Sincronització amb AIBO

Diego Muñoz Galan

26 de maig de 2014

Resum

Índex

Índex de figures

Índex de taules

1 Prefaci

1.1 Motivació

Fa ja 8 anys que l'empresa Sony va deixar de produir la mascota robòtica per excel·lència, el gos AIBO. Des de la seva desaparició del mercat ha caigut en una espiral de desús fins que avui en dia gran part d'ells estan agafant pols al fons d'un armari.

Aquest projecte va partir de una idea de tornar aprofitar els AIBOS per realitzar tasques de cooperació entre ells partint de la part alta de la programació. En vista de la incomoditat per tal de programar la plataforma en qüestió degut a dos factors el primer: l'us de llenguatges i de APIs poc usades en el mon de la robòtica i que tenen corbes de aprenentatge molt costoses a l'inici. El segon és el mal manteniment d'aquests llenguatges en els últims anys. Això va portar a voler recuperar els AIBO d'una forma nova amb un del sistemes operatius actualment més usats en la robòtica, ROS[1].

La integració de l'AIBO amb ROS facilitarà enormement la programació donant la possibilitat d'usar el gran ventall de paquets existents per aquest sistema operatiu. Per altra banda incrementara les seves capacitats de processament en front a altres mètodes donat que es controlara remotament, exprimint així les capacitats del hardware.

El fet de participar en un projecte que permeti al món de la robòtica educativa recuperar una plataforma tan emblemàtica com l'AIBO és un incentiu suficient per a la realització d'aquest. A més a més el propi aprenentatge intrínsec al projecte com és la oportunitat de conèixer ROS des de baix nivell i implementar els propis paquets necessaris és quelcom útil per a un estudiant apassionat per la robòtica.

1.2 Requeriments previs

2 Introducció

En aquest apartat es definirà tant el punt de partida del projecte, el punt final on es pretén arribar com el camí recorregut i metodologia emprada.

2.1 Estat de l'art

Es tractarà de descriure breument la situació actual de la robòtica i el compromis que requereix aquesta entre hardware i software.

[1] *Robotics Operational System*5.1

2.1.1 Robòtica

Des de la primera definició de robot o des de la idea de robot que va exportar Isaac Asimov ha passat més de mig segle i tot i així es pot dir que la robòtica encara es troba en el seu bressol. De fet és difícil definir quin és l'origen de la robòtica i quin va ser el primer robot, el que està clar és que aquesta branca de la ciència va fer grans avanços arrel de la revolució industrial de la ma de d'altres tecnologies com l'electrònica, la informàtica o la intel·ligència artificial.

En un primer moment els robots van ser maquines automatitzades per tal de realitzar tasques que tinguessin un cert risc, requerissin de més o eficiència que la ma d'obra humana pogués oferir dins de la industria. En aquest àmbit actualment es poden trobar braços manipuladors en la majoria de fabriques i és un fet que aquests incrementen el grau d'automatització de l'industria i amb aquest la seva eficiència[10]. Un exemple d'aquest robots són els robots KUKA[2] com el de la Figura1.

Figura 1: Robot manipulador de KUKA.

Altres sectors apart de la industria han donat un gran impuls a la robòtica i han donat peu a la necessitat de robots mòbils. Aquest és el cas de l'exploració espacial que va tenir els seus primers passos amb el automòbil soviètic tele-operat lunokjod[3] i que ara mateix té el seu màxim exponent amb el robot Curiosity[4] enviat a mart en la ultima missió de la NASA. Aquests robots tenen la senzillesa de que es mouen mitjançant rodes, tanmateix existeixen robots que utilitzen extremitats per moure's. Tot i dificultar el control de l'estabilitat i complicar la dinàmica del sistema aquests ofereixen la possibilitat de caminar per una gran varietat de terrenys; un exemple és el robot quadrúpede de DARPA[5] i

[2] http://www.kuka-robotics.com/en/
[3] http://en.wikipedia.org/wiki/Lunokhod_1
[4] http://www.nasa.gov/mission_pages/msl/
[5] http://www.darpa.mil

Boston Dynamics[6] Wildcat mostrat a la Figura2, l'altra avantatge és l'acceptació social que comporta un robot amb una morfologia familiar per a l'humà.

Figura 2: Robot quadrúpede WildCat.

2.1.2 Robòtica en l'educació

Aquesta ultima característica porta a la relació entre humans i robots donant lloc a robots antropomòrfics com el robot de de HONDA ASIMO[7] o robots mascota com és el cas de la plataforma d'estudi d'aquest projecte, l'AIBO de SONY.

Son plataformes com la que s'ha usat en aquest projecte que han permès a universitats i al públic en general interessat en la robòtica investigar a tots els nivells. Altres plataformes que han apropat la robòtica a un public molt ample permetén tant l'investigació com facilitan eines per a l'educació son els descrites a continuació:

- WIFIBOT: Es tracta d'una serie de robots moguts per quatre rodes amb punt d'accés wifi i que permet ser controlat i programat en varis sistemes operatius i llenguatges[8].
- Lego NXT: És un producte modular que permèt una infinitat de combinacions constructives partint sempre d'un modul de processador. Té el seu propi marc de treball[9].
- NAO: És un robot humanoide de dimensions reduïdes amb prestacions d'alta gama comercialitzat per Aldebaran Robotics [10].

[6] http://www.bostondynamics.com/
[7] http://asimo.honda.com/
[8] http://www.wifibot.com/
[9] http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com
[10] http://www.aldebaran.com/en

- ARDrone:
- DARwin:

Havent parlat d'algunes de les plataformes mes usades cal comentar, degut al compromís entre hardware i software que exigeix la robòtica, els sistemes més usats per crear la intel·ligència artificial.

La majoria de robots tenen el seu propi marc de treball que permet que aquests siguin programat en alt nivell permetent d'aquesta manera tant accedir de forma senzilla a sensors, actuadors i sistemes de comunicacions com treballar amb mòduls de programació ja elaborats amb funcions que faciliten la navegació,les comunicacions o la visió. Tanmateix existeixen serts softwares lliures que s'estan establint com a alternativa als propis de cada plataforma. això facilita enormement la feina del programador ja que no ha d'invertir temps en prepararse per als respectius softwares de cada plataforma amb la que treballi. D'aquests plataformes es poden destacar 2:

- Urbi: del qual es pot trobar informació a la secció 3.2.3 o al annex B.
- ROS: És el sistema operatiu de software lliure més estès actualment. Està en continu desenvolupament i abasta una gran quantitat de plataformes. Consultar la secció 5.1 per més informació.

2.1.3 L'AIBO

Amb la plataforma AIBO secció3 s'han realitzat molts projectes d'investigació degut a la seva versabilitat. Sobre tot s'han realitzat projectes relacionats amb visió[12] o comportaments i intel·ligència artificial[17]. Un punt essencial en el desenvolupament de l'AIBO va ser, en part,que durant el periode comprès entre 1999 i 2008 va ser el robot escollit per a realitzar la competició RoboCup Estandard Plataform League, posteriorment sustituit per la plataforma NAO. La RoboCup una competició internacional destinada a promoure la robòtica i on es realitzen probes com la esmentada anteriorment que forma part de la secció de futbol amb robots autònoms, tanmateix hi han altres probes com la competició de simulacres de rescat on l'AiBO també va ser molt utilitzat[14]. En aquest context de la Robocup va ser una gran motivació per a molts projectes que es poden classificar en 3 grans grups segons la finalitat: Visió i reconeixement de les marques per a la posterior situació del subjecte dins del camp[15], comunicacions i control sobre els moviments[16] i rols que ocupa el ròbot dins del camp[13].

Actualment l'AIBO ja no s'utilitza com es feia avans en part per que ja no és la plataforma estandard de la RoboCup i en part perque va deixar de produir-se al 2006 per SONY i el seu llenguatge base R-CODE va deixar de ser mantingut. A mesura que l'AIBO a quedat en desús els softwares han deixat de tenir-lo en conte, és el cas de Urbi

Secció3.2.3 que apartir de la versió 1.5 la llibreria liburbi no és compatible amb l'Aibo. A més a més la recerca d'aquest projecte i sobretot la manipulació i aprenentatge dels propis llenguatges s'ha vist dificultada per la desaparició de la xarxa de gran part de la seva documentació.

2.2 Objectius del projecte

L'objectiu d'aquest projecte és recuperar l'AIBO com a plataforma educativa aportant-li una capa superior de programació que permeti la integració de la plataforma amb ROS. D'aquesta manera el processament de les dades del robot es farà externament. Amb aquest fi es comparant varis camins per tal de buscar la forma optima de realitzar-ho. Com a objectius concrets i en ordre de necessitat es poden marcar:

- Buscar la millor opció per tal de comunicació amb l'AIBO, entenent com a millor un equilibri entre senzillesa i qualitat de la comunicació. Això comportarà per una banda un anàlisi qualitatiu sobre els possibles mètodes i un un anàlisi més exhaustiu dels camins que passin la primera selecció.
- La creació d'un paquet de ROS que permeti extreure totes les variables de l'AIBO en forma de tòpics i alhora controlar les posicions de les articulacions de forma senzilla mitjançant la publicació en un altre tòpic.
- Demostració i comprovació del grau de control aconseguit mitjançant un petit exemple de sincronització.

2.3 Abast del projecte

3 AIBO ERS-7

Al 1999 Sony va llençar al mercat la mascota robòtica AIBO (Artificial Intelligence roBOt, amic en japonès). Aibo és un robot amb forma de gos que va ser ideat per tal d'interaccionar amb el seu amo com una mascota real, és capaç de percebre els estímuls del mon exterior mitjançant una serie de sensors. Aibo porta incorporat un software anomenat AIBO MINT dins d'una targeta de memòria especial per a Aibo, és aquest software el que li permet comportar-se com un gos real, interaccionar amb el seu amo, reconèixer visualment i auditivament i aprendre.

3.1 Hardware

Sony ha desenvolupat diversos models i versions d'Aibo i Aibo Mint, millorant tant els actuadors i sensors com el software. Entre els diferents models amb el que es treballarà

d'ara en endavant és el model ERS-7 que es caracteritza per tenir una càmera de major resolució i un processador més potent.

Figura 3: AIBO ERS-7.

L'Aibo ERS-7 té les següents característiques[15]:

- Àudio:
  - Entrada d'àudio: Micròfon estèreo
  - Sortida d'àudio: Altaveu de 20.8 mm i 500 mW
- Sensors integrats:
  - Sensors de pressió:
    - 1 sobre el cap
    - 3 a l'esquena
  - 1 sensor de contacte a cada pota
  - 1 sensor bolea sota la boca
  - Acceleròmetres en x,y i z
  - 2 sensors de proximitat d'infrarojos situats al morro i al pit
  - Sensor de vibració
- Graus de llibertat
  - 3 graus de llibertat a cada una de les 4 potes
  - 3 graus de llibertat al coll
  - 1 grau de llibertat a cada orella
  - 1 grau de llibertat a la boca
  - 2 graus de llibertat a la cua
- Entrada d'imatge
  - Sensor d'imatge CMOS de 350000 píxels

  - Angles: 56.9º horitzontal i 45.2º vertical
  - Resolucions: 208x160, 104x80, 52x40
  - 30 frames per segon

- Dimensions
  - Altura: 278 mm
  - Llarg: 319 mm
  - Ample: 180 mm
  - Pes amb bateria: 1.65 Kg

- CPU:
  - Processador: MIPS R7000 @ 576 MHz
  - RAM: 64 MB
  - Memòria flaix: 4 MB

- Connectivitat:
  - LAN inalàmbric IEEE 802.11b
  - Xifrat: WEP

3.2 El software

APERIOS és el sistema operatiu en temps real que utilitzen els AIBO. APERIOS OS està destinat y dissenyat per poder treballar amb grans fluxos de dades d'àudio y d'imatge simultàniament en temps real. En un inici AIBO va ser comercialitzat amb una finalitat purament lúdica, però degut al seu atractiu disseny i la seva robustesa va atreure l'atenció dels desenvolupadors per a convertir-se en una poderosa eina per a l'aprenentatge.Això va portar a Sony a publicar un API del seu sistema operatiu anomenada OPEN-R SDK. Arrel d'aquesta API van sorgir altres llenguatges i interfícies com son *URBI*[11] o *Tekkotsu*[12] que posen una capa de programació per sobre de OPEN-R i permeten programar a més alt nivell.

3.2.1 OPEN-R

OPEN-R és un API en C++ que corre sobre el sistema APERIOS proporcionada per Sony i l'estructura de la cual és la que es mostra a la figura 4. OPEN-R diferencia entre

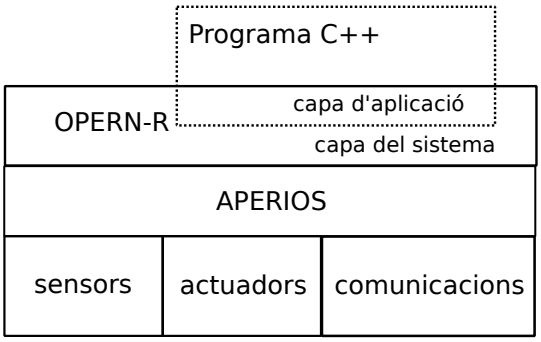

Figura 4: Arquitectura de l'AIBO amb OPEN-R

dos nivells, la capa de sistema on s'accedeix al hardware del robot i la capa d'aplicació que es tracta dels programes fets per l'usuari.

En tractar-se d'un llenguatge de baix nivell és inherent a l'estructura del sistema operatiu que es basa en objectes que interaccionen entre si mitjançant missatges o meta-objectes.El concepte d'objecte és semblant al utilitzat a *UNIX*[13] o a *Windows*[14], amb la diferencia de que són mono-fil i que la comunicació és realitza per missatges que inclouen dades i un identificador que indica el metde en el que s'executarà en arribar a l'objecte destí. Això implica que cada objecte té varis punts d'entrada i sortida que són els mètodes: DoInit(), DoStart(), DoStop(), DoDestroy(). En el pas de missatges entre dos objectes, un d'ells es defineix com a subjecte(el que envia) i l'altre com observador(el que rep) qui inicia la seva execució després de que el missatge enviat faci saltar l'esdeveniment del mètode corresponent[11].

SUBJECTE	**OBSERVADOR**
enviament de dades	
notificació de l'esdeveniment	
	recepció de dades
	esdeveniment preparat

Taula 1: Estructura del pas de missatges a OPEN-R

Una de les majors complicacións alhora de treballar amb OPEN-R és que treballa amb una gran varietat d'arxius que cal modificar per tal de configurar el programa d'aplicacio que es realitzi. Per això és important conèixer bé els arxius amb que es treballa, els més importants es troben llistats a continuació:

- Arxius .h i .cc: són els arxius de programació en C++.

[11] `www.urbiforge.org`
[12] `tekkotsu.org`
[13] `www.unix.org`
[14] `http://windows.microsoft.com/en-us/windows/home`

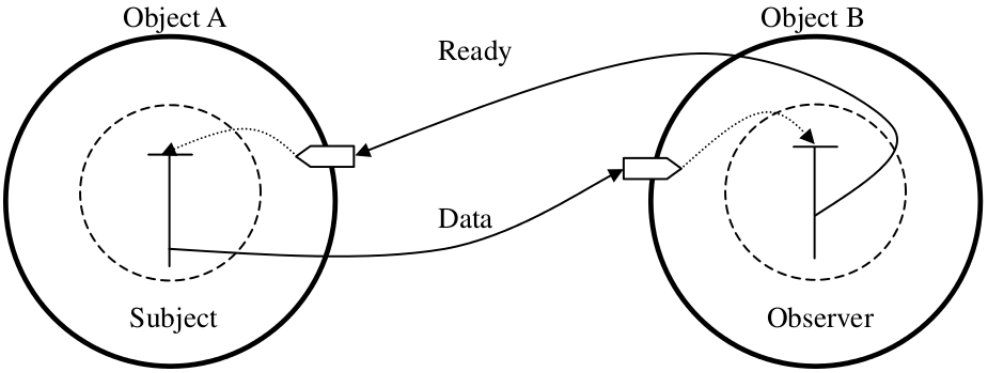

Figura 5: Comunicació entre objectes.

- STUB.config: són arxius de configuració on es defineix com l'objecte es comunica amb altres objectes.
- Arxius .ocf: defineix el comportament dels objectes en quan a temps d'execució, memòria assignada i prioritat d'execució.
- Makefile: és el fitxer per a la configuració global de compilació.
- OBJECT.cfg: és el llistat d'objectes que s'executarà.
- CONECT.cfg: on es determinen les connexions i comunicacions entre els objectes que s'executaran.
- WLANCONFIG.txt[15] arxiu on es defineixen les característiques de la connexió inalàmbrica de l'AIBO.

3.2.2 Tekkotsu

Tekkotsu és una software de desenvolupament per a robots mobils. Originalment va ser creat per al robot AIBO de Sony, però actualment permet programar una amplia gama de robots com són el Chiara[16], iRobot Create[17], HandEye[18] o Lynxmotion Arms[19]. Té una arquitectura basada en objectes i pas d'esdeveniments, igual que OPER-R fent un us complet de l'herència de C++. Proporciona una capa de major nivell que OPEN-R, però permet la crida a OPEN-R, el que significa que la capacitat de control sobre el robot és trobi limitada pel propi hardware i no per aquest software, permetent l'accés als

[15] Aquest ultim archiu cal configurar-lo sigui quin sigui el mètode de programació del AIBO. Veure APENDIX
[16] `http://chiara-robot.org`
[17] `http://chiara-robot.org/Create/`
[18] `http://chiara-robot.org/HandEye/`
[19] `http://www.lynxmotion.com/`

sensors, complet control sobre els actuadors i la comunicacio basada en esdeveniments. Per altra banda permet a l'usuari poder treballar a alt nivell ja que el paquet inclou eines de processament visual bàsic, monitorització, models cinemàtica inversa i suport per a la creació de xarxes inalambriques.

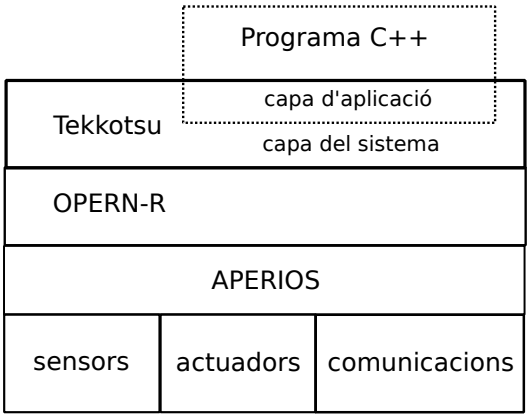

Figura 6: Arquitectura de l'AIBO amb tekkotsu.

A més Tekkotsu proporciona una interficie d'usuari, el ControllerGUI, que permet accedir al AIBO remotament i executar els comportaments que es desitgin i que estiguin programats a la targeta de memoria. Aixo tan es pot fer per la interficie grafica basada en java com per telnet[20] al port 10020[18].

Tekkotsu s'organitza com un conjunt de comportaments o "Behaviorsï classes MotionCommand. Les seves funcions s'executen en dos processos el "Main i el "Motion". En el primer s'encarrega de la percepció i la presa de decisions i el segon fa referencia al control en temps real dels actuadors. Existeix un tercer procés que s'encarrega de la sortida d'àudio[19].

Igual que en OPEN-R, els comportaments, que son com s'anomena a qualsevol programa amb Tekkotsu, tenen una estructura basada en uns mètodes que cal respectar (doStart(), doStop()) que simplifica'n els 4 mètodes definits a OPEN-R, però que permeten mantenir l'estructura dels objectes i la comunicació entre ells basada en esdeveniments.

3.2.3 URBI

URBI és l'acrònim de Universal Real-Time Behavior Interface i es tracta d'una plataforma de software lliure per controlar robots i sistemes complexos en general com AIBO, Bioloid Mindstorm NXT, Pioneer, Wifibot o ARDrone[20]. URBI és un llenguatge basat en scripts d'alt nivell amb l'avantatge de que permet executar comandes en paral·lel. per treballar amb URBI existeixen dos maneres, una es tracta d'utilitzar el script per després incloure

[20] `http://es.wikipedia.org/wiki/Telnet`

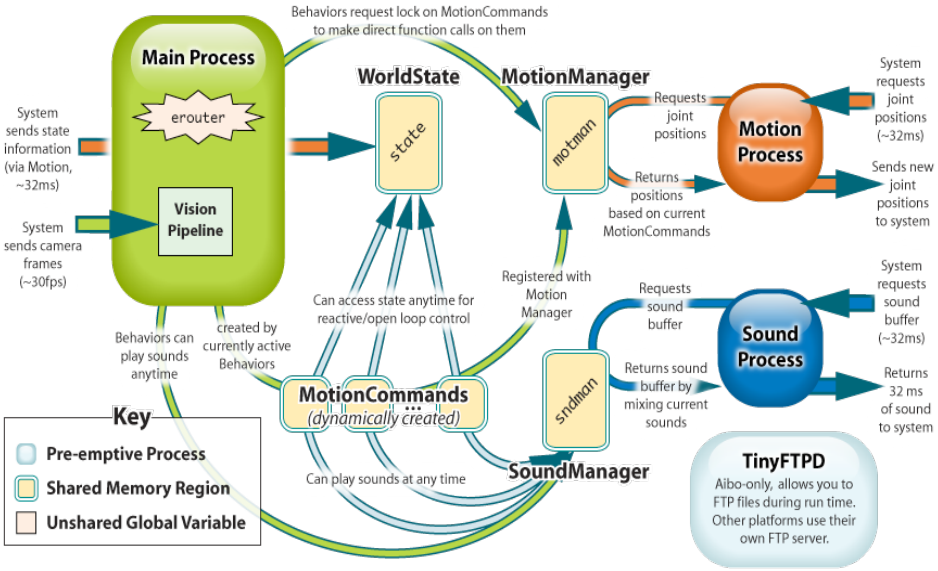

Figura 7: Procés d'execució d'un programa amb Tekkotsu.

un objecte d'OPEN-R a la targeta de memoria de l'AIBO que el propi sistema operatiu APERIOS pot interpretar i executar. La segona manera consisteix en una arquitectura client/servidor on el servidor es tracta com un objecte OPEN-R que va dins del robot i el client envia comandes que son executades per el servidor, Figura 8.

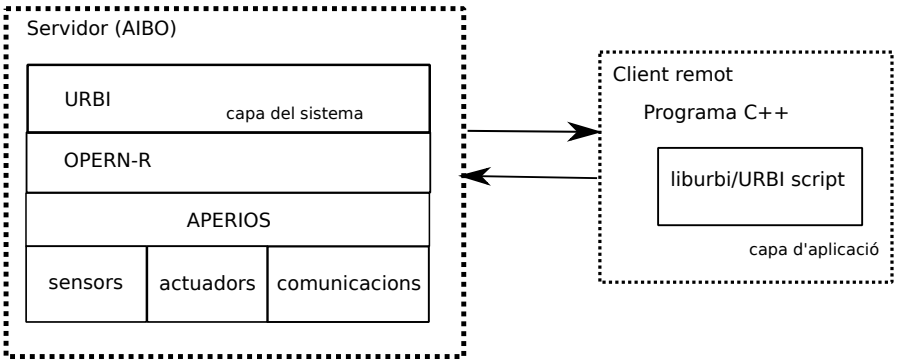

Figura 8: Arquitectura de l'AIBO amb URBI.

Aquesta segona opció obre varies vies per a la programació. Permet, per una banda comunicar-se per telnet i enviar les comandes o bé usar la llibreria liburbi que permet

treballar amb C++, Java i Matlab o varis llenguatges alhora mitjançant varis clients. URBI permet i facilita l'accés a cada una de les articulacions i sensors de l'AIBO, per exemple per consultar el valor de la primera articulació de la cama esquerre del devant cal enviar la comanda `legLF1.val;` i per assignar el seu valor `legLF1.val=20;`[21]. A l'apèndixB és pot trobar més documentació sobre les comandes d'URBI.

4 Comparació d'alternatives per a la programació

Tenint en conte que el que és preten és buscar una forma senzilla i rapida de enviar l'estat dels sensors i articulacions de l'AIBO a un PC remot i enviar a l'AIBO les posicions desitjades de les articulacions per tal de poder treballar posteriorment amb ROS s'han tingut en conte certes consideracions qualitatives un trobades durant la recerca que es troben resumides a la Taula 2.

En els tres casos és necessari programar amb el llenguatge corresponent per al client del PC. A excepció de Urbi seria necessari desenvolupar un programa per a l'AIBO de manera que permetés enviar i rebre les dades necessaries. Amb Urbi no és necessari ja que la llibreria liburbi permet accedir a totes les dades necessaries i també ho permet fer per telnet. Tekkotsu en canvi permet obtenir els valors de sensors i actuadors per telnet però no actuar sobre ells caldria doncs un comportament que implementés aquest fet. De totes formes Tekkotsu hi han hagut problemes per tal de trobar documentació ja que s'ha trobat per a la versió més nova Tekkotsu 5.1 i tot i que Tekkotsu va neixer arrel de l'AIBO aquesta versió sembala no ser compatible amb el ultim OPEN-R SDK. La unica versió que s'ha conseguit compilar i la única trobada apart de la 5.1 és la 4.0.1 de la qual no s'ha trobat cap mena de documentació i d'una a l'altre canvia el mètode de programació. Pel que respecte a R-CODE seria una bona opció ja que es parteix de més baix nivell i per a l'objectiu d'aquest projecte això és un incentiu, però cal invertir molt de temps en l'aprenentatge del llenguatge.

	OPEN-R	**Urbi**	**Tekkotsu**
Necessitat de un programa a l'AIBO	Si	No	Si
Necessitat d'un programa al PC	Si	Si	Si
Permet consultar l'estat per telnet	No	Si	Si
Permet accionar articulacions per telnet	No	Si	No
Dificultat de programació (0-5)	5	2	3
Documentació (0-5)	2	3	1
Plataforma del PC	Windows/ Linux/ OS	Windows/ Linux 32bits/ OS	Linux

Taula 2: Comparació entre llenguatges usats en l'AIBO

Per aquests motius s'ha decidit partir de dos opcions de per a desenvolupar el paquet de ROS que permetrà controlar l'AIBO.

La primera s'usara liburbi en un programa C++ i la segona es provarà de fer la comunicació amb un script de python usant una connexió telnet.

4.1 Lectura de dades

En un primer experiment per comparar les possibles solucions s'ha realitzat la lectura d'una variable del sistema per tal de valorar quantitativament la velocitat de transmissió. S'ha usat el valor de la posició d'una articulació.

El procediment per a l'adquisició de dades serà el mostrat a la Taula 3.

CLIENT	**SERVIDOR**
demanda d'inici del bucle d'enviament	
lectura de les dades	enviament de dades
Escriptura en archiu extern	enviament de dades

Taula 3: Estructura de la lectura d'una variable

Es consulta el temps en el que es rep cada dada reben aquest temps amb precisió de nanosegons i escribint-lo en un archiu de text per al seu posterior tractament.

Es realitzen 15 repliques per a cada cas i cada una d'elles tindrà una durada de 10 segons per tal de poder estimar de forma precisa quina és la millor opció per tal de rebre les dades de L'AIBO.

4.1.1 Metode amb liburbi i C++

Usant la llibreria liburbi es tracta d'enviar per un client urbi la comanda per tal de rebre les dades de la primera articulació de la cama esquerre. L'estructura del programa és la que segueix[21].

- Inicialitzacó del client URBI.
- Inicialització del callback. Aquest serà cridat cada cop que es rebi una dada per a l'etiqueta assignada.
  - Es pren el valor de la variable (tot i que per aquest experiment no es necessiti.).
  - Es consulta el temps en que ha estat rebuda aquesta dada.
  - Es guarda la dada i el temps amb precisió de centenars de nanosegons.
- Demanda del bucle i assignació a una etiqueta.
- inicialització del archiu on es guradaràn les dades.
- Execució del bucle de urbi.

[21] El script es pot trobar a l'annexe C.1

4.1.2 Metode amb telnet i python

La idea d'aquest script[22] es treballar amb urbi com si fos la una terminal de urbi 9 però mitjançant python per tant l'estructura bàsica serà enviar la comanda i llegir la resposta per a que pugui ser tractada. Així doncs l'estructura serà semblant al script de C++:

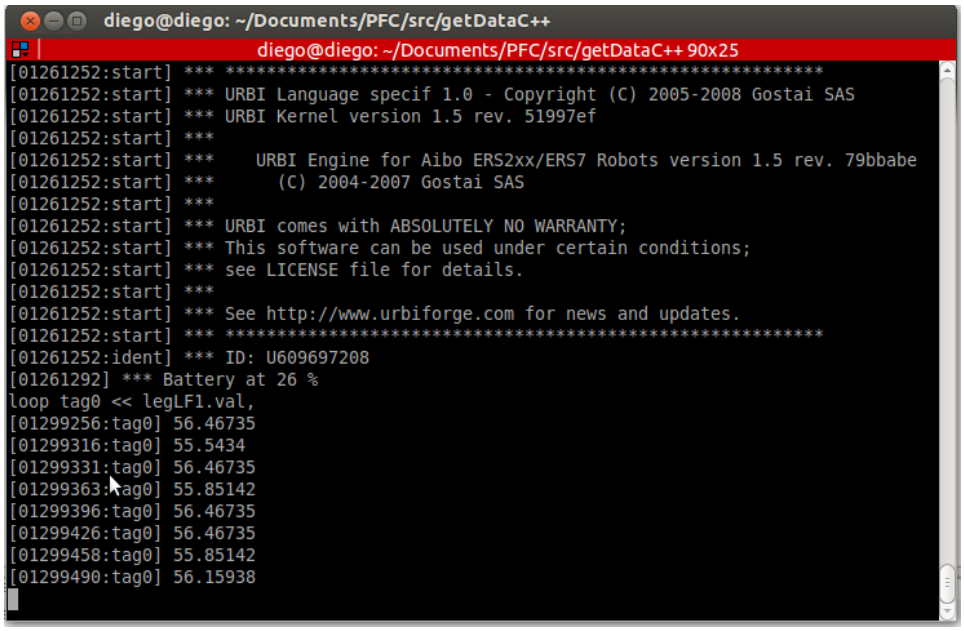

Figura 9: terminal de telnet consultant dades.

- Inicialització de l'objecte telnet.
- Petit retràs per esperar la connexió.
- Lectura i eliminació de la capsalera del terminal urbi rebuda pel terminal.
- Escriptura de la comanda del bucle d'enviament del valor de l'articulació més una variable concreta per facilitar una marca per al tractament de les dades rebudes.
- Bucle de lectura i escriptura.
  - Tractament de les dades:
    - Lectura fins a trobar la marca.
    - Eliminació de les cadenes rebudes tallades per la informació de la bateria[23]
    - Separació de la cadena de caràcters per agafar únicament el valor de l'articulació.

[22] Es pot cunsultar el script a lannexe C.2
[23] Cada poc temps la terminal de urbi escriu el percentatge de bateria restant.

- Escriptura de la dada a l'arxiu.
- Adquisició del temps.
- Escriptura del temps a l'arxiu.

4.1.3 Comparacó

Dels programes anteriorment explicats s'obtenen les 15 repliques de 10 segons cada una. Dels temps extrets s'obté l'invers de les diferències de temps entre dades rebudes, doncs és més intuïtiu i per analitzar el trafic i la velocitats de dades es preferible treballar amb Hertz que amb el seu invers.

De les dades obtingudes se n'extreuen els grafics de les Figures 10 i 11. D'aquests es pot extreure com la variabilitat dels resultats, dins de cada rèplica, del cas en que s'usa liburbi és més alta, però també els valors semblen ser considerablement més alts.

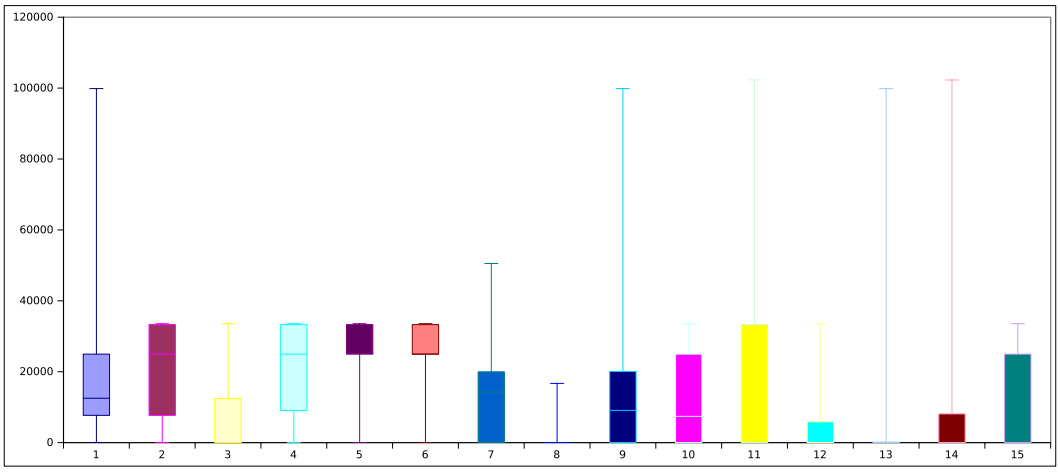

Figura 10: Diagrama de caixa de la freqüència de dades per a les 15 repliques de l'experiment amb liburbi.

Per corroborar aquests dos fets s'ha pres una serie de variables caràcteristiques de cada mostra.Les Figures 13 i 12 mostres de forma més visual aquest fet i permeten afirmar que el primer cas te una velocitat de transmissió de dades mitja més elevat, però la seva variabilitat també és significativament major.

Tanmateix no és sols la velocitat mitja el que interessa sinó que sigui suficientment constant i sobre tot que no hi hagi els temps de congelació en el refresc d'una variable siguin el mes petit possible. Observant doncs les freqüències mínimes Figura14 no sembla que en algun dels dos casos aquestes siguin inferiors altres. Si bé es cert que els buits més grans es donen amb liburbi realitzant un anàlisi sobre la variança (ANOVA) no es pot afirmar que en aquest aspecte siguin poblacions diferents.

En vista dels resultats obtinguts s'ha volgut comprovar com afecta el nombre de dades en la recepció de les variables. Per això s'ha realitzat el mateix experiment però en comptes de llegir únicament el valor d'una articulació, s'envien els valors de totes les articulacions.

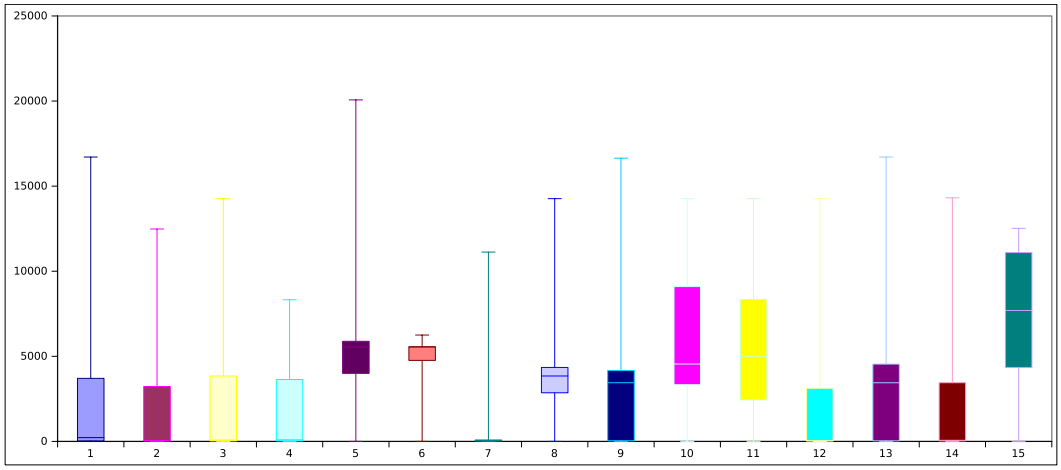

Figura 11: Diagrama de caixa de la freqüència de dades per a les 15 repliques de l'experiment amb telnet.

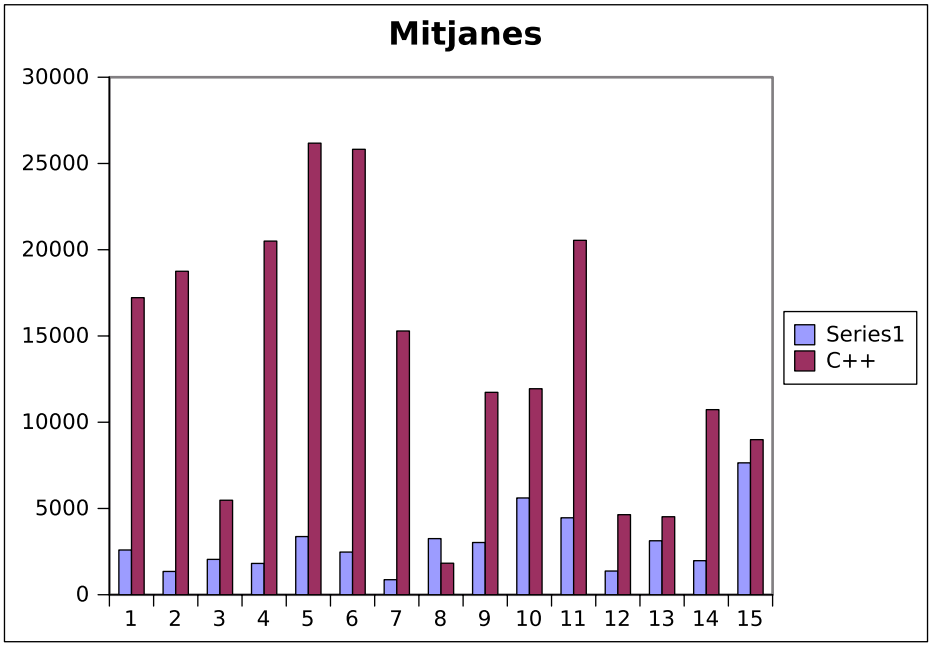

Figura 12: Freqüència mitjana en Hz de les 15 repliques de cada mètode.

Els resultats obtinguts en diagrames de caixes Figures 15 i 16 mostren com les dos mostres son més semblants entre els dos casos. El temps maxim d'adquisició de dades del metode amb telnet ha incrementat fins a igualarse amb el de liburbi. La diferència pot ser que amb aquest mètode es rep una trama que es tracta en paral·lel a la recepció de dades mentre que el cas de liburbi es un sol callback que es va disparant per a cada valor i per tant el cas de telnet li costarà el mateix rebre totes les dades que una mentre que amb liburbi no.

Les mitjanes semblen lleugerament més altes les del programa de liburbi Figura17 tot i que no hi ha les diferències de l'experiment anterior i pel que fa a les desviacions seblen força semblants.

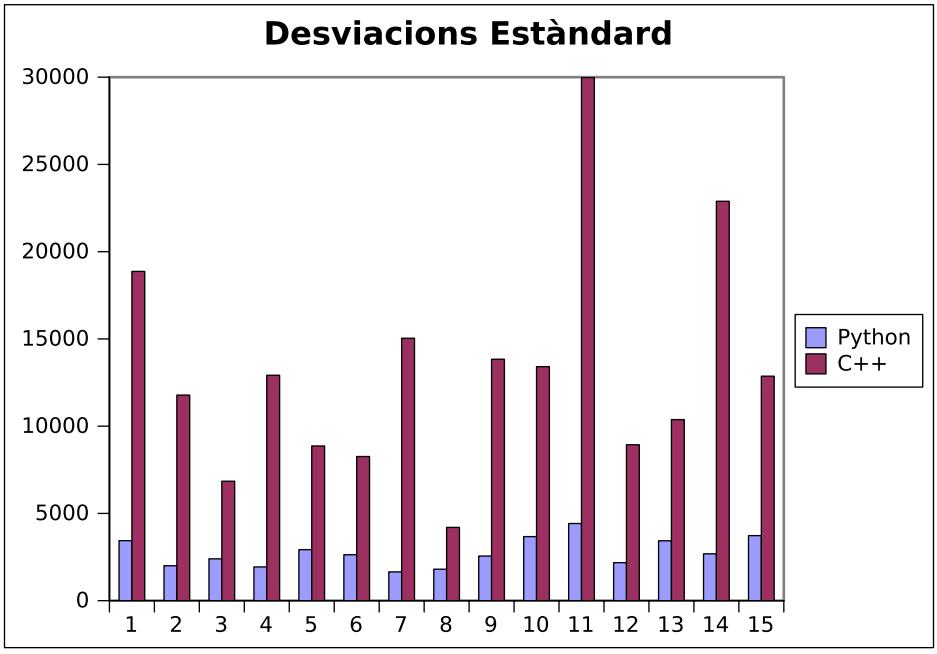

Figura 13: Desviacions estàndard de la freqüència en les 15 repliques dels dos mètodes.

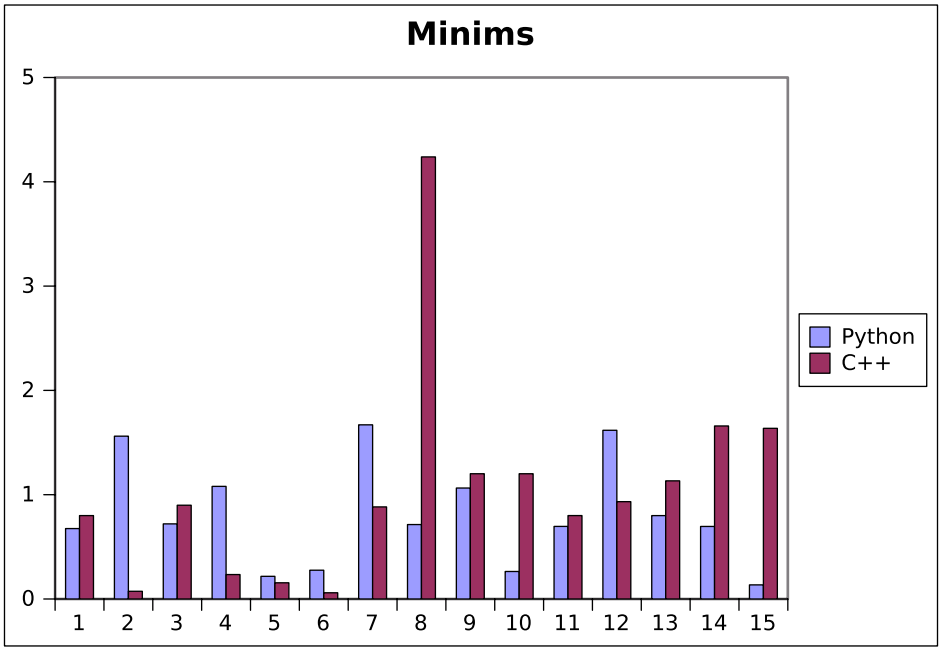

Figura 14: Mínims de la freqüència en les 15 repliques dels dos mètodes.

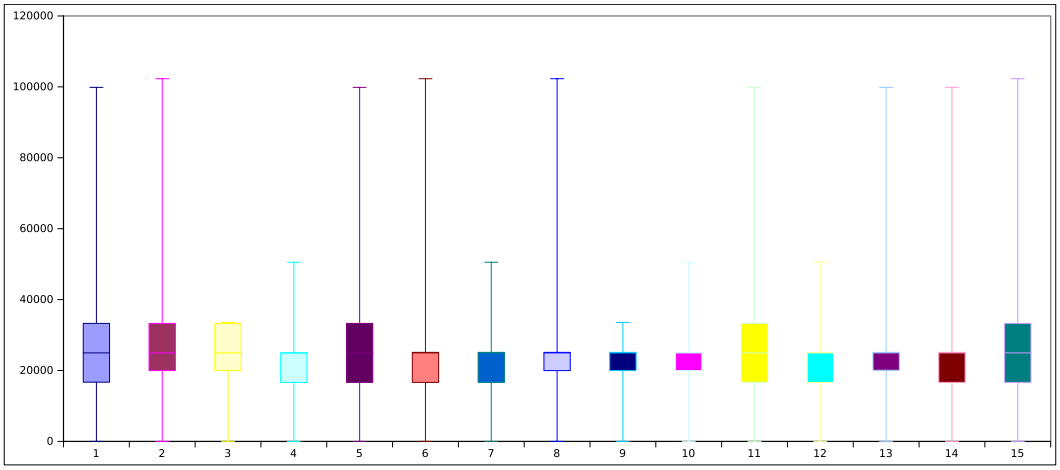

Figura 15: Diagrama de caixa de la freqüència de dades per a les 15 repliques de l'experiment amb liburbi.

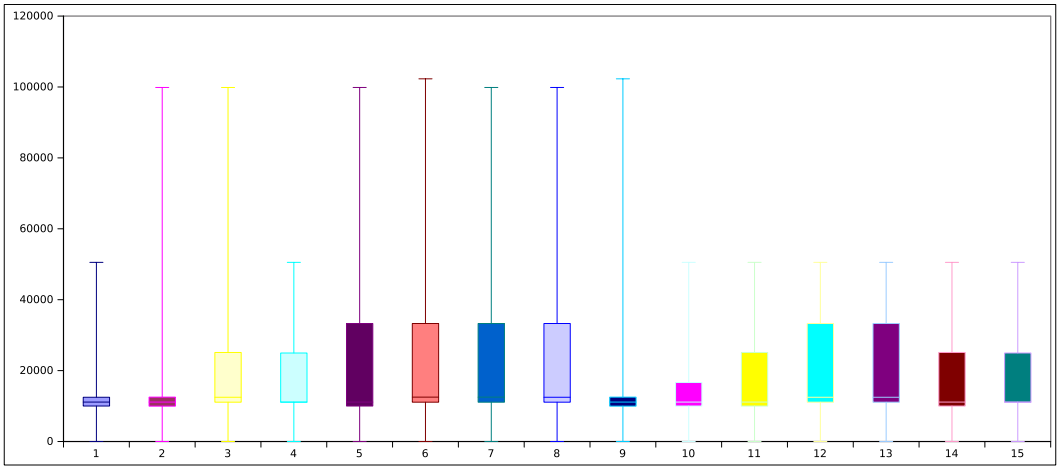

Figura 16: Diagrama de caixa de la freqüència de dades per a les 15 repliques de l'experiment amb telnet.

4.2 enviament de dades

4.3 Elecció del mètode

4.4 Possibles millores

5 Implementació del paquet de ROS

5.1 ROS

5.2 Paquet AiboServer

5.3 Paquet AiboMimic

6 Cooperació i Sincronització

Conclusions

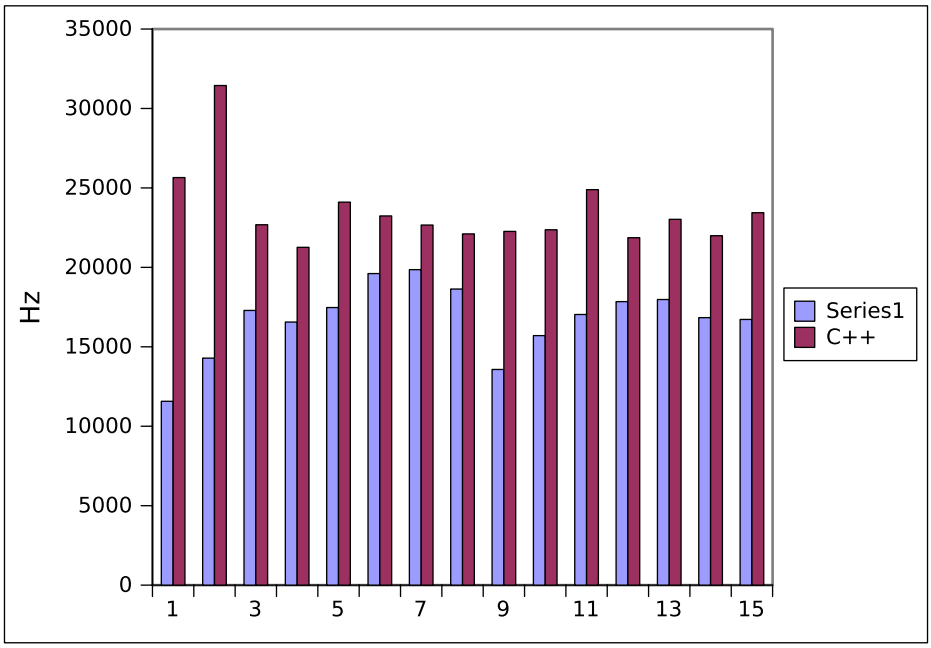

Figura 17: Freqüència mitjana en Hz de les 15 repliques de cada mètode.

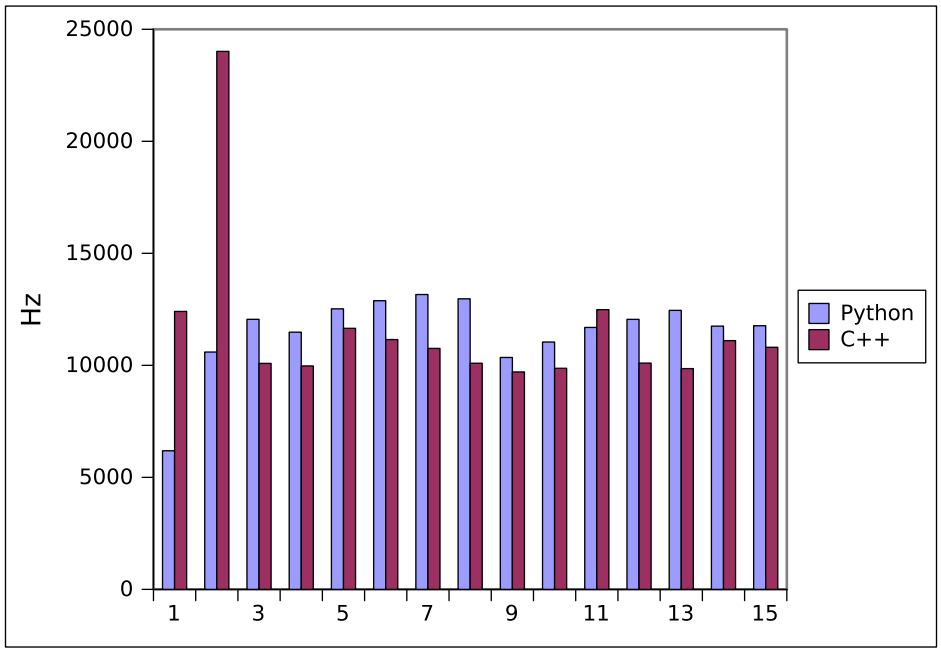

Figura 18: Desviacions estàndard de la freqüència en les 15 repliques dels dos mètodes.

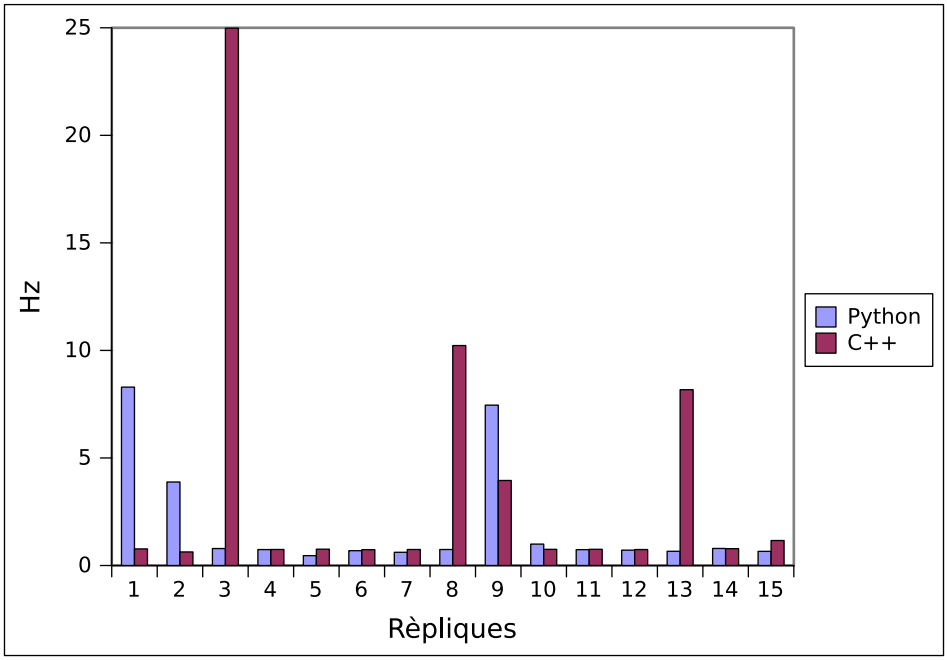

Figura 19: Mínims de la freqüència en les 15 repliques dels dos mètodes.

Replica	Mitjana	Desviació estàndard	Màxim	Minim
1	17220.69	18868.55	99864.38	0.8
2	18751.08	11781.05	33554.43	0.07
3	5478.21	6845.21	33554.43	0.9
4	20500.1	12917.91	33554.43	0.23
5	26177.68	8864.36	33554.43	0.16
6	25822.19	8260.97	33554.43	0.06
7	15288.78	15038.49	50533.78	0.88
8	1826.91	4198.85	16710.37	4.24
9	11732.41	13836.05	99864.38	1.2
10	11941.76	13410.92	33554.43	1.2
11	20545.05	29980.55	102300.1	0.8
12	4640.01	8933.51	33554.432	0.93
13	4520.6	10376.95	99864.38	1.13
14	10729.39	22888.43	102300.1	1.66
15	8989.98	12864.7	33554.43	1.64

Taula 4: dades de C++

Replica	Mitjana	Desviació estàndard	Màxim	Minim
1	17220.69	18868.55	99864.38	0.8
2	18751.08	11781.05	33554.43	0.07
3	5478.21	6845.21	33554.43	0.9
4	20500.1	12917.91	33554.43	0.23
5	26177.68	8864.36	33554.43	0.16
6	25822.19	8260.97	33554.43	0.06
7	15288.78	15038.49	50533.78	0.88
8	1826.91	4198.85	16710.37	4.24
9	11732.41	13836.05	99864.38	1.2
10	11941.76	13410.92	33554.43	1.2
11	20545.05	29980.55	102300.1	0.8
12	4640.01	8933.51	33554.432	0.93
13	4520.6	10376.95	99864.38	1.13
14	10729.39	22888.43	102300.1	1.66
15	8989.98	12864.7	33554.43	1.64

Taula 5: dades de C++

Referències

[1] *KUKA Robotics*, http://www.kuka-robotics.com

[2] *Telnet*, http://www.telnet.org/

[3] *National Aeronautics and Space Administration*, NASA. http://www.nasa.gov/

[4] *Defense Advanced Research Projects Agency*, DARPA. http://www.darpa.mil

[5] *Boston Dinamics*, http://www.bostondynamics.com/

[6] *ASIMO*, The world most advanced Humanoid Robot. HONDA http://asimo.honda.com/

[7] *Wifibot*, http://www.wifibot.com/

[8] *Lego Mindstorm NXT*, http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com

[9] *Adebaran Robotics*, NAO http://www.aldebaran.com/en

[10] CEA, Comité Español de Automàtica, *El libro blanco de la robòtica en España*. 2011.

[11] Sony Corporation, *OPEN-R Programmer's Guide*. 2004.

[12] Xavier Perez, *Vision-based Navigation and Reinforcement Learning Path Finding for Social Robots*. 2010.

[13] Ángel Montero Mora, Gustavo Méndez Muñoz, José Ramon Dominguez Rodriguez, *Metodologías de diseño de comportamientos para AIBO ERS7*. 2009.

[14] RoboCup, http://www.robocup2014.org/

[15] Jesus Morales, *Localización de objetos y posicionamiento en el escenario de RoboCup Four-Legged con un robot AIBO*. 2007.

[16] Jesús Martinez Gómez, *Diseño de un teleoperador con detección de colisiones para robots AIBO*. 2006.

[17] Ricardo A. Tellez, *Aibo Programming*. http://www.ouroboros.org/aibo.html

[18] Ethan Tira-Thompson, *Tekkotsu Quick Reference*, ERS-7, Tekkotsu 3.0.

[19] David S. Touretzky and Ethan J. Tira-Thompson, *Exploring Tekkotsu Programming on Mobile Robots*. Carnegie Mellon University, 2010. http://www.cs.cmu.edu/~dst/Tekkotsu/Tutorial/contents.shtml

[20] *URBI*, `http://www.urbiforge.org/index.php/Main/Robots`

[21] Jean-Christophe Baillie, *URBI Doc for Aibo ERS2xx ERS7 Devices documentation.* 2005.

Annexos

A Configuració dels marcs de treball

A.1 URBI

A.2 Tekkotsu

A.3 OPEN-R SDK

A.4 Configuració wifi

B URBI

C Scripts i programes

C.1 Obtenció dels valors d'una variable usant liburbi per C++

```
1
#include <urbi/uobject.hh>
3 #include <urbi/uclient.hh>
#include <stdio.h>
5 #include <time.h>
#include <iostream>
7 #include <fstream>

9 std::ofstream myfile;

11 void getTime(double& tim)
{
13
   struct timespec tsp;
15
   clock_gettime(CLOCK_REALTIME, &tsp);   //Call clock_gettime to fill tsp
17


19   tim = (double)(tsp.tv_sec+ tsp.tv_nsec*0.000000001);

21  return;
}
23 void saveData( double tim,double value){


25
```

```
  myfile << tim;
  myfile << "\t";
  myfile << value;
  myfile << "\n";
  return;
}
urbi::UCallbackAction onJointSensor(const urbi::UMessage &msg)
  {
  double value = (double)msg.value->val;
  if (msg.tag=="legRF1")

    std::cout << value <<std::flush;
    std::cout << "\n"<<std::flush;
    std::cout.precision(18);
    double tim;

    getTime(tim);
     std::cout << tim <<std::flush;
   std::cout << "\n" <<std::flush;
    saveData(tim,value);


  return urbi::URBI_CONTINUE;
  }
int main(int argc, char** argv)
{
  urbi::UClient* client = new urbi::UClient(argv[1], 54000);
  std::cout << "start"<<std::flush;
  client->setCallback (urbi::callback(onJointSensor),"legRF1");
  client->send("loop legRF1 << legRF1.val,");
  std::cout << "send"<<std::flush;
  myfile.open ("Data1.txt");
  myfile.precision(15);
  urbi::execute();
  myfile.close();

  std::cout << "finish"<<std::flush;

  return(0);
}
```

src/getDataOneLeg/getDataC++/getData.cpp

C.2 Obtenció dels valors d'una variable usant una conexió telnet i python

```
import telnetlib
import time
def tractarData(data):
  datos=[]
  while (data.find("555555")<0):
    data=data+te.read_some()
  if (data.find("Bat")<0):
    #print "dataNow"
    dataNow=data.split("555555")[0]
    #print dataNow
    #print "sobras"
    data=data.split("555555")[1]
    #print data
    dataArray=dataNow.split("\n")
    del dataArray[0]
    #print "dataArray"
    #print dataArray
    leg=0
    for i in range(len(dataArray)):
      if (dataArray[i].find("tag0")>0):
        #print dataArray[i].split("]")[1]
        leg=float(dataArray[i].split("]")[1])
        print "----------------------\n"
        print leg


    return leg, data
  else:
    #print "nfweonfoiwen"
    data=""
    return 0, data
def writeData(f, text):

  f.write(text) # Write a string to a file


te=telnetlib.Telnet("192.168.0.125",54000)
time.sleep(1)
data=1
while data:
  data=te.read_eager()
  print data
te.write("motors on;\r\n")
```

```
te.write("motors.load=0;\r\n")

print "........................"

te.write("loop tag0 << legLF1.val, loop tag19 << 5*111111;")

dataArray, data=tractarData(data)

f = open("Data.txt", "w")
while True:

  leg, data =tractarData(data)
  t= str(leg) + "\t"
  writeData(f,t)
  sec = "{0:.15f}".format(time.time()) + "\n"
  writeData(f,sec)
  print sec

f.close()
```

src/getDataOneLeg/getDataPython/getData.py

C.3 Obtenció dels valors de totes les articulacions usant liburbi per C++

C.4 Obtenció dels valors de totes les articulacions usant una conexió telnet i python

C.5 Moviment d'una articulació de forma sinusoïdal usant liburbi per C++

C.6 Moviment d'una articulació de forma sinusoïdal usant una conecció telnet i python