DESARROLLO DE UNA ARQUITECTURA CLIENTE-SERVIDOR ASÍNCRONA BASADA EN EVENTOS POR NOTIFICACIÓN DE MENSAJES PARA LA COOPERACIÓN Y COORDINACIÓN DE ROBOTS MOVILES.

Autor:	**Francisco Luis Gómez Pérez**
Correo electrónico:	fragope2@posgrado.upv.es
Asignatura:	**Tesina de MAII**
Fecha de entrega:	**14-07-10**

ÍNDICE

1. Introducción

Debido al auge y crecimiento de las comunicaciones en todos los entornos industriales, informáticos y, en general de cualquier sector, cada vez cobra mayor importancia los tipos de arquitectura empleadas en estos sectores, y por ello hay que realizar un buen diseño de esta arquitectura, atendiendo a los diferentes mecanismo de los que disponemos, en función de las tareas y problemas que se quieran resolver.

En el presenta trabajo se describe el uso de una arquitecturas cliente–servidor para la realización de tareas conjuntas y coordinación de varios robots móviles con el objetivo de llegar a realizar alguna tarea conjunta. En este ámbito, cuando disponemos de varios dispositivos encargados de realizar tareas coordinadas con el resto de componentes, existen diversos problemas en lo que respecta a la comunicación de todos estos dispositivos.

Por un lado, se debe de estudiar el tipo de sincronismo que se va a emplear en esta comunicación. Por ello, un aspecto crucial a la hora de diseñar cualquier tipo de aplicación que hace uso de las comunicaciones, es el sincronismo. La arquitectura que se verá más adelante, es asíncrona, mediante eventos por notificación de mensajes. Existen varias herramientas para programar la comunicación de forma síncrona o asíncrona. Básicamente, depende del sistema operativo con el que estemos trabajando y la herramienta de programación. Si es un entorno Windows, una forma fácil de indicar el tipo de sincronismo es mediante el uso de llamadas y funciones facilitadas por la librería “WinSock”, como por ejemplo, es el uso de la función “WSAAsyncSelect” para indicar que el socket utilizado es asíncrono, con una recepción mediante eventos por notificación de mensajes. Esta función se verá con mayor detalle en puntos posteriores. En cambio, en un entorno Unix / Linux, existen otros tipos de herramientas que son similares, como por ejemplo es, el uso de señales para realizar una comunicación asíncrona.

Otro aspecto importante es el tipo de arquitectura a utilizar. Existe infinidad de arquitecturas, tales como “peer-to-peer” (P2P), multicapas, “cliente-servidor”, etc. que, en función de las tareas que se quieran programar se escoge una u otra arquitectura. En el proyecto actual se usa la arquitectura “cliente-servidor” de dos capas, es decir, sólo existen clientes y servidores y no se necesitan de mas capas, como por ejemplo una capa de servidores de bases de datos. La idea de esta comunicación es simple: los clientes solicitan al servidor la realización de alguna tarea o información, de tal forma que cuando éste reciba la información del cliente, realiza el trabajo solicitado y, una vez finalizada dicha tarea, el servidor devuelve algún tipo de información al cliente. Además, se pueden tener varios clientes y varios servidores. En la parte teórica de este trabajo se verán varios ejemplos con este tipo de arquitectura. Por otro lado, en la parte práctica, se realizan diferentes variantes de esta arquitectura, de tal forma que en un primer lugar se realiza sólo con un cliente y un servidor. El cliente será un PC y el servidor será el robot móvil Create de IRobot. En un segundo lugar, se realiza otra variante de esta arquitectura utilizando un cliente y dos servidores (dos robots móviles).

El documento está compuesto de dos partes: por un lado, se verán todos los aspectos teóricos necesarios para abordar el problema. En una segunda parte, se estudian en detalle los experimentos realizados en el laboratorio con el fin de aplicar esta arquitectura a diferentes robots móviles.

En esta primera parte, se estudia en profundidad la arquitectura utilizada (cliente-servidor), así como diferentes ejemplos de entornos donde sería útil la utilización de esta arquitectura. Además, se realiza un estudio de los sockets, que es la herramienta empleada para la comunicación de los diferentes elementos. Como el objetivo es utilizar esta arquitectura para realizar un trabajo conjunto y coordinado, el trabajo práctico consiste en coordinar a dos robots para que sean capaces de recoger una pelota de forma coordinada. Esto da lugar a ver otros aspectos relacionados con esta tarea, como son generación y seguimiento de trayectorias, así como la utilización de una cámara para poder procesar las imágenes captadas y poder guiar a los robots a las posiciones indicadas. Esto conlleva un trabajo adicional (segmentación de imágenes, calibración de la cámara, etc.), el cual es explicado en la parte teórica.

En un segundo apartado se resumen las características más importantes de los diferentes robots y del sistema embebido empleado y se abordan dos casos prácticos. Por un lado, se desarrolla una arquitectura cliente-servidor síncrona para familiarizarnos con el funcionamiento de ésta y de todas las herramientas empleadas. Por otro lado, se desarrolla una arquitectura cliente-servidor con una parte de la comunicación asíncrona, necesaria debido al tipo de tarea que se quiere resolver. Es decir, el objetivo de este segundo trabajo práctico es coordinar a los dos robots existentes en el laboratorio (Koala de K-Team y Create de IRobot) para poder recoger una pelota, que es detectada mediante una cámara cenital. Por tanto, en las siguientes secciones se explicarán todos los aspectos teóricos y prácticos para poder conseguir los objetivos establecidos.

Finalmente, a modo práctico, se estudia una alternativa de comunicación de uno de los dos robots con un dispositivo móvil. Esto es, manejar el Create de IRobot con el dispositivo móvil OMNIA I900 de SAMSUNG, mediante movimientos similares a los de un volante de un automóvil. La comunicación se realiza mediante la tecnología Bluetooth. Este mecanismo de conducción, además, será empleado para llevar de forma manual al Create a la posición de partida deseada para realizar la tarea coordinada expuesta anteriormente.

2. Aspectos teóricos.

Tal y como se ha mencionado en la introducción de este documento, el objetivo es el estudio y explotación de la arquitectura cliente-servidor para poder emplearla en diferentes aplicaciones o situaciones que sean de forma eficaz, la solución optima a dicha situación. Hay que tener en cuenta que al final de este trabajo se ha utilizado este tipo de arquitectura para resolver un problema práctico con el uso de este tipo de arquitectura y de forma asíncrona. Por ello, es necesario estudiar todas las herramientas necesarias para conseguir resolver el problema de la coordinación de estos dos robots para recoger una pelota.

En los siguientes puntos se explicará con detalle todo lo referente a la arquitectura empleada, entornos donde se puede aplicar esta arquitectura, generación de trayectorias con evitación de obstáculos, etc.

2.1 Arquitectura cliente-servidor

Esta arquitectura consiste básicamente en un cliente que realiza peticiones a otro programa (el servidor) que le da respuesta. De lo que se trata, es de realizar una operación conjunta de varios robots para alcanzar un objetivo.

Atendiendo a quién realiza el proceso de la información existen básicamente cuatro tipos de arquitectura cliente-servidor:

- Cliente Activo, Servidor Pasivo: El cliente realiza la totalidad del trabajo de procesado de la información.
- Cliente Pasivo, Servidor Pasivo: Tanto el cliente como el servidor simplemente pasan información.
- Cliente Pasivo, Servidor Activo: El Servidor realiza todo el trabajo de procesado y el cliente simplemente presenta los datos.
- Cliente Activo, Servidor Activo: Tanto el Servidor como el Cliente procesan la información. Ejemplo: Servicios de Correo Electrónico.

En los casos prácticos que se han desarrollado, como se verá más adelante, tanto el servidor como el cliente son activos, pues ambos dispositivos procesan información.

Además, la filosofía de este tipo de arquitectura permite tener varios clientes e incluso varios servidores. En la figura 1 se muestra un ejemplo de este tipo de arquitecturas.

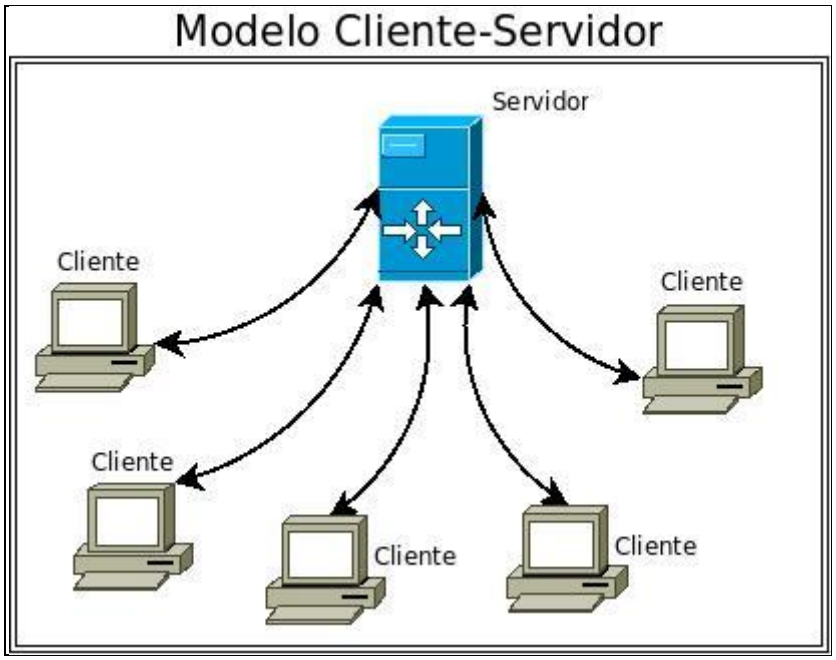


Figura 1. Arquitectura cliente-servidor con un único servidor

Como se ha dicho, no necesariamente tiene que existir un único servidor. Por ejemplo, se puede configurar un servidor para realizar tareas de procesamiento de imágenes y otro servidor para que ejecute operaciones con estos valores. En la figura 2 se muestra este tipo de arquitectura, pero con varios servidores.

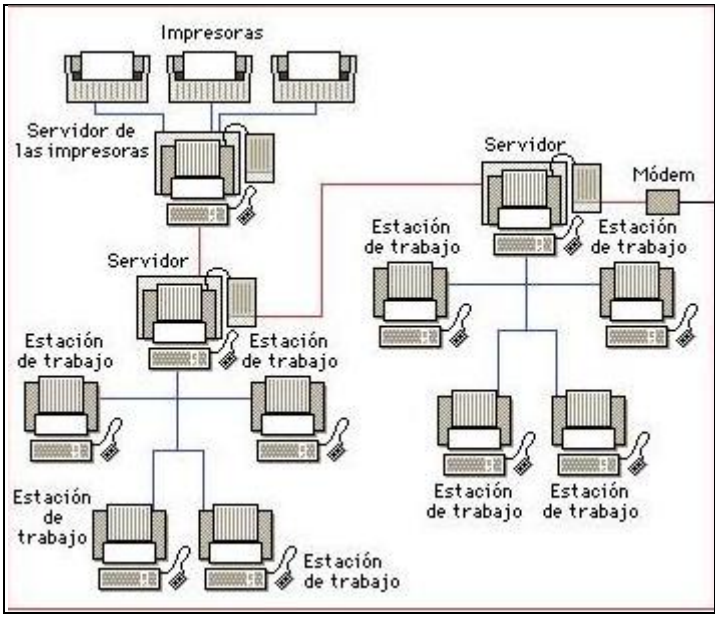


Figura 2. Arquitectura cliente-servidor con tres servidores.

En el caso práctico que se ha realizado se va a disponer de una arquitectura donde sólo hay un cliente y existen dos servidores, como veremos más adelante.

Para programar el funcionamiento de esta arquitectura se van a usar “sockets” en los procesos y además, en diferentes entornos de desarrollo. Respecto al sincronismo, es un punto que afecta a la programación de sockets, aunque afecta a la funcionalidad de la arquitectura. Básicamente este sincronismo se debe de entender como operaciones síncronas o asíncronas en los diferentes dispositivos conectados. Es decir, consideramos operaciones síncronas, aquellas operaciones que son bloqueantes, pues cuando un cliente realiza un envío a un servidor éste está esperando este tipo de envío y se queda bloqueado hasta que lo reciba. También ocurre de la misma manera cuando un servidor envía información a un cliente. Se considera asíncrona cuando estas operaciones son no bloqueantes, de tal forma que un elemento intenta recibir información y, si no se ha enviado nada o la información todavía no se ha preparado, el dispositivo sigue trabajando hasta que tenga información que leer. La forma de recoger esta información, en el presente trabajo, se realiza mediante eventos de notificación de mensajes, ya que, cuando un servidor debe de enviar información al cliente, éste recibe un evento indicando que se tiene información para recibir.

2.1.1 Programación de sockets.

Los sockets proporcionan una comunicación de dos vías, punto a punto entre dos procesos. Los sockets son muy versátiles y son un componente básico de comunicación entre interprocesos e intersistemas. Un socket es un punto final de comunicación al cual se puede asociar un nombre. Existen los siguientes tipos de sockets:

- S*ockets Stream. H*ace uso del protocolo TCP (protocolo de la capa de transporte) que provee un flujo de datos bidireccional, orientado a conexión, secuenciado, sin duplicación de paquetes y libre de errores.
- S*ockets Datagram. H*ace uso del protocolo UDP (protocolo de la capa de transporte), el cual provee un flujo de datos bidireccional, no orientado a conexión, en el cual los paquetes pueden llegar fuera de secuencia, puede haber pérdidas de paquetes o pueden llegar con errores.
- S*ockets Raw. P*ermiten un acceso a más bajo nivel, pudiendo acceder directamente al protocolo IP del nivel de Red. Su uso está mucho más limitado ya que está pensado principalmente para desarrollar nuevos protocolos de comunicación, o para obviar los protocolos del nivel de transporte.

En la arquitectura desarrollada se ha utilizado sockets del tipo Stream o de flujo, que da un flujo de datos de dos vías, fiable, sin duplicados y sin límites de grabación. El flujo opera de forma parecida a una conversación telefónica. El tipo del socket es *SOCK_STREAM*, el cual en el dominio de Internet usa TCP (Transmission Control Protocol), es decir, es “Orientado a Conexión”. La forma de conectarse entre ellos es con un socket y hasta que no esté establecida correctamente la conexión, ninguno de los dos puede transmitir datos. Esta es la parte TCP del protocolo TCP/IP, y garantiza que todos los datos van a llegar de un programa al otro correctamente. Se utiliza cuando la información a transmitir es importante, no se puede perder ningún dato y no importa que los programas se queden "bloqueados" esperando o transmitiendo datos. Si uno de los programas está atareado en otro trabajo y no atiende la comunicación, el otro quedará bloqueado hasta que el primero lea o escriba los datos. En la figura 3 se muestra un esquema básico de funcionamiento de los sockets.

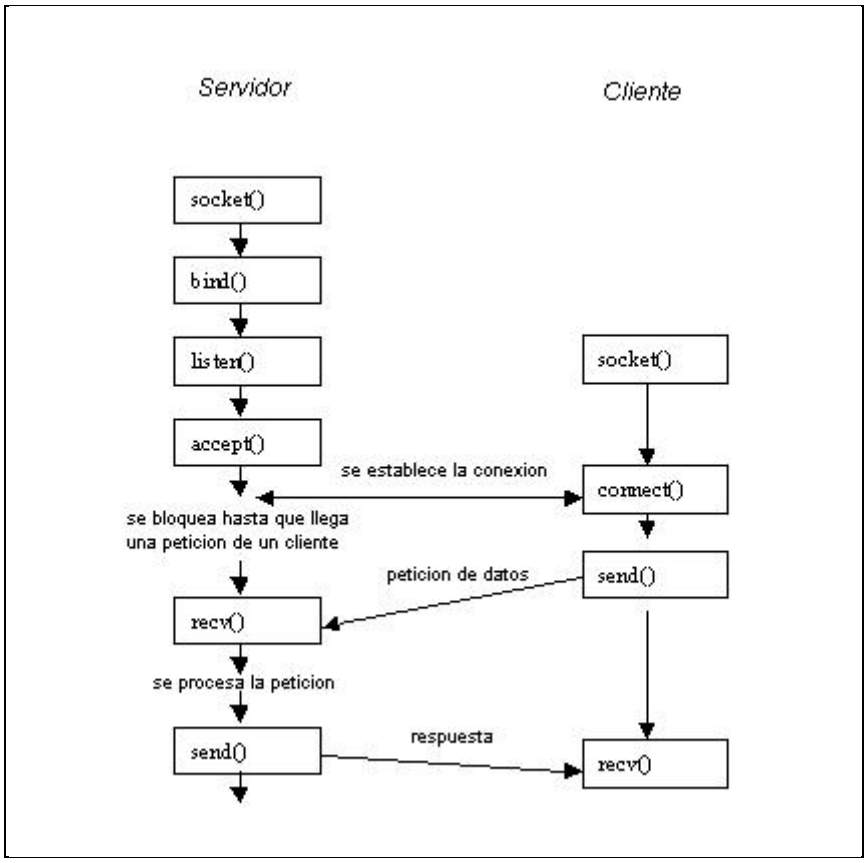


Figura 3. Funcionamiento de los sockets

Cabe destacar que todos los desarrollos que se han realizado han sido con el lenguaje C. A continuación se resumen las funciones utilizadas en los sockets (en función del lenguaje empleado estas funciones pueden variar ligeramente):

- **Socket**(). Los sockets se crean con esta función y devuelve el identificador de socket (sockfd) o -1 si se ha producido algún error.
- **Bind().** Esta función sirve para darle un nombre al socket, es decir, una dirección IP y un número de puerto de la máquina local para enviar y recibir los datos.
- **Listen().** Esta función se invoca únicamente en el servidor y habilita el socket para poder recibir conexiones. Esta llamada solo es necesaria cuando se usan sockets del tipo Stream (SOCK_STREAM).
- **Accept().** Esta función también es utilizada en el servidor una vez se ha invocado a la función listen(). Esta función espera a que algún cliente establezca conexión con el servidor. Es una llamada bloqueante, es decir, la función no finaliza hasta que se realice alguna conexión o sea interrumpida por alguna señal.
- **Connect().** Esta función inicia la conexión con el servidor.
- **Send().** Una vez ya está establecida la conexión con el servidor se pueden enviar datos mediante esta función. La puede utilizar tanto el cliente como el servidor.
- **Recv().** Esta función se utiliza para la recepción de los datos. Esta llamada es bloqueante, por lo que la función no finaliza hasta que se reciban los datos enviados.

2.1.2 Ejemplos de uso de la arquitectura cliente-servidor

Existen multitud de situaciones donde es necesaria la implementación de una arquitectura cliente-servidor, ya sea con varios clientes, varios servidores o ambas cosas. Además, el sincronismo de funcionamiento de esta arquitectura también va a estar condicionada por las tareas a desarrollar. A continuación se exponen varios ejemplos ilustrativos en diferentes entornos donde se pueden aplicar este tipo de arquitectura en base a un funcionamiento asíncrono.

2.1.2.1 Robot "recogedor".

Supongamos una estructura como la mostrada en la figura 4.

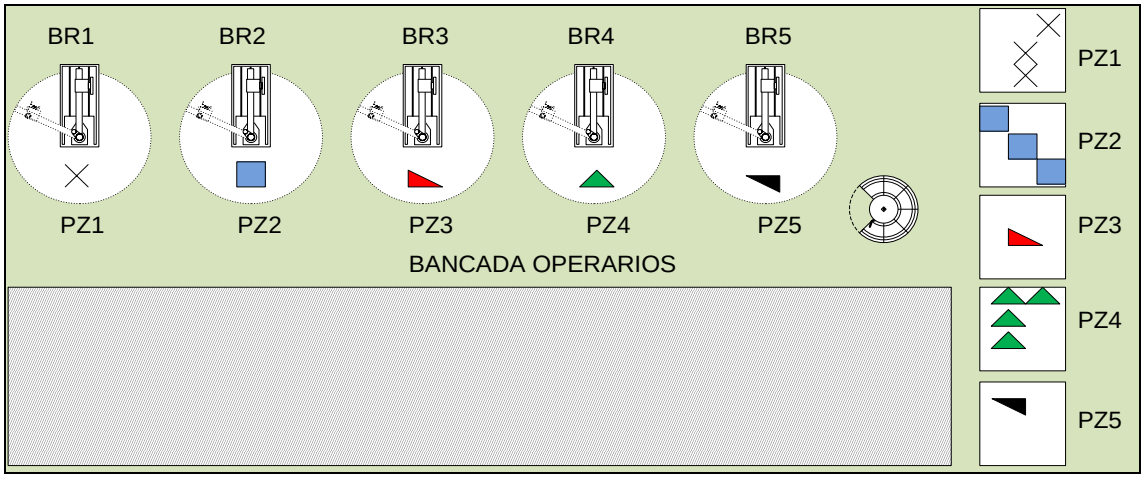


Figura 4. Robot multitarea recogedor de objetos.

En este entorno disponemos de 5 brazos robóticos encargados de terminar 5 piezas diferentes. Debido al espacio reducido de este almacén no es posible instalar una cinta transportadora para colocar los objetos en sus respectivas cajas. Cada robot sólo puede terminar una pieza en concreto. Por ejemplo, en la imagen 4 el brazo robótico BR1 sólo puede terminar piezas PZ1 y estas deben de colocarse en la primera caja, situada en la parte superior derecha de la imagen. Así sucesivamente con cada uno de los robots y piezas. La terminación de cada pieza depende de la complejidad de finalización de ésta. Así, el tiempo empleado para finalizar la pieza 1 es de 20 minutos aproximadamente y el resto de piezas 5 minutos más respecto a la primera pieza (aproximadamente). Lógicamente un brazo robótico no puede empezar la siguiente pieza hasta que el robot recogedor no haya recogido la pieza actual que ha acabado. El robot recogedor es un robot móvil, que lleva incorporado un pequeño brazo robótico para poder coger las piezas que los brazos robóticos van finalizando.

Una posible solución al problema planteado es enviar a este robot móvil a la posición indicada para recoger una pieza cuando uno de los 5 robots envíe un mensaje indicando que ha finalizado la pieza. En caso de que se reciban varios mensajes simultáneamente se atienden en el orden de llegada. Si existiesen cuellos de botella debido a que varios robots están parados mucho tiempo, se puede añadir un segundo robot móvil para evitar esta parada de producción y así poder dividir el trabajo entre los dos robots. Claramente esta situación se

puede resolver con una arquitectura cliente servidor, de tal manera que el robot móvil es el servidor y los cinco brazos robóticos son los clientes. Como el robot no sabe qué brazo va a terminar antes (esto es impredecible, puesto que unos van más rápidos que otros y cualquier suposición de orden es una idea equivocada) no se pueden realizar recepciones bloqueantes, ya que no existe un orden de recepción. Para ello, como se ha indicado, se usa una recepción asíncrona, conforme los clientes van acabando las piezas. Este asincronismo, como se mencionó en la introducción, puede ser programado mediante eventos de notificación por mensajes, por parte del servidor.

2.1.2.2 Limpieza inteligente

Supongamos ahora que tenemos un edificio con varias plantas, donde cada planta tiene un aspecto similar al mostrador en la figura 5.

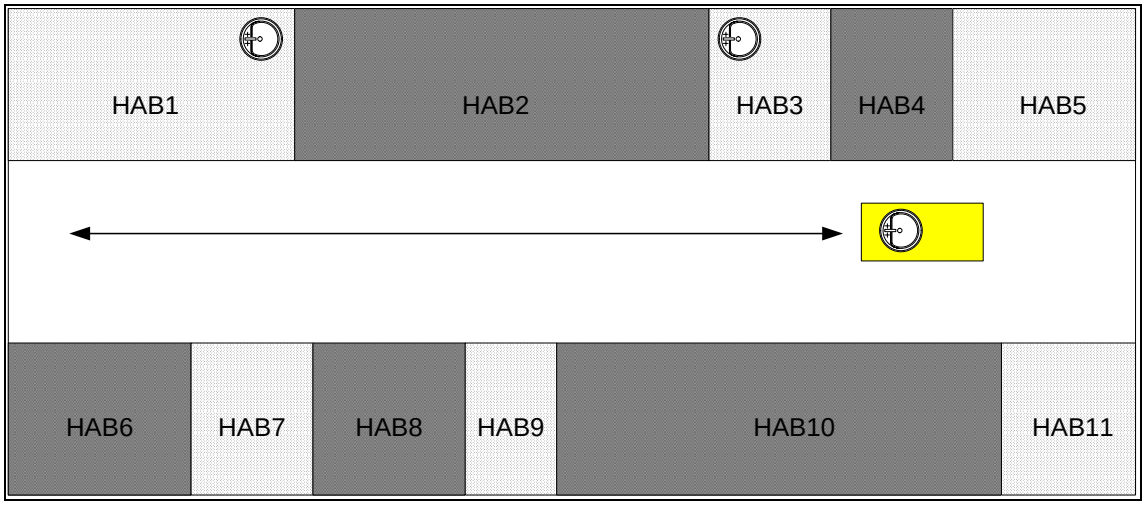


Figura 5. Plantas de un edificio.

El objetivo es limpiar (en este caso aspirar) todo el edifico en tres o cuatro días diferentes, de tal forma que se realice esta tarea cuando menos personal existe en el edificio. Para ello, se dispone de tres robots móviles que disponen de todos los elementos necesarios para poder realizar esta tarea. Por otro lado, disponemos de otro robot móvil, que es el encargado de transportar los diferentes robots a las diferentes áreas. El problema, no es simple, pues existen varias plantas en el edificio. Como podemos observar en este ejemplo, las tareas de cada robot se realizan de forma independiente; en cambio, es necesaria programar una comunicación cliente-servidor de forma asíncrona para que cada robot avise al robot transportador para que lo recoja y lo lleve a otra habitación. El robot transportador es el servidor y el resto de robots son los clientes. Lógicamente, la tarea de recoger los robots es asíncrona pues, en ningún momento podemos saber el orden de finalización de limpieza de los robots. Es decir, si disponemos de tres robots e hiciéramos tres recepciones mediante sockets, uno para cada robot, el programa se quedaría bloqueado hasta que termine de limpiar exactamente el robot del que estamos haciendo la recepción. Esto es completamente ineficaz, ya que pueden terminar los otros dos robots su tarea y el transportador no recibir la información enviada de que debe recogerlos, pues el programa estaría bloqueado esperando la primera recepción.

2.1.2.3 Coordinación de robots móviles con diferente conectividad.

En la imagen 6 se muestra una arquitectura de conexión de varios robots con diferentes PCs.

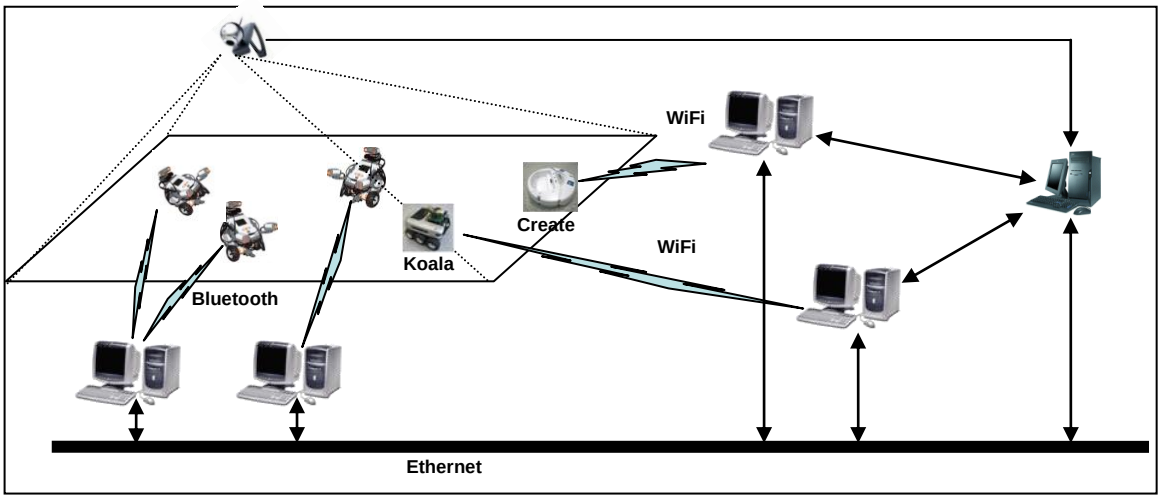


Figura 6. Arquitectura conjunta

En esta arquitectura tenemos un PC servidor que recoge las imágenes de la cámara, las procesa y son enviadas a los PCs que están conectados con el Create de IRobot y el Koala cuando sea solicitado por la red Ethernet. Estos PCs actúan como clientes, en el sentido de que solicita el procesamiento de la imagen al PC servidor cuando así se pida a través de la red (Ethernet). Además, también son clientes que solicitan a los servidores correspondientes (cada PC va conectado de forma inalámbrica a un robot, uno al Koala y otro al Create) la realización de alguna tarea conjunta o independiente.

Hay que tener en cuenta que en esta arquitectura tenemos varias redes formadas. El PC conectado con la cámara y los PCs que están conectados con el Koala y el Create forman una red. Por otro lado, estos dos PCs conectados a ambos robots forman otra red diferente y, finalmente tenemos el resto de redes, que en este caso son dos PCs que se conectan de forma inalámbrica (mediante Bluetooth) a los robots Lego. Adicionalmente, se pueden poner más PCs de tipo cliente, que no estén ubicados físicamente en el mismo lugar. Por ejemplo, se puede tener un PC en el hogar que actúe como cliente y, a través de una VPN se conecta a la red Ethernet de esta arquitectura conjunta para solicitar peticiones al servidor que está conectado con la cámara.

Con esta arquitectura se puede realizar infinidad de trabajos. Por citar algún ejemplo, el Create puede perseguir al Koala en tiempo real mientras que los Lego rodean al Create haciendo un círculo. Esta tarea es bastante compleja, pues el tiempo de procesamiento de la imagen debe de ser el menor posible, ya que si no los Lego podrían chocar con el robot que rodean.

Como podemos observar, existen diversos entornos de trabajo en el que, con una arquitectura cliente-servidor asíncrona puede resolverse tareas que, de forma síncrona sería imposible diseñar o, al menos, serían ineficaces en aspectos de tiempo y porcentaje de utilización de los robots.

2.2 Procesamiento de imágenes.

Otro aspecto fundamental en el desarrollo del presente trabajo es el tratamiento de imágenes. Aunque el objetivo principal es el estudio de la arquitectura planteada de forma asíncrona, no se debe olvidar que esta arquitectura se ha desarrollado en un ámbito con varios robots para recoger una pelota de forma coordinada. Lógicamente, es necesario disponer de algún tipo de cámara para procesar las imágenes y así obtener posiciones iniciales, orientaciones, posición final, etc.

Se ha utilizado una cámara cenital, más concretamente el modelo "QuickCam Pro 3000" de Logitech. El aspecto de esta cámara se muestra en la imagen 7.

Figura 7. Cámara QuickCam Pro 3000 de Logitech.

Después de instalar la cámara en el techo del laboratorio, el siguiente trabajo que hay que realizar es la calibración de ésta. La tarea de calibración es algo tediosa pero necesaria para poder realizar transformaciones de píxeles a coordenadas x, y y orientación. Esto implica la necesidad de emplear un tiempo considerable para esta calibración y así obtener transformaciones de píxeles a coordenadas exactas o, en su caso, muy aproximadas.

Una vez se tiene la cámara calibrada, ya se puede utilizar para obtener imágenes y así poder procesarlas. En las siguientes secciones se explican con mayor detalle todos estos puntos.

2.2.1 Calibración de la cámara.

Lo primero que debemos de hacer es delimitar el área de visión que la cámara detecta. Por ejemplo, podemos delimitar las esquinas del área rectangular para ver visualmente cuales son los límites de esta área y así tener marcado el área de trabajo de ésta.

Existen varias técnicas para realizar la calibración. En este caso se ha empleado la técnica de mínimos cuadrados, ya que es una técnica sencilla y proporciona muy buenos resultados. En el siguiente punto se explica detalladamente en qué consiste esta técnica y en secciones posteriores se exponen brevemente los resultados obtenidos.

2.2.1.1 Calibración de la cámara mediante la técnica de mínimos cuadrados.

Como se ha comentado en el punto anterior, el objetivo es obtener la relación que existe entre un pixel de la imagen y las coordenadas X-Y (X-Y-Z) del espacio, partiendo de un patrón de calibración.

El patrón es un conjunto de N objetos (en este caso cuadrados) cuyas coordenadas de los centros son conocidas: [m_{xi}, m_{yi}]. Luego, mediante visión, obtenemos los centros de dicha imagen: [c_{xi} c_{yi}]. Por tanto, la operación de calibración consiste en obtener la matriz (que es lo que se conoce como matriz de calibración) que relaciona un punto de la imagen P_C con el de la posición P_M. En la ecuación (1) se muestra esta relación.

$$P_M = \begin{bmatrix} m_x, m_y, 1 \end{bmatrix} \quad \hat{P}_M = P_C \hat{\theta} = \begin{bmatrix} c_x, c_y, 1 \end{bmatrix} \begin{bmatrix} \hat{\theta}_{11} & \hat{\theta}_{12} & \hat{\theta}_{13} \\ \hat{\theta}_{21} & \hat{\theta}_{22} & \hat{\theta}_{23} \\ \hat{\theta}_{31} & \hat{\theta}_{32} & \hat{\theta}_{33} \end{bmatrix} \tag{1}$$

El error entre P_M y $\hat{P}_M$ debe de ser el menor posible. Para calcular este error de calibración entre la posición real y la estimada aplicamos la ecuación (2).

$$\underbrace{\begin{bmatrix} e_{x_1} & e_{y_1} & 0 \\ e_{x_2} & e_{y_2} & 0 \\ e_{x_3} & e_{y_3} & 0 \\ \vdots & \vdots & \vdots \\ e_{x_N} & e_{y_N} & 0 \end{bmatrix}}_{E} = \underbrace{\begin{bmatrix} m_{x_1} & m_{y_1} & 1 \\ m_{x_2} & m_{y_2} & 1 \\ m_{x_3} & m_{y_3} & 1 \\ \vdots & \vdots & \vdots \\ m_{x_N} & m_{y_N} & 1 \end{bmatrix}}_{M} - \underbrace{\begin{bmatrix} c_{x_1} & c_{y_1} & 1 \\ c_{x_2} & c_{y_2} & 1 \\ c_{x_3} & c_{y_3} & 1 \\ \vdots & \vdots & \vdots \\ c_{x_N} & c_{y_N} & 1 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} \hat{\theta}_{11} & \hat{\theta}_{12} & \hat{\theta}_{13} \\ \hat{\theta}_{21} & \hat{\theta}_{22} & \hat{\theta}_{23} \\ \hat{\theta}_{31} & \hat{\theta}_{32} & \hat{\theta}_{33} \end{bmatrix}}_{\hat{\theta}} \tag{2}$$

Por otro lado, se debe de escoger un índice J, que es el cuadrado del error, tal y como se muestra en la ecuación (3).

$$J = \frac{1}{2} \sum_{i=1}^{N} e_i^2 = \frac{1}{2} E^T E \tag{3}$$

Para minimizar este índice J se debe de calcular la derivada respecto de la matriz de calibración e igualarla a 0, tal y como se muestra en la ecuación (4).

$$\frac{\partial J}{\partial \hat{\theta}} = -C^T \left(M - C\hat{\theta} \right) = 0 \tag{4}$$

Finalmente, despejando la ecuación anterior, se obtiene la matriz Theta, tal y como se muestra en (5).

$$\hat{\theta} = \left(C^T C\right)^{-1} C^T M \tag{5}$$

Esta matriz es la que relaciona el píxel de la imagen con las coordenadas (X,Y,Z).

2.2.1.2 Calibración de la cámara Pro 3000 de Logitech mediante mínimos cuadrados.

Lo primero que se debe de hacer, es colocar los patrones en el área de la imagen, de forma que sepamos la posición X e Y del centro de cada patrón. Se han utilizado un conjunto de cuadrados negros, ya que se diferencian con bastante facilidad del resto de la imagen. En la figura 8 se muestra el aspecto que tienen estos patrones.

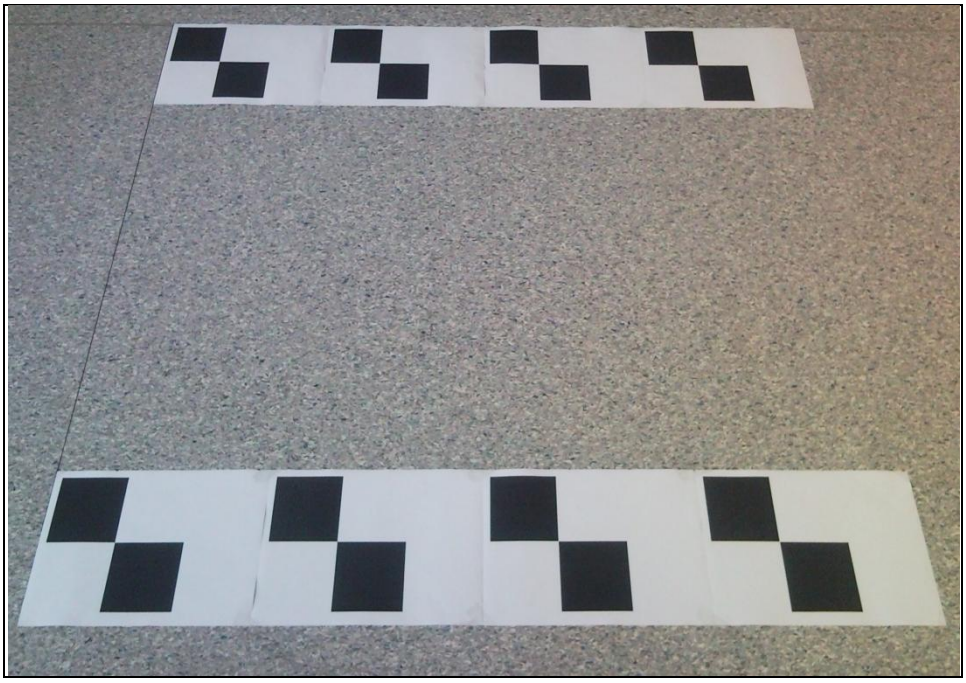


Figura 8. Patrón para la calibración de la cámara

Como podemos observar en la imagen, los cuadrados que se han utilizado como patrón son perfectos y además, sabemos exactamente la posición del centro en mm de cada uno de ellos. Es decir, debemos de establecer un eje de coordenadas. En este caso, la posición 0,0 de la imagen se corresponde con la esquina inferior izquierda y, a partir de ahí, ya sabemos todas las posiciones de los patrones.

Una vez obtenidos los centroides en mm de forma manual de cada patrón, se procede a obtener los centroides de cada uno de estos cuadrados con la cámara. Para ello, se debe de segmentar un total de 8 imágenes, de tal forma que segmentamos inicialmente solo dos cuadrados, tapando el resto de cuadrados, seguidamente otros dos cuadrados, y así sucesivamente hasta que obtenemos las 8 imágenes. De esta forma, podemos obtener los centroides de todos los patrones de la imagen. Además, estos centroides están en píxeles.

Con todos los datos obtenidos ya podemos calcular la matriz de calibración. Esta matriz es la que se necesita para transformar los píxeles a coordenadas reales, tal y como se ha explicado anteriormente. La matriz que contiene todos los centroides en mm (se han obtenido anteriormente de forma manual) es la matriz M, que se muestra en la fórmula (6) y la matriz de centroides calculados con la cámara, es la matriz C y se muestra en la fórmula (7).

$$M = \begin{bmatrix} 148 & 148 & 1 \\ 56 & 56 & 1 \\ 148 & 444 & 1 \\ 56 & 352 & 1 \\ 148 & 740 & 1 \\ 56 & 648 & 1 \\ 148 & 1036 & 1 \\ 56 & 944 & 1 \\ 1058 & 148 & 1 \\ 966 & 56 & 1 \\ 1058 & 444 & 1 \\ 966 & 352 & 1 \\ 1058 & 740 & 1 \\ 966 & 648 & 1 \\ 1058 & 1036 & 1 \\ 966 & 944 & 1 \end{bmatrix} \quad (6)$$

$$C = \begin{bmatrix} centro1X & centro1Y & 1 \\ centro2X & centro2Y & 1 \\ centro3X & centro3Y & 1 \\ centro4X & centro4Y & 1 \\ centro5X & centro5Y & 1 \\ centro6X & centro6Y & 1 \\ centro7X & centro7Y & 1 \\ centro8X & centro8Y & 1 \\ centro9X & centro9Y & 1 \\ centro10X & centro10Y & 1 \\ centro11X & centro11Y & 1 \\ centro12X & centro12Y & 1 \\ centro13X & centro13Y & 1 \\ centro14X & centro14Y & 1 \\ centro15X & centro15Y & 1 \\ centro16X & centro16Y & 1 \end{bmatrix} \quad (7)$$

Con todos estos datos ya podemos calcular la matriz Theta (matriz de calibración) empleando la fórmula (5) vista anteriormente. El tamaño de la matriz es de 3X3 debido a que está normalizada (la última componente es [0 0 1]).

2.2.2 Segmentación de la imagen.

Otro aspecto muy importante es la segmentación de la imagen. Hay que tener en cuenta que en el trabajo desarrollado es necesario detectar las posiciones de los robots y la posición de la pelota, que es la posición de destino. Por eso, es necesario realizar la segmentación de la imagen con una serie de procesos adicionales.

Una vez se ha capturado la imagen, debe de segmentarse en función del área de nuestro interés. Es decir, para distinguir los diferentes elementos en el área de la imagen, lo que se ha utilizados son componentes de diferentes colores. Así, por ejemplo, los obstáculos son de color azul, el Create es de color negro, la pelota a recoger es de color rojo y, finalmente el Koala es de color amarillo. Con toda esta información ya podemos segmentar la imagen con aquellos componentes RGB de interés.

Básicamente, lo que se debe de hacer es segmentar la imagen para obtener la posición y orientación inicial del Create. Cuando hayamos calculado estas posiciones, volvemos a segmentar la imagen inicial para detectar los obstáculos. Esta operación se realiza tantas veces como componentes diferentes tenemos en la imagen.

A continuación, se explica detalladamente la segmentación de la imagen original para calcular la posición y orientación del Create. En el resto de casos se hará de la misma forma, solo que los componentes RGB cambiarán en función del color de interés.

Se parte de la imagen que se muestra en la figura 9.

Figura 9. Segmentación para obtener la posición del robot.

Como podemos observar en la imagen anterior, la posición inicial a detectar es un triángulo isósceles de color negro. El motivo de que sea un triángulo isósceles es para poder detectar la orientación del robot. Por ello, se necesita conocer el punto más alejado del perímetro de este triángulo y el centroide, para obtener la orientación del robot.

El primer paso, es segmentar la imagen para obtener sólo aquellos componentes que se quieren estudiar. En este caso solo queremos detectar el color negro. Para ello, nos quedamos solo con aquellos píxeles que cumplen que la componente R (Red) es menor que 70, G (Green) es menor que 70 y B (Blue) es menor que 70. El resultado de la segmentación se muestra en la figura 10.

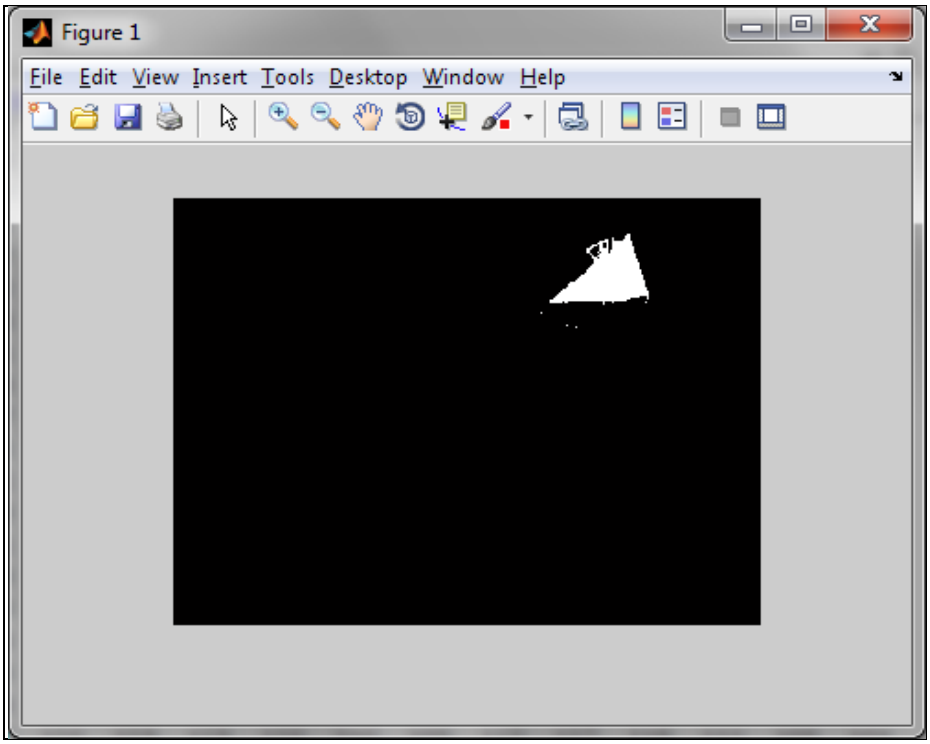


Figura 10. Segmentación de la posición inicial.

Como podemos observar en la imagen, efectivamente se ha detectado el triángulo que se quería. Además, existen píxeles indeseables en la imagen que se deben de eliminar.

El siguiente paso, por tanto, es aplicar un filtro de erosión para eliminar los píxeles aislados y quedarnos solo con los píxeles del triángulo. Este filtro ensancha y realza las zonas claras de la capa activa o selección. Para cada píxel de la imagen, alinea el valor del píxel (luminosidad) con el valor más alto de los 8 circundantes (matriz 3x3). Así se añade un píxel claro sobre áreas claras. Se borrará un píxel aislado en un fondo más claro. Un área clara más grande se dilatará en un píxel en todas las direcciones.

Existen muchos tipos de filtro de erosión y, en este caso se ha aplicado un filtro del tipo “square” o cuadrado con una matriz de 3X3. Es decir, el tipo de filtro indica como es la matriz utilizada para realizar esta erosión. El resultado de aplicar este filtro se muestra en la imagen 11.

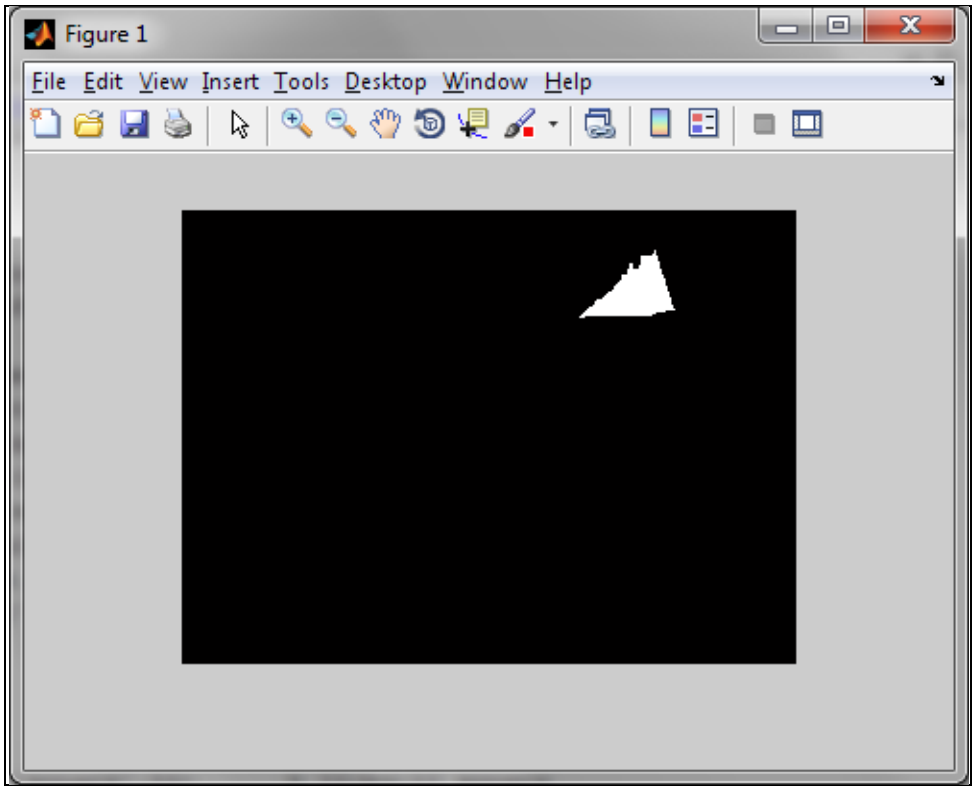


Figura 11. Imagen erosionada

Como podemos ver en la imagen 11 se han eliminado los píxeles aislados de la imagen. Finalmente se debe de aplicar un filtro de dilatación en la imagen anterior. Este filtro amplia y realza las zonas oscuras de la capa activa o selección. Para cada píxel de la imagen, alinea el valor del píxel (luminosidad) con el valor más bajo de los 8 circundantes (matriz 3x3). Así se añade un píxel oscuro en las áreas oscuras. Un píxel aislado en un fondo más claro se cambiará por un gran “pixel”, compuesto por 9 píxeles.

Como en el caso anterior, también existen multitud de filtros de dilatación y, como antes, el filtro que se ha aplicado es del tipo “square”. El resultado se muestra en la imagen 12.

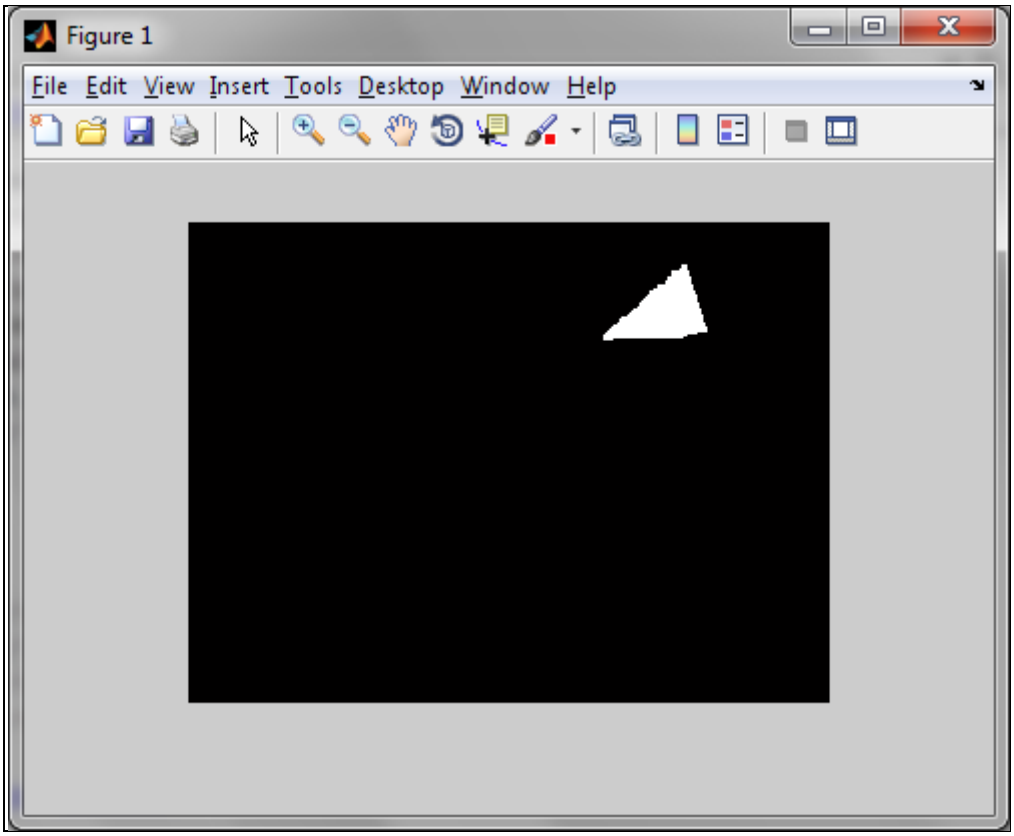


Figura 12. Imagen dilatada

Finalmente, debemos de obtener los objetos que hay en la imagen. Si todo ha ido correctamente, se debe de detectar un único objeto, que es la posición inicial del Create. Una vez se ha realizado esta comprobación, ya podemos calcular el centroide del objeto. Este centroide, por tanto, será la posición inicial X e Y del Create. Además estos valores están en pixeles, por lo que debemos de multiplicarla por la matriz Theta (5) para obtener las coordenadas en mm.

El último paso es obtener la orientación del robot. Para ello calculamos el perímetro del objeto y con los píxeles que se obtienen debemos de estudiar cual es el punto más lejano respecto del centroide. En la imagen 13 se muestra el pixel que se ha detectado en este ejemplo que, efectivamente corresponde con el frontal del robot.

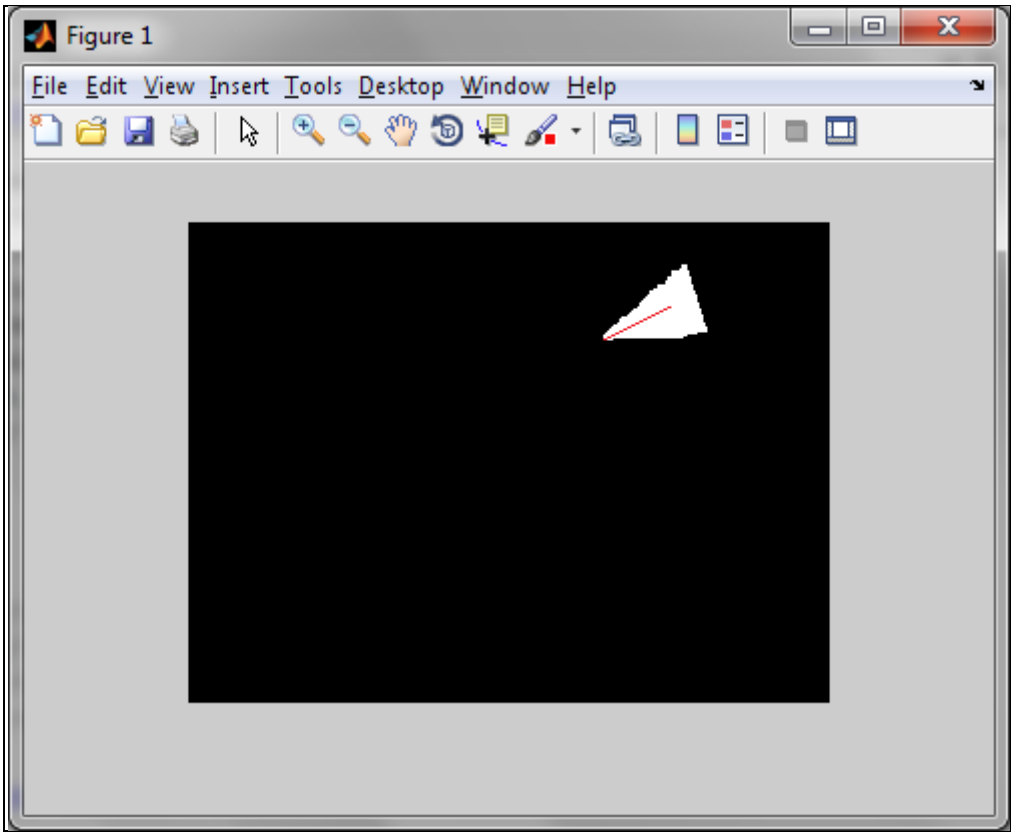


Figura 13. Punto más lejano respecto al centroide del objeto.

Este punto también debemos de transformarlo a mm y obtenemos el arcotangente de la diferencia del punto Y del centroide y la referencia Y del punto más lejano, y la diferencia del punto X del centroide respecto a la referencia X del punto más lejano del perímetro. Este ángulo está en radianes.

Como hemos podido observar, es necesario este tratamiento de las imágenes para poder obtener todo los puntos de interés de la imagen capturada. En el resto de casos, la forma de operar es similar, excepto en la detección de obstáculos y la pelota, pues no se debe de calcular la orientación de estos objetos, ya que no es necesario.

2.2.3 Generación de trayectorias.

Existen multitud de algoritmos para la generación de trayectorias con evitación de obstáculos, tanto estáticos como dinámicos (en movimiento). El método empleado es de descomposición de celdas y, en principio, solo sirve para objetos que no se mueven.

Este método es uno de los más estudiados hasta ahora y consiste en descomponer el espacio en regiones simples llamadas celdas, de tal manera que una ruta entre dos puntos puede ser fácilmente encontrada.

Al tamaño de la celda se le llama RASTER y todas las celdas tienen el mismo tamaño. En el algoritmo empleado simplemente se busca un camino libre de colisiones desde el punto inicial al objetivo, sin realizar ninguna recursividad y explotación de celdas. Es decir, cuando discretizamos la imagen en este conjunto de celdas, ésta se puede clasificar en:

- EMPTY. Si su interior no intersecta con ningún obstáculo.
- FULL. Si está completamente incluida en el interior de un obstáculo.
- MIXED. Ninguna de las dos situaciones anteriores. Contiene un espacio EMPTY y otro espacio FULL.

En la imagen 14 se muestra como queda la imagen después de realizar esta descomposición.

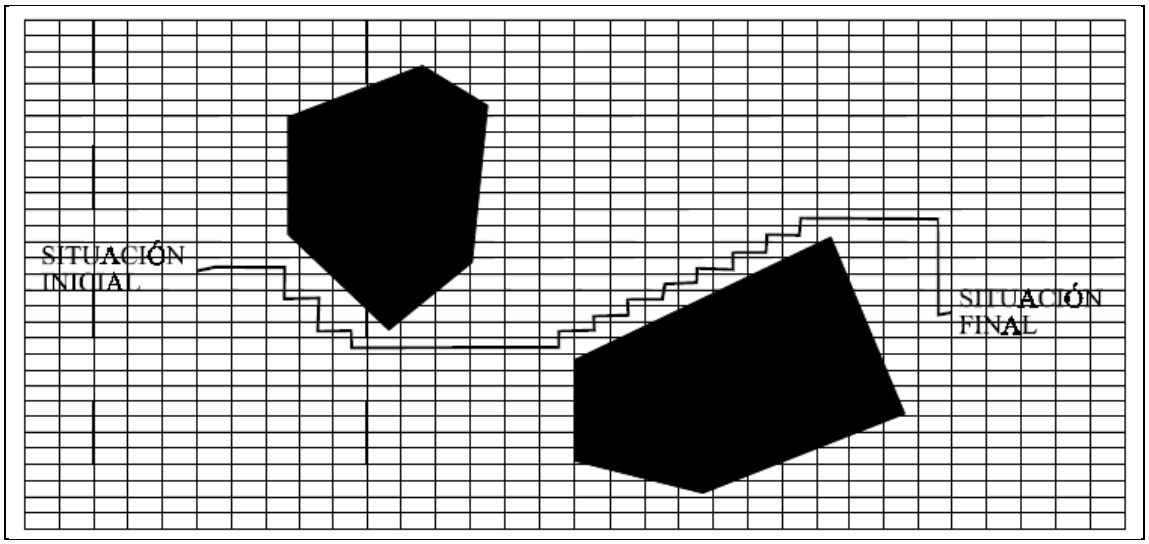


Figura 14. Descomposición de celdas.

Como podemos ver en la imagen anterior, esta descomposición es exacta y no se realiza ninguna explotación de las celdas. Es decir, existe una metodología que consiste en dividir una celda en otras posibles 4 celdas cuando es del tipo MIXED. De esta forma, se consigue una mayor aproximación y exactitud en el camino, pero esta metodología no se ha implementado.

Lo que se hace, básicamente es estudiar los vecinos de una celda para saber si están libres de obstáculos o, por el contrario son del tipo FULL o MIXED. En cualquiera de estas dos opciones se descarta dicha celda, pues produciría una colisión del robot con el obstáculo. Por lo que poco a poco se construye el camino, libre de colisiones, añadiendo estas celdas libres. Se puede hacer varias búsquedas cuando el algoritmo devuelve que no se ha encontrado un camino válido. Por eso, hay que tener en cuenta en poder volver hacia atrás, en la construcción del camino, para explorar alternativas a la trayectoria que se está buscando.

También hay que tener en cuenta que esta trayectoria es una trayectoria cualesquiera y no se asegura que sea la óptima. Para nuestro interés, con encontrar un camino libre de colisiones (en caso de que exista) es suficiente.

En cualquier caso, esta descomposición puede ser representada mediante un grafo de conectividad, de tal forma que cada nodo representa una celda y dos nodos están unidos cuando las celdas son adyacentes. A este grafo solo se insertan aquellos nodos que son EMPTY o MIXED para encontrar el camino libre de colisiones.

Por tanto, la trayectoria que se encuentra lo que hace es unir los centros de los cuadrados, libre de colisiones, desde el punto inicial al destino.

Finalmente, otro aspecto a tener en cuenta es el tamaño del RASTER o celda. Cuando mayor es el tamaño de la celda, menor es la resolución de esta discretización y es posible que no encuentre una trayectoria, aun existiendo ésta. En cambio, si elegimos un RASTER pequeño, es posible que el robot colisione con algún obstáculo por no haber dado un tamaño suficiente a este RASTER. En este sentido, el tamaño del RASTER debe de ser, al menos, el tamaño del robot.

En el algoritmo que se ha usado, al final se ha optado por utiliza un RASTER pequeño para obtener una buena resolución de esta discretización. En cambio, como se ha comentado anteriormente, esto produce problemas debido a que el robot puede colisionar con algún obstáculo. Para solventar este inconveniente, lo que se ha realizado es una dilatación de todos los obstáculos, de tal forma, que se ha dilatado cada obstáculo añadiendo el tamaño del robot. Si pensamos por un momento en esta solución, es completamente válida, pues lo mismo es considerar el tamaño del robot que añadírselo a los obstáculos y, considerar el robot como un punto. Se ha optado por esta opción ya que experimentalmente se han obtenido mejores resultados.

En la imagen 15 se muestra el resultado de aplicar el algoritmo de descomposición de celdas.

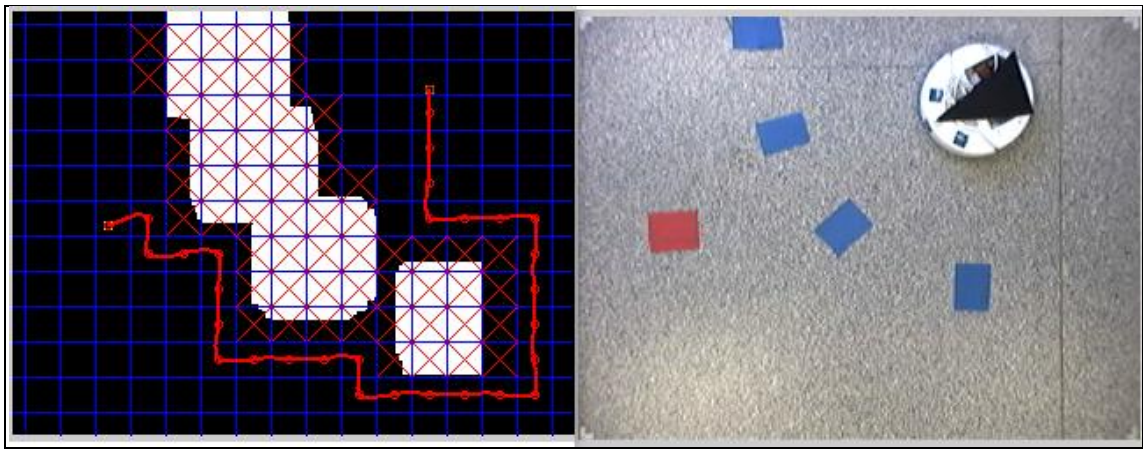


Figura 15. Algoritmo de generación de trayectorias mediante descomposición en celdas.

2.3 Seguimiento de la trayectoria generada.

Otro aspecto muy importante en el presente trabajo es el seguimiento de la trayectoria generada. En este sentido existen infinidad de algoritmos de control de seguimiento de trayectorias y de control de caminos. La diferencia entre trayectoria y camino es que el primero se pude decir que es una curva temporal para cada una de las coordenadas sobre las que se debe realizar un control del robot. Mientras que el camino es una curva definida en el espacio de configuración que debe de seguir el robot sin tener en cuenta el factor tiempo. Es decir, la mayor diferencia entre ambos algoritmos es el tiempo y, en este sentido, para seguir la trayectoria generada libre de colisiones, lógicamente hace falta un algoritmo de control de trayectorias. El algoritmo de control de trayectorias que se ha desarrollado en el presente trabajo es el de "Control de posición por punto descentralizado" usado para seguir la trayectoria libre de colisiones.

2.3.1 Control de posición por "punto descentralizado".

La idea es establecer el control a partir de la posición y velocidad de un punto que está separado una distancia e del eje de tracción del robot, por lo que la posición x_p e y_p de este punto se establece como se muestra en la ecuación (8).

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} x_m + e\cos\theta \\ y_m + e\sin\theta \end{bmatrix} \tag{8}$$

En la imagen 16 se muestra la configuración de estos puntos.

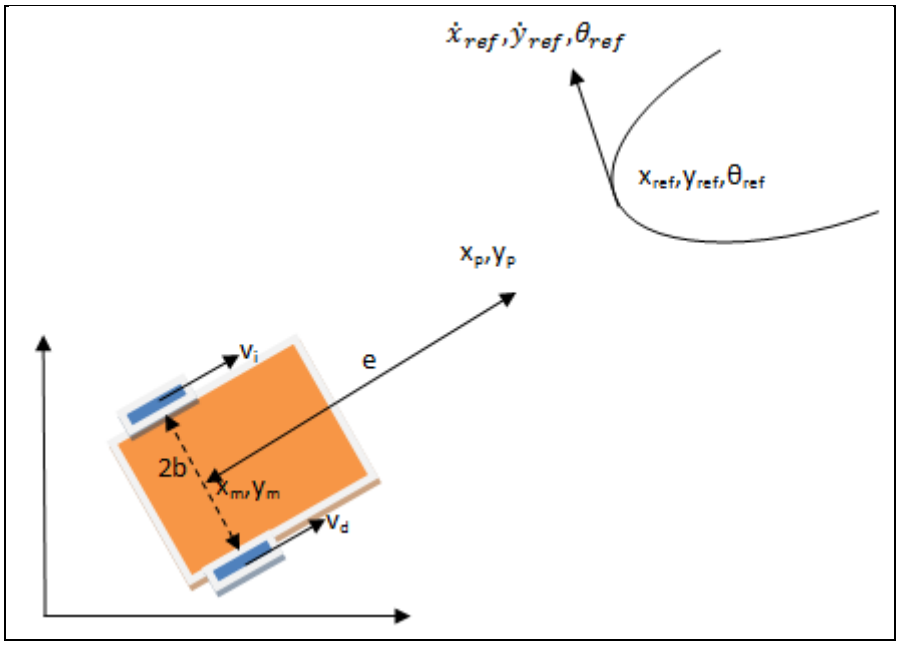


Figura 16. Esquema del punto descentralizado.

Como podemos observar en la imagen anterior, "e" es la distancia del eje del robot al punto descentralizado y "b" es la distancia de la rueda al punto central del eje del robot.

Para calcular las posiciones x_m e y_m del robot lo hacemos directamente con la distancia que ha recorrido el robot. En todo el algoritmo se emplea como unidad de medida el milímetro y para las velocidades, milímetros / segundo. En las ecuaciones 9, 10 y 11 se muestran como calcular las posiciones x_m e y_m, así como la orientación estimada de robot en radianes, respectivamente.

$$X_m = X_m + (\text{Distancia} * \cos(\theta) \tag{9}$$

$$y_m = y_m + (\text{Distancia} * \sin(\theta) \tag{10}$$

$$\theta = \theta + \frac{V_d - V_i}{2b} * T \tag{11}$$

Para calcular las velocidades de la rueda izquierda y derecha se hace de la forma que se muestra en la ecuación (12).

$$\begin{bmatrix} v_i \\ v_d \end{bmatrix} = \frac{1}{e}\begin{bmatrix} ecos(\theta) + bsin(\theta) & esin(\theta) - bcos(\theta) \\ ecos(\theta) - bsin(\theta) & esin(\theta) + bcos(\theta) \end{bmatrix}\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} \tag{12}$$

Como ley de control cinemática se ha desarrollado la acción que se muestra en la ecuación 13.

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \begin{bmatrix} k_{vx}\dot{x}_{ref} \\ k_{vy}\dot{y}_{ref} \end{bmatrix} + \begin{bmatrix} k_{px} & 0 \\ 0 & k_{py} \end{bmatrix}\begin{bmatrix} x_{ref} - (x_m + ecos(\theta)) \\ y_{ref} - (y_m + esin(\theta)) \end{bmatrix} \tag{13}$$

La mayor complejidad del algoritmo es encontrar los valores de las ganancias tanto de posición como de velocidad que obtienen resultados óptimos para la ejecución de éste.

En la parte práctica se explica cuales son los valores de todas estas ganancias que obtienen resultados buenos, así como ejemplos de seguimiento de trayectoria que se han obtenido.

3. Desarrollo práctico.

En este apartado se va a explicar detalladamente los experimentos realizados con las diferentes plataformas para comprobar que efectivamente estas arquitecturas son posibles de desarrollar.

Hay que tener en cuenta que la programación del Create de IRobot es algo incómoda, por lo que hace falta que creemos algún tipo de mecanismo que facilite al programador el desarrollo de aplicaciones para éste. Para ello, lo que se ha hecho es una librería que encapsula todas las posibles funciones del robot en una capa superior y el desarrollador no se tenga que preocupar de escribir los códigos de bajo nivel para desarrollar un programa. De esta forma, se tienen llamadas del estilo "Velocidad_Independiente(R_Izq, R_Der), que establece la velocidad en mm/s a cada una de las ruedas del robot. En la sección 3.2 se explica cómo se ha creado esta DLL y como se debe de emplear.

En una segunda parte se explica con detalle como se ha desarrollado esta arquitectura y cómo interactúan los diferentes componentes involucrados. Todo esto, junto con los correspondientes anexos, facilita al lector el entendimiento de esta arquitectura y el funcionamiento de ésta.

Hay que tener en cuenta que aquí se muestran varios ejemplos de funcionalidad, lo cual no quiere decir que cualquier otra estructura o arquitectura, no funcione. Además, en una primera parte se expone un ejemplo de funcionamiento síncrono del Create de IRobot junto con un PC y la cámara cenital para llegar a un punto destino con generación de trayectorias mediante la evitación de obstáculos. En este apartado también se va explicar una aplicación que se ha desarrollado para llevar al Create de forma automática al punto inicial deseado. Esta aplicación se ha desarrollado para un dispositivo móvil, el cual se comunica con Bluetooth con el robot y dirigimos éste como si fuese la conducción de un automóvil. Por otro lado, se explicará una segunda arquitectura en el que intervienen más componentes, donde el funcionamiento es, en parte, asíncrono. Es decir, tal y como se ha explicado en la introducción, existen partes de una funcionalidad de una arquitectura que son claramente asíncronas. La arquitectura desarrollada sólo presenta una parte asíncrona y la otra síncrona. Si se estudia este punto, se puede ver claramente que aún es más complejo, ya que, debemos de controlar cuando la comunicación es síncrona y cuando asíncrona. Se debe de tener claro que en cualquier momento la comunicación puede ser de alguna de estas dos formas pero, en ningún caso, las dos a la vez. Además, siempre debemos de tener controlado esta sincronía, pues la forma de comunicarse los demás elementos está involucrada y es posible que se obtenga un mal funcionamiento si no controlamos perfectamente este sincronismo.

Finalmente, decir que este tipo de comunicación se puede implementar en cualquier elemento, es decir, en los clientes, servidores o cualquier otro componente que permita la programación con sockets para poder comunicarse con el resto de elementos, no necesariamente deben de ser robots móviles.

3.1 Elementos de la arquitectura.

A continuación se explica detalladamente cada uno de los componentes utilizados en este desarrollo práctico.

3.1.1 Create de IRobot.

Create de IRobot es un robot móvil que presenta una gran oportunidad a estudiantes y educadores para desarrollar sus propias aplicaciones referentes a movimientos, sonidos y comportamientos, e incluso integrar al robot con electrónica adicional para así poder realizar comportamientos más complejos.

El Create se basa en la plataforma del Roomba. Este modelo de robot (el Roomba) es utilizado a nivel doméstico para realizar tareas de limpieza, como son aspirar todo el suelo de una vivienda completa, evitando los obstáculos y, en la medida de lo posible, no dejar ninguna zona sin aspirar. El IRobot Create también puede simular este comportamiento, sólo que se ha eliminado los componentes de limpieza. Por tanto, es un robot pensado para realizar aplicaciones y desarrollos de una forma sencilla que, simplemente se basa en la plataforma del Roomba.

En la imagen 17 se muestra un esquema visto desde la parte superior del robot.

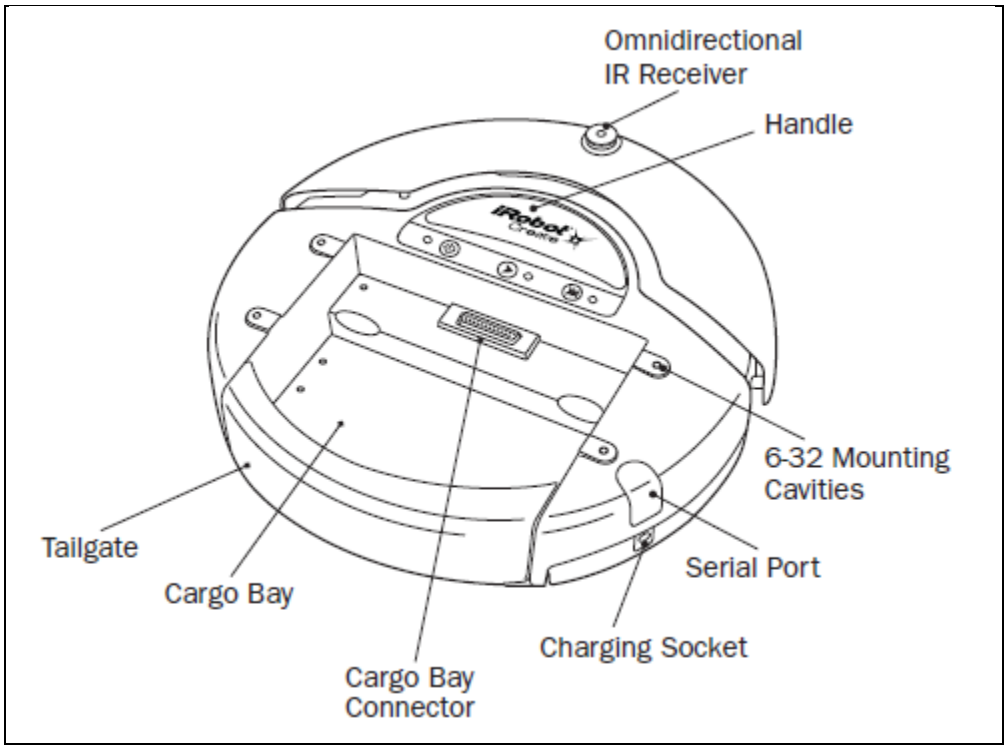


Figura 17. Vista superior del robot Create

Como podemos observar en la imagen, la estructura del robot es bastante simple. Cuenta con un receptor de infrarrojos que es utilizado para no cruzar muros virtuales, creados también con sensores infrarrojos, además de ser utilizado para que el robot encuentre la base y se cargue automáticamente. Por otro lado, dispone una cavidad para poder añadir elementos al robot, como por ejemplo una cámara para que disponga de visión, o un sistema embebido para que el robot tenga autonomía y no sea necesario ningún PC para la programación de éste.

En el "Cargo Bay connector" se puede conectar una antena Bluetooth para comunicarse con un PC o un dispositivo móvil vía Bluetooth o también se puede conectar el "Command Module" que se explicará en secciones posteriores.

Finalmente también disponemos de la conexión con el cable serie para poder enviar y recibir los comandos del robot.

En la imagen 18 se muestra el esquema del robot visto por la parte inferior.

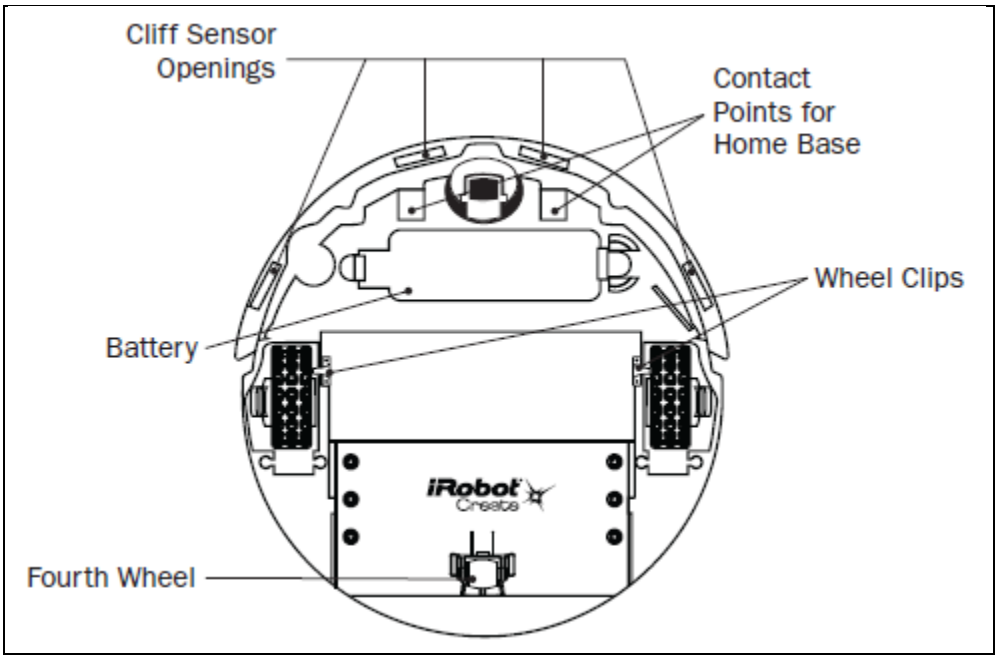


Figura 18. Vista inferior del robot Create.

Como podemos ver en la imagen 17, el robot dispone de 4 sensores ópticos infrarrojos y 2 sensores de choque en la parte delantera para detectar cuando el robot golpea contra algún obstáculo. En el trabajo actual, los sensores de choque o bumper no se utilizan puesto que no tiene sentido evitar obstáculos cuando ya hemos chocado con ellos. En cambio, este tipo de sensores sí que son muy utilizados, como se ha mencionado anteriormente, para tareas de limpieza, de tal forma que cuando el robot choca contra un muro, silla, mesa, etc., sea capaz de bordearlo con la ayuda de estos 2 sensores y no deje ninguna zona sin limpiar.

Para cualquier consulta o aclaración de la arquitectura del Robot se puede consultar la página web: http://store.irobot.com/shop/index.jsp?categoryId=3311368 o el manual adjunto "Manual Final.pdf".

En las siguientes secciones se va a explicar con detalle los sensores de los que dispone el robot, así como diferentes componentes adicionales que podemos adquirir para nuestro robot.

Otra parte fundamental que se verá con detalle es la forma de programar el robot. Aunque, no sea excesivamente complejo, si qué es cierto que disponemos de varias metodologías para interactuar con el robot; por lo que se explicarán cada una de ellas y, en algunos casos, se pondrá algún ejemplo de programación.

3.1.1.1 Sensores.

A continuación se va a explicar brevemente los sensores de los que disponemos en el robot. En cualquier caso, si es necesario instalar cualquier sensor adicional, se puede realizar sin más, puesto que disponemos de hasta 30 conexiones adicionales para añadir cualquier hardware al robot.

3.1.1.1.1 Sensores de choque.

Como se ha comentado anteriormente, el robot dispone de 2 sensores en la parte delantera, también conocidos como “bumper”. El funcionamiento de éstos es muy simple: se trata de un pulsador, que en estado de reposo (no hay choque) el sensor devuelve un 0 lógico. Por el contrario, cuando hay choque, este pulsador es presionado y cuando se consulta el estado de este dispositivo nos devuelve un 1 lógico para indicar la existencia del choque. Cuando ya no hay contacto con ningún obstáculo, el sensor vuelve a su posición inicial con la ayuda de un muelle, por lo que ya no se recibe el “1” lógico del sensor y se recibe un “0” lógico. Cuando queremos obtener el estado de un sensor simplemente enviamos un comando consulta al robot, bien por el cable serie o bien por la conexión bluetooth.

3.1.1.1.2 Sensores Cliff (Infrarrojos).

Otro grupo de sensores que tiene el robot son los sensores ópticos infrarrojos. Cada uno consta de un fotodiodo y un fototransistor para saber si existe peligro de caída. Es decir, tenemos 4 sensores frontales que apuntan hacia el suelo y se usan únicamente para saber si existe un desnivel y así, por ejemplo, poder parar el robot. La forma de consultar el estado de estos sensores es como en el caso anterior, enviando un código para recibir cuál es el estado de estos 4 sensores.

3.1.1.1.3 Encoders.

Las dos ruedas del robot disponen de dos encoders para saber cuánto se ha desplazado el robot, ya sea un desplazamiento lineal o angular. Es un transductor rotativo que transforma un movimiento angular en una serie de impulsos digitales.

Por otro lado, la forma de trabajar con estos encoders es algo limitada. Es decir, lo único que podemos consultar al robot es la distancia que ha recorrido y el ángulo que ha girado. De ninguna forma podemos acceder a los pulsos individuales de cada rueda. Esta tarea, se podrá realizar en posteriores versiones del robot. Por lo tanto, cuando realizamos una consulta de la distancia recorrida, el robot devuelve esta distancia (en milímetros) como la

suma de las distancias recorridas por cada rueda dividida entre dos. Cuando consultamos el ángulo la forma de trabajar será similar. También hay que tener en cuenta que cada vez que realizamos una consulta de estos valores, los contadores serán inicializados para posteriores consultas, por lo que, esto será un aspecto crucial a tener en cuenta en la programación, ya que de alguna manera hay que acumular los valores consultados para no perder en ningún momento la referencia del robot.

Finalmente, después de realizar varios experimentos con el robot, se puede decir que los encoders que utiliza funcionan razonablemente bien en lo que respecta a la distancia recorrida. En cambio, experimentos realizados para consultar los grados que el robot ha girado en un sentido y en otro, se obtiene errores considerables de estos giros. Por ejemplo, si enviamos un comando para que el robot gire 90^o, cuando hacemos la petición para saber cuánto ha girado el robot, obtenemos valores de 88^o-92^o. Estos valores son inadmisibles, puesto que se realizan muchas consultas a lo largo de la ejecución del programa de trayectorias. Este error de 2 grados es para una consulta. Si, por ejemplo, hacemos 10 consultas el error que estamos cometiendo es de 20^o, que, como observamos es un error muy grande. Por lo que únicamente usaremos el encoder para consultas de desplazamiento lineal y, el desplazamiento angular será estimado a través de otros mecanismos.

3.1.1.1.4 Sensores de caída.

El robot también dispone de sensores de caída en cada una de las ruedas. Estos sensores, simplemente sirven para saber si alguna de las ruedas está suspendida en el aire o, por el contrario, está en contacto con el suelo. Estos tipos de sensores son muy utilizados como medida de seguridad, de tal forma que, cuando se detecta que alguna rueda no está en contacto con el suelo, el robot se para automáticamente para evitar posibles daños en el entorno y daños del propio robot.

3.1.1.2 Actuadores.

Además de los sensores, también es necesario disponer de actuadores, que son los componentes del robot donde podemos modificar su comportamiento.

El Create de IRobot dispone de los dos motores que mueven las dos ruedas de alimentación. Básicamente, la forma de trabajar con estos motores es muy simple. Existen dos formas de interactuar con estos actuadores:

- Enviar la velocidad (en mm / s) de forma independiente a cada rueda. Con un simple comando, enviamos al robot cuales son las velocidades que deben de tener cada rueda, de forma independiente.
- Enviar la velocidad y el ángulo deseado. Se envía cual es la velocidad total lineal que se desea (en mm / s) y qué ángulo que debe de girar respecto del centro del robot. Este ángulo se indica en mm.

En cualquier caso, por simplicidad de programación y de cálculos en los algoritmos, se usa el primer método. Es decir, todos los cálculos que se realizan son para obtener las velocidades de la rueda izquierda y derecha de forma independiente, por lo que la primera opción es la que se adapta a nuestras necesidades.

Además, el robot también dispone de una rueda delantera para conseguir una mayor estabilidad y, de forma opcional, se puede instalar una rueda trasera para que, en caso de incorporar componentes adicionales de un peso considerable encima del robot, no se pierda la estabilidad. Estos dos tipos de ruedas son giratorias y no son del tipo actuador, simplemente se utilizan para conseguir estabilidad.

Por lo que los únicos actuadores que va a disponer el robot son los de las ruedas de alimentación. En cualquier caso, siempre se pueden añadir mecanismos al robot que también sean actuadores, como por ejemplo, un pequeño brazo robótico para cualquier otro fin. No existen problemas de conexión, pues dispone de una gran conectividad para componentes adicionales.

3.1.1.3 Componentes adicionales.

En los siguientes apartados se va a explicar con detalle algunos componentes adicionales que podemos adquirir para el robot.

3.1.1.3.1 Command Module.

Este robot, como desventaja, no dispone de ningún hardware para poder realizar un programa residente en el robot y que trabaje de forma independiente. Como mucho, se pueden realizar "scripts" que sí permanecen en el robot, pero como máximo son de 100 bytes. Para ello, se puede adquirir el command module. Este módulo permite realizar programas en C y C++ que residen en dicho módulo, ya que dispone del microprocesador "Atmel AVR ATmega168", entre otros componentes. En la figura 19 se muestra la apariencia que tiene el "command module".

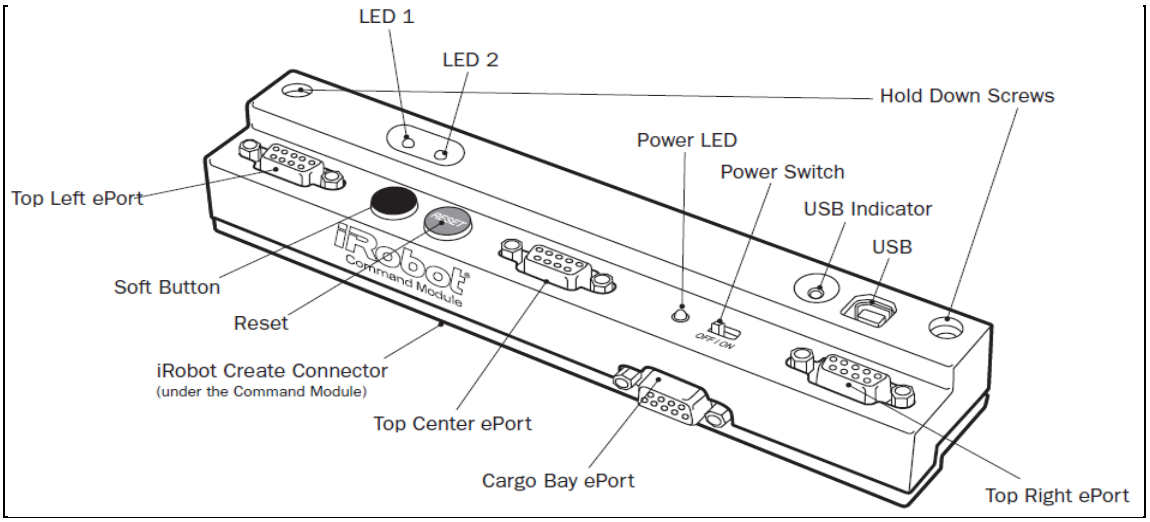


Figura 19. Command Module.

Como podemos observar en la imagen anterior, el command module además de permitir realizar programas en C y C++ para que el robot tenga la autonomía suficiente, dispone de varios conectores y puertos para aumentar la conectividad del robot.

3.1.1.3.2 Bluetooth Adapter Module (BAM).

Otro dispositivo que podemos adquirir para el robot es una antena bluetooth que se conecta en el "cargo bay connector" de éste. De esta forma, podemos realizar la comunicación con el robot de forma inalámbrica. Es decir, lo único que se hace es crear un puerto serie virtual que funciona como si fuese un puerto serie normal y corriente, sólo que éste funciona de forma inalámbrica gracias a este dispositivo. Además, dispone de una serie de conectores para recibir información de ciertos pines del robot o para conectar más actuadores. En la figura 20 se muestra el aspecto que tiene este dispositivo.

Figura 20. Bluetooth Adapter Module.

3.1.1.4 Programación del IRobot Create.

La forma de programar el robot es mediante el envío y recepción de una serie de comandos por el puerto serie o mediante scripts que, también es un conjunto de comandos con un tamaño máximo de 100 bytes. Antes de explicar los diferentes comandos se debe de configurar el puerto serie con las siguientes opciones:

- Baudios: 57600 BPS.
- Bits de datos: 8
- Paridad: No.
- Control de flujo: No.

Esta configuración es necesaria tanto para la comunicación con cable serie como con la antena bluetooth. En el último caso, la configuración del puerto se realiza por programación.

3.1.1.4.1 Programación mediante comandos.

Tal y como se ha mencionado anteriormente, la forma de programar el robot es bien sencilla: se envía un comando al robot para actuar sobre un actuador o para indicar que se quiere recibir datos de algún sensor. Cuando indicamos que se quieren recibir datos de algún sensor, en función de los datos que se piden, estarán empaquetados de manera diferente. Además, conforme se envía el comando solicitando el estado de algún sensor, esta información se recibe posteriormente al envío de la solicitud. Esto es un aspecto a tener en cuenta pues, no se pueden dejar datos en el buffer en ningún momento, ya que si no, la información obtenida sería incorrecta. Además, todos los datos que se reciben o envían al robot son de un tamaño mínimo de 1 byte. Es decir, si solo es necesario el uso de 1 bit para indicar cualquier dato, el tamaño mínimo siempre será de 1 byte.

A continuación se resumen los comandos más utilizados del robot.

3.1.1.4.1.1 Comandos de Inicio.

Estos comandos dejan al robot en modo OI para que esté listo para poder recibir comandos. Es decir, listo para poder enviar y recibir información. Estos comandos son:

- [128]. Este comando siempre hay que enviarlo al robot para que se ponga en modo programación y poder enviar / recibir información.
- [129][Baud Code]. Indica la velocidad en BPS. “Baud Code” puede tomar los siguientes valores:
  - 0 → Indica 300 BPS.
  - 1 → Indica 600 BPS.
  - 2 → Indica 1200 BPS.
  - 3 → Indica 2400 BPS.
  - 4 → Indica 4800 BPS.
  - 5 → Indica 9600 BPS.
  - 6 → Indica 14400 BPS.
  - 7 → Indica 19200 BPS.
  - 8 → Indica 28800 BPS.
  - 9 → Indica 38400 BPS.
  - 10 → Indica 57600 BPS.
  - 11 → Indica 115200 BPS.

3.1.1.4.1.2 Comandos de Modo.

Estos comandos cambian el modo de operar del robot. Son los siguientes:

- [131]. El robot se pone en “modo seguro”. De esta forma se tiene el control del robot pero, se detecta si existe un desnivel cuando el robot avanza o una caída de alguna de las dos ruedas, en cuyo caso el robot se para por seguridad.
- [132]. El robot entra en “modo completo”. De esta forma se tiene el control completo sobre el robot. Es decir, es responsabilidad del usuario el detectar caídas o desniveles mientras se trabaja con el robot.

3.1.1.4.1.3 Comandos de Actuadores.

Son aquellos comandos que operan sobre los actuadores como por ejemplo las ruedas. Se resumen a continuación:

- [137] [Velocidad byte alto] [Velocidad byte bajo][Radio byte alto] [Radio byte bajo]. Sirve apara controlar las ruedas del robot. Los dos primeros bytes indican la velocidad deseada en mm /s y los dos siguiente bytes el radio de giro en mm. El radio de giro es medido desde el centro de giro hasta el centro del robot. Un radio positivo hace girar al robot en sentido anti-horario y un radio negativo en sentido horario. La velocidad máximas del robot son de -500 mm/s y 500 mm/s. mientras que el radio es de -2000 mm y 2000 mm. Existen tres casos especiales del radio:
  - 8000 (hex): Avanzar recto sin giro.
  - FFFF (hex): Girar hacia la derecha sobre sí mismo.
  - 0001 (hex): Gira hacia la izquierda sobre sí mismo.
- [145] [Velocidad derecha byte alto][Velocidad derecha byte bajo] [Velocidad izquierda byte alto][Velocidad byte bajo]. Indica las velocidades de las ruedas de forma independiente. Las velocidades máximas están limitadas al mismo valor que en el caso anterior.
- [147] [Bits de salida]. Se indica un byte (únicamente se usan los últimos 3 bits) para indicar el estado de las 3 salidas digitales del "Cargo Bay Connector". El bit 0 es el pin 19; el bit 1 es el pin 7 y el bit 2 es el pin 20. Un 0 indica 0 V. y un 1 lógico indica 5 V.

3.1.1.4.1.4 Comandos de entrada.

Son comandos que obtienen información del robot, ya sea de sus sensores o de elementos externos conectados. Son los siguientes:

- [142] [ID del Paquete]. Recibimos los datos de los sensores. El ID del paquete indica de qué sensores queremos coger los datos. Tenemos hasta 43 sensores (del 0 al 42).
- [149] [Número de paquetes] [ID Paquete 1] [Packet ID 2]...[Packet ID N]. Es el mismo caso que en el anterior sólo que con una lista. Es decir, si indicamos que N = 2, ID 1 = 20, ID 2 = 24 esperamos recoger el valor de dos sensores, concretamente el del ángulo (20) y el de la temperatura de la batería (24).
- [148] [Número de paquetes] [ID Paquete 1] [Packet ID 2] … [PacketID N]. Es como en el caso anterior pero este método es mejor para obtener una respuesta rápida. Es decir, con el comando 149 obtenemos los valores de los sensores de forma secuencial, mientras que con este último comando llega todo en el mismo paquete y, además se realiza una comprobación mediante checksum.
- [150][byte de solicitud]. Sirve para pausar o reanudar el flujo de la lista de paquetes (comando 148) sin que sea borrado. Un 0 indica que se detenga mientras que un 1 que se reanude.

Los identificadores de los sensores son los siguientes:

- ID = 7. Sensores de choque y de caída. Se recibe un byte con la siguientes estructura:
    - Bit 0. Sensor de golpe derecho
    - Bit 1. Sensor de golpe izquierdo.
    - Bit 2. Sensor de caída de la rueda derecha.
    - Bit 3. Sensor de caída de la rueda izquierda.
    - Bits 4 – 7. En desuso.

  Un 1 indica que hay golpe o caída mientras que si se recibe un cero es que no hay golpe y / o caída.

- ID = 8. Sensor de muro. Se recibe un byte con la siguiente estructura:
    - Bit 0. Indica si existe o no muro (1 indica que hay muro).
    - Bit 1-7. En desuso.
- ID = 9. Sensor infrarrojo izquierdo. Se recibe un byte con la siguiente estructura:
    - Bit 0. Indica si existe altura (gran desnivel, el 1 indica que hay desnivel).
    - Bit 1-7. En desuso.
- ID = 10. Sensor infrarrojo frontal-izquierdo. Se recibe un byte con la siguiente estructura:
    - Bit 0. Indica si existe altura (gran desnivel, el 1 indica que hay desnivel).
    - Bit 1-7. En desuso.
- ID = 11. Sensor infrarrojo frontal-derecho. Se recibe un byte con la siguiente estructura:
    - Bit 0. Indica si existe altura (gran desnivel, el 1 indica que hay desnivel).
    - Bit 1-7. En desuso.
- ID = 12. Sensor infrarrojo derecho. Se recibe un byte con la siguiente estructura:
    - Bit 0. Indica si existe altura (gran desnivel, el 1 indica que hay desnivel).
    - Bit 1-7. En desuso.
- ID = 13. Sensor de muro virtual. Se recibe un byte con la siguiente estructura:
    - Bit 0. Indica la existencia del muro virtual. (1 se detecta el muro).
    - Bit 1-7. En desuso.
- ID = 18. Botones. Se detecta si se ha pulsado el botón de play y avance. Se recibe un byte con la siguiente estructura:
    - Bit 0. Botón Play (1 se ha pulsado / 0 no se ha pulsado).
    - Bit 1. En desuso.
    - Bit 2. Botón avance (1 se ha pulsado / 0 no se ha pulsado).
    - Bit 3-7. En desuso.
- ID = 19. Distancia. Se reciben 2 bytes indicando la distancia recorrida (en mm.) como la suma de la distancia recorrida por cada una de las ruedas entre 2. Cada vez que se consulta esta distancia se inicializan los encoders de las ruedas. El valor recibido es con signo ya que puede ser un valor negativo. El valor va desde -32768 a 32767 mm.

- ID = 20. Ángulo. Se reciben 2 bytes indicando el ángulo de giro del robot (en grados). Como en el caso anterior, cada vez que se consulta el ángulo los valores de los contadores se inicializan. El número es con signo, ya que el ángulo puede ser negativo y va desde -32768 a 32767 grados.
- ID = 21. Estado de la carga de la batería. Se recibe un byte y los valores más significativos que se obtienen son:
    - 0. No cargada.
    - 2. Cargada completamente.
- ID = 22. Voltaje de la batería en milivoltios. Se reciben dos bytes (sin signo) y el valor oscila entre 0 y 65535 mV. Se reciben 2 bytes.
- ID = 23. Corriente de la batería en mA. Una corriente negativa indica consumo y una corriente positiva que se está cargando. El valor oscila entre -32768 y 32767 mA. Se reciben 2 bytes.
- ID = 24. Temperatura de la batería en grados. Es un byte con signo y el valor va desde -128 a 127 grados.
- ID = 25. Carga actual de la batería en mA / hora. Se reciben 2 bytes y va desde 0 a 65535 mA. / hora.
- ID = 26. Capacidad estimada de la carga de la batería en mA. / hora. Se reciben 2 bytes sin signo y el valor oscila entre 0 y 65535 mA. / hora.
- ID = 27. Intensidad de la señal del sensor del muro. Son 2 bytes signo y el valor oscila entre 0 y 4095.
- ID = 28. Se reciben 2 bytes indicando la intensidad del sensor de desnivel izquierdo. El valor oscila entre 0 y 4095.
- ID = 29. Se reciben 2 bytes indicando la intensidad del sensor de desnivel frontal-izquierdo. El valor oscila entre 0 y 4095.
- ID = 30. Se reciben 2 bytes indicando la intensidad del sensor de desnivel frontal-derecho. El valor oscila entre 0 y 4095.
- ID = 31. Se reciben 2 bytes indicando la intensidad del sensor de desnivel derecho. El valor oscila entre 0 y 4095.
- ID = 32. Valor de las entradas digitales del “cargo bay connector”. Se recibe 1 byte de la siguiente manera (valor 1 indica 5 V y 0 0 V):
    - Bit 0. Valor del pin 17.
    - Bit 1. Valor del pin 5.
    - Bit 2. Valor del pin 18.
    - Bit 3. Valor del pin 6.
    - Bit 4. Valor del pin 15.
    - Bit 5- 7. En desuso.
- ID = 33. Valor del pin 4 de la entrada analógica del “cargo bay connector”. El valor oscila entre 0 (0 V.) y 1023 (5 V.).

- ID = 34. Indica si el robot está conectado al "Home Base" o el cargador interno. Se recibe 1 byte de la siguiente forma:
    - Bit 0. Cargador interno. (1 indica que está conectado y 0 no conectado).
    - Bit 1. Home Base. (1 indica que está conectado al home base y 0 no conectado).
    - Bit 2-7. En desuso.

- ID = 35. Modo de conexión del robot. Se recibe 1 byte que puede tomar los siguientes valores:
    - 0 – Off.
    - 1 – Pasivo.
    - 2 – Seguro.
    - 3 – Completo.
- ID = 38. Número de paquetes de datos que se solicitó con el comando 148 y 149. Se recibe 1 bytes y el valor oscila entre 0 y 43.
- ID = 39. Velocidad solicitada. Se reciben 2 bytes indicando cual fue la última velocidad que se solicitó. El valor oscila entre -500 y 500 mm/s.
- ID = 40. Ángulo solicitado. Se reciben 2 bytes indicando cual fue el último ángulo que se solicitó. El valor oscila entre -32768 y 32767 mm.
- ID = 41. Velocidad solicitada de la rueda derecha. Se reciben 2 bytes indicando cual fue la última velocidad que se solicitó a la rueda derecha. El valor oscila entre -32768 y 32767 mm.
- ID = 42. Velocidad solicitada de la rueda izquierda. Se reciben 2 bytes indicando cual fue la última velocidad que se solicitó a la rueda izquierda. El valor oscila entre -32768 y 32767 mm.

3.1.1.4.1.5 Comandos de espera.

Estos comandos son de especial utilidad para la creación de scripts. De hecho, se recomienda su uso solo para el desarrollo de scripts, pues al tratarse de comandos que esperan a que ocurra un evento, es posible que este control sea difícil de programar con comandos por el puerto serie. Estos comandos son:

- [155] [tiempo]. Indica al robot que espere el tiempo indicado por "tiempo" en décimas de segundo. Sólo es un byte y el valor oscila entre 0 y 255.
- [156] [byte alto distancia] [byte bajo distancia]. Espera a que el robot recorra la distancia indicada en los dos bytes. El valor oscila entre -32767 y 32768 mm.
- [157] [byte alto ángulo] [byte bajo ángulo]. Espera a que el robot recorra el ángulo indicado en los dos bytes. El valor oscila entre -32767 y 32768 grados.
- [158][Número de evento]. Espera a que ocurra el evento indicado en "Número de evento". Los posibles valores de eventos se muestran en la tabla 1.

Número de evento.	Evento
1	Rueda caída.
2	Rueda delantera caída.
3	Rueda izquierda caída.
4	Rueda derecha caída.
5	Golpe
6	Golpe izquierdo
7	Golpe derecho
8	Muro virtual
9	Muro
10	Sensor de infrarrojos
11	Sensor de infrarrojos izquierdo
12	Sensor de infrarrojos frontal-izquierdo
13	Sensor de infrarrojos frontal-derecho
14	Sensor de infrarrojos derecho
15	Home Base
16	Botón avance
17	Botón Play
18	Entrada digital 0
19	Entrada digital 1
20	Entrada digital 2
21	Entrada digital 3
22	Modo = Pasivo

Tabla 1. Resumen de eventos de espera.

Para indicar el contrario del evento simplemente indicamos el mismo número de evento pero cambiado de signo. Es decir, si queremos indicar que se espera a que no haya golpe enviamos un -5.

3.1.1.4.2 Programación mediante "scripts".

La programación de un script no es más que un conjunto de comandos que se ejecutan secuencialmente en el robot y reside en el robot, sin que éste tenga que estar conectado a ningún PC. La ejecución de un script consta de dos comandos:

- [152] [Longitud] [Opcode 1][Opcode 2] [Opcode 3] …[Opcode N]. Esta instrucción es la que permite crear el script. Como se ha mencionado anteriormente no es más que la concatenación de varios comandos del robot. La longitud indica los bytes que se van a enviar en total en el script en número de comandos. Este valor no puede superar los 100 bytes.
- [153]. Ejecuta el script que previamente se ha cargado o enviado por el puerto serie. Por ejemplo, si enviamos el script: 152 13 137 1 44 128 0 156 1 144 137 0 0 0 0 realiza un cuadrado de 400 mm y para. Para que se ejecute este script seguidamente enviamos el comando 153. Además, si este comando lo ponemos como último comando de un script, éste se sigue ejecutando de forma infinita, a no ser que sea interrumpido por algún otro comando.

3.1.2 Sistema embebido IGEPv2.

Después de haber visto la arquitectura del Create de IRobot está claro que necesitamos de algún dispositivo que se conecte con el robot y se convierta este sistema en un sistema autónomo, es decir, que no se tenga la necesidad de enviar y recibir los comandos con un PC.

En este sentido se tiene dos posibilidades:

1. Adquirir el Command Module. Una de las posibles opciones sería adquirir el command module para conectarlo con el robot. De esta forma, disponemos de un microprocesador y se pueden realizar programas en C y C++ sin mayor complejidad. Se puede decir que, como inconveniente, es un elemento caro para los componentes que lleva. Realmente la única ventaja que conseguimos es mayor conectividad (conectar mas componentes, sensores, actuadores, etc. al robot) y facilidad de programación.

2. Adquirir un sistema embebido. Se ha optado por esta opción. Con el sistema embebido, además de tener las ventajas que se comentan en el punto 1, es mucho más rápido y robusto que el "Command Module". Además, la mayoría de estos dispositivos llevan un DSP incorporado para realizar instrucciones mucho más rápidas y complejas, como por ejemplo es el tratamiento de imágenes. El único inconveniente que presenta esta tarjeta es que es más cara que el "Command Module". El sistema que se ha adquirido es la tarjeta IGEPv2 de ISEE.

La tarjeta IGEPv2 es un sistema embebido con un tamaño un poco superior al de una tarjeta de crédito (93x65 mm). Las características fundamentales de esta tarjeta son:

- Arquitectura TI OMAP3530.
- CPU ARM CORTEX A8 core a 720 MHz.
- DSP TMS320C64x+.
- GPU POWERVR SGX 530 (OpenGL® ES 2.0, OpenGL® ES 1.0 , OpenVG) + IVA2.2 (Video Hardware Accelerators).
- Administración de energía TPS65950.
- RAM 512MB Mobile Low Power DDR SDRAM a 200 MHz (PoP).
- Flash 512MB NAND (PoP).
- Ethernet 10/100 Mb BaseT (SMSC LAN9221i)
- Wi-Fi IEEE 802.11b/g (Marvell 86w8686B1)
- Bluetooth 2.0 (CSR BC4ROM/21e)
- Antena integrada y conector de antena externa.
- Puerto USB 2.0 OTG x1.
- Puerto USB 2.0 Host x1.
- Puerto MicroSD / MicroSDHC.
- Puerto DVI-D / HDMI.
- Entrada de audio y salida estéreo
- Conector de expansión con I/O, SPI, UART,…
- Conector de expansión con acceso de línea LCD.
- Adaptador de corriente de 5 VDC. o entrada JST.

- Transceptor RS-485.
- Rango de temperatura: -40 a +80º C.
- Precertificado EMI y EMC. Cumple CE.
- Disponible Linux BSP (se facilita un paquete de instalación).
- Etc,

El sistema operativo de la tarjeta IGEPv2 es GNU-Linux Kernel. Tras la compilación e instalación del sistema operativo, ya se puede trabajar directamente con la tarjeta. La apariencia de la tarjeta se muestra en la figura 21.

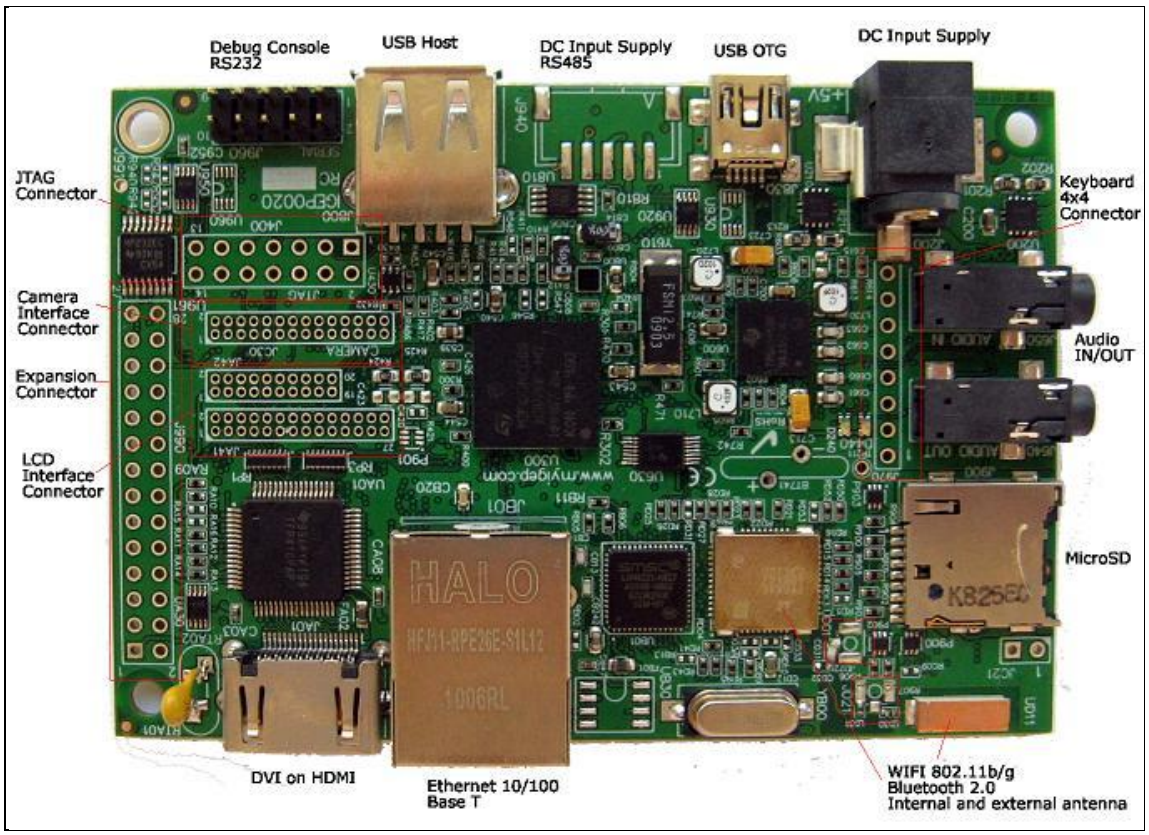


Figura 21. Tarjeta IGEPv2.

3.1.2.1 Conexión de la tarjeta al PC.

Lógicamente para poder copiar ficheros, lanzar programas, etc., es necesario conectar el PC a la tarjeta, ya sea por Ethernet o por Wi-Fi. En cualquier caso, una vez se ha conectado la tarjeta al PC los pasos a seguir para realizar cualquier trabajo con la tarjeta son:

1. Abrir el programa putty.exe.

2. Crear una nueva sesión con las opciones que se muestran en la figura 22.

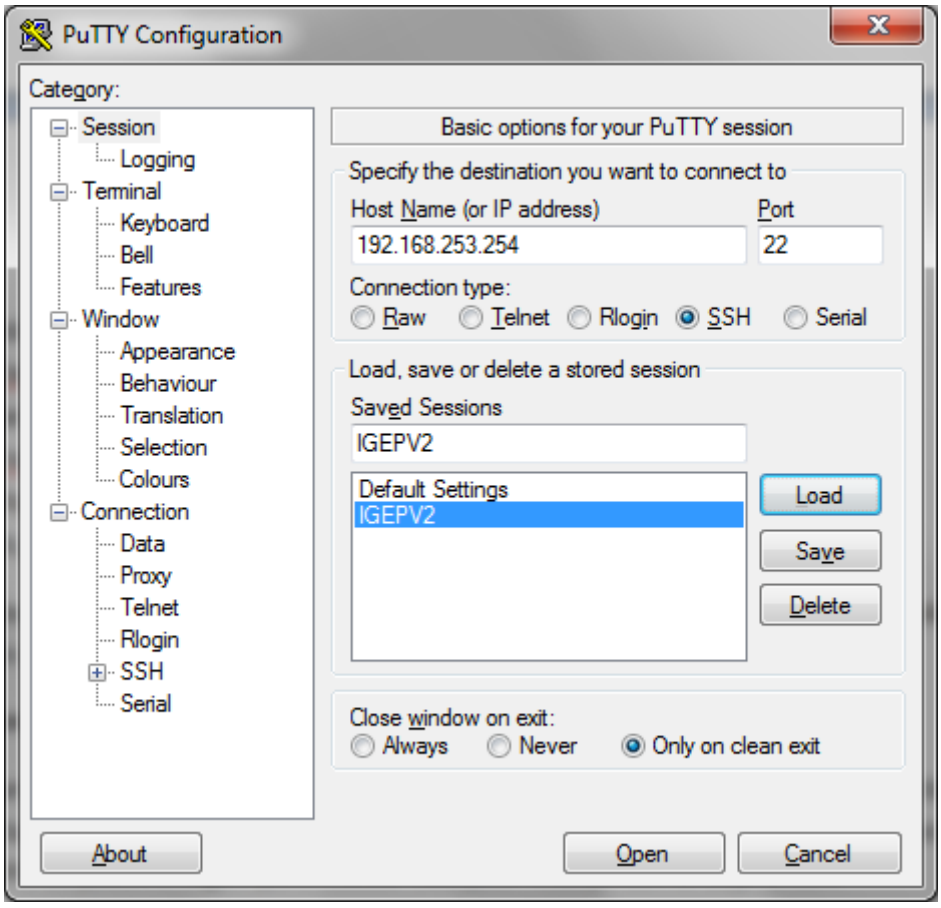


Figura 22. Configuración de una nueva sesión de "Putty" para conectar el PC con la tarjeta.

3. Si todo ha ido bien, se abre una ventana en modo consola donde debemos de introducir el usuario "root" y password "1234567890". Ahora, ya estamos dentro de la tarjeta y podemos ejecutar ficheros o realizar las operaciones básicas con comandos, ya que, cuando estamos conectados con "Putty" a la tarjeta, se está trabajando con comandos de Linux.

3.1.2.2 Copiado de ficheros.

Para copiar ficheros de la tarjeta al PC o del PC a la tarjeta es necesario el programa "pscp.exe" ya que, con el programa visto en el punto anterior no se puede realizar el copiado de ficheros. Lógicamente, antes de copiar ningún fichero debemos de estar conectados a la tarjeta. Para utilizar el programa se ejecuta directamente en ms-dos y el formato para copiar ficheros del PC a la tarjeta es: pscp –scp "FicheroOrigen" "FicheroDestino".

3.1.2.3 Conexión de la tarjeta al IRobot Create.

Tal y como se ha justificado al principio del documento, se hace necesario el uso de esta tarjeta para conseguir que el Create sea autónomo. Es decir, el objetivo es poder realizar un programa que se ejecute en la tarjeta y envíe y reciba información por el puerto serie. Para conseguir esto, se conecta la tarjeta al Create con un cable serie directamente y el programa que ejecutemos ya se encargará de enviar y recibir información por el puerto serie que hemos conectado.

3.1.2.4 Realizar programas para que se ejecuten en la tarjeta IGEPv2.

Para poder realizar aplicaciones que se ejecuten en la tarjeta se puede adquirir el Code::Blocks. Es un entorno de desarrollo para programar en C y C++ y está disponible tanto para Linux como para Windows. Por último, es necesario instalar el compilador para que efectivamente el programa funcione en la tarjeta. El compilador usado es: "arm-none-linux-gnueabi".

3.1.3 Cámara cenital.

Tal y como se ha comentado anteriormente hace falta una cámara para poder obtener las imágenes y así procesarlas en función de la tarea que se quiera hacer. Las características de la cámara "Pro 3000" se resumen en la tabla 2.

General	
Tipo de dispositivo	Cámara Web
Anchura	4 cm
Profundidad	3 cm
Altura	4 cm
Peso	300 Gramos.
Cámara	
Tipo	Color
Formato video digital	AVI
Imagen fija	JPEG, BMP, TIFF, PCX, PSD, PNG, TGA - 640 x 480 - hasta 30 imágenes por segundo
Soporte de audio	Sí
Características	Compatibilidad con USB
Sensor de imagen	
Tipo	CCD 1/4"
Velocidad	30 fps
Balance de blanco	Automático
Construcción de la lente	
Iris del objetivo	F/2.0
Margen mínimo de enfoque	15 cm
Interfaces	
Interfaz de ordenador	USB
Adaptador	USB Integrado
Expansión / Conectividad	
Interfaces	1 x USB - 4 PIN USB tipo A
Diversos	
Cumplimiento de normas	Plug and Play
Software / Requisitos del sistema	
Capacidad mínima de RAM	64 MB
Espacio libre mínimo de disco duro	100 MB
Parámetros de entorno	
Temperatura de funcionamiento	0^0C - 40^0C
Humedad relativa	10 – 90 %

Tabla 2. Características de la cámara "Pro 3000" de Logitech.

3.1.4 Koala de K-team.

El Koala es un robot móvil de tamaño medio (dimensiones de 32x32x20 cm), que permite desarrollar, desde aplicaciones relacionadas con la vigilancia, transporte de objetos y planificación de trayectorias a seguir, hasta aplicaciones más complejas como la Telemanipulación o la detección y reconocimiento de objetos.

Tal y como se muestra en la Figura 23, se trata de un robot de 6 ruedas todo-terreno, lo que le posibilita para trabajar desde entornos de oficina hasta terrenos más irregulares de hasta una inclinación máxima de 43 grados. Para ello cuenta con dos servomotores de corriente continua con encoders incrementales. Además, gracias a sus 16 sensores infrarrojos de proximidad distribuidos por todo el robot, se puede obtener una percepción real de los objetos que hay en su entorno. Tal y como se comentará posteriormente, en el robot se ha instalado una tarjeta adicional de control y una tarjeta de comunicaciones inalámbricas.

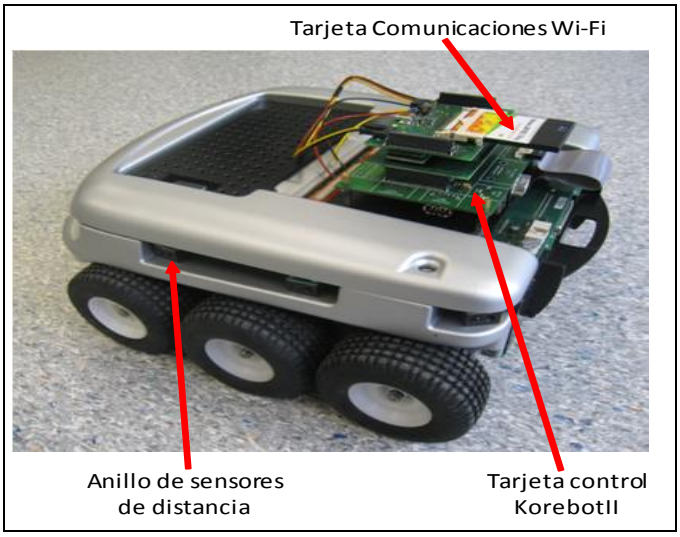


Figura 23. Robot Koala (K-Team)

Una característica importante del Koala es su gran potencia de cálculo, puesto que cuenta con un procesador “Motorola 68331 @ 22 MHz” con 1 Mbyte de RAM y ROM. Además, permite introducirle una gran cantidad de expansiones, mediante las cuales podemos dotarle de aún más características de las que ya tiene.

Para la programación y el control del robot se tienen diversas opciones. La más simple es mediante la utilización del puerto RS232 disponible en el robot. Mediante este puerto y una aplicación de hyperterminal se pueden utilizar entornos como el LabView o MATLAB. En este caso el algoritmo de control se ejecuta en el PC conectado con el robot. De esta forma el PC puede leer información del robot (como el valor de los encoders, de los sensores de distancia, del nivel de baterías etc.). A partir de esta información el PC puede enviarle por ejemplo las referencias de velocidades de las ruedas o el avance que el robot debe realizar.

La segunda opción consiste en la ejecución embebida de los algoritmos de control. Para ello el fabricante proporciona un entorno de desarrollo y un compilador que permite programar las aplicaciones en lenguaje C, de forma que permite la ejecución del programa compilado y almacenado previamente en la memoria ROM del robot.

En este trabajo, para poder dotar al robot de un sistema de comunicaciones abierto y flexible que pueda integrarse en una red de computadores se ha optado por añadir al robot una tarjeta de control adicional, encastrando sobre ésta una tarjeta de comunicaciones Wi-Fi.

La tarjeta de control incorporada es la Korebot II. Es una plataforma embebida potente de dimensiones muy reducidas basada en el *gumstix Verdex PRO* con un microprocesador Intel XSCALE PXA-270 a 600MHz" con 128 Mbytes de RAM y 32 Mbytes de Flash que incorpora su propio sistema operativo (Linux 2.6.x, OpenEmbedded Ansgtröm Distribution).

La Korebot II tiene disponible varios conectores, de los que podemos destacar un puerto USB (que permite conectar cualquier tipo de dispositivo USB directamente sobre la placa, como por ejemplo, una videocámara USB, ratón o teclado) y un slot Compact Flash. En éste último se puede conectar cualquier extensión de pocket PC, como tarjetas de almacenamiento flash, Bluetooth, etc.

La tarjeta Korebot II funciona con un sistema operativo Linux. Para la programación de ésta el fabricante proporciona un compilador cruzado y una librería de funciones que permite el control del robot móvil. Las aplicaciones de control se programan en un PC, transfiriéndose el programa ejecutable a la Korebot II vía comunicación serie.

La ventaja de añadir la Korebot II al robot no sólo radica en que tiene mayor memoria y potencia de cálculo, sino que además, permite conectar al robot diferentes dispositivos. En esta aplicación, para dotar al robot de comunicaciones inalámbricas se ha conectado en el slot Compact Flash la tarjeta Wireless Lan Compact Flash Ambicom WL5400G-CF. Se trata de una tarjeta Wireless Lan 802.11 b/g con unas dimensiones de (56x42x5 mm) y un peso de 10 gramos que permite conectarse a cualquier red inalámbrica sin necesidad de cables con una velocidad de transferencia de datos desde 1 hasta 54Mbps.

3.1.5 SAMSUNG I900 OMNIA.

Tal y como se ha mencionado al principio del documento, se ha desarrollado un programa para poder mover el robot Create de IRobot de forma dirigida hasta la posición inicial deseada. La comunicación del móvil con el Create será de forma inalámbrica mediante Bluetooth. Este dispositivo tiene las características necesarias para dicha programación. A continuación se detallan las características más importantes del dispositivo móvil relacionadas con el programa desarrollado. Para cualquier consulta relacionada con este dispositivo, se puede visitar la página http://www.samsung.com/es/consumer/mobile-phone/mobile-phones/touch-screen/SGH-I900XKMFOP/index.idx?pagetype=prd_detail&tab=spec&fullspec=F.

- Sistema operativo: Windows Mobile 6.1.
- Pantalla táctil LCD TFT de 3.2" WQVGA.
- 16 Gb. de memoria interna.
- Conexiones: Bluetooth, WiFi y USB.
- Acelerómetros para obtener la orientación del móvil.
- Aplicación PC Sync para la conexión con el PC vía USB o Bluetooth, para así poder transferir archivos del PC al dispositivo o del dispositivo al PC.

En la imagen 24 se muestra el aspecto del dispositivo.

Figura 24. OMNIA I900 de SAMSUNG.

3.1.6 PC cliente.

Por último, se dispone de un PC que será el que actúe como cliente en la comunicación. Además, en este PC es donde tenemos conectada la cámara cenital y el que se encargará de realizar todo el procesamiento de la imagen. Es decir, en este PC se ha desarrollado un programa que se conecta con los servidores, que son los robots. Cuando realizamos la petición de recoger la pelota conjuntamente con los robots, es cuando se captura la imagen, se procesa y se realiza todo el trabajo por parte de todos los elementos conectados en la red. En punto posteriores se explica detalladamente toda la logística y funcionamiento de dicho programa, así como las diferentes alternativas que proporciona.

3.2 Creación de una librería (LIB) para la programación del Create de IRobot.

Como se ha comentado anteriormente, es necesario crear algún tipo de librería que encapsule los comandos del Create, ya que la programación con este tipo de comandos es algo meticulosa y dificultaría la depuración de los programas. Para ello, se ha creado con “Microsoft Visual Studio 2008” una librería (tanto .lib como .dll) que contiene todas las llamadas para enviar y recibir información del robot. Estas librerías se pueden incluir en todos aquellos programas que permitan incluir librerías, es decir, prácticamente con todas las herramientas de programación existentes.

Por otro lado, hay que tener en cuenta que la librería solo puede ser usada para programas desarrollados en el PC. Es decir, cualquier desarrollo que se quiera realizar en una nueva arquitectura que no sea el PC no podrá ser incluida. Si la programación sigue siendo C o C++ los ficheros fuente de dicha librería pueden ser incluidos en un nuevo proyecto sin mayor complejidad. En este sentido, para los desarrollos realizados en la tarjeta IGEPv2 no se ha usado esta librería, puesto que se utiliza la herramienta Code::Blocks y, aunque sea lenguaje C, hay ciertas llamadas y librerías que son exclusivamente para la tarjeta, por lo que no se ha podido usar. La forma de conectarse por el puerto serie virtual, por ejemplo, es diferente a la usada en la librería. Lo que se ha hecho es incluir los ficheros de encabezado y fuente en los proyectos y, adaptarlos a esta nueva herramienta.

En la tabla 3 se resumen las funciones de la librería que se han implementado.

Función	Descripción
bool Velocidad(int nVelocidad, int nRadio);	Pone las dos ruedas del robot a girar a la velocidad indicada y con el ángulo de giro indicado
bool Velocidad_Recto(int nVelocidad);	Velocidad para que vaya recto
bool Gira_Derecha(int nVelocidad);	Velocidad para que gire sentido horario
bool Gira_Izquierda(int nVelocidad);	Velocidad para que gire sentido antihorario
bool Velocidad_Independiente(int nRueda_Izq, int nRueda_Der);	Cambia la velocidad de las ruedas de forma independiente
bool Salidas_Digitales(int nSalida1, int nSalida2, int nSalida3);	Cambia el valor de las tres salidas digitales del robot (pines 19, 7 y 20 del cargo bay conector
bool Parachoque_Nivel_Ruedas(bool &bParachoque,bool &bRueda);	Obtiene el estado del parachoques y de la caida de las 3 ruedas
bool Muro(bool &bMuro);	Obtiene el estado de muro para saber si hay obstaculo o no
bool Desnivel(bool &bDesnivel);	Obtiene si alguna de las ruedas esta desniveladas, que son desnivel izquierdo, izquierdo frontal, derecho y derecho-frontal
bool Muro_Virtual(bool &bMuroV);	Obtiene si hay choque con el muro virtual. De momento en desuso
bool Infrarojos(short int &nEstado);	Obtiene el valor del infrarojo. De momento en desuso
bool Botones(bool &bPlay,bool &bAvance);	Obtiene el estado de los botones de play y avance
bool Distancia(short int &nDistancia);	Obtiene la distancia en mm que ha recorrido el

	robot. Cada vez que se llama a esta función se inicializa el contador a 0. Es decir, si la llamo una vez y me devuelve 5 mm, ese miusmo contador se vuelve a inicializar a 0 desde ese momento.
bool Angulo(short int &nAngulo);	Obtiene el ángulo engrados de las ruedas del robot. Cada vez que se llama a esta función se inicializa el contador a 0.
bool Estado_Bateria(short int &nEstado);	Obtiene como se encuentra la bateria. Cargando, espera,...
bool Voltaje_Bateria(short int &nVoltaje);	Obtiene el voltaje de la bateria en mV
bool Corriente_Bateria(short int &nMiliamperios);	Obtiene la corriente de la bateria en mA
bool Temperatura_Bateria(short int &nTemperatura);	Obtiene la temperatura de la bateria en grados Celsius
bool Carga_Bateria(short int &nCarga);	Obtiene la carga que le queda a la bateria en mA hora
bool Capacidad_Bateria(short int &nCapacidad);	Obtiene la capacidad de la bateria en mA hora
bool Senal_Muro(short int &nSMuro);	Obtiene la intensidad del sensor de muro
bool Senal_Desnivel_Izquierdo(short int &nSIzq);	Obtiene la intensidad del sensor de desnivel izquierdo
bool Senal_Desnivel_Izquierdo_Fr(short int &nSIzq_Fr);	Obtiene la intensidad del sensor de desnivel izquierdo-frontal
bool Senal_Desnivel_Derecho_Fr(short int &nSDer_Fr);	Obtiene la intensidad del sensor de desnivel derecho-frontal
bool Senal_Desnivel_Derecho(short int &nSDer);	Obtiene la intensidad del sensor de desnivel derecho
bool Senal_Desniveles(short int &nSIzq,short int &nSIzq_Fr,short int &nSDer_Fr,short int &nSDer);	Obtiene la intensidad del sensor de los 4 desniveles
bool Entradas_Digitales(bool &bEntrada0, bool &bEntrada1, bool &bEntrada2, bool &bEntrada3);	Obtiene el valor de las 4 entradas digitales

bool Entrada_Analogica(short int &nEntrada);	Obtiene el valor de la entradas analogica
bool Fuente_Carga(bool &bHome,bool &bInterno);	Obtiene las fuentes de carga disponibles
bool Modo_OI(short int &nModo);	Obtiene el modo OI en el que esta trabajando
bool Velocidad_Solicitada(short int &nVelocidad);	Obtiene el ultimo valor de velocidad que se pidio. En teoria debe de coincidir con la velocidad del robot
bool Radio_Solicitado(short int &nRadio);	Obtiene el ultimo valor de radio solicitado. En teoria debe de coincidir con el radio de las ruedas del robot
bool Velocidad_Solic_Derecha(short int &nVelocidad_Der);	Obtiene el ultimo valor de velocidad de la rueda derecha que se pidio.
bool Velocidad_Solic_Izquierda(short int &nVelocidad_Izq);	Obtiene el ultimo valor de velocidad de la rueda izquierda que se pidio.
enum Evento {Rueda_Caida = 1, Rueda_Caida_FR = 2, Rueda_Caida_Izq = 3, Rueda_Caida_Der = 4, Parachoque = 5, Parachoque_Izq = 6, Parachoque_Der = 7, Muro_Virtual = 8, Muro = 9, Desnivel = 10, Desnivel_Izq = 11, Desnivel_Izq_Fr = 12, Desnivel_Der_Fr = 13, Desnivel_Der = 14, Home_Base = 15, Boton_Avance = 16, Boton_Play = 17, Entrada_Digital_0 = 18, Entrada_Digital_1 = 19, Entrada_Digital_2 = 20, Entrada_Digital_3 = 21, Modo_OI_Pasivo = 22, NO_Rueda_Caida = 255, NO_Rueda_Caida_FR = 254, NO_Rueda_Caida_Izq = 253, NO_Rueda_Caida_Der = 252, NO_Parachoque = 251, NO_Parachoque_Izq = 250, NO_Parachoque_Der = 249,	Definicion de estructura para el evento de espera

NO_Muro_Virtual = 248, NO_Muro = 247, NO_Desnivel = 246, NO_Desnivel_Izq = 245, NO_Desnivel_Izq_Fr = 244, NO_Desnivel_Der_Fr = 243, NO_Desnivel_Der = 242, NO_Home_Base = 241, NO_Boton_Avance = 240, NO_Boton_Play = 239, NO_Entrada_Digital_0 = 238, NO_Entrada_Digital_1 = 237, NO_Entrada_Digital_2 = 236, NO_Entrada_Digital_3 = 235, NO_Modo_OI_Pasivo = 234};	
bool Esperar_Tiempo(int nTiempo);	Con esta función espera el tiempo indicado en nTiempo en decimas de segundo
bool Esperar_Distancia(int nDistancia);	Con esta función se espera a ejecutar la siguiente instrucción hasta que recorra la distancia indicada en mm. Es decir, hasta que no se recorra nDistancia mm el robot no ejecuta la siguiente instruccion.
bool Esperar_Angulo(int nAngulo);	Con esta función se espera a ejecutar la siguiente instrucción hasta que recorra el angulo indicado en grados.
bool Esperar_Evento(Evento evento);	Con esta función esperamos a que ocurra el evento que se especifica en el parametro. Este parámetro es del tipo Evento y según el nombre de la enumeración ya indica en si que evento es el que se espera

Tabla 3. Resumen de las funciones implementadas en la librería.

3.3 Procesamiento de las imágenes.

Se han desarrollado dos alternativas diferentes haciendo uso de herramientas diferentes. Por un lado, se han realizado varias funciones en MATLAB para generar la trayectoria, a partir de una imagen dada, teniendo en cuenta los obstáculos. Lógicamente, estas funciones implementadas en MATLAB de alguna forma se deben de incluir en la programación del PC cliente, que es el encargado de recoger las imágenes. Para ello, es necesario crear una librería y poderla incluir en el proyecto de “Visual Studio 2008”, que es donde se ha desarrollado todo el código referente al PC cliente. La forma de crear esta librería en MATLAB y poderla incluir en un proyecto de “Visual Studio” se explica en el anexo A.

Básicamente, la forma de obtener las posiciones iniciales, finales y la de los obstáculos de una imagen dada, es tal y como se ha explicado en el apartado 2.2.2 del presente documento. Lo que hacemos es segmentar la imagen para detectar, mediante la extracción de los componentes RGB y la segmentación de la imagen, las posiciones de todos los elementos que intervienen en esta tarea, como son los robots, la posición final y los obstáculos. Cada uno de estos componentes tiene diferentes valores RGB para poder distinguirlos perfectamente. En la tabla 4 se resumen estos componentes.

Elemento	Color	Red (R)	Green (G)	Blue (B)
Create	Negro	R < 70	G < 70	B < 70
Koala	Amarillo	200 <= R < 256	200 <= G < 256	80 <= B <= 180
Obstáculos	Azul	40 < R < 120	90 < G < 200	150 <= B < 256
Pelota	Rojo	130 <= R < 256	10 <= G <= 90	10 <= B <= 90

Tabla 4. Componentes RGB de los diferentes elementos.

Una vez calculados estos puntos mediante la segmentación, debemos de calcular la trayectoria libre de colisiones. Para ello se ha empleado el algoritmo descrito en el punto 2.2.3 de este documento (algoritmo descomposición de celdas).

En una segunda parte, y como alternativa a la implementación desarrollada en MATLAB, se ha implementado la segmentación de imágenes haciendo uso de "OpenCV 2.1" para no depender de la librería de MATLAB. Esto es debido a que, al incluir la librería de MATLAB en un proyecto de "Visual Studio" ralentiza el funcionamiento del programa. Además, experimentalmente, se ha demostrado que si queremos capturar más de una imagen y procesarla con esta librería el tiempo es excesivo (tiempos de más de dos segundos) y, por tanto, esto es un gran impedimento para realizar tareas donde se requiera un conjunto de imágenes para realizar algún tipo de trayectoria. Por ejemplo, supongamos que cada 500 ms se quiere procesar una imagen para corregir la trayectoria del robot con algún tipo de filtro. Esto sería imposible, pues el tiempo de respuesta cada vez que se llama a la función de la librería es muy alto y sería imposible hacer este trabajo.

La forma de trabajar con OPENCV es muy similar a la empleada con MATLAB solo que las funciones cambian. Es decir, también se captura la imagen y se segmentan las imágenes realizando una erosión y dilatación de ésta. El resultado es el mismo pero la ejecución en lo que respecta a tiempo, es mucho más rápido, para capturar imágenes de forma continuada. El tiempo que pasa de una captura a la siguiente es menor a 100 ms.

En la imagen 25 se muestra un ejemplo de la segmentación para obtener las posiciones de los robots y de la pelota.

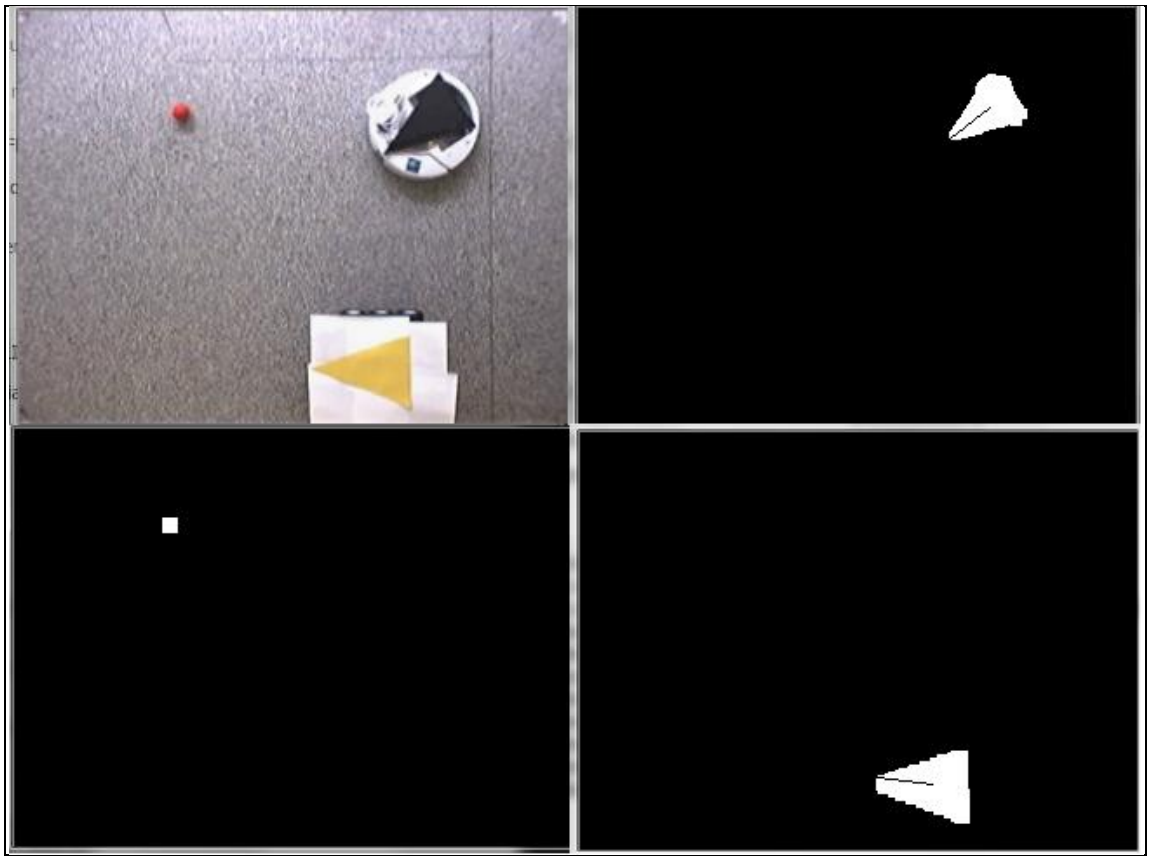


Figura 25. Segmentación para obtener las posiciones iniciales de los robots y la posición final.

Como podemos observar en la imagen anterior, la segmentación funciona perfectamente, pues cada elemento es detectado de forma exacta. Además, para el caso de los robots, también se calcula el punto más lejano del perímetro para calcular la orientación de éste. En la imagen se muestra una línea recta desde el centroide del robot al punto más lejano, para comprobar que efectivamente este punto es el correcto.

3.4 Arquitectura síncrona. Seguimiento de trayectorias.

Después de ver todos los aspectos teóricos y prácticos referentes al procesamiento de imágenes, generación de trayectorias y arquitecturas empleadas en la comunicación, se debe de desarrollar algún tipo de aplicación para poder estudiar los resultados de todo lo expuesto.

En una primera parte de este trabajo se ha desarrollado una arquitectura simple de comunicaciones para comprobar, en una primera fase, que el desarrollo de todos los puntos expuestos es posible y así estudiar los diferentes comportamientos. Lógicamente, antes de pasar a un comportamiento asíncrono de la comunicación hay que comprobar que la comunicación síncrona funciona perfectamente.

Para ello, se ha desarrollado una aplicación conjunta con diferentes objetivos. Por un lado, comprobar que la generación de trayectorias y procesamiento de imágenes funciona perfectamente. Por otro lado, también se comprueba que la acción de control para el seguimiento de esta trayectoria es correcta y, por tanto, puede ser empleado para conseguir algún objetivo más complicado. En esta arquitectura intervienen sólo dos elementos en la

comunicación, que son el Create de IRobot y un PC, donde tenemos conectada la cámara cenital. El robot, junto con la tarjeta IGEPv2 actúan de servidor, de tal forma que están a la espera de que el cliente se conecte para realizar alguna tarea. El cliente es el PC y puede indicar diferentes tipo de tareas: realizar un rectángulo, círculo, seguir una trayectoria con evitación de obstáculos, etc. y el robot recibe esta información para realizar la ejecución de la tarea. Otro punto relevante para esta programación es la ejecución mediante hilos. Es decir, el robot, al ser el elemento servidor, tiene un hilo principal y es el encargado de recibir la tarea que se quiere realizar. Una vez se obtiene esta información del cliente, este proceso crea un hilo que será el encargado de realizar la tarea encomendada. Esta funcionalidad es mucho más abierta y correcta pues, el elemento servidor nunca se puede quedar bloqueado esperando a que acabe una ejecución. Por ejemplo, supongamos que por cualquier motivo debemos de realizar una parada de emergencia. Este comando, en cualquier momento debe de ser posible enviarlo al servidor para que el robot se pare inmediatamente.

Resumiendo, se dispone por tanto, de dos programas:

1. Programa servidor. Instalado en la tarjeta IGEPv2. Esta tarjeta es que la que está unida al Create y encargada de procesar todos los datos para enviar y recibir al robot. A partir de ahora, esta arquitectura de Sistema embebido – robot será un elemento fusionado, teniendo en cuenta que cuando se menciona al Create, realmente el programa está embarcado en el sistema embebido. Este programa, como se ha mencionado, actúa de servidor y está a la espera de conexiones de clientes para realizar tareas.

2. Programa cliente. Es un PC donde está conectada la cámara cenital y actúa de cliente. Es decir, en función de la tarea que se quiera realizar, le envía el comando oportuno al servidor para realizar la tarea. Además, si tiene que ver con el seguimiento de una trayectoria con evitación de obstáculos, también obtiene la imagen de la cámara y la procesa para obtener las diferentes posiciones. Este programa es el que se ha desarrollado con “Microsoft Visual Studio 2008” y es el encargado de todas las peticiones que se realizan a los servidores. En el anexo B se explica detalladamente el funcionamiento de dicho programa, a nivel de usuario, y cuáles son las posibilidades que brinda. El programa tiene el aspecto que se muestra en la figura 26.

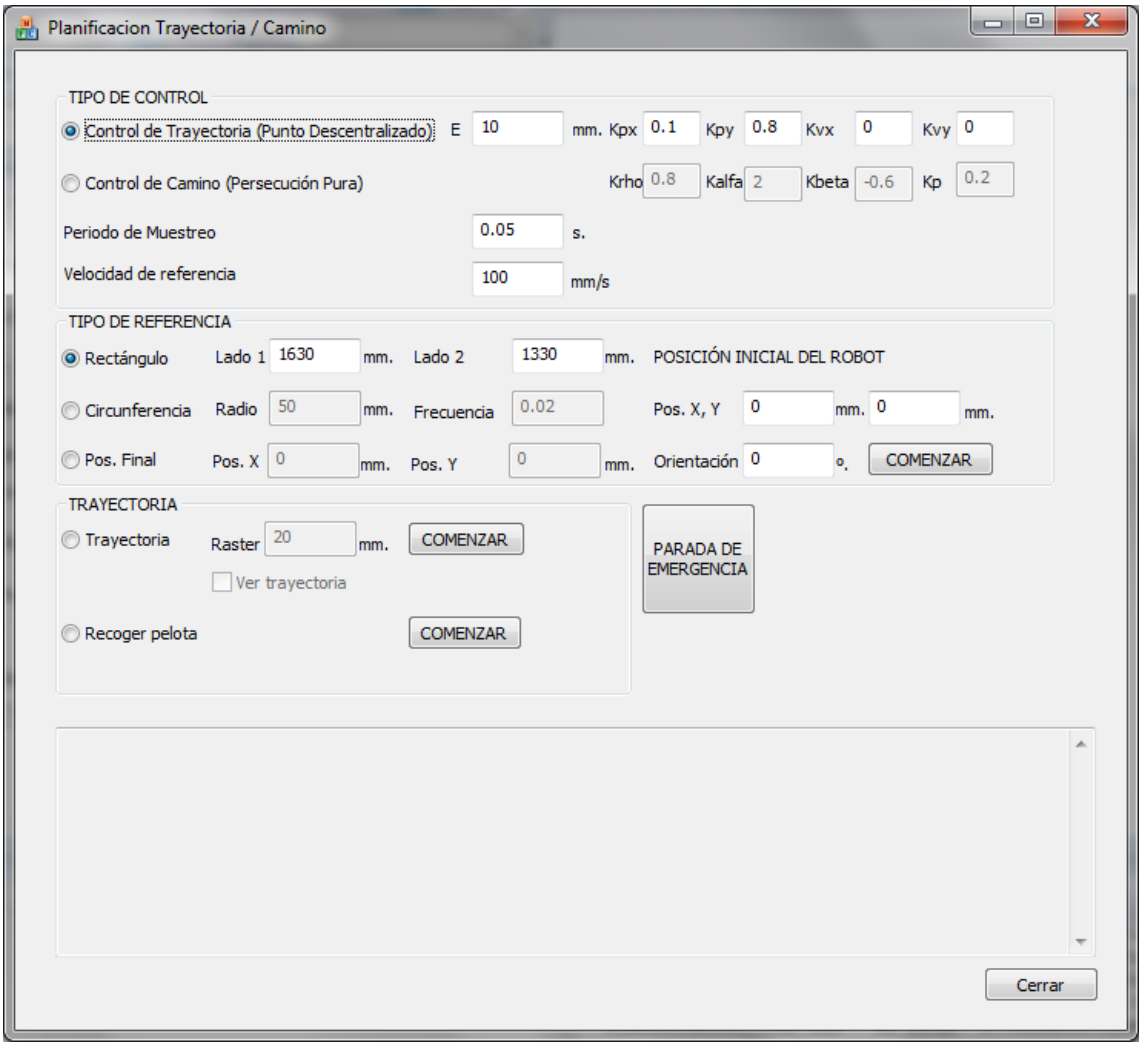


Figura 26. Programa cliente desarrollado.

En la figura 27 se muestra la arquitectura implementada.

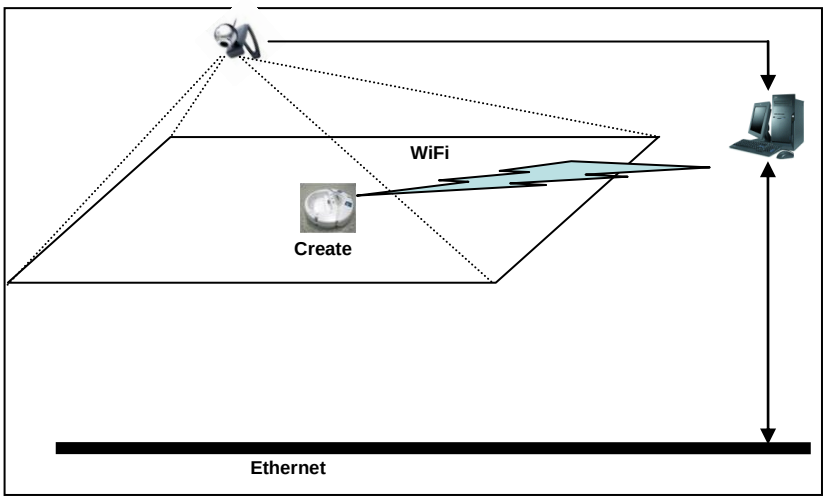


Figura 27. Arquitectura cliente-servidor síncrona.

En esta arquitectura solo existe un servidor y un cliente. Además, esta arquitectura es síncrona a nivel de comunicación. Es decir, siempre en todo momento, tanto en la parte del cliente como en la del servidor, se sabe a priori cual es la información que va a llegar y en qué momento del tiempo. Por ello, de una forma esquemática, la funcionalidad se esta arquitectura es:

1. El servidor está esperando nueva peticiones.

2. El cliente realiza la petición de conexión con el servidor.

3. El servidor acepta dicha petición y espera a recibir la información para realizar una tarea.

4. El cliente envía dicha información y el servidor, una vez procesada, comienza la ejecución de dicha tarea. El cliente se queda a la espera de que el servidor termine la ejecución.

5. El servidor realiza la tarea y, una vez finalizada se lo comunica al cliente.

6. El cliente reciba la información para saber que el servidor ha terminado la ejecución de la tarea y cierra la conexión.

Básicamente el funcionamiento de las comunicaciones y conexiones es similar en todos los casos, independientemente de la tarea a realizar.

Además, en el programa cliente se ajustan todos los parámetros referentes al algoritmo de seguimiento de la trayectoria. De esta forma, podemos ajustar dichos parámetros y estudiar cuáles son los que proporcionan mejores resultados.

En la imagen 28 se muestra el resultado del seguimiento de una trayectoria con forma rectangular, de lados 1630 mm y 1330 mm.

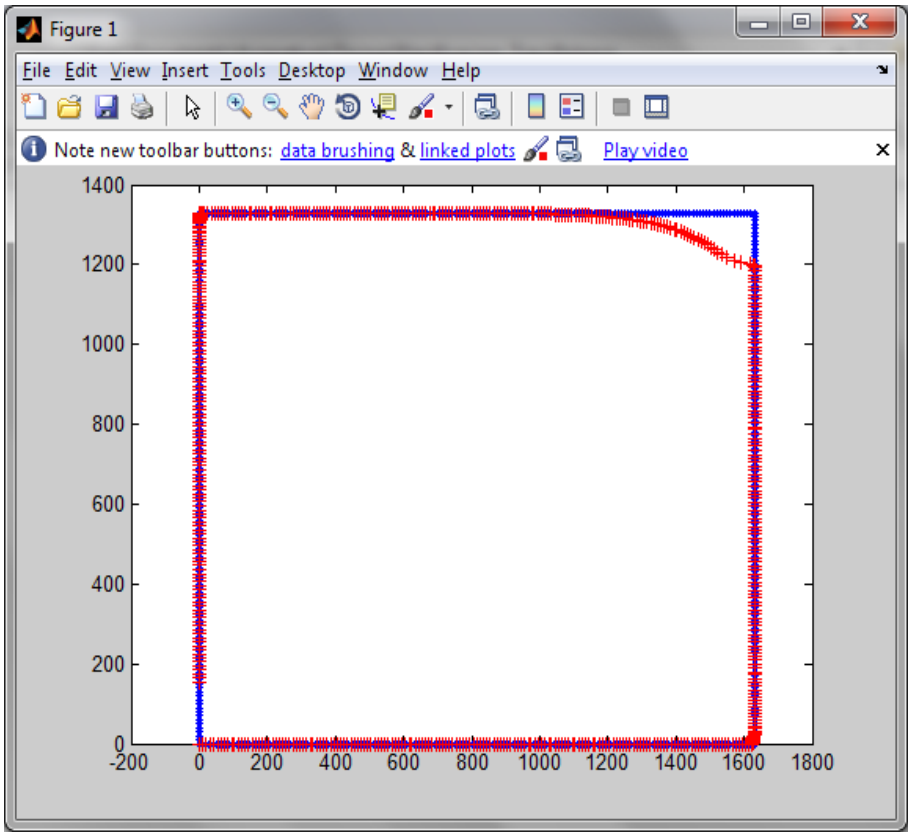


Figura 28. Resultado de seguir una trayectoria con forma rectangular.

Como observamos en la imagen anterior, el color azul es la trayectoria que se pretende seguir y el color rojo son las posiciones reales del robot a la hora de seguir esta trayectoria. Como podemos observar, el resultado obtenido es óptimo ya que realiza el seguimiento de la trayectoria perfectamente. Cabe destacar que, al intentar llegar a la posición final, que es la posición 0,0, el robot se aproxima mucho a esta posición pero no logra una posición exacta. Esto es debido a que, en cada periodo de muestreo se comete un pequeño error, que se va acumulando a lo largo de la trayectoria y termina sin conseguir esta posición. Esto también es debido a la forma de parar el algoritmo de seguimiento de trayectorias. Es decir, la forma en que se ha implementado esta trayectoria es muestreándola cada cierto periodo de muestreo y es donde debería de estar el robot en un instante de tiempo dado. En cambio, si la finalización de la trayectoria hubiera sido un umbral de cercanía, por ejemplo, si está a un 98 % de la posición final que se pare la ejecución, el resultado es que si que hubiera conseguido esta posición pero, de una forma menos realista.

Los parámetros indicados para conseguir este resultado son:

- Velocidad de referencia = 100 mm /s.
- Periodo de muestreo (T) = 0.05 s.
- Distancia del punto descentralizado (E) = 10 mm.
- Ganancia K_{px} (Ganancia de posición en la coordenada X)= 0.1
- Ganancia K_{py} (Ganancia de posición en la coordenada Y)= 0.8
- Ganancia K_{vx} (Ganancia de velocidad en la coordenada X)= 0
- Ganancia K_{vy} (Ganancia de velocidad en la coordenada X)= 0

Otro ejemplo, es el seguimiento de un círculo de radio 500 mm. El resultado se muestra en la figura 29.

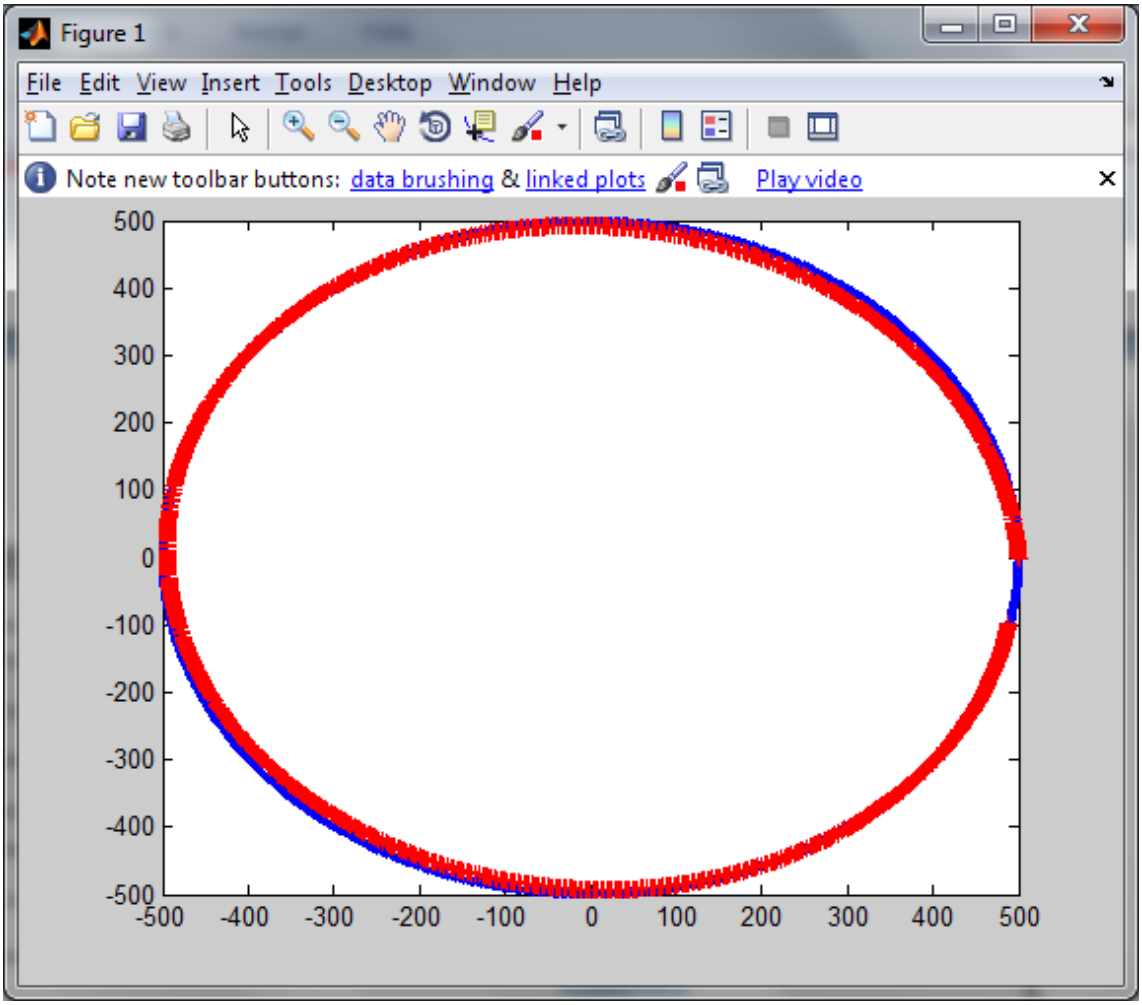


Figura 29. Resultado de seguir una trayectoria con forma circular.

Como en el caso anterior, el resultado del seguimiento de la trayectoria es muy bueno y ocurre como en el caso del rectángulo que, debido a los errores cometidos en cada periodo, se acumulan y el robot no llega de forma exacta a la posición final, pero realiza una buena aproximación.

En cualquier caso e independientemente del tipo de trayectoria que se quiera realizar, en lo que respecta a nivel de comunicaciones entre los diferentes elementos de la arquitectura, en la imagen 30 se muestra un esquema de la logística de estas comunicaciones.

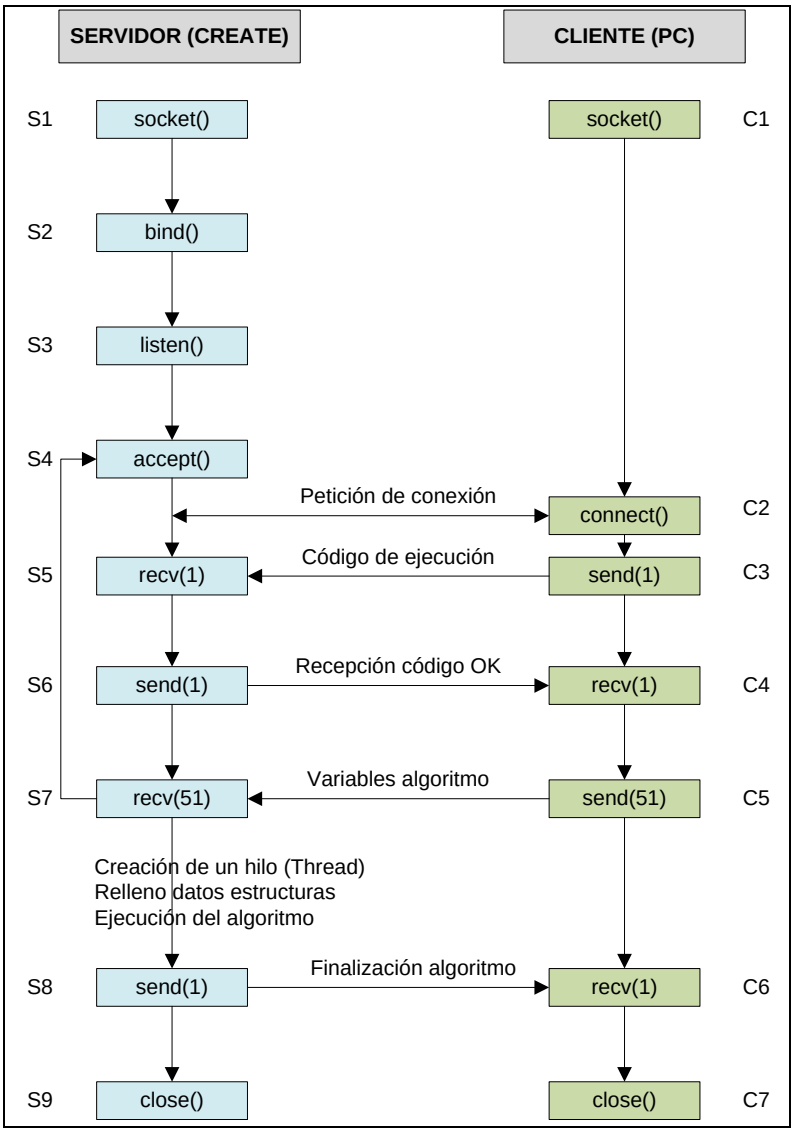


Figura 30. Comunicación de sockets en una arquitectura síncrona

De forma resumida, los pasos que se siguen en la imagen 28 se resumen a continuación:

SERVIDOR

- S1. Inicialización del socket.
- S2. Ponemos el nombre y puerto del socket: Dirección IP: “192.168.253.254”, Puerto = 3490.
- S3. Prepara al socket para recibir conexiones.
- S4. Llamada bloqueante y sirve para esperar peticiones de conexiones de los clientes.
- S5. El servidor recibe 1 byte que contiene el código de comando que se quiere ejecutar. Los posibles códigos son:

- Código 10. Seguimiento de una trayectoria. Ejecuta el algoritmo del punto descentralizado o persecución pura para seguir una trayectoria generada previamente, como por ejemplo, un rectángulo, círculo, posiciones inicial y final manuales, etc.
- Código 20. Parada de emergencia. Parada urgente del funcionamiento del robot. Pone las velocidades de las ruedas a 0 mm/s y termina la ejecución de cualquier hilo del programa. El servidor se queda a la espera de nueva peticiones.
- Código 30. Parada y ejecución del algoritmo de seguimiento de trayectoria. Para la ejecución de algoritmo de seguimiento de trayectorias, recibe la nueva información de parámetros y posiciones y vuelve a lanzar otra vez el algoritmo. Opción utilizada para cambiar posiciones de forma dinámica cuando se está ejecutando el algoritmo.
- Código 50. Trabajo coordinado con otros servidores con comunicaciones asíncronas. Esta opción es para realizar la tarea conjunta con el Koala para recoger la pelota. Esta comunicación tiene una parte asíncrona y se verá en el siguiente punto.

- S6. Se envía 1 byte con código 0 al cliente para indicar que se ha recibido correctamente el código anterior. Si ocurre algún error se envía un "1" al cliente para parar la ejecución actual.
- S7. Recepción de 51 bytes, que contiene información de toda la estructura para el funcionamiento del programa del servidor. Este paquete está encapsulado siguiendo la siguiente estructura, que se muestra en la figura 31.

TIPO	DATOS	DATOS COMUNES	TIPO CONTROL	DATOS CONTROL	FIN (\0)
0	1-8	9-28	29	30-49	50

Figura 31. Paquete de datos

- Tipo. En caso de querer seguir una trayectoria preestablecida se indica el tipo de referencia: 0 (rectangular), 1 (Circular), 2 (Alcanzar una posición final) y 3 (trayectoria obtenida con la cámara cenital con evitación de obstáculos).
- Datos. Se indican los valores del tipo de trayectoria. Estos valores pueden ser:
  - Rectangular. Se indican los valores de los lados (los bytes de 1 a 4 indican el LADO 1 y de 5-8 el LADO 2).
  - Circular. Se indican los valores del radio y de la frecuencia (bytes de 1 a 4 indican el radio y de 5 a 8 la frecuencia).
  - Posición final. Se indica las posiciones X e Y a alcanzar (bytes de 1 a 4 indican la posición X y de 5 a 8 indican la posición Y).
  - Trayectoria. Se indica el tamaño de referencias obtenidas de la trayectoria a seguir y el tamaño del RASTER (bytes de 1 a 4 el tamaño y de 5 a 8 el RASTER). Estos datos son necesarios para pre-calcular la trayectoria en función del periodo de muestreo pues, no tienen porqué coincidir el recorrido de un periodo de muestreo y el tamaño del RASTER. Por ejemplo, si el RASTER (traducido en mm aunque realmente se pasa el

tamaño en píxeles) fuera de 360 mm significa que de la trayectoria obtenida, cada 360 mm se dispone de un nuevo punto; en cambio, si el periodo de muestreo es de 50 ms y la velocidad de referencia del robot es de 100 mm/s en un periodo se ha recorrido 5 mm por lo que no coincide con el RASTER. Para ello hay que precalcular esta trayectoria con todos estos valores.

- Datos comunes. Aquí se encapsulan los siguientes datos, independientemente de la trayectoria y del algoritmo de control.
  - Bytes de 9 a 12. Velocidad de referencia
  - Bytes de 13 a 16. Periodo de muestreo.
  - Bytes de 17 a 20. Posición inicial X del Create.
  - Bytes de 21 a 24. Posición inicial Y del Create.
  - Bytes de 25 a 28. Orientación inicial del robot.
- Tipo de control. Aquí se indica el algoritmo a ejecutar. Puede ser el del punto descentralizado, para trayectorias (valor 0) o el de persecución pura para caminos (valor 1).
- Datos de Control**.** Esta parte depende directamente del valor anterior:

  Para el algoritmo de punto descentralizado los valores son:

  - Bytes de 30 a 33. Punto E (en mm.)
  - Bytes de 34 a 37. Ganancia K_{px}.
  - Bytes de 38 a 41. Ganancia K_{py}.
  - Bytes de 42 a 45. Ganancia K_{vx}.
  - Bytes de 46 a 49. Ganancia K_{vy}.

  Para el algoritmo de persecución pura los valores son:

  - Bytes de 30 a 33. Ganancia K_{rho}.
  - Bytes de 34 a 37. Ganancia K_{alpha}.
  - Bytes de 38 a 41. Ganancia K_{beta}.
  - Bytes de 42 a 45. Ganancia K_{p}.
  - Bytes de 46 a 49. En desuso.
- Fin de paquete. Se indica un '\0' para saber que es el final del paquete.

- S8. Se envía un 0 al cliente para indicar que el proceso ha finalizado correctamente y puede proceder a cerrar el socket e informar al usuario.
- S9. Se cierra el socket creado por la función accept(). Es decir, el socket principal nunca se cierra pues está siempre esperando nuevas conexiones.

CLIENTE.

- C1. Creamos el socket para conectarse con el servidor.
- C2. Realiza la petición al servidor para conectarse.
- C3. Envía el comando del código de la tarea que se quiere ejecutar. Se han explicado en la parte S5 del servidor.

- C4. Recepción de la respuesta del servidor. Si todo ha funcionado correctamente, se recibe un “0” y se sigue con la ejecución del programa. En caso contrario, se recibe un “1” se informa al usuario del error, se cierra el socket y se para la ejecución de la tarea actual.
- C5. El cliente envía los 51 bytes de datos para que el servidor pueda procesar toda la información, como es la ejecución del algoritmo de seguimiento de trayectorias, tipo de trayectoria a generar, etc. Este paquete de datos se ha explicado en la parte S7 del servidor.
- C6. Recibimos el código de finalización del servidor. Los códigos que se pueden recibir son:
    - “0”. Proceso finalizado correctamente.
    - “1”. Proceso finalizado con errores.
    - “2”. Se ha activado una parada de emergencia. Para indicar que la finalización del algoritmo se debe a que el cliente ha realizado una parada de emergencia.
- C7. Finalmente se cierra el socket cliente.

Cuando el programa cliente cierra el socket y comprueba que toda la comunicación ha sido correcta, copia, de forma automática el fichero de resultados que se encuentra residente en el sistema embebido al PC y automáticamente abre el MATLAB para poder examinar estos valores. Simplemente, en el programa servidor, cada vez que se ejecuta la acción de control, se guarda en dicho fichero la posición de la trayectoria donde debería de estar el robot y la posición real donde se encuentra, para luego poder comparar estos resultados.

Como hemos podido observar en los ejemplos anteriores, los resultados obtenidos en estos experimentos son realmente buenos y en lo que respecta al tratamiento de imágenes, posiciones y algoritmos de control de seguimiento de trayectorias funciona perfectamente. Hay que tener en cuenta que, a nivel de comunicación, todas las conexiones, recepciones y envío entre el cliente y el servidor se realizan de forma síncrona, de tal forma que las funciones de recepción son bloqueantes y no hay ninguna logística de asincronismo.

En la siguiente sección, se expone un ejemplo práctico de asincronismo, con las mismas herramientas empleadas en este punto, para conseguir realizar una tarea con dos servidores al mismo tiempo (los dos robots, el Create y el Koala).

3.5 Arquitectura asíncrona. Recoger una pelota.

En este caso, la arquitectura que se tiene se muestra en la figura 32.

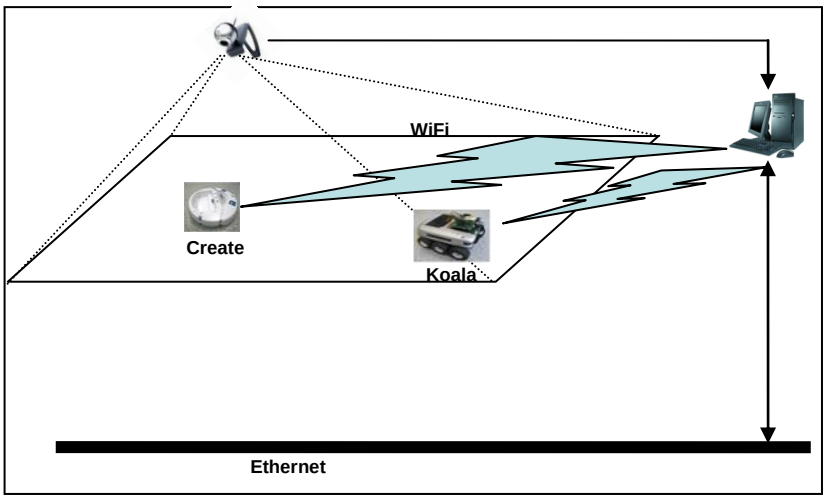


Figura 32. Arquitectura cliente – servidor síncrona-asíncrona

Como hemos podido observar en los puntos anteriores, la forma de trabajar y de realizar la comunicación es simple, debido a que todas las comunicaciones presentes están programadas de forma síncrona. En este apartado, en cambio, vamos a estudiar cómo realizar una aplicación para que estas comunicaciones sean, en parte, asíncronas.

Para llevar a cabo este asincronismo existen funciones tanto para Windows como para Unix que facilitan la programación de esta parte. En este trabajo, debido a la tarea que se quiere realizar, solo ha sido necesario realizar este asincronismo en la parte del cliente. En cualquier caso, este asincronismo puede desarrollarse en cualquier dispositivo, independientemente de que sea cliente o servidor. Más concretamente, el asincronismo se ha realizado mediante la notificación de mensajes en el cliente.

Antes de explicar cómo se ha programado este apartado, hay que tener en cuenta el problema que se quiere resolver. Dos robots, que en este caso, son el Create y el Koala son dos servidores independientes que están a la espera de recibir alguna petición de conexión por parte del cliente. Además, el cliente, sigue siendo un PC que solicita la realización de la tarea conjunta de ambos robots. Estos tres elementos forman una única red del tipo ad-hoc, de tal forma que existe conectividad entre los tres dispositivos. Cuando en el PC se solicita que se recoja una pelota, lo que ocurre, de forma resumida, es lo siguiente (en caso de que ocurra algún error, se cierran los sockets y no se realiza dicha tarea):

1. El PC procesa la imagen de la cámara para detectar las posiciones originales de los robots, así como sus orientaciones y la posición final, que en este caso es la pelota. Hay que tener en cuenta que solo se debe de encontrar una posición inicial para el Create, una posición inicial para el Koala y, finalmente una posición final, la pelota. Si por cualquier motivo, a la hora de segmentar la imagen se detecta más de un componente o no se detecta ninguno, el programa sigue obteniendo imágenes hasta que efectivamente cumpla este requerimiento.

2. Una vez se obtienen las características de las posiciones, en la parte de cliente se realizan dos peticiones de conexión: una para conectarse con el Create y otra para conectarse con el Koala. Si se consigue, entonces vamos al punto 3.

3. Se indican todos los datos necesarios al Create para que se sitúe enfrente de la pelota mediante un algoritmo de seguimiento de trayectoria de punto descentralizado. A la misma vez, se solicita al Koala que vaya detrás de la pelota y el algoritmo que realiza es mediante un control por persecución pura.

4. Hasta el punto 3 todas las comunicaciones son síncronas. A partir de aquí la recepción del cliente la convertimos en asíncrona. El motivo es bien simple. Ninguno de los dos robots puede hacer ninguna tarea hasta que efectivamente los dos lleguen a las posiciones indicadas. Es decir, el Create, no puede empujar la pelota hasta que el Koala esté situado detrás de ésta. Lo mismo ocurre con el Koala, éste no puede recoger la pelota hasta que el Create esté situado en su posición. El problema es aún más grave, pues no sabemos con certeza el orden de llegada de los robots a las posiciones. Recordemos que, en una comunicación síncrona, las recepciones en las comunicaciones son llamadas bloqueantes, por tanto, si hacemos una recepción del Create y llega antes el Koala es posible que se bloqueen las llamadas debido a que el orden es incorrecto. Lo mismo ocurre si fuese a la inversa. Para solucionar este tipo de problema, realizamos en este apartado una recepción asíncrona, de tal forma que cuando cada robot llega a su posición final, éste envía la información al cliente para indicárselo. Este último recibe dicha información mediante un evento por notificación de mensajes. Es decir, el cliente puede estar realizando otras tareas y en el momento que uno de los dos servidores envía información salta un evento para indicarnos que ya hay datos en el socket que pueden ser leídos. Por tanto, hasta que no lleguen los dos robots, la ejecución del programa no puede continuar. Además, en caso de que llegasen los dos robots al mismo tiempo, la información de recepción no se perdería en ningún caso, pues estos mensajes se encolarían por orden de llegada al cliente y se procesan las peticiones en el orden de llegada. En cualquier caso, en este ejemplo práctico, esto no puede ocurrir, pues cada servidor es un socket independiente y cuando recibimos la notificación de que existan datos para ser leídos del socket, son buffers diferentes.

5. Cuando se ha asegurado que ambos robots han llegado a su posición, la comunicación la volvemos a convertir en síncrona; Entonces se envía un nuevo comando al Create para que empuje la pelota una determinada distancia.

6. Al terminar el Create de empujar esta pelota, avisa al cliente y éste envía las ordenes a ambos servidores para que vuelvan a su posición inicial. Si todo ha sido correcto, el Koala lleva consigo la pelota y la deja en su posición.

Aunque el ejemplo aquí visto no sea muy complejo, se puede tener una idea clara de la funcionalidad de este tipo de asincronismo y porqué es necesario emplearlo. Gracias a las herramientas facilitadas tanto en Windows como en UNIX, esta programación es relativamente sencilla y en cualquier momento podemos emplearla. Además, una gran ventaja que presenta, es que un socket no tiene porqué ser síncrono o asíncrono en su totalidad, tal y como se ha visto, sino que en cualquier momento podemos convertir la comunicación en asíncrona y volver a hacerla síncrona. Hay que tener en cuenta que, en cualquier caso, es responsabilidad del programador hacer un uso correcto de estas herramientas, pues si hacemos la comunicación asíncrona, las recepciones ya no se tratan de la misma forma, por lo que al hacer una recepción síncrona el programa devuelve un error, ya que la forma de recibir dicho datos es por notificación de mensajes.

En la figura 33 se muestra un esquema gráfico del funcionamiento de estas comunicaciones

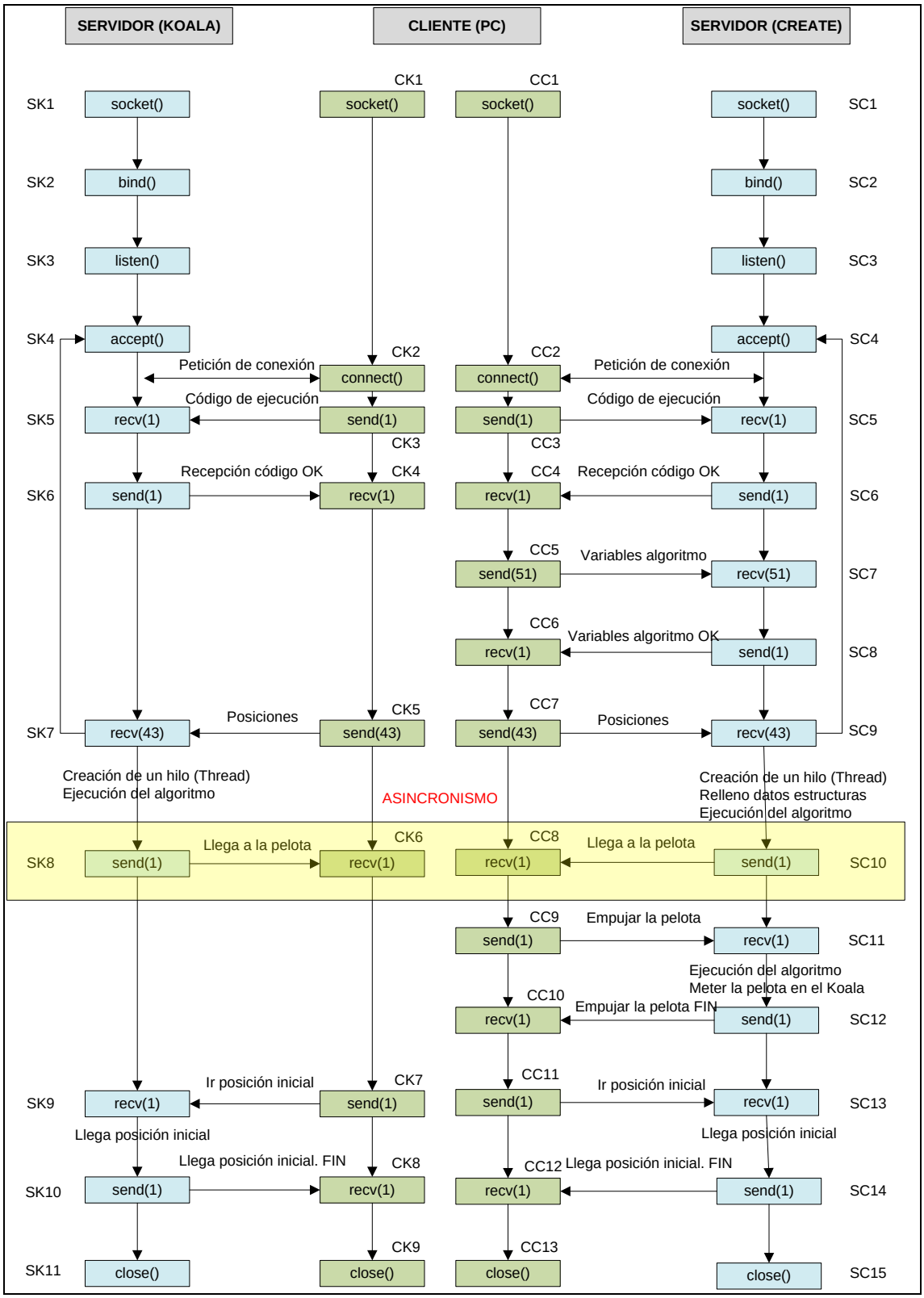


Figura 33. Comunicación de sockets síncrona y asíncrona.

A continuación se resumen brevemente las funciones de la imagen anterior:

SERVIDOR (KOALA)

- SK1. Inicialización del socket.
- SK2. Ponemos el nombre y puerto del socket: Dirección IP: “192.168.253.252”, Puerto = 3491.
- SK3. Prepara al socket para recibir conexiones del cliente.
- SK4. Llamada bloqueante y sirve para esperar peticiones de conexiones de los clientes.
- SK5. En el momento que el cliente ya está conectado, se espera a recibir el código 50, que indica que se quiere realizar el trabajo coordinado con el Create.
- SK6. Si el código es correcto y no hay ningún error envía al cliente un “0” para indicar que puede seguir la ejecución. En caso de que exista algún error envía un “1 y para la ejecución.
- SK7. Se reciben un paquete de 43 bytes con la información de la posición y orientación inicial del Koala, así como la posición y orientación final a lograr. El formato de paquete de datos que se envía se muestra en la imagen 34.

POS. INICIAL X	POS. INICIAL Y	ORIENTACIÓN INICIAL	POS. FINAL X	POS. FINAL Y	ORIENTACIÓN FINAL	FIN(\0)
0-6	7-13	14-20	21-27	28-34	35-41	42

Figura 34. Paquete de datos de posiciones

- SK8. En el momento que el Koala llegue a la posición indicada, envía el código 70 al cliente para saber que ya está preparado para recoger la pelota y está a la espera de que el Create también llegue a su posición.
- SK9. Recibe el código “0” para indicar que vuelva a su posición inicial con la pelota.
- SK10. Envía el código “0” para indicar al cliente que ya ha acabado su tarea.
- SK11. Se cierra el socket.

CLIENTE (CONEXIÓN CON KOALA)

- CK1. Crea un socket para conectarse con el servidor KOALA.
- CK2. Realiza la petición de conexión al KOALA. En caso de que se conecte sigue la ejecución.
- CK3. Envía el código 50 para indicar que se va a realizar la tarea conjunta con el Create.
- CK4. Recibe la aceptación del código. Si se recibe un “0” es que todo es correcto y se sigue con la ejecución. Si ha habido algún error entonces se recibe un “1” y se para la ejecución, cerrando el socket.
- CK5. Envía el paquete de datos que contiene la posición y orientación inicial del Koala, así como la posición final a alcanzar. Lógicamente, antes de cualquier tarea con los sockets, lo primero que se hace es obtener la imagen de la cámara y procesarla, para obtener todas las posiciones y orientaciones necesarias para la ejecución del algoritmo.

- CK6. Recepción asíncrona. En este punto convertimos el socket en asíncrono. De esta forma, el programa sigue con su ejecución y cuando el Koala llegue a la posición final indicada, éste enviará el código correspondiente y en la parte de cliente se recibe una notificación mediante un mensaje, indicando que se puede leer del socket del Koala, pues existe información válida.
- CK7. Se convierte nuevamente el socket en síncrono. Cuando el Create haya empujado la pelota, entonces enviamos el código "0" al servidor Koala para indicar que la pelota ya se ha empujado y, por tanto el robot puede volver a su posición original.
- CK8. Recepción del código de finalización. En este punto se recibe un "0" por parte del Koala, indicando que ya ha terminado su ejecución correctamente. En caso de que exista algún error se recibe un "1" y se cierra el socket antes de tiempo.
- CK9. Finalmente cerramos el socket de conexión con el servidor Koala.

SERVIDOR (CREATE)

- SC1. Inicialización del socket.
- SC2. Ponemos el nombre y puerto del socket: Dirección IP: "192.168.253.254", Puerto = 3490.
- SC3. Prepara al socket para recibir conexiones del cliente.
- SC4. Llamada bloqueante y sirve para esperar peticiones de conexiones de los clientes.
- SC5. En el momento que el cliente ya está conectado, se espera a recibir el código 50, que indica que se quiere realizar el trabajo coordinado con el Koala.
- SC6. Si el código es correcto y no hay ningún error envía al cliente un "0" para indicar que puede seguir la ejecución. En caso de que exista algún error envía un "1 y para la ejecución.
- SC7. Se recibe el paquete de datos para la ejecución del algoritmo y la estructura de datos. El formato de este paquete es el que se explicó en el apartado anterior, figura 31.
- SC8. Se envía al cliente el código "0" para indicar que el paquete de estructura se ha recibido correctamente y puede seguir con la ejecución. En caso de que existe algún tipo de error entonces se envía un "1".
- SC9. Se recibe el paquete de posiciones. El formato de este paquete es el que se muestra en la figura 34, solo que con las posiciones del robot Create y la posición final la que le corresponde a este robot. Es decir, el Create debe de quedarse enfrente de la pelota, a una distancia de 300 mm por delante de la pelota, aproximadamente; en cambio, el Koala debe de situarse detrás de la pelota a una distancia incrementada en 300 mm, aproximadamente.
- SC10. Cuando el Create llega a su posición y está preparado para empujar la pelota, se lo comunica al cliente, enviándole el código "0". En caso de que haya error entonces envía el código "1".

- SC11. El Create recibe el código “0”, que quiere decir, que ya puede empujar la pelota 300 mm para meterla en el Koala. Hay que tener en cuenta que esta acción solo se llevará a cabo cuando los dos robots estén preparados o, lo que es lo mismo, cuando el cliente reciba los datos de que ambos robots han llegado a la posición final de la pelota.
- SC12. El Create envía el código “0” al cliente para indicar que ya ha empujado la pelota y, por tanto está preparado para volver a su posición inicial.
- SC13. Ahora el Create recibe el código “0” por parte del cliente indicándole que ya puede volver a su posición de partida. Como en los casos anteriores, ante cualquier error se recibe el código “1” y se para la ejecución.
- SC14. Cuando el robot llega a su posición inicial envía al cliente el código “0” para informarle de que ha terminado su ejecución correctamente. En caso de que haya habido algún error entonces envía el código “1”.
- SC15. Se cierra el socket de comunicación con el cliente.

CLIENTE (CONEXIÓN CON CREATE)

- CC1. Crea un socket para conectarse con el servidor CREATE.
- CC2. Realiza la petición de conexión al CREATE. En caso de que se conecte sigue la ejecución.
- CC3. Envía el código 50 para indicar que se va a realizar la tarea conjunta con el Koala.
- CC4. Recibe la aceptación del código. Si se recibe un “0” es que todo es correcto y se sigue con la ejecución. Si ha habido algún error, entonces se recibe un “1” y se para la ejecución, cerrando el socket.
- CC5. Envía el paquete de datos del algoritmo y de las estructuras al servidor Create para que éste pueda procesar el algoritmo del punto descentralizado.
- CC6. Recibe el código de aceptación “0” para indicar que el paquete de datos es correcto. Si se recibe un “1” es que hay algún error, se para la ejecución y se cierran los sockets.
- CC7. Se envía el paquete de posiciones al servidor Create. Como se ha mencionado, este paquete contiene toda la información relevante a posiciones iniciales y posición a alcanzar.
- CC8. Recepción asíncrona. En este punto convertimos el socket en asíncrono. Es decir, la recepción de datos ahora es por notificación de mensajes. Cuando el Create llegue a su posición final y se coloque en frente de la pelota, éste enviará la información al cliente, de tal forma que se recibirá una notificación con un mensaje indicando que existe información en el socket servidor del Create que, ha sido enviado y puede ser leído.
- CC9. Se convierte nuevamente el socket en síncrono. Se envía la orden al Create (un “0”) para indicar que ya puede empujar la pelota.
- CC10. Cuando el Create termine de realizar su tarea recibimos el comando “0” para saber que éste ya ha terminado su ejecución. Si ocurre algún error, entonces recibimos un “1” y se para la ejecución, cerrando los sockets.
- CC11. Se envía el comando “0” al servidor Create para indicarle que vuelva a su posición inicial.

- CC12. Recibimos el código “0” para saber que el Create ya ha alcanzado su posición de partida.
- CC13. Finalmente, se cierra el socket de conexión.

Como hemos podido observar con el ejemplo anterior, de una forma sencilla se ha conseguido una comunicación asíncrona cuando así se ha requerido en la ejecución de esta aplicación. No hay que olvidar que, en cualquier momento de la aplicación, por interés de este tipo de aplicación, los sockets se vuelven a programar de forma síncrona, por lo que en la parte de los servidores hay que tener en cuenta este punto para que la funcionalidad de la aplicación conjunta sea correcta.

Para ver cuál es el comportamiento obtenido de la aplicación anterior, supóngase la imagen de partida que se muestra en la figura 35.

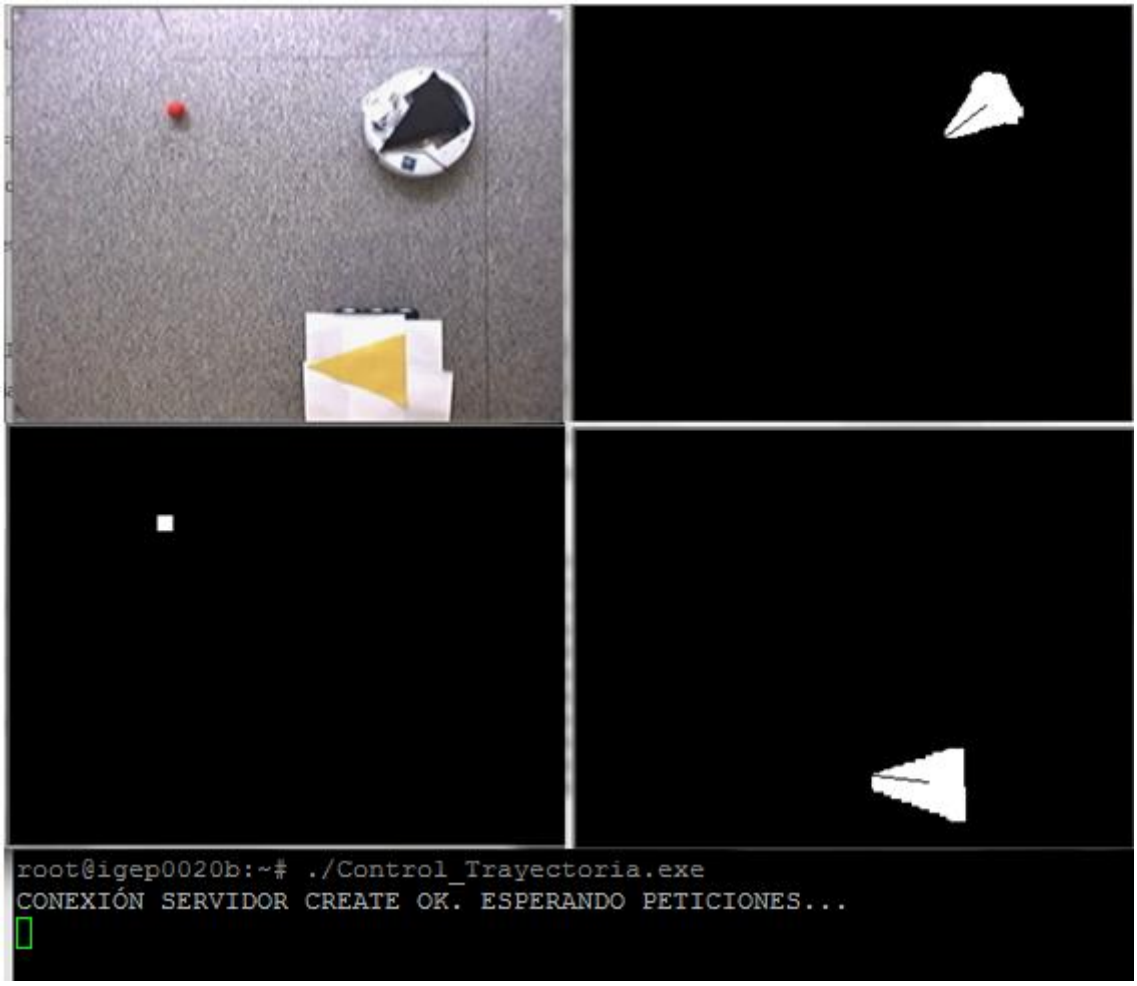


Figura 35. Imagen de partida para la coordinación de los robots.

Como podemos observar en la imagen anterior, la segmentación de la imagen se realiza de forma correcta. En la parte inferior de esta imagen, se muestra la consola de conexión con el sistema embebido, de tal forma que la aplicación está esperando a que el PC cliente se conecte.

Cuando el PC cliente se conecta y envía el código correspondiente, si todo funciona correctamente, el Create y el Koala van a su posición final y el resultado se muestra en la figura 36.

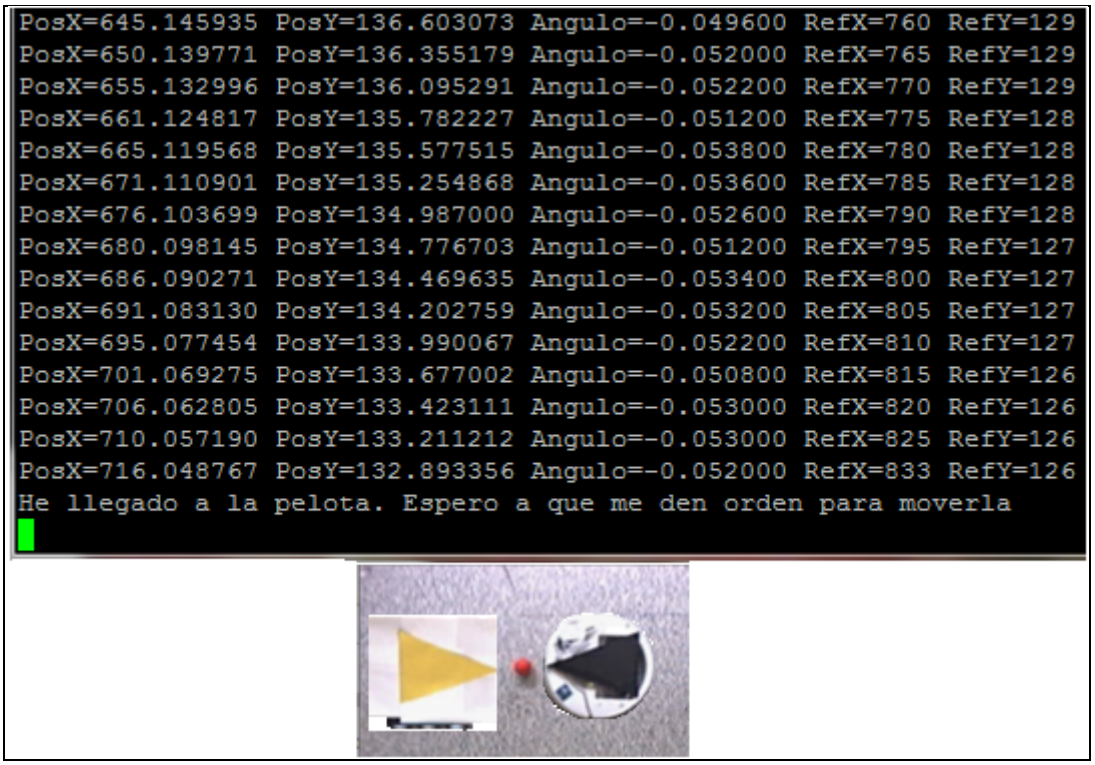


Figura 36. Los dos robots llegan a su posición final

En la imagen anterior, el programa del servidor Create muestra que ha sido el primero en llegar y está esperando a recibir la orden de mover la pelota. Justo cuando están los dos preparados (de la imagen 34, la figura inferior) el Create empuja la pelota y el Koala la recoge, volviendo ambos a su posición inicial, tal y como se muestra en la figura 37.

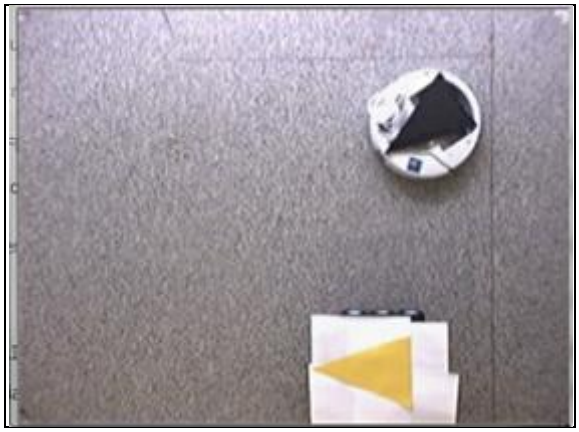


Figura 37. Los dos robot llegan a su posición de partida y el Koala dispone de la pelota.

Por otro lado, tal y como se ha comentado en punto anteriores, el sincronismo de los sockets se puede realizar tanto para sistemas operativos Windows como UNIX. .

En Windows y con la herramienta “Microsoft Visual Studio” la forma de indicar que un socket es asíncrono es mediante la función:

int WSAAsyncSelect(SOCKET *s*,HWND *hWnd*,unsigned int *wMsg*,long *lEvent*), donde:

- s. Es el socket de conexión.
- hWnd. Identificador de la ventana que recibirá el mensaje cuando ocurra un evento en la red.
- wMsg. Mensaje que recibe cuando ocurre el evento.
- lEvent. Sirve para indicar en qué eventos queremos realizar este asincronismo. Existen multitud de eventos pero los más importantes son:
    - FD_READ. Establece la notificación de mensaje para las lecturas
    - FD_WRITE. Establece la notificación de mensajes para las escrituras.
    - FD_CONNECT. Establece la notificación de mensajes para las peticiones de conexión de los clientes.
    - FD_ACCEPT. Establece la notificación de mensajes para las aceptaciones de conexión de los servidores.
    - FD_CLOSE. Establece la notificación de mensajes para cuando los sockets se cierran.

La función devuelve 0 si no se produce ningún error. En cualquier otro caso, la función devuelve el código de error que se ha producido.

Para este ejemplo solo nos ha interesado realizar la parte de recepción de forma asíncrona, con notificación mediante mensajes. En este caso, en la variable “lEvent” se ha indicado FD_READ.

Por otro lado, el nombre del mensaje indicado en wMsg también debemos de asociarlo con alguna función para que, cuando salte este evento se ejecute la función indicada.

Si la programación en lugar de realizarla en Windows y con las herramientas que dicho sistema operativo nos facilita, se realiza bajo UNIX, entonces las funciones utilizadas para establecer el socket asíncrono son las siguientes:

```
fcntl(sd, F_SETFL, O_ASYNC | O_NONBLOCK);

fcntl(sd, F_SETOWN, getpid());
```

La primera instrucción establece que las operaciones de entrada y de salida del socket “sd” sean no bloqueantes, del tipo asíncrono. Después, mediante la segunda instrucción lo que se hace es capturar una señal del tipo SIGIO para saber cuando el socket ha cambiado, más concretamente cuando ha ocurrido un envío o recepción de dicho socket. Como podemos observar, en ambos casos la forma de trabajar es similar, sólo que en este último caso, en lugar de mensajes se realiza mediante señales.

4. Conclusiones.

De una forma relativamente sencilla, se ha podido realizar una arquitectura cliente-servidor con varios dispositivos implicados. Como se ha dicho al principio de este documento, no necesariamente deben de ser robots móviles, puede ser cualquier elemento que disponga de un sistema operativo y que soporte la programación de sockets. Lógicamente, si se quiere una comunicación inalámbrica, también debe de tener una tarjeta Wi-Fi para poder realizar la comunicación.

Estos experimentos se pueden extrapolar a cualquier entorno y sirven de base para realizar tareas más complejas, como las vistas en la sección 2.1.2.

Respecto a los resultados obtenidos de los algoritmos de control para el seguimiento de trayectorias, también cabe mencionar que los resultados han sido muy buenos. Se pueden realizar varias mejoras para eliminar los errores cometidos en cada uno de los puntos de esta trayectoria, por ejemplo, mediante el uso de algún tipo de filtro para la corrección de la trayectoria, en cada uno de estos puntos, aunque como hemos podido observar, no ha afectado en gran medida para resolver el problema propuesto.

Por otro lado, en lo que respecta a la segmentación de las imágenes y el tratamiento de éstas, ha resultado sencillo la utilización de MATLAB y de OPENCV para trabajar estas imágenes y así poder obtener los puntos característicos. Sí que es cierto que dependiendo de la luz incidente sobre el área de trabajo de la cámara, los valores de los píxeles cambian ligeramente, por lo que en función de esta iluminación, puede verse afectado la extracción de características. Para ello, una alternativa hubiera sido extraer los componentes HSV de la imagen y trabajar con ellos, pues en principio, se vería menos afectado por la iluminación. En cualquier caso, no se ha implementado esta opción debido a que la utilización de las componentes RGB de cada píxel ha dado buenos resultados y no se han tenido grandes problemas debido a esta luminosidad.

Se puede decir que, a la hora de implementar estas arquitecturas y la programación de toda esta comunicación, es donde menos problemas han surgido. Todo lo referente al tratamiento de las imágenes, generación de trayectorias, evitación de obstáculos, seguimiento de trayectorias, creación de la librería con MATLAB y creación de la librería para el robot Create es donde más conflictos y dificultades han surgido.

Después de realizar varios experimentos para comprobar el comportamiento del robot Create de IRobot a la hora de seguir una trayectoria, se puede afirmar que los resultados son admisibles, siempre y cuando los parámetros del algoritmo del “punto descentralizado” para el seguimiento de la trayectoria sean los adecuados. Además, como se ha comentado en puntos anteriores, la lectura del ángulo girado por el robot, es realmente mala, puesto que se comete un gran error, que se va acumulando a lo largo de la trayectoria. Por ejemplo, supongamos que se quiere realizar el seguimiento del rectángulo que se ha visto en puntos anteriores, en cada esquina el robot debe de girar 90^o, si se comete un error, aproximadamente de 2^o, entonces, al final del cuadrado se tiene un error de 8^o-10^o. Por ello, se ha comprobado que la estimación del ángulo del robot funciona mucho mejor que la lectura directa de éste.

En cambio, la lectura de la distancia que ha recorrido el robot funciona muy bien, pues como se ha visto, el resultado de este seguimiento de la trayectoria es realmente óptimo.

Cabe destacar, que se han implementado otros tipos de algoritmos para el seguimiento del camino, como es la "persecución pura". Después de realizar varias pruebas, el seguimiento por "punto descentralizado" funciona mucho mejor que el de persecución pura. Estos algoritmos de control están relacionados directamente con el robot. Es decir, en el Koala, por ejemplo, se ha utilizado en algoritmo de "persecución pura" y funciona perfectamente. Es cierto que hay que emplear un tiempo considerable en encontrar los valores de las ganancias y parámetros que hacen funcionar perfectamente el algoritmo. Por lo que en algunos casos el algoritmo de seguimiento de trayectoria funciona mejor en unos robots móviles que en otros y el de persecución pura funciona mejor en otros robots móviles. En este último algoritmo, además, se puede indicar con qué orientación queremos que el robot llegue a un destino. Esta orientación, en cambio, no se puede indicar en el algoritmo de "punto descentralizado". Para la parte práctica no ha sido necesario indicar al Create la orientación para llegar a un punto destino.

En definitiva, tal y como se ha comentado, la complejidad de todo este trabajo reside en las tareas adicionales implementadas para poder resolver el problema propuesto. La implementación y diseño de la arquitectura, así como la programación de todos los dispositivos implicados, es relativamente sencillo, pudiendo añadir o quitar cualquier dispositivo de la arquitectura en un momento dado, de una forma relativamente fácil. Es responsabilidad del diseñador y programador, tener en cuenta los diferentes dispositivos y hacer un buen uso de la sincronía en toda la comunicación. El tratamiento de una comunicación síncrona o asíncrona difiere en gran medida en toda la arquitectura y en toda la programación ya que si, se convierte el socket de uno de los dispositivos en asíncrono y no se prepara la recepción del otro dispositivo para atender esta asincronía, todo el funcionamiento de la arquitectura será anómala y los resultados que se tendrán, estarán muy alejados de los esperados. Hay que tener en cuenta que cuando se realiza un asincronismo, independientemente de la forma que se programe, el elemento que recibe la información del socket, de alguna forma tiene que saber que existe información válida, leerla y extraerla, pues si no, se quedará información en el buffer y el comportamiento será completamente inestable.

5. Trabajos futuros.

Por un lado, se puede ampliar la arquitectura diseñada, añadiendo más elementos en la comunicación. Por ejemplo, se pueden añadir los robots LEGO para que realicen alguna tarea conjunta con el Create y el Koala en tiempo real. Por citar algún ejemplo, mientras el Create persigue al Koala, que los LEGO vayan bordeando al Create. Como vemos, en esta tarea es muy importante el tiempo de procesamiento de las imágenes, pues cada poco tiempo debe de tomarse una nueva imagen para calcular las nuevas posiciones de todos los robots y actualizarlas en los diferentes robots / servidores.

Otra ampliación que se puede realizar en la arquitectura presente, es la comunicación por una VPN. Es decir, de lo que se trata es de poner uno o varios clientes fuera de esta red y que soliciten dicha cooperación desde un lugar diferente a donde se encuentra los robots. La forma de conectar ambas redes sería mediante la creación de una VPN y así poder crear esta red de forma virtual y, mediante este mecanismo poder solicitar el trabajo deseado.

Por otro lado, se puede conectar un pequeño brazo robótico al Create de IRobot para que, en lugar de empujar la pelota, la coja con dicho brazo y la introduzca en el Koala o en otra zona deseada (por ejemplo una canasta). Existen diferentes brazos robóticos en el mercado pero también se puede crear algún tipo de dispositivo similar de forma “casera”.

Finalmente, cabe mencionar que el Create, aparte de llevar el sistema embebido no dispone de ningún elemento de procesamiento para conseguir algún tipo de autonomía. En este sentido, se puede instalar una cámara web en el Create y aprovechar el DSP de dicho sistema embebido para el procesamiento de las imágenes capturadas por la cámara. De esta forma, se pueden programar diferentes algoritmos para seguir una trayectoria o llegar a un punto destino, teniendo en cuenta los obstáculos, tanto estáticos como en movimiento. Además, también se podría instalar algún tipo de sensor sonar o similar para añadir una segunda metodología de evitación de obstáculos.

6. Referencias.

- iRobot: Educational & Research Robots. Disponible en http://store.irobot.com/shop/index.jsp?categoryId=3311368.
- ISEE - IGEP Platform Homepage. Disponible en: http://www.igep-platform.com/.
- The open source, cross platform Code::Blocks. Disponible en: http://www.codeblocks.org/.
- Centro de desarrollo de Microsoft para consultas de "Microsoft Visual Studio 2008". Disponible en http://msdn.microsoft.com/es-es/default.aspx.
- Mathworks. Disponible en http://www.mathworks.com/.
- OpenCV. Disponible en http://opencv.willowgarage.com/documentation/c/index.html.

7. Anexos.

7.1 Anexo A. Creación de una librería con MATLAB.

En este punto se va a explicar cómo crear una librería en MATLAB para poderla usar en un proyecto de "Visual Studio 2008 (C++)". Existen muchas opciones para realizar esta librería en función del tipo de librería que queramos crear. Es decir, se pueden crear librerías para C#, .NET, C++,… y por ello existen multitud de opciones. Aquí, solo se van a explicar los pasos necesarios para crear una librería del tipo .LIB, con sus correspondientes ficheros .DLL, .cpp y .h. Estos ficheros los creará automáticamente MATLAB. La versión de MATLAB que se ha utilizado es 2008b de 32 bits. A continuación se explica los pasos a seguir.

1. Selección del compilador. Lo primero que se debe de hacer es seleccionar el compilador en MATLAB. Para ello ejecutamos "mbuild –setup". Después de ejecutar esta opción, nos pregunta si queremos localizar los compiladores para MATLAB. Escogemos la opción "n" NO para que muestre todos los compiladores instalados en el PC. Hay que tener en cuenta que antes de realizar todo este proceso, previamente tiene que estar instalado el "Microsoft Visual Studio 2008". En función de los diferentes compiladores que tengamos en la máquina, nos ofrece diferentes opciones, tal y como se ilustra en la imagen A1.

```
>> mbuild -setup
Please choose your compiler for building standalone MATLAB applications:

Would you like mbuild to locate installed compilers [y]/n? n

Select a compiler:
[1] Lcc-win32 C 2.4.1
[2] Microsoft Visual C++ 6.0
[3] Microsoft Visual C++ .NET 2003
[4] Microsoft Visual C++ 2005
[5] Microsoft Visual C++ 2005 Express Edition
[6] Microsoft Visual C++ 2008

[0] None

Compiler:
```

Figura A1. Compiladores instalados en el PC

Según la imagen anterior, debemos de seleccionar el compilador correspondiente de "Microsoft Visual C++ 2008" que, en este caso, es la opción 6. Una vez establecido el compilador, debemos de introducir la ruta de donde tengamos "Microsoft Visual Studio 2008" instalado. Esto es necesario introducirlo correctamente, pues MATLAB necesita de una serie de ejecutables y ficheros para poder crear esta librería. En este caso, la ruta donde se tiene instalado "Microsoft Visual Studio 2008" es en "C:\Program Files (x86)\Microsoft Visual Studio 9.0". Si todo ha ido bien, debe de aparecer una ventana igual que la que se muestra en la imagen A2.

```
Compiler: 6

The default location for Microsoft Visual C++ 2008 compilers is C:\Program Files\Micr
but that directory does not exist on this machine.

Use C:\Program Files\Microsoft Visual Studio 9.0 anyway [y]/n? n
Please enter the location of your compiler: [C:\Program Files\Microsoft Visual Studio

Please verify your choices:

Compiler: Microsoft Visual C++ 2008
Location: C:\Program Files (x86)\Microsoft Visual Studio 9.0

Are these correct [y]/n? y

********************************************************************************
  Warning: Applications/components generated using Microsoft Visual Studio
           2008 require that the Microsoft Visual Studio 2008 run-time
           libraries be available on the computer used for deployment.
           To redistribute your applications/components, be sure that the
           deployment machine has these run-time libraries.
********************************************************************************

Trying to update options file: C:\Users\Fran\AppData\Roaming\MathWorks\MATLAB\R2008b\
From template:              C:\PROGRA~2\MATLAB\R2008b\bin\win32\mbuildopts\msvc90comp

Done . . .

>>
```

Figura A2. Resultado de indicar la ruta del compilador.

2. Creación de la librería. Una vez establecido correctamente el compilador que se va a usar para crear la librería debemos de generar dicha librería. Para ello, debemos de situarnos en el directorio donde tengamos los ficheros .m que queremos incluir en la librería. En este caso, se ha creado una función en un fichero .m, la cual emplea otros ficheros .m necesarios para la generación de trayectorias y otras tareas. Cuando generamos esta librería, los ficheros auxiliares se incluyen de forma automática para poder hacer uso de la función que hemos realizado. El comando necesario para crear esta librería es: “mcc -g -W cpplib:libdetecta -T link:lib detecta”, donde:

- -g. Depurar. Incluye información de los símbolos de depuración.
- -W cpplib:libdetecta. Funciones “Wrapper”. Especifica el tipo de ficheros que debe de crear el compilador. En este caso va a generar una librería con el código fuente incluido en un fichero .cpp. Lógicamente, también creará un fichero .h de encabezado para poder usar las funciones implementadas en el fichero .cpp. “libdetecta” es el nombre que queremos que tengan los ficheros generados.
- -T link:lib. Indica que va a generar ficheros para C/C++. Además, la forma de linkar estos ficheros será mediante una librería compartida del tipo DLL. Es decir, creará con esta opción dos ficheros adicionales que sirven para linkar las llamadas en el programa donde incluyamos esta librería. Una DLL que es necesaria que se encuentre en el mismo lugar que el ejecutable de la aplicación donde vayamos a usar esta librería; y un

fichero .LIB necesario para incluir en el proyecto de la aplicación y así sepa "linkar" las llamadas usadas de esta librería.

- detecta. Este es el fichero .m que queremos usar para crear la librería. Es decir, si en este fichero disponemos de dos funciones, la librería creará dos funciones para poder usarlas en la aplicación donde incluyamos estos ficheros. Además, se puede indicar más de un fichero .m y MATLAB ya se encarga de generar todas las funciones de todos los ficheros indicados.

Con esta opción, por tanto, se crean varios ficheros con el nombre indicado (en este caso libdetecta). Los ficheros que debemos incluir en el proyecto para poder usar las funciones de de la librería son: libdetecta.cpp, libdetecta.h, libdetecta.lib y libdetecta.dll.

7.2 Anexo B. Manual de usuario del programa "cliente".

Este programa es el que se ha desarrollado para el PC cliente y ofrece diferentes alternativas de funcionamiento. En la figura B1 se ilustra el aspecto del programa.

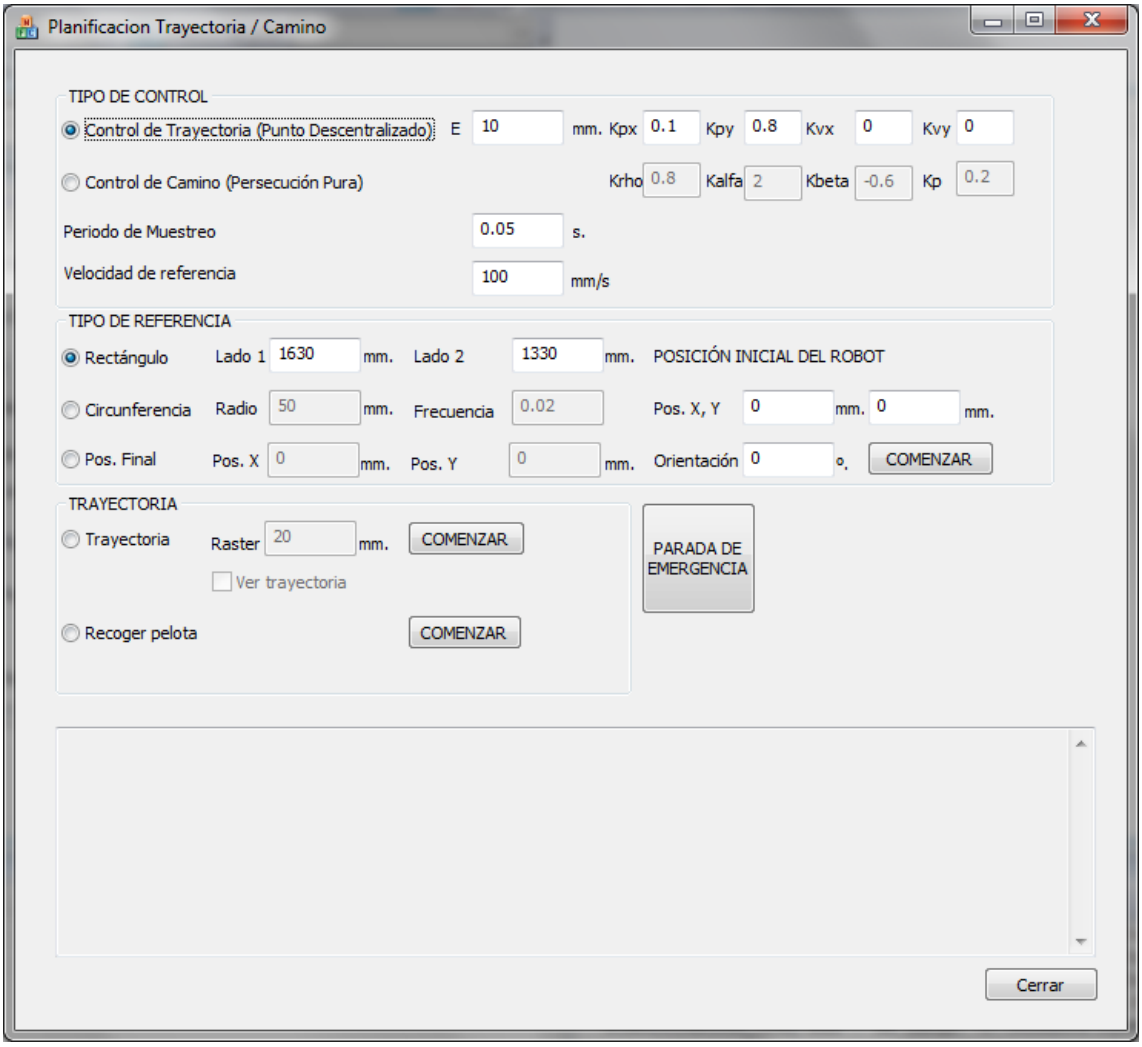


Figura B1. Programa cliente.

Como podemos observar en la imagen anterior, en la parte superior de la ventana se indica que tipo de algoritmo se quiere aplicar para el seguimiento de la trayectoria o camino. Si indicamos que se quiera realizar un seguimiento de la trayectoria mediante el algoritmo de punto descentralizado debemos indicar los valores de las siguientes variables:

- K_{px}. Ganancia para indicar mayor o menor importancia a los cálculos relacionados con la coordenada X.
- K_{py}. Ganancia para indicar mayor o menor importancia a los cálculos relacionados con la coordenada Y
- K_{vx}. Ganancia para indicar mayor o menor importancia la velocidad de referencia, en relación con el eje X.
- K_{vy}. Ganancia para indicar mayor o menor importancia la velocidad de referencia, en relación con el eje Y.

Si se selecciona el control de camino por persecución pura se deben indicar los siguientes valores:

- K_{rho}. Ganancia relacionada con la distancia existente entre el punto actual y el punto objetivo.
- K_{alpha}. Ganancia relacionada con el ángulo del robot. Para girar con mayor o menor fuerza.
- K_{beta}. Ganancia relacionada con el ángulo actual del robot. Indica la importancia del ángulo. Es decir, de forma indirecta está relacionada también con la fuerza de giro del robot.
- K_p. Ganancia relacionada con las velocidades de las ruedas izquierda y derecha del robot. Realmente con esta ganancia conseguimos que las velocidades de ambas ruedas sean más o menos agresivas para conseguir el punto objetivo.

Independientemente del algoritmo seleccionado, también debemos de indicar los valores de las siguientes variables (esto valores son utilizados por ambos algoritmos):

- Periodo de muestreo (T). En esta variable indicamos el periodo de muestreo en segundos. Hay que tener en cuenta que este periodo es el usado en el seguimiento de la trayectoria y, por tanto, es una variable sumamente importante. Periodos grandes generan trayectoria y controles muy separados en el tiempo, por lo que las acciones de control son realmente agresivas y es posible que el robot no actúe de forma adecuada. Periodos pequeños hacen funcionar mejor al robot, ya que se obtiene más puntos para las trayectorias y la acción de control se calcula muchas más veces, pero hay que tener cuidado, pues periodos pequeños pueden hacer que el algoritmo no calcule a tiempo las acciones de control y el comportamiento del robot sea anómalo. Periodos de 30 a 50 ms hacen funcionar los algoritmos de forma óptima. Como se ha comentado, este periodo es usado para establecer cada cuanto tiempo se va a calcular la acción de control y aplicarla sobre cada una de las ruedas. Por otro lado, este valor es usado para crear las diferentes trayectorias. Por ejemplo, supongamos que se quiere realizar una línea recta de 100 mm, de tal forma que la posición inicial es X = 0, Y = 0 y la posición final es X = 100, Y = 0. Si el periodo de muestreo es de 50 milisegundos y la velocidad de referencia del robot es de 100 mm/s entonces podemos calcular esta trayectoria con una simple regla de 3. Es decir, si para recorrer 100 mm hace falta 1 segundo, entonces en 50 ms cuanto ha recorrido el robot en línea recta. A los 50 ms si el robot va en línea recta en el eje X, entonces se debería de encontrar en la posición X = 5, Y = 0. De esta forma calculamos todas las trayectorias. Así, cada 50 milisegundos obtenemos la posición actual del robot y la comparamos con la posición de donde debería de estar. Con estos datos se obtienen los valores de control y se aplican sobre las dos ruedas.
- Velocidad de referencia. Esta es la velocidad en mm/s que se quiere conseguir con el robot. Experimentalmente, se ha demostrado que velocidades de 100 mm/s a 150 mm/s obtiene resultados muy buenos para periodos de 50 ms.

Después de indicar todos los valores antes mencionados, se debe de indicar cuál es la tarea que se quiere realizar. Existen tres posibilidades:

- Referencia. Con esta opción lo que se hace es generar una trayectoria rectangular, circular o lineal en función del tipo que se haya seleccionado. En caso de que sea rectangular, indicamos los valores de los lados del rectángulo. En caso de que sea circular se indica el radio de esta circunferencia y la frecuencia de velocidad. Este último valor está relacionado con las veces que se desea realizar esta trayectoria circular. Finalmente, si se indica que es del tipo "Posición final", lo que se hace es una trayectoria lineal para conseguir la posición indicada. En cualquiera de estos tres casos se debe indicar, de forma manual, cual es la posición inicial del robot, en mm y la orientación de éste en grados.
- Trayectoria. Generación de una trayectoria libre de obstáculos. Con esta opción debemos de indicar cuál es el tamaño del RASTER en mm. Esta funcionalidad es la encargada de obtener la imagen de la cámara y generar una trayectoria libre de obstáculos. La posición inicial es un triángulo de color negro (situado encima del Create), la posición final algún objeto de color rojo y, finalmente los obstáculos son de color azul, tal y como se ha explicado en punto anteriores.
- Recoger Pelota. Esta tarea es la que ejecuta la orden a ambos robots para la coordinación de éstos con el objetivo de recoger una pelota. Es en este punto donde se presenta la solución implementada de forma asíncrona para conseguir este objetivo. Hay que tener en cuenta que los robots deben de estar conectados de forma inalámbrica con el PC cliente y formar una única red entre los tres componentes.

En cualquier caso, independientemente del tipo de algoritmo a ejecutar, así como la referencia a seguir, se puede pulsar sobre el botón "PARADA DE EMERGENCIA", el cuál envía un comando al Create para que inmediatamente se pare y pare la ejecución de cualquier algoritmo.

7.3 Anexo C. Moviendo el Create de IRobot con el OMNIA I900 de Samsung.

Para desarrollar aplicaciones para este dispositivo, debemos de tener instalado previamente en el PC los siguientes componentes:

- Microsoft Visual Studio 2008
- ActiveSync (facilitado con el software del móvil o, en su defecto, se puede bajar de la página oficial de Samsung: http://www.samsung.com/es/consumer/detail/support.do?group=mobile-phone&type=mobile-phones&subtype=touch-screen&model_nm=SGH-I900&disp_nm=I900%20Omnia&language=&cate_type=all&dType=D&mType=SW&vType=&prd_ia_cd=01012000&model_cd=SGH-I900XKMFOP&menu=download)
- SDK de Windows Mobile 6 necesaria para poder realizar programas que se ejecuten en Windows Mobile 6 y 6.1 http://www.microsoft.com/downloads/details.aspx?familyid=06111a3a-a651-4745-88ef-3d48091a390b&displaylang=en
- SDK de SAMSUNG OMNIA I900 para poder acceder a los valores de los acelerómetros del móvil. http://innovator.samsungmobile.com/down/cnts/toolSDK.detail.view.do?platformId=2&cntsId=4980

El aspecto del programa se muestra en la imagen C1.

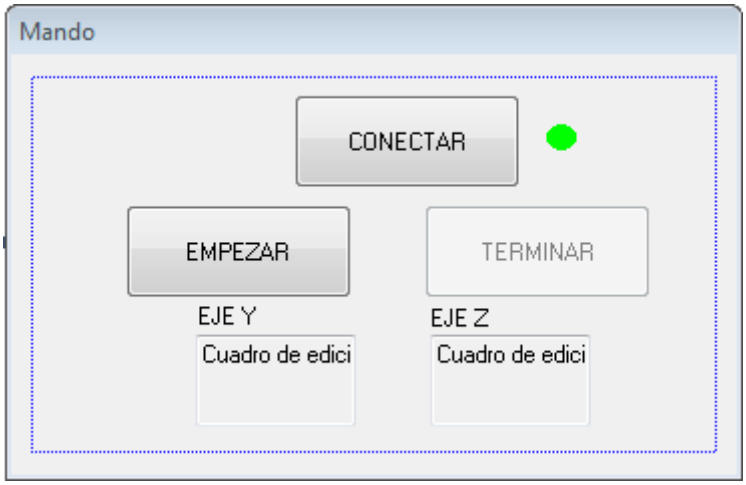


Figura C1. Programa mando.exe.

Para poder ejecutar el programa correctamente lo primero que debemos de hacer es configurar la pantalla del móvil de forma horizontal. Este punto es muy importante ya que si no, los valores que obtenemos de los acelerómetros no son los correctos y el comportamiento del robot no es el esperado.

Este programa sólo consta de una ventana, que es la mostrada en la figura 2. Para conectarse y desconectarse vía Bluetooth con el Robot pulsamos al botón Conectar. Cuando la luz está en verde significa que se ha conectado correctamente con el robot; en cambio, si la luz está roja es que el móvil no está conectado con el robot.

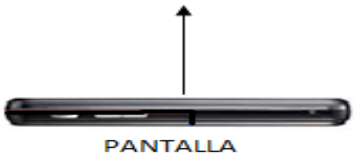
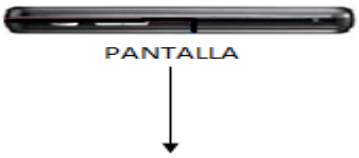
Una vez ya estamos conectados con el robot (se ha pulsado el botón CONECTAR y la luz está verde) simplemente pulsamos sobre el botón EMPEZAR y el programa se pone en marcha hasta que pulsemos sobre el botón TERMINAR. Lógicamente no podemos terminar la ejecución del programa si éste no ha empezado (por eso en la imagen aparece el botón TERMINAR deshabilitado) y lo mismo ocurre a la inversa, es decir, no podemos empezar la ejecución del programa si éste ya está ejecutándose y no se ha parado.

La librería facilitada con la SDK de SAMSUNG nos permite acceder a las 3 coordenadas (x, y, z) de los movimientos del móvil, aunque para nuestro caso sólo hace falta obtener los valores de 2 coordenadas. Esto tiene sentido, pues el robot se mueve sobre un plano horizontal y los valores que podemos obtener son las coordenadas x e y del robot (NO es necesario tener en cuenta la orientación del robot). Por otro lado, no debemos de confundir los valores de Y, Z que aparecen en la ventana de la aplicación con los valores del robot respecto de un eje de coordenadas. Los valores que se muestran en este formulario, simplemente son los valores que se recogen de los acelerómetros del móvil para tener una visualización de éstos.

Según la documentación de la SDK de SAMSUNG los valores que se obtienen de los acelerómetros van desde -2 a 2 unidades, aunque después de realizar varios experimentos con el dispositivo se puede afirmar que estos valores realmente van desde -1.1 a 1.1 unidades, tanto para el caso de la Y como el de la Z. En la imagen C2 se muestra de forma gráfica los ejes de los acelerómetros.

Figura C2. Los 3 ejes de coordenadas de los valores obtenidos por los acelerómetros.

En la tabla C1 se muestra de una forma más detallada como varían los valores de Y, Z obtenidos en función de cómo se mueve el móvil.

Posición del móvil	Valor Y	Valor Z
	0 < Y < 1.1	--
	-1.1 < Y < 0	--
	--	0 < Z < 1.1
	--	-1.1 < Z < 0
	-1.1	0
	1.1	0

PANTALLA 	0	1.1
 PANTALLA	0	-1.1

Tabla C1. Resumen de los movimientos del OMNIA.

Como podemos observar en la tabla anterior, los movimientos del móvil son realmente simples y son similares a como funcionaría un volante de un automóvil. Es decir, en función de los movimientos del móvil el robot deberá hacer lo siguiente:

- 0 < Y < 1.1. El robot debe de girar hacia la derecha (sentido horario)
- -1.1 < Y < 0. El robot debe de girar hacia la izquierda (sentido anti horario)
- 0 < Z < 1.1. El robot debe de avanzar hacia delante.
- -1.1 < Z < 0. El robot debe de retroceder hacia atrás.

Lógicamente estos movimientos también están relacionados con la magnitud del valor. Por ejemplo, si el valor de Y es 0.1 el robot debe de girar hacia la derecha; en cambio, si vale 1, también debe de girar hacia la derecha pero con mayor velocidad. En otras palabras, el signo de la coordenada Y nos va a indicar si hay que girar hacia la derecha o izquierda y la magnitud con qué velocidad se quiere hacer. Por otro lado, el signo de la coordenada Z nos va a indicar si el robot debe de ir hacia adelante o atrás y la magnitud con qué velocidad se desea realizar el movimiento.

Una vez hecha la similitud anterior de los movimientos del móvil con el robot es fácil encontrar una fórmula que relacione la velocidad de giro por un lado, y la velocidad de avance / retroceso por otro.

Los valores que debemos de calcular, por tanto, son las velocidades de las ruedas izquierda y derecha que debemos de introducir en el robot para transformar el movimiento del móvil en el movimiento lineal y angular del robot. La forma de calcular la velocidad de la rueda izquierda se muestra en la ecuación C1 y de la rueda derecha en la ecuación C2.

$$Vel\_Izq = (ACELERACION * Z) + (RADIO * Y) \quad (C1)$$

$$Vel\_Der = (ACELERACION * Z) - (RADIO * Y) \quad (C2)$$

Donde la ACELERACION es una constante que indica la importancia de la componente Z y el RADIO es otra constante que indca la importancia de la componente Y. Los valores de estas constantes están relacionadas directamente con la velocidad máxima que soporta el robot. El valor de ACELERACION que hemos usado para los experimentos es de 300 y el del RADIO de 200, lo que significa que el robot presentará una mayor velocidad lineal que angular.

Para poder comprender mejor los resultados de esta fórmulas, se presenta en la tabla C2 los resultados de estas velocidades para diferentes valores obtenidos de los acelerómetros, en lo que respecta al eje Y, Z que, junto con la tabla 1 de movimientos del móvil, se puede ver de forma fácil que los movimientos que realiza el robot son los indicados por el móvil.

Y	Z	Vel_Izq (mm/s)	Vel_Der (mm/s)	Comportamiento
-1	0	-200	200	El robot gira sobre sí mismo hacia la izquierda.
1	0	200	-200	El robot gira sobre sí mismo hacia la derecha.
0	-1	-300	-300	El robot retrocede hacia atrás a 300 mm/s de forma lineal.
0	1	300	300	El robot avanza a 300 mm/s de forma lineal.
0.5	0.5	250	50	El robot avanza girando hacia la derecha
0.5	-0.5	-50	-250	El robot retrocede girando hacia la izquierda
-0.5	-0.5	-250	-50	El robot retrocede girando hacia la derecha
-0.5	0.5	50	250	El robot avanza girando hacia la izquierda
0	0	0	0	El robot no se mueve

Tabla C2. Valores de las velocidades obtenidos para diferentes valores de los acelerómetros. ACELERACION = 300 y RADIO = 200.

Hay que tener en cuenta que cuando las velocidades de las ruedas son positivas el robot se mueve hacia adelante y con velocidades negativas hacia atrás. De esta forma si una velocidad es positiva y la otra negativa con la misma magnitud, el robot girará sobre sí mismo a una determinada velocidad.

Parea ver los resultados reales de los exprimentos ver el video Mov_manual.avi adjunto con este documento.

7.3.1 Aspectos de programación del OMNIA de SAMSUNG.

Para poder hacer uso de las llamadas a los acelerómetros se debe de incluir el encabezado <smiAccelerometer.h> al proyecto. Además, debemos de incluir la librería SamsungMobileSDK_1.lib al proyecto para que se pueda usar la librería de SAMSUNG.

Para poder obtener el vector de valores de los acelerómetros del móvil simplemente llamamos a la función SmiAccelerometerGetVector(&vector), la cual nos almacena los valores en la variable vector que es del tipo SmiAccelerometerVector. En dicho vector ya tenemos los valores capturados de los acelerómetros del móvil.

Por otro lado, también debemos de tener en cuenta que cuando el programa empieza a ejecutarse, éste está en un bucle infinito, el cual se ejecuta cada 50 ms para volver a obtener los valores de las coordenadas Y, Z y calcular nuevamente las velocidades. Hay que tener en cuenta que es necesario lanzar esta ejecución en un hilo, pues sino no habría forma de parar el programa.

Finalmente, respecto a la conexión del Bluetooth, como se ha mencionado anteriormente, ésta se ha realizado de forma automática. Es decir, no es necesario realizar ninguna consulta para conectarse con el robot puesto que esta conexión ya se realiza cuando pulsamos el botón CONECTAR. Hay que tener en cuenta que tanto el dispositivo Bluetooth del móvil como el del robot deben de estar libres para poder realizar la conexión. Además, como esta conexión se realiza de forma automática, necesitamos saber cuál es el identificador de la antena Bluetooth del robot. Este identificador es único para cada antena y en caso de tener que cambiar la antena del robot se debería de obtener el nuevo código y sustituirlo.

Para cualquier otra consulta a nivel de programación, se puede consultar la documentación **Mando.html** que se encuentra dentro de la carpeta \Mando-HTML adjunta a este documento, donde se explica detalladamente la programación de esta parte del proyecto.

7.4 Anexo D. Programas para la realización de la parte práctica.

A continuación se exponen todos los programas necesarios para la realización de los diferentes programas que se han explicado en este documento.

- Desarrollo de la aplicación cliente
  - Microsoft Visual Studio 2008 (ó 2010).
  - MATLAB 2008b (cualquier otra versión puede funcionar, aunque hay que tener en cuenta que la metodología para la creación de la librería puede cambiar).
  - OpenCV 2.1. Para poder usar dichas librería en "Visual Studio" y realizar todo el tratamiento de imágenes.
- Desarrollo de la aplicación para el dispositivo móvil OMNIA i900 de SAMSUMG.
  - Microsoft Visual Studio 2008 (ó 2010).
  - ActiveSync 4.5 o una versión posterior, para la comunicación del dispositivo con el PC.
  - SDK de Windows Mobile 6 para desarrollar aplicaciones en Windows Mobile 6.1 o versiones anteriores.
  - SDK de SAMSUNG OMNIA I900 para poder acceder a los valores de los acelerómetros y cualquier otro sensor o valor de este dispositivo móvil.
- Desarrollo de la aplicación servidor (del robot Create)
  - Code::Blocks. Para desarrollar aplicaciones que se ejecuten en el sistema embebido IGEPv2.
  - Compilador "arm-none-linux-gnueabi" para poder compilar con "Code::Blocks" y obtener un ejecutable que funcione en la tarjeta IGEPv2.
- Comunicación con el sistema embebido IGEPv2.
  - Putty. Programa para poder conectarse de forma inalámbrica o mediante cable con la tarjeta IGEPv2 y así poder copiar ficheros y arrancar los ejecutables.
  - Pscp. Programa para poder copiar ficheros desde el sistema embebido al PC y viceversa.

7.5 Anexo E. Repositorio adjunto con la documentación.

Junto a este documento existen diferentes carpetas y ficheros que facilitan al usuario el entendimiento de las diferentes partes implicadas en esta arquitectura, código fuente de cada uno de los programas, así como videos demostrativos de los resultados obtenidos. En la tabla E1 se explica el significado de cada uno de estos ficheros.

Ruta	Descripción
CODEBLOCKS/manual_en.pdf	Manual de usuario del Code::Blocks.
IGEPv2/Manual.pdf	Manual de referencia del sistema embebido IGEPv2.
IRCR_DLL-HTML/IRCR_DLL.html	Documentación del código fuente en formato HTML de la DLL del Create. Ejecutar IRCR_DLL.html.
IROBOT CREATE/*	Diferentes ficheros *.pdf con documentación del Create de IRobot.
Mando-HTML/Mando.html	Documentación del código fuente en formato HTML del programa creado para el OMNIA de SAMSUNG para mover el Create. Ejecutar Mando.html.
SAMSUNGSDKOMNIA/SDK.pptx	Información relevante para usar los acelerómetros del OMNIA de SAMSUNG y otros elementos.
VIDEOS/PC/*	Diferentes videos ilustrativos que muestran el resultado de diferentes ejecuciones, es decir, lo que sucede en el PC cliente y en el servidor Create.
VIDEOS/RESULTADOS/*	Diferentes videos ilustrativos que muestran los resultados de diferentes ejecuciones. Concretamente, la ejecución del programa del móvil y la ejecución del trabajo coordinado de ambos robots. Estarán disponibles para el día de la presentación.
INTERFAZ_CLIENTE-HTML/Interfaz.html	Documentación del código fuente en formato HTML de la interfaz desarrollada para el PC cliente.
SERVIDOR_CREATE-HTML/ServCr.html	Documentación del código fuente en formato HTML del programa desarrollado para el sistema embebido IGEPv2 que actúa de servidor en el Create.

Tabla E1. Contenido adicional adjunta con la documentación