UNIVERSIDAD
POLITECNICA
DE VALENCIA

Programa Cliente / Servidor para control de un Robot con ruedas Omnidireccionales.

Escola Tècnica
Superior d'Enginyeria
Informàtica

PROYECTO FINAL CARRERA

Realizado por Laureano Ortega Estarlich

Dirigido por Ángel Valera Fernández

Glosario

1. Introducción

1. Introducción
2. Objetivos

– DESARROLLO TEORICO

2. Robótica

1. Introducción a la robótica
2. Autómatas
3. Robótica
4. Breve historia de la robótica
5. Clasificación de los robots
6. Tipos de robots
    1. Robots industriales
        1. Manipuladoras
        2. Robots de repetición
        3. Robots de control por computador
    2. Robots Inteligentes
    3. MicroRobots
7. Robótica móvil
    1. Clasificación de los robots móviles
        1. Robots rodantes
        2. Robots andantes
        3. Robots reptadores
        4. Robots nadadores
        5. Robots voladores
8. Programas del controlador del robot
9. Componentes
    1. Estructura
    2. Sensores
    3. Actuadores
10. WIFI
    1. Inicios
    2. Como trabajan las redes WLAN
    3. Configuraciones para redes de radiofrecuencia
    4. Asignación de canales
    5. Seguridad

3. **Robotino Festo**

    1. Introducción y Características
    2. Entorno de programación
    3. Placa de control EA09
    4. Otras características
        1. Posibilidades de ampliación
        2. Programación del micro-controlador
        3. Técnica de regulación
        4. Hardware en bucle

4. **Aplicación Cliente-Servidor**

    1. Introducción
    2. Definición
    3. Ventajas e inconvenientes.

5. **Otras arquitecturas**

    1. Peer-to-Peer
        1. Definición
        2. Características deseables
        3. Problemática
    2. Cliente-Cola-Cliente
        1. Definición
        2. Problemática

6. **Netbeans**

    1. Introducción
    2. NetBeans IDE

– **DESARROLLO PRACTICO**

– **Problemática a resolver**

1. **Cliente**

    1. Introducción
    2. Inicio Conexiones
    3. Fin Conexión
    4. Envió Datos

5. Control Manual
6. Log del Servidor

2. Servidor

1. Introducción
2. Inicio servidor
3. Conclusión

3. Clase Servidor

1. Main
2. Hilo Servidor
3. Circular
4. Bezier

– INSTALACION SERVIDOR

1. Programas

1. Putty
    1. Definición
    2. Características
    3. Ejecución del servidor

2. WinSCP

1 Introducción

1.- Introducción

La presente memoria se dividirá en 2 apartados claves para presentar el desarrollo e complementación de un cliente-servidor para controlar un Robot

La memoria se dividirá en 2 apartados claramente diferenciados, el desarrollo teórico y el desarrollo practico de la misma.

En el desarrollo teórico, haremos un breve repaso a la historia de la robótica, los diferentes tipos de robot. Se realizara una descripción del robot usado para realizar el proyecto (Festo robotino), dentro de la cual se indicaran sus características técnicas. También realizaremos una descripción de las solución tomada para realizar la comunicación entre cliente/robot : Arquitectura Cliente-Servidor, con otras arquitecturas que se podrían haber usado (Arquitectura Peer-to-Peer, Arquitectura Cliente-Cola-Cliente).

En el desarrollo practico, se indicaran los pasos que se han seguido para la creación de la aplicación servidora y cliente. Dentro de la parte cliente se incluirá una descripción del código y de la interfaz gráfica del mismo, se incluirán unas indicaciones para hacer uso de la aplicación. En la aplicación Servidora se describirá los pasos seguidos en su creación, incluyendo los segmentos de código con su correspondiente explicación

A parte de las 2 partes teórica y practica, se explicaran las acciones que puede realizar el robot (Curva de Bezier).

2.- Objetivos

Los objetivos marcados para este proyecto serian los siguientes :

- Creación de un servidor en el robot.
- Creación de una aplicación cliente la cual se conectara al servidor.
- La conexión se realizara mediante Wifi.
- El Servidor tratara a los clientes de manera independiente. Evitando que si esta ejecutando una orden otro cliente pueda acceder a el.

DESARROLLO TEORICO

1 Robótica

1.- Introducción a la robótica.

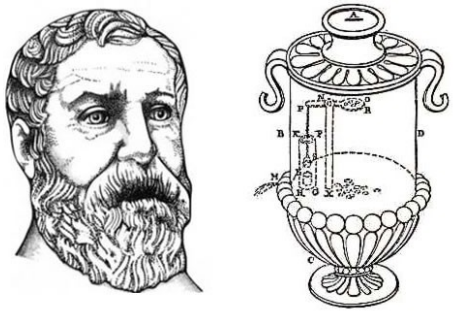
El hombre durante toda su historia ha convivido con el deseo de construir máquinas capaces de realizar tareas que facilitasen su trabajo. Desde hace cientos de años antes de Cristo, ya se intentaban crear dispositivos, denominados artefactos o máquinas, que tuvieran un movimiento sin fin y que no estuvieran controlados ni supervisados por personas.

2.- Autómatas.

Los primeros autómatas que aparecen en la historia son ingenios mecánicos, más o menos complicados, que desarrollaban una tarea de forma continua. Sin embargo, las primeras máquinas construidas no tenían una utilidad práctica, sino que su principal objetivo era entretener a sus dueños. Generalmente funcionaban por medio de movimientos ascendentes de aire o agua caliente. El vertido progresivo de un líquido o la caída de un peso provocaba rupturas de equilibrio en diversos recipientes provistos de válvulas; otros mecanismos se basaban en palancas o contrapesos. Mediante sistemas de este tipo se construían pájaros artificiales que podían “cantar" o "volar", o puertas que se abrían solas.

El origen de los autómatas se remonta al Antiguo Egipto, donde las estatuas de algunos de sus dioses despedían fuego por los ojos, poseían brazos mecánicos manejados por los sacerdotes del templo o emitían sonidos cuando los rayos del sol los iluminaba. Estos ingenios pretendían causar temor y respeto entre la gente del pueblo.

Sin embargo, los primeros datos descriptivos acerca de la construcción de un autómata aparecen en el siglo I. El matemático, físico e inventor griego Herón de Alejandría describe múltiples ingenios mecánicos en su libro Los Autómatas, por ejemplo aves que vuelan, gorjean y beben.



Herón de Alejandría junto uno de sus inventos, un dispensador de agua sagrada operado por monedas.

Por lo tanto podemos definir que un robot es un autómata con diferentes configuraciones y elementos físicos para realizar las acciones para la cual es desarrollado.

3.- Robótica

La robótica es la ciencia y la tecnología de los robots. Se ocupa del diseño, manufactura y aplicaciones de los robots. La robótica combina diversas disciplinas como son: la mecánica, la electrónica, la informática, la inteligencia artificial y la ingeniería de control. Otras áreas importantes en robótica son el álgebra, los autómatas programables y las maquinas de estados.
El término robot se popularizó con el éxito de la obra RUR (Robots Universales Rossum, escrita por Karel Capek en 1920. En la traducción al inglés de dicha obra, la palabra checa *robota*, que significa *trabajos forzados*, fue traducida al inglés como *robot*.

4.- Breve historia de la robótica.

FECHA	DESARROLLO
SigloXVIII.	A mediados del J. de Vaucanson construyó varias muñecas mecánicas de tamaño humano que ejecutaban piezas de música.
1801	J. Jaquard invento su telar, que era una máquina programable para la urdimbre.
1805	H. Maillardet construyó una muñeca mecánica capaz de hacer dibujos.
1946	El inventor americano G.C Devol desarrolló un dispositivo controlador que podía registrar señales eléctricas por medio magnéticos y reproducirlas para accionar un máquina mecánica. La patente estadounidense se emitió en 1952.
1951	Trabajo de desarrollo con teleoperadores (manipuladores de control remoto) para manejar materiales radiactivos. Patente de Estados Unidos emitidas para Goertz (1954) y Bergsland (1958).
1952	Una máquina prototipo de control numérico fue objetivo de demostración en el Instituto Tecnológico de Massachusetts después de varios años de desarrollo. Un lenguaje de programación de piezas denominado APT (Automatically Programmed Tooling) se desarrolló posteriormente y se publicó en 1961.
1954	El inventor británico C. W. Kenward solicitó su patente para diseño de robot. Patente británica emitida en 1957.
1954	G.C. Devol desarrolla diseños para Transferencia de artículos programada. Patente emitida en Estados Unidos para el diseño en 1961.
1959	Se introdujo el primer robot comercial por Planet Corporation. El cual estaba controlado por interruptores de fin de carrera.
1960	Se introdujo el primer robot 'Unimate", basada en la transferencia de artic, programada de Devol. Utilizan los principios de control numérico para el control de manipulador y era un robot de transmisión hidráulica.
1961	Un robot Unimate se instaló en la Ford Motors Company para atender una máquina de fundición de troquel.

FECHA	DESARROLLO
1968	Un robot móvil llamado 'Shakey" se desarrollo en SRI (standford Research Institute), estaba provisto de una diversidad de sensores así como una cámara de visión y sensores táctiles y podía desplazarse por el suelo
1971	El 'Standford Arm", un pequeño brazo de robot de accionamiento eléctrico, se desarrolló en la Standford University.
1973	Se desarrolló en SRI el primer lenguaje de programación de robots del tipo de computadora para la investigación con la denominación WAVE. Fue seguido por el lenguaje AL en 1974. Los dos lenguajes se desarrollaron posteriormente en el lenguaje VAL comercial para Unimation por Víctor Scheinman y Bruce Simano.
1974	ASEA introdujo el robot Irb6 de accionamiento completamente eléctrico.
1974	Kawasaki, bajo licencia de Unimation, instaló un robot para soldadura por arco para estructuras de motocicletas.
1974	Cincinnati Milacron introdujo el robot T3 con control por computadora.
1975	El robot 'Sigma" de Olivetti se utilizó en operaciones de montaje, una de las primitivas aplicaciones de la robótica al montaje.
1976	Un dispositivo de Remopte Center Compliance (RCC) para la inserción de piezas en la línea de montaje se desarrolló en los laboratorios Charles Stark Draper Labs en Estados Unidos.
1978	El robot T3 de Cincinnati Milacron se adaptó y programó para realizar operaciones de taladro y circulación de materiales en componentes de aviones, bajo el patrocinio de Air Force ICAM (Integrated Computer- Aided Manufacturing).
1978	Se introdujo el robot PUMA (Programmable Universal Machine for Assambly) para tareas de montaje por Unimation, basándose en diseños obtenidos en un estudio de la General Motors.
1979	Desarrollo del robot tipo SCARA (Selective Compliance Arm for Robotic Assambly) en la Universidad de Yamanashi en Japon para montaje. Varios robots SCARA comerciales se introdujeron hacia 1981.
1980	Un sistema robótico de captación de recipientes fue objeto de demostración en la Universidad de Rhode Island. Con el empleo de visión de máquina el sistema era capaz de captar piezas en orientaciones aleatorias y posiciones fuera de un recipiente.

FECHA	DESARROLLO
1981	Se desarrolló en la Universidad de Carnegie- Mellon un robot de impulsión directa. Utilizaba motores eléctricos situados en las articulaciones del manipulador sin las transmisiones mecánicas habituales empleadas en la mayoría de los robots.
1982	IBM introdujo el robot RS-1 para montaje, basado en varios años de desarrollo interno. Se trata de un robot de estructura de caja que utiliza un brazo constituido por tres dispositivos de deslizamiento ortogonales. El lenguaje del robot AML, desarrollado por IBM, se introdujo también para programar el robot SR-1.
1983	Informe emitido por la investigación en Westinghouse Corp. bajo el patrocinio de National Science Foundation sobre un sistema de montaje programable adaptable (APAS), un proyecto piloto para una línea de montaje automatizada flexible con el empleo de robots.
1984	Robots 8. La operación típica de estos sistemas permitía que se desarrollaran programas de robots utilizando graficos interactivos en una computadora personal y luego se cargaban en el robot.

5.- Clasificación de los robots

La potencia del software en el controlador determina la utilidad y flexibilidad del robot dentro de las limitantes del diseño mecánico y la capacidad de los sensores. Los robots han sido clasificados de acuerdo a su generación, a su nivel de inteligencia, a su nivel de control, y a su nivel de lenguaje de programación. Éstas clasificaciones reflejan la potencia del software en el controlador, en particular, la sofisticada interacción de los sensores. La generación de un robot se determina por el orden histórico de desarrollos en la robótica. Cinco generaciones son normalmente asignadas a los robots industriales. La tercera generación es utilizada en la industria, la cuarta se desarrolla en los laboratorios de investigación, y la quinta generación es un gran sueño.

1.- Robots Play-back, los cuales regeneran una secuencia de instrucciones grabadas, como un robot utilizado en recubrimiento por spray o soldadura por arco. Estos robots comúnmente tienen un control de lazo abierto.

2.- Robots controlados por sensores, estos tienen un control en lazo cerrado de movimientos manipulados, y hacen decisiones basados en datos obtenidos por sensores.

3.- Robots controlados por visión, donde los robots pueden manipular un objeto al utilizar información desde un sistema de visión.

4.- Robots controlados adaptablemente, donde los robots pueden automáticamente re-programar sus acciones sobre la base de los datos obtenidos por los sensores.

5.- Robots con inteligencia artificial,donde las robots utilizan las técnicas de inteligencia artificial para hacer sus propias decisiones y resolver problemas.

La Asociación de Robots Japonesa (JIRA) ha clasificado a los robots dentro de seis clases sobre la base de su nivel de inteligencia:

1.- Dispositivos de manejo manual, controlados por una persona.

2.- Robots de secuencia arreglada.

3.- Robots de secuencia variable, donde un operador puede modificar la secuencia fácilmente.

4.- Robots regeneradores, donde el operador humano conduce el robot a través de la tarea.

5.- Robots de control numérico, donde el operador alimenta la programación del movimiento, hasta que se enseñe manualmente la tarea.

6.- Robots inteligentes, los cuales pueden entender e interactuar con cambios en el medio ambiente.

6.- Tipos de robot

6.1.- Robots Industriales

La creciente utilización de robots industriales en el proceso productivo, ha dado lugar al desarrollo de controladores industriales rápidos y potentes, basados en microprocesadores, así como un empleo de servos en bucle cerrado que permiten establecer con exactitud la posición real de los elementos del robot y su desviación o error. Esta evolución ha dado origen a una serie de tipos de robots, que se citan a continuación:

6.1.1.- Manipuladores

Son sistemas mecánicos multifuncionales, con un sencillo sistema de control, que permite gobernar el movimiento de sus elementos de los siguientes modos:

- Manual: Cuando el operario controla directamente la tarea del manipulador.

- De secuencia fija: cuando se repite, de forma invariable, el proceso de trabajo preparado previamente.

- De secuencia variable: Se pueden alterar algunas características de los ciclos de trabajo

Existen muchas operaciones básicas que pueden ser realizadas de forma óptima mediante manipuladores. Por ello, estos dispositivos son utilizados generalmente cuando las funciones de trabajo son sencillas y repetitivas.

6.1.2.- Robots de repetición o aprendizaje

Son manipuladores que se limitan a repetir una secuencia de movimientos, previamente ejecutada por un operador humano, haciendo uso de un controlador manual o un dispositivo auxiliar. En este tipo de robots, el operario durante la fase de enseñanza se vale de una pistola de programación con diversos pulsadores o teclas, o bien de joysticks, o bien utiliza un maniquí, o desplaza directamente la mano del robot. Actualmente, los robots de aprendizaje son los más conocidos en algunos sectores de la industria, y el tipo de programación que incorporan recibe el nombre de "gestual".

6.1.3.- Robots con control por computador

Son manipuladores o sistemas mecánicos multifuncionales, controlados por un computador, que habitualmente suele ser un microordenador. El control por computador dispone de un lenguaje específico de programación, compuesto por varias instrucciones adaptadas al hardware del robot, con las que se puede diseñar un programa de aplicación utilizando solo el ordenador. A esta programación se le denomina “textual” y se crea sin la intervención del manipulador.

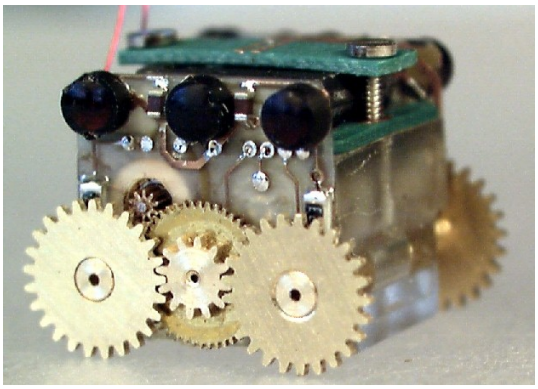
Las grandes ventajas que ofrece este tipo de robots, hacen que se vayan imponiendo en el mercado rápidamente, lo que exige la preparación urgente de personal cualificado, capaz de desarrollar programas de control que permitan el manejo del robot.

6.2.- Robots Inteligentes

Son similares a los del grupo anterior, pero tienen la capacidad de poder relacionarse con el mundo que les rodea a través de sensores y de tomar decisiones en función de la información obtenida en tiempo real. De momento, son muy poco conocidos en el mercado y se encuentran en fase experimental, donde grupos de investigadores se esfuerzan por hacerlos más efectivos, al mismo tiempo que más económicamente asequibles. El reconocimiento de imágenes y algunas técnicas de inteligencia artificial son los campos que más se están estudiando para su posible aplicación en estos robots.

6.3.- Micro-robots

Con fines educacionales, de entretenimiento o investigación, existen numerosos robots de formación o micro-robots a un precio muy asequible, cuya estructura y funcionamiento son similares a los de aplicación industrial.



Estos robots que un día se hicieron un hueco en universidades y centros de investigación, puesto que eran una forma económica de experimentar con múltiples tareas robóticas, hoy en día se pueden encontrar en centros docentes de todo tipo, incluidas escuelas de primaria e institutos. El personal de dichos centros ha apostado por este tipo de robots para estimular el interés de sus alumnos por la ciencia y la tecnología y los resultados son como se ha podido observar altamente satisfactorios.

7.- Robótica Móvil

En el apartado anterior se ha realizado un desglose de los diferentes tipos de robots existentes atendiendo a su aplicación, pero más allá de este aspecto práctico hay otro hecho característico de los robots modernos que les confiere un mayor grado de libertad y utilidad. Esta característica es el movimiento en el espacio físico, es decir, la posibilidad de desplazarse por el entorno para observarlo e interactuar con él, y de esta forma emular con mayor fidelidad las funciones y capacidades de los seres vivos.

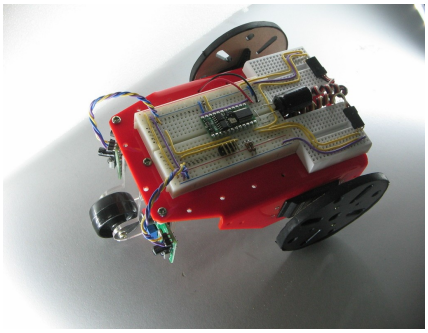
7.1.- Clasificación de los robots móviles

De la misma forma que se ha descrito en el apartado anterior en base a diferentes criterios se puede establecer una taxonomía dentro del colectivo de los robots móviles. Si por ejemplo se atiende a sus características estructurales y funcionales se puede establecer la siguiente clasificación:

7.1.1.- Robots rodantes

Son aquellos que, como su nombre indica, se desplazan haciendo uso de ruedas, generalmente montadas por pares en una configuración 2+2 como las de un vehículo por mera simplicidad. Habitualmente solo dos de sus ruedas presentan tracción y otras dos dirección, de forma que sea posible maniobrar el robot con un solo servomotor.

También es frecuente encontrar distribuciones de ruedas montadas en modo triciclo, donde una rueda sirve para la dirección y las otras dos aportan la tracción. Otra opción es que la tercera rueda simplemente sea una rueda 'loca' y las otras dos aporten tanto la tracción como la dirección, mediante el método de las orugas tratado más adelante.



Existen algunos casos especiales en los que se usan otras configuraciones que dotan al robot de mejor adaptación a terrenos difíciles. En estos casos los algoritmos de control de movimiento adquieren una mayor complejidad, proporcional al número de elementos direccionables de forma independiente.

Por último cabría considerar a los robots con orugas como un tipo de robot rodante en el que se substituyen las ruedas por un mecanismo de oruga para la tracción. La dirección se consigue parando una de las orugas o haciéndolas girar en sentido contrario.

7.1.2.- Robots andantes

Respecto a los robots construidos a imagen y semejanza humana, con dos piernas, las técnicas de control necesarias son varias, pero todas ellas hacen uso de complejos algoritmos para poder mantener el equilibrio y caminar correctamente. Todos ellos son capaces de caminar bien sobre suelos planos y subir escaleras en algunos casos, pero no están preparados para caminar en suelos irregulares.

Algunos incluso pueden realizar tareas como bailar, luchar o practicar deportes, pero esto requiere una programación sumamente compleja que no siempre está a la altura del hardware del robot y de su capacidad de procesamiento.

7.1.3.- Robots reptadores

Una clase curiosa de robots, creados basándose en animales como las serpientes, su forma de desplazarse es también una imitación de la usada por estos animales. Están formados por un número elevado de secciones que pueden cambiar de tamaño o posición de forma independiente de las demás pero coordinada, de forma que en conjunto provoquen el desplazamiento del robot.

7.1.4.- Robots nadadores

Estos robots son capaces de desenvolverse en el medio acuático, generalmente enfocados a tareas de exploración submarina en zonas donde no es posible llegar pos ser de difícil acceso o estar a profundidades que el cuerpo humano no tolera.

Aparte de lo puramente anecdótico, se ha demostrado que la estructura corporal de los peces así como el movimiento que realizan durante su desplazamiento en el agua, es uno de los métodos más óptimos de movimiento submarino dado que aprovecha la energía de forma muy eficiente y permite mayor control en la navegación, produciendo mucho menos ruido y turbulencias.

Es por todo esto que se está tendiendo a estudiar y emular en lo posible el comportamiento de estos animales a la hora de crear nuevos robots subacuáticos.

7.1.5.- Robots voladores

Conquistados los dominios del mar y la tierra solo queda una meta por alcanzar en el mundo de la robótica, ser capaces de poner robots en el cielo. Para ello y por el momento existen dos aproximaciones, en función de su principio de vuelo y estructura:

- **Helicópteros**: habitualmente helicópteros RC convencionales a los que se les añade la electrónica necesaria para tener visión artificial y capacidad de toma de decisiones autónoma.

- **Drones o aviones no tripulados**: actualmente en uso por el ejército de EEUU para tareas de logística en operaciones militares, apoyo cartográfico así como tareas de espionaje. Aunque no es habitual pueden estar dotados tanto de contramedidas para repeler agresiones como de armamento para realizar ataques.

8.- Programas en el Controlador de robot

1.- Nivel de inteligencia artificial, donde el programa aceptará un comando como "levantar el producto" y descomponerlo dentro de una secuencia de comandos de bajo nivel basados en un modelo estratégico de las tareas.

2.- Nivel de modo de control, donde los movimientos del sistema son modelados, para lo que se incluye la interacción dinámica entre los diferentes mecanismos, trayectorias planeadas, y los puntos de asignación seleccionados.

3.- Niveles de servosistemas, donde los actuadores controlan los parámetros de los mecanismos con el uso de una retroalimentacion interna de los datos obtenidos por los sensores, y la ruta es modificada sobre la base de los datos que se obtienen de sensores externos. Todas las detecciones de fallas y mecanismos de corrección son implementadas en este nivel.

En la clasificación final se considerara el nivel del lenguaje de programación La clave para una aplicación efectiva de los robots para una amplia variedad de tareas, es el desarrollo de lenguajes de alto nivel. Existen muchos sistemas de programación de robots, aunque la mayoría del software más avanzado se encuentra en los laboratorios de investigación. Los sistemas de programación de robots caen dentro de tres clases :

1.- Sistemas guiados, en el cual el usuario conduce el robot a través de los movimientos a ser realizados.

2.- Sistemas de programación de nivel-robot, en los cuales el usuario escribe un programa de computadora al especificar el movimiento y el sensado.

3.- Sistemas de programación de nivel-tarea, en el cual el usuario especifica la operación por sus acciones sobre los objetos que el robot manipula.

9.- Componentes

9.1.- Esctructura

La estructura es el esqueleto, el soporte fundamental que constituye tanto la forma como la funcionalidad del robot. Sirve de sujeción para toda la electrónica, sensores, actuadores y también es parte del aparato motriz como prolongación de los actuadores. Está formada generalmente por una mezcla de partes rígidas y flexibles, fijas y móviles y entre sus materiales destacan los plásticos, metales y aleaciones resistentes a la par que ligeras como la fibra de carbono o derivados del aluminio.

Determina el medio en el que se va a poder desenvolver el robot, así como las actividades que será capaz de realizar. También protege partes sensibles de la electrónica de golpes, polvo, agua y otros agentes externos. Como ya se ha comentado, debe ser un compromiso entre resistencia y ligereza, adaptándose de la mejor manera posible al tipo de tareas para las que se diseña el robot que va a poseer dicha estructura.

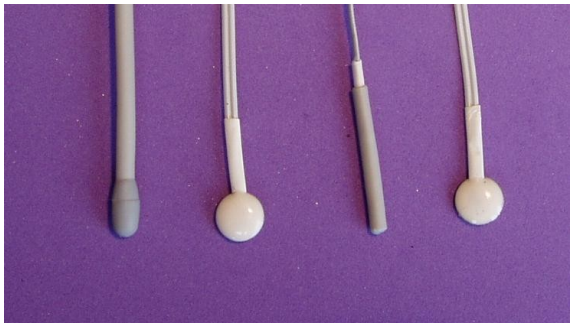
9.2.- Sensores

Estos elementos son los encargados de adquirir información del entorno y transmitirla a la unidad de control del robot. Una vez esta es analizada, el robot realiza la acción correspondiente a través de sus actuadores.

Los sensores constituyen el sistema de percepción del robot, es decir, facilitan la información del mundo real para que el robot la interprete. Los tipos de sensores más utilizados son los siguientes:

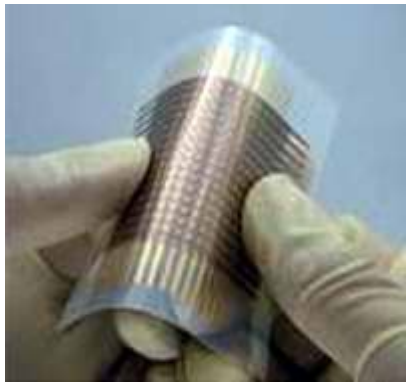
- **Sensor de proximidad**: Detecta la presencia de un objeto ya sea por rayos infrarrojos, por sonar, magnéticamente o de otro modo.

- **Sensor de Temperatura**: Capta la temperatura del ambiente, de un objeto o de un punto determinado.



- **Sensor magnéticos**: Captan variaciones producidas en campos magnéticos externos. Se utilizan a modo de brújulas para orientación geográfica de los robots.

- **Sensores táctiles, piel robótica**: Sirven para detectar la forma y el tamaño de los objetos que el robot manipula. La piel robótica se trata de un conjunto de sensores de presión montados sobre una superficie flexible.



- **Sensores de iluminación:** Capta la intensidad luminosa, el color de los objetos, etc. Es muy útil para la identificación de objetos. Es parte de la visión artificial y en numerosas ocasiones son cámaras.

- **Sensores de velocidad, de vibración (Acelerómetro) y de inclinación:** Se emplean para determinar la velocidad de actuación de las distintas partes móviles del propio robot o cuando se produce una vibración. También se detecta la inclinación a la que se encuentra el robot o una parte de él.

- **Sensores de fuerza:** Permiten controlar la presión que ejerce la mano del robot al coger un objeto.

- **Sensores de sonido:** Micrófonos que permiten captar sonidos del entorno.

- **Micro interruptores:** Muy utilizados para detectar finales de carrera.

9.3.- Actuadores

Los actuadores son los sistemas de accionamiento que permiten el movimiento de las articulaciones del robot. Se clasifican en tres grupos, dependiendo del tipo de energía que utilicen:

- **Hidráulicos:** se utilizan para manejar cargas pesadas a una gran velocidad. Sus movimientos pueden ser suaves y rápidos.

- **Neumáticos:** son rápidos en sus respuestas, pero no soportan cargas tan pesadas como los hidráulicos.

- **Eléctricos**: son los más comunes en los robots móviles. Un ejemplo son los motores eléctricos, que permiten conseguir velocidades y precisión necesarias.

- Ejemplos de actuadores son motores, relés y contadores, electro válvulas, pinzas, etc.

10.- Wifi

WLAN (Wireless Local Area Network, en inglés) es un sistema de comunicación de datos inalámbrico flexible, muy utilizado como alternativa a las redes LAN cableadas o como extensión de éstas. Utiliza tecnología de radiofrecuencia que permite mayor movilidad a los usuarios al minimizar las conexiones cableadas. Las WLAN van adquiriendo importancia en muchos campos, como almacenes o para manufactura, en los que se transmite la información en tiempo real a una terminal central. También son muy populares en los hogares para compartir el acceso a Internet entre varias computadoras.

Características

- Movilidad: permite transmitir información en tiempo real en cualquier lugar de la organización o empresa a cualquier usuario. Esto supone mayor productividad y posibilidades de servicio.

- Facilidad de instalación: al no usar cables, se evitan obras para tirar cable por muros y techos, mejorando así el aspecto y la habitabilidad de los locales, y

reduciendo el tiempo de instalación. También permite el acceso instantáneo a usuarios temporales de la red.

- Flexibilidad: puede llegar donde el cable no puede, superando mayor número de obstáculos, llegando a atravesar paredes. Así, es útil en zonas donde el cableado no es posible o es muy costoso: parques naturales, reservas o zonas escarpadas.

10.1.- Inicios

Los pioneros en el uso de redes inalámbricas han sido los radioaficionados mediante sus emisoras, que ofrecen una velocidad de 9600 bps. Pero si hablamos propiamente de redes inalámbricas debemos remontarnos al año 1997, en el que el organismo regulador IEEE (Institute of Electronics and Electrical Engineer) publicó el estándar 802.11 (802 hace referencia al grupo de documentos que describen las características de las LAN) dedicado a redes LAN inalámbricas.

Dentro de este mismo campo y anteriormente, en el año 1995, tenemos la aparición de Bluetooth, una tecnología de Ericsson con el objetivo de conectar mediante ondas de radio los teléfonos móviles con diversos accesorios. Al poco tiempo se generó un grupo de estudio formado por fabricantes que estaban interesados en esta tecnología para aplicarla a otros dispositivos, como PDAs, terminales móviles o incluso electrodomésticos.

Pero el verdadero desarrollo de este tipo de redes surgió a partir de que la FCC, el organismo americano encargado de regular las emisiones radioeléctricas, aprobó el uso civil de la tecnología de transmisiones de espectro disperso (SS o spread spectrum, en inglés), pese a que en un principio lo prohibió por el uso ampliado del espectro. Dicha tecnología ya se usaba en ámbitos militares desde la Segunda Guerra Mundial debido a sus extraordinarias características en cuanto a la dificultad de su detección y su tolerancia a interferencias externas.

A pesar, de que como hemos visto, esta tecnología ya tiene una antigüedad de más de diez años, no ha sido hasta ahora cuando este tipo de redes se ha desarrollado eficazmente debido a la disminución de precios de los dispositivos que la integran. En la actualidad cada vez más se encuentran equipos que pueden competir en precios con los modelos para redes cableadas.

10.2.- Cómo trabajan las redes WLAN

Se utilizan ondas de radio para llevar la información de un punto a otro sin necesidad de un medio físico guiado. Al hablar de ondas de radio nos referimos normalmente a portadoras de radio, sobre las que va la información, ya que realizan la función de llevar la energía a un receptor remoto. Los datos a transmitir se superponen a la portadora de radio y de este modo pueden ser extraídos exactamente en el receptor final.

A este proceso se le llama modulación de la portadora por la información que está siendo transmitida. Si las ondas son transmitidas a distintas frecuencias de radio, varias portadoras pueden existir en igual tiempo y espacio sin interferir entre ellas. Para extraer los datos el receptor se sitúa en una determinada frecuencia, frecuencia portadora, ignorando el resto. En una configuración típica de LAN sin cable los puntos de acceso (transceiver) conectan la red cableada de un lugar fijo mediante cableado normalizado.

El punto de acceso recibe la información, la almacena y la transmite entre la WLAN y la LAN cableada. Un único punto de acceso puede soportar un pequeño grupo de usuarios y puede funcionar en un rango de al menos treinta metros y hasta varios cientos. El punto de acceso (o la antena conectada al punto de acceso) es normalmente colocado en alto pero podría colocarse en cualquier lugar en que se obtenga la cobertura de radio deseada. El usuario final accede a la red WLAN a través de adaptadores. Estos proporcionan una interfaz entre el sistema de operación de red del cliente y las ondas, mediante una antena.

La naturaleza de la conexión sin cable es transparente a la capa del cliente.

10.3.- Configuraciones de red para radiofrecuencia

Pueden ser de muy diversos tipos y tan simples o complejas como sea necesario. La más básica se da entre dos ordenadores equipados con tarjetas adaptadoras para WLAN, de modo que pueden poner en funcionamiento una red independiente siempre que estén dentro del área que cubre cada uno. Esto es llamado red de igual a igual (peer to peer). Cada cliente tendría únicamente acceso a los recursos del otro cliente pero no a un servidor central. Este tipo de redes no requiere administración o preconfiguración.

Instalando un Punto de Acceso se puede doblar la distancia a la cual los dispositivos pueden comunicarse, ya que estos actúan como repetidores. Desde que el punto de acceso se conecta a la red cableada cualquier cliente tiene acceso a los recursos del servidor y además gestionan el tráfico de la red entre los terminales más próximos. Cada punto de acceso puede servir a varias máquinas, según el tipo y el número de transmisiones que tienen lugar. Existen muchas aplicaciones en el mundo real con un rango de 15 a 50 dispositivos cliente con un solo punto de acceso.

Los puntos de acceso tienen un alcance finito, del orden de 150 m en lugares u zonas abiertas. En zonas grandes como por ejemplo un campus universitario o un edificio es probablemente necesario más de un punto de acceso. La meta es cubrir el área con células que solapen sus áreas de modo que los clientes puedan moverse sin cortes entre un grupo de puntos de acceso.

Esto es llamado roaming, mediante el cual el diseñador de la red puede elegir usar un Punto de Extensión (EPs) para aumentar el número de puntos de acceso a la red, de modo que funcionan como tales pero no están enganchados a la red cableada

como los puntos de acceso.

Los puntos de extensión funcionan como su nombre indica: extienden el alcance de la red retransmitiendo las señales de un cliente a un punto de acceso o a otro punto de extensión. Los puntos de extensión pueden encadenarse para pasar mensajes entre un punto de acceso y clientes lejanos de modo que se construye un puente entre ambos.

Uno de los últimos componentes a considerar en el equipo de una WLAN es la antena direccional. Por ejemplo: si se quiere una LAN sin cable a otro edificio a 1 km de distancia. Una solución puede ser instalar una antena en cada edificio con línea de visión directa. La antena del primer edificio está conectada a la red cableada mediante un punto de acceso. Igualmente en el segundo edificio se conecta un punto de acceso, lo cual permite una conexión sin cable en esta aplicación.

10.4.- Asignación de Canales

Los estándares 802.11b y 802.11g utilizan la banda de 2.4 – 2.5 Ghz. En esta banda, se definieron 11 canales utilizables por equipos WIFI, los que pueden configurarse de acuerdo a necesidades particulares. Sin embargo, los 11 canales no son completamente independientes (canales contiguos se superponen y se producen interferencias) y en la práctica sólo se pueden utilizar 3 canales en forma simultánea (1, 6 y 11). Esto es correcto para USA y muchos países de América Latina, pues en Europa, el ETSI ha definido 13 canales. En este caso, por ejemplo en España, se pueden utilizar 4 canales no-adyacentes (1, 5, 9 y 13). Esta asignación de canales usualmente se hace sólo en el Access Point, pues los "clientes" automáticamente detectan el canal, salvo en los casos en que se forma una red "Ad-Hoc" o punto a punto cuando no existe Access Point.

10.5.- Seguridad

Uno de los problemas de este tipo de redes es precisamente la seguridad ya que cualquier persona con una terminal inalámbrica podría comunicarse con un punto de acceso privado si no se disponen de las medidas de seguridad adecuadas. Dichas medidas van encaminadas en dos sentidos: por una parte está el cifrado de los datos que se transmiten y en otro plano, pero igualmente importante, se considera la autenticación entre los diversos usuarios de la red. En el caso del cifrado se están realizando diversas investigaciones ya que los sistemas considerados inicialmente se han conseguido descifrar. Para la autenticación se ha tomado como base el protocolo de verificación EAP (Extesible Authentication Protocol), que es bastante flexible y permite el uso de diferentes algoritmos.

3 Robotino Festo

3.1.- Introducción y Características

En este apartado presentaremos el robot usado para el desarrollo del proyecto, describiremos sus características físicas como las de programación.

La compañía festo ofrece para el aprendizaje y programación, el robotino festo, una herramienta de trabajo con una gran versatilidad a la hora de su programación y el hardware que tiene integrado.

A continuación se ofrecen las características del robot :

- Diámetro: 370 mm
- Altura, incl. carcasa: 210 mm
- Peso total: aprox. 11 kg

Chasis con:

- Regleta protectora de goma con sensor de protección de colisiones integrado
- 9 sensores de distancia infrarrojos analógicos
- Sensor inductivo analógico
- 2 sensores ópticos digitales
- Cámara web en color con interface USB y compresión jpeg

Control:

- Embedded PC 104 plus con procesador AMB LX800 (800 MHz), sistema operativo Ubuntu Realtime Linux y numerosas interfaces de comunicación:
- Ethernet, 2 x USB, 2 x RS232, puerto paralelo y conexión VGA

- Wireless LAN Access Point de gran capacidad provisto de antena según 802.11g y 802.11b. Conmutable al modo Client. Opcionalmente con codificación WPA2.
- Placa de control EA09 con microcontrolador LPC2377 de 32 bits:
- Activación de cuatro motores de corriente continua
- FPGA (Xilinx Spartan3) para una rápida lectura de los datos de los sensores
- Interfaz Ethernet para acceso externo directo a la regulación del motor
- Dos conectores E/S de 20 contactos para la integración de otros componentes eléctricos

Acceso directo :

- Con el teclado de lámina integrado en la carcasa del sistema de control se puede controlar Robotino® incluso sin WLAN.

- Inicio del proceso de arranque para el ordenador de control

- Selección de idioma (DE, EN, ES, FR)

- Informaciones de estado

- Indicación de estado de los acumuladores estancos

- Configuración de la conexión en red

- Selección de programas Demo autónomos

- Arranque de programas autónomos específicos del usuario

3.2.- Entorno de programación

El corazón del mando PC 104 es el sistema operativo de tiempo real Linux, instalado en una tarjeta CF de 1 GB. Mediante una interfaz serie, este sistema se comunica con la nueva placa de control EA09 para evaluar los datos de los sensores y activar las unidades de accionamiento de Robotino ®. Se puede comunicar de forma directa con un programa Linux en el PC 104 o a través de W-LAN con Robotino® View u otra aplicación de PC externa:

- API con biblioteca para programar con .Net, C++, C, C# y JAVA.

- Numerosos programas de ejemplo muestran la aplicación del API.
- Interfaz MatLab y LabView.

- Actualización en línea del sistema operativo Robotino® vía W-LAN.

- Descarga de programas Robotino® View 2 en el PC 104.

- Depuración en línea vía W-LAN de programas Robotino® View 2 del PC 104 en el Windows PC.

- Comunicación de datos vía W-LAN entre la aplicación Robotino® View 2 en Windows PC y un programa Robotino® View 2 en el mando Robotino®.

3.3.- Placa de control EA09

La placa de control EA09 le permite iniciarse profesionalmente en las técnicas de regulación de motores eléctricos. Esta placa sustituye a la pletina EA estándar utilizada en la primera versión de Robotino®.

El núcleo de esta placa es un microcontrolador de 32 bits que genera directamente las señales de modulación por ancho de pulsos necesarias para activar cuatro motores eléctricos de corriente continua. Por motivos de compatibilidad, la comunicación con el PC 104 se realiza mediante la misma interfaz serie que en la primera versión de Robotino®.

Para leer los valores de codificador de los motores, se utiliza un FPGA. Eso permite calcular directamente en el microcontrolador, p. ej. los datos de odometría y, en su caso, los datos de corrección adicionales dependientes de los sensores.

Con ello se logra una considerable mejora de la precisión.

3.4.- Otras características

3.4.1.- Posibilidades de ampliación

La placa de control incluye las siguientes conexiones adicionales para poder efectuar ampliaciones posteriormente:

- 8 entradas analógicas de 0 – 10 V, 50 Hz
- 8 entradas y salidas digitales (de 24 V, a prueba de cortocircuitos y con protección contra sobrecarga)
- 2 relés para actuadores adicionales

3.4.2.- Programación del microcontrolador

El microcontrolador es accesible desde fuera y puede utilizarse directamente para programar aplicaciones propias. El firmware del microcontrolador puede actualizarse a través del sistema operativo Robotino®.

3.4.5.- Técnica de regulación

La placa de control EA09 ha sido equipada con una tarjeta de interfaz que proporciona cuatro interfaces Ethernet, una de las cuales conecta directamente con el exterior. Con el software de pantalla EA09View,
usted puede explorar y visualizar, con una frecuencia de 1 KHz, los datos de los reguladores de motor (valores nominales y reales, valores de tensión y de corriente) y así analizar en detalle la parametrización de los reguladores PID de los motores.

3.4.6.- Hardware en bucle

Si por ejemplo usted crea en MatLab su propio regulador de motor, podrá controlar y regular los motores de Robotino® con este regulador de software a través de la interfaz Ethernet, . El API de MatLab necesario para la comunicación con la placa de control EA09 está incluido.

4 Arquitectura Cliente-Servidor

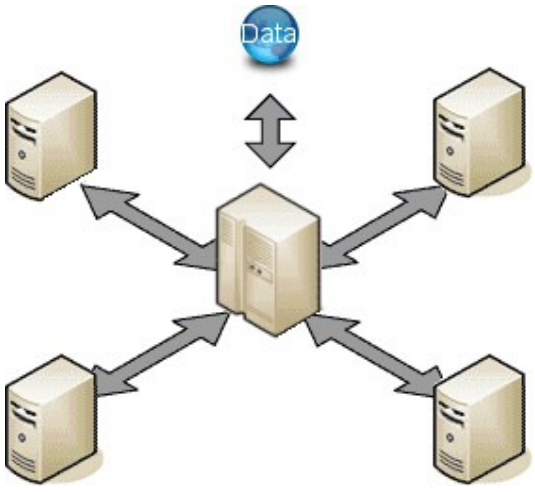
1.- Introducción

Con la proliferación de ordenadores personales de bajo coste en el mercado, los recursos de sistemas de información existentes en cualquier organización se pueden distribuir entre ordenadores de diferentes tipos: ordenadores personales de gama baja, media y alta, estaciones de trabajo, miniordenadores o incluso grandes ordenadores.

2.- Definición

El concepto de cliente/servidor proporciona una forma eficiente de utilizar todos estos recursos de máquina de tal forma que la seguridad y fiabilidad que proporcionan los entornos mainframe, se traspasa a la red de área local. A esto hay que añadir la ventaja de la potencia y simplicidad de los ordenadores personales.

La arquitectura cliente/servidor un modelo para el desarrollo de sistemas de información en el que las transacciones se dividen en procesos independientes que cooperan entre sí para intercambiar información, servicios o recursos. Se denomina cliente al proceso que inicia el diálogo o solicita los recursos y servidor al proceso que responde a las solicitudes.



Típico ejemplo de cliente-servidor

En este modelo las aplicaciones se dividen de forma que el servidor contiene la parte que debe ser compartida por varios usuarios, y en el cliente permanece sólo lo particular de cada usuario.

Los clientes realizan generalmente funciones como:

- Manejo de la interfaz de usuario.
- Captura y validación de los datos de entrada.
- Generación de consultas e informes sobre las bases de datos.
- Por su parte los servidores realizan, entre otras, las siguientes funciones:
- Gestión de periféricos compartidos.
- Control de accesos concurrentes a bases de datos compartidas.
- Enlaces de comunicaciones con otras redes de área local o extensa.

Siempre que un cliente requiere un servicio lo solicita al servidor correspondiente y éste le responde proporcionándole el servicio correspondiente. Normalmente, pero no necesariamente, el cliente y el servidor están ubicados en distintos procesadores. Los clientes se suelen situar en ordenadores personales y/o estaciones de trabajo y los servidores en procesadores departamentales o de grupo.

Entre las principales características de la arquitectura cliente/servidor se pueden destacar las siguientes:

- El servidor presenta a todos sus clientes una interfaz única y bien definida.
- El cliente no necesita conocer la lógica del servidor, sólo su interfaz externa.
- El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.
- Los cambios en el servidor implican pocos o ningún cambio en el cliente.

3.- Ventajas e inconvenientes

Ventajas

- Aumento de la productividad:

- Los usuarios pueden utilizar herramientas que le son familiares, como hojas de cálculo y herramientas de acceso a bases de datos.

- Mediante la integración de las aplicaciones cliente/servidor con las aplicaciones personales de uso habitual, los usuarios pueden construir soluciones particularizadas que se ajusten a sus necesidades cambiantes.

- Una interfaz gráfica de usuario consistente reduce el tiempo de aprendizaje de las aplicaciones.

- Menores costes de operación:

- Permiten un mejor aprovechamiento de los sistemas existentes, protegiendo la

inversión. Por ejemplo, la compartición de servidores (habitualmente caros) y dispositivos periféricos (como impresoras) entre máquinas clientes permite un mejor rendimiento del conjunto.

- Proporcionan un mejor acceso a los datos. La interfaz de usuario ofrece una forma homogénea de ver el sistema, independientemente de los cambios o actualizaciones que se produzcan en él y de la ubicación de la información.

- El movimiento de funciones desde un ordenador central hacia servidores o clientes locales origina el desplazamiento de los costes de ese proceso hacia máquinas más pequeñas y por tanto, más baratas.

- Mejora en el rendimiento de la red:

- Las arquitecturas cliente/servidor eliminan la necesidad de mover grandes bloques de información por la red hacia los ordenadores personales o estaciones de trabajo para su proceso. Los servidores controlan los datos, procesan peticiones y después transfieren sólo los datos requeridos a la máquina cliente. Entonces, la máquina cliente presenta los datos al usuario mediante interfaces amigables. Todo esto reduce el tráfico de la red, lo que facilita que pueda soportar un mayor número de usuarios.

- Tanto el cliente como el servidor pueden escalarse para ajustarse a las necesidades de las aplicaciones. Las UCPs utilizadas en los respectivos equipos pueden dimensionarse a partir de las aplicaciones y el tiempo de respuesta que se requiera.

- La existencia de varias UCPs proporciona una red más fiable: un fallo en uno de los equipos no significa necesariamente que el sistema deje de funcionar.

- En una arquitectura como ésta, los clientes y los servidores son independientes los unos de los otros con lo que pueden renovarse para aumentar sus funciones y capacidad de forma independiente, sin afectar al resto del sistema.

- La arquitectura modular de los sistemas cliente/servidor permite el uso de ordenadores especializados (servidores de base de datos, servidores de ficheros, estaciones de trabajo para CAD, etc.).

- Permite centralizar el control de sistemas que estaban descentralizados, como por ejemplo la gestión de los ordenadores personales que antes estuvieran aislados.

Inconvenientes

- Hay una alta complejidad tecnológica al tener que integrar una gran variedad de productos.

- Requiere un fuerte re-diseño de todos los elementos involucrados en los sistemas de información (modelos de datos, procesos, interfaces, comunicaciones, almacenamiento de datos, etc.). Además, en la actualidad existen pocas herramientas que ayuden a determinar la mejor forma de dividir las aplicaciones entre la parte cliente y la parte servidor.

- Es más difícil asegurar un elevado grado de seguridad en una red de clientes y servidores que en un sistema con un único ordenador centralizado.

- A veces, los problemas de congestión de la red pueden degradar el rendimiento del sistema por debajo de lo que se obtendría con una única máquina (arquitectura centralizada). También la interfaz gráfica de usuario puede a veces ralentizar el funcionamiento de la aplicación.

- El quinto nivel de esta arquitectura (bases de datos distribuidas) es técnicamente muy complejo y en la actualidad hay muy pocas implantaciones que garanticen un funcionamiento totalmente eficiente.

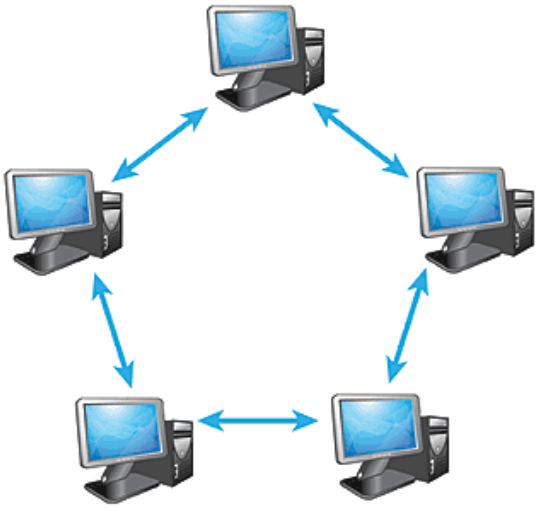
- Existen multitud de costes ocultos (formación en nuevas tecnologías, licencias, cambios organizativos, etc.) que encarecen su implantación.

5 Otras Arquitecturas

1.- Arquitectura Peer-to-Peer

1.1.- Definición

Una red peer-to-peer o red de pares o red entre iguales o red entre pares o red punto a punto (P2P, por sus siglas en inglés) es una red de computadoras en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Es decir, actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red. Las redes P2P permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados.



Típico Ejemplo de red P2P

El hecho de que sirvan para compartir e intercambiar información de forma directa entre dos o más usuarios ha propiciado que se utilicen para intercambiar archivos cuyo contenido está sujeto a las leyes de copyright, lo que ha generado una gran polémica entre defensores y detractores de estos sistemas.

Las redes peer-to-peer aprovechan, administran y optimizan el uso del ancho de banda de los demás usuarios de la red por medio de la conectividad entre los mismos, y obtienen así más rendimiento en las conexiones y transferencias que con algunos métodos centralizados convencionales, donde una cantidad relativamente pequeña de servidores provee el total del ancho de banda y recursos compartidos para un servicio o aplicación.

Dichas redes son útiles para diversos propósitos. A menudo se usan para compartir ficheros de cualquier tipo (por ejemplo, audio, vídeo o software). Este tipo de red también suele usarse en telefonía VoIP para hacer más eficiente la transmisión de datos en tiempo real.

La eficacia de los nodos en el enlace y transmisión de datos puede variar según su configuración local, velocidad de proceso, disponibilidad de su conexión a la red y capacidad de almacenamiento en disco.

1.2 Características deseables

- **Escalabilidad.** Las redes P2P tienen un alcance mundial con cientos de millones de usuarios potenciales. En general, lo deseable es que cuantos más nodos estén conectados a una red P2P, mejor será su funcionamiento. Así, cuando los nodos llegan y comparten sus propios recursos, los recursos totales del sistema aumentan. Esto es diferente en una arquitectura del modo servidor-cliente con un sistema fijo de servidores, en los cuales la adición de clientes podría significar una transferencia de datos más lenta para todos los usuarios. Algunos autores advierten que, si proliferan mucho este tipo de redes, cliente-servidor,

podrían llegar a su fin, ya que a cada una de estas redes se conectarán muy pocos usuarios.

- **Robustez.** La naturaleza distribuida de las redes peer-to-peer también incrementa la robustez en caso de haber fallos en la réplica excesiva de los datos hacia múltiples destinos, y —-en sistemas P2P puros—- permitiendo a los peers encontrar la información sin hacer peticiones a ningún servidor centralizado de indexado. En el último caso, no hay ningún punto singular de falla en el sistema.

- **Descentralización.** Estas redes por definición son descentralizadas y todos los nodos son iguales. No existen nodos con funciones especiales, y por tanto ningún nodo es imprescindible para el funcionamiento de la red. En realidad, algunas redes comúnmente llamadas P2P no cumplen esta característica, como Napster, Edonkey..

- **Repartición de costes entre los usuarios.** Se comparten o donan recursos a cambio de recursos. Según la aplicación de la red, los recursos pueden ser archivos, ancho de banda, ciclos de proceso o almacenamiento de disco.

- **Anonimato.** Es deseable que en estas redes quede anónimo el autor de un contenido, el editor, el lector, el servidor que lo alberga y la petición para encontrarlo, siempre que así lo necesiten los usuarios. Muchas veces el derecho al anonimato y los derechos de autor son incompatibles entre sí, y la industria propone mecanismos como el DRM para limitar ambos.

- **Seguridad.** Es una de las características deseables de las redes P2P menos implementada. Los objetivos de un P2P seguro serían identificar y evitar los nodos maliciosos, evitar el contenido infectado, evitar el espionaje de las comunicaciones entre nodos, creación de grupos seguros de nodos dentro de la red, protección de los recursos de la red... La mayor parte de los nodos aún están bajo investigación, pero los mecanismos más prometedores son: cifrado multiclave, cajas de arena, gestión de derechos de autor (la industria define qué puede hacer el usuario; por ejemplo, la segunda vez que se oye la canción se apaga), reputación (permitir acceso sólo a los conocidos), comunicaciones seguras, comentarios sobre los ficheros, etc.

1.3.- Problemática

El mayor problema para usar este tipo de arquitectura es que normalmente se usa para la transmisión y recepción de archivos. Ya que no se necesita de un servidor fijo para su utilización si no que este puede estar repartido junto a los clientes.

En nuestro caso es necesario un servidor fijo, y que se pueda adaptar a los cambios de Ip, ya que esta puede ser dinámica y los clientes deben ser capaces de

localizarlo por su nombre.

2.- Arquitectura Cliente-Cola-Cliente

2.1.- Definición

Si bien la clásica arquitectura C/S requiere uno de los puntos terminales de comunicación para actuar como un servidor, que puede ser algo más difícil de aplicar, la arquitectura Cliente-Cola-Cliente habilita a todos los nodos para actuar como clientes simples, mientras que el servidor actúa como una cola que va capturando las peticiones de los clientes (un proceso que debe pasar sus peticiones a otro, lo hace a través de una cola, por ejemplo, una consulta a una base de datos, entonces, el segundo proceso conecta con la base de datos, elabora la petición, la pasa a la base de datos, etc.). Esta arquitectura permite simplificar en gran medida la implementación de software. La arquitectura P2P originalmente se basó en el concepto "Cliente-Cola-Cliente".

2.1- Problemática

El mayor problema si usáramos esta arquitectura es el uso de la cola ya que el servidor debería gestionar de alguna manera los clientes que acceden a el, usar prioridades seria una solución, añadiendo carga al servidor ya que debería de estar pendiente tanto de la entrada de clientes, la gestión de los que ya están dentro de él y los problemas que podría causar si la gestión de prioridades no es la adecuada.

6 NETBEANS

1.- Introducción

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio de 2000 y continúa siendo el patrocinador principal de los proyectos.

2.- Netbeans ID

El IDE NetBeans es un IDE - una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java - pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE NetBeans. El IDE NetBeans es un producto libre y gratuito sin restricciones de uso.

El NetBeans IDE es un IDE de código abierto escrito completamente en Java usando la plataforma NetBeans. El NetBeans IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactoring.

NetBeans IDE 6.5, la cual fue lanzada el 19 de noviembre de 2008, extiende las características existentes del Java EE (incluyendo Soporte a Persistencia, EJB 3 y JAX-WS). Adicionalmente, el NetBeans Enterprise Pack soporta el desarrollo de Aplicaciones empresariales con Java EE 5, incluyendo herramientas de desarrollo visuales de SOA, herramientas de esquemas XML, orientación a web servicies (for BPEL), y modelado UML. El NetBeans C/C++ Pack soporta proyectos de C/C++, mientras el PHP Pack, soporta PHP 5.

Modularidad. Todas las funciones del IDE son provistas por módulos. Cada módulo provee una función bien definida, tales como el soporte de Java, edición, o soporte para el sistema de control de versiones. NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiéndole al usuario comenzar a trabajar inmediatamente.

DESARROLLO PRACTICO

PROBLEMÁTICA A RESOLVER

Vamos a desarrollar el problema que planteaba el proyecto y como se desarrollo hasta llegar a la solución que mas se adaptaba al problema que nos enfrentábamos.

El mayor problema que se nos presentaba es que el DLL del robot servia para poder realizar la siguiente funcion : 1 Pc -----> +1 Robot. Bien como planteamos al inicio del proyecto nosotros queremos tener la relación a la inversa : +1 Pc ----> 1 Robot.

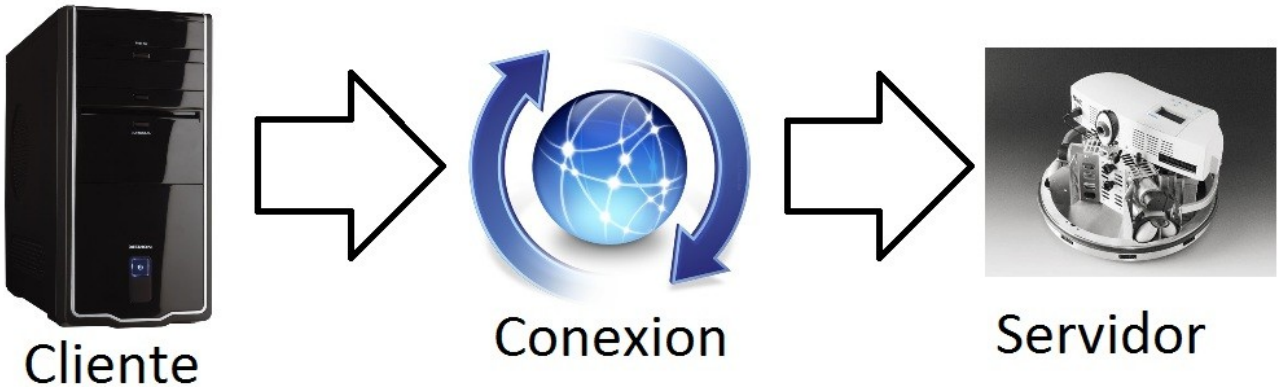
Selección del lenguaje de programación.

Una vez planteado el problema principal pasaremos al siguiente, que lenguaje elegir para realizar la aplicación cliente-servidor. De entre los muchos lenguajes posibles : C, C++, C#, Phyton, Visual Basic, Java. De todos los lenguajes mencionados seleccionamos Java por las siguientes razones, entre muchas otras :

- Uso extendido.
- Compatibilidad con todos los sistemas operativos.
- Facilidad de entendimiento.

Una vez seleccionado el lenguaje, la manera mas fácil de abordar el programa era usar el elemento de interconexión en Java : Socket. Al mismo tiempo haremos uso de las liberias ya definidas en el robotino para poder gestionar la conexión interna y las instrucciones para que el robot las ejecute.

Por lo tanto a grandes rasgos el problema queda resuelto de la siguiente manea :



El esquema anterior indica como funciona la aplicación realizada.

El cliente conecta al robot mediante una conexión wifi y el programa cliente, el robot recibe la conexión y el servidor rechazara o no la conexión (si se ha llegado al máximo de clientes permitidos.

Todo esto se realiza mediante Hilos de ejecución lo cual nos permite gestionar los clientes de manera individual, haciendo mucho mas fácil saber que hilo esta ejecutando que acción en cada momento.

Una vez la orden es indicada al robot, este haciendo uso de la librería "rec.robotino" ejecutara las instrucciones correspondientes para realizar la acción necesaria.

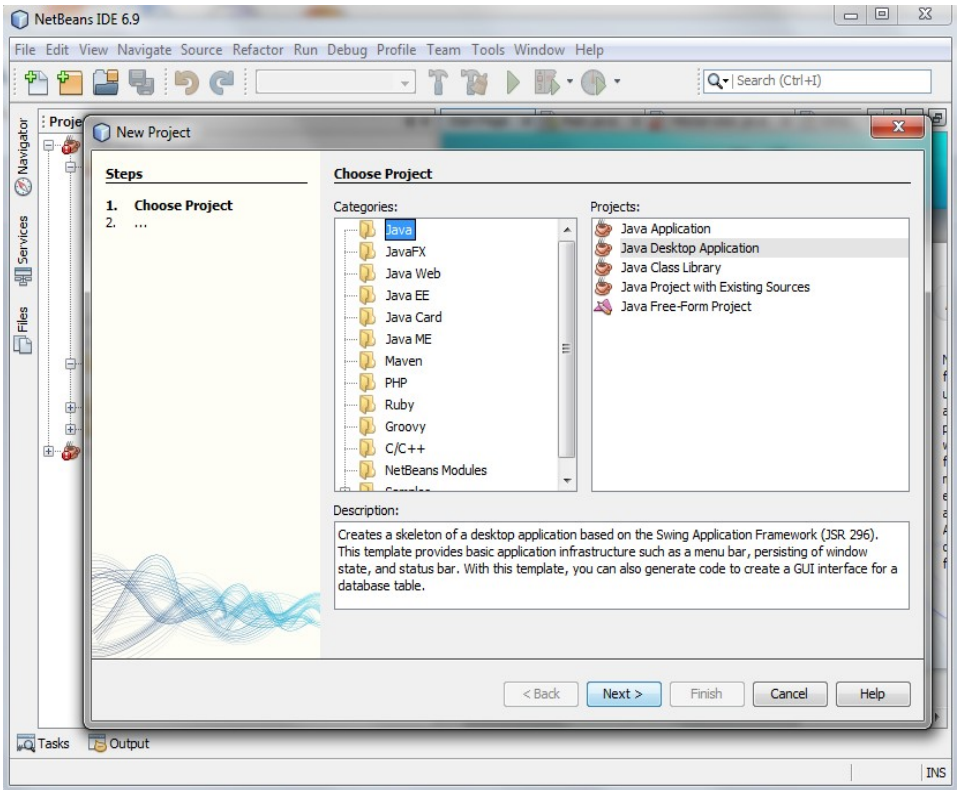
A grandes rasgos estos han sido los pasos necesarios para resolver el problema que se planteo al inicio de este proyecto.

CREACION DEL CLIENTE

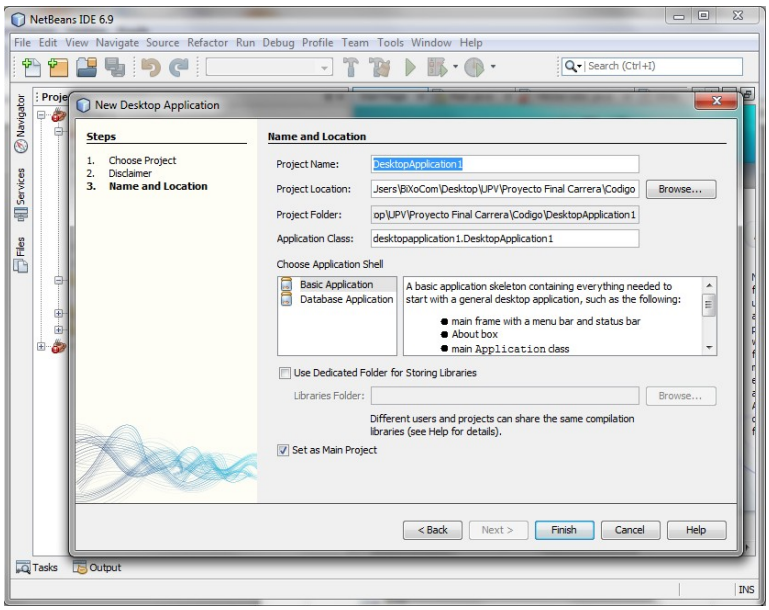
En este apartado se explicara mediante una ayuda visual como se creo el proyecto cliente, la gran ventaja de usar el Netbeans, es que dentro de las opciones a la hora de crear un proyecto, te ofrece alternativas ya creadas las cuales solo debes modificar para adaptarlas a tus necesidades.

Bien indicaremos paso a paso como crear el proyecto en netbeans.

1. Ejecutamos NetBeans.
2. En el menú File , seleccionamos New Proyect.
3. Se abrirá una ventana en la cual se indicaran los diferentes proyecto y los lenguajes que pueden ser programados en java.
4. Bien dentro de la Opción : Java, seleccionaremos : Java Deskpot Application.

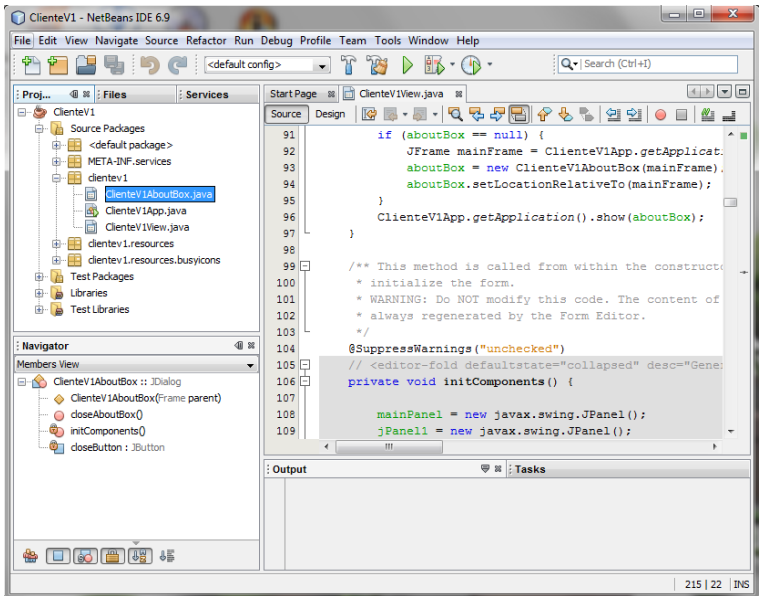


5. Una vez indicado nos pedirá un nombre de proyecto y la carpeta donde se almacenara.



Una vez hecho lo anterior se nos presentara en el IDE principal, en la ventana proyectos el proyecto que acabamos de crear. Para acceder a los elementos modificables deberemos seguir los siguientes pasos.

1. Dentro de la ventana proyecto desplegaremos el proyecto que hemos creado.
2. Una vez desplegado accederemos a la carpeta : Source Packages que contiene las clases para poder ser modificadas, añadir nuevas clase, elementos visuales, etc...
3. Una vez allí seleccionaremos aquel apartado que tenga el mismo nombre que el dado al proyecto y accederemos a las clases principales y creadas por NETBEANS.



1 Cliente

1.- Introducción

En este apartado vamos a tratar la parte correspondiente al cliente, presentando tanto los componentes visuales, interfaz del programa, como los componentes internos del mismo, programación realizada.

La función principal de esta aplicación es la de ofrecer una interfaz, entendible e intuitiva que ofrezca a la persona que vaya a usarla las características que para ellos ha sido programada.

2.- Pantalla Inicial

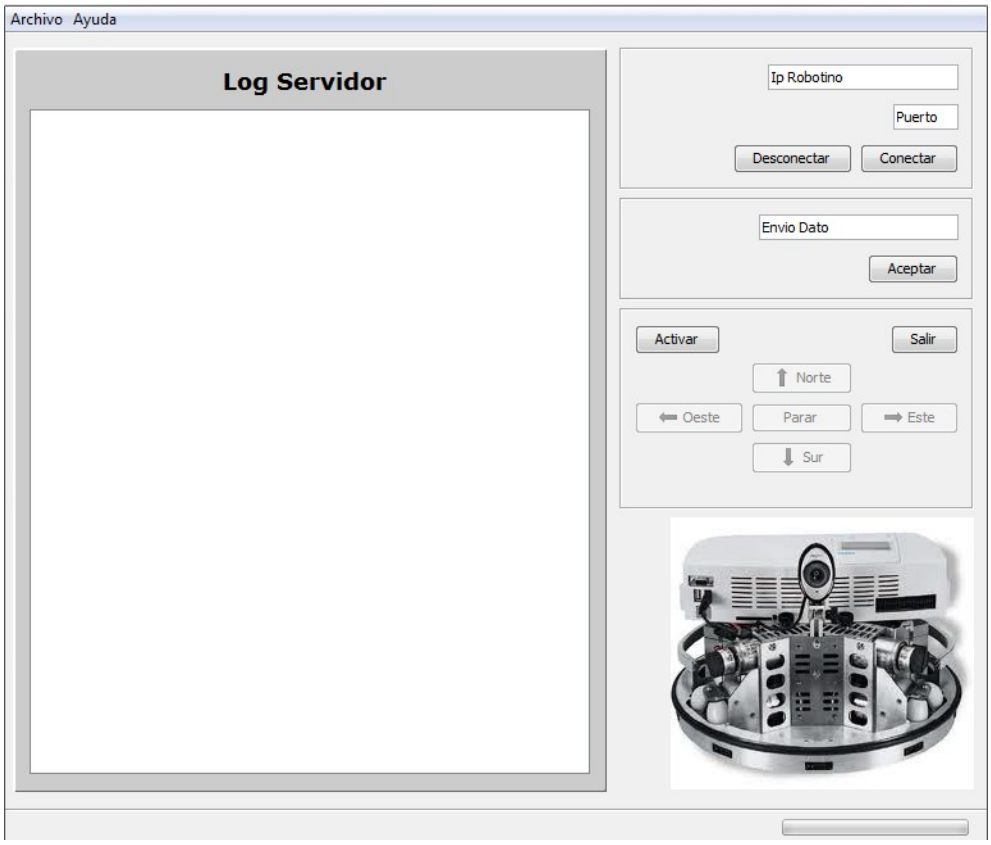


Figura 1. Vista general del programa servidor

En esta imagen se divisan los diferentes componentes de los cuales, pasaremos a explicar su comportamiento y el código que lo componen.

2.- Inicio conexión

La siguiente figura muestra la parte de inicio de conexión entre el cliente y el servidor.

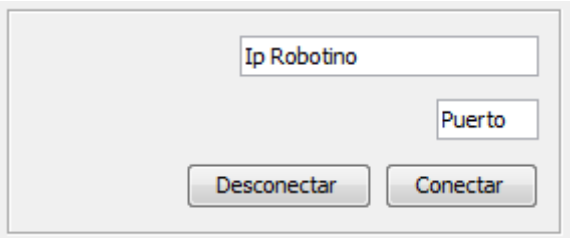


Figura 2. Conexión con el servidor.

Para realizar la conexión el cliente realizara los siguientes pasos :

1. Introducirá la dirección o nombre del robot.
2. Introducirá el puerto de conexión con el robot.
3. Presionara botón conexión.

En caso de error se mostrar una ventana de error indicando la causa del mismo.

Si todo sale correctamente se mostrara una ventana indicando el éxito en la conexión y apareciendo en el log del servidor, la información y opciones que nos ofrecerá el servidor.

A continuación se muestra el código usado para la conexión del robot :

```
ActionListener al = new ActionListener() {
   public void actionPerformed(ActionEvent ae) {

      try
      {
         s_cli = new Socket(ipRobot.getText(), Integer.parseInt(puertoRobot.getText()));
         JOptionPane.showMessageDialog(null, "Conexion Establecida");
         auxLee = new leesocket();
         new java.lang.Thread(auxLee).start();
         ipRobot.setEnabled(false);
         puertoRobot.setEnabled(false);
         Conectar.setEnabled(false);



      }
      catch(Exception ex)
      {
         JOptionPane.showMessageDialog(null, ex.getMessage());
      }
   }
};
Conectar.addActionListener(al);
```

El código mostrando realiza las siguiente acciones :

- Intenta crear un socket cliente con el servidor.
- Si la conexión se realiza se muestra una ventana donde se indica el inicio de la misma.

- En caso negativo se capturara la causa del error y se mostrara con otra ventana.
- Si todo ha salido correctamente se inicia un Hilo, el cual se encargara de recibir los mensajes del robot.
  - new java.lang.Thread(auxLee).start();

3.- Fin conexión

Partiendo de la Imagen (Figura 2). La desconexion es bastante simple, una vez en el menú inicial del robot, pulsaremos sobre el robot y se procederá a indicarle al servidor que cierre la conexión y libere esos recursos.

Código :

```
ActionListener descon = new ActionListener()
{

   public void actionPerformed(ActionEvent ae)
   {
      try
      {
         if(s_cli != null)
         {
            PrintWriter esc = new PrintWriter(s_cli.getOutputStream(), true);
            esc.println(0);
            auxLee.cerrarConexion();
            JOptionPane.showMessageDialog(null, "Conexion cerrada.");
            Conectar.setEnabled(true);
            ipRobot.setEnabled(true);
            ipRobot.setText("Ip Robot");
            puertoRobot.setEnabled(true);
            puertoRobot.setText("Puerto");

         }
         else
         {
            JOptionPane.showMessageDialog(null, "Conexion no establecida");
         }
      }
      catch(IOException exc)
      {
         JOptionPane.showMessageDialog(null, exc.getMessage());
      }
   }

};

Desconectar.addActionListener(descon);
```

Al igual que con la conexión es bastante simple, se enviá al servidor la señal de fin de conexión y todos los botones que estaba deshabilitados se habilitan para poder iniciar otra conexión.

4.- Envió de datos

La siguiente figura muestra la sección desde la cual se le indicaran al servidor, las acciones que queremos realizar en el robot, teniendo en cuenta cual es la acción seleccionada desde el menú inicial.

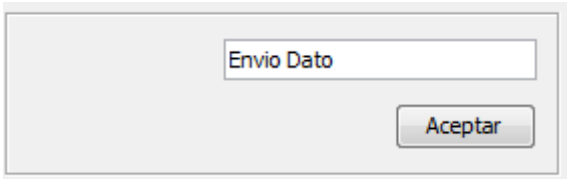


Figura 3. Envió de datos al servidor

La función de este elemento es muy simple, se introduce el dato a enviar (opción, lista de datos, etc...) y al pulsar sobre "Aceptar" estos se envían al servidor.

Código :

```
ActionListener env = new ActionListener()
{
   public void actionPerformed(ActionEvent ae)
   {
      try
      {
         PrintWriter esc = new PrintWriter(s_cli.getOutputStream(), true);
         esc.println(campoDatos.getText());
         campoDatos.setText("");
      }
      catch(Exception e)
      {
         JOptionPane.showMessageDialog(null, e.getMessage());

      }
   }
};

Enviar.addActionListener(env);
```

5.- Activar control manual

En la siguiente figura se muestra el control manual del robot, una vez indicado el inicio del mismo, el robot solo realizara las acciones que se indiquen en ellas ignorando cualquier otra orden.

Figura 4. Control manual.

En este caso al tener varios elementos gráficos, especificaremos cada parte del codigo según su función

1. Activar y desactivar la conexión.

Activar

```
ActionListener actCM = new ActionListener()
{
   public void actionPerformed(ActionEvent ae)
   {
      try
      {
                                    PrintWriter esc = new
PrintWriter(s_cli.getOutputStream(), true);
         esc.println("4");
         Norte.setEnabled(true);
         Sur.setEnabled(true);
         Oeste.setEnabled(true);
         Este.setEnabled(true);
         Parar.setEnabled(true);
         ActivarCM.setEnabled(false);
         Enviar.setEnabled(false);
         campoDatos.setEnabled(false);
      }
      catch(Exception e)
      {
          JOptionPane.showMessageDialog(null, "Error al
activar control manual.");

      }
   }
};

ActivarCM.addActionListener(actCM);
```

Desactivar

```
ActionListener salCM = new ActionListener()
{
   public void actionPerformed(ActionEvent ae)
   {
      try
      {
                                    PrintWriter esc = new
PrintWriter(s_cli.getOutputStream(), true);
         esc.println("salircm");
         Norte.setEnabled(false);
         Este.setEnabled(false);
         Oeste.setEnabled(false);
         Este.setEnabled(false);
         ActivarCM.setEnabled(true);
         SalirCM.setEnabled(false);
         Enviar.setEnabled(true);
         campoDatos.setEnabled(true);
scrollRectToVisible
      }
      catch(Exception e)
      {
                  JOptionPane.showMessageDialog(null,
e.getMessage());

      }
   }
};

SalirCM.addActionListener(salCM);
```

2. Indicar la dirección.

El siguiente código solo indica hacia donde tiene que dirigirse el robot según que botón sea pulsado.

Código (Boton de di):

```
        ActionListener sur = new ActionListener()
{
   public void actionPerformed(ActionEvent ae)
   {
      try
      {
         PrintWriter esc = new PrintWriter(s_cli.getOutputStream(), true);
         esc.println("d");
         esc.close();
      }
      catch(Exception e)
      {
         JOptionPane.showMessageDialog(null, e.getMessage());

      }
   }
};

Este.addActionListener(der);
```

El codigo anteriormente indicado, es idéntico para todos los botones de control, donde lo único que se modifica es la variables direccion, la cual tomara el valor en la cual se quiere que el robot se mueva.

6.- Log del servidor

En este apartado pasaremos a explicar el elemento que recibe todos los datos de servidor y los muestra por pantalla.

Para explicar este apartado lo dividiremos en 2 apartados, el apartado visual y el apartado de código, ya que a diferencia de los anteriores este tiene unas salvedades en su código que hay que explicar.

6.1.- Apartado Visual

La siguiente figura muestra el elemento gráfico donde se muestra lo que recibe el cliente :

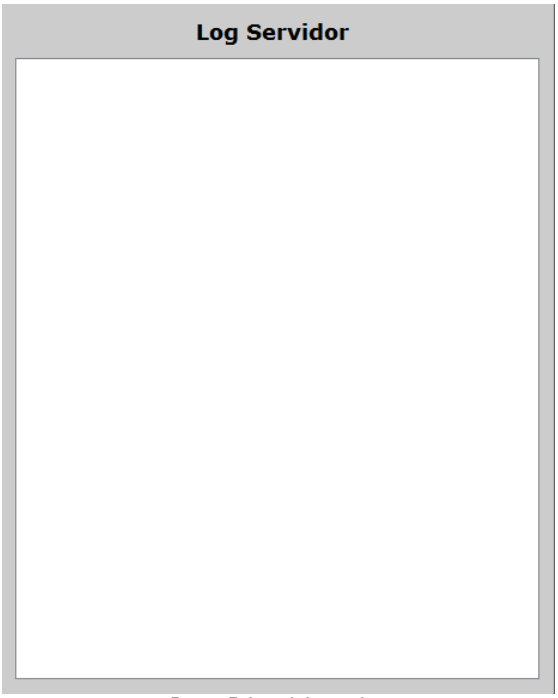


Figura 5. Log del servidor.

6.2.- Codigo del log

En este apartado, se explicara de que manera se ha implementando el codigo necesario para poder recibir los elementos del servidor.

El principal problema que he tenido para implementar la recepción de datos del servidor, es que al intentar recibir los datos en primer plano todo el programa quedaba bloqueado no pudiendo realizar ninguna acción hasta que el servidor finalizaba, mostrando todo el mensaje, pero sin poder interactuar con el.

Por lo tanto opte por la opción de usar un elemento el cual se puede ejecutar sin interferir en el programa principal.

El elemento usado son los Hilos, los cuales se pueden ir ejecutando de manera recurrente sin interrumpir el hilo principal de ejecución

A partir de lo anteriormente mencionado, describimos el codigo :

```
auxLee = new Lesotho();
new java.lang.Thread(auxLee).start();
```

El codigo anterior crea e inicial el hilo el cual leerá los datos recibidos desde el servidor al cliente.

Cabecera para la clase privada de lectura :

```
private class leeSocket implements java.lang.Runnable {
```

Variables privadas para leer los datos

```
private Scanner sca;
private boolean cerrar = false;
```

Método de creación del elemento para la lectura del Socket del servidor, se crea una variables Scanner la cual nos permitirá leer todos los datos de entrada.

```
public Lesotho() throws IOException
{
        sca = new Scanner(s_cli.getInputStream());
}
```

El siguiente método se usara para cerrar la lectura de datos y así finalizar la ejecución del hilo liberando los recursos.

```
public void cerrarConexion()
{
        try
        {
                s_cli.close();
                cerrar = true;
        }
        catch (IOException ex)
        {

        }

}
```

El método run permite el inicio de la ejecución del hilo, el cual realizara las ordenes que se encuentren en su interior, en el caso que nos ocupa, la lectura de los datos del servidor. El while interno se usa para la ejecución continua del hilo y que solo finalizara cuando la variable "cerrar" tenga el valor true.

```
public void run()
{

        while(!cerrar)
        {
                try
                {
                        jTextArea1.append(sca.nextLine());
                        jTextArea1.append(System.getProperty("line.separator"));
                        jTextArea1.setCaretPosition(jTextArea1.getDocument().getLength());
                }
                catch(NoSuchElementException ex)
                {

                }
        }

}
}
```

Con todo lo anteriormente expuesto se define la función básica del cliente :

- Establecer conexión.

- Recibir las diferentes opciones del servidor.
- A partir de las opciones recibidas interactuar con el robot.

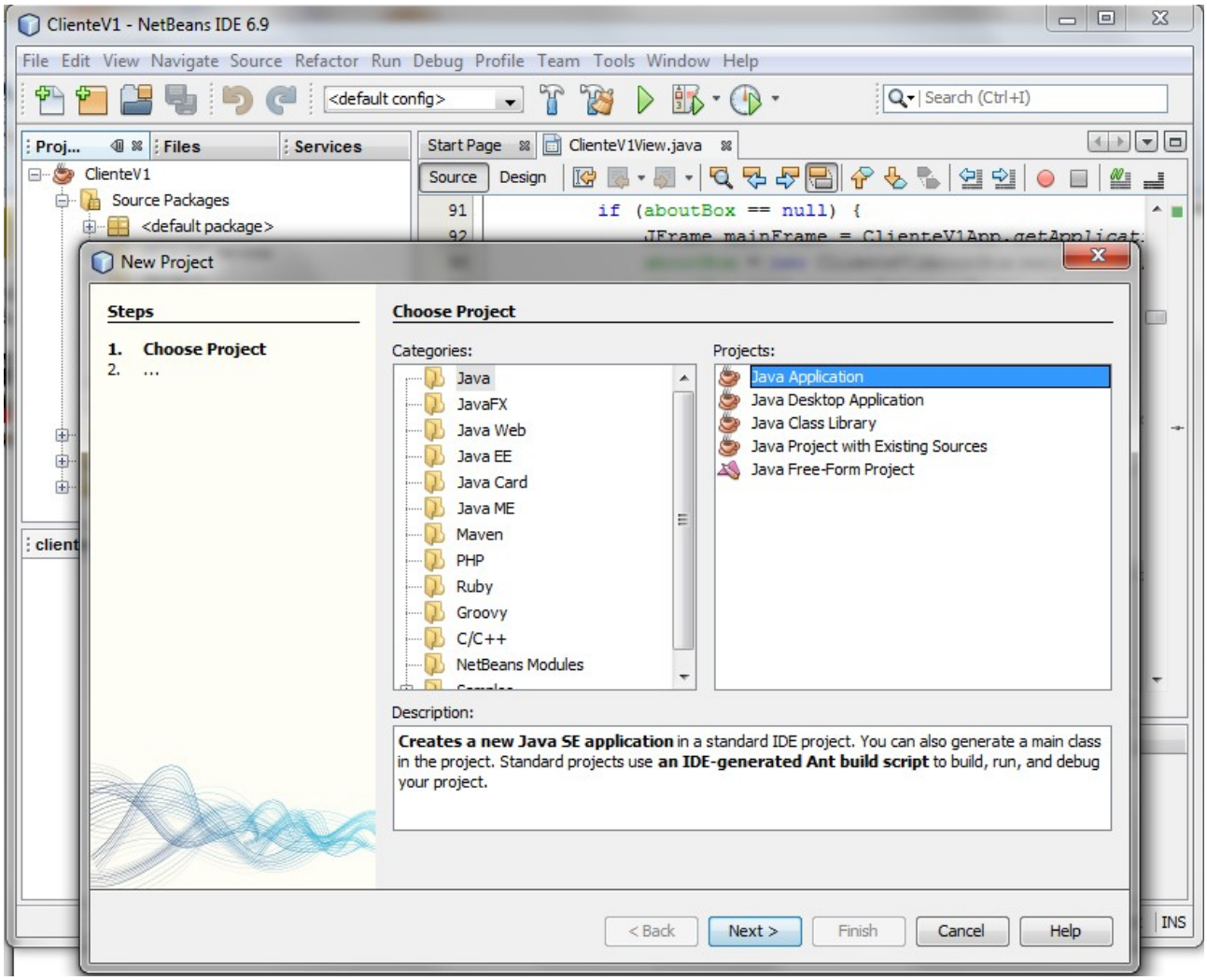
A groso modo son las características principales del cliente, donde realmente esta el grueso del proyecto es el servidor, ya que en nuestro caso es el que dirige todo el robot.

7 Servidor

Creación del Servidor

Como se explico en la sección "Creación de un Cliente", aquí indicaremos los pasos para poder crear el paquete que contendrá el servidor del robot, la diferencia mas significativa es que este no posee interfaz gráfica.

1. Ejecutamos NetBeans.
2. En el menú File , seleccionamos New Proyect.
3. Se abrirá una ventana en la cual se indicaran los diferentes proyecto y los lenguajes que pueden ser programados en java.
4. Bien dentro de la Opción : Java, seleccionaremos : Java Application.



Las siguientes acciones son idénticas a las usadas para la creación del servidor, el acceso a las clases y la modificación.

1.- Introducción

En este apartado se hablara del desarrollo practico del servidor, este se encuentra en el servidor y ofrecerá todas las posibles acciones que el cliente podrá realizar en el.

El servidor esta basado en un servidor Socket multihilo el cual permite que puedan acceder a el mas de un cliente, en el caso de nuestro servidor como máximo 2 clientes, haciendo independiente las acciones de cada uno de los clientes con el servidor. En igual medida se usa una variable global para que los clientes no puedan estar accediendo a los elementos de hardware del robot al mismo tiempo.

Una vez realizada la introducción, pasemos a ver como esta implementado dicho servidor.

2.- Inicio del servidor

Bien para explicar como inicia el servidor hay que dejar claro algunos conceptos, Socket, ServerSocket, Hilo. Los cuales servirán para entender como se ha implementado la funcionalidad principal del mismo.

2.1 ServerSocket

Cuando se crea un Server Socket este se mantiene a la escucha de la llegada de nuevos clientes. La manera básica de implementación es la siguiente :

- ServerSocket ss = new ServerSocket(puerto) ; De esta manera se crea un Servidor en el puerto especificado. Como máximo por defecto puede aceptar 50 clientes.
- Según las especificaciones de Sun java :
  - Constructores :

	ServerSocket() Crea un socket servidor desvinculado.
	ServerSocket(int port) Crea un socket servidor en el puerto indicado.
	ServerSocket(int port, int backlog) Crea un socket servidor en el puerto indicado, con un backlog.
	ServerSocket(int port, int backlog, InetAddress bindAddr) Crea un socket servidor en el puerto indicado, con un backlog, y una dirección Ip local.

- Metodos :

Socket	**accept**() Método de escucha para las conexiones que se llevan a cabo en el cliente.
void	**bind**(SocketAddress endpoint) Vincula el SocketServidor en la direccion especificada.
void	**bind**(SocketAddress endpoint, int backlog) Vincula el SocketServidor en la dirección especificada, incluyendo el backlog.
void	**close**() Cierra el socket.
ServerSocketChannel	**GetChannel**() Devuelve el objeto ServerSocketChannel si lo tuviera asociado.
InetAddress	**getInetAddress**() Devuelve la direccion local del servidor.
int	**getLocalPort**() Devuelve el puerto donde esta escuchando el socket.
SocketAddress	**getLocalSocketAddress**() Devuelve la dirección donde esta vinculado el socket, en caso de no estarlo devolverá “null”.
int	**getReceiveBufferSize**() Devuelve el valor de SO_RCVBUF, es el tamaño de buffer propuesto por el servidor para comunicarse con los clientes.
boolean	**getReuseAddress**() SO_REUSEADDR esta activado o no.
int	**getSoTimeout**() Devuelve el SO_TIMEOUT.
protected void	**implAccept**(Socket s) Subclase que sirve para reescribir el método accept.
boolean	**isBound**() Devuelve el estado del socket, si esta o no vinculado.
boolean	**isClosed**(Devuelve si el socket o no esta cerrado.
void	**setReceiveBufferSize**(int size) Indica un tamaño para el Buffer de recepcion.
void	**setReuseAddress**(boolean on) Activa/desactiva el SO_REUSEADDR.
Static void	**setSocketFactory**(SocketImplFactory fac)
void	**setSoTimeout**(int timeout) Activa/desactiva SO_TIMEOUT con el tiempo especificado en milisegundos.
String	**toString**() Devuelve los datos implementados en el socket : Dirección y puerto.

2.2 Socket

Clase que se usa para la comunicación entre 2 maquinas, en este caso entre el robot y el cliente, esta clase se implementa tanto en el servidor como en el cliente, ya que aunque la aplicación servidora crea un ServerSocket, necesita de un Socket cliente para poder comunicarse con la aplicación cliente, implemente y uso :

- Socket s = new Socket(nombre_servidor o ip, puerto; De esta manera se crea el socket cliente a la ip o nombre del servidor y el puerto que se le indique.

  - Constructores :

	Socket() Crea un socket sin conexión
	Socket(InetAddress address, int port) Crea un socket en la dirección especificada y el puerto indicado.
protected	**Socket**(SocketImpl impl) Crea un socket sin conexión, con una interfaz de usuario especifica.
	Socket(String host, int port) Conecta un socket al host y puerto indicado.
	Socket(String host, int port,InetAddress localAddr, int localPort)

  - Metodos :

void	**bind**(SocketAddress bindpoint) Vincula el socket a la direccion local.
void	**close**() Cierra el socket
void	**connect**(SocketAddress endpoint) Conecta el socket con el server.
void	**connect**(SocketAddress endpoint, int timeout) Conecta el server con el servidor, añadiendo un timeout.
SocketChannel	**getChannel**() Devuelve el objeto ServerSocketChannel si lo tuviera asociado.
InetAddress	**getInetAddress**() Devuelve la dirección donde esta conectado el socket.
InputStream	**getInputStream**() Devuelve el buffer de entrada del socket.
boolean	**getKeepAlive**() Indica si SO_KEEPALIVE esta true o false.
InetAddress	**getLocalAddress**()

	Devuelve la dirección local donde esta vinculado el server.
int	**getLocalPort**() Devuelve el puerto local donde esta vinculado el server.
SocketAddress	**getLocalSocketAddress**() Devuelve la dirección a la que esta vinculada el punto final del socket.
boolean	**getOOBInline**() Devuelve si OOBINLINE es true o false.
OutputStream	**getOutputStream**() Devuelve el buffer de entrada del socket.
int	**getPort**() Devuelve al cual esta conectado al puerto.
int	**getReceiveBufferSize**() Devuelve el valor de SO_RCVBUF, tamaño del buffer para la entrada del Socket.
SocketAddress	**getRemoteSocketAddress**() Devuelve la dirección del socket donde el socket se ha conectado.
boolean	**GetReuseAddress**() Devuelve si SO_REUSEADDR es true o false.
int	**getSendBufferSize**() Devuelve el valor de SO_SNDBUF, tamaño del buffer para la salida del Socket.
int	**getSoLinger**() Devuelve las caracteristicas de SO_LINGER.
int	**getSoTimeout**() Devuelve las caracteristicas de SO_LINGER.
boolean	**getTcpNoDelay**() Devuelve si TCP_NODELAY es true o false.
int	**getTrafficClass**() Devuelve el tipo de trafico o el tipo de servicio indicado en la cabecera ip de los paquetes IP enviados desde el socket.
boolean	**isBound**() Devuelve si el socket esta vinculado o no.
boolean	**isClosed**() Devuelve si el socket esta cerrado o no.
boolean	**isConnected**() Devuelve si el socket esta conectado o no.
boolean	**isInputShutdown**() Devuelve si la entrada del socket esta cerrada.
boolean	**isOutputShutdown**() Devuelve si la salida del socket esta cerrada.
void	**sendUrgentData**(int data) Envia un byte de urgencia.
void	**setKeepAlive**(boolean on) Activa/desactiva SO_KEEPALIVE.
void	**setOOBInline**(boolean on) Activa/desactiva OOBINLINE (recipiente TCP de datos urgentes). Por defecto

	esta opción esta deshabilitada y el socket recibe el dato urgente de manera silenciosa.
void	**setReceiveBufferSize**(int size) Define el SO_RCVBUF con el valor especifico del Socket.
void	**setReuseAddress**(boolean on) Activa/desactiva SO_REUSEADDR.
void	**setSendBufferSize**(int size) Indica el valro especido del socket de la opcion SO_SNDBUF.
Static void	**setSocketImplFactory**(SocketImplFactory fac)
void	**setSoLinger**(boolean on, int linger) Activa/desactiva SO_LINGER con el tiempo especificado en milisegundos
void	**setSoTimeout**(int timeout) Activa/desactiva SO_TIMEOUT con el timeout especificado en milisegundos.
void	**setTcpNoDelay**(boolean on) Activa/desactiva TCP_NODELAY (Activa/desactiv el algoritmo de Nagle's).
void	**setTrafficClass**(int tc) Indica que clase de trafico o tipo de socket de servicio en la cabecera IP o en los paquete del Socket.
void	**shutdownInput**() Desactiva el Input.
void	**shutdownOutput**() Desactiva el Output
String	**toString**() Convierte el Socket en un String.

2.3 Hilo

Un hilo es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea.

Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocidos como un proceso. El hecho de que los hilos de ejecución de un mismo proceso compartan los recursos hace que cualquiera de estos hilos pueda modificar éstos. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente.

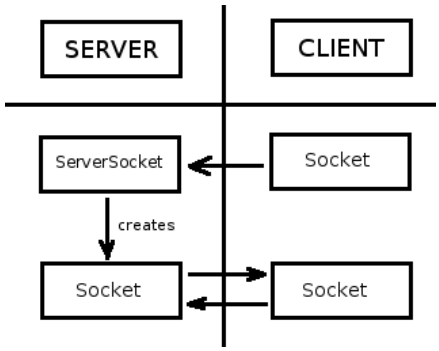
Lo que es propio de cada hilo es el contador de programa, la pila de ejecución y el estado de la CPU(incluyendo el valor de los registros).

El proceso sigue en ejecución mientras al menos uno de sus hilos de ejecución siga activo. Cuando el proceso finaliza, todos sus hilos de ejecución también han terminado. Asimismo en el momento en el que todos los hilos de ejecución finalizan, el proceso no existe más y todos sus recursos son liberados.

3.- Conclusion

Tenemos un Servidor el cual creara mediante hilos de ejecución diferentes servidores-clientes, los cuales se encargaran de la comunicación Robot-PC.

La siguiente imagen intenta explicar un poco la función, cada hilo generara las siguiente acción :



8 CLASES SERVIDOR

1.- Main

Clase principal del servidor, en ella se define tanto el ServerSocket , el numero de clientes que se podrán acceder así como los posibles problemas que tenga el crear el socket.

Codigo:

```
public static void main(String[] args) throws IOException, InterruptedException
{
    ServerSocket ss = null;
    try
    {
       ss = new ServerSocket(0);
    }
    catch(Exception e)
    {
      System.out.println("Error");
    }
    System.out.println("Servidor Abierto Puerto :" + ss.getLocalPort());

    iniciar_servidor(ss);
}
```

El metodo anterior crea el Socket servidor en cualquer puerto libre, para indicamos en la orden un '0' en el puerto, en caso de error se indicaria por consola si por lo contrario se realiza la apertura de manera correcta se indica el puerto.

```
private static void iniciar_servidor(ServerSocket ss) throws IOException
{
    Socket s;
    while(true)
    {
       s = ss.accept();
       n_cli++;
       if(n_cli == 3)
       {
          HiloServidor hs = new HiloServidor(s, true);
          hs.start();
          n_cli--;
       }
       else
       {
          HiloServidor hs = new HiloServidor(s);
          hs.start();
          System.out.println("Numero de clientes : " + n_cli);
       }
    }
}
```

El codigo anteriormente expresado seria el encargado de crear y lanzar los hilos servidor cada vez que un cliente es aceptado

2.- Hilo servidor

Como hemos dicho con anterioridad el servidor creaba hilos para poder trabajar con cada uno de los clientes, bien el siguiente código, lo que define es exactamente ese hilo de ejecución independiente.

Se definen todos los elementos necesarios para el tratamiento del cliente : Socket, PrintWriter (método para mostrar los mensajes en el cliente), Scanner (método de lectura del cliente), y aquellas variables necesarias.

En igual medida se definen aquellos métodos necesarios para conseguir datos necesarios para algunas de las características y metodologías que usa el robot (se explicaran con mas detenimiento mas adelante), como puede ser la curva de Bezier y el control manual.

Codigo :

```
private final double pi = 3.14159265358979323846;
private final double max_vel = 350.0;

private Socket s;
private Scanner leer;
private PrintWriter escribir;
private boolean cierre = false;
```

Variables de uso en el hilo :

- Socket s : Nos permite la conexión entre servidor-cliente.
- Scanner leer : Permite la lectura de los datos introducidos por el cliente.
- PrintWriter escribir : Objeto que se usa para escribir la informacion al cliente.
- Boolean cierra : Variable que se usa para finalizar la ejecución del servidor hasta que esta no sea true, el servidor continuara su ejecución.

```
public HiloServidor(Socket s) throws IOException
{
   this.s = s;
   leer = new Scanner(s.getInputStream());
   escribir = new PrintWriter(s.getOutputStream(),true);
}
```

Este método genera el hilo-servidor si este es aceptado por el servidor.

```
public HiloServidor(Socket s, boolean rechazado) throws IOException
{
   escribir = new PrintWriter(s.getOutputStream(),true);
   escribir.println("Máximo de conexiones alcanzado.");
   escribir.close();
   s.close();
}
```

Este método genera el hilo-servidor informa que se llego al máximo de clientes y lo cierra.

```
private void menu()
{
   escribir.println("- Parametros por defecto -");
   escribir.println("PI : " + pi);
   escribir.println("Velocidad Maxima : " + max_vel);
   escribir.println("--------------------");
   escribir.println("Menu Control Robotino");
   escribir.println("Ordenes Disponibles :");
   escribir.println("1.- Bezier");
   escribir.println("2.- Circular");
   escribir.println("3.- Bezier general (2D, 3D)");
   escribir.println("4.- Encaminamiento");
   escribir.println("--------------------");
   escribir.flush();
}
```

Método que imprime el menú del robot con todas las opciones que están disponibles en el robot.

```
/* Obtención de los puntos de bezier */
private float[][] puntos_bezier() throws IOException
{
   PrintWriter escriBezier = new PrintWriter(s.getOutputStream(), true);
   escriBezier.println("Curva de bezier :");
   escriBezier.println("Ejemplo :");
   escriBezier.println("Puntos : 0.0 0.0, 0.0 100.0, 100.0 100.0, 100.0 0.0");
   escriBezier.flush();
   boolean fin_puntos = false;
   String lista_puntos = "";
   String[] coords;
   float[][] puntosBezier = new float[4][4];
   while(!fin_puntos)
   {
      /* Obetenemos los puntos de la curva */
      Scanner l_puntos = new Scanner(s.getInputStream());
      lista_puntos = l_puntos.nextLine();
      /* Una vez capturados obetenemos un array string con los puntos
       * seraparados en sub arrays.
       */
      coords = lista_puntos.split(", ");
      /* Comprobamos el numero de puntos y su formato */
      if(coords.length > 4)
      {
         escribir.println("Numero excesivo de puntos.");
      }
      else if(coords.length < 4)
      {
         escribir.println("Faltan coordenadas.");
      }
      else
      {
         for(int i = 0; i <= coords.length-1; i++)
         {
            String[] coords_uni = coords[i].split(" ");
            for(int j = 0; j <= coords_uni.length-1; j++ )
            {
               try
               {
                  puntosBezier[i][j] = Float.valueOf(coords_uni[j]);
               }
               catch(NumberFormatException ex)
```

```
                    {
                        escribir.println(ex.getMessage());
                        break;
                    }
                }
            }
            fin_puntos = true;
        }
    }
    escriBezier.close();
    return puntosBezier;
}
```

El código anterior sirve para poder capturar los puntos necesarios para la curva de bezier, esta curva se compondrá de 4 puntos, los cuales serán introducidos por el usuario. Los puntos serán del tipo Float.

```
public float[][] bezier_general() throws IOException
  {
        escribir.println("Curva de Bezier general : ");
        escribir.println("Procederemos a introducir los valores.");
        escribir.println("Dimensiones de la curva : 2 o 3");

        int aux = 0, tam = 0;
        Scanner escriBezier = new Scanner(s.getInputStream());
        boolean opcion = false;
        while(!opcion)
        {
            /* Capturamos la dimension de la curva */
            try
            {
               aux = Integer.parseInt(escriBezier.nextLine());
               System.out.println(aux);
               if(aux == 2 || aux == 3 )
               {
                   /* Capturamos el numero de puntos */
                  escribir.println("Numero de puntos :");
                  try
                  {
                     tam = Integer.parseInt(escriBezier.nextLine());
                     opcion = true;
                  }
                  catch(Exception e)
                  {
                     escribir.println("Introduzca un valor numerico");
                  }

               } else
               {
                 escribir.println("Error en la dimesion.");
                 escribir.println("Intorduza la dimesion : 2 o 3");
               }
            }
            catch(Exception e)
            {
               escribir.println("Introduzca un valor numerico");
            }
        }


        escribir.println("Bezier general de : " + aux + " dimensiones y Nº de puntos : " + tam);
        escribir.println("Introduce los puntos separados por comas.");
```

```
String lista_puntos;
String[] coords;
float[][] bezgen = null;

/*** Array de coordenadas 2D ***/

if(aux == 2)
{
   /* Variables */

   boolean ok = false;

   while(!ok)
   {
      int z = 0;
      escribir.println("Ejemplo 2D : x y, x y");
      bezgen  = new float[tam][2];

      lista_puntos = escriBezier.nextLine();

      coords = lista_puntos.split(", ");

      System.out.println("Puntos capturados : " + lista_puntos);

      if(tam ==  coords.length)
      {

          for(int i = 0; i <= coords.length-1; i++)
      {
         String[] coords_uni = coords[i].split(" ");

         for(int j = 0; j <= coords_uni.length-1; j++ )
         {
            try
            {
               bezgen[i][j] = Float.valueOf(coords_uni[j]);
            }
            catch(NumberFormatException ex)
            {
               escribir.println(ex.getMessage());
               break;
            }
         }

      }
         ok = true;
      }
      else
      {
         escribir.println("Numero excesivo o inferior de coordenadas.");
         escribir.println("Coordenadas a introducir : " + tam);
         escribir.println("Coordenadas introducizas : " + coords.length);
      }
   }
}

/*** Array de coordenadas 3D ***/

if(aux == 3)
{
   /* Variables */
```

```
                boolean ok = false;

                while(!ok)
                {

                   escribir.println("Ejemplo 3D : x y z, x y z");
                   bezgen = new float[tam][3];

                   lista_puntos = escriBezier.nextLine();

                   coords = lista_puntos.split(", ");

                   System.out.println("Puntos capturados : " + lista_puntos);


                   if(tam ==  coords.length)
                   {

                       for(int i = 0; i <= coords.length-1; i++)
                   {
                      String[] coords_uni = coords[i].split(" ");

                      for(int j = 0; j <= coords_uni.length-1; j++ )
                      {
                         try
                         {
                            bezgen[i][j] = Float.valueOf(coords_uni[j]);
                         }
                         catch(NumberFormatException ex)
                         {
                            escribir.println(ex.getMessage());
                            break;
                         }
                      }

                   }
                      ok = true;
                   }
                   else
                   {
                      escribir.println("Numero excesivo o inferior de coordenadas.");
                      escribir.println("Coordenadas a introducir : " + tam);
                      escribir.println("Coordenadas introducizas : " + coords.length);
                   }
                }
             }
             return bezgen;
          }
```

El anterior método se usa para capturar los puntos de una curva de Bezier general. A diferencia de la Bezier anterior, esta no tiene predefinido un tamaño ni una dimensión si no que tendrá X numero de puntos y 2 o 3 Dimensiones, el usuario introducirá los puntos, comprobando los posibles errores y mostrando el error correspondiente.

En el siguiente apartado trataremos el método que gestiona todas las llamadas a los servicios que puede ofrecer el servidor.

```
escribir.println("Conexion Establecida" );

        menu();

        while(!cierre)

          try
          {
            int op = 9999;

            System.out.println(String.valueOf(servidor_robot.Main.ocupado));
            while(! servidor_robot.Main.ocupado )
            {

              System.out.println("Introduce la opcion : ");
              leer.reset();
              op = Integer.parseInt(leer.nextLine());
              //System.out.println(op);
              if ( op >= 0 || op <= 9 )
              {
                break;
              }
            }
```

Esta primera sección de código se encarga de mostrar que la conexión ha sido establecida y mostrar el menú al cliente. Una vez mostrado el menu y mientras el cliente no decida salir de la aplicación (while), se le pedira que introducza una opcion valida. Aparte se ofrece una guarda por si el robot esta ejecutando alguna orden, gracias al uso de la variable global "ocupado", la cual ofrece una guarda para que el robot solo ejecute las ordenes de solo 1 cliente.

```
            if( servidor_robot.Main.ocupado == true)
            {

              escribir.println("Robot ejecutando ordenes.");
              escribir.flush();
              menu();
            }
            else
            {
```

Esta sección le indica al cliente si el servidor esta ocupado e informa al cliente. De que no puede ejecutar su orden.

```
if(op == 0)
              {
                try
                {
                  servidor_robot.Main.n_cli--;
                  System.out.println("Conexion cerrada por el cliente");
                  s.close();
                  cierre = true;
                }
                catch (IOException ex)
                {

                }
              }
              else if (op > 4 || op < 0 )
```

```
            {
               escribir.println("Opcion seleccionada incorrecta, introduzca nuevo valor.");
               escribir.flush();
            }
```

La sección anterior, se encarga de indicar si el cliente, quiere cerrar la conexión o por el contrario la opción que ha seleccionado es incorrecta.

Bien por lo tanto podemos definir que tenemos 2 elementos de guarda para poder acceder a los servicios ofrecidos por el servidor :

1. Comprobación que el robot no esta ejecutándose.
2. Que la opción que se indica es valida para el servidor.

A continuación se definirán todos las posibles opciones y las llamadas que se realizan para las acciones al robot.

Todas las opciones tienen elementos en común, cada vez que se inicia el servicio se bloquea el acceso a las acciones del robot designando a la variable global (ocupado) el valor 'true'.

Una vez finalizada la acción a realizar se liberara el acceso, designando a la variable global el valor 'false', aparte se llamara al método menú, para mostrar las acciones otra vez.

Código de bezier :

```
  else if (op == 1)
{
   servidor_robot.Main.ocupado = true;
   /* Metodo de captura de puntos */

    int tiempo=0;
    boolean ok = false;

     while(!ok)
     {
        escribir.println("Indique la duración de la curva :");
        try
        {
           tiempo = Integer.valueOf(leer.nextLine());
           ok = true;
           float[][] puntFin = puntos_bezier();
           Bezier auxBezier = new Bezier(puntFin);
           new java.lang.Thread(auxBezier).start();
           while(tiempo != 0)
           {
              tiempo = (int) (tiempo - 0.01);
           }
           auxBezier.desconexion();
        }
           catch(Exception ex)
        {
              escribir.println(ex.getMessage());
        }
```

```
        }
      servidor_robot.Main.ocupado = false;
      menu();
    }
```

En la curva de bezier, aparte de indicar el conjunto de puntos de la curva, le indicaremos el tiempo que queremos que la curva se ejecute.

Código de bezier general (Solo se indica el código ya que no existe una diferencia exagerada).

```
else if (op == 3)
{
  servidor_robot.Main.ocupado = true;

  System.out.println("Inicio Bezier general");

  float[][] puntFin = puntos_bezier();
  int tiempo=0;
  boolean ok = false;

  while(!ok)
  {
    escribir.println("Indique la duración de la curva :");
    try
    {
      tiempo = Integer.valueOf(leer.nextLine());
      ok = true;
    }
    catch(Exception ex)
    {
      escribir.println(ex.getMessage());
    }
  }

  BezierGeneral auxBezier = new BezierGeneral(puntFin, tiempo);
  new java.lang.Thread(auxBezier).start();
  System.out.println("Fin curva bezier general");

  servidor_robot.Main.ocupado = false;

  menu();

}
```

Código de circular :

```
else if (op == 2)
        {
          servidor_robot.Main.ocupado = true;

          escribir.println("Movimiento Circular");
          escribir.println("Para finalizar : parar");
          escribir.flush();

          Circular c_robot = new Circular();
          new java.lang.Thread(c_robot).start();

          boolean parar = false;

          while(!parar)
```

```
        {
           if(leer.nextLine().equals("parar"))
           {
              parar = true;
           }
        }

        c_robot.fin();
        servidor_robot.Main.ocupado = false;
        menu();

     }
```

El código anterior corresponde a la acción Circular, en esta acción el cliente tiene el poder de finalizar la misma. Se introduce la palabra parar y la acción se finalizara.

Código de control manual :

```
else if (op == 4)
    {

      servidor_robot.Main.ocupado = true;

      escribir.println("-- Control Manual Activado -- ");

      ControlManual CM = new ControlManual();
      CM.init();

      while(leer.nextLine().equals("salircm"))
      {
         CM.setVelocidad(leer.nextLine());
      }

      CM.finConexion();

      servidor_robot.Main.ocupado = false;

      menu();

    }
```

El código anterior se encarga del control manual del robot, en este código se controla cuando se activa y se desactiva el control numérico con el robot.

3.- Código común

El siguiente código es común en todas las clases de ejecución en el robot por lo que con solo explicar su funcionalidad una vez no es necesario repetir código en todos las clases siguiente.

```
protected Com auxcom;
protected Motor m1;
protected Motor m2;
protected Motor m3;
protected OmniDrive auxom;
protected Bumper auxbump;
```

Las variables anteriores son necesarias para que el robot ejecute las ordenes introducidas en el robot. Los elementos son los siguientes :

- Com auxcom : Realiza la conexión Com con el robot.
- Motor m1, m2, m3 : cada uno de los 3 motores del robot.
- OmniDrive auxom : Driver de control del robot permite la conducción del robot indicandole la velocidad a cada una de ellos.
- Bumper auxbump : Sensor de colisión.

Una vez hecha la declaración de las variables, estas deben ser creadas e inicializadas para poder trabajar con ellas, lo cual se realiza de la siguiente manera :

```
auxcom = new Com();
m1 = new Motor();
m2 = new Motor();
m3 = new Motor();
auxom = new OmniDrive();
auxbump = new Bumper();
```

Con el código anterior creamos las variables. Este código se encontrara

```
/* Indicamos el controlador de motores */
m1.setComId(auxcom.id());
m2.setComId(auxcom.id());
m2.setComId(auxcom.id());

/* Indicamos  el numero a cada motor */
m1.setMotorNumber(0);
m2.setMotorNumber(1);
m3.setMotorNumber(2);

/* Configuramos el OmniDrive y el Bumper */
auxom.setComId(auxcom.id());
auxbump.setComId(auxcom.id());
```

Gracias al código anterior inicializamos todas las variables necesarias para el control del robot y su conducción.

Conexión	Desconexión

public void conexion()
{
 	
try
{
auxcom.setAddress("172.26.1.1");
auxcom.connect();
	public void fin()
	{
	auxcom.disconnect();
	}

```
    }
    catch(Exception e)
    {
       System.out.println(e.getMessage());
    }

}
```

Los 2 codigos especificados anteriormente sirven para la conexión y desconexión del robot.

4.- Circular

Esta clase define un movimiento circular del robot, es una clase llamada desde el Hilo Servidor y solo se detendrá cuando el cliente lo indique con la orden especifica.

Codigo

```
protected final float[] startVector = new float[]
{
   200.0f, 0.0f
};
```

Vector inicial del robot.

```
public void ejec()
{

   float[] dir;
   float a = 0.0f;
   long startTime = System.currentTimeMillis();
   while (!Thread.interrupted() && auxcom.isConnected() && false ==auxbump.value())
   {
      long elapsedTime = System.currentTimeMillis() - startTime;

      //rotate by 360 degrees every 10 seconds
      dir = rotate(startVector, a);
      a = 360.0f * elapsedTime / 10000;
     auxom.setVelocity(dir[0], dir[1], 0);
     auxcom.waitForUpdate();
   }

}

private float[] rotate(float[] in, float deg)
{
   final float pi = 3.14159265358979f;

   float rad = 2 * pi / 360.0f * deg;

   float[] out = new float[2];
   out[0] = (float) (cos(rad) * in[0] - sin(rad) * in[1]);
   out[1] = (float) (sin(rad) * in[0] + cos(rad) * in[1]);
   return out;
```

```
}
```

Método para la ejecución del movimiento circular en ellos se calculan las variables necesarias, una vez se calculan se le indica al robot la velocidad de las ruedas la cual gracias a la diferencia entre ellas, permite la realización de dicho movimiento.

```
public void run()
{
   ini();
   conexion();
   ejec();
}
```

Método para la ejecución del hilo, en dicho método se llama a todos los métodos que permiten la realización de la acción reclamada por el cliente.

3.- Beizier y Bezier general

1.- Introducción

Se denomina curvas de Bézier a un sistema que se desarrolló hacia los años 1960, para el trazado de dibujos técnicos, en el diseño aeronáutico y de automóviles. Su denominación es en honor a Pierre Bézier, quien ideó un método de descripción matemática de las curvas que se comenzó a utilizar con éxito en los programas de CAD.

Las curvas de Bézier fueron publicadas, por primera vez en 1962 por el ingeniero de origen francés Pierre Bézier, que las usó posteriormente, con profusión, en el diseño de las diferentes partes de los cuerpos de un automóvil, en sus años de trabajo en la Renault. Las curvas fueron desarrolladas por Paul de Casteljau usando el algoritmo que lleva su nombre. Se trata de un método numéricamente estable para evaluar las curvas de Bézier.

Posteriormente, los inventores del PostScript, lenguaje que permitió el desarrollo de sistemas de impresión de alta calidad desde el ordenador, introdujeron en ese código el método de Bézier para la generación del código de las curvas y los trazados. El lenguaje PostScript sigue empleándose ampliamente y se ha convertido en un estándar de calidad universal; por ello, los programas de diseño vectorial como Adobe Illustrator, el extinto Macromedia FreeHand, Corel Draw, tres de los más importantes programas de dibujo vectorial y otros como Inkscape, denominan como "bézier" a algunas de sus herramientas de dibujo, y se habla de "Trazados bézier", "pluma bézier", "lápiz bézier", etc. Su facilidad de uso la ha estandarizado en el diseño gráfico, extendiéndose también a programas de animación vectorial como Adobe Flash, y retoque fotográfico (bitmap) como Photoshop y Gimp, donde se usa para crear formas cerradas o selecciones.

La idea de definir geométricamente las formas no es demasiado compleja: un punto del plano puede definirse por coordenadas. Por ejemplo, un punto A tiene unas coordenadas (x1, y1) y a un punto B le corresponde (x2,y2). Para trazar una recta entre ambos basta con conocer su posición.

Si en lugar de unir dos puntos con una recta se unen con una curva, surgen los elementos esenciales de una curva Bézier: los puntos se denominan puntos de anclaje o nodos. La forma de la curva se define por unos puntos invisibles en el dibujo, denominados puntos de control, manejadores o manecillas.

2.- Bezier

El siguiente código es el usado para realizar la curva de bezier cuadrática.

Codigo :

```
private float[][] array = new float[3][3];

  public void ejecucion()
  {
    float traveltime = 10.0f;

    Double theta = (0.0 * 3.14159265358979323846 / 180.0);

    float v_x_abs = 0.0f;
        float v_y_abs = 0.0f;
    float v_x_rel = 0.0f;
    float v_y_rel = 0.0f;
        float t = 0.0f;

    /* Metodo para realizar el recorido de la curva */

    while(bumper.value() == false && t < 1.0 && com.isConnected())
    {
      t += 0.01;

      v_x_abs = (3*array[0][0]-3*array[0][0]-3*t*t*(array[0][0]-3*array[1][0]+3*array[2][0]-array[3][0])
+2*t*(3*array[0][0]-6*array[1][0]+3*array[2][0]))/traveltime;
      v_y_abs = (3*array[0][1]-3*array[0][1]-3*t*t*(array[0][1]-3*array[1][1]+3*array[2][1]-array[3][1])
+2*t*(3*array[0][1]-6*array[1][1]+3*array[2][1]))/traveltime;

      v_x_rel = (float) java.lang.Math.cos((theta)*v_x_abs + java.lang.Math.sin(theta)*v_y_abs);
      v_y_rel = (float) (-java.lang.Math.sin(theta) * v_x_abs + java.lang.Math.cos(theta) * v_y_abs);

      omniDrive.setVelocity( v_x_rel, v_y_rel, 0 );
      com.waitForUpdate();
    }

    /* Indicamos que la curba de bezier finalizo */
    fin = true;

  }

  public boolean fin()
  {
    return fin;
  }

  public void run()
  {
    ini();
    conexion();
    ejec();
  }
```

3.- Bezier General

A diferencia de la bezier cubica esta curva, se define por 2 características :

- Definiremos la dimensión de la curva.
- Y el tamaño de puntos.

En el apartado del hilo-servidor indicamos el método necesario para capturar los puntos de la curva e introducir la dimensión deseada. Al mismo tiempo hemos indicado los métodos necesarios para la conexión y desconexión interna del robot. Por lo tanto

Por lo tanto solo explicaremos los métodos necesarios para la confección y ejecución de la curva de bezier.

Antes de pasar con los métodos explicaremos que metodología hemos usado para el calculo, en el caso de las curvas de orden cubico y cuadrático el calculo es bastante simple, pero en curvas de grado superior es necesario el uso de algoritmos mas complejos para su calculo (dividirla en trozos por ejemplo). Pero en nuestro caso hemos usado el Polinomio de Bernstein, el cual pasamos a explicar a continuación.

Definición del algoritmo de Bernstein :

Los polinomios de Bernstein o polinomios en la base de Bernstein son una clase particular de polinomios (en el campo de los números reales que son utilizados dentro del ámbito del análisis numérico.

El algoritmo de evaluación más numéricamente estable es el de de Casteljau.

Polinomio de Bernstein : $$P(x) = \sum_{k=0}^{n} c_k B_k^n(x)$$

donde los $B_k^n(\cdot)$ son elementos de la base de los polinomios de Bernstein, definidos de:

$$B_i^n(x) = \binom{n}{i}(1-x)^{n-i}x^i \quad \text{si} \quad x \in [0,1];$$

o, más en general:

$$B_i^n(x) = \binom{n}{i}\frac{(b-x)^{n-i}(x-a)^i}{(b-a)^n} \quad \text{si} \quad x \in [a,b];$$

Una vez explicado la metodología usada, la transformamos en código java, el cual ejecutara lo anteriormente indicado.

```
        public void ejec()
{

  /* Calculo la constante n */
      int n = beziergen.length;  //número de puntos menos 1

  /* Variables coordenada */

  float t = 0.0f;

  for(int k = 0; k != tiempo && auxbump.value() == false && auxcom.isConnected() ; k += 0.01)
  {
    t = k / tiempo;
    float aux_x = 0.0f;
    float aux_y = 0.0f;

    for(int i = 0; i <= n ; i++)
    {
      aux_x += aux_x + beziergen[i][i] * bezier(i,n,t);
      aux_y += aux_y + beziergen[i][i] * bezier(i,n,t);
    }
    auxom.setVelocity(aux_x, aux_y, 0.0f);

  }

}
```

El anterior método es le general para el calculo y ejecución de la curva, en ella se usa la variable t, la cual es introducida por el usuario para saber el numero de iteraciones y aproximaciones de la curva.

El método siguiente se usa para el calculo del polinomio de bernstein :

```
        private float bezier(int i, int n, float t)
{
  float var1 = (float) java.lang.Math.pow(t, i);
  float var2 = (float) java.lang.Math.pow((1-t), (n-i));
  float aux = nsobrem(n,i)*var1*var2;

  return aux;

}
```

Lo siguientes métodos se usan para calcular los elementos necesaiors del polinomio de bernstein y asi obtener los puntos de la curva de bezier :

```
private float nsobrem(int n, int m)
{
  return factorial(n) / factorial(m) * factorial(m-n);
}

private float factorial(int i)
{
```

```
    float factorial = 1f;

    while(i != 0)
    {
      factorial = factorial * i;
      i--;
    }

    return factorial;
  }
```

4.- Control Manual

El siguiente código es el usado para el control manual del robot. A diferencia de los anteriores métodos este no se ejecuta con hilos ya que la orden con el robot es la de modificar la velocidad de las rueda/s que nos interese para dirigir el robot.

Por lo tanto no es necesario una ejecución en hilo como eran los anteriores casos que si necesitaban de un tiempo de ejecución propio y al no ser ejecutado en hilos bloqueaban todo el programa.

```
  private float speed = 200.0f;
  private Motor motor1;
  private Motor motor2;
  private Motor motor3;
  private OmniDrive omniDrive;
  private Bumper bumper;
  private Com com;
```

Como en los casos anteriores se definen las variables de control, aparte se define una variable de velocidad fija para el robot.

```
  public ControlManual()
  {
    com = new Com();

    motor1 = new Motor();
    motor2 = new Motor();
    motor3 = new Motor();
    motor1.setMotorNumber(0);
    motor2.setMotorNumber(1);
    motor3.setMotorNumber(2);

    omniDrive = new OmniDrive();
    bumper = new Bumper();

    omniDrive.setComId(com.id());
    bumper.setComId(com.id());

    init();
  }

  public final void init()
  {
```

```
    com = new Com();

    motor1 = new Motor();
    motor2 = new Motor();
    motor3 = new Motor();
    motor1.setMotorNumber(0);
    motor2.setMotorNumber(1);
    motor3.setMotorNumber(2);

    omniDrive = new OmniDrive();
    bumper = new Bumper();

    omniDrive.setComId(com.id());
    bumper.setComId(com.id());

    /* Conexion con el robot */
     try
    {
       com.setAddress("172.26.1.1");
       com.connect();
    }
    catch(Exception e)
     {
       System.out.println(e.getMessage());
    }
  }

  public void finConexion()
  {
    com.disconnect();
  }
```

En igual medida creamos e inicializamos todas las variables, la diferencia radica que aquí ya establecemos la conexión con el robot una vez inicializada la variable "auxcom".

Aparte se incluye el metodo "finConexion", para finalizar la conexión con el robot.

```
  public void setVelocidad(String dir)
  {
    if( dir.equals("n"))
    {
       omniDrive.setVelocity(speed, 0.0f, 0.0f);
    }
    if( dir.equals("s"))
    {
       omniDrive.setVelocity(-speed, 0.0f, 0.0f);
    }
    if( dir.equals("d"))
    {
       omniDrive.setVelocity(0.0f, speed, 0.0f);
    }
    if( dir.equals("i"))
    {
       omniDrive.setVelocity(0.0f, -speed, 0.0f);
    }
```

```
    if( dir.equals("stop"))
    {
      omniDrive.setVelocity(0.0f, 0.0f, 0.0f);
    }
  }
```

Y por ultimo de conducción del robot, este método es muy simple, según la tecla presionada por el cliente, se le enviá un string con la dirección que el robot debe tomar, actualizando el omnidrive.

INSTALACION Y EJECUCION SERVIDOR

1 Programas

1.- Putty

1.- Definición

PuTTY es un cliente SSH, Telnet, login, y TCP Raw con licencia libre. Disponible originalmente sólo para Windows, ahora también está disponible en varias plataformas Unix, y se está desarrollando la versión para Mac OS clásico y Mac OS X. Otra gente ha contribuido con versiones no oficiales para otras plataformas, tales como Symbian para teléfonos móviles.

2.- Características

Algunas características de PuTTY son:

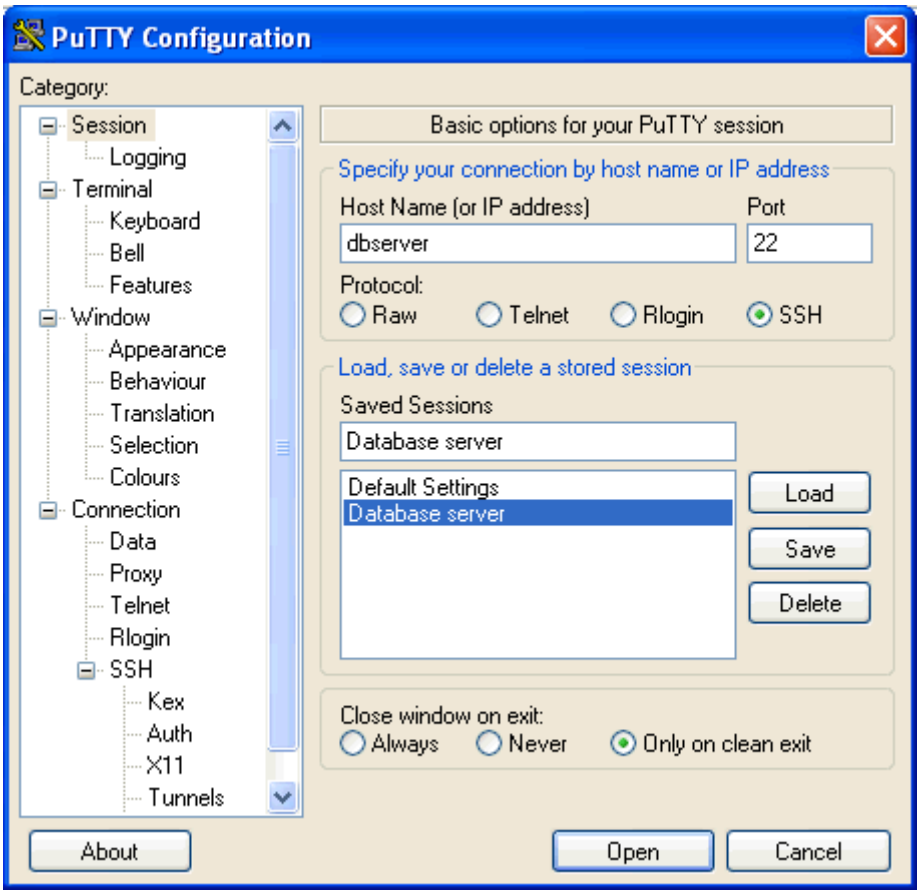
- El almacenamiento de hosts y preferencias para uso posterior.
- Control sobre la clave de cifrado SSH y la versión de protocolo.
- Clientes de línea de comandos SCP y SFTP, llamados "pscp" y "psftp" respectivamente.
- Control sobre el redireccionaminento de puertos con SSH, incluyendo manejo empotrado de reenvío X11.
- Completos emuladores de terminal xterm, VT102, y ECMA-48.
- Soporte Ipv6.
- Soporte 3DES, AES, RC4, Blowfish, DES.

- Soporte de autenticación de clave pública.
- Soporte para conexiones de puerto serie local.

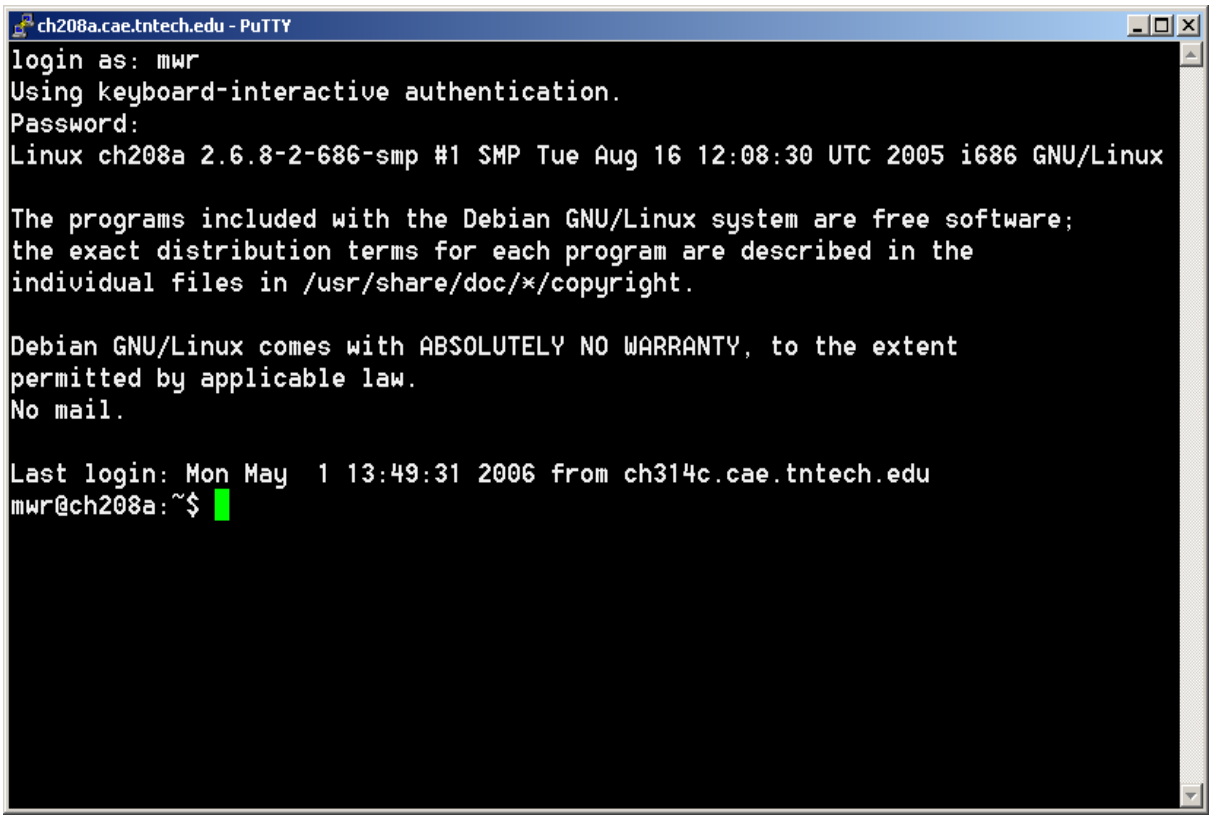
El nombre PuTTY proviene de las siglas Pu: Port unique TTY: terminal type. Su traducción al castellano seria: Puerto único para tipos de terminal.

3.- Ejecución del servidor

1. En la siguiente imagen se muestra la pantalla de inicio, donde se indicara la dirección y el puerto donde nos vamos a conectar.



2. Una vez iniciada la conexión con el robotino se mostrara una ventana consola la cual nos indicara que introduzcamos el usuario y la contraseña. La siguiente ventana muestra la consola típica que se nos mostrara al loguear con Putty en un sistema Unix remoto (como el caso del robotino).



3. Situados ya en la carpeta inicial del robotino, estaremos accediendo al directorio donde estara copiado el servidor del robot. Para su ejecución solo deberos acceder a la carpeta “dist” y utilizar la siguiente orden :
    - java -jar Sevidor_Robot

4. Una vez indicada la orden anterior, para finalizar la ejecución, usaremos Ctrl + C.

2.- WinSCP

1.- Definición

WinSCP es una aplicación de Software Libre. WinSCP es un cliente SFTP gráfico para Windows que emplea SSH. El anterior protocolo SCP también puede ser empleado. Su función principal es facilitar la transferencia segura de archivos entre dos sistemas informáticos, el local y uno remoto que ofrezca servicios SSH.

Esta página es una pequeña introducción en castellano, ya que la mayor parte de la documentación de WinSCP se encuentra únicamente en inglés, así como la mayor parte del contenido referido en los enlaces.

2.- Características

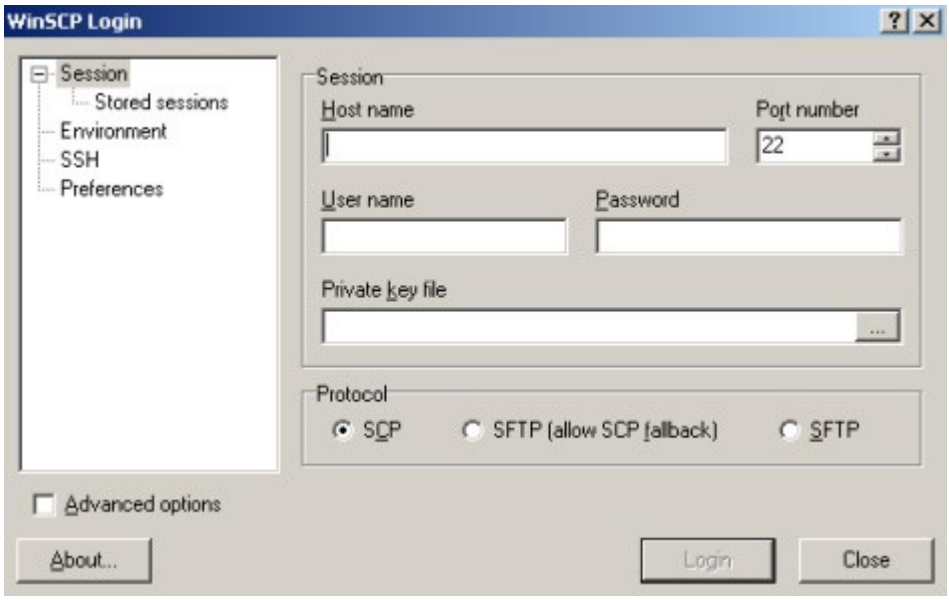
Interfaz gráfica (GUI) :

- Disponible en varios idiomas.
- Integración con Windows (drag&drop, URL, accesos directos)
- Soporte de las operaciones comunes de archivo.
- Soporte de protocolos SCP y SFTP sobre SSH-1 y SSH-2.
- Soporte de operaciones programadas (batch), guiones e interfaz de línea de comandos.
- Sincronización de directorios de varias maneras, semi o completamente automatizadas.
- Editor de texto integrado.
- Soporte de autenticación mediante contraseña SSH, método keyboard-interactive, clave pública o Kerberos (GSS).
- Se integra con Pageant (Agente SSH de PuTTY) para ofrecer soporte completo de autenticación mediante clave pública.
- Interfaces similares al Explorador de Windows (panel único) o al Comandante Norton (panel dual).
- Opcionalmente es posible guardar la información de sesión.

- Posibilidad de almacenar la configuración del programa en un archivo de configuración en vez de en el registro de Windows, lo que facilita su uso desde unidades portátiles, como discosUSB y GVC.

3.- Ejecución

1. Una vez instalado, al ejecutar el programa nos aparecera la siguiente imagen :



En ella introduciremos los datos para realizar la conexión con el robotino, tales como la IP, el nombre de usuario y la contraseña, procediendo a presionar el boton de "Login"

2. Ya logueados lo q nos mostrara sera la típica ventada de intercambio de archivos FTP por lo que solo queda proceder a subir los archivos al robotino.