Disseny i desenvolupament d'una plataforma robòtica educativa

Memòria del projecte
d'Enginyeria Tècnica en
Informàtica de Sistemes
realitzat per
Domènec Castillo Castells
i dirigit per
Salvador Alcántara Cano

Escola d'Enginyeria
Sabadell, Juny de 2012

El sotasignat, Salvador Alcántara Cano,
professor de l'Escola d'Enginyeria de la UAB,

CERTIFICA:

Que el treball al que correspon la present memòria
ha estat realitzat sota la seva direcció per
Domènec Castillo Castells
I per a que consti firma la present.
Sabadell, Juny de 2012

Signat: ***Salvador Alcántara Cano***

FULL DE RESUM – PROJECTE FI DE CARRERA DE L'ESCOLA D'ENGINYERIA

Títol del projecte: Desenvolupament d'una plataforma robòtica educativa	
Autor[a]: Domènec Castillo Castells	**Data:** Juny 2012
Tutor: Salvador Alcántara Cano	
Titulació: Enginyeria Tècnica en Informàtica de Sistemes	
Paraules clau (mínim 3) • Català: Robot, Android, WebSockets, placa IOIO, educatiu, Python, electrònica. • Castellà: Robot, Android, WebSockets, placa IOIO, educativo, Python, electrónica. • Anglès: Robot, Android, WebSockets, IOIO board, educational Python.	

Resum del projecte (extensió màxima 100 paraules)

• Català:

La finalitat del projecte, com el nom indica, tracta de dissenyar i desenvolupar una plataforma robòtica educativa, que pugui ser controlada des d'un entorn amigable i accessible des del major nombre de situacions possible, evitant en la mesura del que sigui possible, dependre d'un sistema operatiu concret. Així mateix es busca desenvolupar un projecte que sigui fàcilment reproduïble amb el mínim cost econòmic.

• Castellà:

La finalitat del proyecto, como su nombre indica, trata de disseñar i desarrollar una plataforma robótica educativa, que pueda ser controlada des de un entorno amigable i accesible des del mayor número de situaciones possible, evitando en la medida de lo possible, depender de un sistema operativo concreto. Asimismo se busca desarrollar un proyecto que sea fácilmente reproducible con el mínimo coste económico.

• Anglès:

The project's goal, as its name suggests, is to dessign and develop an eductional robotic platform that can be controlled from a friendly environment and attainable from most situations, avoiding as far as possible, depending on a particular operating system. Likewise, seeks to develop a project that is easy to reproduce with minimal economical cost.

Índex

1. Introducció

En aquest capítol presentarem la idea general del projecte, així com un resum general de l'estructura de la memòria i la manera de treballar fins arribar a l'objectiu final de la solució.

1.1. Objectiu del projecte

L'objectiu d'aquest projecte és el de crear una plataforma robòtica, és a dir, un petit robot amb sistema de locomoció i diferents sensors per tal de recollir dades de l'entorn i poder interactuar amb ell.

Com a requisits principals, es busca un sistema que sigui multi-plataforma, i fàcil d'interactuar, per tal de que pugui ser accessible pel màxim número de persones.

També es busca un sistema fàcil de reproduir, on es necessiti poca inversió per tal de tenir-lo en funcionament, així serà accessible pel major nombre d'usuaris i poder ser aplicat en la majoria dels àmbits de l'educació o el lleure.

1.2. Descripció del projecte

El sistema en sí està separat en diferent parts.

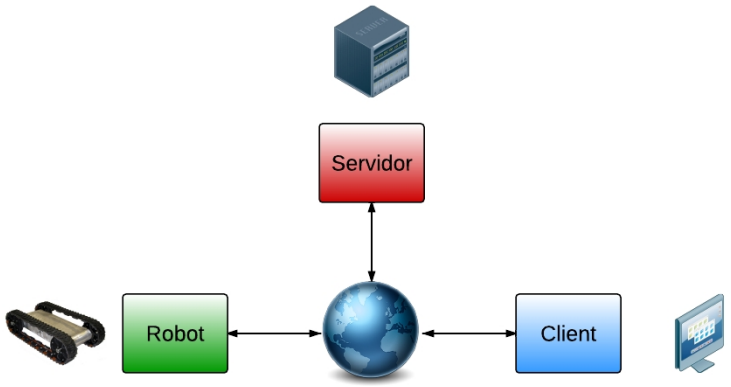


Figura 1 – Esquema del sistema.

Per una banda tenim la plataforma física del robot, que ha de ser controlada remotament, per poder dirigir-la des de qualsevol lloc. Aquest hardware ha d'estar controlat per un microcontrolador o CPU, capaç de poder-se comunicar sense fils amb un servidor que li enviï les ordres, que és una altra de les parts.

El servidor, es comunicarà tant amb el robot com amb el programa client on interactuarà l'usuari per rebre informació i donar instruccions a la plataforma. Ha de ser capaç d'oferir una connexió ràpida i bidireccional (*full-dúplex*) per tal de controlar al robot en temps real, si no fos així, derivaria en un mal funcionament o en el pitjor dels casos la pèrdua de control de la plataforma, que pot malmetre'n els components.

El programa client (*Front end)* serà el responsable d'interactuar amb el robot de manera directa, ja sigui manualment o per la via de codi de programació, per assignar-li diferents accions en resposta a certs esdeveniments.

La manera més eficient i pràctica per poder arribar a la solució, es tractarà a l'estudi de viabilitat, on es buscaran tots els recursos i mecanismes que més s'adaptin al nostre problema.

1.3. Motivacions personals

Un dels meus *hobbies* des de la meva infància ha estat la robòtica. Durant la carrera d'Enginyeria Tècnica Informàtica de Sistemes he pogut interactuar dins de diferents àmbits, tant a nivell de hardware i electrònica, com a nivell de programació. Per tal de poder aplicar tots aquests coneixements adquirits en un sol projecte que requereix de tots ells, va sorgir la idea de la plataforma robòtica encarada a l'àmbit educatiu. El fet d'encarar-la a l'educació es deu a l'alt preu dels paquets robòtics disponibles a la venda per portar projectes a l'àmbit empíric i testejar-ne els seus comportaments. Aproximadament els kits de robots programables oscil·len en uns 300€, un preu que es pretén abaratir considerablement després d'aquest projecte.

Imatge 1 – Robot Lego Mindstorm.

Imatge 2 – Robot Boebot.

Espero que amb aquest sistema es puguin iniciar força persones en el món de la robòtica, on interactuen diverses disciplines, en alguns casos, d'una complexitat extrema. També espero que sigui la base de futurs projectes a desenvolupar, ja que proporcionarà un sistema sòlid i versàtil.

1.4. Estructura de la memòria

La present memòria s'estructura en nou capítols, cadascun d'ells subdividit amb els seus apartats que descriuen tota la feina feta en el projecte.

En el primer capítol introduïm el projecte en general, amb els seus objectius, la metodologia utilitzada i l'estructura de la memòria.

En el segon capítol realitzem l'estudi de viabilitat, on s'aprofundeix en els objectius i les parts del projecte, l'estat de l'art, i quina és la millor manera de portar-los a terme.

En el tercer capítol descriurem la plataforma robòtica, les parts que la componen, física i electrònica, i la programació utilitzada en ella.

En el quart capítol ens centrem en el servidor del projecte, utilitzat per rebre i transmetre la informació entre client i robot, i els mecanismes de programació utilitzats per desenvolupar-lo.

En el cinquè capítol es descriu el programa client desenvolupat per a un navegador web, tots els llenguatges de programació utilitzats durant el procés, així com les seves funcionalitats.

El sisè capítol descriu totes les proves que s'han realitzar per desenvolupar el projecte i la metodologia de posada en marxa del sistema.

El setè capítol presenta les conclusions extretes de tot el projecte, amb opinions objectives i subjectives.

El vuitè capítol conté tota la bibliografia utilitzada per desenvolupar el projecte, des de llibres a recursos d'Internet.

Finalment el novè capítol conté un annex, on hi ha un índex d'imatges i la descripció dels continguts del CD que s'entrega amb aquesta memòria.

1.5. Metodologia de desenvolupament

La metodologia de desenvolupament que s'ha seguit per a la consecució d'aquest projecte ha sigut evolutiva i seqüencial, dividida en diferents etapes, donada la naturalesa experimental d'aquest.

Inicialment s'ha realitzat un estudi de viabilitat per determinar si el projecte es pot dur a terme, i quina és la millor manera de fer-lo. Un cop fet l'anàlisi es procedeix al disseny i construcció de la plataforma robòtica, la seva intercomunicació amb el servidor, i aquest amb el programa client.

Finalment es duent a terme els diferents test que posen a prova el sistema, així com el desenvolupament de la documentació del projecte.

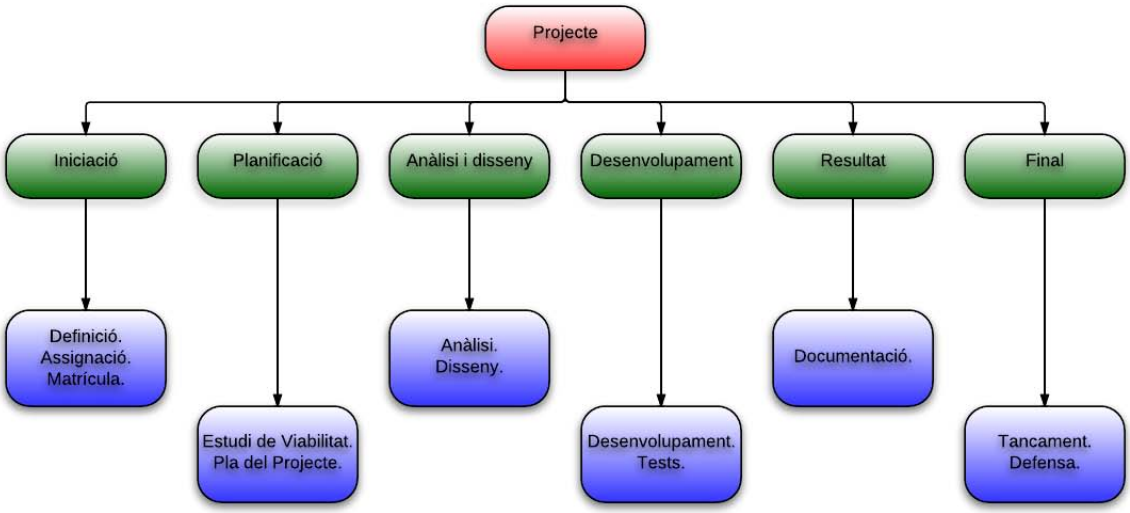


Figura 2 – Metodologia de desenvolupament.

2. Estudi de viabilitat

2.1. Introducció

L'estudi de viabilitat ens permetrà determinar si el projecte es realitzable o no. A més hem d'escollir els mecanismes per buscar la solució al problema que tractem. Un mal estudi de viabilitat ens pot portar pèrdues o en el pitjor dels casos el no assoliment de la solució al projecte i el fracàs.

Inicialment descriurem la situació a tractar, així com les parts descrites anteriorment i les possibles solucions per cada una d'aquestes, la plataforma robòtica, el servidor i el programa client.

Un cop decidides les millors solucions per cada apartat, es procedirà a calcular el cost del projecte, així com a estructurar i planificar-ne el desenvolupament. El desenvolupament del projecte s'haurà d'adequar al calendari planificat.

2.2. Descripció de la situació a tractar

El sistema en sí és senzill, un robot mecànic i amb un sistema de locomoció capaç de rebre ordres a temps real inalàmbricament.

Un servidor per rebre i transmetre informació tant al robot com al programa client (*Back end*) amb la major velocitat possible, asíncrona i bidireccionalment.

El programa client, capaç de rebre la informació dels sensors del robot i transmetre les ordres.

En el projecte, volem controlar el robot inalàmbricament, per tant la manera més viable d'aconseguir un bon control i aprofitament de les noves tecnologies actuals, és fer passar les comunicacions per una xarxa, per tal de poder-lo dirigir des del màxim nombre de zones possibles.

Per tal de poder establir un bon compliment dels requisits del sistema i poder arribar a una bona solució, primer aclarirem els objectius.

2.3. Objectius

Els objectius del projecte són els següents :

1. Crítics
   - Creació d'una plataforma robòtica amb sistema de locomoció i comunicació inalàmbrica.
   - Programar la plataforma robòtica per executar les ordres del servidor i enviar informació.
   - Creació d'un servidor remot que es comuniqui amb el robot i el programa client.
2. Prioritaris
   - Aconseguir una comunicació en temps real i asíncrona entre les diferents parts.
   - Crear un programa client que es connecti al servidor remot per poder controlar al robot.
   - Dotar al programa client amb la capacitat d'executar codi extern per poder crear programes de control.
   - Baix cost econòmic per al desenvolupament.
   - Aplicació multi-plataforma
3. Secundaris
   - Utilització d'un sensor d'orientació.
   - Utilització d'un sensor GPS.
   - Utilització d'un giroscopi o sensor de nivell.
   - Utilització d'un sensor d'imatge.

2.4. Estat de l'art

En aquest apartat s'exposa l'estat de l'art de cada una de les parts i quina és la millor manera de desenvolupar-la.

Donat que un dels objectius principals del sistema és la comunicació en temps real, el primer que hem de fer és decidir quin mecanisme de transmissió de dades serà el responsable de la comunicació, ja que pot arribar a marcar tota la resta del projecte.

Seguidament buscarem les millors solucions per a cada una de les parts del projecte.

2.4.1. Interfície de comunicació del sistema

Ens centrarem ara en la recerca del millor mètode per poder comunicar les nostres diferents parts del projecte : robot, servidor i client.

Donat que la part principal del projecte recau en la plataforma robòtica, hem de trobar un sistema de comunicació que compleixi els objectius de baix cost i que sigui compatible amb la majoria de protocols existents. Així doncs, després d'escollir la millor opció per la comunicació inalàmbrica del robot, ens centrarem en el sistema escollit per les altres parts, el servidor i el programa client, que vindrà donat per l'elecció d'aquest últim.

2.4.1.1. Comunicació del robot amb el sistema

En els últims temps, la telefonia mòbil ha experimentat una evolució enorme, convertint un simple telèfon mòbil en tot un centre multimèdia. La majoria de telèfons mòbils tenen connexió *WIFI*, de xarxa mòbil *3GS*, *Bluetooth*... capaços de connectar-se a Internet per comunicar-se amb el món. A més, a nivell de programació poden acceptar la majoria de protocols de comunicació, *TCP-IP*, *UDP*, utilització de *Sockets*, *WebSockets*, *HTTP*...

Les seves característiques de hardware fan que amb les connexions pertinents es puguin connectar a sistemes físics per tal de controlar-los, ja sigui mitjançant *Bluetooth*, el port sèrie *USB*, *WIFI*...

La seva potència de processament, a més, ens permet realitzar operacions de càlcul a gran velocitat, ja sigui per tractar imatges, so, equacions complexes... les possibilitats són infinites.

Dins la gama dels mòbils actuals, ens trobem amb tres sistemes operatius principals :

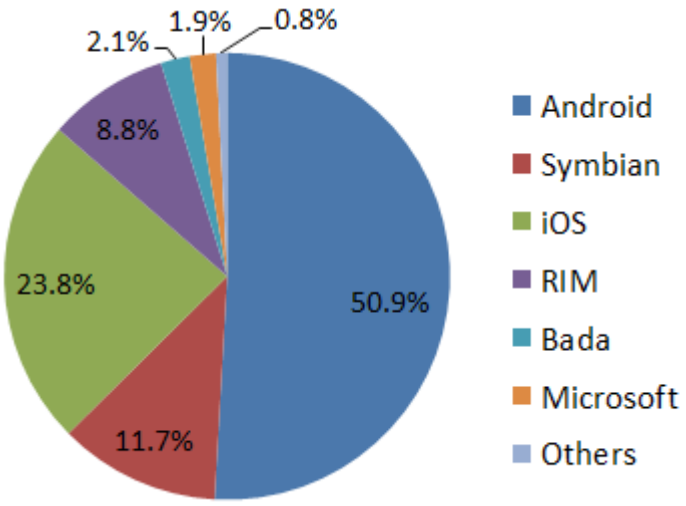


Figura 3 – Utilització dels Sistemes Operatius en SmartPhones.

Aquests són :

- *IOS*, el sistema operatiu dels mòbils d'*Apple*, el *IPhone*.
- *Android*, el sistema operatiu de *Google* que compta amb un ampli ventall de marques de telèfon mòbil.
- *Symbian*, el sistema operatiu de la companyia *Nokia*.

Donat el nostre projecte on es busca un baix cost econòmic, i per tant la utilització de sistemes de codi lliure, juntament amb les recents notícies i eines de desenvolupament utilitzant el seu sistema, i l'amplia cobertura de mercat, el sistema operatiu escollit per desenvolupar el projecte és *Android*.

La decisió del telèfon mòbil en el projecte es deu a que abaratirà enormement els components físics per a la comunicació inalàmbrica, que poden ser realment costosos, a més, la seva àmplia gama de sensors ens dóna una bona base de partida per transmetre informació de l'entorn.

L'adquisició d'un telèfon amb sistema operatiu *Android* és realment senzilla, ja que el mercat de la telefonia ofereix nombroses ofertes i més de la meitat del usuaris el tenen. A més com que no és necessari un mòbil amb una gran potència encara obre més el ventall de possibilitats per a la seva adquisició. D'altra banda també promovem el reciclatge dels terminals donant-ne un possible ús al final de la seva vida útil.

Cal destacar que el mòbil es podrà utilitzar amb la plataforma robòtica sense la necessitat de modificar-lo, possibilitant així habilitar l'ús del terminal amb el projecte i fora d'ell.

Imatge 3 – Logotip del sistema Android.

2.4.1.2. Comunicació global del sistema

En quant a l'àmbit de les comunicacions de les diferents parts, el protocol *TCP-IP* és el més viable, ja que és el més estès mundialment en la majoria dels sistemes, i ens permet fer funcionar la plataforma educativa ja sigui dins una xarxa interna, present en la majoria d'escoles, llars i empreses, o bé a través de la xarxa d'*Internet*.

Ara bé, donat que la comunicació del sistema ha de ser asíncrona i bidireccional, i que hem de proporcionar un sistema multi-plataforma pel projecte, la opció més coherent és la utilització de *Sockets* ja sigui utilitzant el protocol *TCP* o *UDP*. Donada la naturalesa del nostre projecte, la comunicació més ràpida ens la dóna el protocol *UDP*, però aquesta ens limita l'àmbit d'interacció, obligant-nos a crear un programa extern capaç de rebre els paquets d'informació.

La solució definitiva en quant al programa client seria l'ús d'un navegador web per tal de comunicar-se amb el servidor. D'aquesta manera ens evitem la utilització de diferents programes segons el sistema operatiu o la plataforma on es trobi el servidor, a més dels avantatges de tenir un codi interpretat i que no fa falta compilar, accessible des de qualsevol xarxa i en la majoria dels entorns.

Els protocols que intervenen quan s'utilitza un navegador web, solen ser principalment el protocol *HTTP*, *HTTPS*, *FTP* i recentment *WS*.

Mitjançant l'ús de *Javascript* i utilitzant la tecnologia *AJAX*, podem ser capaços de proporcionar una comunicació asíncrona, si ens quedem escoltant al servidor, utilitzant el protocol *HTTP*. La contrapartida és que sempre necessitem rebre una resposta i s'han de programar diverses línies de codi per poder aconseguir una bona comunicació.

Una tecnologia recent és la dels *WebSockets*. Els *WebSockets* ens permeten establir un canal de comunicació asíncrona i Full-Dúplex utilitzant un *Socket TCP*. En el nostre cas, aquesta tecnologia s'adapta perfectament al nostre projecte, i ens permet utilitzar navegadors web pel programa client.

Estudiem ara el mecanisme dels *WebSockets* en profunditat i la seva possible adaptació per al nostre sistema.

2.4.1.3. WebSockets

Com ja hem dit anteriorment, els *WebSockets* ens permeten una comunicació asíncrona i full-dúplex per mitja d'un *Socket TCP*. A més obren una connexió persistent entre client-servidor, de manera que un cop feta la connexió podrem rebre i enviar sense necessitat de tornar a connectar.

Ara estudiarem les compatibilitats que existeixen sobre les nostres diferents parts del sistema i els *WebSockets*.

Malgrat que és una tecnologia recent, està sent regulada per la *W3C* (*Web Applications Working Group*) i té una gran acceptació dins el món de les comunicacions web. La regulació del protocol ha proporcionat una *API* principal ja integrada en els últims navegadors, que utilitza *Javascript*. Els navegadors compatibles són :

- *Mozilla Firefox* 4 o superior
- *Google Chrome* 4 o superior
- *Internet Explorer* 10 o superior
- *Opera* 11 o superior
- *Opera* en mòbils 11 o superior
- *Safari* 5 o superior
- *Safari* dins del *IOS* 4.2

Web Sockets - **Working Draft**

Bidirectional communication technology for web apps

Resources: WebSockets information | Chromium blog post | Wikipedia

Global user stats*:

Support:	22.29%
Partial support:	13.57%
Total:	35.86%

	IE	Firefox	Safari	Chrome	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser	
Two versions back	7.0	3.5	3.2	10.0	10.6	3.2			2.1	
Previous version	8.0	3.6	4.0	11.0	11.0	4.0-4.1		10.0	2.2	
Current	9.0	4.0	5.0	12.0	11.1	4.2-4.3	5.0-6.0	11.0	2.3	3.0
Near future		5.0	5.1	13.0	11.5					
Farther future	10.0	6.0		14.0	12.0					

Note: Firefox 4, Opera 11 and Opera Mobile 11 will have their support disabled by default, due to an unresolved protocol-level security issue. This may occur in other browsers as well. Support can be manually enabled. Microsoft is currently experimenting with the technology.

Feedback

Imatge 4 – Compatibilitat de WebSockets segons el navegador web.

Ja tenim doncs la millor comunicació possible per part del programa client, ara falta veure si la podrem aplicar tant en el servidor com en el robot. Hi ha diverses llibreries desenvolupades en diferents llenguatges de programació, que ens podrien adaptar la comunicació en el nostre sistema.

Anem a veure doncs la possible comunicació amb el robot, és a dir, amb *Android*. Quan es tracta d'una tecnologia nova i amb un sistema en concret com és el cas d'*Android*, les llibreries a utilitzar són escasses :

- *jWebSocket*
  - Està escrita en *Java* i *Javascript.* Ens serveix per comunicar-nos per mitjà de navegadors web, però el seu suport per *Android* és limitat i no ofereix una solució robusta de cara a fer una comunicació de tipus servidor.
- *Websocket-android-phonegap*
  - És una altra implementació per utilitzar conjuntament amb navegadors web, però clarament encarada a *HTML*, a més s'utilitza conjuntament amb la *API Phonegap*, que ens força a escriure tot el programa en *HTML*, *CSS* i *Javascript*, i converteix el codi a una aplicació *Android*.
- *Autobahn WebSockets*
  - Ofereix suport per els últims estàndards, versions de la llibreria per *Android*, *Javascript* i *Python*, a més d'una excel·lent documentació i diversos tests i exemples.

De les tres opcions ens quedem amb *Autobahn WebSockets*, ja que el codi està a lliure disposició sota la llicència *Apache* i no ens limita a tractar amb entorns *HTML*. A més gràcies a la seva implementació amb *Python* ens permetrà programar el servidor de manera fàcil, i donades les característiques de *Python*, de manera multi-plataforma. Pot córrer de manera nativa sota *Android* i segueix els últims estàndards, ja que és el projecte en paral·lel d'una versió comercial, pel que ens assegurem continuïtat i suport de les llibreries.

Així doncs, ja tenim el sistema de comunicació del projecte, que ens permetrà comunicar les tres parts del projecte d'una manera robusta i persistent, i complint tots els objectius que buscàvem.

Imatge 5 – Logo d'Autobahn WebSockets.

2.4.2. Hardware del robot

Ja que hem decidit utilitzar el sistema operatiu *Android* per portar a terme el desenvolupament sistema, ara ens toca buscar els possibles mecanismes de hardware compatibles amb el terminal que podem fer servir per implementar la plataforma robòtica. Necessitem doncs un microcontrolador o placa específica que pugui comunicar-se amb *Android* a través d'una connexió sèrie, *WIFI* o *Bluetooth*, donades les limitacions hardware del mòbil.

Veiem doncs les possibles alternatives per integrar el sistema del robot amb el telèfon *Android* .

2.4.2.1. Microcontrolador

Un microcontrolador és un circuit integrat programable. Ens permet instal·lar-li un codi per fer certes accions en resposta a certs esdeveniments. Com a exemple, podríem parlar del famosos microcontrolador de la família *PIC*, entre d'altres.

Imatge 6 – Microcontrolador Pic 16F84A.

Per comunicar el nostre microcontrolador amb el telèfon mòbil, hauríem de proveir-lo d'un sistema adaptat a la comunicació serie del *USB* del terminal com és l'estàndard *RS232*. També podríem afegir-li comunicació *WIFI* o *Bluetooth*, però totes aquestes mesures portarien una enorme feina, ja que per poder parlar amb el mòbil per *USB* per exemple, hauríem d'adaptar-nos al estàndard d'*Android*, que només permet comunicació en mode debugable, amb el protocol *ADB* (*Android Debug Bridge*). Això és complicat, ja que hem de fer que el nostre microcontrolador funcioni com a *Host*. Es podria aconseguir fer funcionar, però el projecte s'allargaria enormement desenvolupant el robot, i no podríem centrar-nos en el que realment ens preocupa.

2.4.2.2. Arduino

Arduino és una plataforma electrònica oberta, de lliure ús, que compta amb una àmplia comunitat. Bàsicament és un microcontrolador muntat sobre una placa ja preparada per la comunicació amb l'ordinador i les opcions més generals que ofereixen aquests, com *I2C*, *PWM*, *AC/DC*...

Es connecta a l'ordinador per mitjà d'un *USB* i es programa amb el llenguatge *C* utilitzant un *IDE* multi-plataforma programat en *Java.* El seu ús es simple, versàtil, i ràpid. Permet desenvolupar infinitat de projectes fàcilment, i compta amb una extensa recopilació de tutorials i projectes a la web. Depenent del tipus de projecte a desenvolupar, o segons la potència que necessitem, hi ha diferents models de la placa disponibles, com *Arduino Nano* o *Arduino MEGA*.

Imatge 7 – Placa Arduino MEGA.

Com hem dit abans, no el podem connectar directament al mòbil amb la seva comunicació en sèrie, ja que ens hem d'adaptar al llenguatge debugable d'*Android*, i hem d'actuar com a *Host USB*.

Les alternatives per la comunicació amb el terminal utilitzant la placa *Arduino* com a base són les següents .

2.4.2.2.1. Amarino

Amarino és el nom d'unes llibreries preparades per treballar específicament amb *Android* i *Arduino*. Estan encarades a utilitzar un mòdul *Bluetooth* com per exemple el *BlueSMIRF Gold*. S'instal·la un programa servidor a l'*Android* per fer de pont i intèrpret amb un programa que es pot desenvolupar amb la seva llibreria i que pugui fer les accions pertinents.

Després de fer alguns test, he pogut comprovar que utilitzar *Amarino* amb *Android* i *Arduino*, concretament amb el mòdul *Bluetooth BlueSMIRF Gold*, ha funcionat perfectament i amb bons resultats.

La contrapartida és l'elevat preu d'un mòdul *Bluetooth* per connectar-lo a *Arduino*, que no bé amb la placa, i que oscil·la entre els 50€. A més la idea de dependre d'un programa pont per poder executar el teu programa, la limitació de velocitat de 56KB (és més que suficient pel nostre projecte), i les possibles interferències que es poden donar, quan el robot i el mòbil estan a uns 5 centímetres (ja que el terminal estarà a la plataforma robòtica) fan que sigui una opció inviable pels objectius del nostre projecte.

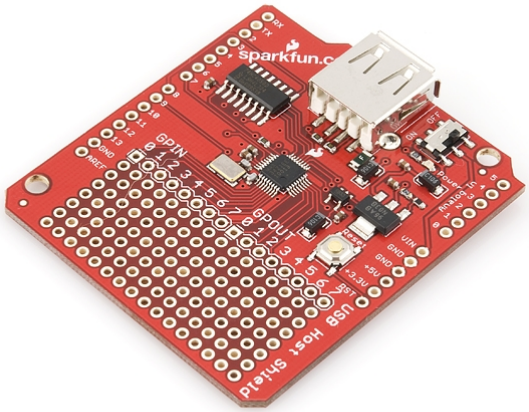
Així mateix, la opció del *WIFI* té les mateixes similituds en quant a ser un mètode no apropiat pel nostre projecte. Hem decidit escollir el telèfon mòbil per abaratir costos, seria absurd integrar un mòdul *WIFI* en el microcontrolador per poder parlar amb el mòbil.

Imatge 8 – Logo del projecte Amarino.

2.4.2.2.2. Microbridge

És una implementació del protocol *ADB* per microcontroladors. Malgrat hem situat a l'*Arduino* i a *Microbridge* junts, aquest últim no necessita exclusivament de la placa *Arduino* per poder parlar amb *Android*. Podríem utilitzar un altre microcontrolador, però donades les seves facilitats, utilitzarem l'*Arduino*.

Microbridge són un conjunt de llibreries, que compten amb una versió per l'*IDE* d'*Arduino* (en llenguatge *C*) per poder parlar amb *Android*. Per tal d'establir la comunicació, necessitem fer servir l'*Arduino* com un *Host USB*, per això s'utilitza conjuntament amb un altre mòdul anomenat *USB Host Shield*, que ens permet fer precisament això.



Imatge 9 – Placa USB Host Shield.

Després de fer servir un *Arduino Duemilanove* i una placa *USB Host Shield*, he pogut comprovar que les llibreries funcionen satisfactòriament. De moment és la millor opció pel projecte, però durant la seva utilització en conjunt amb algunes funcions específiques d'*Android*, m'he topat amb que perdia la connexió i no era prou robusta. Aquest fet pot ser catastròfic durant la utilització del projecte, ja que pot fer que perdem el control sobre la plataforma robot fent inútil la finalitat del projecte.

2.4.2.3. IOIO

IOIO és una placa dissenyada per treballar específicament amb *Android*. Implementa el protocol *ADB* i és controlada directament des de l'aplicació del mòbil, és a dir, que no necessitem programar la placa. *IOIO Manager* és una completa *API* que s'inclou a l'aplicació que volem desenvolupar i ens permet controlar totes les funcionalitats de les que disposa un microcontrolador, això és, entrades i sortides digitals i analògiques, *PWM*, *I2C*, *SPI*, *UART*...

Imatge 10 – Placa IOIO.

El millor de la placa és que en un sol mòdul tenim tant el microcontrolador com la connexió al terminal mòbil, a més la controlem des d'*Android*, per tant ja no hem de programar la placa. També abarateix el cost, ja que el seu preu ronda uns 40€, preu inferior del que costaria una placa *Arduino* i el mòdul de comunicació amb *Android*.

Després de fer diverses proves utilitzant la extensa documentació i tutorials, he pogut constatar que és relativament senzill utilitzar les seves capacitats, a més proveeix una comunicació robusta sense desconnexions.

2.4.2.4. Decisió final

Després de comprovar els pros i els contres de totes les possibilitats per connectar el mòbil *Android* a un hardware físic per poder controlar el robot, la opció més econòmica i pràctica és la d'utilitzar la placa *IOIO*. La seva testejada *API*, juntament amb la reducció de costos i la compatibilitat amb el 99% dels dispositius *Android*, fa que sigui la solució òptima.

Amb aquesta placa podem controlar qualsevol dispositiu electrònic des d'una aplicació d'*Android*, donant-li una gran versatilitat per infinitat de projectes, cosa que la fa idònia per el nostre projecte.

2.4.2.5. Components mecànics del robot

El nostre robot necessita desplaçar-se per poder interactuar amb l'entorn, per això li proporcionarem un sistema de locomoció capaç de fer-ho, és a dir, motors.

Ens centrarem en dos tipus de motors donada la baixa complexitat de moviment que necessitem pel nostre robot, motors de corrent continua i servomotors. Altres motors com motors pas a pas, motors sense escombretes o motors de corrent alterna, són innecessaris, ja que donen més potència de la que es farà servir, en el cas dels motors de corrent alterna i els motors sense escombretes. Els motors pas a pas són massa precisos en moviment i més cars. Un afegit és que tots els dispositius exceptuant els servomotors, necessiten un circuit adicional normalment anomenat driver per poder ser controlats correctament pel microcontrolador. El cas dels motors de corrent continua és tolerable envers els altres casos.

Imatge 11 – Motor pas a pas genèric.

2.4.2.6. Motors de corrent continua

La forma més factible de controlar un motor de corrent continua és amb un pont en H. S'utilitzen transistors i díodes per protegir el circuït que els controla, per poder controlar el sentit i la potencia dels motors, separant el voltatge de funcionament d'aquests, que depèn dels motors utilitzats, envers el voltatge de la lògica del sistema, que per norma és de 5 volts. Amb l'ajuda d'un pols modulat *PWM* podem controlar la tensió de sortida que s'aplica al motor, per tant, podem controlar-ne la velocitat.

Existeixen circuits integrats preparats específicament per proporcionar aquest mecanisme, com el *L293D*.

El punt a tenir més en compte a l'hora d'utilitzar aquests motors són les baixades o pujades de tensió de les bateries, a causa de la rotació dels motors, que es pot comportar com un generador de tensió, donada la seva naturalesa electromagnètica. Quan es fa girar el motor es generen baixades de tensió, per tant hem de separar la font d'alimentació dels motors de la de la lògica, és a dir, el microcontrolador.

Un altre punt en contra és que el moviment dels motors pot no ser constant si no hi ha una bona alimentació o un bon sistema d'engranatges, cosa que fa que tinguin un moviment progressiu.

En quant a control, necessitem 3 pins de control per cada motor, 2 per controlar el sentit i 1 per la potència o velocitat. Això fa un total de 6 pins pel control dels 2 motors necessaris pel robot, que no és important, ja que disposem de pins de sobra.

En resum, és un bon sistema de locomoció però el fet d'un driver extra per controlar-los, així com el seu possible comportament progressiu, fa que no siguin del tot òptims donades les necessitat de moviment del nostre robot.

Imatge 12 – Driver de motors L293D.

2.4.2.7. Servomotors

Els servomotors són similars a un motor de corrent continua, tret que ja vénen muntats dins una caixa de plàstic amb un sistema d'engranatges reductors i un circuit de control per controlar-los. Generalment s'utilitzen en sistemes de radiocontrol per controlar la direcció d'un cotxe o les ales d'una avioneta, ja que es pot controlar la seva posició amb un angle de rotació de 180º.

En quant a les connexions, es pot controlar la seva velocitat i direcció amb un sol senyal, per tant només necessitarem un pin de control. Les entrades d'un servomotor són un pol positiu, un pol negatiu o terra, i una entrada de control. Per controlar el servomotor, utilitzarem la modulació per ample de polsos *PWM*, de la que disposem des de la nostra placa *IOIO*.

Amb una simple modificació o comprant el model adequat, es pot treure la limitació de rotació de 180º i convertir-lo amb un motor de rotació continua. A més per norma general el voltatge d'alimentació és de 5 volts, que és més fàcil d'aconseguir que la d'un motor de corrent continua, que sol ser més alta. Amb la caixa d'engranatges reductora ens assegurem una velocitat constant i no progressiva, i el seu control és relativament senzill.

2.4.2.8. Motor escollit

Comparant els dos possibles motors a aplicar en el nostre projecte, ens quedem sens dubte amb el servomotor, que ens facilitarà la vida i no necessita de cap component extra per ser controlat, a més de ser més fiable i fàcil de dirigir.

Després de cercar per Internet, hem trobat un servomotor de rotació continua prou potent com per moure el nostre robot i a un preu assequible. El seu cost és de 12€ per unitat i el model és el GWS S35 STD.

Imatge 13 – Servomotor GWS S35 STD.

2.4.2.9. El mòbil Android

No ens podem oblidar de que disposem d'un terminal mòbil amb *Android* a la plataforma robòtica. Els *Smartphones* disposen de diversos sensors que utilitzen per actuar en conjuntament amb les seves aplicacions i oferir diferents opcions.

Els sensors més comuns en tots els terminals són el giroscopi i el sensor d'orientació, per tant els podem fer servir per al nostre projecte sense por de no trobar-los en diferents dispositius.

També hem de tenir en compte que disposem d'un sensor *GPS* per situar la nostra ubicació, que en absència de senyal, utilitza la triangulació de la xarxa mòbil o els serveis de *Google* per suplir la seva funcionalitat.

En última instància, podem utilitzar la càmera del nostre dispositiu per enviar imatges a temps real i formar així un vídeo.

2.4.2.9.1. Sensor giroscopi

Un sensor giroscopi és un dispositiu que mesura la velocitat angular. En els dispositius d'*Android* ens serveix per desenvolupar aplicacions o jocs que requereixin moure l'aparell per interaccionar. Depenent de la velocitat de moviment, ens donarà una mesura o una altra La sensibilitat del dispositiu és bastant bona, a més ens dóna informació dels 3 eixos de moviment.

En el robot utilitzarem aquest sensor per saber la inclinació, però també el podrem fer servir per a futures implementacions, per tant, el tindrem en compte.

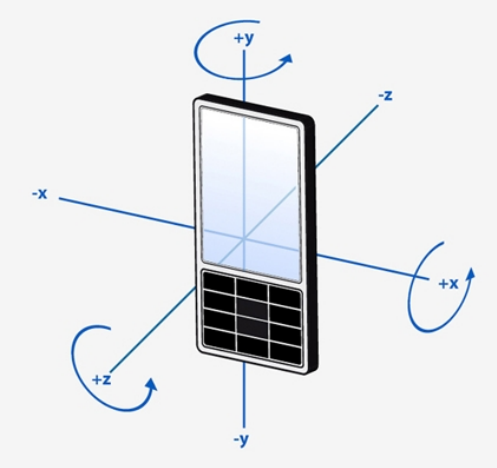


Figura 4 – Eixos de rotació d'un mòbil.

2.4.2.9.2. Sensor d'orientació

El sensor d'orientació ens permet fer servir el camp magnètic per saber la posició respecte el nord. Té una sensibilitat que ens permet orientar-nos en els 3 eixos, però no ens permet detectar la velocitat de desplaçament a diferència del giroscopi. Així doncs amb aquest sensor podrem saber si el robot gira a dreta o esquerra, o si s'inclina cap algun cantó. Una informació molt útil si hem de fer-lo anar en alguna direcció en concret, o per saber inestabilitats en el terreny. A efectes pràctics és una brúixola.

És per tant un sensor molt a tenir en compte i que farem servir en el nostre projecte.

2.4.2.9.3. Sensor GPS

El posicionament global és una pràctica habitual en els nostres temps, tant que molts dels vehicles moderns el porten incorporat, així com la gran majoria dels *Smartphones*. A la plataforma *Android* es disposa d'un sensor *GPS* per adquirir la posició, en la seva absència, es pot fer servir la connexió a Internet per triangular la posició per mitja dels serveis de *Google*. Ens assegurem així, tenir sempre les coordenades de posició a la nostra plataforma.

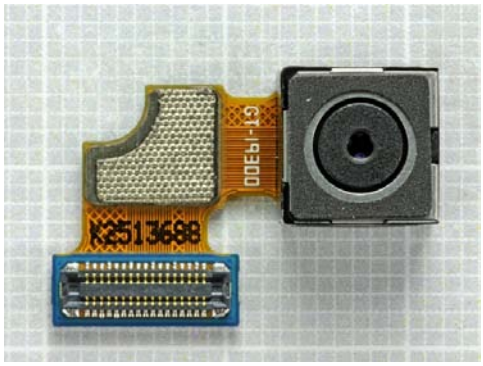
Aquesta informació pot ser útil en alguns projectes per tal d'establir rutes o actuar d'una determinada forma depenent de la situació geogràfica on ens trobem.

Ja que pot ser una informació que ens pot obrir un gran ventall de possibilitats, també l'utilitzarem en el projecte, per obrir la porta a futures millores i usos.

2.4.2.9.4. Càmera

Un dels sensors que ens pot aportar més informació és la càmera. Utilitzant aquest dispositiu serem capaços de detectar objectes, situar-nos, localitzar successos... Per tal d'integrar-lo en el nostre sistema enviarem la informació de les imatges pel *WebSocket* tenint en compte que tindrem un cert decalatge de la senyal.

És sense cap dubte el sensor que aporta més càrrega de comunicació en tot el sistema, però els avantatges d'informació que ens aporta, s'ho valen.



Imatge 14 – Sensor d'una càmera.

2.4.3. El servidor

Anteriorment hem decidit que utilitzaríem la tecnologia dels *WebSockets* per fer la comunicació de tot el sistema. Les llibreries que s'adaptaven més al nostre projecte tenen una versió en el llenguatge *Python*.

Python és un llenguatge lliure i interpretat, és a dir, utilitza un interpret que té diferents versions segons el sistema operatiu per poder fer córrer el nostre programa, exactament igual que el llenguatge *Java*. Va ser creat en els anys 80 com a successor del llenguatge de programació *ABC*. Permet la programació orientada objectes i és fortament tipat, ja que utilitza salts de línia i tabulacions que defineixen seccions de codi. En els últims anys ha esdevingut un llenguatge de gran ús, ja que és molt fàcil de llegir i entendre, a més permet desenvolupar programes per quasi la totalitat dels àmbits que es poden donar. Com a mostra de la seva popularitat i de totes les propietats, és un dels 3 grans llenguatges que utilitza *Google* per desenvolupar les seves aplicacions.

Per tant, no tindrem cap problema a l'hora de desenvolupar la part del nostre servidor fent servir la tecnologia de *Python*, que juntament amb les llibreries d'*Autobahn WebSocket*, compliran perfectament els nostres objectius.

Imatge 15 – Logo de Python.

2.4.4. El programa client

Utilitzant el navegador web com a plataforma principal del programa client i *Javascript* per comunicar-nos per *WebSockets*, tenim les eines necessàries per desenvolupar una interfície atractiva i dinàmica. Utilitzar *HTML* i *CSS* ens permetrà crear una estructura gràfica interactiva per poder operar amb el robot. També podem fer ús del gràfics vectorials per desenvolupar per exemple una brúixola interactiva, o una barra de nivell.

En quant al possible objectiu de carregar codi per ser executat com a programa de la plataforma robòtica, utilitzarem les funcions de la consola de desenvolupador que incorporen tant el navegador *Firefox* com *Chrome*, que ens permeten modificar el *DOM* (*Document Object Model*) de la pàgina web i afegir-hi codi que pot ser executat.

2.5. Descripció del sistema

Ja tenim doncs totes les parts del nostre sistema definides. Utilitzarem les eines descrites anteriorment per desenvolupar-ne cada una de les seves parts i arribar als objectius del nostre projecte.

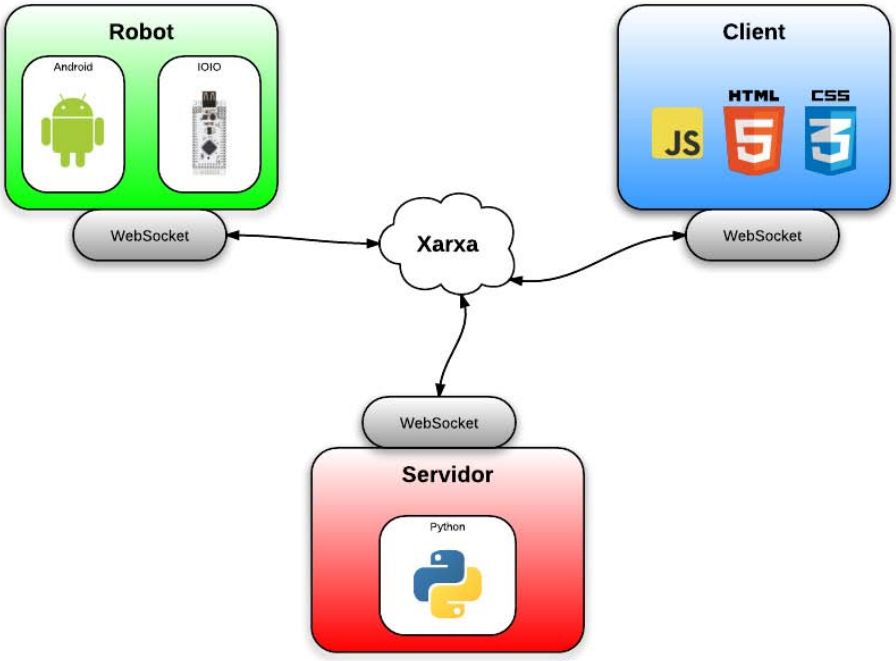


Figura 5 – Parts del sistema i tecnologies utilitzades.

Resumint, implementarem tot el sistema utilitzant una comunicació amb *WebSockets*. La plataforma robòtica utilitzara la placa *IOIO* per controlar el hardware i connectar-se amb el mòbil *Android*. El client utilitzarà *Javascript* per poder utilitzar *WebSockets* i *HTML* i *CSS* per dissenyar la interfície gràfica. El servidor es desenvoluparà amb *Python* juntament amb la llibreria de *WebSockets Autobahn*.

2.6. Recursos

En tot projecte s'utilitzen una sèrie de recursos de diferents tipus, com poden ser els programes utilitzats, el material, el personal que hi participa... Anem a veure els diferents recursos utilitzats en aquest projecte.

2.6.1. Recursos humans

Al ser un projecte de final de carrera, l'únic recurs humà que ens serà necessari serà el propi alumne, que realitzarà totes les fases. Per tant no implicarà un cost monetari de desenvolupament.

2.6.2. Recursos hardware

Com hem especificat anteriorment quins components utilitzarem, el recursos hardware a utilitzar són els següents.

Plataforma robòtica:

- 1 x Placa *IOIO*
- 1 x Mòbil *Android* (Ens serveix un *Smartphone* de gama baixa)
- 2 x Servomotors *GWS S35 STD*.
- 2 x Rodes de goma
- 1 x Mini protoboard
- 4 x Piles AA
- 1 x Pila 9 volts
- 1 x Porta-piles de 4 unitats
- 4 x Resistències 4.7 KΩ
- Cablejat
- Regulador de voltatge *L7805CV*
- Connectors mascle
- Connectors femella
- Cable *USB* compatible amb el terminal *Android*

Material general :

- 1 x Multímetre
- 1 x Estany de soldar
- 1 x Soldador
- 1 x Tub de pega
- 1 x Tisores

A més per desenvolupar i fer funcionar el sistema necessitarem :

- 1 x PC amb sistema operatiu *Windows*, *Linux* o *Mac*
- 1 x Navegador compatible amb *WebSockets*
- Connexió a una xarxa on es pugui implementar el protocol *TCP-IP*

2.6.3. Recursos software

Per tal de desenvolupar el codi, fer dissenys i esquemes, i executar el sistema, necessitarem els següents programes informàtics:

- Navegador compatible amb *WebSockets*
- Intèrpret de *Python*
- Llibreries de *WebSockets Autobahn*
- Llibreries de la placa *IOIO*
- Llibreria *Jquery*
- Llibreria *Gmap*
- *API* de *Google Maps*
- *Eclipse* 3.6.2
- Servei online *Lucidchart*
- *Fritzing*
- *LibreOffice*

2.6.4. Pressupost

Un dels objectius del projecte és que sigui un projecte econòmic, per tal de que pugui arribar al màxim de clients possibles. Tot seguit contabilitzarem el cost de tots els recursos, per poder determinar-ne la seva viabilitat posteriorment.

El cost el dividirem en tres apartats : recursos hardware, recursos software i recursos humans.

Recursos hardware	
1 x Placa *IOIO*	39,8€
1 x Mòbil *Android*	50€
2 x Servomotors *GWS S35 STD*	20€
2 x Rodes de goma	5€
1 x Mini protoboard	3,25€
4 x Piles AA	2,75€
1 x Porta-piles de 4 unitats	0,5€
2 x Resistències 4.7KΩ	0,10€
Cablejat	1,5€
Regulador de voltatge *L7805CV*	1,6€
Connectors mascle	1,2€
Connectors femella	1,2€
Cable *USB* compatible amb terminal el *Android*	8€
1 x Multímetre	25€
1 x Estany de soldar	3€
1 x Soldador	30€
1 x Tub de pega	3€
1 x Tisores	1,5€
1 x PC amb *Windows*/*Linux*/*Mac*	500€
1 x Pila de 9 volts	2€
Recursos Software	
Navegador compatible amb *WebSockets*	Gratuït
Intèrpret Python	Gratuït
Llibreries de *WebSockets Autobahn*	Gratuït
Llibreries de la placa *IOIO*	Gratuït
Llibreria *Jquery*	Gratuït
Llibreria *Gmap*	Gratuït
API de *Google Maps*	Gratuït
Eclipse 3.6.2	Gratuït
Servei online *Lucidchart*	Gratuït
Fritzing	Gratuït
LibreOffice	Gratuït
Recursos humans	
395 hores de disseny i desenvolupament (15€ / hora)	5925€
	TOTAL 6624,4

Taula 1 – Cost del projecte en un entorn professional.

Aquest seria el cost del projecte si el portéssim a l'àmbit professional. Com que el projecte ha estat desenvolupat per l'alumne, que ja posseeix un mòbil *Android*, a més del material bàsic, i ja que s'ha intentat utilitzar el màxim material reciclat possible, el cost del projecte ha estat el següent :

Recursos hardware	
1 x Placa *IOIO*	39,8€
2 x Servomotors *GWS S35 STD*	20€
1 x Mini protoboard	3,25€
4 x Piles AA	2,75€
1 x Porta-piles de 4 unitats	0,5€
2 x Resistències 4.7KΩ	0,10€
Regulador de voltatge *L7805CV*	1,6€
1 x Tub de pega	3€
1 x Tisores	1,5€
1 x Pila de 9 volts	2€
TOTAL 74,5€	

Taula 2 – Cost real del projecte a càrrec de l'alumne.

2.7. Avaluació de riscos

En tot projecte hi ha circumstàncies que ens podem modificar el progrés o la consecució del mateix, és per això que cal detectar-les i fer-ne un pla de contingència.

El possibles riscos i les seves mesures preventives que ens poden sorgir en aquest projecte s'especifiquen a continuació.

2.7.1. Llista de riscos

Els diferents riscos aplicables en el projecte són:

- R1. Planificació temporal optimista : No s'acaba en la data prevista, augmenten els recursos.
- R2. Manca alguna tasca necessària : No es compleixen els objectius del projecte.
- R3. Pressupost poc ajustat : Pla de projecte. Pèrdues econòmiques.
- R4. No es fa correctament la fase de test : Desenvolupament, implantació. El sistema no funciona bé pels casos desitjats.
- R5. Incompliment d'alguna norma, reglament o legislació : Repercussions legals.
- R6. Mal-funcionament del material electrònic : En qualsevol fase. Pèrdues econòmiques, endarreriment del projecte.
- R7. Abandonament del projecte: En qualsevol fase. Pèrdues econòmiques.

2.7.2. Catalogació de riscos

Veiem la possibilitat de que es produeixi un risc, així com el seu impacte.

	Probabilitat	Impacte
R1	Alta	Crític
R2	Alta	Crític
R3	Alta	Crític
R4	Alta	Marginal
R5	Alta	Crític
R6	Baixa	Crític
R7	Baixa	Catastròfic

Taula 3 – Probabilitat dels riscos i el seu impacte.

2.7.3. Pla de contingència

Possibles mesures per solucionar els problemes.

	Solució
R1	Ajornar alguna funcionalitat, afrontar possibles pèrdues, fer una assegurança.
R2	Revisar el Pla de Projecte, modificar la planificació.
R3	Renegociar amb el client, afrontar possibles pèrdues, fer una assegurança.
R4	Renegociar amb el client, ajornar funcionalitat, modificar planificació i pressupost.
R5	Demanar un ajornament, negociar amb el client, afrontar pèrdues.
R6	Reemplaçar les peces defectuoses, minimitzar el retràs en el projecte.
R7	No té solució.

Taula 4 – Solucions proposades a possibles problemes.

2.8. Planificació del projecte

Una bona planificació pot facilitar el desenvolupament del projecte i la seva consecució. En el nostre cas, el projecte segueix un caire evolutiu i seqüencial, de manera que no podem seguir una fase sense haver finalitzat la anterior.

Les fases de desenvolupament del projecte són les següents :

nº	Descripció de l'activitat	Durada	Recursos	Pred.
1	Inici del projecte : assignació i matriculació	2h	Alumne	
2	Planificació	30h	Alumne	
3	Estudi de viabilitat	20h	Alumne	1
4	Aprovació Estudi Viabilitat (Punt de control)	1h	Alumne	3
5	Pla de projecte	8h	Alumne	4
6	Aprovació Pla del Projecte (Punt de control)	1h	Alumne	5
7	Anàlisi de l'aplicació	26h	Alumne	
8	Anàlisi de requisits (casos d'ús)	6h	Alumne	6
9	Anàlisi de hardware	6h	Alumne	8
10	Anàlisi de la legalitat	3h	Alumne	9
11	Documentació de l'anàlisi (Punt de control)	10h	Alumne	10
12	Aprovació de l'anàlisi (Punt de control)	1h	Alumne	11
13	Disseny de l'aplicació	19h	Alumne	
14	Disseny del hardware	10h	Alumne	12
15	Disseny modular de l'aplicació	2h	Alumne	14
16	Disseny de d'interfície	2h	Alumne	15
17	Disseny de les proves (test)	2h	Alumne	16
18	Documentació del disseny	2h	Alumne	17
19	Aprovació del disseny (Punt de control)	1h	Alumne	18
20	Desenvolupament de l'aplicació	218h	Alumne	
21	Preparació entorn de desenvolupament	10h	Alumne	19
22	Plataforma mecànica del robot	25h	Alumne	21
23	Ensamblatge servomotors	3h	Alumne	22
24	Electrònica de la plataforma	25h	Alumne	23
25	Programació *Android* per comunicar-se amb *IOIO*	25h	Alumne	24
26	Programació *Android* per comunicar-se amb el servidor	50h	Alumne	25
27	Desenvolupament del programa client	80h	Alumne	26
28	Test i proves	63h	Alumne	
29	Proves mecàniques robot	10h	Alumne	27
30	Proves unitàries *IOIO*	10h	Alumne	29
31	Proves unitàries *Android*	10h	Alumne	30
32	Proves unitàries servidor	10h	Alumne	31

33	Proves d'estres servidor	2h	Alumne	32
34	Documentació del desenvolupament i test	20h	Alumne	33
35	Aprovació del desenvolupament i test (Punt de control)	1h	Alumne	34
36	Generació de documents (memòria del projecte)	30h	Alumne	35
37	Tancament del projecte	1h	Alumne	36
38	Defensa del projecte	6h	Alumne	37

Taula 5 – Planificació del projecte en el temps.

2.9. Conclusions

Després de buscar la manera més viable d'arribar a l'objectiu final de projecte i de calcular-ne el seu cost i la seva planificació, he arribat a la conclusió que el projecte és realitzable. Hem buscat les opcions més econòmiques i que ens poden aportar més versatilitat al projecte, així com software lliure i en desenvolupament continu, que ens garanteixen durabilitat i fiabilitat pel futur.

3. La plataforma robòtica

El robot es pot separar en dues parts principals, la part hardware o mecànica i electrònica, i la part lògica de programació. Seguidament detallarem la construcció de la plataforma robòtica utilitzant els servomotors i les piles, així com el circuït necessari per connectar-lo amb la placa *IOIO*. Després explicarem l'esquema de la lògica del programa utilitzat en el sistema *Android* i les seves funcions.

3.1. Part física

Dins de la part física ens trobem amb dos tipus de hardware, la plataforma en sí, que servirà per muntar-hi els servomotors, la electrònica i la placa *IOIO*, i el circuit electrònic per controlar el voltatge de la font d'alimentació (piles) i la senyal dirigida als motors.

Abans de dissenyar la plataforma de suport, crearem el circuit de control de les bateries i els servomotors, per després buscar la millor manera de posicionar-los.

3.1.1. L'electrònica

El circuit que necessitem ha de controlar en primera instància el voltatge a suplir als servomotors des de les 4 piles AA de 1.5 volts. Per aconseguir-ho utilitzarem el regulador de voltatge *L7805CV*, que ens convertira els 6 volts a 5 volts.

La alimentació de la placa *IOIO* la farem amb una pila de 9 volts, ja que el circuit integrat ja porta un regulador de voltatge propi. El fet de tenir dues fonts d'alimentació diferents, és per evitar variacions de la diferència de potencial quan es fan servir els motors. Al ser components electromecànics, quan es posen en funcionament poden produir pics o baixades de voltatge deguts a la inducció magnètica, això produeix que el voltatge baixi de 5 volts, el que ens porta a un reset de la placa *IOIO* o el seu mal-funcionament.

Com em comentat a l'estudi de viabilitat, la placa *IOIO* ens aporta diferents pins per controlar una gran varietat de dispositius. Per controlar els servomotors necessitem un pols *PWM* i un voltatge de 5 volts. Malauradament per defecte els pins de sortida només poden proporcionar 3.3 volts. Per solucionar aquest problema, utilitzarem els pins anomenats open-drain o de col·lector obert, que toleren una entrada de voltatge de 5 volts. Així, si proporcionem un voltatge de cinc volts, juntament amb una resistència de 4.7KΩ per limitar el corrent, cada cop que el pin proporcioni una sortida *PWM* ho farà amb una sortida de 5 volts, la que necessitem per posar en marxa els nostres servomotors.

El circuit que muntarem sobre la mini protoboard serà el següent:

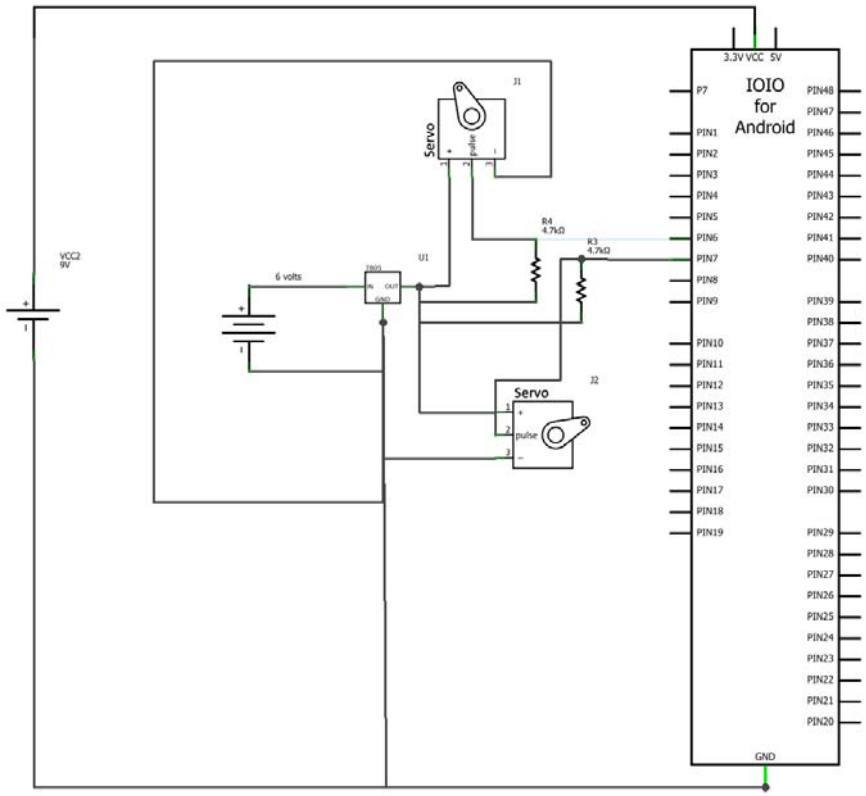


Figura 6 – Esquema electrònic de la plataforma robòtica.

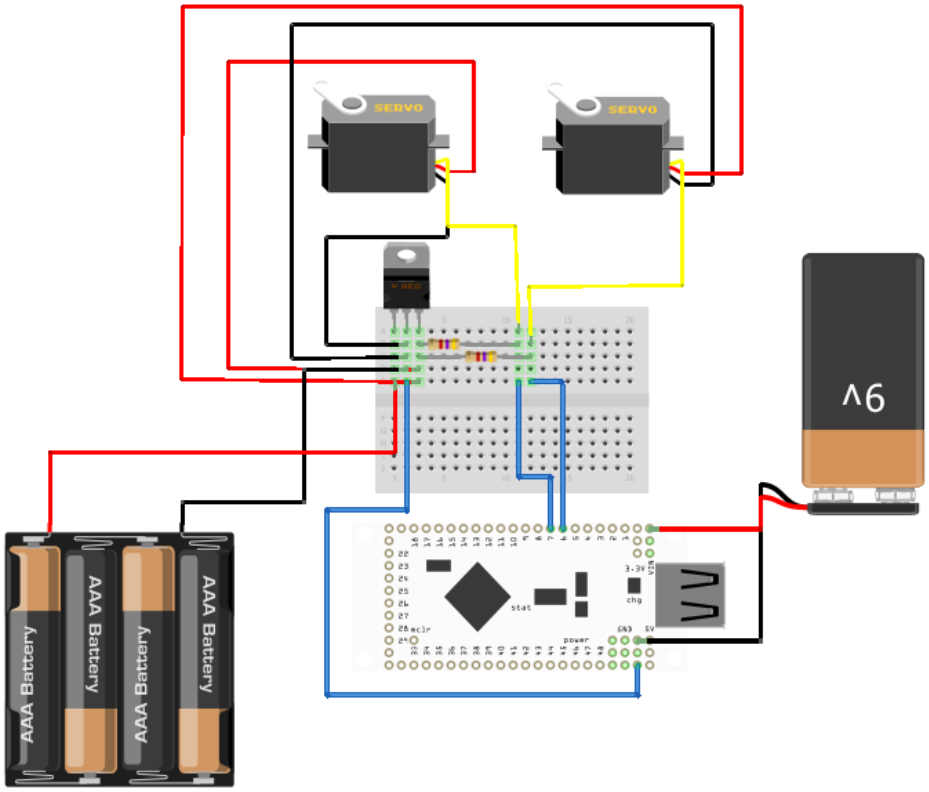


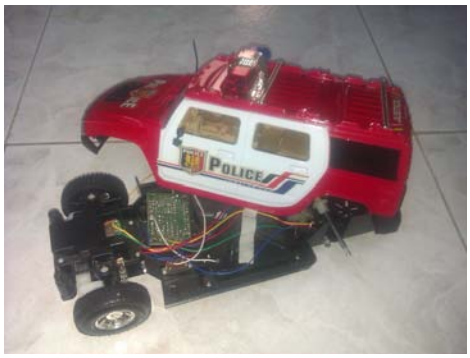
Figura 7 – Muntatge dels components electrònics del robot.

Bàsicament en el circuit regulem el voltatge de les bateries i proporcionem una diferència de potencial suficient per a moure els servomotors. Un punt a tenir en compte és que encara que s'utilitzin diferents fonts d'alimentació en el que podríem separar com a dos circuits electrònics diferents, hem d'unir els esquemes compartint la mateixa toma de terra, si no, el servomotor no rebria el corrent controlat amb la senyal de la placa *IOIO*.

3.1.2. La mecànica

En aquest ens centrarem a construir la plataforma física que contindrà tots els components. Sent fidel a l'objectiu d'utilitzar el mínim pressupost possible, ens topem amb una paraula clau, el reciclatge.

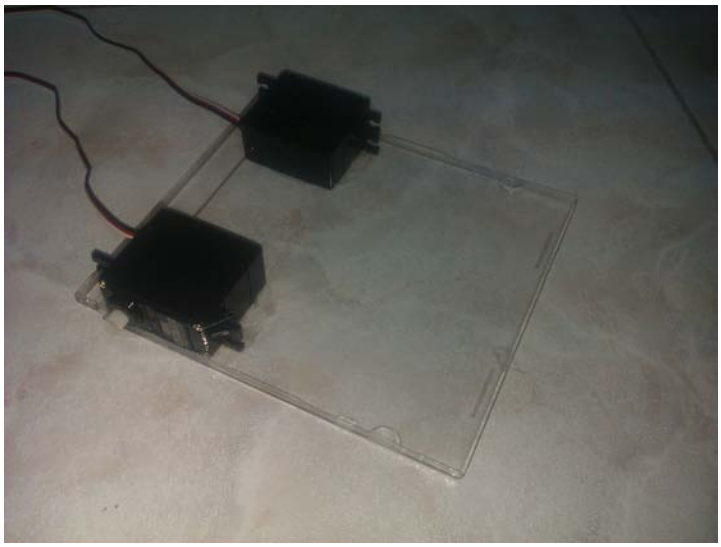
Per tal de construir el suport, utilitzarem una simple caràtula de CD o DVD, ja que el plàstic es pot treballar fàcilment i ens dóna la forma perfecta. Per les rodes, farem servir les rodes d'una joguina vella adaptan-les a les rodes dentades que porten per defecte els servomotors quan els comprem.



Imatge 16 – Joguina per reciclar.

Imatge 17 – Rodes adaptades als servomotors.

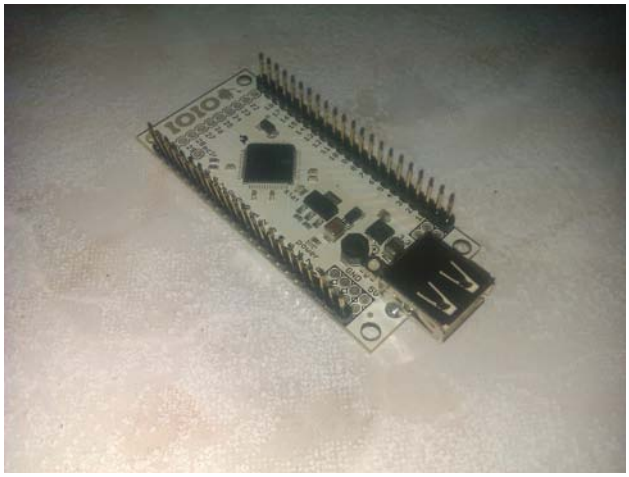
Un cop tenim les rodes adaptades, enganxarem els servomotors a la caràtula del CD procurant que estiguin ben alineats, per no tenir problemes de direcció durant el funcionament del robot.



Imatge 18 – Servomotors enganxats a la caràtula de CD.

Com que la caràtula de CD és massa flexible, utilitzarem un llistó de fusta per reforçar-la, després amb ajuda de pega enganxarem el suport de les piles per la part inferior, la mini protoboard amb la cinta adhesiva que ja porta, i farem ús d'un ferro roent per fer els forats necessaris per collar la placa *IOIO*.

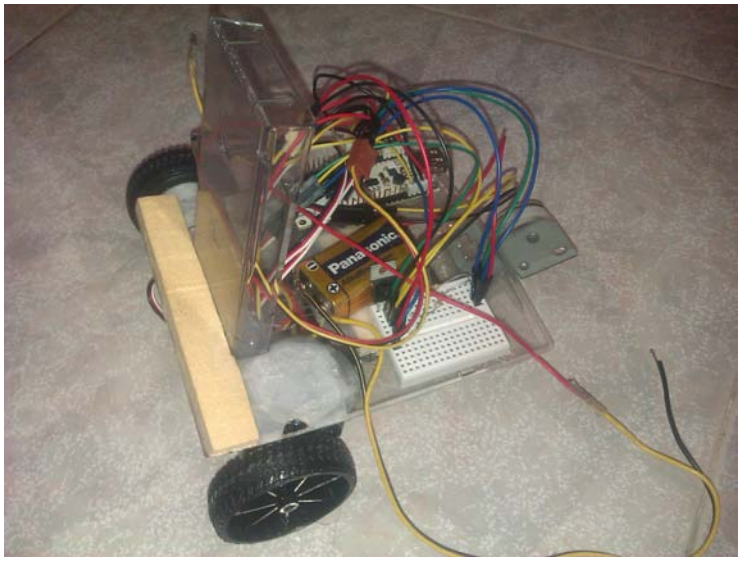
Com que utilitzem una protoboard, no ens farà falta soldar els components del circuit que hem dissenyat anteriorment. Per tal de fer les connexions amb la placa i les piles, utilitzarem uns connectors mascle per als pins de la placa IOIO.



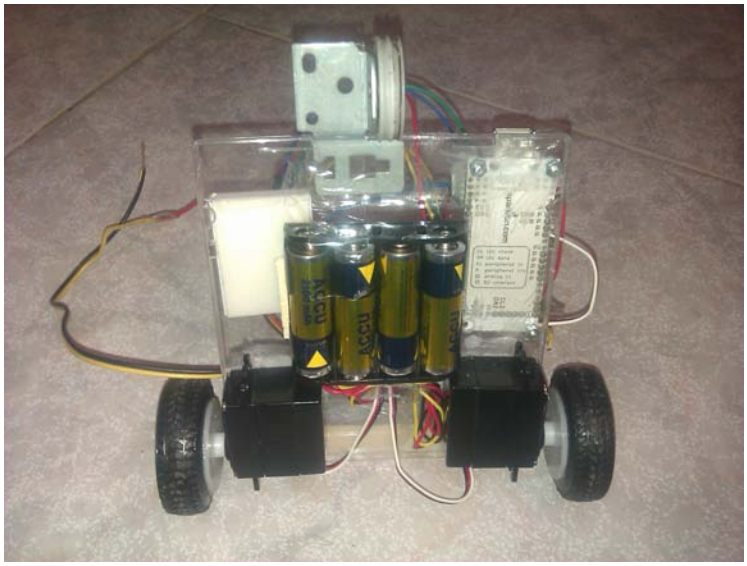
Imatge 19 – Conectors soldats a la placa IOIO.

Ara utilitzarem els connectors femella d'una font d'alimentació d'ordinador vella per col·locar-los en els pins que necessitem, i muntarem el circuit completament. Finalment utilitzarem una peça d'una fotocopiadora vella per fer la roda del darrera del robot, per tal de que no s'arrossegui pel terra. Respecte a la roda posterior, seria convenient fer servir una roda boja, amb mobilitat de 360º, però com utilitzarem el robot en espais d'interior i la roda no ofereix quasi fricció, ens servirà perfectament.

Per últim i per poder col·locar el mòbil en una posició favorable per utilitzar la càmera i fer un bon ús del sensors, utilitzarem la carcassa de plàstic d'un casset per fer-li de suport. El resultat de tota la operació queda de la següent manera.



Imatge 20 – Plataforma robòtica muntada.



Imatge 21 – Part inferior de la plataforma robòtica.

3.2. Programació

La lògica del robot la portarà el terminal *Android* connectat a la placa *IOIO*, és per això que el programa de control serà en *Java* i utilitzant la *API* 2.2 d'*Android*, que és la més utilitzada a dia actual, i que ens permet tenir compatibilitat en les versions posteriors. A més és un requisit mínim de la llibreria de *WebSockets Autobahn*. Per tal de fer la programació utilitzarem el programa *Eclipse* 3.6.2 i la *API* de *Java* 1.6, ja que és el recomanat per utilitzar les llibreries de *IOIO*, perquè donen menys problemes i està més testejada.

Abans de començar a programar hem de conèixer com funciona el sistema operatiu *Android* i la seva *API* 2.2.

3.2.1. Android API 2.2

Les llibreries d'*Android* són un conjunt de classes que utilitzarem com a base per extendre-les i fer les nostres aplicacions. La classe base s'anomena `Activity` i ens permet crear una activitat pel sistema, on li podem donar una interfície gràfica capaç d'interaccionar amb l'usuari. Per tant crearem una classe que extengui la classe `Activity` i utilitzarem codi XML inclòs dins la API per desenvolupar la interfície gràfica. També utilitza un arxiu manifest en XML per declarar informació de l'aplicació, com la versió *API* utilitzada, els permisos necessaris que necessita i diverses opcions extres.

Al ser un sistema operatiu que s'aplica a diferents dispositius amb recursos limitats, com són la memòria *RAM* i d'emmagatzematge, així com la potència del processador, el mateix sistema s'encarrega de finalitzar i tancar el processos inactius o que consumeixen massa recursos pel seu nivell de prioritat. Ho hem de tenir en compte, ja que si l'usuari no és conscient de que l'activitat està en funcionament, la nostra prioritat baixarà i podrà ser tancada, a més en un dispositiu poc potent o amb masses programes oberts, el mateix sistema també ens pot tancar els processos.

El nostre programa actuarà com un servidor, que rebrà i enviarà les dades, per tant ha d'estar sempre en funcionament i ha de ser capaç d'executar altres processos autònoms que s'encarreguin per exemple d'enviar els valors dels sensors. La classe `Activity` és limitada en aquest sentit, pel simple motiu que si estem constantment escoltant un event, bloquejarem la interfície gràfica, i això no ens interessa. Les alternatives que ens ofereix la *API* d'*Android* són les classes `Service` i `Thread`.

Un servei d'*Android* està pensat com un complement a una activitat, per fer alguna operació en segon pla durant un temps limitat. A efectes pràctics es tracta d'un *thread* adaptat a l'entorn de l'aplicació *Android* però hi ha una limitació que hem de tenir en compte. El servei s'executa en el mateix procés principal que l'aplicació i si l'utilitzem per estar escoltant constantment com si fos un servidor utilitzant les llibreries *d'Autobahn*, *IOIO* i les

classes per atendre els sensors, el *GPS* i la càmera, a la llarga ens donarà problemes i errors en el sistema. A més no està pensat per tenir múltiples serveis dins d'una activitat, cosa que no vol dir que no es pugui fer, però no ens donarà un bon rendiment.

Un *thread* ens ofereix totes les opcions de la classe `Service` i ens treu la limitació d'executar-se en el mateix process principal i ens permet crear-ne de diferents segons els necessitem, per tant la classe `Thread` serà la nostra millor opció per utilitzar conjuntament amb l'activitat d'*Android*.

3.2.2. Estructura de l'aplicació

Donades les funcionalitats de la *API* 2.2 d'*Andoid* explicades anteriorment utilitzarem una classe activitat per presentar a l'usuari una interfície per iniciar el servidor. El servidor de l'aplicació correrà sobre un *thread* i iniciarà una connexió utilitzant *WebSockets* per parlar amb el servidor remot, que ens pemetrà enviar les dades al robot.

Les funcionalitats que inclourem en el nostre programa són:

- Ús del sensor d'orientació.
- Ús del sensor giroscopi.
- Ús del sensor *GPS*.
- Ús de la càmera.
- Connexió amb la placa *IOIO*

Totes aquestes funcionalitats requereixen estar permanentment escoltant les dades que els hi transmet el sistema operatiu *Android*.

Per la recepció del sensors d'orientació i giroscopi podem fer servir la mateixa classe per atendre els enviaments de les dades, és una classe genèrica del sistema anomenada `SensorEventListener`.

Per el sensor *GPS* i la càmera, el sistema operatiu *Android* també ens proveeix de dues classes diferents per atendre les recepcions de la informació, són la classe `LocationListener` i `PreviewCallback`.

Per fer la comunicació amb la placa *IOIO* i poder parlar amb el robot, utilitzarem les llibreries que ens ofereix la *API IOIO* i concretament utilitzarem la classe `IOIOFactory`, que ens permet manipular la connexió sense haver de dependre d'una classe del sistema *Android*.

Per tant tenint tots aquests recursos i les classes utilitzades, l'estructura de l'aplicació es basarà en una activitat que permetrà iniciar el servidor a l'usuari. Seguidament el servidor iniciat (creat amb un *thread*) iniciarà a la vegada diferents *threads* per manipular i atendre tota la informació dels diferents sensors i establir la comunicació amb la placa del robot. Tot aquest sistema ens permetrà utilitzar l'activitat i la interfície gràfica de l'usuari sense bloquejar-la de manera que l'usuari podrà parar el servidor i iniciar-lo quan vulgui.

La representació de l'estructura gràfica del programa queda de la següent manera:

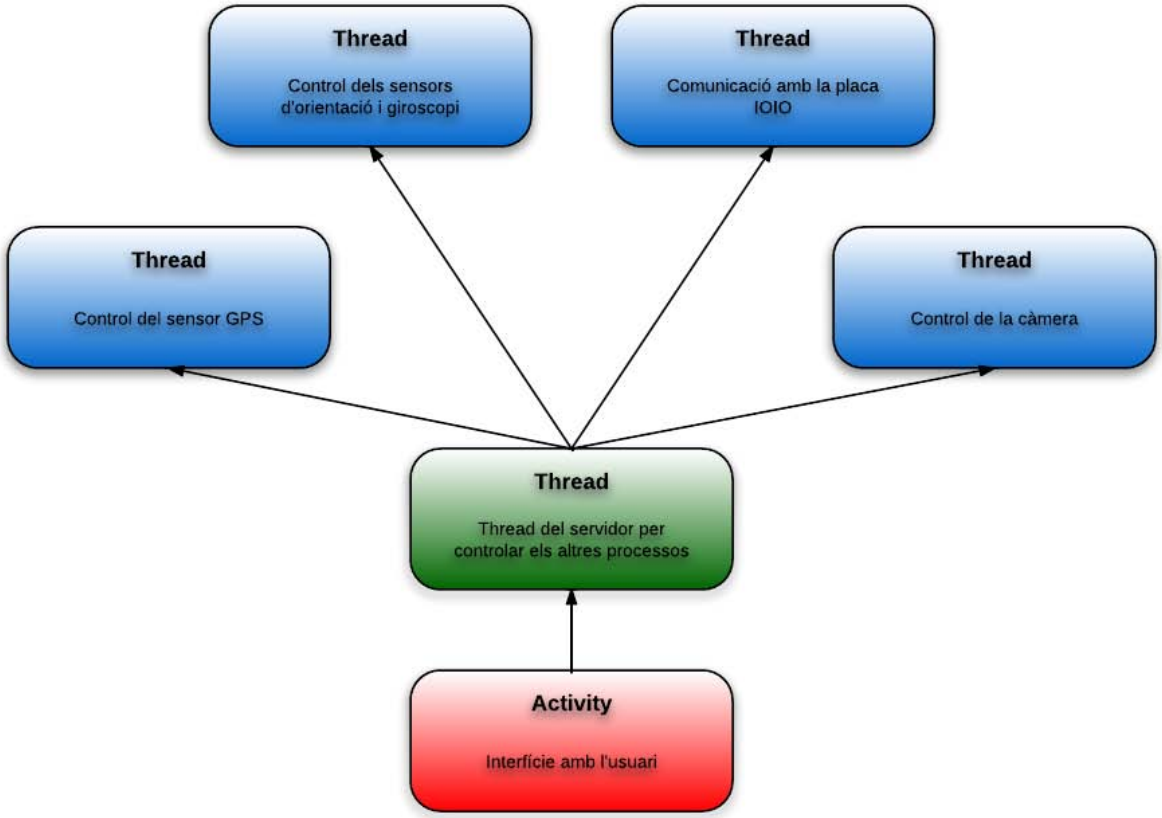


Figura 8 – Estructura de processos del programa robot.

3.2.3. L'activitat de l'aplicació

La funció bàsica de l'extensió de la classe `Activity` és com hem comentat anteriorment la de proporcionar una interfície gràfica amb la qual l'usuari podrà iniciar el servidor amb els paràmetres que ens facin falta, també utilitzarà les característiques derivades de la classe per poder fer ús de les diferents funcionalitats del sistema. Per tal de fer-ho crearem la classe `RobotActivity`, a partir d'aquesta programarem una serie de camps de text on introduirem la direcció *IP* o la ruta del servidor, el port que utilitzarem, i el nom i contrasenya del robot. El codi amb els mètodes principals de la classe extesa queden de la següent manera:

```
public class RobotActivity extends Activity {

    //Variables utilitzades
    ...
    ThreadServidor servidor;
    SensorManager managerSensors = null;
    LocationManager mlocManager = null;
    SurfaceView vista;
    ThreadSensors sensors=null;
    ThreadGPS gps=null;
    RobotActivity activitat;
    ...

    //Widgets visuals per la UI
    EditText botodireccioip;
    EditText botoportip;
    EditText botorobotid;
    EditText botorobotpass;
    Button botostart;
    Button botostop;
    ....

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        //Inici dels managers dels sensors
        ...
        //Assignem el widgets a les variables globals
        ...
        //Configurem el boto per iniciar i parar el servidor
        ...
     }
}
```

Com podem veure a la classe, definim les variables globals que ens faran falta per mantenir en contacte els *threads* i l'activitat, així com els managers per tractar la informació dels sensors. També definim les variables dels *widgets* que utilitzarem per crear els camps de text on introduirem la informació necessària per al servidor, i els botons d'iniciar i parar el servidor, i les accions a emprendre quan són premuts.

L'únic mètode que sobreescrivim derivat de la classe `Activity` és el mètode `onCreate`, que s'executa quan l'aplicació s'inicia per primera vegada, on donarem valor a les variables i configurarem la interfície gràfica. La resta de mètodes de la classe seran les accions per defecte que el sistema utilitza per aquestes i no ens fa falta tractar-los, ja que no hem de fer-ne una modificació específica.

Respecte a la configuració del projecte creat amb l'*Eclipse*, ens fa falta modificar l'arxiu `main.xml` per definir l'aspecte gràfic i posicionar els diferents *widgets*, així com definir els textos per defecte dels camps. També modificarem l'arxiu `AndroidManifest.xml` per

afegir els permisos que utilitzarà l'aplicació, si no, no podria funcionar, al estar l'accés restringit. El permisos són els següents:

- `permission.ACCESS_FINE_LOCATION`: Per adquirir la posició del *GPS*.
- `permission.ACCESS_COARSE_LOCATION`: Per adquirir la posició del *GPS* utilitzant internet.
- `permission.ACCESS_NETWORK_STATE`: Per poder utilitzar funcionalitats específiques amb els *WebSockets*.
- `permission.INTERNET`: Per poder fer servir la connexió a *Internet*.
- `permission.ACCESS_WIFI_STATE`: Per saber l'estat de la connexió *WIFI*.
- `permission.CHANGE_WIFI_STATE`: Per canviar l'estat de la connexió *WIFI*.
- `permission.CAMERA`: Per poder fer servir el sensor de la càmera.
- `permission.UPDATE_DEVICE_STATS`: Per poder utilitzar els sensors d'orientació i el giroscopi.
- `hardware.camera`: Per poder fer servir el sensor de la càmera.
- `hardware.camera.autofocus`: Per poder fer servir la funcionalitat d'enfocar automàticament de la càmera.

L'aspecte final de l'aplicació serà aquest:

Imatge 22 – Aspecte gràfic de l'aplicació.

3.2.4. El thread servidor

El *thread* servidor correrà en segon pla un cop iniciat, i serà el responsable de posar en marxa tots els altres *threads* de control de sensors i també de parar-los quan faci falta, és el *thread* més important. Per construir-lo utilitzarem la classe `Thread` com a base per adquirir les seves funcionalitats. La classe l'anomenarem `ThreadServidor`.

```
public class ThreadServidor extends Thread{
      //Variables utilitzades
      ...

      public ThreadServidor(RobotActivity app, SurfaceView v,
 SensorManager m, LocationManager g, String ip, String port, String
 robot, String pass)
      {
            ...
      }

     public void run()
     {
       ...
     }

     public void parar()
     {
       ...
     }
 }
```

El funcionament del *thread* és senzill, quan el creem des de l'activitat, es llança el mètode `ThreadServidor` que és el constructor de la classe, on utilitzem punters cap a les variables enviades de la classe `RobotActivity` amb la informació que necessitem per iniciar el servidor. Seguidament creem els diferents *threads* inicialitzant les classes creades, que descriurem en els apartats corresponents, per el control dels sensors i la placa, i ens posem en espera.

Els mètodes `run i parar`, són els que utilitzarà l'activitat per iniciar i parar l'execució del servidor. Com hem descrit anteriorment, aquest és un *thread* per separar l'activitat del procés principal de l'activitat i poder garantir l'estabilitat del sistema.

3.2.5. El thread del sensors

La classe per controlar els sensors d'orientació i el giroscopi s'anomena `ThreadSensors` i no és una extensió d'un *thread*, si no que implementa l'objecte `SensorEventListener` integrat a la *API* 2.2 d'*Android* i que ens permet crear un *handler* per escoltar la informació d'aquests.

Aquesta classe necessita connectar-se al servidor remot utilitzant la funcionalitat dels *WebSockets* per poder enviar la informació que adquireix, per tant, farem la connexió utilitzant els paràmetres que ha passat la classe `ThreadServidor` quan ha creat la dels sensors.

```
public class ThreadSensors implements SensorEventListener{
     //Variables de la classe
     ...

     public ThreadSensors(SensorManager sm, String ip, String port,
String robot, String pass)
     {
           ...
           conectarWebSocket();
     }

    public void parar()
    {
      ...
    }

     @Override
     public void onSensorChanged(SensorEvent event) {
           // TODO Auto-generated method stub

     }
     private void conectarWebSocket() {
           ...
           @Override
           public void onOpen() {
                ...
           }
           @Override
           public void onTextMessage(String missatge) {
                ...
           }

           @Override
           public void onClose(int code, String reason) {
                ...
           }
    }
}
```

Quan es crea la classe es llança el constructor, on fem servir apuntadors per agafar la informació que ens ha passat la classe `ThreadServidor`. Concretament necessitem la variable del tipus `SensorManager` per integrar-la en el mètode `SensorEventListener` i que funcioni correctament. També agafem la direcció del servidor i la informació de l'usuari, és a dir, la identificació d'usuari i la possible contrasenya. Un cop iniciades les variables es llança el mètode `conectarWebSocket`.

Quan és llança el mètode anterior, creem una connexió de *WebSockets* amb la informació que ha introduït l'usuari utilitzant la interfície gràfica. Dins d'aquest mètode utilitzem la llibreria *Autobahn WebSockets*, d'on farem servir tres mètodes en concret per atendre les peticions, aquests són:

- `onOpen`: Quan aconseguim establir la connexió persistent amb el servidor remot, s'activa aquesta funció, que farem servir per iniciar enviar un missatge amb la informació de l'usuari per poder ser autentificats correctament. També activem una variable booleana indicant que la connexió està establerta.
- `OnTextMessage`: Quan rebem un missatge del servidor el tractem aquí, en principi no n'hem de rebre cap, però ho implementem per a futures millores.
- `OnClose`: Quan perdem la connexió per motiu que sigui s'executa. Canviam la variable de connexió establerta a fals.

Un cop ens hem connectat al servidor remot amb el *WebSocket* i hem activat la variable booleana de connexió, el mètode `onSensorChanged` que hem sobreescrit de la classe `SensorEventListener` adquirirà la informació dels sensors d'orientació i el giroscopi, i enviarà una cadena al servidor amb les dades. Si la connexió no ha estat establerta, no enviarem res, ja que no arribarà mai i a més provocaria un error. La cadena enviada serà de la forma `@Robot:sensors:usuari:tipus:dades`, on tipus pot ser *`bruixola`* o *`acceleracio`* depenent del sensor que ha enviat les dades. El servidor remot rebrà el missatge i en farà el seu tractament.

El mètode `parar` es crida des de la classe `ThreadServidor` per aturar la transferència de dades quan s'atura el servidor.

3.2.6. El thread del gps

La classe que utilitzarem per la funcionalitat del *GPS*, és pràcticament idèntica a la classe sensors, amb la diferència que implementa la classe `LocationListener` de la *API* 2.2 d'*Android*, que ens permetrà agafar geoposicionament a partir de la informació del *GPS* o bé de la triangulació de la senyal d'*Internet*. Anomenarem a la classe `ThreadGPS`.

```
public class ThreadGPS implements LocationListener{
     //Variables de la classe
     ...
    public ThreadGPS(LocationManager sm, String ip, String port,
String robot, String pass)
     {
           ...
           conectarWebSocket();
     }

    public void parar()
    {
     ...
    }

     @Override
     public void onLocationChanged(Location loc){
           // TODO Auto-generated method stub
           ...
     }

     private void conectarWebSocket() {
           ...
           @Override
          public void onOpen() {
               ...
          }
           @Override
          public void onTextMessage(String missatge) {
               ...
          }

          @Override
          public void onClose(int code, String reason) {
               ...
          }
    }
}
```

Com acabem de dir, la dinàmica de funcionament és exactament igual que en la classe `ThreadSensors`. La connexió amb *WebSockets* es fa de la mateix manera i també implementem el mètode `parar` per parar la comunicació de dades quan aturem el servidor.

El mètode derivat de la classe pare per tractar el sensor *GPS* és `onLocationChanged`, que s'executa cada vegada que canvia la nostra posició. Quan això passa, enviarem una cadena de text al servidor remot de la forma `@Robot:gps:usuari:dades`, on aquest en farà el tractament corresponent.

3.2.7. El thread de la càmera

La classe que s'encarrega del sensor de la càmera l'anomenarem `ThreadCamara` i no hereda de cap altra classe. Això es deu a que hem volgut crear una classe que sigui capaç de configurar les funcionalitats del sensor de la càmera per després iniciar la funció en execució contínua que rep la informació, que és el mètode `PreviewCallback`. D'aquesta manera podríem iniciar i parar el sensor de la càmera si ho necessitem, ja que la càmera depen d'un objecte tipus `SurfaceView` que prové de l'activitat, i aquesta és la millor manera de tractar-lo. L'estructura general de la classe es mostra a continuació.

```
public class ThreadCamara {

      //Variables de la classe
      ...


      public ThreadCamara(SurfaceView v, String ip, String port,
String robot, String pass)
      {
            ...
      }
      private void conectarWebSocket() {
            ...
            @Override
            public void onOpen() {
                  ...
            }
            @Override
            public void onTextMessage(String missatge) {
                  ...
            }

            @Override
            public void onClose(int code, String reason) {
                  ...
            }
    }
    private void configurar()
    {
        ...
    }

    public void iniciar()
```

```
    {
          conectarWebSocket();
          configurar();
    }

    public void parar()
    {
        ...
    }

    private class camara_PreviewCallback implements PreviewCallback {

           public void onPreviewFrame(byte[] data, Camera camera)
           {
                ...
           }
    }
}
```

Com podem observar, tenim el mateix mètode `conectarWebSocket` que en les classes que tractaven la informació dels sensors i del *GPS*. Quan el `ThreadServidor` executa el mètode `iniciar` dens connectem al servidor remot amb *WebSockets* utilitzant la informació que ens passa l'usuari, exactament de la mateixa manera en que ho hem fet anteriorment. Com s'ha pogut observar a la classe de l'activitat, només es demana un port per connectar amb el servidor, però la càmera la fem anar per un de diferent. El que fem, simplement és agafar el port següent al donat, en el nostre cas el 9001. Seguidament s'executa el mètode `configurar` que consisteix en configurar el sensor de la càmera amb les prestacions que necessitem, concretament utilitzant un mode de càmera d'esports, ja que estarem en moviment, l'ús de l'autoenfocament, 30 imatges per segon i una resolució de 320x240 píxels. Per finalitzar, a la mateixa funció, assignem la recepció de dades a un objecte de la classe que creem en el mateix codi anomenada `camara_PreviewCallback` i que implementa les funcionalitats que ens aporta la *API* 2.2 d'*Android* per tractar les imatges adquirides per el sensor càmera fent ús del ja esmentat mètode `PreviewCallback`.

La funció que llança el mètode que s'encarrega de la càmera, s'anomena `onPreviewFrame` i es dispara cada vegada que es rep una nova imatge. El tractament d'aquesta és senzill, agafem l'array de bytes de la imatge i la convertim del format *YUV* (que és el format per defecte que utilitza *Android* per les imatges de la càmera) al format *JPG*, i seguidament passem el nou conjunt de bytes directament al *WebSocket* sense necessitat de guardar-lo.

3.2.8. El thread de comunicació amb la placa IOIO

Aquesta és potser la classe més important en tot el program. La seva funcionalitat és la de connectar-se amb la placa física del robot i proporcionar una connexió fiable per tenir el control en tot moment. Utilitzarem la herència d'una classe `Thread` per construir-la.

```
public class ThreadIOIO extends Thread
  {
        //Variables de la classe
        ...


        public ThreadIOIO(String ip, String port, String robot, String
  pass)
        {
              ...
        }

         private void conectarWebSocket() {
               ...
               @Override
              public void onOpen() {
                   ...
              }
             @Override
              public void onTextMessage(String missatge) {
                   ...
              }

             @Override
              public void onClose(int code, String reason) {
                   ...
               }
         }
        @Override
        public final void run()
        {
              //Conectar a IOIO
              setup();
              conectarWebSocket();
        }

         protected void setup() throws ConnectionLostException
         {
               ...
         }

        public void parar()
        {
              ...
        }
  }
```

La classe `ThreadIOIO` agafa altra vegada les dades de connexió de l'usuari fent ús del constructor d'aquesta. Quan el `ThreadServidor` executa el mètode `run` heretat de la classe `Thread`, el primer que farem serà connectar-nos a la placa *IOIO*. Això ho aconseguirem fent ús de les llibreries proporcionades pels dissenyadors de la mateixa placa, que ens permeten fer-ne un bon ús, mitjançant un objecte de la classe `IOIOFactory`, que com hem explicat anteriorment, ens permet interaccionar amb els circuits sense haver d'heretar d'una activitat per fer-ne els mètodes i funcions. Després de realitzar la connexió satisfactòriament, utilitzarem una variable booleana per saber si el sistema està connectat quan fem servir altres funcions. Seguidament hem de configurar els pins que utilitzarem de la placa, especificant-ne el seu funcionament.

Per al nostre projecte només utilitzarem dos pins de sortida, tolerants a una entrada de voltatge de 5 volts, per proporcionar un pols modulat *PWM*, que indicarà la velocitat i la direcció que ha d'utilitzar el servomotor. També utilitzarem el llum led que integra la placa perquè s'encengui o s'apagui cada vegada que rebem un missatge.

Un pols *PWM* és un senyal digital que està format per un tren de polsos, on podem trobar dos estats, 0 o 1 (en termes de voltatge estem parlant de 0 a 3.3 volts) repartits en el temps de forma constant amb un període i amplada de pols determinats.

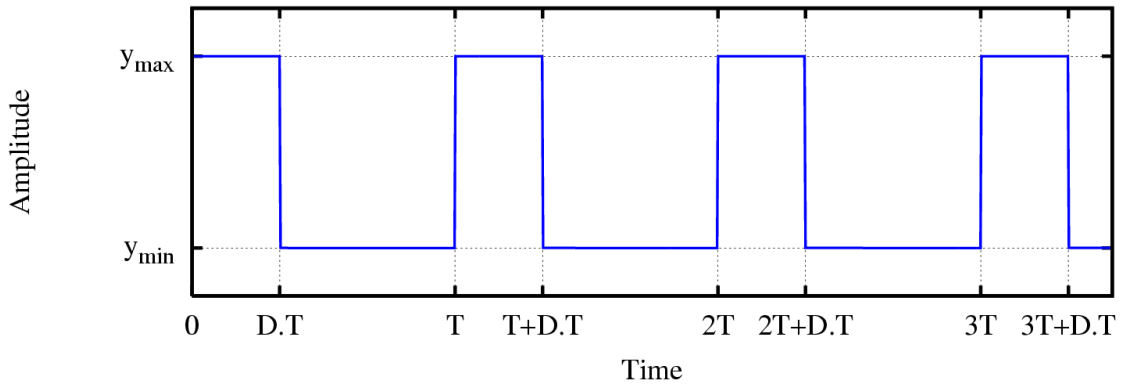


Figura 9 – Senyal PWM.

A la imatge podem observar un senyal *PWM*, on `T` és el període i `D.T` és l'amplada del pols. La relació entre el període i l'amplada de pols s'anomena cicle de treball i s'expressa en forma de percentatge. En la imatge es pot observar un cicle de treball d'un 33%. Per aquests tipus de senyals és molt més comú parlar de la freqüència del senyal, que no és res més que el número de períodes per segon. S'ha de destacar que l'únic element variable és l'amplitud del pols, ja que el període o freqüència no varien.

Bàsicament els usos d'un senyal *PWM* són dos. El primer utilitza els polsos per simular un senyal altern, de tal manera que podem obtenir una diferència de potencial positiva. Un

exemple molt comú d'aquest comportament, és aplicar aquest senyal a un led, el que ens permetrà encendre'l i apagar-lo progressivament sense notar que parpadeja, si el senyal té una freqüència prou elevada.

El segon ús més utilitzat és el de codificar informació, i és el que utilitzarem nosaltres per a controlar el servomotor. El circuit integrat del servomotor calcularà l'amplada del pols i la freqüència i descodificarà la relació com la informació que li passem.

Per al nostre servomotor *GWS S35 STD* necessitem un senyal *PWM* que treballi amb un marge de 16 a 20 milisegons, per tant donada la relació anterior entre la freqüència i el període, utilitzarem una freqüència de 50Hz. Per tal de moure'l enviarem una amplada de pols que vagi de 1 a 2 milisegons, sent 1.5 milisegons la posició de parada.

Ara que ja tenim la informació necessària que enviarem al servomotor, configurarem els dos pins de sortida (els pins 6 i 7) per enviar una senyal *PWM* amb la funció `openPwmOutput` de la llibreria *IOIO*, on li especificarem el pin i la freqüència de 50Hz. Utilitzant la funció `setPulseWidth`, aplicarem l'ample de pols a la sortida de forma constant expressat amb microsegons, per tant els valors aniran de 1000 a 1500.

Fent ús del mètode `conectarWebSocket` emprat en les anteriors classes ens connectarem al servidor remot per rebre la informació transmesa des del client amb la funció `onTextMessage`, on rebrem un missatge de la forma `@Usuari:ioio:usuari:valor1:valor2`, sent els camps `valor1` i `valor2`, els temps a aplicar a l'ample de pols per els motors esquerra i dret respectivament.

Per últim, si perdem la connexió amb la placa *IOIO*, durant el funcionament del programa, atendrem la excepció produïda i farem una reconnexió.

4. El servidor

Com el seu nom indica, la missió del servidor és la de servir les peticions que se li demanen. Utilitzarem les llibreries de *WebSockets Autobahn* amb el llenguatge de programació *Python*, i farem que totes les peticions rebudes des d'un client, siguin enviades al robot.

Degut al enorme volum de dades que pot comportar l'enviament de la informació de la càmera, obrirem dos ports per tal de minimitzar la latència.

4.1. Estructura del servidor

Anteriorment hem comentat que crearem dos ports d'entrada per rebre i enviar informació. Un port s'encarregarà de la informació de la càmera, mentre que l'altre enviarà la informació dels sensors i enviarà les ordres al robot.

La implementació de *WebSockets* utilitza una cua de memòria per tal de fer l'enviament i la recepció de dades, de manera que si ens trobem en una zona de baixa connexió, tard o d'hora la informació s'acabarà enviant, encara que sigui amb un gran decalatge. Aquesta és una característica de la connexió *TCP* i malgrat pot fer que el nostre robot vagi per via lliure durant una estona, en els pitjors del casos, al final sempre rebrà totes les ordres i acabarà per parar-se.

El cas anterior és el pitjor possible, que es donarà només davant la pèrdua de connexió, però que dins el contorn on està pensat ser executat, no hauria de passarà mai. Per tal de contemplar aquest cas, simplement implementarem un mecanisme per aturar el robot quan es perdi la connexió, i així evitar possibles danys.

Normalment un servidor fa servir diferents fils o *threads* per tal de millorar el seu rendiment i poder rebre totes les peticions i servir-les amb la màxima eficàcia. Donada l'arquitectura dels *WebSockets*, que són asíncrons, i la transferència de dades que tindrem, no hi ha la necessitat de fer un servidor multiprocés, ja que la mateixa implementació de cua dels *WebSockets* ja realitzarà aquest propòsit. A més, el fet d'utilitzar una connexió persistent i asíncrona entre client-servidor, fa inviable la implementació de *threads* per augmentar el rendiment.

Així doncs, l'estructura del servidor queda de la següent manera.

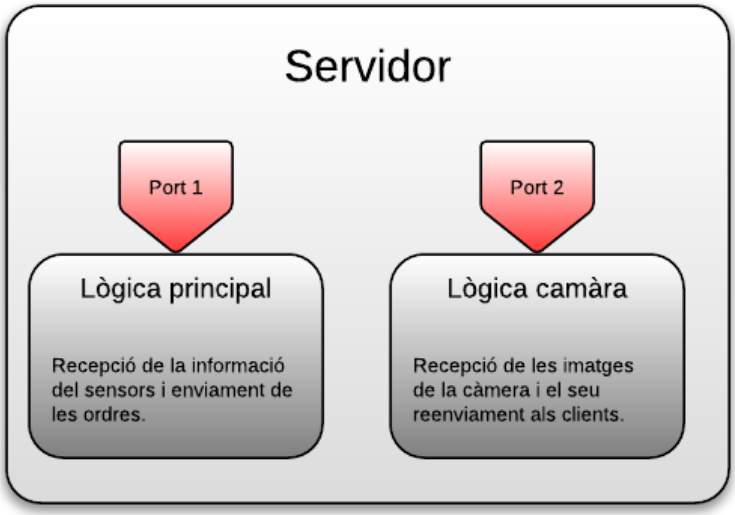


Figura 10 – Estructura del servidor.

4.2. Programació

Anteriorment em comentat que farem servir dos *WebSockets* per servir la necessitats del nostre sistema. Anem a veure doncs la programació de cadascun d'ells.

4.2.1. Servidor principal

El servidor principal s'encarrega de transmetre la informació dels sensors al client i les ordres del client al robot. Per tal de portar a terme aquesta tasca, transmetrem una cadena de text amb la informació de les diferents variables separades per comes. La única part que el servidor necessita comprovar, és l'inici de la cadena, on s'especificarà si prové d'un robot o d'un client.

Veiem l'esquema de funcionament del programa.

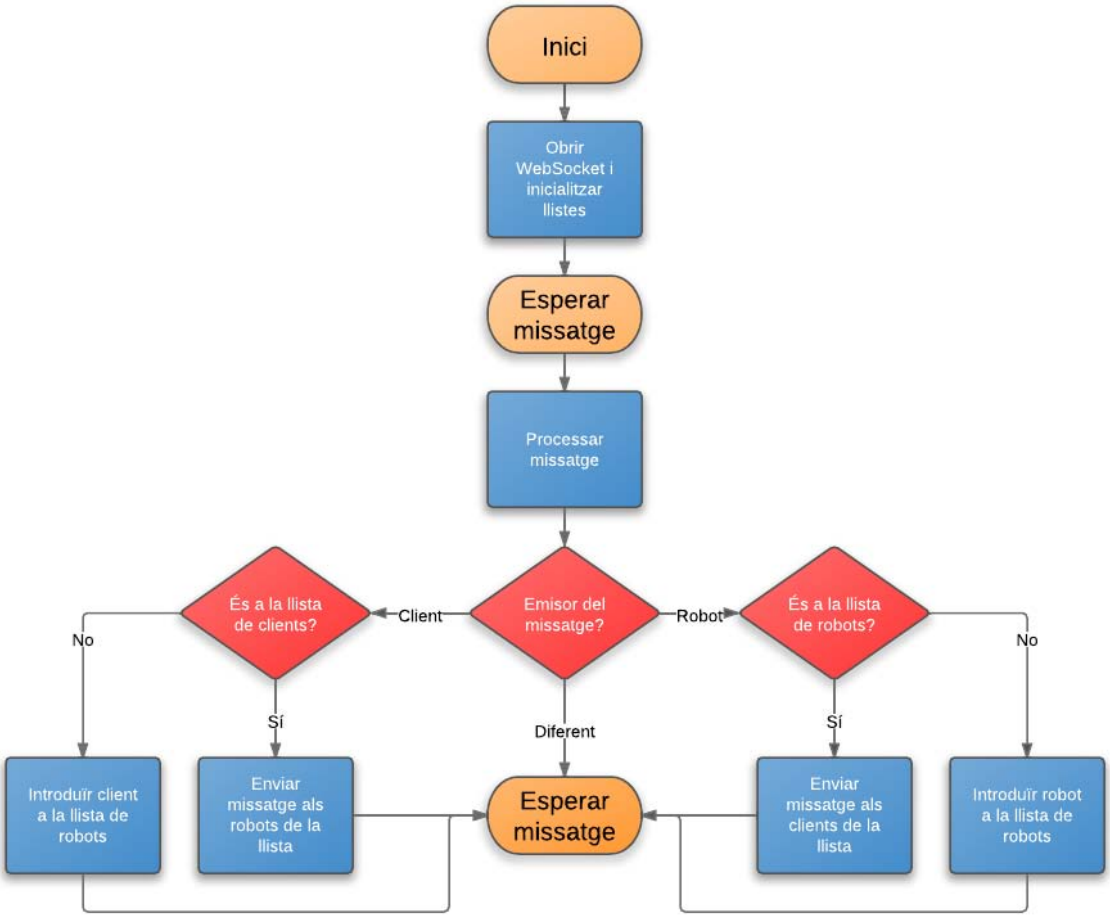


Figura 11 – Esquema de funcionament del servidor principal.

Com podem veure en l'esquema de funcionament, primer iniciem el servidor, on preparem els ports del *WebSocket* i preparem les llistes que farem servir per guardar els clients. Utilitzarem dues classes específiques de la llibreria *Autobahn WebSockets*. La primera s'encarrega d'atendre les connexions i fa la funció de *handler*, s'anomena `ProtocolServidor`.

```
class ProtocolServidor(WebSocketServerProtocol):

  def onOpen(self):
     self.factory.registrarUsuari(self)

  def onMessage(self, msg, binary):
     if not binary:
        self.factory.enviarMissatge(msg,self)

  def connectionLost(self, reason):
     WebSocketServerProtocol.connectionLost(self, reason)
     self.factory.eliminarUsuari(self)
```

Ens trobem amb diferents mètodes:

- `def onOpen(self)`: Quan es connecta un client o robot és atès per aquesta funció i executa el mètode `enviarMissatge(msg,self)` per registrar l'usuari.
- def onMessageOpen(self): Quan es rep un missatge de l'emisor que sigui, és acceptat per aquesta funció que llança el mètode `enviarMissatge(msg,self)` per fer-ne el tractamnet que faci falta.
- def `connectionLost(self, reason)`: Quan desconecta un usuari és atès per aquesta funció, i llança el mètode `eliminarUsuari(self)` per eliminar l'usuari.

La segona classe que utilitzem reacciona al protocol dels *WebSockets* rebut per la classe `ProtocolServidor` i implementa els mètodes a realitzar a la lògica del sistema. S'anomena `ServidorPrincipal`.

```
class ServidorPrincipal(WebSocketServerFactory):

 protocol =  ProtocolServidor

 def __init__(self, url):
    WebSocketServerFactory.__init__(self, url)
    self.clients_temp = []
    self.clients = []
    self.robots = []

 def registrarUsuari(self, client):
    if not client in self.clients_temp:
       print "Client conectat no confirmat: " + client.peerstr
       self.clients_temp.append(client)

 def eliminarUsuari(self, client):
    if client in self.clients:
       print "Client confirmat desconectat " + client.peerstr
       self.clients.remove(client)

    ...

 def enviarMissatge(self,msg,client):
    missatge = msg.split(':')
    if missatge[0] == '@Robot':
       ...
    elif missatge[0] == '@Usuari':
       ...
    else:
       print "Missatge ALTRE"
```

Després de fer servir el constructor de *Python* `def __init__(self, url)` per inicialitzar les variables, es presenten els diferents mètodes de la lògica del servidor principal:

- `registrarUsuari(self, client)`: Quan un nou client es connecta es col·loca a la llista `clients_temp` sempre que no estigui a la llista d'autoritzats, per després verificar-lo quan enviï un missatge.
- `eliminarUsuari(self, client)`: Quan es desconnecta un client, l'eliminem de la llista, ja sigui la de temporals `clients_temp` o la d'usuaris confirmats `clients` o `robots`.
- `enviarMissatge(self,msg,client)`: Quan rebem un missatge s'executa aquest mètode per redireccionar-lo a on pertoqui, tant al client com al robot. També s'encarrega d'introduir el client a la llista corresponent de robots o de client després de comprovar-ne l'origen.

La cadena d'informació rebuda si prové del client és de la forma `@Usuari:nom:contraseña:...` i la del robot `@Robot:nom:contraseña:...` on després de la identificació de la procedència i la informació d'autorització, segueix la resta d'informació de les variables, descrita en l'apartat de la plataforma robòtica.

En aquest projecte no es fa l'autenticació de l'usuari utilitzant per exemple una base de dades per comprovar-ne les dades, ja que és un sistema de proves pensat per un sol usuari, però l'estructura del programa està perfectament dissenyada per complir aquest requisit en futures millores de multi-usuari.

4.2.2. Servidor de la càmara

Bàsicament el servidor de la càmera segueix la mateixa estructura que el servidor principal, amb la diferència de que només rep missatges binaris amb la informació de les imatges de la càmera del telèfon mòbil i un missatge per confirmar la connexió del client. La informació binària de la imatge s'envia al client.

L'esquema de funcionament és el següent:

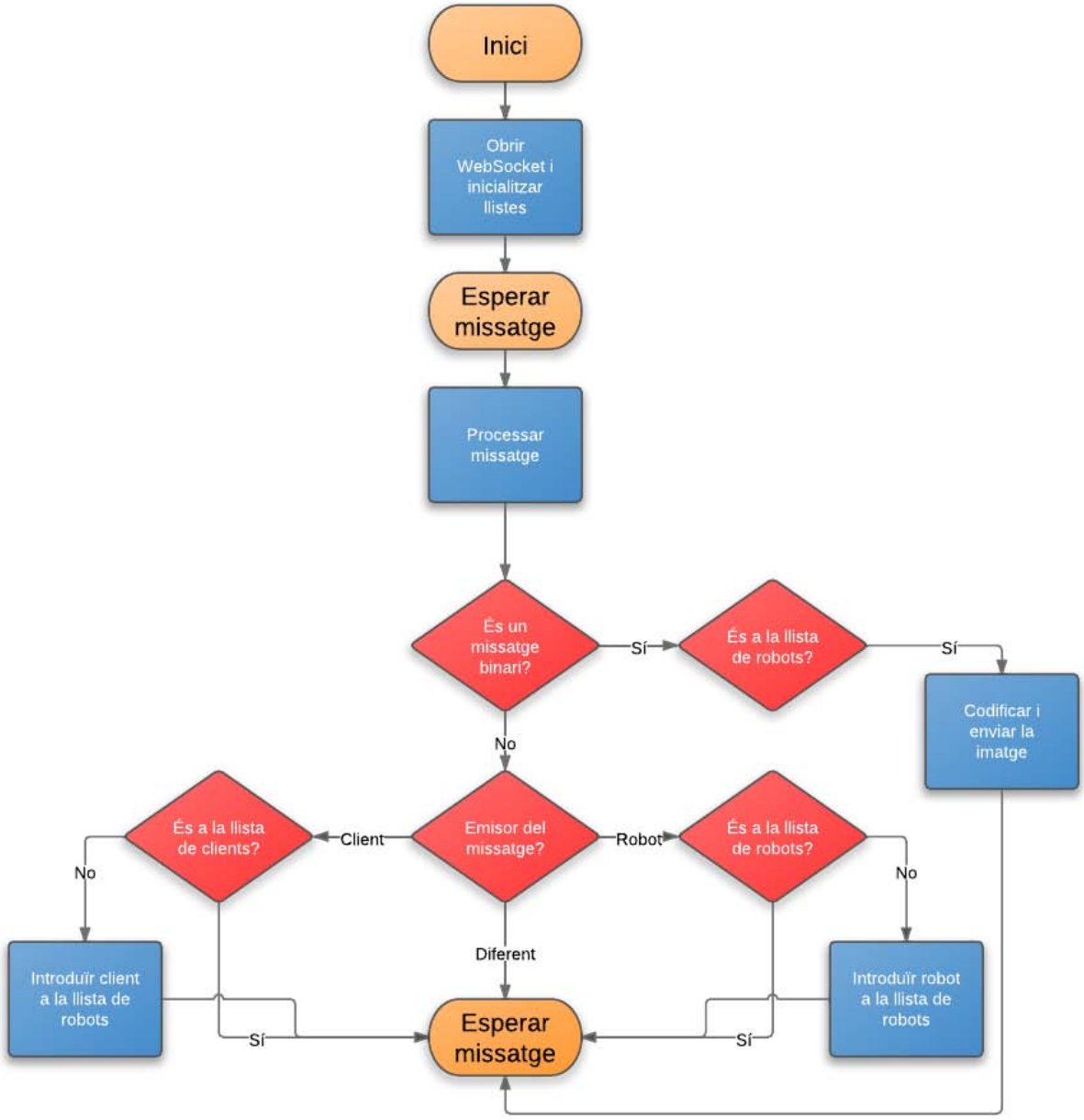


Figura 12 – Esquema de funcionament del servidor de la càmera.

Les classes del programa i els seus mètodes de forma general es presenten a continuació.

```
 class ProtocolServidor(WebSocketServerProtocol):

   def onOpen(self):
      ...

   def onMessage(self, msg, binary):
      if binary:
         self.factory.enviarImatge(msg,self)
      else:
         self.factory.registrarClient(msg,self)

   def connectionLost(self, reason):
      ...

class ServidorCamara(WebSocketServerFactory):

   protocol =  ProtocolServidor

   def __init__(self, url):
      WebSocketServerFactory.__init__(self, url)
      self.clients_temp = []
      self.clients = []
      self.robots = []
      self.imatge = "aaa"

   def registrarUsuari(self, client):
      ...

   def eliminarUsuari(self, client):
      ...

   def enviarImatge(self,msg,client):
      print "Missatge binari!!!"
      if client in self.robots:
         self.imatge = msg
         for c in self.clients:
            print "enviat a " + c.peerstr
            client.sendMessage(base64.b64encode(self.imatge),False)

   def registrarClient(self,msg,client):
      missatge = msg.split(':')
      if missatge[0] == '@Robot':
         ...
      elif missatge[0] == '@Usuari':
         ...
      else:
         print "Missatge ALTRE"
```

Com acabem de dir, la estructura és exactament la mateixa, on s'utilitzen les classes `ProtocolServidor` i `ServidorCamara` per tal d'acceptar el protocol dels *WebSockets* i executar-ne la lògica. Els mètodes són els mateixos però hi ha diferències en quant a la recepció dels missatges `onMessage`. Quan el missatge no és binari procedim a confirmar els usuaris conectats amb el mètode `registrarClient`, mentre que quan és un missatge binari, enviem el contingut al client amb el mètode `enviarImatge`.

Detallem els nous mètodes a continuació:

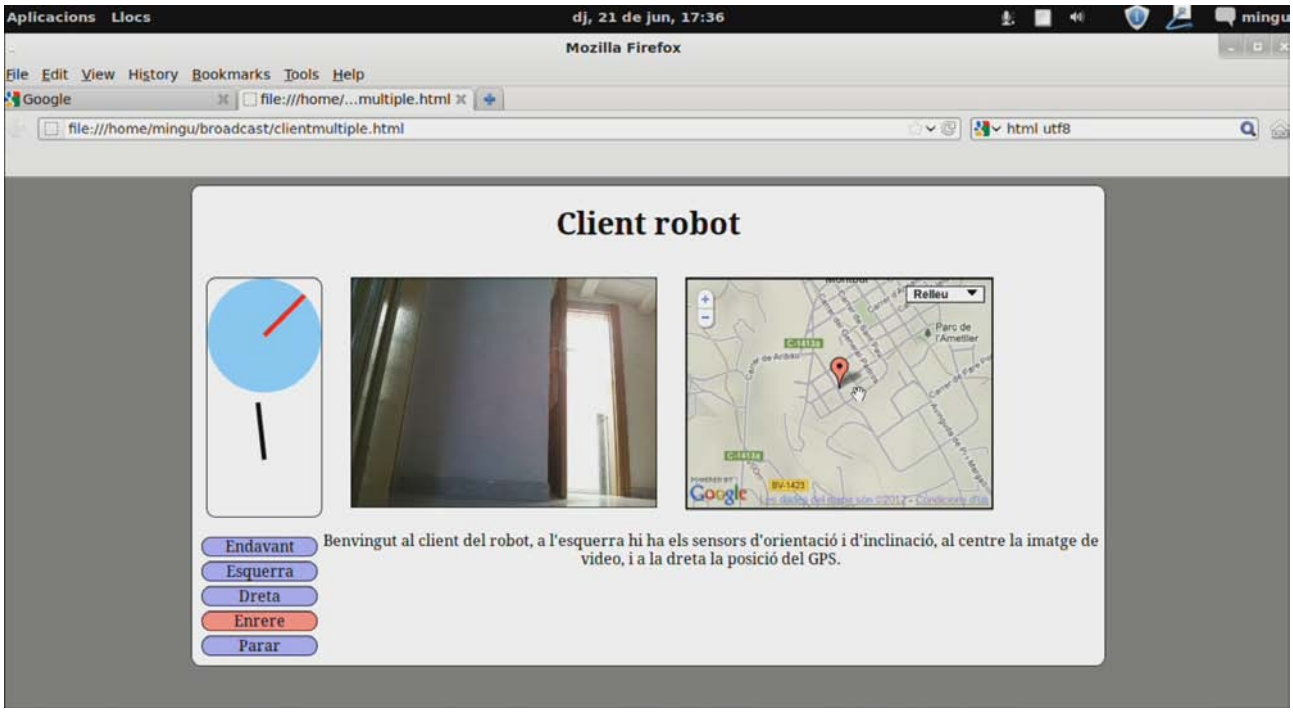
- `def enviarImatge(self,msg,client)`: És el mètode utilitzat per re-enviar la imatge al navegador després de comprovar que prové d'un robot. Per tal de que la imatge sigui compatible amb el navegador, la codifiquem en base 64 i seguidament la enviem a tots els clients connectats al sistema.
- `def registrarClient(self,msg,client)`: Aquí separem la cadena de text rebuda per comprovar si és un client o un robot, seguidament l'afegim a la llista que pertoqui.

5. El client

Anteriorment hem especificat el funcionament del sistema i les diferents parts del programa d'*Android* amb el format dels missatges que enviarà. Ara utilitzarem un navegador web per tal d'utilitzar la connexió dels *WebSockets* per rebre i enviar dades al sistema.

La manera de fer un programa que un navegador web, pugui suportar, és introduint-lo en un arxiu *HTML*. Per tant utilitzarem els llenguatges *HTML* i *CSS* per maquetar i dissenyar la pàgina que veurem, i el llenguatge *Javascript* per fer el programa executable.

Ara doncs, descriurem les diferents parts del programa client en funció del seu llenguatge de programació.



Imatge 23 – Client web del robot en funcionament.

5.1. Programació HTML i CSS

Els llenguatges *HTML* i *CSS* no són estrictament llenguatges de programació, simplement són llenguatges de marcat on existeixen un conjunt d'etiquetes que el navegador pot interpretar a forma de petites comandes i les executa.

Utilitzarem el llenguatge *HTML* per definir la estructura física del document, és a dir, el text, les seccions on es mostraran els objectes i per configurar la pàgina, el llenguatge a

utilitzar, i les llibreries i arxius necessaris que s'han d'enllaçar amb el document.

El llenguatge *CSS* és utilitzat per a definir l'estil dels elements definits pel codi *HTML*, com es veuran, el color de la lletra, la posició dels elements... és a dir, un llenguatge de disseny de la pàgina.

A continuació mostrem les principals seccions del codi *HTML* per explicar-ne les seves parts.

```
  <html>
    <head>
       <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
       <link rel="stylesheet" type="text/css" href="estil.css" />
       <script src="http://maps.google.com/maps?
file=api&amp;v=2&amp;key=CLAUGOOGLEMAPS"type="text/javascript"></scrip
t>
       <script type="text/javascript" src="jquery-
1.7.2.min.js"></script>
       <script type="text/javascript" src="jquery.gmap-1.1.0-
min.js"></script>
       <script type="text/javascript">
          ...
       </script>
    </head>
    <body>
       <div id="contenedor">
          <h1>Client robot</h1>
          <div id="esquerra">
             <div id="canvasBruixola"></div>
             <div id="canvasNivell"></div>
          </div>
          <div id="centre">
             <img src="aaa" id="laimatge" alt="" />
          </div>
          <div id="dreta">
             <div id="mapagps"></div>
          </div>
          <div id="baix">
             <input type="hidden" id="bruixolax" value="0">
             ...
             <div id="contenedorbotons">
                <div id="up">Endavant</div>
                ...
             </div>
              <div id="contenedorbotonsdret"><p>
                Text descriptiu</p>
             </div>
             <div id="contenedorbotonsbaix"></div>
          </div>
       </div>
    </body>
</html>
```

En el codi, es veuen les principals seccions que componen una pàgina *HTML*, les etiquetes `<html>` i `</html>` marquen l'inici i la fi del document. Seguidament a la secció `<head>` i `</head>` trobem la configuració de la codificació utilitzada pels caràcters de text, en el nostre cas *UTF8*, i els documents enllaçats a la pàgina, i també codi *Javascript* inclòs directament en el document per programar diferents funcionalitats. Aquí enllacem bàsicament dos tipus de documents, les llibreries de *Javascript* que utilitzarem, i el document de disseny en *CSS*. Respecte les llibreries de *Javascript*, utilitzarem la llibreria *Jquery*, la *API* de *Google Maps* i una llibreria dissenyada per treballar amb *Jquery* anomenada *Gmap* que ens simplificarà la feina de mostrar les coordenades del *GPS* amb la *API* de *Google Maps*.

Ara ens trobem amb les etiquetes `<body>` i `</body>`, que contenent tots els elements i l'estructura que es mostrarà a la pàgina. Utilitzarem uns contenedors anomenats `<div>` per ordenar els continguts i posteriorment poder ser estilitzats amb *CSS*. La distribució és simple, tenim un contenedor principal que engloba tots els elements, dins aquest tenim quatre contenedors més. Tres d'aquests estan a la mateixa línia i contindran la representació gràfica dels sensors d'orientació i d'acceleració, el vídeo de la càmera i finalment el mapa de *Google Maps*. El quart contenedor es troba situat a la línia inferior d'aquests tres i està dividit internament en dos apartats més, a l'esquerra es mostraran els botons d'estat de moviment del robot (endavant, esquerra, dreta, enrerre i parar) i a la dreta un petit text de benvinguda i explicació del sistema.

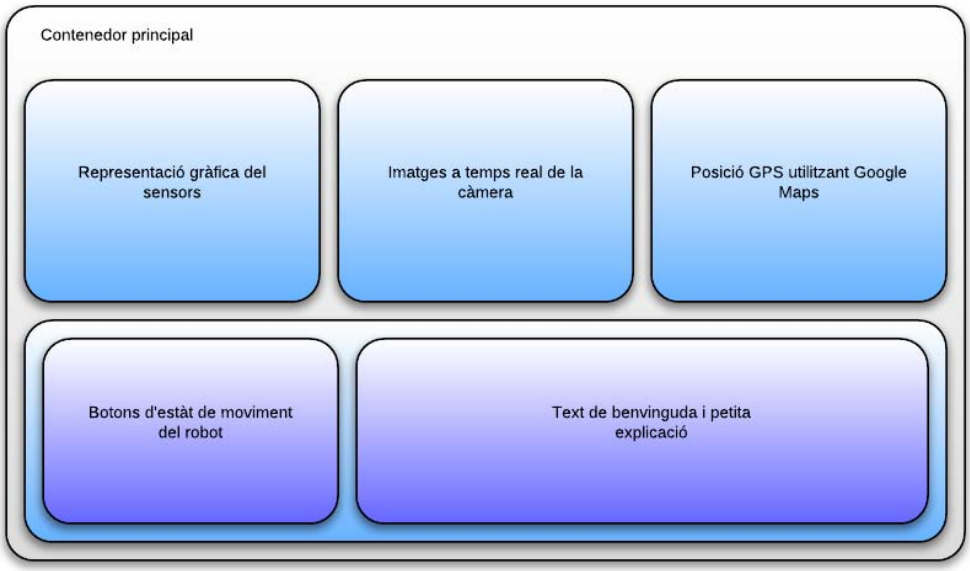


Figura 13 – Estructura HTML del document.

En quant al codi *CSS* no entrarem en detall del seu funcionament, només destacar com s'ha dit anteriorment, que la posició dels contenedors, així com tot el disseny de tota la pàgina està definida en ell. A continuació mostrem una part del codi *CSS* per fer una idea de com funciona.

```
body {
   background-color:#777;
}
#contenedor {
   -webkit-border-radius: 10px;
   -moz-border-radius: 10px;
   border-radius: 10px;
   background-color:#fff;
   margin-left: auto;
   margin-right: auto;
   width: 960px;
   text-align:center;
   border-style:solid;
   border-color:black;
   border-width:1px;
   align: center;
}

#esquerra {
   float:left;
   ...
}
```

5.2. Programació Javascript

Utilitzarem el llenguatge *Javascript* per programar tota la lògica del nostre programa client i obtenir el comportament que desitgem. La representació gràfica dels sensors, la comunicació amb *WebSockets*, l'us de *Google Maps* que mostri la posició del *GPS*, i l'ús del teclat, són exemples del que farem amb *Javascript*.

Bàsicament en el codi que situarem en la secció `<head>` de les etiquetes *HTML* utilitzarem les llibreries importades, per poder desenvolupar el que desitgem. Tenim varies funcions, la funció `rotarbruixola`, `rotarnivell`, `window.onload`, `inputmotors`, `document.onkeydown`, `resetejarColorBotons` i `checkKey`.

```
<script type="text/javascript">
    //Variables globals
    ...
    function rotarbruixola(ang){
        ...
    }

    function rotarnivell(ang){
        ...
```

```
    }

    window.onload = function() {
        //Dibuix inicial bruixola
        //Dibuix inicial nivell
        $('#stop').css("background-color",'#F78181');
        $("#mapagps").gMap({ markers: [{ latitude:   41.387917,
                    longitude: 2.1699187,
                    html: "Robot"}],
                    zoom: 18,
                    maptype: G_HYBRID_MAP,
        });
        var wsurirob = "ws://localhost:9000";
        var wsuricam = "ws://localhost:9001";

            if ("WebSocket" in window) {
               webSocketrob = new WebSocket(wsurirob);
               webSocketcam = new WebSocket(wsuricam);
            }
            else {
               webSocketrob = new WebSocket(wsurirob);
               webSocketcam = new WebSocket(wsuricam);
            }

        webSocketcam.onmessage = function(e) {
            ...
        }
        webSocketcam.onopen = function(e) {
            ...
        }
        webSocketrob.onmessage = function(e) {
            //Tractament de la cadena de text
            ...
        }

        webSocketrob.onopen = function(e) {
            ...
        }
    }

    function inputmotors(input) {
        //Valor dels motors
        ...
    }

    document.onkeydown = checkKey;
    function resetejarColorBotons() {
        ...
    }

    function checkKey(e) {
        ...
    }
</script>
```

La funció que porta la major càrrega de contingut és `window.onload`, que s'executa quan tota la pàgina ha estat carregada. Aquí definim dos objectes del tipus *WebSocket*, un per la transmissió de les dades del robot, i l'altra per les dades de la càmera. Cada element *WebSocket* té les funcions del seu protocol que hem estat veient fins ara, aquestes són `onmessage`, `onopen`, `send`. L'element `webSocketrob` s'encarrega de la informació dels sensors i les dades que enviarem al robot. Quan ens connectem, el primer que fem és enviar un missatge de registre al servidor, tal i com fèiem amb el programa del robot, i un missatge de notificació a l'usuari conforme s'ha establert la connexió. Seguidament programem el comportament a seguir quan rebem un missatge, on tractarem la cadena rebuda i agafarem els diferents elements separats pel caràcter ":". Segons el tipus de missatge, depenent del tipus de sensor, guardarem el valor en els camps de tipus text definits amb *HTML*, que modificaran el comportament gràfic i prendrem les accions pertinents.

Per al cas dels sensors d'orientació i acceleració utilitzarem gràfics vectorials per representar-los. Els gràfics vectorials es dibuixen sobre un element de tipus `<canvas>`, disponible a partir de la versió 5 de *HTML*. L'avantatge dels gràfics vectorials sobre els convencionals és que podem dibuixar qualsevol gràfic a partir d'equacions matemàtiques, sense haver de dependre d'un mapa de bits estàtics. Això ens permet fer figures i animacions gràfiques amb total llibertat, que s'escalen a la resolució utilitzada per l'usuari. Dibuixarem un cercle que contingui una busca per simular el comportament d'una brúixola, on la part superior és el nord. En quant als sensors d'acceleració, utilitzarem les seves dades per fer una altra busca, però en aquest cas representarà el nivell d'inclinació del robot a dreta i a esquerra, on la posició vertical indica una inclinació de zero graus. Els mètodes per dibuixar els elements en funció de l'angle que rebem del servidor són `rotarbruixola` i `rotarnivell`.

Quan rebem les coordenades de latitud i longitud del *GPS*, utilitzarem la *API* de *Google Maps* per representar-les en un mapa. És important saber que necessitem una clau d'accés que proporcionen els serveis de *Google* gratuïtament per poder utilitzar aquesta *API*. També utilitzarem la llibreria *Jquery* i *Gmap*, la primera ens simplifica la vida per treballar amb *Javascript*, proporcionant-nos diverses funcionalitats extres, i la segona ens permet treballar amb la llibreria de *Google Maps* de manera molt simple. Així que quan rebem les coordenades simplement agafem l'element del contenedor del mapa i li carreguem un objecte *Gmap* amb les seves dades per veure'n la representació.

L'objecte `webSocketcam` controla la connexió del *WebSocket* de la càmera, i quan ens connectem també enviem un missatge de registre al servidor i notifiquem a l'usuari de la connexió. Seguidament configurem el comportament en la recepció d'un missatge d'aquest tipus, que conté una imatge en format binari codificat en base 64. Agafem el contingut i el carreguem en una etiqueta del tipus imatge `<img>` per representar-lo gràficament.

La resta d'operacions que s'executen a la funció `window.onload` serveixen per

inicialitzar els elements gràfics dels sensors a zero, així com la imatge del vídeo de color negre, i el mapa *GPS* situat en un punt concret, en el nostre cas a la Plaça Catalunya de Barcelona. També configurem el botó d'estat de moviment `parar` com actiu.

La resta de funció `document.onkeydown` controla quan és accionada una tecla i de quina es tracta. Quan s'acciona, passa les dades a la funció `checkKey` que s'encarregarà de executar un mètode concret depenent de la tecla. En el nostre cas, utilitzarem els cursors del teclat per anar endavant, esquerra, dreta o enrerre, i qualsevol altra tecla per parar. Quan premem una tecla utilitzarem el mètode `resetejarColorBotons` per posar tots els estats de moviment com a no activats i el mètode `inputmotors` per enviar el valor dels motors al servidor, que dependran de la tecla en qüestió.

Aquesta és tota la programació realitzada amb *Javascript*, per poder controlar tots els esdeveniments que influeixen en el comportament del nostre client i que ens proporciona controlar al robot.

Per enviar codi extra des del client al servidor, utilitzarem la finestra de debugació que porten els navegadors moderns, i que ens permeten afegir codi *Javascript* en el *DOM* del document. Així, utilitzarem per exemple els elements de *Jquery* per seleccionar l'objecte del valor dels sensors (`$(sensor).val`) i la funció `inputmotors` per enviar la resposta de moviment.

6. Test de funcionament

Durant la realització del projecte és inherent anar pas a pas i provar mètode a mètode les funcionalitats que volem implementar. Pel cas del programa d'*Android* hem fet ús de la classe `android.util.Log` que ens permet debugar el programa veient la informació que ens interessa per la consola d'*Eclipse*.

Pel cas dels servidors, també hem utilitzat la clase log `from twisted.python import log` per veure els missatges tant del robot com del client i poder observar que tot funciona correctament.

En quant a la part del client, la codificació de *HTML* i *CSS* es testeja codificant el resultat i comprovant-ne els resultats al navegador, per veure si són els desitjats. El codi *Javascript* l'hem testejat utilitzant les opcions de debugació del mateix navegador, que indiquen totes les funcionalitats programats i on està l'error en cas que succeeixi.

Posar el sistema en funcionament és molt senzill, primer hem de posar en funcionament els dos servidors que utilitzarem, el servidor principal que s'encarrega de les dades des robot `servidor.py`, i el servidor de la càmera `servidorcam.py`, que tractarà les imatges d'aquesta. Executant-los n'hi haurà prou per posar-los en funcionament, però si volem canviar el port que utilitzaran, que per defecte són el 9000 i el 9001, haurem de canviar el codi de la part inferior, modificant el valor `"ws://localhost:9000"` en cada un d'ells.

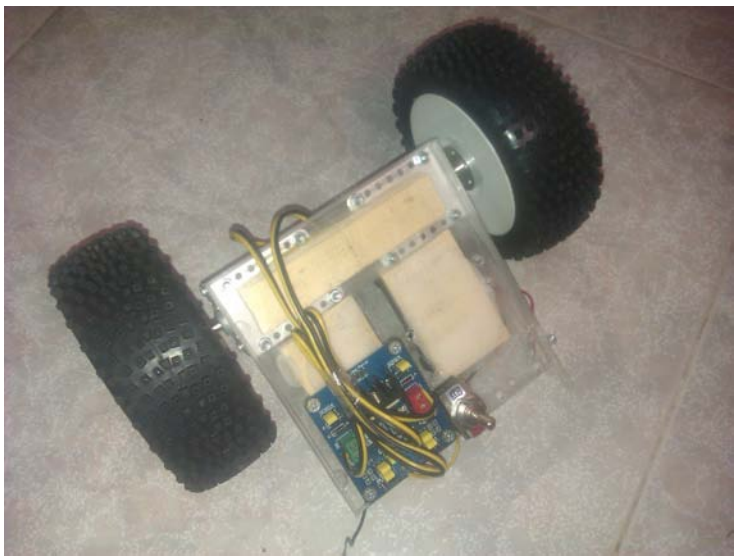
Seguidament inicialitzarem el client simplement visualitzant la pàgina web en el navegador compatible amb *WebSockets*. També podríem inicialitzar abans el programa del robot, ja que al ser una comunicació asíncrona, les dades les rebrem quan es connecti el robot.

Per últim per a la posada en marxa del robot, hem de connectar les bateries al sistema electrònic, després connectar el cable *USB* de la placa *IOIO* al terminal *Android*. Si la connexió és correcta, la llum led de la placa *IOIO* parpellejarà indicant la configuració satisfactòria. S'ha de tenir en compte, que per poder comunicar el terminal *Android* amb la placa *IOIO* hem de configurar el mòbil perquè utilitzi la opció per debugar, si no, no podrem enviar cap missatge entre ells, ja que no estarà activat el protocol *ADB*. Ara que ja ho tenim tot connectat, simplement executarem l'aplicació `Robot` en el terminal i després d'indicar la direcció i port del servidor així com el nostre nom d'usuari, ja podrem prémer el botó `start` i iniciar el servidor robot. Com hem comentat abans, hi ha també un camp per introduir una contrasenya on el valor és indiferent, ja que està pensat per a futures implementacions. Quan la comunicació sigui iniciada, apareixerà un missatge de notificació a la pantalla.

Si tots els passos anteriors són satisfactoris visualitzarem la transmissió de missatges a les consoles dels servidors, així com les dades al programa client en el navegador.

Durant la realització del projecte, he utilitzat el mòbil de la marca *HTC Wildfire*, que ha funcionat a la perfecció. Vaig notar però que a afegia nous sensors al programa, disminuïa la fluïdesa de la recepció en el programa client, fent-me pensar que la connexió *WebSocket* es saturava massa. Res més lluny de la realitat, per a posteriors proves, vaig utilitzar un mòbil de la marca *Samsung Galaxy SII*, que em va demostrar que no només el programa *Android* era fàcilment portable, si no que a més la latència de transmissió de dades era quasi inapreciable, observant un decalatge d'uns 12ms.

Per poder completar l'estudi de viabilitat, així com construir la plataforma robot, vaig provar diferents plaques de control, com la placa *Android* i la placa *USB Host Shield*. També vaig construir una segona plataforma robot utilitzant motors de corrent continua i un driver adient per controlar-los, provant així que les possibles modificacions i funcionalitat que es poden derivar de la placa *IOIO* són realment àmplies.



Imatge 24 – Estructura robòtica amb motors DC.

De la part del servidor, també he testejat d'utilitzar-lo amb una *IP* que no sigui local i provar la connexió a través d'*Internet*, obrint els ports del *router* amb el protocol *NAS*. He pogut observar que incrementa el decalatge sobretot en la transmissió de les dades la càmera entre altres coses, ja que el servei de pujada de dades del meu operador és molt baix. Utilitzant la connexió *WIFI* per la connexió a través d'*Internet*, també hi ha un decalatge, però és mínim, indicant-nos que podem controlar el robot des de qualsevol lloc del món on

tinguem *Internet*, demostrant la alta potència del sistema.

A l'annex hi ha tots un seguit de vídeos que demostren els diferents tests utilitzats per el desenvolupament del robot, així com un vídeo demostratiu del funcionament de projecte definitiu.

7. Conclusions i possibles ampliacions

Amb la realització d'aquest projecte hem creat un sistema software i hardware capaç de dirigir i controlar una plataforma robòtica fàcilment manejable i ampliable. Un cop el robot es connecta a una xarxa o *Internet* mitjançant el terminal *Android*, podem transmetre informació al servidor de manera local o remota, ja sigui en una *IP* d'una xarxa local, o una direcció web estàtica. Hem aconseguit complir tots els objectius creant un sistema sòlid i de grans prestacions per un preu molt reduït.

A nivell subjectiu durant la realització d'aquest projecte he pogut gaudir molt de la feina feta mentre veia com progressava el projecte. El fet d'utilitzar una tecnologia relativament nova com els *WebSockets* és una de les parts que més dificultats m'ha suposat, degut a la falta d'informació i llibreries estables i òptimes. El segon apartat que també ha suposat un esforç afegit ha estat la transmissió de vídeo de la càmera al servidor, i d'aquest al client. Bàsicament la dificultat radicava en que no havia fet mai cap projecte d'aquest estil, a més el canvi de la codificació del vídeo per passar-lo a un format més suportat globalment i fer tot aquesta operació a temps real i amb *streamming*, han els punts d'inflexió. Espero que el projecte serveixi per a gent amb bones idees, però de vegades amb poc recursos, per poder posar-les en pràctica, així com a base per reduir la feina inicial a fer durant una tasca i poder-se centrar en el veritable objectiu.

També m'agradaria destacar la enorme potència de càlcul i les eines que aporta tenir un *smartphone* com a cervell del robot, poder tenir en un mateix sistema un sensor *GPS*, una brúixola, sensors d'inclinació i imatges d'una càmera sense comptar els possibles sensors extres que es poden afegir, pel cost total que té el projecte, és tot un luxe.

De cara a possibles ampliacions, bàsicament ens centraríem en la interfície d'usuari, afegint opcions extres, com el ja mencionat registre d'usuaris, o funcions per activar i desactivar recursos del terminal Android per optimitzar energia. També podem implementar diverses funcionalitats com per exemple la llum led del telèfon, la reproducció d'àudio a través del terminal, afegir nous sensors hardware a la plataforma robòtica o fins i tot modificant l'estructura, per controlar qualsevol dispositiu físic que se'ns pugui ocórrer. Les possibilitats són infinites. El client del sistema s'ha decidit fer utilitzant un navegador web per tal d'obtenir la màxima portabilitat possible, però si és necessari, i ja que el servidor funciona amb el llenguatge *Python*, podem fer un programa client que no depengui del navegador web, aconseguint així més potència de càlcul de cara a les nostres necessitats i no estar subjectes a les limitacions del llenguatge Javascript.

Així que aquí tanco la explicació de tot el sistema i el deixo en mans d'aquells que el vulguin utilitzar, on l'únic que realment limita els nostres projectes és la nostra imaginació.

8. Bibliografia i referències

8.1. Manuals i llibres

Els manuals i els llibres utilitzats en aquest projecte han estat els següents:

- *González Duque, Raúl (2009). Python para todos. Creative Commons.*
- *Brian W. Evans (2008). Arduino Programming Notebook. Standard Copyright License.*
- *Ableson, F.; Collins, C.; Sen, R. (2010) Android: Guía para desarrolladores. Anaya Multimedia.*

8.2. Pàgines web

Mitjançant internet s'ha obtingut la majoria d'informació del projecte:

- *Descripció i informació de la plataforma de desenvolupament arduino (Castellà)*
  http://www.arduino.cc/es/
- *Informació per al desenvolupament d'aplicacions en Android (Anglès)*
  http://developer.android.com/index.html
- *Mini framework en Python pel desenvolupament del servidor (Anglès)*
  http://www.cherrypy.org/
- *Normativa de projectes d'enginyeria tècnica.*
  http://www.uab.cat/Document/541/595/Normativa_PFCNovembre2010.pdf (Català)
- *Informació de la placa IOIO i la llibreria (Anglès)*
  https://github.com/ytai/ioio/wiki
- *Informació dels WebSockets (Castellà)*
  http://es.wikipedia.org/wiki/WebSocket
- *Compatibilitat de navegadors i WebSockets (Anglès)*
  http://caniuse.com/websockets
- *Per consultar conceptes concrets, enciclopèdia interactiva (Castellà)*
  http://es.wikipedia.org/wiki/Wikipedia:Portada
- *Informació dels dispositius mòbils i el mercat (Anglès)*
  http://www.gartner.com/it/page.jsp?id=1848514
- *Llibreria de WebSockets Autobahn (Anglès)*
  http://autobahn.ws/

- *Llibreria websocket-android-phonegap*
  (Anglès)
  https://github.com/anismiles/websocket-android-phonegap
- *Llibreria jwebsocket*
  (Anglès)
  http://jwebsocket.org/
- *Servei Lucidchart*
  (Anglès)
  https://www.lucidchart.com/
- *Programa de disseny de circuits Fritzing*
  (Anglès)
  http://fritzing.org/
- *Llibreria Jquery*
  (Anglès)
  http://jquery.com/
- *Llibreria Gmap*
  (Anglès)
  http://gmap.nurtext.de/
- *API de Google Maps*
  (Anglès)
  https://developers.google.com/maps/?hl=es
- *Servei Libreoffice*
  (Anglès)
  http://es.libreoffice.org/
- *Eclipse IDE*
  (Anglès)
  http://www.eclipse.org/

9. Annex

9.1. Índex d'imatges i taules

Imatges:

Figures:

Taules:

9.2. Contingut del CD

Dins de CD entregat conjuntament a la memòria hi ha tots els arxius necessaris per poder posar en marxa el sistema. A l'arrel trobarem l'arxiu pdf de la memòria i un arxiu llegeix-me que explica el contingut, juntament amb les carpetes que tenen el contingut.

- Memòria Domènec Castillo.pdf
  - Arxiu de la memòria en pdf, signat digitalment pel director del projecte.
- Llegeixme.txt
  - Arxiu en format text que explica els continguts del CD.
- Vídeos
  - Carpeta que conté tots els vídeos demostratius dels tests, així com el del projecte en funcionament.

- Documentació
  - Aquesta carpeta conté arxius de documentació utilitzats en el projecte, com els datasheets dels components electrònics.
- Programació
  - Conté tres carpetes diferents, on hi ha el codi de les parts del projecte, robot, client i servidor.

Memòria del projecte realitzat per

Domènec Castillo Castells

Sabadell, Juny del 2012