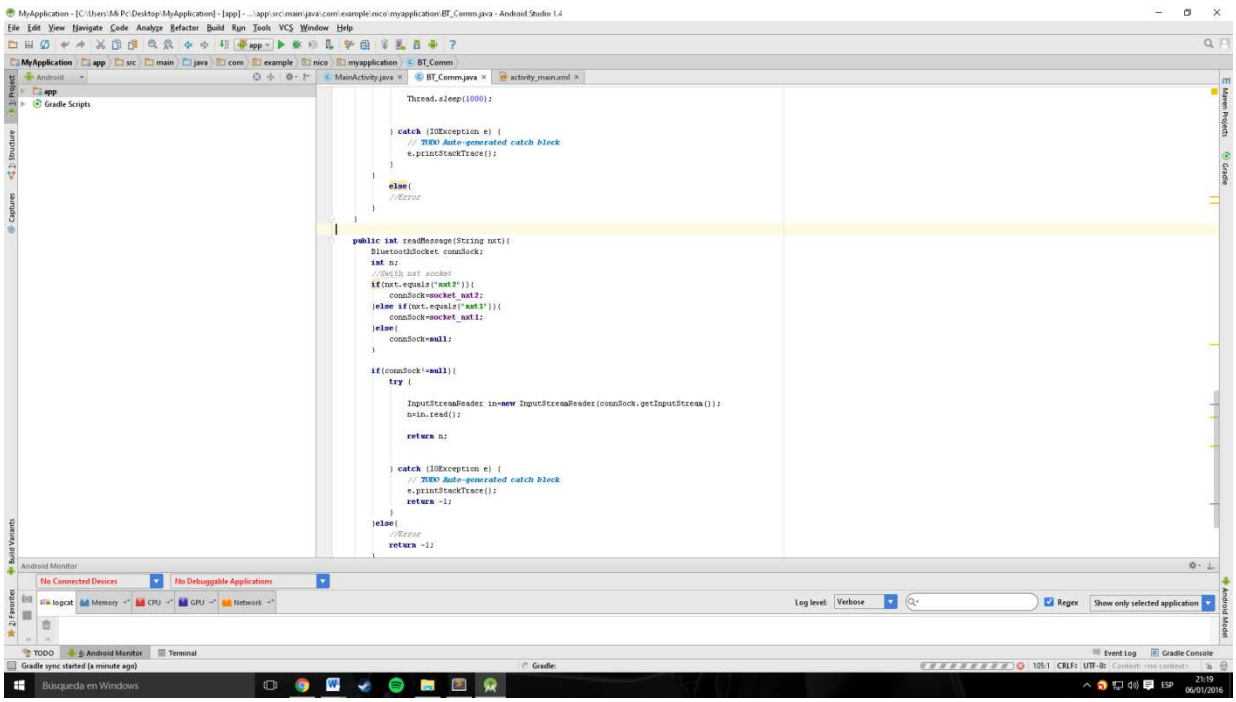


Fig 7.3. Interfície Android Studio.

7.2.2. Disseny gràfic de l'app

En aquest apartat veurem la part gràfica de l'aplicació que es mostra en pantalla tot explicant la disposició dels botons. La col·locació dels botons ha sigut realitzada mitjançant les eines que ofereix el programa. La pantalla està dividida en graelles i a cada casella col·loquem el botó que ens interessi amb la mida que vulguem.

L'aplicació té tres botons: dos situats a la part esquerra un a sobre l'altre anomenats FWD i BWD que controlen el moviment del robot i un tercer a la dreta anomenat CATCH que acciona la grua.

Consta de dues barres de progrés: una entre els botons FWD i BWD que controla el gir del robot i una altra situada a sota del botó CATCH que controla la velocitat del robot.

Per últim, situats a sobre del botó CATCH hi ha dos quadres de text. El superior mostra l'estat de moviment en el qual es troba el robot i l'inferior mostra quin és el moviment que hem de fer per agafar la pilota.

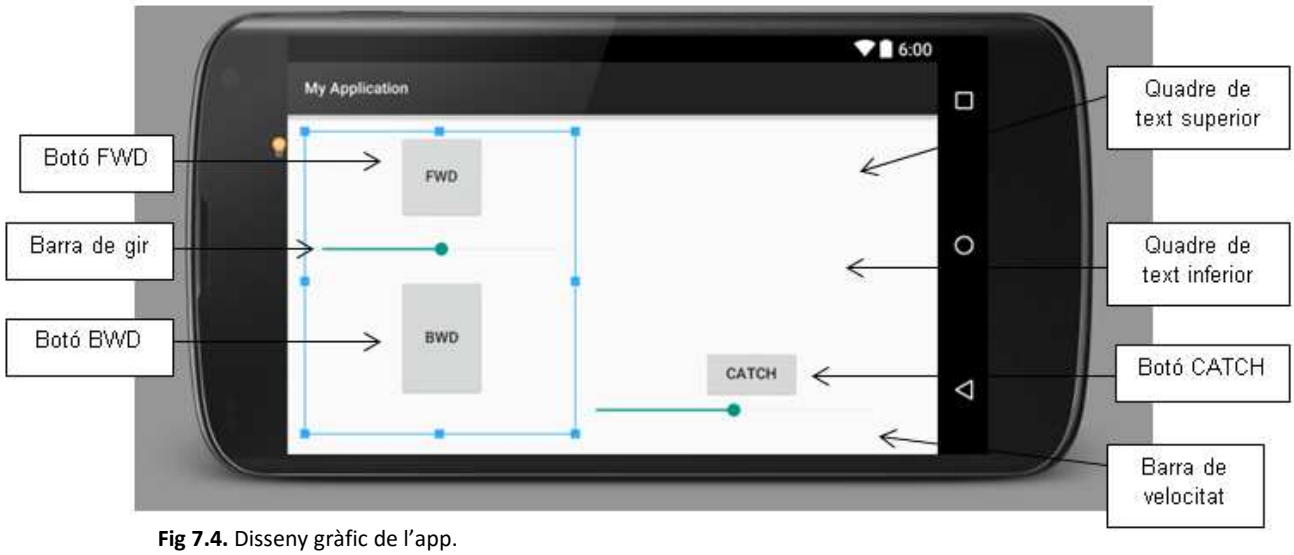


Fig 7.4. Disseny gràfic de l'app.

7.2.3. Tasca BT_Comm

Com en qualsevol codi escrit en Java comencem amb les importacions dels paquets. Aquests determinen quines classes seran utilitzades al programa.

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.UUID;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.util.Log;
```

Tot seguit tenim la classe principal anomenada BT_Comm on es troba escrit tot el comportament de l'aplicació. Per a una millor comprensió del codi està estructurat en petites tasques. La primera d'elles és la d'activar el bluetooth. El que realitza aquesta tasca és activar el Bluetooth en cas que no estigui activat. Creem l'objecte localAdapter i determinem que si no està activat llavors s'activi i mentre estigui activat no realitzi res.

```
//Enables Bluetooth if not enabled
    public void enableBT(){
        localAdapter=BluetoothAdapter.getDefaultAdapter();
        //If Bluetooth not enable then do it
        if(localAdapter.isEnabled()==false){
            localAdapter.enable();
            while(!(localAdapter.isEnabled())){
            }
        }
    }
```

A continuació trobem la tasca que connecta ambdós NXT amb el mòbil anomenada connectToNXTs. Creem els dos objectes "nxt_2" i "nxt_1". Amb la funció "try" apliquem el mètode que fa que els dos objectes es connectin per bluetooth al mòbil. Si la connexió és exitosa el boleà "success" que inicialment està en false passa a ser true. Si no és exitosa, el boleà segueix sent false i s'identifica el tipus d'error perquè l'aplicació no es colapsi.

```
    public  boolean connectToNXTs(){
        //get the BluetoothDevice of the NXT
        BluetoothDevice nxt_2 = localAdapter.getRemoteDevice(nxt2);
        BluetoothDevice nxt_1 = localAdapter.getRemoteDevice(nxt1);
```

```
        //try to connect to the nxt
        try {
            socket_nxt2 = nxt_2.createRfcommSocketToServiceRecord(UUID
                    .fromString("00001101-0000-1000-8000-
00805F9B34FB"));
            socket_nxt1 = nxt_1.createRfcommSocketToServiceRecord(UUID
                   .fromString("00001101-0000-1000-8000-
00805F9B34FB"));
            socket_nxt2.connect();
            socket_nxt1.connect();
            success = true;
        }
        catch (IOException e) {
            Log.d("Bluetooth","Err: Device not found or cannot
connect");
            success=false;
        }
        return success;
    }
```

La tasca "writeMessage" és la que determina com s'envien els missatges. El primer que realitza és mitjançant la funció "if"-"else" mirar a quin dels dos NXT ha d'enviar el missatge. Si no s'ha d'enviar a cap NXT la connexió és nul·la, de manera que no s'envia. Si la connexió no és nula, mitjançant la funció "try" s'envia el missatge al NXT corresponent. Declarem l'objecte "out" i amb els mètodes "write (msg)" i el "flush" enviem el missatge.

```
public void writeMessage(byte msg, String nxt) throws
InterruptedException{
        BluetoothSocket connSock;
        //Swith nxt socket
        if(nxt.equals("nxt 2")){
            connSock=socket_nxt2;
        }else if(nxt.equals("nxt 1")){
            connSock=socket_nxt1;
        }else{
            connSock=null;
        }
        if(connSock!=null){
            try {
```

```
   OutputStreamWriter out=new
OutputStreamWriter(connSock.getOutputStream());
                out.write(msg);
                out.flush();
                Thread.sleep(1000);
            } catch (IOException e) {
                    e.printStackTrace();
            }
        }
            else{
            //Error
        }
    }
```

Per últim tenim la tasca readMessage, la que serveix per a poder llegir els missatges enviats pels NXT. Declarem com a variable entera la "n" que serà el missatge que rebem del NXT. Comprobem les connexions de la mateixa manera que la anterior per saber quin NXT és el que ens envía el missatge. Declarem l'objecte "in" i amb el mètode "read" ens retorna la "n", és a dir, el missatge enviat pel NXT. Si hi ha un error el missatge rebut és -1.

```
public int readMessage(String nxt){
    BluetoothSocket connSock;
    int n;
    //Swith nxt socket
    if(nxt.equals("nxt2")){
        connSock=socket_nxt2;
    }else if(nxt.equals("nxt1")){
        connSock=socket_nxt1;
    }else{
        connSock=null;
    }

    if(connSock!=null){
        try {
       InputStreamReader in=new
InputStreamReader(connSock.getInputStream());
            n=in.read();

            return n;


        } catch (IOException e) {
                e.printStackTrace();
                return -1;
        }
    }else{
        //Error
        return -1;
```

```
    }
}
```

7.2.4. Tasca MainActivity

Primerament tenim les importacions. En aquest cas importem la clase que hem creat anteriorment anomenada BT_Comm.

```
import android.app.Activity;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;
import com.example.nico.myapplication.BT_Comm;
import java.io.IOException;
```

Com en l'anterior tenim la clase principal on s'engloben totes les altres tasques. La primera tasca (onCreate) forma part de la plantilla d'internet i la seva funció és activar les tasques de la clase BT_Comm, com engegar el bluetooth, conectar els NXT, etc. Després definim els boleans "isFwd" i "isBwd" com a falsos, és a dir, que el robot roman quiet. Creem els objectes "yourSeekBar" i "yourSeekBar1" i definim quina és la tasca que determina el funcionament de la barra de progrés.

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  btComm= new BT_Comm();
  btComm.enableBT();
  btComm.connectToNXTs();

  isFwd=false;
  isBwd=false;

  SeekBar yourSeekBar=(SeekBar) findViewById(R.id.barragir);
  yourSeekBar.setOnSeekBarChangeListener(new controladorgir());
  SeekBar yourSeekBar1=(SeekBar)
```

```
findViewById(R.id.barravelocitat);
        yourSeekBar1.setOnSeekBarChangeListener(new
controladorvelocitat());
    }
```

A continuació trobem una tasca que es crea automàticament quan creem un nou projecte d'Android Studio. Serveix per a posar botons i opcions a la barra superior.

```
    public boolean onCreateOptionsMenu(Menu menu) {
            getMenuInflater().inflate(R.menu.menu_main, menu);
            return true;
        }
```

Tot seguit tenim la tasca anomenada "moviment" que determina el funcionament dels botons que controlen la part motriu del robot, és a dir, que permeten el seu desplaçament. Creem la variable entera "id" que ens ajudarà a determinar quin botó ha sigut premut mitjançant el seu nombre d'identificació. Inicialment determinem que el byte enviat és -1. Creem els objectes "text", "botobwd" i "botofwd" i definim els botons als quals fan referència.

A continuació, amb la funció "if" determinem com funcionarà el botó "fwd". Si premem el botó "fwd" i el robot està retrocedint, el botó farà la funció de stop, apareixerà un text que marqui l'estat al que passa a estar el robot després d'accionar el botó i apareixerà un text al propi botó que posa FWD. En canvi, si el robot estava quiet al moment de pitjar el botó "fwd", llavors avançarà i apareixerà un text que marqui que està avançant. Al seu torn, al botó bwd apareixerà un text que posa STOP.

Amb la mateixa metodologia funciona el botó "bwd". De manera conceptual, el que fa aquesta tasca és el següent: nosaltres per frenar el robot hem de prémer la tecla que el faci anar en la direcció contrària a la qual està anant. Quan estigui quiet, les tecles realitzen la seva funció normal. En canvi, quan estigui en moviment la tecla oposada farà la funció d'aturar-se. Tot això ho controlem mitjançant els booleans "isFwd" i "isBwd" de manera que puguem saber en quin estat es troba el robot.

Finalment ordenem amb la funció "try" que s'enviï el missatge corresponent, ja sigui avançar, retrocedir o aturar-se.

```
public void moviment(View view) {
  int id = view.getId();
  Byte msg=-1;
  TextView text = (TextView) findViewById(R.id.textIt);
  Button botobwd = (Button)findViewById(R.id.bwd);
  Button botofwd = (Button)findViewById(R.id.fwd);

  if (id == R.id.fwd) {
      //si premem Fwd i està retrocedint, es para.
      if(isBwd){
          msg=2;
          isBwd=false;
          text.setText("Stop");
          botofwd.setText("Fwd");
      }
      //si premem Fwd i està quiet, avança.
      else if (!isFwd){
          msg=1;
          isFwd=true;
          text.setText("Fwd");
          botobwd.setText("Stop");
      }
  }
```

Botó fwd

```
  if (id == R.id.bwd) {
      //si premem Bwd i esta avançant, es para.
      if(isFwd){
          msg=2;
          isFwd=false;
          text.setText("Stop");
          botobwd.setText("Bwd");
      }
      //si premem Bwd i esta quiet, retrocedeix.
      else if(!isBwd){
          msg=3;
          isBwd=true;
          text.setText("Bwd");
          botofwd.setText("Stop");
      }
  }
```

Botó bwd

```
        try {
            btComm.writeMessage(msg, "nxt 2");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

Tot seguit tenim una altra tasca creada automàticament pel programa que determina el funcionament de les barres d'acció, quelcom estándar al sistema Android.

```
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
```

Després tenim la classe "controladorgir" amb la qual determinem com controlar el gir del robot. El gir el controlem mitjançant una barra de progrés. Segons la posició de la barra, l'angle de gir serà un o altre. La barra té posicions diferents. Inicialment el byte és -1 i està a la posició 2, és a dir, al mig (va del 0 al 4). Amb la funció "switch-case" determinem quin byte s'enviarà segons la posició on es trobi la barra, tot definint la posició de la barra com un nombre enter comprès entre 0 i 4 inclosos. Amb la funció "try-catch" enviem el byte corresponent.

```
    public class controladorgir implements
    SeekBar.OnSeekBarChangeListener {

            public void onProgressChanged(SeekBar barragir, int
    posicio, boolean fromUser) {
                Byte msg = -1;

                switch (posicio) {
```

```
                case 0:
                    msg = 4;
                    break;
                case 1:
                    msg = 5;
                    break;
                case 2:
                    msg = 6;
                    break;
                case 3:
                    msg = 7;
                    break;
                case 4:
                    msg = 8;
                    break;
            }

            try {
                btComm.writeMessage(msg, "nxt 2");

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        public void onStartTrackingTouch(SeekBar seekBar) {}
        public void onStopTrackingTouch(SeekBar seekBar) {}
    }
```

A continuació hi ha la classe "controladorvelocitat" amb la qual podrem variar la velocitat del robot. Funciona de la mateixa manera que la barra de gir, la diferència és que els missatges enviats són diferents.

```
    public class controladorvelocitat implements
SeekBar.OnSeekBarChangeListener {

     public void onProgressChanged(SeekBar barravelocitat, int
posicio, boolean fromUser) {
        Byte msg = -1;

        switch (posicio) {
            case 0:
```

```
                msg = 9;
                break;
            case 1:
                msg = 10;
                break;
            case 2:
                msg = 11;
                break;
            case 3:
                msg = 12;
                break;
            case 4:
                msg = 13;
                break;
        }
        try {
            btComm.writeMessage(msg, "nxt 2");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    public void onStartTrackingTouch(SeekBar seekBar) {}
    public void onStopTrackingTouch(SeekBar seekBar) {}
}
```

Finalitzant ja el programa trobem la tasca anomenada "grua" que controla el funcionament de la grua del robot. Aquest procés, al ser un automatisme l'únic que hem d'ordenar és d'iniciar-lo i esperar resposta. Per això és molt curta.

```
public void grua(View view){
        try {
            btComm.writeMessage((byte) 14, "nxt 1");
            escoltarnxt();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

Esperem resposta mitjançant la tasca "escoltarnxt". Inicialment el missatge rebut és -1. Mentre el missatge sigui -1, el robot es manté a l'espera

de resposta. Creem l'objecte "text" i mitjançant la funció "if" determinem quin text apareixerà en cada cas segons el missatge que rebem del NXT. Hi ha tres opcions: "moure dreta", "moure esquerra" o "posició correcta".

```
public void escoltarnxt(){
        int n=-1;
        while (n==-1) {
            n= btComm.readMessage("nxt 1");
        }
        TextView text = (TextView) findViewById(R.id.text2);
        if(n==99){
            text.setText("Moure esquerra");
        }
        else if(n==100){
            text.setText("Posicio correcta");
        }
        else if(n==101){
            text.setText("Moure dreta");
        }
    }
}
```

7.3. Programació dels bricks NXT.

El programa dels bricks NXT està dividit en dues parts. Per una part tenim el programa anomenat "grua" que correspon al brick número 1 i s'encarrega de controlar el funcionament de la grua. Per l'altra tenim el programa anomenat "motriu" que correspon al brick número 2 i s'encarrega del desplaçament del robot.

Ambdós NXT estan connectats al mòbil, que funciona com a màster com abans he mencionat, però entre ells no estan connectats de manera que els dos programes funcionen de manera independent l'un de l'altre, però depenen del mòbil.

Aquests programes segueixen el mateix esquema. Mitjançant la sentència while que depèn del booleà "bucle" (inicialment en estat true) mantenim el NXT en mode espera fins que es realitzi la connexió bluetooth amb el mòbil. A la pantalla LCD apareix el missatge "waiting" fins que es connecta, que passa a ser "connected". Llavors creem l'objecte "dis" que és el canal de comunicació entre el mòbil i el NXT i sobre el qual aplicarem mètodes per llegir els missatges rebuts. A continuació l'acció entra en un altre while dins l'anterior que depèn del booleà anomenat "isrunning" (inicialment en estat true). Dins aquest while es determinen les accions a seguir segons els missatges

rebuts, els quals redirecciones a les tasques designades diferents en cada un dels programes.

```
Boolean isrunning = true;
Boolean bucle = true;
    //Bucle que manté el nxt buscant connexió.
    while (bucle){
        LCD.drawString("waiting",0,0);
        LCD.refresh();
        //Esperant per la connexió.
            NXTConnection btc = Bluetooth.waitForConnection();
            btc.setIOMode(NXTConnection.RAW);
        LCD.clear();
        LCD.drawString("connected",0,0);
        LCD.refresh();

        //Canal per anar ficant dades.
        DataInputStream dis = btc.openDataInputStream();
        //Bucle per a llegir la informació que va sent rebuda
pel brick. S'executa quan hi ha connexió.
        while (isrunning){
            //Inicialment el missatge és -1, no fa res (cap
ordre).
            Byte n=-1;
            //Mirem si hi ha algun missatge al canal (el canal
es dis.).
            try{
                n=dis.readByte();
//Cas per tancar el programa comú als dos programes (grua i
motriu).
                if (n==111){
                    isrunning=false;
                    bucle=false;
                }
                else if (//missatges//){
                    //Instruccions//
                }
                else if (//missatges//){
                    //Instruccions//
```

```
                    }
                    else if (//missatges//){
                    //Instruccions//
                    }
                }
                catch(Exception e){
                }
                LCD.clear();
                LCD.drawInt(n, 4, 4);
            }
            //Tancar el canal.
            try {
                dis.close();
            }
            catch(Exception e){}
            try {
                Thread.sleep(100);
            }
            catch (Exception e){
            }
        LCD.clear();
        LCD.drawString("closing",0,0);
        LCD.refresh();
        btc.close();
        LCD.clear();
       }
}
```

Aquesta plantilla és comuna a ambdós programes i segueix l'esquema d'un programa extret d'internet. Als apartats que venen a continuació s'explica com es tracten els missatges rebuts i amb quin programa han sigut escrits i compilats.

7.3.1. Sublime Text 2 i CMD per compilar

El programa que he utilitzat per escriure el codi és el Sublime Text 2. Es tracta d'un editor de text especial per a codi amb funcions molt útils a l'hora de navegar pel nostre codi i editar-lo. Per exemple, consta d'una

columna a l'esquerra on es pot veure el codi sencer i es pot navegar més fàcilment per ell. També, per facilitar la visualització i diferenciar els elements classifica el que apareix el nostre codi mitjançant colors. A més, a l'hora d'editar-lo disposa d'eines com escriure a més d'una línia alhora, marcar totes les paraules iguals a la que seleccionem...

Els programes els he compilat i transferit al NXT mitjançant la CMD de l'ordinador i els comandaments que apareixen a la pàgina de leJOS següents:

```
Nxjc exemple.java
Nxj –o exemple.nxj exemple
```

A més, per a poder passar els programes al brick vaig haver d'instalar el firmware leJOS substituint l'original, també mitjançant la web de leJOS.

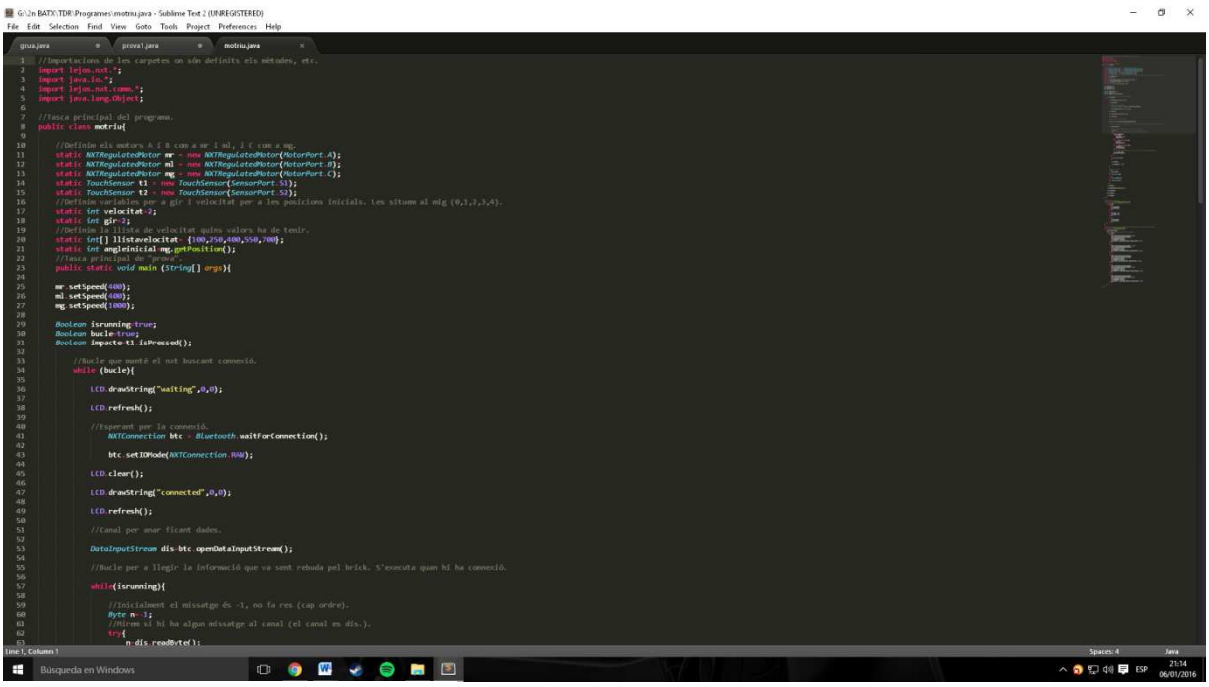


Fig 7.5. Interfície Sublime Text 2.

7.3.2. Programació de la grua (NXT 1)

Iniciem el programa com sempre, amb les importacions.

```
//Importacions de les carpetes on són definits els mètodes, etc.
import lejos.nxt.*;
import java.io.*;
import lejos.nxt.comm.*;
import java.lang.Object;
```

A continuació declarem els objectes mA, mB i mC i determinem quines son les seves propietats (es tracta de tres servomotors, així que els declarem igual). De la mateixa manera declarem els objectes "ultrasonicsensorL", "ultrasonicsensorR" i "colorsensor" com a sensors, dos d'ultrasons i un de color.

```
  // Definim els motors A, B i C com mA, mB i mC respectivament.
    static NXTRegulatedMotor mA = new NXTRegulatedMotor(MotorPort.A);
    static NXTRegulatedMotor mB = new NXTRegulatedMotor(MotorPort.B);
    static NXTRegulatedMotor mC = new NXTRegulatedMotor(MotorPort.C);



// Definim els sensors ultraasonics i de color S1, S2, S3.
    static UltrasonicSensor ultrasonicsensorL = new
UltrasonicSensor(SensorPort.S1);
    static UltrasonicSensor ultrasonicsensorR = new
UltrasonicSensor(SensorPort.S2);
    static ColorSensor colorsensor = new ColorSensor(SensorPort.S3);
```

Declarem "angleinicialA", "angleinicialB" i "angleinicialC" com a variables de tipus enter. D'aquesta manera captem amb el mètode ".getPosition();" en quin angle es troben els motors de la grua per després operar sobre ells.

```
//Agafem el valor del angle en el qual es troba cada motor.
    static int angleinicialA=mA.getPosition();
    static int angleinicialB=mB.getPosition();
    static int angleinicialC=mC.getPosition();
```

Abans de començar amb la plantilla per rebre missatges, i ja dins la tasca principal del programa definim les velocitats dels motors de la grua.

```
//Tasca principal de "grua".
    public static void main (String[] args){


    mA.setSpeed(100);
    mB.setSpeed(200);
    mC.setSpeed(100);
```

El tractament de missatges al programa de la grua ens porta a les següents tasques: si el missatge és 111, els booleans "isrunning" i "bucle" passen a ser falsos, per tant es surt dels whiles abans explicats, doncs la condició no es compleix. Si el missatge és 14 (el que envia el botó CATCH) llavors s'acudeix a la tasca agafarpilota(btc);.

```
        try{
                n=dis.readByte();
                //Si el missatge es 111, la tasca s'atura.
                if (n==111){
                    isrunning=false;
                    bucle=false;
                }
                //Si el missatge és 14 (botó catch), es recorre a
la tasca agafar pilota.
                else if (n==14){
                    agafarpilota(btc);
                }
            }
            catch(Exception e){
            }
```

La tasca "agafarpilota" determina a quina altra tasca haurà d'acudir el robot segons la informació que rebi dels sensors. Primer determinem les variables "distanceL" i "distanceR" com a variables enteres que prenen el valor del sensor ultrasònic situat a l'esquerra del robot i el situat a la dreta del robot respectivament. Llavors determina quin objecte està més pròxim i envia a la

tasca corresponent segons si és el de la dreta o el de l'esquerra. Suposem per tant que sempre que hi ha un objecte serà una pilota. Si les distàncies són iguals, contemplem el cas però no realitzem res al respecte.

```
//Tasca agafar pilota
public static void agafarpilota (NXTConnection btc){
    //Introduim les variables enteres distanceR i distanceL.
    int distanceR, distanceL;
    distanceR=ultrasonicsensorR.getDistance();
    distanceL=ultrasonicsensorL.getDistance();
    //Cas per acudir a la tasca pilota dreta.
    if (distanceR<distanceL){
        pilotadreta(distanceR, btc);
    }
    //Cas per acudir a la tasca pilota esquerra.
    else if (distanceL<distanceR){
        pilotaesquerra(distanceL,btc);
    }
    else{
    }
}
```

Si la distància mesurada pel sensor esquerra és més petita que la distància mesurada pel sensor de la dreta s'acudeix a la tasca "pilotaesquerra". Si no, s'acudeix a la tasca "pilotadreta". Aquestes dues tasques realitzen el mateix procés: primer envien un missatge que serà rebut pel mòbil mitjançant la tasca "enviarmissatge". Segons la distància a la qual es trobi l'objecte detectat s'envia un missatge diferent (pot ser que estigui a la posició correcta, sigui necessari moure el robot a la dreta o a l'esquerra).

```
//Tasca pilota dreta.
public static void pilotadreta (int distance, NXTConnection
btc){
    Byte z;
    //Casos segons on estigui de distància la pilota.
    if ((14>=distance)&&(distance>=8)){
        z=100; //posició correcta
        enviarmissatge(z, btc);
        mouregruaR();
```

```
            }

            else if (distance<8){
                z=99; //Si és 99 cal anar a l'esquerra.
                enviarmissatge(z, btc);
            }

            else{
                z=101; //Si és 101 cal anar a la dreta.
                enviarmissatge(z, btc);
            }
        }
        //Tasca pilota esquerra.
        public static void pilotaesquerra (int distance,
    NXTConnection btc){
            Byte z;
            //Casos segons on estigui de distància la pilota.
            if ((14>=distance)&&(distance>=8)){
                z=100;
                enviarmissatge(z, btc);
                mouregruaL();
            }

            else if (distance<8){
                z=101;
                enviarmissatge(z, btc);
            }

            else{
                z=99;
                enviarmissatge(z, btc);
            }
        }
```

El missatge és enviat amb una tasca anomenada "enviarmissatge" que té la mateixa estructura que la de rebre missatges.

```
//Tasca enviarmissatge per enviar missatge del nxt al mòbil.
    public static void enviarmissatge (Byte n, NXTConnection
btc){
          DataOutputStream dis=btc.openDataOutputStream();
        try {
               dis.writeByte(n);
               dis.close();
        }
        catch(Exception e){
        }
    }
```

Quan la posició de l'objecte detectat és la correcta s'acudeix a la tasca "mouregruaL" o "mouregruaR" segons on estigui l'objecte. Aquestes tasques són iguals: mitjançant el mètode ".rotate(angle);", fem que els servomotors girin l'angle que nosaltres desitgem i mitjançant el mètode ".waitComplete();" fem que el programa esperi a executar la següent ordre fins que acabi la que s'està duent a terme, si no es mourien tots els motors alhora. La diferència està en el motor de la base de la grua (mA), doncs depenent de si ha de girar la grua a l'esquerra o a la dreta l'angle serà negatiu o positiu (gira en sentit horari o antihorari). Hem de tenir en compte en quin angle està el motor inicialment per a poder posicionar-lo a l'angle desitjat. Ho determinem mitjançant les variables abans explicades i operem segons l'angle al qual ha de girar el motor amb una simple resta. Quan ha acabat el procés s'acudeix a la tasca "colorpilota" en tots dos casos.

```
//Tasca per moure grua a la dreta.
    public static void mouregruaR(){
        mA.rotate(115-angleinicialA, true);
        mA.waitComplete();
        mB.rotate(-670-angleinicialB, true);
        mB.waitComplete();
        mA.rotate(-25-angleinicialA, true);
        mA.waitComplete();
        mC.rotate(-140-angleinicialC, true);
```

```
        mC.waitComplete();
        mB.rotate(670-angleinicialB, true);
        mB.waitComplete();
        mA.rotate(-90-angleinicialA, true);
        mA.waitComplete();
        colorpilota();
    }

    //Tasca per moure grua a l'esquerra.
    public static void mouregruaL(){
        mA.rotate(-115-angleinicialA, true);
        mA.waitComplete();
        mB.rotate(-670-angleinicialB, true);
        mB.waitComplete();
        mA.rotate(25-angleinicialA, true);
        mA.waitComplete();
        mC.rotate(-140-angleinicialC, true);
        mC.waitComplete();
        mB.rotate(670-angleinicialB, true);
        mB.waitComplete();
        mA.rotate(90-angleinicialA, true);
        mA.waitComplete();
        colorpilota();
```

Amb la tasca "colorpilota" detectem de quin color és la pilota que hem agafat. Hi ha dues opcions, blava o vermella. Amb el mètode ".getColorID();" aconseguim traduir la informació que rep el nostre sensor (anomenat colorsensor) a un nombre enter i ho guardem a la variable "resultat". Per al color blau el nombre enter corresponent és 2 i per a vermell és 0 (informació de l'API de leJOS). Llavors amb la sentència if/else valorem els casos possibles i actuem en conseqüència: si el color detectat és 2 (blau), s'acudeix a la tasca "pilotablava"; si el color detectat és 0 (vermell), s'acudeix a la tasca "pilotavermella"; i si el color detectat no correspon a cap d'aquest dos valors es torna a deixar l'objecte al terra invertint el signe dels angles de rotació dels motors de la tasca "mouregruaL"

```
public static void colorpilota(){
        int resultat;
        resultat=colorsensor.getColorID();
        if (resultat==2){
            pilotablava();
        }
        else if(resultat==0){
            pilotavermella();
        }
        else{
            mA.rotate(-115-angleinicialA, true);
            mA.waitComplete();
            mB.rotate(-670-angleinicialB, true);
            mB.waitComplete();
            mA.rotate(25-angleinicialA, true);
            mA.waitComplete();
            mC.rotate(140-angleinicialC, true);
            mC.waitComplete();
            mB.rotate(670-angleinicialB, true);
            mB.waitComplete();
            mA.rotate(90-angleinicialA, true);
            mA.waitComplete();
        }
    }
```

Finalment, les tasques "pilotablava" i "pilota vermella" són iguals i l'únic que canvia és el sentit de rotació del motor "mA", com en el cas de les tasques "mouregruaL" i "mouregruaR". D'aquesta manera el robot diposita la pilota en una secció de la caixa o en altra depenent del seu color.

```
public static void pilotablava(){
        mA.rotate(30, true);
        mA.waitComplete();
        mC.rotate(45, true);
        mC.waitComplete();
        mC.rotate(-265, true);
        mC.waitComplete();
        mA.rotate(-30, true);
```

```
        mA.waitComplete();
    }
    public static void pilotavermella(){
        mA.rotate(-30, true);
        mA.waitComplete();
        mC.rotate(45, true);
        mC.waitComplete();
        mC.rotate(-265, true);
        mC.waitComplete();
        mA.rotate(30, true);
        mA.waitComplete();
    }
```

7.3.3. Programació de la part motriu (NXT 2)

Al principi trobem les importacions.

```
//Importacions de les carpetes on són definits els mètodes, etc.
import lejos.nxt.*;
import java.io.*;
import lejos.nxt.comm.*;
import java.lang.Object;
```

Tot seguit, dins de la classe principal del programa declarem els objectes "mr", "ml" I "mg" com a motors que són als quals els hi aplicarem els mètodes necessaris posteriorment.

```
public class motriu{
    //Definim els motors A i B com a mr i ml, i C com a mg.
    static NXTRegulatedMotor mr = new
NXTRegulatedMotor(MotorPort.A);
    static NXTRegulatedMotor ml = new
NXTRegulatedMotor(MotorPort.B);
    static NXTRegulatedMotor mg = new
NXTRegulatedMotor(MotorPort.C);
```

Definim les variables "gir" i "velocitat" que dependran de la posició de les barres de gir i velocitat del mòbil. Establim que inicialment estan situades a 2 (posició central).

```
//Definim variables per a gir i velocitat per a les posicions
inicials. Les situem al mig (0,1,2,3,4).
    static int velocitat=2;
    static int gir=2;
```

Tot seguit determinem els valors de la llista de velocitat, que seran els valors que adoptin els motors depenent d'on es situi la barra del mòbil de velocitat. També, de la mateixa manera que hem fet amb els motors de la grua, guardem l'angle en el qual està el motor de gir (mr) inicialment per poder operar sobre ell més tard.

```
//Definim la llista de velocitat quins valors ha de tenir.
    static int[] llistavelocitat= {100,250,400,550,700};
    static int angleinicial=mg.getPosition();
```

Ja dins la tasca principal del programa definim les velocitats a les quals actuaran els diferents motors de la grua (hem de tenir en compte quins motors tenen reducció i quins no per poder realitzar el moviment a la velocitat desitjada. Per això el motor "mg" va a una velocitat més elevada que la resta).

```
    public static void main (String[ ] args){

    mr.setSpeed(400);
    ml.setSpeed(400);
    mg.setSpeed(1000);
```

Com en l'anterior programa, dins de la part comuna anteriorment explicada es tracten els missatges rebuts d'una determinada manera. Introduim dins el bloc "try" de la sentència try/catch el que ha de fer el programa segons quin missatge rebi. Com en l'anterior, utilitzem la sentencia if/else. Si el missatge és 111 el programa s'atura i es tanca. Si el missatge està comprès entre els valors enters 4 i 8, ambdós inclosos, li restem 4 al missatge rebut (n) per a poder operar amb valors entre 0 i 4 i s'acudeix a la tasca "girvelocitat".

Si el missatge està comprès entre els valors enters 9 i 13, ambdós inclosos, li restem 9 al missatge rebut per a poder operar també amb valors entre 0 i 4 i s'acudeix a la tasca "girvelocitat".

Per últim, si el missage és diferent a -1 (l'inicial) i després d'haver contemplat tots els casos anteriors, és a dir, si el missatge és diferent a -1 i no és cap valor entre 4 i 13, s'acudeix a la tasca "tractarMissatge".

```
try{
                    n=dis.readByte();
                    if (n==111){
                        isrunning=false;
                        bucle=false;
                    }
                    else if ((n>=4) && (n<=8)){
                        gir=n-4;
                        girvelocitat();
                    }
                    else if ((n>=9) && (n<=13)){
                        velocitat=n-9;
                        girvelocitat();
                    }
                    else if (n!=-1){
                    tractarMissatge(n);
                    }
                }
catch(Exception e){
}
```

La tasca "tractarMissatge" simplement determina per a tres casos (missatges 1, 2 i 3) si el robot anirà endavant o enrrere, o bé es pararà.

```
//Tasca tractar missatge.
public static void tractarMissatge (Byte n){
    //Casos segons el missatge enviat (n).
    switch (n){
        case 1:
            mr.backward();
            ml.backward();
        break;

        case 2:
            mr.stop(true);
            ml.stop(true);
        break;

        case 3:
            mr.forward();
            ml.forward();
        break;
    }
}
```

La tasca "girvelocitat" és la més complexa del programa. Quan volem que el robot giri no només hem de rotar el motor de gir (mg) un determinat angle, doncs quan girem les rodes motrius interiors al gir tindran una velocitat menor a les rodes exteriors al gir perquè el seu radi será menor. És per això que hem d'establir la relació entre les revolucions de la roda interior al gir i les de la roda exterior per a cada una dels dos angles de gir i així modificar les velocitats dels motors "mr" i "ml". Les equacions que expressen aquesta relació són les següents:

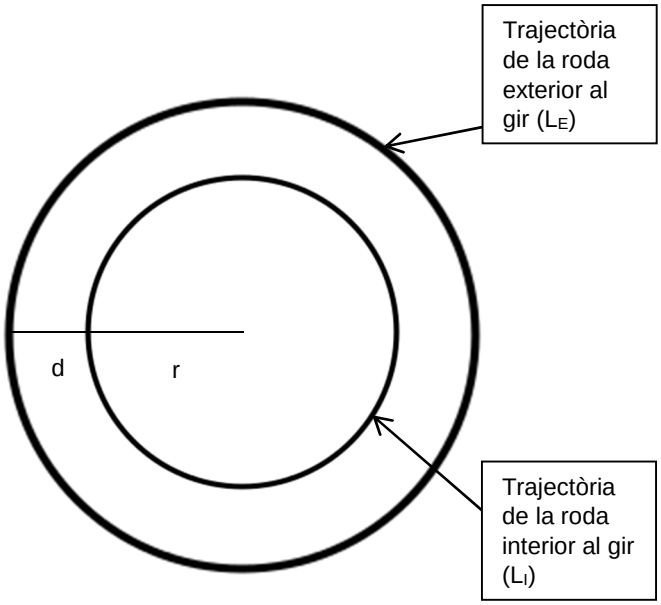


Fig 7.6. Esquema del gir del robot.

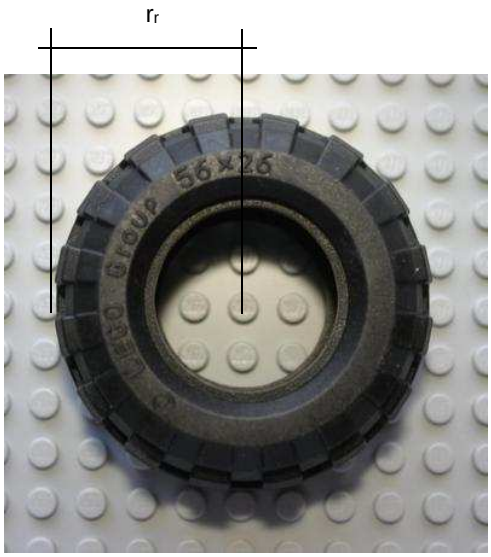


Fig 7.7. Roda del robot.

L_E = Longitud de la circumferència descrita per la roda exterior

L_I = Longitud de la circumferència descrita per la roda interior

r = Radi de la circumferència interior

d = Distància entre les rodes

} r+d = radi de la circumferència exterior

r_r = Radi de la roda del robot

L_r = Longitud de la circumferència de la roda del robot ($L_r = 2\pi r_r$)

n_E = Nombre de voltes que fa la roda exterior

n_I = Nombre de voltes que fa la roda interior

$$L_I = 2\pi r \longrightarrow L_I = n_I L_r = n_I 2\pi r_r \longrightarrow n_I 2\pi r_r = 2\pi r \longrightarrow n_I r_r = r$$

$$L_E = 2\pi(r+d) \longrightarrow L_E = n_E L_r = n_E 2\pi r_r \longrightarrow n_E 2\pi r_r = 2\pi(r+d) \longrightarrow n_E r_r = (r+d)$$

$$\left.\begin{array}{l} n_I r_r = r \\ n_E r_r = r + d \end{array}\right\} \quad n_I r_r = n_E r_r - d$$

Amb aquesta equació, calculant els paràmetres r_r i d obtenim la relació entre les revolucions de la roda exterior (n_E) i les de la roda interior (n_I). El que passa és que hem de tenir en compte l'angle de gir, per tant amb aquesta equació no som capaços de resoldre el problema.

Per solucionar el problema vaig decidir mesurar-ho sobre el terreny, ja que em resultaria més fàcil. Vaig col·locar les rodes en un angle determinat i vaig marcar les dues rodes (interior i exterior) al mateix punt. Vaig fer avançar el robot fins que la roda exterior completés una volta, i vaig mirar l'angle que havia descrit la interior. Amb una simple operació sabem la seva relació:

$$\frac{\varphi_I}{\varphi_E} = Relació$$

A partir d'aquesta operació (més senzilla que l'anterior) em va ser molt més fàcil calcular la relació. Per al gir més tancat l'operació va ser $\frac{\varphi_I}{\varphi_E} = \frac{220^o}{360^0} \approx 0{,}610$ i per al gir més obert va ser $\frac{\varphi_I}{\varphi_E} = \frac{260^o}{360^o} \approx 0{,}725$.

Un cop calculada la relació la introduïm al programa de la següent manera. Amb la sentència switch/case elaborem els 5 casos possibles. Dins de cada cas creem les variables "girl" i "girr" i apliquem la relació a la roda corresponent. Si per exemple girem a la dreta al màxim (gir tancat) s'aplicarà la relació 0,615 a la roda dreta, doncs en aquest cas aquesta és la interior. Per tant la variable a la qual se li aplica la relació és "girr". A més, hem de tenir en compte a quina velocitat està anant el robot en l'instant al qual girem, per tant la relació la apliquem al valor en el qual estigui de la llista de velocitat que hem intorduit abans ("llistaveocitat[velocitat]"). Aquest exemple seria el pertanyent al cas 0.

És aquest el motiu pel qual la modificació de la barra de gir del mòbil i la de la barra de velocitat es tracten a la mateixa tasca, doncs la relació del gir entre les rodes exteriors i interiors depèn de la velocitat a la qual vagin.

Per a controlar el motor "mr" i determinar l'angle al qual ha de rotar hem de tenir en compte en quina posició està, doncs no seria el mateix girar desde la posició a la qual les rodes estan rectes fins al màxim angle de gir a l'esquerra que des del màxim angle de gir a la dreta fins el màxim angle de gir a l'esquerra.

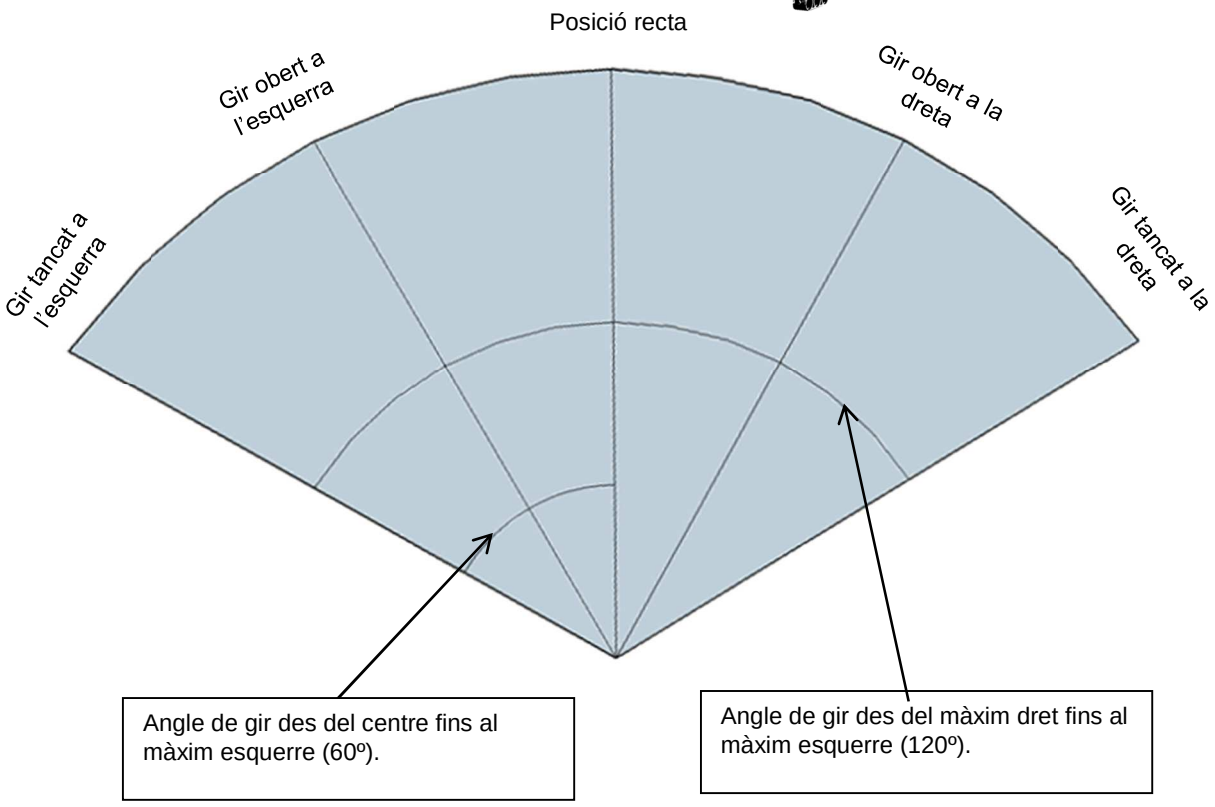


Fig 7.8. Esquema de l'angle de gir del robot.

Per a poder abarcar totes les possibilitats utilitzem el mètode ".getPosition();" dins de cada cas i el restem (junt amb l'angle inicial al qual estava al motor, que sempre l'hem de tenir en compte com a referència) al angle que volem que roti el motor (el qual introduïm nosaltres). L'únic que cal fer és aplicar aquest plantejament per a tots els casos només canviant l'angle de rotació del motor i la relació de gir entre les rodes exteriors i interiors. Així podem veure que l'estructura del codi per a cada cas és igual i únicament canvien aquest factors.

```
public static void girvelocitat (){
        //Definim variables girr i girl.
        double girr, girl;
        //Casos en funció de la posició de la barra gir.
        switch (gir){
            case 0:
                girl=(llistavelocitat[velocitat]*0.610);
                girr=(llistavelocitat[velocitat]);
                mr.setSpeed((int)girr);
                ml.setSpeed((int)girl);
                mg.rotate(810-(mg.getPosition()-angleinicial), true);
                break;
```

Canviar la velocitat entre les rodes exterior i interior.

Mètode per determinar l'angle de gir de "mg".

```
            case 1:
                girl=(llistavelocitat[velocitat]*0.725);
                girr=(llistavelocitat[velocitat]);
                mr.setSpeed((int)girr);
                ml.setSpeed((int)girl);
                mg.rotate(540-(mg.getPosition()-angleinicial), true);
            break;

            case 2:
                girl=(llistavelocitat[velocitat]);
                girr=(llistavelocitat[velocitat]);
                mr.setSpeed((int)girr);
                ml.setSpeed((int)girl);
                mg.rotate(0-(mg.getPosition()-angleinicial), true);
            break;

            case 3:
                girl=(llistavelocitat[velocitat]);
                girr=(llistavelocitat[velocitat]*0.725);
                mr.setSpeed((int)girr);
                ml.setSpeed((int)girl);
                mg.rotate(-540-(mg.getPosition()-angleinicial), true);
            break;

            case 4:
                girl=(llistavelocitat[velocitat]);
                girr=(llistavelocitat[velocitat]*0.610);
                mr.setSpeed((int)girr);
                ml.setSpeed((int)girl);
                mg.rotate(-810-(mg.getPosition()-angleinicial), true);
            break;

        }
    }
}
```

8. Conclusions

Per concluir el treball em disposo a realitzar una valoració del mateix mirant si els objectius han sigut aconseguits, quines dificultats m'han sorgit i quines millores es podrien realitzar sobre el meu projecte per tal de millorar el funcionament del robot o ampliar les seves funcions.

Com els objectius están dividits en teòrics i pràctics a aquest apartat del treball el tractaré de la mateixa manera.

Amb aquest treball he pogut aprendre de manera autònoma com funcionen els sistemes de control i al mateix temps, gràcies a aquest coneixement adquirit comprenc el funcionament de molts dels elements amb els quals convisc, si més no puc establir relacions entre el que veig i el que he après. És a dir, he pogut suportar en coneixements teòrics el que veiem tots dia a dia i que ens envolta fent-nos la vida més fàcil que al cap i a la fi és l'objectiu de la tecnologia.

Respecte a la programació, gràcies a aquest projecte he adquirit una visió més general de la seva metodologia (en concret la del llenguatje Java) i al cap i a la fi m'hé adonat, com deia Seymour Papert, que per aprendre no és necessari sempre rebre una bona instrucció, si no tractar de cercar la informació necessària per dur a terme un projecte i aprendre dels errors. És més important compendre el concepte d'un assumpte i quin és el seu funcionament i després cercar la informació necessària que no pas emmagatzemar tota la informació al cervell de manera metòdica, doncs el que ens pot passar si fem això és que no puguem resoldre cap problema que surti del nostre esquema perquè no ho haurem interioritzat.

En quant a Lego Mindstorms he canviat totalment la meva idea sobre el producte, doncs m'he informat i he vist que és una "joguina" molt més complexa del que sembla. Mai hauria pensat que amb aquest producte es podrien realitzar tantes creacions diferents i oferirira un ventall de possibilitats tan gran al usuari, doncs he vist des d'un robot que realitza el cub de rubik o un altre que fa sudokus fins a vaixells o grues immenses. També he pogut conèixer gràcies a la investigació en aquest assumpte la quantitat de competicions que es

realitzen arreu del món i l'interès que té la gent en la robòtica en general, tant a nivell aficionat com el meu fins a nivell professional.

La part pràctica del treball ha complert els objectius en gran part. He pogut realitzar el programa dels NXT i compendre el seu funcionament de manera empírica més que teòrica i he après a traduir les meves idees conceptuals sobre com havia de funcionar el robot en un codi escrit que les transformés en realitat. M'he adonat que, amb la programació, podem dur a terme totes les nostres idees i convertir-les en fets reals, ja sigui en llenguatge Java o qualsevol dels centenars existents actualment.

He pogut realitzar l'aplicació del mòbil però m'agradaria saber utilitzar més l'Android Studio, doncs el trobo un programa molt interessant del qual no he pogut conèixer totes les seves característiques i funcions. Així doncs m'agradaria conèixer més en profunditat aquest programa per a poder realitzar projectes amb ell en un futur.

En quan al disseny de peces en 3D he pogut resoldre tots els problemes que m'anaven sorgint i tot i que al principi em costava realitzar una peça que s'adaptés perfectament al que jo volia i havia de fer molts intents (com en el cas de la cremallera) estic satisfet del que he pogut aprendre respecte a aquest tema.

Pel que fa referència al muntatge del robot he après i m'he adonat d'un aspecte molt important de la tecnologia en gran part gràcies a aquesta part del treball: el procés tecnològic. He pogut comprovar de primera mà com una idea o un prototip inicial va adquirint forma. No hi ha hagut cap part de l'estructura la qual l'hagi realitzat a la primera. Sempre em plantejava com podria millorar-la, fer-la més eficient, més funcional, menys voluminosa... Al cap i a la fi el que estava fent era repetir el procés tecnològic: Quan havia acabat el que tenia en ment tornava a analitzar el problema que em sorgia i com podria solventar-lo i millorar l'estructura que havia creat buscant les peces adequades i muntant i desmuntant fins a aconseguir el que volia. Fins i tot amb el robot acabat encara em venen idees a la ment sobre com millorar-lo, que és del que parlaré a continuació.

De totes les millores que faria al robot mencionaré les més importants o les que més canvi suposarien deixant de banda petits ajustos que millorarien el funcionament del robot acual però no afegirien res de nou.

Primerament m´hagués agradat d’alguna manera elaborar un sistema de control per a corregir l’error produït a l’hora d’efectuar el gir: d’aquesta manera el robot desplaçaria en una línia recta perfecte. Tot i així, donada la complexitat de la tasca tenint en compte que l’he de realitzar en llenguatge Java i que depenc d’una connexió bluetooth no em seria possible d’elaborar-ho.

També introduiria un sistema de paraxocs mitjançant els sensors de tacte(el qual ja vaig intentar implementar) que fes la funció de aturar el robot quan aquest xoqués al avançar o retrocedir i que es mostrés un missatge a la pantalla del mòbil que indiqués que ha xocat.

M’agradaria també haver solucionat un problema que de vegades succeeix a l’hora de detectar el color de la pilota. Les condicions de llum ambient a vegades fan que el robot no detecti el color de manera precisa i per tant no és capaç de classificar la pilota. Segons he investigat es pot crear un programa per tenir en compte les condicions de llum ambient. El que faria seria que el robot et permetés introduir unes condicions de llum ambient per a poder detectar el color de manera precisa, tot i que és una tasca també força complicada.

Per últim només vull mencionar que amb el treball he complert un dels objectius principals que tenia: saber com era el món de la robòtica i viure’l de primera mà per poder decidir si és al que m’hi vull dedicar en un futur de manera professional, i val a dir que l’ha complert de manera exitosa.

9. Glossari

La robòtica

1. **Ordit:** Sèrie de fils que van de llarg a llarg de la peça de roba teixida, paral·lelament a les vores.

Evolució del brick

1. **Firmware:** Conjunt d'instruccions d'un programa informàtic que s'ubica en una memòria ROM, flash o similar. Aquestes instruccions determinen la lògica primària que exerceix el control dels circuits de la computadora o dispositiu.
2. **Software:** Conjunt de programes, instruccions i regles informàtiques que permeten executar diferents tasques en una computadora.
3. **Hardware:** Conjunt de elements físics que formen part d'una computadora, tant siguin interns com externs a la computadora.
4. **Microcontrolador:** Circuit integrat programable capaç d'executar les ordres guardades a la seva memoria.
5. **Memòria RAM: "**Random Acces Memory" (memòria d'accés aleatori). Component utilitzat en informàtica per a emmagatzemar programes i dades informatives. Es tracte d'una memòria que perd les seves dades quan deixa de rebre energia (memòria volàtil). S'utilitza per a manejar dades i informació circumstancialment amb programes i softwares. És la que permet el funcionament d'aquests programes ja que tracta amb la informació amb la que s'executen els programes. Un com tancat el programa, la infomació es perd perquè el procés finalitza.
6. **Memòria ROM:** "Read Only Memory" (memòria només de lectura). Component que emmagatzema informació i només permet la lectura d'aquesta, no la seva destrucció. A diferència de la memòria RAM, aquest tipus de memòria no perd la informació que conté quan deixa de rebre energia (memòria no volàtil). Aquest tipus de memòria s'utilitzen per a emmagatzemar programes de configuració del sistema, d'inici, suports físics i d'altres programes que no precisen d'actualitzacions constants.
7. **Bit:** Dígit del sistema de numeració binari.

8. **Pantalla LCD:** "Liquid Crystal Display". Pantalla prima i plana formada per un cert nombre de píxels (depenent de la resolució) col·locats davant d'una font lluminosa o reflectora. Utilitzada en dispositius electrònics alimentats amb piles ja que té un consum molt baix.
9. **Memòria Flash:** Tipus de memòria semblant a la ROM basada en la EEPROM (Electrically Erasable Programmable Read-Only Memory) que permet guardar informació de manera permanent (amb falta d'energia) però també borrar-la o modificar-la.
10. **Bluetooth:** Denominació atribuïda a les xarxes inalàmbriques que funcionen a una freqüència segura de 2'4 GHz i que permeten la transmissió de dades i de veu entre diferents dispositius.
11. **Host:** Terme utilitzat en el món de la informàtica traduït com a amfitrió utilitzat per a designar les computadores connectades a una xarxa que proveeixen i utilitzen serveix de la mateixa.
12. **Perl:** Llenguatge de programació inventat per Larry Wall l'any 1987 que pren característiques de llenguatges com el C, l'AWK, el sed i el Lisp entre d'altres. Va ser adoptat per la seva destresa en el processat de text i no tenir cap de les limitacions dels altres llenguatges de script.
13. **Ruby:** Llenguate de programació orientat a objectes creat per Yukihiro "Matz" Matsumoto al 1995. Combina una sintaxis inspirada en Python i Perl i característiques de la programació orientada a objectes similars a Smalltalk.

10. Bibliografia i webgrafia

La robòtica i els sistemes automàtics i de control

https://es.wikipedia.org/wiki/Rob%C3%B3tica

http://www.profesormolina.com.ar/tecnologia/robotica/historia.htm

https://roboticstoday.wikispaces.com/Historia+de+la+Rob%C3%B3tica

AGUSTÍ, C. (Ed.); "Sistemes automàtics i de control" a: *Tecnologia industrial. 2n Batxillerat.*Madrid, 2008, Mc. Graw Hill, pp. 213-247.

VILLANUEVA, O. i ÁLVAREZ, A. (Dir.); "La Robótica: Construcció d'un seguidor de línies amb Lego Mindstorms NXT", Sant Fost, 2011.

Lego mindstorms

http://www.citilab.eu/es/que-esta-pasando/videos/archivo/la-historia-de-lego-mindstorms-empezo-en-el-ano-97-con-los-primeros-

http://www.lego.com/es-ar/mindstorms/history

http://es.wikipedia.org/wiki/Lego_Mindstorms

Disseny 3D

http://www.3dcadportal.com/un-proceso-de-modelado-mas-rapido-flexible-e-intuitivo-de-la-informacion-en-sketchup-2015-de-trimble.html

http://www.netfabb.com/basic.php

http://www.tr3sdland.com/2012/12/software-cura-para-impresion-3d-sprinter-vs-marlin/

http://www.robertcailliau.eu/Lego/Dimensions/zMeasurements-en.xhtml

Programació del robot

http://aprenderaprogramar.es/index.php?option=com_content&view=article&id=483:switch-en-java-condicional-de-seleccion-diagrama-de-flujo-y-ejemplo-de-aplicacion-ejercicio-cu00637b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

http://aprenderaprogramar.es/index.php?option=com_content&view=article&id=481:if-else-if-else-if-java-estructura-o-esquema-de-decision-condicional-ejemplos-de-uso-ejercicios-cu00636b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

http://puntocomnoesunlenguaje.blogspot.com.es/2012/04/metodos.html

http://aprenderaprogramar.es/index.php?option=com_content&view=article&id=411:conceptos-de-objetos-y-clases-en-java-definicion-de-instancia-ejemplos-basicos-y-practicos-cu00619b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

http://aprenderaprogramar.es/index.php?option=com_content&view=article&id=426:ique-es-una-clase-java-concepto-atributos-propiedades-o-campos-constructor-y-metodos-cu00623b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/paquetes.htm#El comando import

https://es.wikipedia.org/wiki/Android_Studio

http://www.lejos.org/nxj.php

Glossari

www.rae.es

http://www.definicionabc.com/tecnologia/

http://definicion.de/

https://es.wikipedia.org/wiki/Perl

https://es.wikipedia.org/wiki/Ruby

11. Agraïments

En primer lloc m'agradaria agrair al INS Montmeló per proporcionar-me el material necessari per realitzar el projecte. Sense aquests kits que disposava el centre no hagués pogut realitzar el treball, doncs són molt cars. També agraeixo pel mateix motiu a en M*** A***** C**** ja que sense les peces que em va prestar no hagués pogut completar el robot.

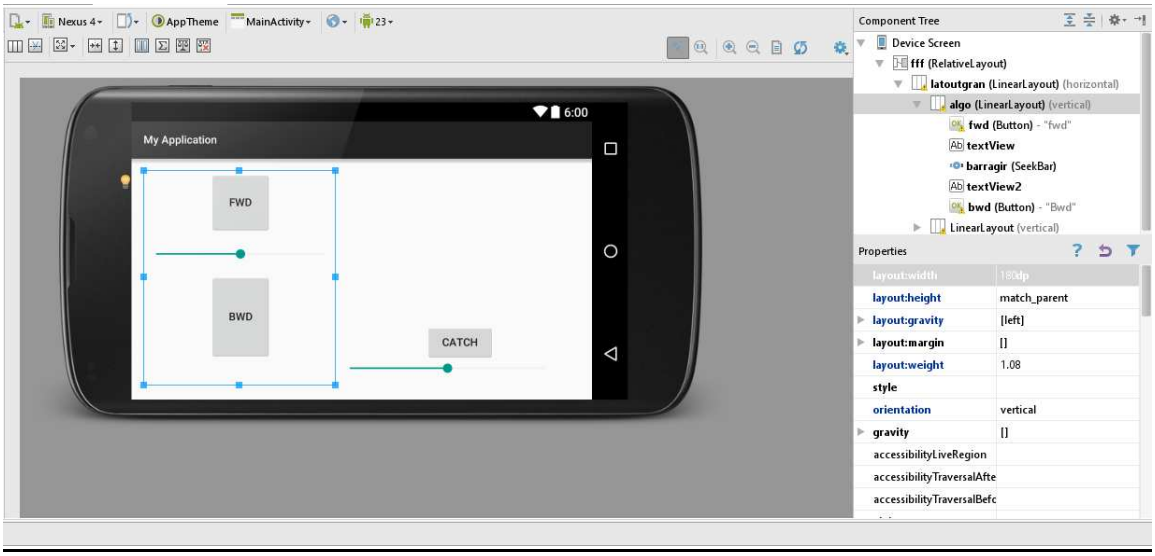
En segon lloc vull mostrar els meus agraïments a l'E*** P******, professor de la Universitat de Barcelona, per introduir-me al món de la programació i permetre'ns participar al concurs de robòtica que es va celebrar a la facultat de matemàtiques de la Universitat de Barcelona amb motiu del "matesfesta". Sense haver participat en aquest concurs no hauria conegut aquest món tan meravellós de la robòtica i no hauria realitzat aquest treball. Amb aquest motiu també agrair a M**** T***** D***-G*******, professora de tecnologia de l'Institut de Montmeló, per confiar en nosaltres per participar en el concurs i acompanyar-nos durant el procés, i també per l'atenció prestada i l'ajuda a l'hora de redactar la memòria escrita del treball.

Per últim, les meves més sinceres mostres d'agraïment a en N****** F****** O****, alumne de la Universitat de Barcelona, per haver-me ajudat tant a l'hora de programar el robot, sempre disponible quan em sorgia un dubte i resolent els errors que donava el codi quan jo era incapaç de solucionar-los. I sobretot per haver-me ensenyat tant la programació en Java, doncs al cap i a la fi començava de zero i ell és qui m'ha ensenyat tot el que sé fins ara, tot i que espero en un futur poder arribar a saber-ne més que ell.

12. Annex

A continuació tots els programes complets sense explicacions ni talls i al final un CD amb vídeos del robot mostrant el seu funcionament.

Programa de l'aplicació del móvil



- Tasca BT_Comm

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.UUID;


import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;

import android.util.Log;

public class BT_Comm {

    //Target NXTs for communication
    final String nxt2 = "00:16:53:1C:62:E6";
    final String nxt1 = "00:16:53:19:22:17";

    BluetoothAdapter localAdapter;
    BluetoothSocket socket_nxt1,socket_nxt2;
    boolean success=false;



    //Enables Bluetooth if not enabled
    public void enableBT(){
        localAdapter=BluetoothAdapter.getDefaultAdapter();
        //If Bluetooth not enable then do it
        if(localAdapter.isEnabled()==false){
            localAdapter.enable();
```

```
            while(!(localAdapter.isEnabled())){

            }
        }

    }

    //connect to both NXTs
    public  boolean connectToNXTs(){



        //get the BluetoothDevice of the NXT
        BluetoothDevice nxt_2 = localAdapter.getRemoteDevice(nxt2);
        BluetoothDevice nxt_1 = localAdapter.getRemoteDevice(nxt1);
        //try to connect to the nxt
        try {
            socket_nxt2 = nxt_2.createRfcommSocketToServiceRecord(UUID
                    .fromString("00001101-0000-1000-8000-
00805F9B34FB"));


            socket_nxt1 = nxt_1.createRfcommSocketToServiceRecord(UUID
                   .fromString("00001101-0000-1000-8000-
00805F9B34FB"));


            socket_nxt2.connect();

            socket_nxt1.connect();

            success = true;



        }
        catch (IOException e) {
            Log.d("Bluetooth","Err: Device not found or cannot
connect");
            success=false;


        }
        return success;

    }


    public void writeMessage(byte msg, String nxt) throws
InterruptedException{
        BluetoothSocket connSock;

        //Swith nxt socket
        if(nxt.equals("nxt 2")){
            connSock=socket_nxt2;
        }else if(nxt.equals("nxt 1")){
            connSock=socket_nxt1;
        }else{
            connSock=null;
        }
```

```
        if(connSock!=null){
            try {

                OutputStreamWriter out=new
OutputStreamWriter(connSock.getOutputStream());
                out.write(msg);
                out.flush();

                Thread.sleep(1000);


            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
            else{
            //Error
        }
    }

    public int readMessage(String nxt){
        BluetoothSocket connSock;
        int n;
        //Swith nxt socket
        if(nxt.equals("nxt2")){
            connSock=socket_nxt2;
        }else if(nxt.equals("nxt1")){
            connSock=socket_nxt1;
        }else{
            connSock=null;
        }

        if(connSock!=null){
            try {

                InputStreamReader in=new
InputStreamReader(connSock.getInputStream());
                n=in.read();

                return n;


            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                return -1;
            }
        }else{
            //Error
            return -1;
        }

    }

}
```

- **Tasca MainActivity**

```
import android.app.Activity;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;
import com.example.nico.myapplication.BT_Comm;
import java.io.IOException;


public class MainActivity extends Activity {
    int variable;
    BT_Comm btComm;
    Boolean isFwd, isBwd;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btComm= new BT_Comm();
        btComm.enableBT();
        btComm.connectToNXTs();
        //Definim que al<principi ni va endevant ni va endarrere
(false)
        isFwd=false;
        isBwd=false;
        //definim quina es la tasca que determina el funcionament de
la barra de gir
        SeekBar yourSeekBar=(SeekBar) findViewById(R.id.barragir);
        yourSeekBar.setOnSeekBarChangeListener(new controladorgir());
        //definim quina es la tasca que determina el funcionament de
la barra de velocitat
        SeekBar yourSeekBar1=(SeekBar)
findViewById(R.id.barravelocitat);
        yourSeekBar1.setOnSeekBarChangeListener(new
controladorvelocitat());
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it
is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);

        return true;
    }
    public void moviment(View view) {
        int id = view.getId();
        Byte msg=-1;
        TextView text = (TextView) findViewById(R.id.textIt);
        //definim els botons fwd i bwd
        Button botobwd = (Button)findViewById(R.id.bwd);
        Button botofwd = (Button)findViewById(R.id.fwd);

        if (id == R.id.fwd) {
            //si premem Fwd i esta retrocedint, es para.
            if(isBwd){
```

```
                msg=2;
                isBwd=false;
                text.setText("Stop");
                botofwd.setText("Fwd");
            }
            //si premem Fwd i esta quiet, avansa.
            else if (!isFwd){
                msg=1;
                isFwd=true;
                text.setText("Fwd");
                botobwd.setText("Stop");
            }
        }

        if (id == R.id.bwd) {
            //si premem Bwd i esta avansant, es para.
            if(isFwd){
                msg=2;
                isFwd=false;
                text.setText("Stop");
                botobwd.setText("Bwd");
            }
            //si premem Bwd i esta quiet, retrocedeix.
            else if(!isBwd){
                msg=3;
                isBwd=true;
                text.setText("Bwd");
                botofwd.setText("Stop");
            }

        }

        try {
            btComm.writeMessage(msg, "nxt 2");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    //tancar l'aplicacio
    public void onBackPressed(){
        Byte msg =111;
        try {
            btComm.writeMessage(msg, "nxt 1");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        try {
            btComm.writeMessage(msg, "nxt 2");

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        this.finish();

    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
```

```
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
    public class controladorgir implements
SeekBar.OnSeekBarChangeListener {

        public void onProgressChanged(SeekBar barragir, int posicio,
boolean fromUser) {
            Byte msg = -1;

            switch (posicio) {
                case 0:
                    msg = 4;
                    break;
                case 1:
                    msg = 5;
                    break;
                case 2:
                    msg = 6;
                    break;
                case 3:
                    msg = 7;
                    break;
                case 4:
                    msg = 8;
                    break;
            }

            try {
                btComm.writeMessage(msg, "nxt 2");

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        public void onStartTrackingTouch(SeekBar seekBar) {}

        public void onStopTrackingTouch(SeekBar seekBar) {}

    }

    public class controladorvelocitat implements
SeekBar.OnSeekBarChangeListener {

        public void onProgressChanged(SeekBar barravelocitat, int
posicio, boolean fromUser) {
            Byte msg = -1;

            switch (posicio) {
                case 0:
                    msg = 9;
                    break;
                case 1:
                    msg = 10;
```

```
                    break;
                case 2:
                    msg = 11;
                    break;
                case 3:
                    msg = 12;
                    break;
                case 4:
                    msg = 13;
                    break;
            }
            try {
                btComm.writeMessage(msg, "nxt 2");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

        }
        public void onStartTrackingTouch(SeekBar seekBar) {}

        public void onStopTrackingTouch(SeekBar seekBar) {}

    }

    public void grua(View view){
            try {
                btComm.writeMessage((byte) 14, "nxt 1");
                escoltarnxt();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
    }

    public void escoltarnxt(){
        int n=-1;
        while (n==-1) {
            n= btComm.readMessage("nxt 1");
        }
        TextView text = (TextView) findViewById(R.id.text2);
        if(n==99){
            text.setText("Moure esquerra");
        }
        else if(n==100){
            text.setText("Posicio correcta");
        }
        else if(n==101){
            text.setText("Moure dreta");
        }
    }
}
```

Programa de la part motriu

```
//Importacions de les carpetes on són definits els mètodes, etc.
import lejos.nxt.*;
import java.io.*;
import lejos.nxt.comm.*;
import java.lang.Object;

//Tasca principal del programa.
public class motriu{
    //Definim els motors A i B com a mr i ml, i C com a mg.
    static NXTRegulatedMotor mr = new NXTRegulatedMotor(MotorPort.A);
    static NXTRegulatedMotor ml = new NXTRegulatedMotor(MotorPort.B);
    static NXTRegulatedMotor mg = new NXTRegulatedMotor(MotorPort.C);
    //Definim variables per a gir i velocitat per a les posicions
inicials. Les situem al mig (0,1,2,3,4).
    static int velocitat=2;
    static int gir=2;
    //Definim la llista de velocitat quins valors ha de tenir.
    static int[] llistavelocitat= {100,250,400,550,700};
    static int angleinicial=mg.getPosition();
    //Tasca principal de "prova".
    public static void main (String[ ] args){

    mr.setSpeed(400);
    ml.setSpeed(400);
    mg.setSpeed(1000);

    Boolean isrunning = true;
    Boolean bucle = true;

        //Bucle que manté el nxt buscant connexió.
        while (bucle){

            LCD.drawString("waiting",0,0);

            LCD.refresh();
```

```
            //Esperant per la connexió.
                NXTConnection btc = Bluetooth.waitForConnection();

                btc.setIOMode(NXTConnection.RAW);

            LCD.clear();

            LCD.drawString("connected",0,0);

            LCD.refresh();

            //Canal per anar ficant dades.

            DataInputStream dis = btc.openDataInputStream();

            //Bucle per a llegir la informació que va sent rebuda pel
brick. S'executa quan hi ha connexió.

            while (isrunning){

                //Inicialment el missatge és -1, no fa res (cap
ordre).
                Byte n=-1;
                //Mirem si hi ha algun missatge al canal (el canal es
dis.).
                try{
                    n=dis.readByte();
                    if (n==111){
                        isrunning=false;
                        bucle=false;
                    }
                    else if ((n>=4) && (n<=8)){
                        gir=n-4;
                        girvelocitat();
                    }
                    else if ((n>=9) && (n<=13)){
```

```
                        velocitat=n-9;
                        girvelocitat();
                    }
                    //Si el byte es diferent a -1 i no es 111, va a la
tasca tractarMissatge.
                    else if (n!=-1){
                    tractarMissatge(n);
                    }
                }
                catch(Exception e){
                }
                LCD.clear();
                LCD.drawInt(n, 4, 4);
            }
            //Tancar el canal.
            try {
                dis.close();
            }
            catch(Exception e){}

            try {
                Thread.sleep(100);
            }
            catch (Exception e){
            }
         LCD.clear();
         LCD.drawString("closing",0,0);
         LCD.refresh();
         btc.close();
         LCD.clear();
        }
    }
    //Tasca tractar missatge.
    public static void tractarMissatge (Byte n){
        //Casos segons el missatge enviat (n).
        switch (n){
            case 1:
```

```
                mr.backward();
                ml.backward();
            break;

            case 2:
                mr.stop(true);
                ml.stop(true);
            break;

            case 3:
                mr.forward();
                ml.forward();
            break;
        }
    }
    //Tasca per modificar la velocitat entre la tracció dreta i
esquerra a l'hora de girar.
    public static void girvelocitat (){
        //Definim variables girr i girl.
        double girr, girl;
        //Casos en funció de la posició de la barra gir.
        switch (gir){
            case 0:
                girl=(llistavelocitat[velocitat]*0.610);
                girr=(llistavelocitat[velocitat]);
                mr.setSpeed((int)girr);
                ml.setSpeed((int)girl);
                mg.rotate(810-(mg.getPosition()-angleinicial), true);
            break;

            case 1:
                girl=(llistavelocitat[velocitat]*0.725);
                girr=(llistavelocitat[velocitat]);
                mr.setSpeed((int)girr);
                ml.setSpeed((int)girl);
                mg.rotate(540-(mg.getPosition()-angleinicial), true);
            break;
```

```
            case 2:
                girl=(llistavelocitat[velocitat]);
                girr=(llistavelocitat[velocitat]);
                mr.setSpeed((int)girr);
                ml.setSpeed((int)girl);
                mg.rotate(0-(mg.getPosition()-angleinicial), true);
            break;

            case 3:
                girl=(llistavelocitat[velocitat]);
                girr=(llistavelocitat[velocitat]*0.725);
                mr.setSpeed((int)girr);
                ml.setSpeed((int)girl);
                mg.rotate(-540-(mg.getPosition()-angleinicial), true);
            break;

            case 4:
                girl=(llistavelocitat[velocitat]);
                girr=(llistavelocitat[velocitat]*0.610);
                mr.setSpeed((int)girr);
                ml.setSpeed((int)girl);
                mg.rotate(-810-(mg.getPosition()-angleinicial), true);
            break;

        }
    }
}
```

Programa de la grua

```
//Importacions de les carpetes on són definits els mètodes, etc.
import lejos.nxt.*;
import java.io.*;
import lejos.nxt.comm.*;
import java.lang.Object;

//Tasca principal del programa.
public class grua{

    // Definim els motors A, B i C com mA, mB i mC respectivament.
    static NXTRegulatedMotor mA = new NXTRegulatedMotor(MotorPort.A);
    static NXTRegulatedMotor mB = new NXTRegulatedMotor(MotorPort.B);
    static NXTRegulatedMotor mC = new NXTRegulatedMotor(MotorPort.C);
// Definim els sensors ultraasonics i de color S1, S2, S3.
    static UltrasonicSensor ultrasonicsensorL = new
UltrasonicSensor(SensorPort.S1);
    static UltrasonicSensor ultrasonicsensorR = new
UltrasonicSensor(SensorPort.S2);
    static ColorSensor colorsensor = new ColorSensor(SensorPort.S3);
    //Agafem el valor del angle en el qual es troba cada motor.
    static int angleinicialA=mA.getPosition();
    static int angleinicialB=mB.getPosition();
    static int angleinicialC=mC.getPosition();

    //Tasca principal de "grua".
    public static void main (String[] args){

    mA.setSpeed(100);
    mB.setSpeed(200);
    mC.setSpeed(100);

    Boolean isrunning=true;
    Boolean bucle=true;
```

```
        //Bucle que manté el nxt buscant connexió.
        while (bucle){

            LCD.drawString("waiting",0,0);

            LCD.refresh();

            //Esperant per la connexió.
                NXTConnection btc = Bluetooth.waitForConnection();

                btc.setIOMode(NXTConnection.RAW);

            LCD.clear();

            LCD.drawString("connected",0,0);

            LCD.refresh();

            //Canal per anar ficant les dades enviades i rebudes.

            DataInputStream dis=btc.openDataInputStream();

            //Bucle per a llegir la informació que va sent rebuda pel
brick. S'executa quan hi ha connexió.

            while (isrunning){

                //Inicialment el missatge es -1, no fa res (cap
ordre).
                Byte n=-1;
                //Mirem si hi ha algun missatge al canal (el canal es
dis.).
                try{
                    n=dis.readByte();
                    //Si el missatge es 111, la tasca s'atura.
                    if (n==111){
                        isrunning=false;
```

```
                        bucle=false;
                    }
                    //Si el missatge és 14 (botó catch), es recorre a
la tasca agafar pilota.
                    else if (n==14){
                        agafarpilota(btc);
                    }
                }

                catch(Exception e){
                }

                LCD.clear();

                //A la pantalla es mostra el missatge rebut.
                LCD.drawInt(n, 4, 4);
            }
            //Tancar el canal.
            try {
                dis.close();
            }
            catch(Exception e){}

            try {
                Thread.sleep(100);
            }
            catch (Exception e){

            }
        LCD.clear();
        LCD.drawString("closing",0,0);
        LCD.refresh();
        btc.close();
        LCD.clear();
       }
    }
```

```
    //Tasca agafar pilota
    public static void agafarpilota (NXTConnection btc){
        //Introduim les variables enteres distanceR i distanceL.
        int distanceR, distanceL;
        distanceR=ultrasonicsensorR.getDistance();
        distanceL=ultrasonicsensorL.getDistance();
        //Cas per acudir a la tasca pilota dreta.
        if (distanceR<distanceL){
            pilotadreta(distanceR, btc);
        }
        //Cas per acudir a la tasca pilota esquerra.
        else if (distanceL<distanceR){
            pilotaesquerra(distanceL,btc);
        }
        //Quan no es dona cap dels anteriors casos, s'acudeix a pilota
doble.
        else{
            pilotadoble(distanceL, btc);
        }
    }

    //Tasca pilota dreta.
    public static void pilotadreta (int distance, NXTConnection btc){
        Byte z;
        //Casos segons on estigui de distància la pilota.
        if ((14>=distance)&&(distance>=8)){
            z=100; //posició correcta
            enviarmissatge(z, btc);
            mouregruaR();
        }

        else if (distance<8){
            z=99; //Si és 99 cal anar a l'esquerra.
            enviarmissatge(z, btc);
        }

        else{
```

```
            z=101; //Si és 101 cal anar a la dreta.
            enviarmissatge(z, btc);
        }
    }

    //Tasca pilota esquerra.
    public static void pilotaesquerra (int distance, NXTConnection
btc){
        Byte z;
        //Casos segons on estigui de distància la pilota.
        if ((14>=distance)&&(distance>=8)){
            z=100;
            enviarmissatge(z, btc);
            mouregruaL();
        }

        else if (distance<8){
            z=101;
            enviarmissatge(z, btc);
        }

        else{
            z=99;
            enviarmissatge(z, btc);
        }
    }
    //Tasca enviar missatge per enviar missatge del nxt al mòbil.
    public static void enviarmissatge (Byte n, NXTConnection btc){
          DataOutputStream dis=btc.openDataOutputStream();
        try {
               dis.writeByte(n);
               dis.close();
        }
        catch(Exception e){
        }

    }
```

```
//Tasca per moure grua a la dreta.
public static void mouregruaR(){
    mA.rotate(115-angleinicialA, true);
    mA.waitComplete();
    mB.rotate(-670-angleinicialB, true);
    mB.waitComplete();
    mA.rotate(-25-angleinicialA, true);
    mA.waitComplete();
    mC.rotate(-140-angleinicialC, true);
    mC.waitComplete();
    mB.rotate(670-angleinicialB, true);
    mB.waitComplete();
    mA.rotate(-90-angleinicialA, true);
    mA.waitComplete();
    colorpilota();
}

//Tasca per moure grua esquerra.
public static void mouregruaL(){
    mA.rotate(-115-angleinicialA, true);
    mA.waitComplete();
    mB.rotate(-670-angleinicialB, true);
    mB.waitComplete();
    mA.rotate(25-angleinicialA, true);
    mA.waitComplete();
    mC.rotate(-140-angleinicialC, true);
    mC.waitComplete();
    mB.rotate(670-angleinicialB, true);
    mB.waitComplete();
    mA.rotate(90-angleinicialA, true);
    mA.waitComplete();
    colorpilota();
}
```

```
public static void colorpilota(){
    int resultat;
    resultat=colorsensor.getColorID();
    if (resultat==2){
        pilotablava();
    }
    else if(resultat==0){
        pilotavermella();
    }
    else{
        mA.rotate(-115-angleinicialA, true);
        mA.waitComplete();
        mB.rotate(-670-angleinicialB, true);
        mB.waitComplete();
        mA.rotate(25-angleinicialA, true);
        mA.waitComplete();
        mC.rotate(140-angleinicialC, true);
        mC.waitComplete();
        mB.rotate(670-angleinicialB, true);
        mB.waitComplete();
        mA.rotate(90-angleinicialA, true);
        mA.waitComplete();
    }
}

public static void pilotablava(){
    mA.rotate(30, true);
    mA.waitComplete();
    mC.rotate(45, true);
    mC.waitComplete();
    mC.rotate(-265, true);
    mC.waitComplete();
    mA.rotate(-30, true);
    mA.waitComplete();
}
```

```
    public static void pilotavermella(){
        mA.rotate(-30, true);
        mA.waitComplete();
        mC.rotate(45, true);
        mC.waitComplete();
        mC.rotate(-265, true);
        mC.waitComplete();
        mA.rotate(30, true);
        mA.waitComplete();
    }
}
```