UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DESARROLLO DE CONTROLADORES PARA LA COORDINACIÓN DE ROBOTS MÓVILES MEDIANTE COMUNICACIONES INALÁMBRICAS

Trabajo Fin de Grado en Ingeniería Electrónica Industrial y Automática

Autora: Emima Ioana Jiva

Tutora: Marina Vallés Miquel

Cotutor: Ángel Valera Fernández

Escuela Técnica Superior de Ingeniería del Diseño

Universidad Politécnica de Valencia

Valencia, Julio 2018

AGRADECIMIENTOS

En primer lugar, quisiera agradecer a mi familia por apoyarme y motivarme en seguir adelante en todos estos años.

También quiero agradecer a mis tutores Marina Vallés Miquel y Ángel Valera Fernández por acompañarme y ayudarme a lo largo de todo este camino en la realización del proyecto.

Además, también quisiera agradecer a mis amigas Melisa, Debo y Eveline por los momentos divertidos que hemos pasado juntas y que me han ayudado a seguir adelante en los momentos más difíciles.

RESUMEN

Para el desarrollo de aplicaciones de robótica móvil resulta de gran importancia contar con un sistema eficiente de comunicación inalámbrica que permita monitorizar la situación en que se encuentra, modificar sus pautas de comportamiento o, como es el caso de este trabajo, permita establecer comunicación con otros robots móviles para coordinarse en la realización de una tarea más compleja. Dependiendo del protocolo de comunicaciones soportado por el robot móvil la tipología de red que pueda implantarse será distinta. En este sentido, en los últimos años se ha ido extendiendo el uso de las redes WiFi para proporcionar comunicaciones inalámbricas en los robots móviles, lo cual permite una tipología de red de tipo malla en la que cualquier nodo de red puede comunicar con cualquier otro. Esta ventaja se podría usar para muchas aplicaciones, pero en el caso de este trabajo se va a demostrar su uso en el caso del desarrollo de controladores para robots móviles que necesiten una coordinación entre ellos para llevar a cabo una determinada tarea. En el caso del presente trabajo se va a usar como plataforma robótica Lego Mindstorms EV3. Esta versión de Lego Mindstorms ofrece la posibilidad de comunicar vía WiFi el "ladrillo" que hace las veces de controlador, a diferencia de las versiones anteriores que únicamente disponían de comunicación Bluetooth. Esta nueva versión permite además tener un sistema operativo basado en Linux (ev3dev) en una tarjeta SD y que el controlador de Lego arranque con este operativo. Esta última característica permite que la programación de los controladores sea similar a la que se haría en cualquier tarjeta empotrada de las que se pueden utilizar para estas aplicaciones y que ésta pueda realizar con distintos lenguajes de programación. En el caso de este trabajo, dicha programación se realizará mediante lenguaje C de manera que se han programado en este lenguaje los distintos algoritmos de control, la comunicación necesaria entre ellos y se ha organizado el código en cada robot con una estructura multihilo para que la comunicación no influya en las prestaciones de control. Los distintos algoritmos programados serían fácil mente extrapolables a otras plataformas que trabajen con el sistema operativo Linux.

Palabras clave: robótica, robótica móvil, Lego Mindstorm, comunicación inalámbrica, programación multitarea, sockets, hilos

RESUM

Per al desenvolupament d'aplicacions de la robòtica mòbil resulta de gran importància comptar amb un sistema eficient de comunicació sense fil que permeta monitoritzar la situació a la qual es troba, modificar les seues pautes de comportament o, com és el cas d'aquest treball, establir comunicació amb altres robots mòbils per a coordinar-se en la realització d'una tasca més complexa. Depenent del protocol de comunicacions suportat per el robot mòbil la tipologia de xarxa que puga implantar-se serà diferent. En aquest sentit, en els últims anys s'ha anat estenent l'ús de les xarxes Wi-Fi per a proporcionar comunicacions sense fil en el robots mòbils, la qual cosa permet una tipologia de xarxa de tipus malla en la qual qualsevol node de xarxa pot comunicar amb qualsevol altre. Aquest avantatge es podria usar per a moltes aplicacions, però en el cas d'aquest treball es va a demostrar el seu ús en el desenvolupament de controladors per a robots mòbils que necessiten una coordinació entre ells per a dur a terme una determinada tasca. En el cas del present treball s'utilitzarà com a plataforma robòtica Lego Mindstorms EV3. Aquesta versió de Lego Mindstorms ofereix la possibilitat de comunicar via Wi-Fi el "brick" que fa de controlador, a diferència de les versions anteriors que únicament disposaven de comunicació Bluetooth. Aquesta nova versió a més permet tenir un sistema operatiu basat en Linux (ev3dev) dins d'una targeta SD i que el controlador de Lego treballe amb aquest operatiu. Aquesta última característica permet que la programació dels controladors siga pareguda a la qual es faria en qualsevol targeta encastrada les quals es poden utilitzar per a aquestes aplicacions i que aquesta puga realitzar-se amb diferents llenguatges de programació. En el cas d'aquest treball, la programació es realitzarà utilitzant llenguatge C de manera que s'han programat en aquest llenguatge els diferents algorismes de control, la comunicació necessària entre ells i s'ha organitzat el codi en cada robot amb una estructura multitasca per a que la comunicació no influïsca en les prestacions de control. Els diferents algoritmes programats serien fàcilment extrapolables a altres plataformes que treballen amb el sistema operatiu Linux.

Paraules clau: robòtica, robòtica mòbil, Lego Mindstorm, comunicació sense fil, programació multitasca, sockets, fils

ABSTRACT

For the development of mobile robotic applications it is very important to have an efficient wireless communication that allows you monitor the situation in which you are, modify behavior guidelines or, as in this case is, allow communication with others mobile robots to coordinate for doing a difficult task. Depending on communication protocol supported by the mobile robot, the type of network that can be implemented will be different. In this sense, in recent years the use of Wi-Fi networks has been extended to provide us wireless communications in mobile robots, which allows a type of mesh network in which any network node can communicate with each other. This advantage can be used for many applications, but in this case it will be demonstrate for the development of controllers for mobile robots that need coordination between them to do a task. In this case it will be used Lego Mindstorms EV3. This version of Lego Mindstorms offers the possibility of communicating via Wi-Fi the "brick" that acts as a controller, unlike older versions that only had Bluetooth communication. This new version allows having an operating system based on Linux (ev3dev) on an SD card and the Lego controller starts with this operating system. This last feature allows the programming of controllers to be similar to what would be done in any embedded card that can be used for this applications and that this can be done with different programming languages. In this case, the programming will be done using C language, so the different control algorithms have been programmed in this language, the communication between them and the code has been organized with a multithread structure so that the communication does not influence the control features. The algorithms would be extrapolated to other platforms which work with Linux operating system.

Keywords: robotics, mobile robotics, Lego Mindstorms, Wireless communication, multitask, sockets, threads.

ÍNDICE DEL CONTENIDO

ÍNDICE DE LAS IMÁGENES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado en Ingeniería Electrónica Industrial y Automática

DOCUMENTO 1 : MEMORIA

Escuela Técnica Superior de Ingeniería del Diseño

Universidad Politécnica de Valencia

Valencia, Julio 2018

1. INTRODUCCIÓN

1.1. INTRODUCCIÓN Y MOTIVACIÓN

Cada vez más, la tecnología está presente en la sociedad y va adquiriendo importancia en todos los sectores de la vida. Uno de los objetivos más importantes del avance de la tecnología es el deseo de realizar máquinas que sustituyan el trabajo pesado del ser humano, ya que de esta forma se puede ahorrar tiempo y dinero en el desarrollo de una actividad.

Otro de los sectores que está en pleno desarrollo son las comunicaciones inalámbricas ya que el hecho de no utilizar cables para hacer conexiones entre diferentes dispositivos, trae consigo una infinidad de ventajas y presenta muy pocas desventajas.

En este proyecto se ha intentado aprovechar estos últimos avances de la tecnología y se ha deseado diseñar aplicaciones que ayuden a realizar los trabajos pesados en los que haga falta más de un robot, empleando comunicaciones inalámbricas.

1.2. OBJETIVOS Y CONSIDERACIONES

El objetivo principal de este proyecto es la realización de varias aplicaciones en las que tengan que cooperar varios robots Lego EV3 para realizar una misma tarea, comunicándose entre ellos sin utilizar cables, es decir, inalámbricamente.

Para la conexión SSH del PC con el ladrillo del robot se ha utilizado la conexión vía Wi-Fi que es una innovación de esta generación de Lego ya que las anteriores versiones no presentaban esta oportunidad.

Para la realización de la conexión entre los ladrillos que cooperan entre ellos, se utilizan los sockets. Por tanto se debe tener en cuenta a lo largo de todo el proyecto que la creación de sockets se va a realizar en todas las aplicaciones ya que de esta forma se podrán enviar y recibir datos los robots entre ellos. Así pues cada robot tendrá la información sobre la posición del otro de manera que puedan llegar a sincronizarse entre ellos y lleven a cabo la tarea en la que se necesita la ayuda de los dos robots.

Una de las cosas que también se debe tener en consideración en este proyecto, es que la función que utiliza el socket para recibir datos es una función bloqueante, es decir, que el sistema se queda parado hasta que esa función reciba algo. Esta propiedad influye en el mal funcionamiento de la aplicación y por tanto se debe evitar que el sistema se quede bloqueado debido a esto. Una de las soluciones que se ha considerado correcta es la utilización de los hilos, aunque también podrían existir otras soluciones válidas.

Para la programación del ev3dev existe la posibilidad de utilizar bastantes lenguajes, pero en este caso se ha decidido utilizar el C, debido a su facilidad a la hora de usarse y a la eficiencia que este lenguaje presenta.

2. PARTE TEÓRICA

2.1. INTRODUCCIÓN A LA ROBÓTICA

Hace ya miles de años, los seres humanos empezaron a construir máquinas que imitaban partes del cuerpo humano con el fin de que estas puedan hacer los trabajos más tediosos que antes de que estas máquinas existan, los debía hacer un ser humano. Fue así como en el año 1923 un escritor checo, Karel Capek, acuñó el término "robot" en su obra dramática Rossum's Universal Robots a partir de la palabra checa *robota*, que significa trabajos forzados y que fue traducida al inglés como robot.

Hoy en día, según la RAE, podemos definir el concepto robot como: "Máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas solo a personas". Años más tarde también apareció el término de robótica definiendo a la ciencia que estudia los robots.

Con el paso del tiempo, la robótica ha ido adquiriendo una mayor importancia en la sociedad y se ha introducido en diferentes sectores. Por ello en la actualidad podemos hacer una clasificación de las distintas clases de robots:

- **Poli articulados:** Este grupo está formado por robots que tienen distintas formas y configuraciones, pero todos ellos son sedentarios. Esto significa que son robots estructurados para moverse en un determinado espacio de trabajo, según uno o más sistemas de coordenadas y con un número limitado de grados de libertad.
- **Móviles:** este tipo de robots tienen capacidad de desplazarse siguiendo un camino guiándose a través de la información recibida por sus sensores.
- **Androides:** intentan reproducir, total o parcialmente la forma y el comportamiento del ser humano.
- **Zoomórficos:** tienden a imitar a diversos seres vivos. Se pueden agrupar a su vez en dos categorías: caminadores y no caminadores.

Fig. 2.1 Robot poli articulado [1]

Fig. 2.2 Robot móvil [2]

Fig. 2.3 Robot androide [3]

Fig. 2.4 Robot zoomórfico [4]

Para realizar este proyecto se van a utilizar robots móviles. Los robots móviles emplean diferentes tipos de locomoción mediante ruedas que les proporciona características y propiedades diferentes en cuanto a la eficiencia energética, dimensiones, cargas útiles y maniobrabilidad. A continuación se explicarán brevemente las características de locomoción más comunes en robots móviles.

- **Ackerman (tipo coche):** se utiliza en vehículos de cuatro ruedas convencionales. El sistema se basa en dos ruedas motrices en la parte trasera y dos ruedas directrices en la delantera. Esta configuración nos permite un reparto de peso no tan exigente ya que disponemos de cuatro puntos de apoyo. El problema es que al trazar una curva cada una de las ruedas directrices describe una circunferencia de distinto radio.

  Para evitar esto, Ackerman, estableció que era necesario que la prolongación de los brazos de dirección se corte a la altura del eje trasero. De esta forma la rueda interior en una curva girará más que la exterior y no será arrastrada. El mayor problema de este tipo de locomoción es la maniobrabilidad

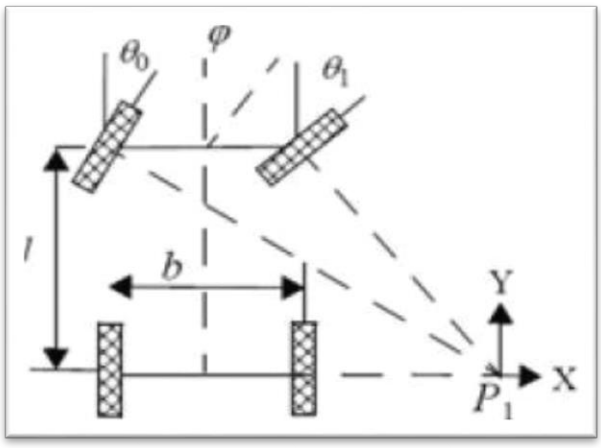

Fig. 2.5 Configuración Ackerman [5]

- **Configuración triciclo:** Se basa en una rueda delantera que sirve tanto para tracción como para el direccionamiento. El eje trasero, que tiene dos ruedas laterales, es pasivo y las ruedas se mueven libremente. El centro de gravedad suele desplazarse cuando el vehículo se desplaza por una pendiente, de esta manera se pierde tracción, por lo que es conveniente situar el centro de gravedad cerca del suelo.

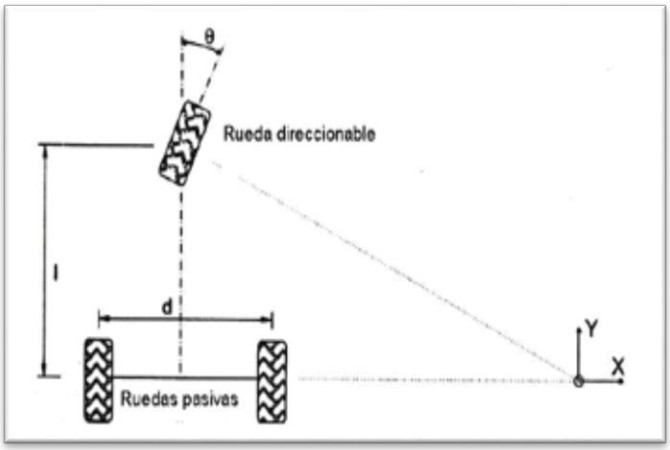

Fig. 2.6 Configuración triciclo [6]

- **Configuración omnidireccional:** Se utilizan tres ruedas directrices y motrices. Este tipo de configuración tiene tres grados de libertad, por tanto puede realizar cualquier movimiento y posicionarse en cualquier posición y en cualquier orientación. No tiene ningún tipo de limitación cinemática.

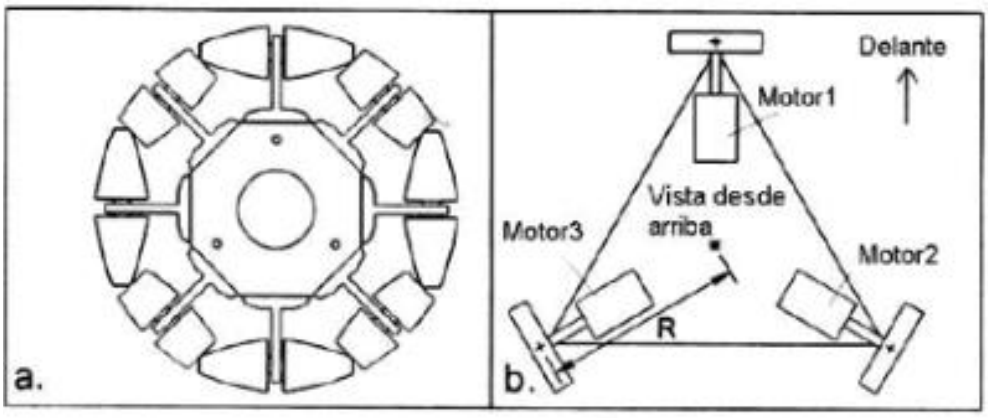

Fig. 2.7 Configuración omnidireccional [7]

- **Configuración síncrona:** Tres o más ruedas se conectan entre sí mediante coronas de engranajes o correas concéntricas y todas tienen un sistema de tracción, de esta manera se consigue que todas apunten a la misma dirección y giren a la misma velocidad. Uno de los principales inconvenientes es que necesita una compleja sincronización para poder funcionar correctamente.

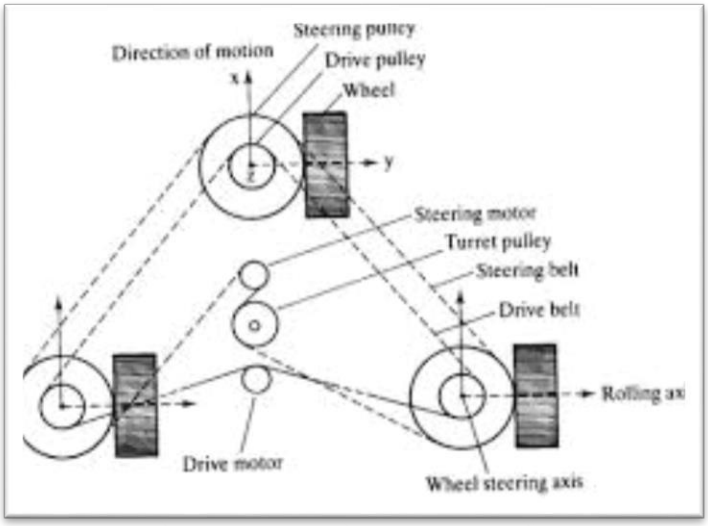

Fig. 2.8 Configuración síncrona [8]

- **Tracción diferencial**: Dos ruedas motrices independientes se sitúan a lo largo de un eje perpendicular a la dirección del motor. Cada una de estas ruedas depende de un motor, por lo que al establecerse velocidades distintas en cada una de ellas, se podrán crear trayectorias y giros. En la mayoría de los casos resulta complicado mantener el equilibrio del robot utilizando únicamente estas dos ruedas, por tanto se añade una tercera a la cual se le denomina "rueda loca", que gira libremente y no depende de ningún motor.

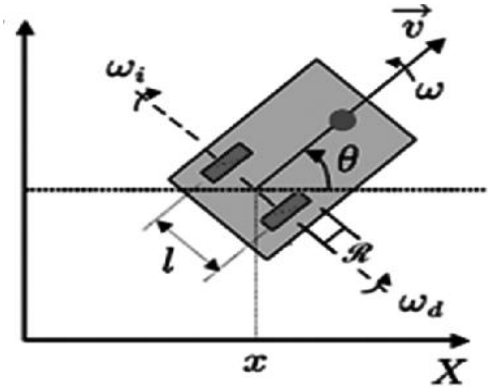

Fig. 2.9 Tracción diferencial [9]

Dada la sencillez que presenta la tracción diferencial a la hora implementar controladores, tanto para la posición como para la velocidad, en este proyecto se trabajará con robot móvil de tracción diferencial.

2.2. GENERACIÓN DE TRAYECTORIAS A PARTIR DE MODELOS MATEMÁTICOS

Para controlar un robot resulta imprescindible crear la trayectoria con los puntos por donde debe pasar el robot para que desde una posición inicial llegue a la posición final deseada. Estas trayectorias se suelen definir a partir de unos puntos indicados por el usuario y en función del método utilizado se pasa por esos puntos o no. Pero aunque se pase o no por esos puntos, cada método asegura la llegada del robot al punto final deseado.

2.2.1. MÉTODOS DE GENERACIÓN DE CURVAS

Principalmente existen 2 métodos para generar curvas a partir de un conjunto de puntos:

- **Interpolación:** la trayectoria generada utilizando los puntos que se indican pasan obligatoriamente por estos puntos. Un ejemplo muy claro de este método es la Spline Cúbica Natural

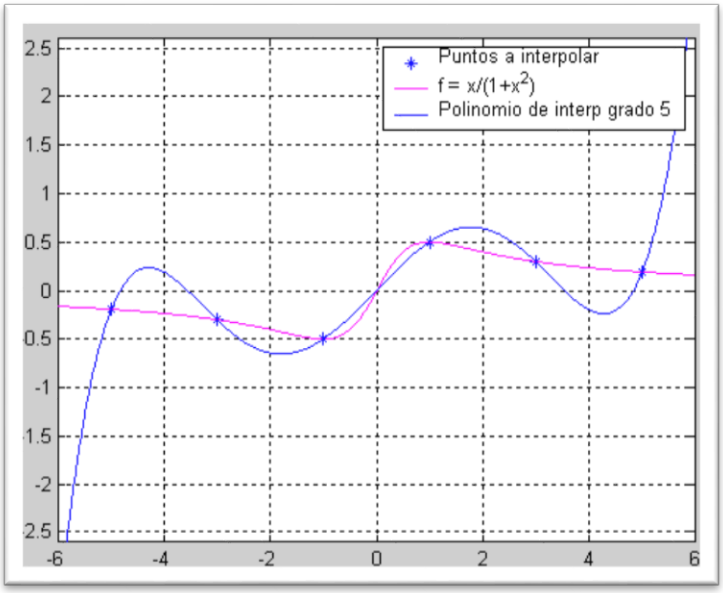

Fig. 2.10 Interpolación de puntos [10]

- **Aproximación:** la trayectoria que se genera no pasa exactamente por los puntos indicados sino que se crea una trayectoria que pasa relativamente cerca de esos puntos. Aunque sí que llega al punto final deseado. El ejemplo más conocido de este tipo de método don las curvas de Bézier.

2.2.2. SPLINE CÚBICA NATURAL

Se trata de interpolación segmentaria cúbica. Esto significa que si tenemos j+1 puntos, para generar las curvas habrá j segmentos. Lo que quiere decir que el primer segmento interpola los dos primeros puntos, el segundo interpola los dos siguientes y así sucesivamente. El polinomio generado al realizar la interpolación entre los dos puntos, es de tercer grado.

$$s(t) = a + bt + ct^2 + dt^3$$

Representado en forma matricial quedaría de la siguiente forma:

$$\begin{bmatrix} s_x \\ s_y \end{bmatrix} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}$$

Donde t solo puede comprender valores entre 0 y 1

Para explicar este método hay que tener en cuenta que se debe cumplir una serie de condiciones:

En el caso de los puntos interiores (los que no están al principio o al final de la trayectoria), el final de un segmento debe coincidir con el principio del siguiente. Además la primera y la segunda derivada del punto final de un segmento deben ser iguales a la primera y segunda derivada del punto inicial del segmento siguiente.

Otra condición que afecta al inicial y final de la trayectoria es que en ambas partes la segunda derivada debe ser igual a 0.

Con el fin de obtener los puntos de la trayectoria se necesita averiguar los valores de "*a,b,c,d"* de la matriz (hay que tener en cuenta que los parámetros "*a,b,c,d* "son distintos en el caso de la x y en el caso de la y, en este caso trataremos la obtención de *"a,b,c,d"* relacionados con la y pero los relacionados con la x se averiguan de la misma forma solo sustituyendo la y por la x)

Así se tiene que:

$$s(t) = a + bt + ct^2 + dt^3$$

$$s'(t) = bt + 2ct^1 + 3dt^2$$

$$s''(t) = 2c + 6dt^1$$

Considerando t=0 y teniendo en cuenta las condiciones explicadas anteriormente se tiene que:

$$a = y_i$$

$$b = D_i$$

$$c = 3(y_{i+1} - y_i) - 2D_i - D_{i+1}$$

$$d = 2(y_i - y_{i+1}) + D_i + D_{i+1}$$

La D significa derivada.

Teniendo en cuenta que en los puntos comunes de los distintos segmentos, las derivadas son iguales, se despeja y se llega a la siguiente conclusión:

$$3(y_{i+1} - y_i) = D_{i-1} + 4D_i + D_{i+1}$$

Organizándolo en matrices se tiene que:

$$\begin{bmatrix} 2 & 1 & 0 & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & 4 & 1 & 0 & \dots & \dots & \dots & \dots & \dots \\ 0 & 1 & 4 & 1 & 0 & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_n \end{bmatrix} = \begin{bmatrix} 3(y_1 - y_0) \\ 3(y_2 - y_0) \\ 3(y_3 - y_1) \\ \dots \\ 3(y_n - y_{n-1}) \end{bmatrix}$$

Donde se tiene que despejar la matriz que contiene las derivadas y de esta forma sustituir los valores obtenidos en las expresiones de "*a,b,c,d* " para averiguar el valor de estos parámetros. Una vez obtenidos estos parámetros se puede sacar los distintos polinomios de tercer grado que representa a los segmentos de la interpolación.

Debido a la dificultad a la hora de realizar el cálculo de la inversa, este método resulta más difícil de aplicar y además hace que el tiempo de cómputo sea elevado, lo que puede llevar a un retraso de la respuesta del robot. Por este motivo, en el presente proyecto se ha decidido utilizar el método de Bézier para la generación de trayectoria.

.

2.2.3. CURVAS DE BÉZIER

En este tipo de curvas lo que se pretende es conseguir la unión de dos puntos utilizando una curva. Los puntos que están en el interior de la trayectoria (los que no son el extremo) se utilizan para hacer que la curva que une los dos puntos se "acerque "a ellos pero sin pasar por ellos.

Así pues, la curva de Bézier queda definida de la siguiente forma:

$r(t) = \sum_{i=0}^{n} P_i * B_i^n\,(t)$

Donde t, solo puede tomar valores entre 0 y 1.

P_i Son los puntos que se suministran para realizar la curva.

$B_i^n(t)$ Se conocen como funciones de ponderación y determinan el peso de los puntos a la hora de formar la curva.

Estas funciones de ponderación tienen la siguiente expresión:

$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$

Si se desarrolla esta expresión para 3 puntos (son los puntos que se va a utilizar para la generación de la trayectoria en una de las aplicaciones del proyecto) quedaría de la siguiente forma:

Para i=0, pertenece el P_0

$$\binom{2}{0} t^0 (1-t)^2 = \frac{2!}{0!\,(2-0)!} * t^0 * (1-t)^2 = (1-t)^2$$

Para i=1, pertenece el P_1

$$\binom{2}{1} t^1 (1-t)^1 = \frac{2!}{1!\,(2-1)!} * t^1 * (1-t)^1 = 2t(1-t)^1$$

Para i=2, pertenece el P_2

$$\binom{2}{2} t^2 (1-t)^0 = \frac{2!}{2!\,(2-2)!} * t^2 * (1-t)^0 = t^2$$

Así pues la ecuación final de la curva de Bézier utilizando 3 puntos quedaría de la siguiente forma:

$$\boldsymbol{r(t) = P_0(1-t)^2 + P_1 2t(1-t)^1 + P_2 t^2}$$

Esta es la expresión que se utilizará posteriormente para crear la referencia de la trayectoria en una de las aplicaciones.

2.3. CONTROL DE POSICIÓN EN ROBOTS MÓVILES

2.3.1. CONTROL POSICIÓN POR PUNTO DESCENTRALIZADO

Como ya se ha explicado anteriormente los robots móviles diferenciales presentan dos ruedas que giran de forma independiente. De modo que controlando la velocidad de las dos ruedas se puede ir a la posición deseada y por tanto controlar también la trayectoria que sigue el robot.

En este proyecto se ha decidido realizar el control utilizando el algoritmo del punto descentralizado. El control se realiza a partir de la posición y velocidad de un punto que se encuentra separado una distancia ***e*** desde el eje de tracción del robot.

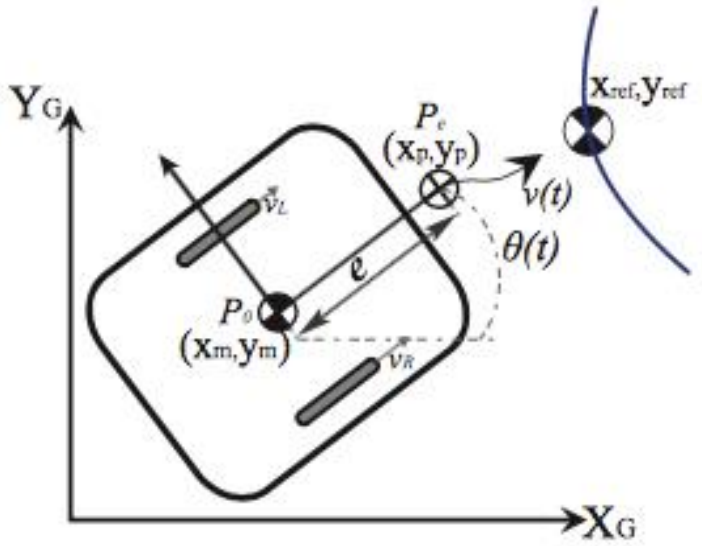

Fig. 2.11 Puntos importantes en el control por punto descentralizado [15]

De una manera resumida el funcionamiento de este algoritmo se explica a continuación: Utilizando las ecuaciones de la cinemática directa del robot, a partir de la velocidad lineal y angular de las ruedas se puede hallar la posición en x, la posición en y, además del ángulo en el que se encuentra el robot en ese momento. Teniendo las referencias de la trayectoria que se quiere describir (x_{ref}, y_{ref}) se realiza la resta con la posición real del robot y se halla el error entre la posición donde se quiere estar y la posición donde se esta realmente. Aplicando el control cinemático correspondiente se puede obtener la velocidad necesaria en los ejes para alcanzar la posición deseada $(\dot{x}_p, \dot{y}_p)$. Haciendo uso del modelo de la cinemática inversa, a partir de estas velocidades se puede obtener las velocidades de referencia de las ruedas necesarias para el control de la trayectoria. Obteniendo la velocidad angular de referencia y sabiendo la velocidad angular del robot, utilizando un regulador (en este caso de tipo PID) se calcula una acción de control que se le aplica al robot con el fin de controlar su posición.

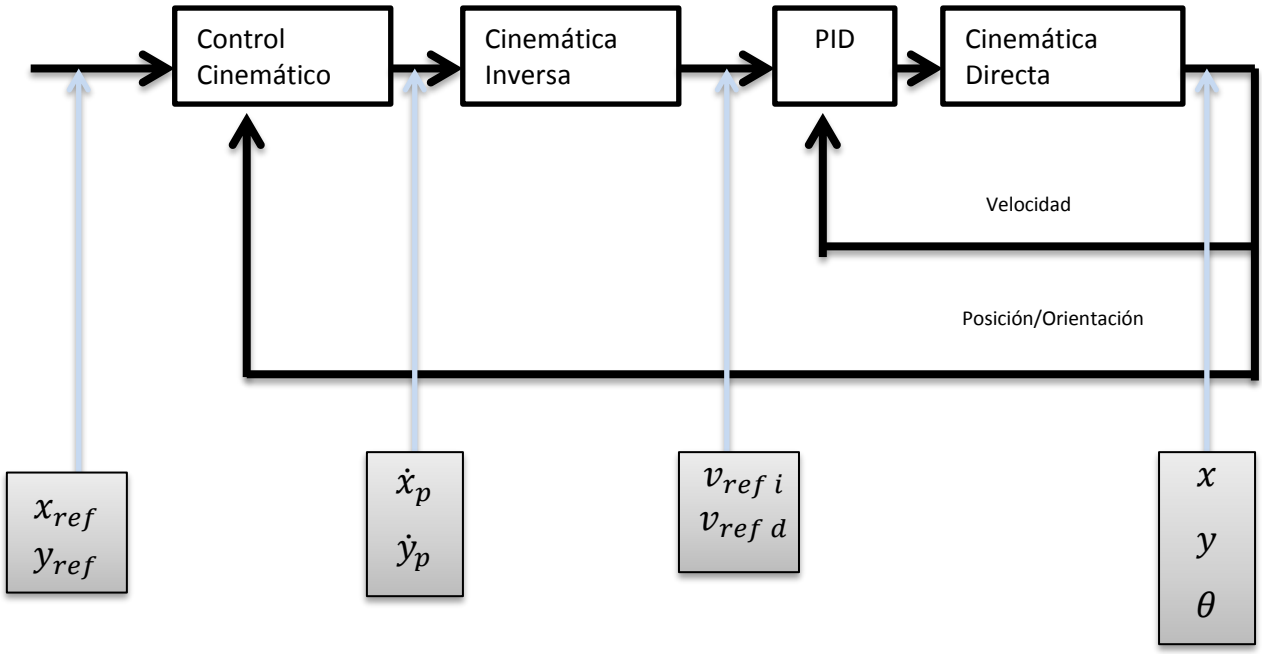

Fig. 2.12 Esquema control por punto descentralizado

Mediante la lectura de los encoders se obtiene las velocidades angulares de las dos ruedas que al dividirlas por el radio de la rueda se convierten en las velocidades lineales.

La velocidad lineal del robot se calcula como:

$$v = \frac{v_i + v_d}{2}$$

Donde v_i y v_d son las velocidades lineales de las ruedas izquierda y derecha, respectivamente. Esto quiere decir que la velocidad lineal del robot es la media de la velocidad de las ruedas.

En este tipo de tracción cada rueda gira a una velocidad diferente, el robot gira alrededor de un punto que está sobre la línea que une las dos ruedas. Este punto se denomina centro de curvatura instantáneo (ICC).

Las ruedas deben seguir una trayectoria que se mueva alrededor del ICC a la misma velocidad angular, por tanto:

$$\omega(R - b) = v_i \qquad \omega(R + b) = v_d$$

Donde ω es la velocidad angular del robot, b es la distancia de cada rueda al centro de curvatura y R es el radio de curvatura instantáneo de la trayectoria del robot. Así pues la velocidad angular del robot se puede calcular con la siguiente expresión:

$$\omega = \frac{v_i - v_d}{2b}$$

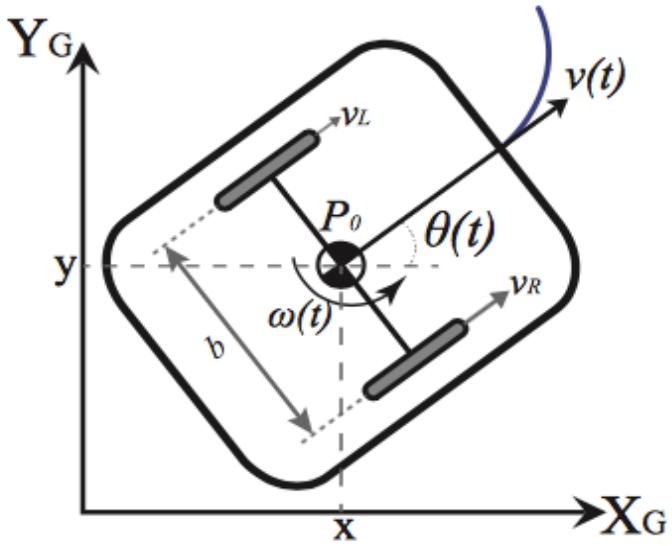

Fig. 2.13 Robot móvil con tracción diferencial [15]

2.3.1.1. CINEMÁTICA DIRECTA

La cinemática directa se utiliza para que a partir de la velocidad lineal y angular del robot, se determine la posición real de este. Las ecuaciones anteriores se pueden expresar de la siguiente forma:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ -1/2b & 1/2b \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix} \tag{2.1}$$

2

La integración de las siguientes ecuaciones da como resultado la posición y orientación del robot:

(2.2)

$$\dot{x} = v \cdot \cos\theta$$

$$\dot{y} = v \cdot \sin\theta$$

$$\dot{\theta} = \omega$$

Representando de una forma matricial las anteriores ecuaciones quedaría de la siguiente forma:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\cos\theta & 0\\ \sin\theta & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix} \tag{2.3}$$

Combinando las expresiones 2.1 y 2.3 la posición real del robot tiene la siguiente expresión:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \frac{1}{2b}\begin{bmatrix}\mathrm{b}\cdot\cos\theta & \mathrm{b}\cdot\cos\theta\\ \mathrm{b}\cdot\sin\theta & \mathrm{b}\cdot\sin\theta\\ -1 & 1\end{bmatrix}\begin{bmatrix}v_i\\ v_d\end{bmatrix} = \begin{bmatrix}\cos\theta\cdot\dfrac{v_i+v_d}{2}\\ \sin\theta\cdot\dfrac{v_i+v_d}{2}\\ \dfrac{v_i-v_d}{2b}\end{bmatrix} \tag{2.4}$$

Según se ha dicho anteriormente, para obtener la posición y la orientación se debe integrar los resultados obtenidos de la expresión anterior. Por tanto una aproximación integral de estas funciones se puede obtener utilizando la siguiente expresión:

$$\begin{bmatrix}x_{k+1}\\ y_{k+1}\\ \theta_{k+1}\end{bmatrix} = \begin{bmatrix}x_k + v_k T_S\cos\theta_K\\ y_k + v_k T_S\sin\theta_K\\ \theta_k + \omega_k T_s\end{bmatrix} \tag{2.5}$$

Donde T_S es el periodo de muestreo utilizado.

Después de hallar la posición y la orientación real del robot y teniendo las referencias de la trayectoria se aplica un control cinemático. Se trata de un control proporcional con pre alimentación de la velocidad y que tiene la siguiente expresión:

$$\begin{bmatrix}\dot{x}_p\\ \dot{y}_p\end{bmatrix} = \begin{bmatrix}k_{vx}\cdot\dot{x}_p\\ k_{vy}\cdot\dot{y}_p\end{bmatrix} + \begin{bmatrix}k_{px} & 0\\ 0 & k_{py}\end{bmatrix}\begin{bmatrix}x_{ref}-(x+e\cdot\cos(\theta))\\ y_{ref}-(y+e\cdot\sin(\theta))\end{bmatrix} \tag{2.6}$$

2.2.1.2 CINEMÁTICA INVERSA

El modelo cinemático inverso se utiliza para averiguar las velocidades de referencia de las ruedas del robot y de esta manera poder determinar qué acción de control aplicarle a cada rueda para que alcance la posición deseada.

Las coordenadas del punto descentralizado vienen definidas por la siguiente expresión:

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} x + e \cdot \cos(\theta) \\ y + e \cdot \sin(\theta) \end{bmatrix} \tag{2.7}$$

Si se calcula la derivada se obtiene:

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \begin{bmatrix} \dot{x} - e \cdot \cos(\theta) \cdot \dot{\theta} \\ \dot{y} + e \cdot \sin(\theta) \cdot \dot{\theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -e \cdot \cos(\theta) \\ 0 & 1 & e \cdot \sin(\theta) \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{2.8}$$

Para conocer la posición y orientación se parte de la ecuación 2.3 de la cinemática directa:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ -1/2b & 1/2b \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix} \tag{2.9}$$

Si se unen las expresiones 2.8 y 2.9 se obtiene que:

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \begin{bmatrix} 1 & 0 & -e \cdot \cos(\theta) \\ 0 & 1 & e \cdot \sin(\theta) \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ -1/2b & 1/2b \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix} \tag{2.10}$$

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \frac{1}{2b} \begin{bmatrix} \mathrm{b}\cos(\theta) + e\sin(\theta) & \mathrm{b}\cos(\theta) - e\sin(\theta) \\ \mathrm{b}\sin(\theta) - e\cos(\theta) & \mathrm{b}\sin(\theta) + e\cos(\theta) \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix}$$

Aplicando la inversa de esta matriz, se puede obtiene las velocidades de referencia que hay que aplicar a las ruedas:

$$\begin{bmatrix} v_{ref\ i} \\ v_{ref\ d} \end{bmatrix} = \frac{1}{e} \begin{bmatrix} \mathrm{e}\cos(\theta) + b\sin(\theta) & \mathrm{e}\sin(\theta) - b\cos(\theta) \\ \mathrm{e}\cos(\theta) - b\sin(\theta) & \mathrm{e}\sin(\theta) + b\cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} \tag{2.11}$$

Una vez obtenidas estas velocidades lineales de referencia de las ruedas, se calculan la velocidad angular de referencia y sabiendo las velocidades reales del robot se calcula el error, el cual se utiliza en el controlador calculado para obtener la acción de control que se necesita aplicar a los motores con el fin de seguir la trayectoria deseada.

2.4. LEGO MINDSTORMS

2.4.1. EVOLUCIÓN LEGO MINDSTORMS

En 1998 la empresa LEGO creó una plataforma de juguetes basada en la robótica educativa para niños y jóvenes. La empresa pone a disposición de los usuarios kits de robótica que incluyen diversas piezas con las que se puede montar una gran variedad de prototipos de robots parecidos a los que se utilizan en la vida real como por ejemplo: un coche, un ascensor, un robot limpiador. Esta empresa nos proporciona también un software que permite la programación, de los robots construidos, de una forma interactiva. En la actualidad se conocen 3 generaciones de Lego Mindstorms:

- **Ladrillo RCX (Robotics Command Explorer):** Apareció en 1998. Al principio se programaba con un lenguaje gráfico bastante intuitivo, posteriormente se añadieron muchos más lenguajes. Poseía 3 puertos para sensores y 3 puertos para actuadores. Al principio las comunicaciones se realizaban mediante puerto serie aunque posteriormente existía la posibilidad de hacerlo por infrarrojos con un periférico USB. En cuanto a las especificaciones del hardware, utilizaba un micro controlador Renesas de 8-bits H8/300, una ROM de 16Kb y una RAM de 32Kb.No presentaba WIFI y tampoco tenía Bluetooth.
- **Ladrillo NXT:** Lanzada en 2006 esta generación presenta dos versiones: la versión educativa y la 2.0. La comunicación con el ladrillo se hace mediante cable USB y Bluetooth que, también permite la comunicación con otros bloques y dispositivos móviles. El hardware del controlador principal tiene un microcontrolador ATMEL ARM7 de 32 bits a 48MHz, una memoria flash de 256KB y una RAM de 64KB. El controlador de los motores utiliza un micro controlador AVR de 8-bit, una memoria flash de 4Kb y una RAM de 512 bytes. Presenta tres puertos para actuadores y cuatro puertos para sensores.
- **Ladrillo EV3:** La novedad que presenta este ladrillo es la posibilidad de añadirle un módulo Wi-Fi y de esta manera poder conectar varios bloques a la misma red. Esto da la posibilidad de comunicar, el PC con el bloque, a una distancia superior a la que nos permite la comunicación vía Bluetooth. Además facilita la conexión de bloques y de esta manera se pueden crear aplicaciones donde se necesite la contribución y coordinación de más de un robot para realizar tareas que un solo robot no puede realizar. Este es el ladrillo utilizado en el proyecto, por tanto a continuación se explicará de una forma más detallada sus características.

Fig. 2.14 Lego RCX [12]

Fig. 2.15 Lego NXT [13]

Fig. 2.16 Lego EV3 [14]

2.4.2. CARACTERÍSTICAS FÍSICAS DEL EV3

- Cuatro puertos de entrada para conectar los sensores.
- Cuatro puertos de salida para conectar los motores.
- Un puerto USB para conectar el ladrillo a un ordenador.
- Un puerto USB para añadir un conector Wi-Fi.
- Un puerto para tarjeta Micro SD
- Un altavoz integrado.
- Receptor de señales infrarrojas.
- Receptor Bluetooth y Wi-Fi.
- El procesador principal es del tipo ARM de Atmel, modelo AR91SAM7S256. Es de 32 bits, con una memoria Flash de 256Kb, una memoria RAM de 64 Kb y una frecuencia de 48 MHz.
- Procesador secundario AVR de Atmel, modelo ATmega 48. Tiene una memoria Flash de 4Kb, una memoria RAM de 512Kb y una frecuencia de 8 MHz.
- Conexión con Bluetooth mediante un módulo CSR BlueCoreTM 4 v2.0 con sistema EDR. Tiene 47 Kb de memoria RAM interna, 8Mb de memoria flash externa y trabaja a 26 MHz.
- La comunicación con el PC se hace mediante USB 2.0, con una velocidad de 12 Mbits/s. Todos los puertos de salida soportan el protocolo I2C.

2.4.3. LENGUAJES DE PROGRAMACIÓN DEL EV3

A la hora de hablar de la programación la opción más fácil para programar el robot es utilizar el software EV3 que ha desarrollado la misma empresa que ha creado los robots aunque este software no permite realizar aplicaciones complicadas. Otros programas que pueden ser utilizados son los siguientes:

- **EV3Basic**: compilador de Microsoft Small Basic orientado al EV3.
- **Ev3_scratch**: se utiliza un lenguaje de programación visual. No se ejecuta en el ladrillo del robot sino que las órdenes son enviadas por un navegador vía Bluetooth.
- **LabVIEW:** software utilizado para diseñar sistemas con un lenguaje de programación visual gráfico. Abarca los componentes del hardware, así pues se puede utilizar todo el hardware con un único entorno de desarrollo. Este entorno utiliza el lenguaje G por eso permite desarrollar programas muy complejos con una interfaz de usuario completa.
- **RobotC:** lenguaje de programación basado en un entorno Windows. Utiliza un lenguaje en texto basado en el lenguaje C. Es el único lenguaje de programación que tiene un depurador en tiempo real.

2.4.4. PROGRAMACIÓN DEL SISTEMA OPERATIVO EV3DEV

El ev3dev ofrece la posibilidad de elegir entre diversos lenguajes de programación a la hora de programarlo. Las opciones ofrecidas son las siguientes:

- **Python:** se trata de un lenguaje de propósito general preparado para crear todo tipo de programas, por ejemplo, se puede realizar tanto aplicaciones Windows como páginas web. Al principio se desarrolló para Unix, aunque en la actualidad cualquier sistema puede ser compatible cuando existe un intérprete programado para él. Se trata de un lenguaje interpretado, esto quiere decir que no se debe compilar el código antes de realizar la ejecución. Es un lenguaje orientado a objetos. Entre sus principales ventajas está la rapidez y facilidad con el que se puede desarrollar el programa, mientras que el principal inconveniente es la velocidad baja
- **JavaScript:** Se utiliza básicamente para el desarrollo y diseño de páginas web. No necesita ser compilado ya que los navegadores son los que se encargan de interpretar ese código. Se trata de un lenguaje que puede ser ejecutado en cualquier plataforma existente. Entre sus principales ventajas se encuentra el hecho de ser un lenguaje sencillo además de tener gran cantidad de efectos visuales. Como principal desventaja se puede destacar el hecho de que los recursos no son tan extensos.
- **Java:** el propósito general de este lenguaje es la orientación de objetos. Es un lenguaje que se aprende muy rápido debido a su sencillez y dispone de muchos recursos y documentación para poder aprender. La desventaja que se puede destacar es el hecho de ser muy poco eficiente si se compara con otros lenguajes.
- **Java (LeJOS-Compatible):** se trata de la plataforma Java para LEGO Mindstorm.
- **Go:** se trata de un lenguaje de programación desarrollado por Google. Es simple confiable y eficiente pero no presenta algunas herramientas y características que los demás lenguajes presentan y que son muy útiles, como por ejemplo, la programación orientada a objetos.
- **C++:** es un lenguaje de programación orientada a objetos, se trata de la continuación del lenguaje en C. Es muy potente a la hora de la creación de sistemas complejos

además es un lenguaje muy robusto. Puede compilar y ejecutar código de C. Pero es mucho más difícil de usar que otros lenguajes.

- **C:** de propósito general asociado al sistema operativo UNIX. De nivel medio y utilizado para la programación de sistemas. Es altamente transportable y flexible (permite programar con múltiples estilos). El uso de punteros permite acceso a la memoria de bajo nivel. Es muy eficiente Pero también tiene ciertas desventajas: es bastante difícil de usar si no se domina el tema de la programación, además no es un lenguaje muy visual y esto dificulta aún más el uso a personas que no dominan mucho el lenguaje. Debido a la gran eficiencia que ofrece este lenguaje de programación, se ha decidido utilizarlo para implementar las aplicaciones de este proyecto.

El lenguaje de programación elegido en este caso es el C, y como consecuencia se necesita un editor y un compilador con el fin de poder hacer funcionar el código escrito en C.

Para ello el editor que se permite usar es el nano, un editor de texto para sistemas Unix basado en curses. Debido a la dificultad de usar este editor y a su poca austeridad se ha decidido editar el código en un PC y posteriormente transferirlo al ladrillo EV3.

2.4.5. HERRAMIENTAS PARA LA EDICIÓN REMOTA DEL CÓDIGO

2.4.5.1. CONEXIÓN SSH

Utilizando el ladrillo EV3 se puede arrancar un sistema operativo alternativo desde una tarjeta microSD, en este caso se ha utilizado el ev3dev, que es un sistema operativo basado en Linux Debian.

Con el fin de poder trabajar sobre este sistema operativo utilizando el ordenador se hace una conexión vía SSH y de esta manera se puede controlar el sistema operativo que tiene el ladrillo, empleando un PC.

SSH es un protocolo que permite conectarse a un equipo de forma remota mediante una red LAN/WAN. Esto hace que se pueda controlar completamente el ladrillo EV3, de forma que todo lo que ejecutemos afectará al ladrillo y se visualizará en el ordenador que tenemos conectado a este.

Este tipo de conexión utiliza técnicas criptográficas para garantizar que todas las comunicaciones hacia y desde el servidor remoto sucedan de forma encriptada. Estos son los pasos que se sigue para conectar un cliente de SSH:

1. El cliente abre una conexión TCP al puerto 22 del servidor. Lo hace al puerto 22 por defecto aunque posteriormente se puede modificar.
2. El cliente y el servidor acuerdan la versión del protocolo a utilizar.
3. El cliente envía al servidor la clave pública que va a utilizar.
4. El servidor compara la clave pública recibida con la que tiene almacenada

5. El servidor crea una cadena aleatoria y selecciona un algoritmo para encriptar todo ello usando la clave pública. Todo esto es enviado al cliente. El mensaje encriptado solo puede des encriptarse con la llave privada asociada
6. El cliente des encripta el contenido con su clave privada. Si todo esto es correcto se inicia la sesión.

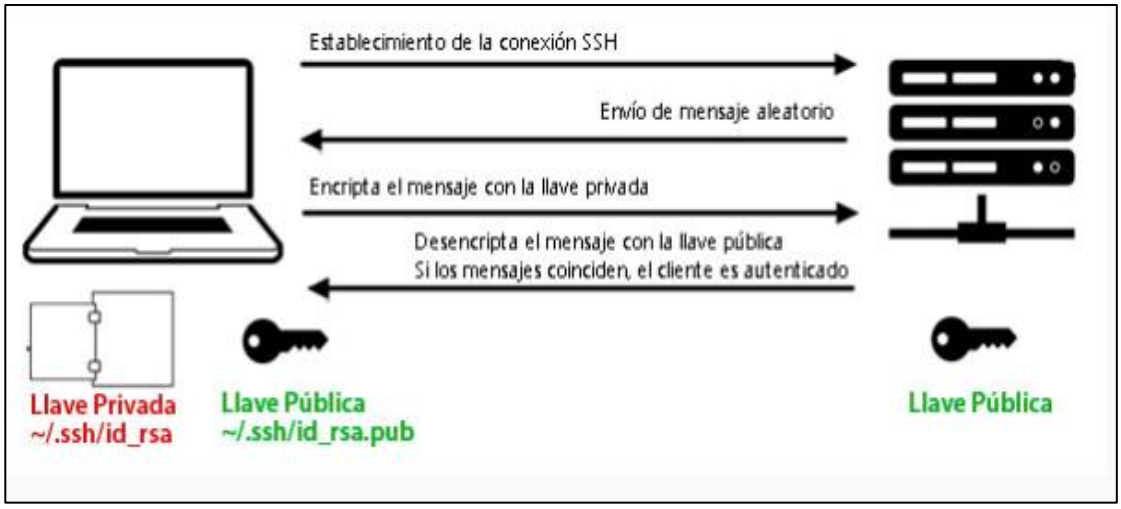

Fig. 2.17 Pasos para la conexión SSH [17]

2.4.5.2. PUTTY

Putty es un cliente SSH y Telnet que da la posibilidad de conectarse a servidores remotos iniciando una sesión en ellos y esto nos ofrece la posibilidad de ejecutar comandos. En nuestro caso se ha utilizado este programa para poder acceder al sistema operativo del ladrillo EV3 a través de un ordenador cuyo sistema operativo era el Windows.

2.4.5.3. WINSCP

Esta aplicación es un cliente SCFTP gráfico para Windows que emplea SSH. Se emplea para facilitar la transferencia segura de archivos entre dos sistemas informáticos, el local (que en nuestro caso es el ordenador) y uno remoto (el ladrillo EV3) que ofrezca servicios SSH. Para facilitar la transferencia de archivos del ordenador al ladrillo del Lego se ha utilizado esta aplicación.

2.4.6. HERRAMIENTAS DE COMPILACIÓN

Para la compilación del código existen varias opciones. La primera que se ha intentado ha sido utilizar el programa Docker. Se trata de un programa que permite realizar una compilación cruzada. Esto significa que a la hora de compilar el código, este programa se encarga de generar el código para el procesador del EV3, ya que es este el procesador que lo va a ejecutar. Este programa se ha probado en este proyecto pero no se ha conseguido hacer funcionar.

La otra opción, la que se ha decidido aplicar, es utilizar el compilador GCC. Se trata de un compilador integrado del proyecto GNU, que sirve también para el lenguaje en C.

Este compilador recibe el programa fuente en el lenguaje C (aunque también puede aceptar otros lenguajes) y crea un programa ejecutable cuyo lenguaje es el que la máquina necesita para funcionar.

Una vez instalado el sistema operativo, este compilador ya viene integrado. Por este motivo y por la sencillez de trabajar con él, se ha decidido que el compilador que se usará en el proyecto será el GCC.

2.5. SISTEMAS DE CONEXIONES INALÁMBRICAS

Actualmente se puede decir que se está viviendo una revolución tecnológica de las comunicaciones inalámbricas ya que, como se puede observar en la vida cotidiana, cada vez se utilizan menos los cables para comunicar dispositivos. Una de las principales ventajas de esta tecnología es la movilidad ya que no depende de un cable. Además es evidente que se necesita menos material y, tanto en la instalación como en el mantenimiento, este tipo de tecnología es más sencilla. Por este motivo en el actual proyecto se ha querido aprovechar las ventajas de esta tecnología para diseñar una aplicación en la que colaboren varios robots sin que estos estén unidos por un cable ya que de esta forma no habrá restricciones de movimiento debido a los cables.

Hoy en día existen bastantes tipos de tecnologías inalámbricas pero las más destacadas son las siguientes:

2.5.1. IDENTIFICACIÓN POR RADIO FRECUENCIA

Es un sistema que almacena y recupera datos remotos utilizando principalmente chips electrónicos denominados etiquetas. Estas se componen de un circuito integrado, un transductor radio y una antena que les permite recibir y responder a peticiones por radiofrecuencia. La información que envía es un identificador único para cada objeto. Existen tres tipos de etiquetas:

- Pasiva: No tienen fuente de alimentación y transmiten solo una pequeña información como el identificador del objeto. Se activan con el lector RFID.
- Activa: Tienen su propia batería y pueden transmitir información de manera continua
- Híbrida: Lleva batería, aunque solo transmite información en presencia de un lector RFID. La batería la tiene para transmitir información a distancias más largas que en las de tipo pasivo.

2.5.2. BLUETOOTH

La tecnología Bluetooth transmite inalámbricamente datos y voz a través de ondas de radio que operan en la banda ISM a 2.4 GHz. Para hacerlo usa las Redes Inalámbricas de Área Personal.

Funciona utilizando una técnica a la que se denomina salto de frecuencias. Va cambiando de frecuencia y se mantiene un cierto tiempo para después saltar a otra diferente. Se ha descubierto que el tiempo que transcurre entre los saltos de frecuencia es de unos 625 microsegundos lo que significa que en un minuto cambia 1600 veces de frecuencia.

Cuando en un mismo canal coinciden más de un dispositivo que tiene la tecnología Bluetooth se forman unas redes donde existe un maestro que gestiona la comunicación de la red y establece su reloj y unos esclavos que escuchan al maestro y sincronizan su reloj con el reloj establecido por el maestro. Estas redes son conocidas como piconets.

Como principal inconveniente de esta tecnología está el hecho de que tiene un radio de alcance bastante corto. De esta manera se pueden clasificar los dispositivos de la siguiente forma:

- Dispositivos de clase 1: La potencia máxima permitida es de 100mW y el alcance es de 100 metros.
- Dispositivos de clase 2: Son los más habituales y su radio de alcance está entre 5 y 10 metros mientras que su máxima potencia permitida es de 2.5mW.
- Dispositivos de clase 3: Tan solo son alcanzables a 1 metro y la potencia máxima es de 1mW.

2.5.3. ZIG BEE

Se trata de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal. Se suele utilizar en comunicaciones seguras con baja tasa de envío de datos. Tiene una gran similitud con la tecnología Bluetooth aunque también hay ciertas diferencias, como por ejemplo:

- Admite más dispositivos en una red.
- Tiene menor consumo, por tanto es más eficiente energéticamente.
- Mayor alcance, puede llegar hasta 300 m de alcance
- Menor velocidad que en Bluetooth, por este motivo actualmente solo se utiliza para la comunicación entre dispositivos en el ámbito de la domótica.

2.5.4. WI-FI

Se trata de una tecnología que permite la interconexión inalámbrica de dispositivos electrónicos. Permite conectarlos entre sí o a internet a través de un punto de acceso de red inalámbrica.

A grandes rasgos su funcionamiento se podría explicar de la siguiente manera: el adaptador inalámbrico de un ordenador traduce los datos en forma de señal de radio y los transmite utilizando una antena. El "router" inalámbrico recibe la señal y la decodifica, posteriormente envía la información a Internet utilizando una conexión física de Ethernet.

Existen varios tipos de dispositivos Wi-FI que se pueden clasificar en dos grupos:

Dispositivos de distribución o de red:

- Puntos de acceso: permite conectar dispositivos a una red existente.
- Repetidores: Se conectan a una red existente con señal débil y crean una señal más fuerte a la que se pueden conectar los equipos.
- Enrutadores: toman la conexión a internet y permite el acceso a todos los dispositivos que se conecten tanto si lo hacen por cable como si lo hacen en forma inalámbrica.

Dispositivos terminales:

- Tarjetas PCI: son las tarjetas agregadas a los ordenadores de sobremesa.
- Tarjetas PCMCIA: hoy en día están en desuso ya que no permiten alcanzar una velocidad de transmisión demasiado alta. Se utilizaron en los primeros ordenadores portátiles.
- Tarjetas USB: son las más utilizadas hoy en día y las más sencillas de conectar. Este tipo de tarjetas se han utilizado para la conexión de los robots entre ellos y con el ordenador.

Dada la importancia y la amplitud que tiene hoy en día este tipo de tecnología, sería conveniente comentar algunas ventajas:

- Permite el acceso a internet desde distintos puntos dentro de un espacio bastante amplio.
- Una vez ya configuradas las redes estas permiten el acceso de varios dispositivos sin necesidad de modificar alguna infraestructura o utilizar cables.
- Debido a la compatibilidad entre dispositivos con la marca Wi-Fi, se puede utilizar esta tecnología en cualquier parte del mundo.

Aunque las ventajas son muy importantes también existen ciertas desventajas que se deben tener en cuenta a la hora de utilizar esta tecnología:

- Presenta la velocidad menor si se compara con la conexión cableada ya que a veces existen interferencias y pérdidas de señal.
- El principal problema radica en la seguridad. Existen programas que pueden calcular la contraseña de la red y acceder a ella. Las claves de tipo WEP son las más fáciles de conseguir mientras que las más seguras son las del tipo WPA y WPA2.
- No es compatible con otro tipo de conexión sin cable.
- La potencia de la conexión es afectada por diversos agentes físicos que existen alrededor.

El robot Lego tiene la opción de utilizar la tecnología Bluetooth y la Wi-FI, para este trabajo se ha decido utilizar este último tipo de conexión ya que permite conectar los robots a mayor distancia entre ellos que en el caso del Bluetooth.

2.6. PROTOCOLOS DE COMUNICACIÓN TCP Y UDP

Con el fin de poder crear un canal de comunicación entre la máquina que hace de cliente y la máquina servidor es imprescindible hacer uso de algún protocolo de transporte. Los dos protocolos más importantes son los siguientes:

- **Protocolo UDP (User Datagram Protocol):** es un protocolo basado en el intercambio de datagramas. Permite el envío de estos datagramas a través de la red sin que anteriormente se haya establecido una conexión. Es considerado un protocolo no muy seguro ya que no tiene confirmación ni control de flujo y por este motivo no se ofrece ninguna garantía de que los paquetes lleguen en el mismo orden en el cual fueron enviados. Además ni siquiera garantiza la llegada de estos paquetes. Se utiliza en la transmisión de voz o vídeo ya que en este caso es más importante transmitir con velocidad y no es necesario garantizar la llegada de absolutamente todos lo bytes. Su principal ventaja es que provoca poca carga adicional en la red ya que además de ser simple también utiliza cabeceras muy simples.
- **Protocolo TCP (Transmission Control Protocol):** es fundamental en Internet. Se trata de un protocolo fiable ya que posee mecanismos que garantizan la integridad y la llegada en orden de los mensajes. Al tener estas garantías el protocolo utiliza comprobaciones adicionales y como consecuencia es menos eficiente que el UDP, aunque facilita la programación ya que no hay que tener en cuentas los problemas existentes en la red. Algunas de las características de este protocolo son:

    1. Orientado a la conexión: dos máquinas se conectan para intercambiar datos. Los sistemas se sincronizan para manejar el flujo de paquetes y adaptarse a la congestión de la red.

    2. Operación full-dúplex: la conexión TCP es un par de circuitos virtuales cada uno de los cuales están en una dirección. Únicamente los dos sistemas finales sincronizados pueden usar la conexión.

    3. Revisión de errores: existe una técnica denominada *checksum* que se usa para verificar que los paquetes no estén corruptos.

    4. Acuses de recibo: el receptor le transmite una señal al transmisor para indicarle que el paquete enviado ha sido recibido

    5. Control de flujo: cuando el transmisor transmite demasiado rápido y llega a desbordar un buffer el receptor descarta paquetes. De esta manera los acuses de que no ha llegado correctamente el paquete llegan al transmisor y este baja la velocidad de transferencia o deja de transmitir.

    6. Servicio de recuperación de paquetes: el receptor puede pedir que le reenvíen un paquete. De igual forma si el paquete no se notifica como recibido, el transmisor lo envía de nuevo.

En este proyecto se utiliza el protocolo TCP debido a la fiabilidad en la transmisión de datos ya que para crear las aplicaciones es importante que se transmitan todos los datos y en el orden en el cual fueron enviados.

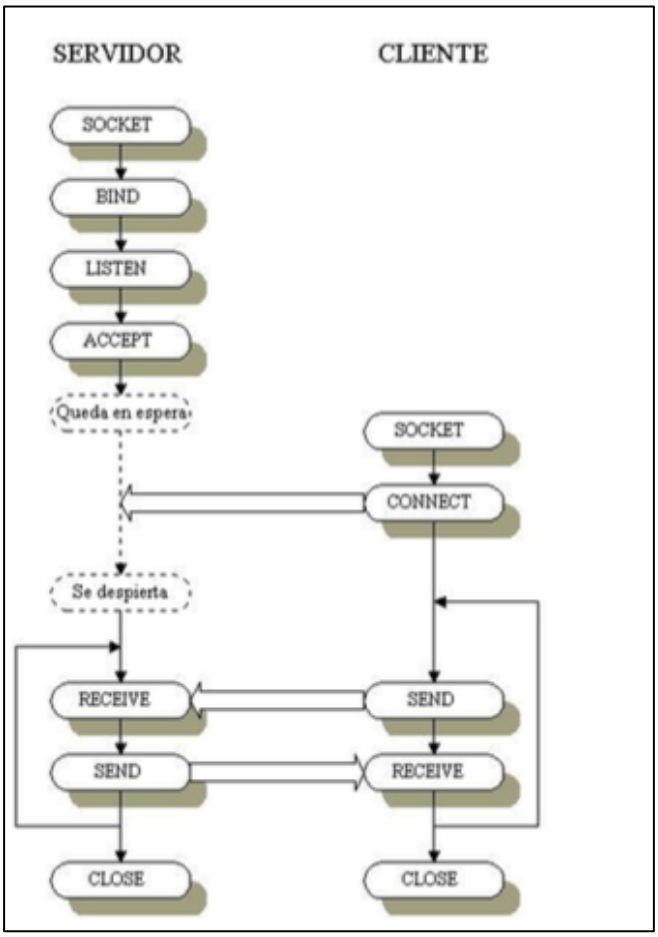

Fig. 2.18 Esquema cliente-servidor TCP simple [18]

2.6.1. SOCKETS

En el presente trabajo se realizarán varias aplicaciones distribuidas. La aplicación distribuida se define como aquella que tiene distintos componentes que se ejecutan en entornos separados, normalmente en diferentes plataformas conectadas a través de una red. Existen tres típicas aplicaciones distribuidas: de dos niveles (cliente-servidor), tres niveles (cliente-middleware-servidor) y multinivel. Para este trabajo se ha decidido utilizar el modelo cliente-servidor.

En este modelo de aplicación distribuida existen dos roles: cliente y servidor. El servidor dispone de recursos necesarios para ofrecer el servicio solicitado por el cliente. Así pues se puede generar una colaboración entre el cliente y el servidor ya que el cliente solicita al servidor un determinado servicio mientras que el servidor le concede los recursos para la

ejecución de dicho servicio. Este tipo de aplicación puede tener el cliente y el servidor en la misma máquina aunque normalmente esto no ocurre así.

En nuestro proyecto se tiene un robot Lego que hará de servidor mientras el otro será el cliente. En este caso es igual el robot que haga de servidor o de cliente ya que ambos tienen las mismas características y no influye en el desarrollo de la aplicación.

Para poder crear las aplicaciones se tendrán que intercambiar datos entre el servidor y el cliente y para ello necesitamos crear un canal de comunicación.

Cuando se trata de procesos que están en máquinas distintas la comunicación se lleva a cabo a través de redes de computadoras, que es una tarea muy compleja. Para resolver está complejidad se utiliza el diseño por capas. Cada capa utiliza funciones de la capa inferior y ofrece funciones a la capa superior. Es decir, cada capa recibe de la capa inferior un paquete que está compuesto por control y datos y la envía a la capa superior.

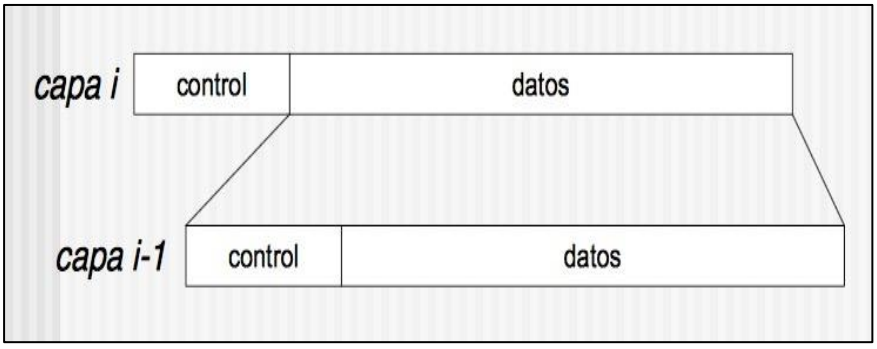

Fig. 2.19 Envío de datos de una capa a otra [19]

Uno de los modelos más importantes del diseño por capas es el modelo TCP/IP que está compuesto por cuatro capas diferentes:

- **Capa de aplicación:** Está compuesta por servicios y aplicaciones de comunicación estándar que puede utilizar todo el mundo. Asegura de que la información se transmita al receptor de un modo comprensible para el sistema, además administra las conexiones y terminaciones entre los sistemas que cooperan. Los protocolos más utilizados en esta capa son: DNS, SMTP, HTTP, FTP.

- **Capa de transporte:** Administra la transferencia de datos. Garantiza que los datos recibidos sean idénticos a los transmitidos. Los protocolos más utilizados son TCP y UDP.

- **Capa de Internet:** Se encarga de administrar las direcciones de datos y la transferencia entre redes. Los protocolos principales son IP e ICMP.
- **Capa de red:** Se encarga de la transferencia de datos en el medio de red y de definir las características del hardware de red.

Dentro de esta estructura, los sockets son una interfaz en la capa de transporte.

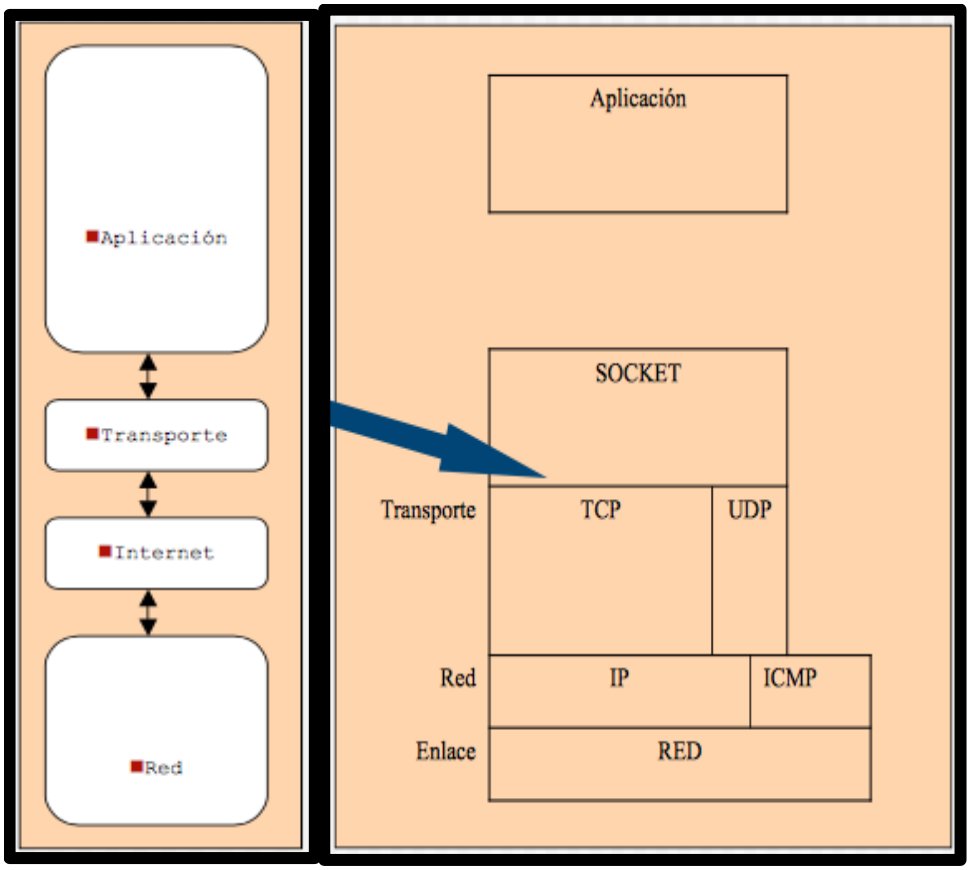

Fig. 2.20 Capas del modelo TCP/IP [19]

El socket es un mecanismo de comunicación que permite la comunicación bidireccional entre procesos que estén en la misma máquina o en procesos de máquinas diferentes. Dicho de otra forma es un punto de comunicación entre dos procesos por el cual se puede emitir o recibir información. La particularidad que presenta este mecanismo de comunicación frente a otros es que posibilitan la comunicación aun cuando ambas máquinas funcionen en distintos sistemas unidos mediante una red.

El mecanismo de comunicación vía sockets sigue los siguientes pasos:

- El servidor crea un socket con nombre y espera la conexión
- El cliente crea un socket sin nombre
- El cliente realiza una petición de conexión al servidor

- El cliente realiza la conexión a través de su socket mientras el servidor mantiene su socket original con nombre.

Todo socket viene definido por dos características fundamentales:

1. El tipo de socket: indica la naturaleza y el tipo de comunicación.
2. El dominio de socket: dice dónde se encuentran los procesos que se van a intercomunicar.

Existen diferentes tipos de sockets, los más destacables son los siguientes:

- SOCK_DGRAM: este tipo de socket no está orientado a conexión y sirve para datagramas. Además el envío de datagramas es de tamaño limitado, el mensaje se puede perder, duplicar o llegar fuera de secuencia. El protocolo que se utiliza en la capa transporte es el UDP.
- SOCK_STREAM: orientado a conexión, cuando se conecta se hace una búsqueda de un camino libre entre el servidor y el cliente. Este camino se mantiene en toda la conexión. Se utiliza para realizar comunicaciones fiables, de dos vías y con tamaño variable de los mensajes de datos. En la capa de transporte se utiliza el protocolo TCP.
- SOCK_RAW: permiten un acceso a más bajo nivel, pudiendo acceder directamente al protocolo IP del nivel de Red. Tiene un uso más limitado ya que está pensado para desarrollar nuevos protocolos de comunicación o para hacer uso de facilidades ocultas de un protocolo existente.
- SOCK_SEQPACKET: Tiene las características del SOCK_STREAM pero el tamaño de los mansajes es fijo.

Aunque existen diferentes dominios de comunicación los más empleados son:

- Dominio AF_UNIX: El cliente y el servidor están en la misma máquina. Cada socket tiene una dirección única y en este caso se utiliza como dirección el nombre de un archivo local.
- Dominio AF_INET: El servidor y el cliente pueden estar en cualquier máquina de la red Internet y la comunicación es basada en el conjunto de protocolos TCP/IP. El socket se identifica mediante el par: dirección IP de la máquina, número de puerto.

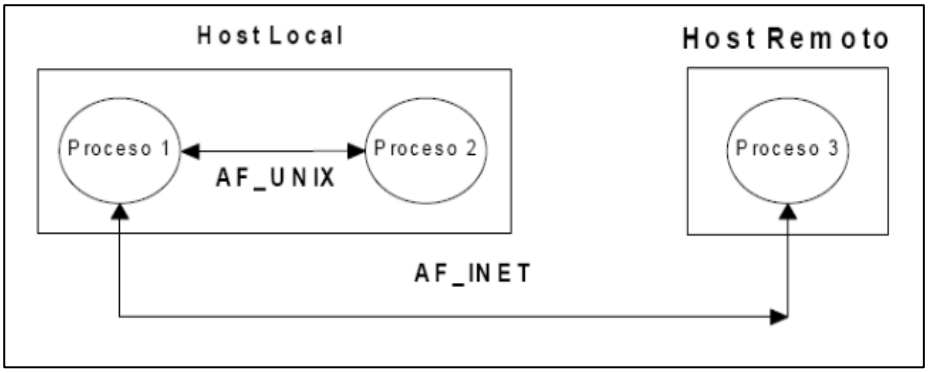

Fig. 2.22 Dominio AF_UNIX y AF_INET [20]

2.7. PROGRAMACIÓN MULTITAREA.HILOS POSIX

Cuando se habla de tarea en programación, se hace referencia al programa que está en ejecución. Aunque se puede tener una sola tarea, en la mayoría de aplicaciones se necesita más de una, en ese caso aparece el término multitarea. Se conoce como multitarea al tipo de procesamiento en el cual el sistema operativo se encarga de administrar distintas tareas en ejecución. Estas tareas compiten y comparten recursos del sistema.

La multitarea puede realizarse de dos formas diferentes:

Cooperativa: las tareas se ceden el control a otras de una forma voluntaria. En el caso de que una tarea exceda en tiempo debido a algún fallo, las demás tareas pueden quedarse sin tiempo para ejecutarse y eso puede provocar una paralización del sistema.

Preventiva: ceden al sistema operativo el reparto del tiempo de ejecución que de cada una de las tareas. De esta forma si una tarea se queda bloqueada, el sistema operativo le cede el tiempo de ejecución a la otra. Así se puede evitar el bloqueo del sistema

En este proyecto se ha necesitado hacer uso de la programación multitarea ya que es imprescindible la ejecución de varios procesos a la vez. Para la programación multitarea se han utilizado los threads (hilos).

Los hilos son un mecanismo por el cual un programa puede hacer más de una cosa al mismo tiempo, es decir, creando hilos se consigue realizar varias tareas simultáneamente. A diferencia de lo que se denominan procesos, no tienen su propio espacio en la memoria sino que tienen acceso todos al mismo espacio de memoria común es decir comparten recursos. Esto significa que cuando uno de los hilos modifica un dato en la memoria los otros hilos también sufrirán modificación en ese dato. Por este motivo es muy importante la correcta sincronización cuando varios hilos utilizan el mismo objeto.

Mientras alguno de los hilos de ejecución no termine el proceso sigue en ejecución. Cuando todos los hilos de ejecución finalizan, el proceso también lo hace y todos sus recursos son liberados.

A la hora de ejecutar un programa, la CPU utiliza el valor del contador de programa para determinar cuál instrucción debe ejecutar a continuación. El flujo de instrucciones resultante se denomina hilo de ejecución del programa. Cada hilo de ejecución del programa se asocia con un hilo, un tipo de datos abstracto que representa el flujo de control dentro de un proceso. Cada hilo tiene su propia pila de ejecución, valor de contador de programa, conjunto de registros y estado. Al crear muchos hilos dentro de un mismo proceso, se puede lograr paralelismo con un gasto extra bajo aunque presentan ciertas complicaciones en cuanto a la necesidad de sincronización.

Para poder sincronizarse correctamente los hilos tienen un estado de ejecución ya que de esta forma se puede conseguir evitar el problema creado por la compartición de recursos. Los principales estados son:

- **Creación**: Se crea un proceso y por tanto se crea un hilo para ese proceso. Una vez creado un hilo este puede crear otro dentro del mismo proceso.
- **Bloqueo**: Si un hilo necesita esperar que ocurra algo para poder seguir, este se bloquea. De esta forma otro hilo que esta "esperando" para ser ejecutado pasará a ejecutarse.
- **Desbloqueo:** Una vez terminado el proceso que bloqueó al hilo, el hilo desbloqueado pasa a la cola de los que están "esperando" para ejecutarse.
- **Terminación:** una vez finalizado el proceso del hilo, este libera su contexto y su pila.

Cuando un proceso realiza una operación de E/S bloqueadora, como la lectura, se bloquea hasta que la entrada está disponible. El hecho de que se bloquee puede traer problemas.

En el caso de este proyecto, se hace uso de la función que recibe los datos enviados por el otro robot y esta función es una operación bloqueante. Para solucionar este problema el sistema operativo permite crear varios hilos en un mismo proceso, modelo que se denomina multihilo. Los sistemas operativos pueden implementar hilos de dos formas:

- **Multihilo apropiativo:** permite al sistema operativo determinar cuándo debe haber un cambio de contexto.
- **Multihilo cooperativo:** depende del mismo hilo abandonar el control cuando llega a un punto de detención.

Los dos modelos tradicionales de control de hilos son el de hilos a nivel de usuario y el de hilos a nivel de núcleo (kernel).

- **Hilos a nivel de usuario:** se ejecutan sobre un sistema operativo existente. Los hilos dentro del proceso son invisibles para el núcleo Estos compiten entre sí por los recursos asignados a un proceso. Los programas que utilizan un paquete de hilos a nivel de usuario se ligan con una biblioteca especial en la que cada función de biblioteca y llamada del sistema está dentro de una envoltura. El código de envoltura llama al sistema de tiempo de ejecución para que se encargue de la gestión de los hilos. Los cambios de contexto entre hilos, son rápidos ya que no involucran llamadas al sistema. Cuando este tipos de hilos invocan llamadas al sistema bloqueantes, normalmente se bloquea todo el proceso. No puede aprovechar un sistema multiprocesador, ya que el sistema operativo ve un único hilo de ejecución.
- **Hilos a nivel de núcleo (kernel):** El núcleo ve a cada hilo como una entidad programable. Compiten por los recursos de procesador en todo el sistema. El bloqueo y la activación de hilos está a cargo del núcleo. Los cambios de contextos son más lentos si los comparamos con los hilos de usuario.

También existen lo que se denomina modelos de **hilos híbridos** . El usuario programa, en términos de hilos, a nivel de usuario y luego indica cuántas entidades programables por el núcleo están asociadas al proceso. En la ejecución se produce una correspondencia entre hilos a nivel de usuario y las entidades programables por el núcleo para poder conseguir el

paralelismo. Proporciona flexibilidad y gran rendimiento potencial al programador de la aplicación.

El modelo de programación utilizado en este proyecto (POSIX), es un modelo híbrido que presenta suficiente flexibilidad para apoyar hilos tanto a nivel de usuario como a nivel de núcleo en implementaciones específicas del estándar. Este modelo presenta dos niveles de programación: hilos y entidades de núcleo. Los hilos son análogos a aquellos a nivel de usuario mientras que las entidades del núcleo son programadas por el núcleo.

3. PARTE PRÁCTICA

3.1. OBTENCIÓN DE LA FUNCIÓN DE TRANSFERENCIA

El primer paso realizado ha sido obtener la función de transferencia de la velocidad del motor ya que es indispensable a la hora de diseñar el controlador PID.

Para ello se ha aplicado al motor una acción de control constante y se ha obtenido la gráfica que representa la velocidad en función del tiempo. Este proceso se ha realizado con varias acciones de control para poder sacar una media de los parámetros de la FdT y de esta manera asegurarse de que la función de transferencia obtenida se aproxima lo máximo a la real. (Sólo se representará una de las gráficas obtenidas ya que la obtención de los parámetros se ha realizado de igual forma en todos los casos).

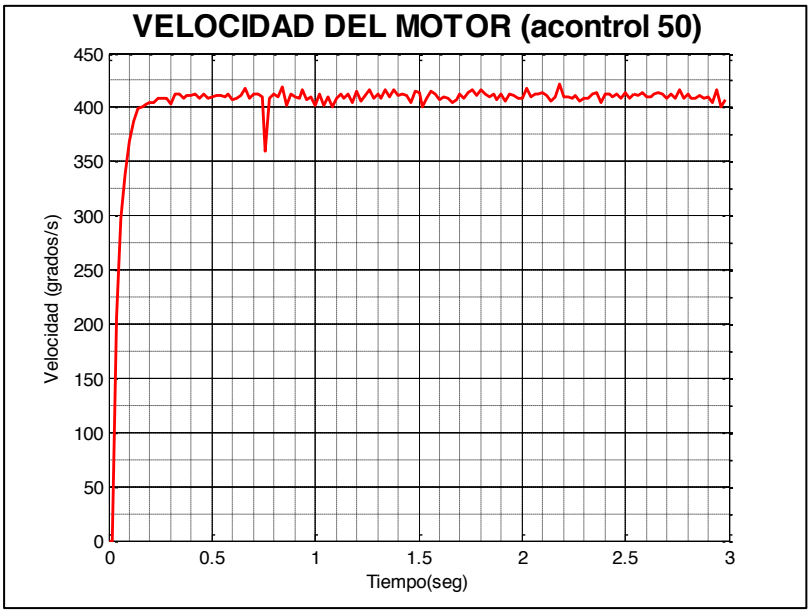

Fig. 3.1 Velocidad del motor con una acción de control de 50m

La FdT del motor es de primer orden por lo tanto tendrá la siguiente forma:

$$Gp_{vel}(s) = \frac{K}{(1 + \tau \cdot s)}$$

El cálculo de la ganancia (K), se obtiene al dividir el valor final de la salida entre el valor de la entrada (en este caso 50). El valor de la constante de tiempo será igual al tiempo que transcurre desde que se introduce la acción de control hasta que el sistema alcanza el 63.2% del valor final.

Así pues se puede observar que el valor final de la salida es de aproximadamente 580 (grados/s) que al dividirse entre 50 y al pasarse a radianes, resulta ser 0.1439 (rad/s).

Midiendo (de una forma aproximada) el tiempo que tarda el sistema en alcanzar el 63.2% del valor final, el resultado es de 0.05 segundos.

Siguiendo este procedimiento al aplicar varias acciones de control y obteniendo una media de los resultados, se ha llegado a la conclusión de que la función de transferencia de la velocidad tiene la siguiente expresión:

$$Gp_{vel}(s) = \frac{0.14}{(1 + 0.05 \cdot s)}$$

3.2. DISEÑO DEL REGULADOR PID

En este caso se pretende seguir la trayectoria indicada de una manera rápida, estable y sin error en régimen permanente. Por ello se ha decidido diseñar un regulador PID ya que la acción derivativa proporciona rapidez y estabilidad al sistema, mientras que la acción integral se encarga de eliminar el error en régimen permanente.

Aunque existen diferentes formas de diseñar un regulador, en este proyecto, se va a emplear el método empírico de Ziegler-Nichols aunque posteriormente se hace un ajuste manual con el fin de mejorar la respuesta.

Dentro de este método existen dos formas de diseñar los reguladores: mediante la respuesta a escalón del proceso en bucle abierto y mediante la respuesta oscilatoria mantenida en bucle cerrado. Se ha decidido utilizar la primera opción.

Para empezar se introduce un escalón a la entrada y se mide el tiempo en que la salida del sistema tarda en alcanzar 28.3% (t1) y el 63.2% (t2). Utilizando estos tiempos se pueden obtener T_A y T_B.

$T_B=1.5\ (t2-t1)$

$T_A=t2-T_B$

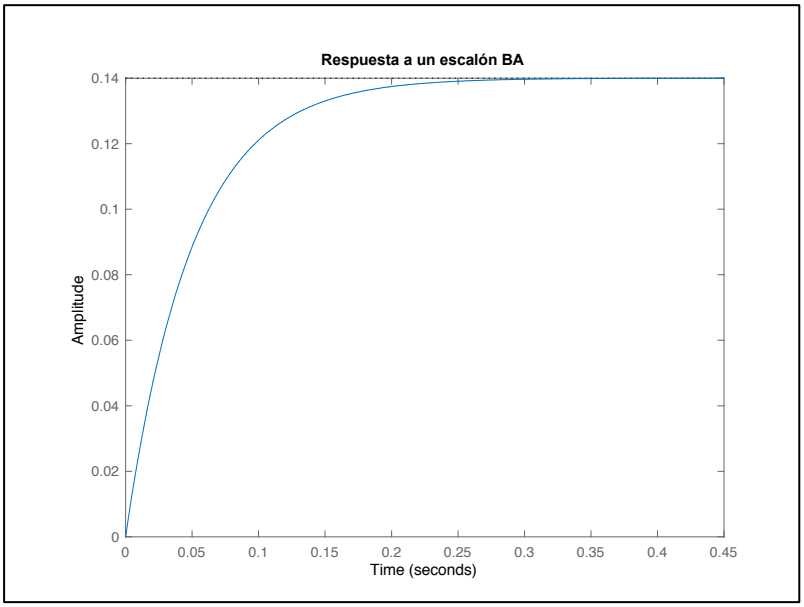

Fig. 3.2 Respuesta ante un escalón de entrada [28]

Así pues se puede observar en la gráfica que el valor del tiempo cuando la salida alcanza el 28.3% de su valor final, es 0.0161 segundos. El tiempo cuando se alcanza el 63.2% del valor final de la salida, es de 0.0507 segundos. Por tanto se tiene que T_B=0.0519 y T_A=0.0188.

Una vez obtenido T_B y T_A se utilizan las siguientes expresiones para sacar los parámetros de los reguladores continuos:

$$K_{PID} = 1.2 \cdot \frac{T_B}{T_A} = \mathbf{3.312}$$

$$T_I = 2 \cdot T_A = \mathbf{0.0376}$$

$$T_d = 0.5 \cdot T_A = \mathbf{0.0094}$$

Una vez obtenidos los parámetros de los reguladores continuos se calculan los parámetros de los reguladores discretos para posteriormente introducirlos en el esquema de Simulink y de esta forma observar los resultados.

$$q_0 = K_{PID} \cdot \frac{T_d + T}{T} = \mathbf{4.86}$$

$$q_1 = K_{PID} \cdot (-1 + \frac{T}{T_I} - \frac{2 \cdot T_d}{T}) = \mathbf{-4.66}$$

$$q_2 = K_{PID} \frac{T_d}{T} = \mathbf{1.556}$$

Introduciendo un escalón en la entrada del sistema controlado por el regulador diseñado se obtiene la siguiente gráfica:

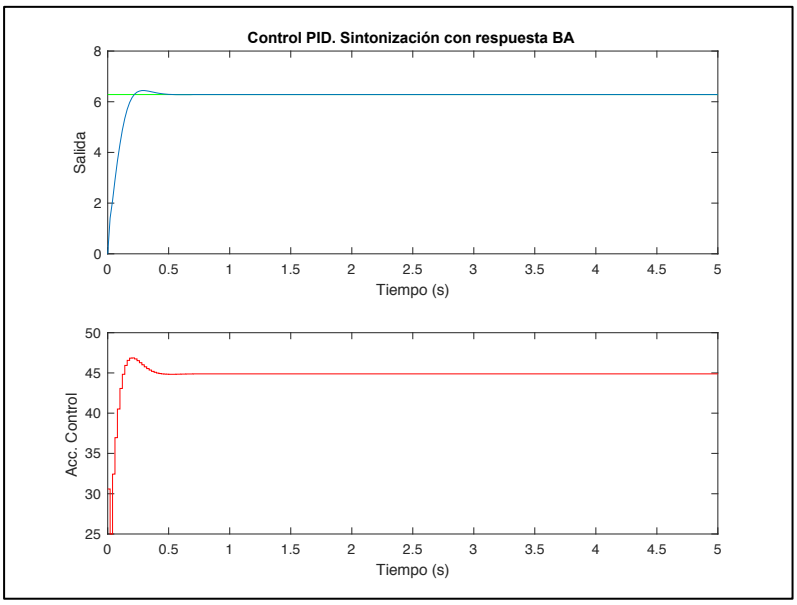

Fig. 3.3 Respuesta con el PID diseñado [28]

Como se puede apreciar, en los primero instantes, se produce una sobre oscilación que aunque sea bastante pequeña convendría eliminarla para asegurar así el buen funcionamiento del regulador. Por este motivo se ha hecho un ajuste manual en el que el regulador final es el siguiente:

$$\boldsymbol{u(k) = 4 \cdot e(k) - 2.8 \cdot e(k-1) - 0.9 \cdot e(k-2) + u(k-1)}$$

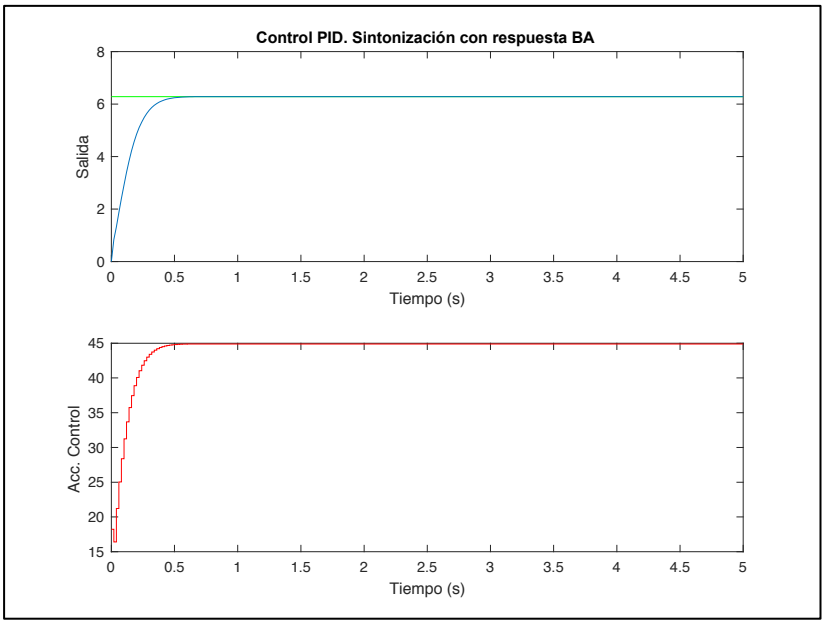

Fig. 3.4 Respuesta con el PID ajustado [28]

Una vez obtenido el regulador se pasa a implementar el código que contiene la parte del control por punto descentralizado.

3.3. CREACIÓN DE LAS REFERENCIAS

Para hacer el control de la posición lo primero que se debe crear son las referencias, es decir, los puntos que debe seguir el robot para que genere la trayectoria deseada.

En este proyecto se han empleado diversas trayectorias:

1. Una línea recta en el eje x, que empieza en la coordenada 0.5m y recorre un metro (termina en 1.5m) [28]

```
refx=(0.5+(1.500-0.500)*i/(num_itera));
refy=0;
```

2. Para comprobar el funcionamiento de los reguladores se ha generado una trayectoria circular de 0.5m de radios [28]

```
refx=0.5*cos(2*3.14159*0.04*Ts*i);
refy=0.5*sin(2*3.14159*0.04*Ts*i);
```

3. También para comprobar el diseño de los reguladores se ha probado una trayectoria cuadrada de 1m de lado. [28]

```
if (i < num_itera/4)
{
 refx=0.5+(1.500-0.500)*i/(num_itera/4);
 refy=0;

}
if (i < num_itera/2)
{
 refx=1.500;
 refy=(1.000-0)*(i-(num_itera/4))/(num_itera/4);
}
if (i < num_itera*3/4)
{
 refx=1.500-(1.500-0.500)*(i-2*(num_itera/4))/
(num_itera/4);
refy=1.000;
}
else
```

```
{
  refx=0.500;
  refy=1.000-(1.000-0)*(i-3*(num_itera/4))/
 (num_itera/4);
}
```

4. En las últimas dos aplicaciones se han generado curvas de Bézier entre distintos puntos, como por ejemplo:

```
P0x=0.8;
P1x=0.8;
P2x=1.6;
P0y=1.2;
P1y=0.4;
P2y=0.4;
parat=(i*0.02)/tfinal;
paratcua=parat*parat;
refx=((1.0-2.0*parat+paratcua)*P0x)+((2.0*parat-
2.0*paratcua)*P1x)+paratcua*P2x;
refy=((1.0-2.0*parat+paratcua)*P0y)+((2.0*parat-
2.0*paratcua)*P1y)+paratcua*P2y;
```

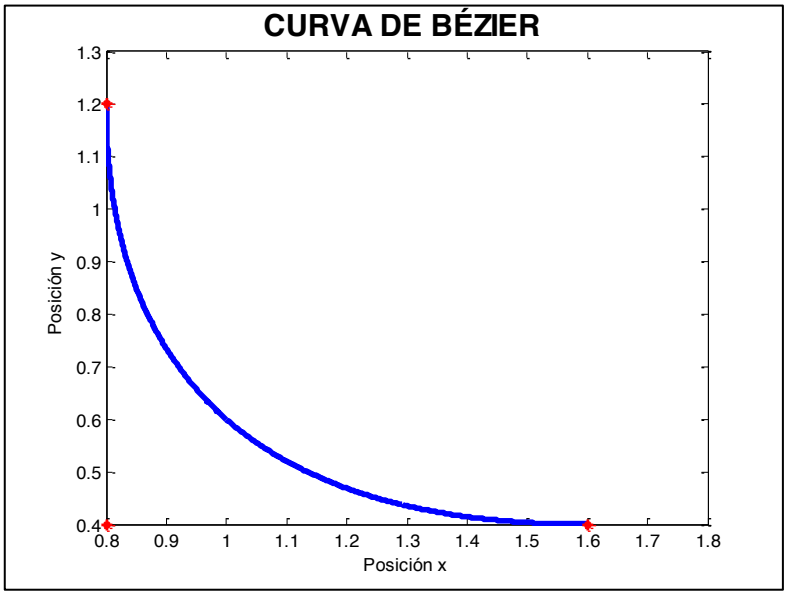

Fig. 3.5 Curva de Bézier obtenida

3.4. CÓDIGO CONTROL TRAYECTORIA

En primer lugar se crea la referencia de la trayectoria que se quiere seguir en función de i, que es un contador que se incrementa dentro del bucle hasta que su valor alcance un número determinado de iteraciones. En este caso el número del iteraciones es el tiempo que se quiere dar para que funcione la aplicación, entre el periodo de muestreo (0.02 segundos).

Un ejemplo de creación de las referencias de la trayectoria sería el siguiente (en este caso se crea una trayectoria recta de un metro de longitud (teniendo la x inicial en 0.5), aunque más adelante también se crearán otras trayectorias):

```
refx=(0.5+(1.500-0.500)*i/(num_itera));
refy=0;
velxref=(refx-refxante)/0.02;
velyref=0;
```

Posteriormente se lee el valor de los encoders de los motores utilizando la función **`tacho_get_position()`** que necesita como parámetros de entrada el puerto al que está conectado el motor y otro valor por defecto que en este caso se deja a 0, y devuelve el número de cuentas de encoder que es igual a los grados girados por el encoder.

A continuación se multiplica por una constante (en este caso se ha declarado como **`pul2rad`**) para convertir los grados en radianes.

```
pos_d_nueva=tacho_get_position(OUTD,0)*pul2rad;
pos_i_nueva=tacho_get_position(OUTB,0)*pul2rad;
```

Una vez hecha la lectura y la transformación a radianes se obtiene las velocidades angulares de las ruedas, restando la posición actual menos la posición del instante anterior y dividiendo entre el periodo de muestreo.

```
vel_ang_d=(pos_d_nueva-pos_d_anterior)/Ts;
vel_ang_i=(pos_i_nueva-pos_i_anterior)/Ts;
```

Para obtener las velocidades lineales se divide las velocidades angulares entre el radio de la rueda (en este caso 0.061 metros)

```
vel_lin_d=vel_ang_d*radio_rueda;
vel_lin_i=vel_ang_i*radio_rueda;
```

La velocidad lineal y angular del robot se obtiene aplicando las fórmulas de la matriz 2.1

```
Vel_Lin_robot=(vel_lin_d+ vel_lin_i)/2;
Vel_Ang_robot=(vel_lin_d- vel_lin_i)/(2*b);
```

Después de obtener la velocidad lineal y angular del robot se calcula la posición real del robot utilizando la expresión 2.5.

```
x=x+vel_Lin_robot*Ts*cos(ang);
y=y+vel_Lin_robot*Ts*sin(ang);
ang=ang+vel_ang_robot*Ts;
```

A continuación se aplica la expresión 2.6 que es el control cinemático considerando los siguientes valores: **kvx=kvy=4, y kp=kpy=0.5**

```
x_deriv=kvx*velxref+kpx*(refx-(x+(e*cos(ang))));
y_deriv=kvy*velyref+kpy*(refy-(y+(e*sin(ang))));
```

Se calcula las velocidades de referencia de las dos ruedas (expresión 2.11):

```
vref_i=(((e*cos(ang)+b*sin(ang))*x_deriv)+((e*sin(ang)-
b*cos(ang))*y_deriv))/e;
vref_d=(((e*cos(ang)-
b*sin(ang))*x_deriv)+((e*sin(ang)+b*cos(ang))*y_deriv))/e;
```

Seguidamente se calculan las velocidades angulares de referencia de las dos ruedas y el error de las velocidades angulares:

```
wref_d=vref_d/radio_rueda;
wref_i=vref_i/radio_rueda;

error_vel_d=wref_d-vel_ang_d;
error_vel_i=wref_i-vel_ang_i;
```

Utilizando el error de las velocidades angulares se calcula la acción de control necesaria, utilizando el regulador PID antes obtenido:

```
control_d=(4*error_vel_d)-(2.8*error_vel_d1)+(0.9*error_vel_d2)
+controld1;

control_i=(4*error_vel_i)-(2.8*error_vel_i1)+(0.9*error_vel_i2)
+controli1;
```

Como las acciones de control aplicadas a los motores solo pueden estar dentro del rango de [-100 100], se debe saturar la acción de control en caso de que esta no esté dentro del rango:

```
if (control_d>100)
     {
          control_d=100;
     }
if (control_d<-100)
     {
          control_d=-100;
     }
if (control_i>100)
```

```
        {
                control_i=100;
        }

if (control_i<-100)

        {
                control_i=-100;
        }
```

Una vez calculada la acción de control, para aplicar la al motor se utiliza la función **`tacho_set_duty_cycle_sp()`** que tiene como parámetros de entrada el puerto donde está conectado el motor y un valor por defecto que e este caso se deja a 0. Esta función debe ir acompañada de la función **`tacho_set_command()`** donde se le debe indicar el puerto al que está conectado el motor y como segundo parámetro debe ponerse la expresión **`TACHO_RUN_DIRECT.`**

```
tacho_set_duty_cycle_sp(OUTB,control_i);
tacho_set_duty_cycle_sp(OUTD,control_d);
tacho_set_command(OUTB, TACHO_RUN_DIRECT );
tacho_set_command(OUTD, TACHO_RUN_DIRECT);
```

Para comprobar el correcto funcionamiento del regulador PID y del control por punto descentralizado, se han programado los robots para seguir una trayectoria circular y una cuadrada:

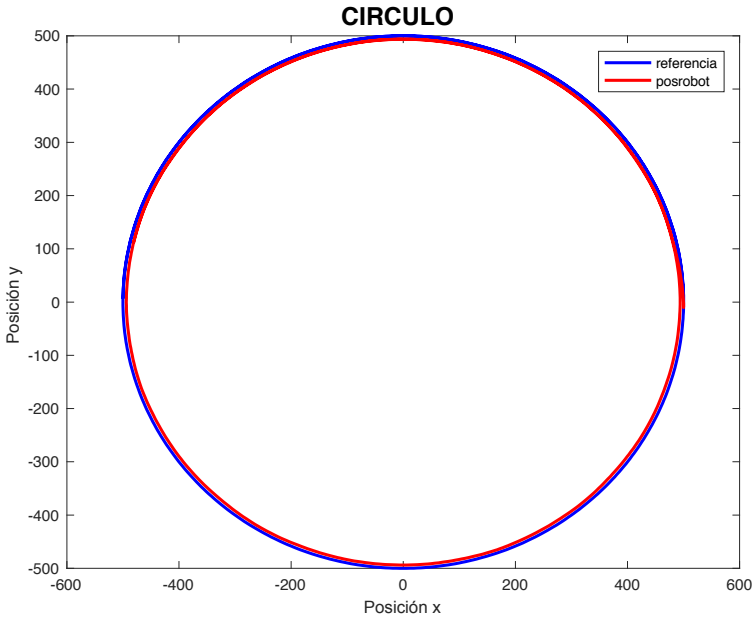

Fig. 3.6 Comprobación con la trayectoria circular, el diseño PID

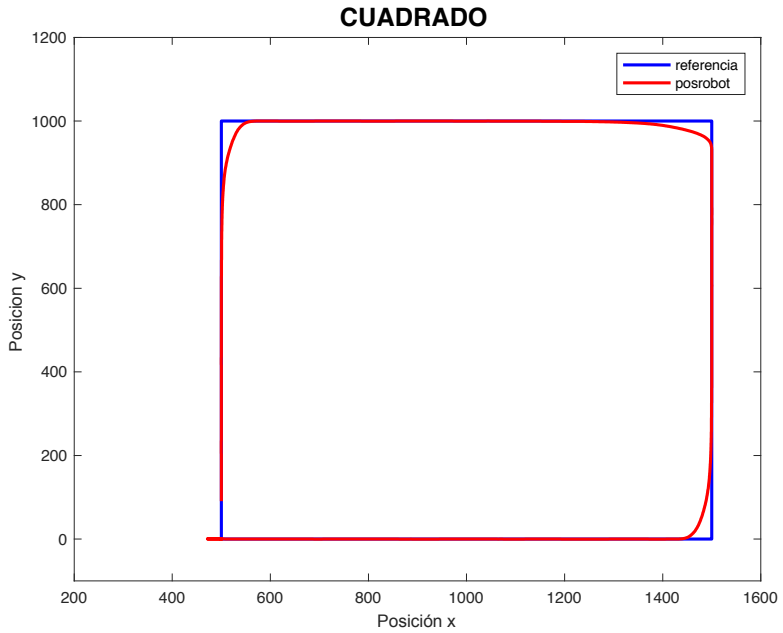

Fig. 3.7 Comprobación con la trayectoria cuadrada, el diseño PID

Como se puede observar en las gráficas las trayectorias se siguen de una forma correcta, sin error y de una forma estable. Se puede dar por válido la implementación del código del control de la trayectoria por punto descentralizado así como también el diseño del regulador PID.

3.5. CREACIÓN SOCKETS

Tal como se ha explicado en la parte teórica, para conectar los ladrillos de los Lego, se han creado sockets. En la primera parte se explicará la creación del socket en el servidor y posteriormente se detallará la programación del socket en el cliente.

3.5.1. SERVIDOR

Para empezar, los sockets se abren llamando a la función *socket(),* esta función retorna un descriptor de socket, que es de tipo *int.* Si existe algún error en la creación del socket, retorna -1 y la variable global *errno* se establece con un valor que indica el error que se produjo.

```
conexion = socket(int dominio, int tipo, int protocolo);
```

conexión: es el descriptor de socket que devuelve la función. Este valor se utilizará posteriormente para conectarse, recibir conexiones, enviar y recibir datos.

dominio: dominio en el que se realiza la conexión. En este caso será AF_INET ya que el servidor y el cliente se encuentran en máquinas diferentes.

tipo: puede ser SOCK_STREAM, SOCK_DGRAM SOCK_RAW o SOCK_SEQPACKET, dependiendo del tipo de socket que se quiera crear. En este caso es SOCK_STREAM porque nos proporciona una comunicación fiable.

protocolo: al poner esta variable a 0 el sistema selecciona automáticamente el protocolo más apropiado.

Así pues en este caso la función se ha declarado de la siguiente forma:

```
conexion = socket(AF_INET, SOCK_STREAM, 0);
```

Después de realizar la apertura del socket, se avisa al sistema operativo de ello y se utiliza la función *bind ( )* para darle un puerto al socket, es decir, una dirección IP y número de puerto del host local donde se producirá la escucha. Solo es necesario llamar esta función cuando se programa el servidor ya que cuando se programa el cliente, el kernel le asigna al socket la dirección IP y número de puerto al llamar ciertas funciones.

La función *bind ( )* presenta la siguiente estructura:

```
resp=bind(conexion,(struct sockaddr *)&my_addr, int addrlen);
```

resp: es el descriptor que devuelve la función.

conexión :es el descriptor que ha devuelto la función socket ( )

my_addr : es un puntero a una estructura *sockaddr* que contiene la IP del host local y el número del puerto que se va a asignar al socket. Esta estructura lleva varios campos, entre los que es obligatorio rellenar los siguientes:

> ***sin_family*** es el tipo de conexión, se pone mismo tipo que el indicado en el primer parámetro de *socket( ).*
>
> ***sin_port*** , número correspondiente al puerto al que se quiere conectar.
>
> ***sin_addr.s_addr***, se trata de la dirección del cliente
>
> Antes de utilizar la función *bind* se ha rellenado los campos obligatorios de la estructura *sockaddr* de la siguiente forma:
>
> ```
> server.sin_addr.s_addr=inet_addr("192.168.1.129");
> server.sin_family = AF_INET;
> server.sin_port = htons(6000);
> ```
>
> (htons () se encarga de convertir un short int de host byte order a network byte order)

addrlen : es establecido al tamaño de la estructura *sockaddr,* normalmente se utiliza *sizeof(struct sockaddr).*

La declaración de la función *bind ( )* se ha hecho de la siguiente forma:

```
resp=bind(conexion,(struct sockaddr *)&server, sizeof(server));
```

Esta función devuelve un 0, si se produjo un error devuelve un -1.

Después de realizar todo este proceso se debe avisar al sistema de que comience a atender a dicha conexión de red. Para ello se utiliza la función *listen ( ).* A partir de este momento el sistema registra la conexión de cualquier cliente para transferirla a la aplicación cunado lo

solicite. Si llegan conexiones de clientes más rápido de lo que la aplicación es capaz de atender, el sistema almacena en una cola las mismas y se podrán obtener luego según se vaya pidiendo.

```
int listen(int conexion,int backlog);
```

conexión : es el descriptor devuelto por la función socket () que será utilizado para recibir conexiones

backlog : es el número máximo de conexiones en la cola de entrada de conexiones. Las conexiones entrantes quedan en espera en esta cola hasta que son aceptadas. En este caso solo se aceptará una única conexión

Así pues la función *listen ( )* se ha programado de la siguiente forma:

```
listen(conexion,1);
```

El siguiente paso a realizar es la aceptación de las conexiones de clientes. Para ello se utiliza la función *accept ()*.Esta función le indica al sistema operativo que le dé al siguiente cliente se la cola. Si no existen clientes disponibles, se queda bloqueada hasta que algún cliente se conecte. Devuelve un nuevo descriptor de socket al recibir la conexión del cliente en el puerto configurado.

```
int accept( int conexion,(struct void *addr, int *addrlen);
```

conexión :descriptor devuelto por la función socket ()

addr : puntero a una estructura de tipo *sockaddr*. Aquí se almacena información de la conexión entrante. Se utiliza para determinar qué host está llamando y desde qué número de puerto

addrlen, tamaño de la estructura *sockaddr*. Normalmente se utiliza *sizeof(struct sockaddr)*

De esta forma en el código del proyecto la función accept se llama de la siguiente forma:

```
conexion_cliente=accept(conexion,(struct sockaddr *)&cliente,
&longc);
```

La comunicación se cierra utilizando la función *close ()*.

3.5.2. CLIENTE

El primer paso, en el caso del cliente, es igual al servidor. Se abre un socket utilizando la función *socket ()* (está explicada en el apartado de la conexión del servidor).

En el programa del cliente se declara de la siguiente forma:

```
conexion = socket(AF_INET, SOCK_STREAM, 0);
```

Después de abrir el socket se declaran dos estructuras tipo *sockaddr* que contienen la información sobre la IP y el puerto del servidor y del cliente también

```
server.sin_addr.s_addr=inet_addr("192.168.1.129");
server.sin_family = AF_INET;
```

```
server.sin_port = htons(6000);
cliente.sin_addr.s_addr=inet_addr("192.168.1.130");
cliente.sin_family = AF_INET;
cliente.sin_port = htons(6000);
```

Una vez abierto el socket, se solicita la conexión con el servidor utilizando la función *connect*(). Dicha función queda bloqueada hasta que el servidor acepte la conexión. En el caso de que no haya servidor, se sale dando un error. Tiene la siguiente estructura:

```
connect(int conexion,struct sockaddr *serv_addr,int addrlen)
```

conexión: es el descriptor del socket devuelto por la función *socket ()*

serv_addr :es una estructura sockaddr que contiene la dirección IP y número de puerto del servidor.

addrlen :tamaño de la estructura sockaddr. Se utiliza *sizeof(struct sockaddr)*

La llamada de esta función, en el proyecto, se realiza de la siguiente forma:

```
connect(conexion,(struct sockaddr *)&server, sizeof(server))
```

3.5.3. FUNCIONES PARA ENVIAR Y RECIBIR

Una vez establecida la conexión, se puede empezar la transferencia de datos. *Send( )* y *recv ( )* son las dos funciones que sirven para la transferencia de datos en los sockets STREAM. Tienen la siguiente estructura:

Para enviar datos se utiliza:

```
send(int conexion, const void*msg,int len, unsigned int flags)
```

conexión : descriptor del socket por el cual se enviarán los datos.

msg : puntero a los datos que se quieren enviar

len : longitud de los datos en bytes

flags : se pone por defecto el valor de 0.

La función retorna la cantidad de datos que se han enviado, que puede ser menor que la cantidad de datos que se escribieron en el buffer para enviar. Enviará la cantidad máxima de datos que pueda manejar. El programador tiene que asegurarse del correcto envío de datos, comparando la cantidad de datos enviado con *len*. Si no se han enviado todos los datos, se pueden enviar en la próxima llamada a esta función

Para recibir datos se utiliza:

```
send(int conexion, void *buf,int len, unsigned int flags)
```

conexión : descriptor del socket por el cual se recibirán los datos.

buf : puntero a un buffer donde se almacenarán los datos recibidos

len : longitud del buffer *buf*

flags :se pone por defecto el valor 0

Si no existen datos para recibir en ese socket, la llamada de la función *recv ( )* se bloquea (no retorna nada) hasta que llegan los datos. Existe la opción de hacer que esta función no sea bloqueante de forma que cuando no hay datos para recibir, la función retorna un -1 y establece la variable *errno= EWOULDBLOBK.*

Retorna el número de bytes recibidos.

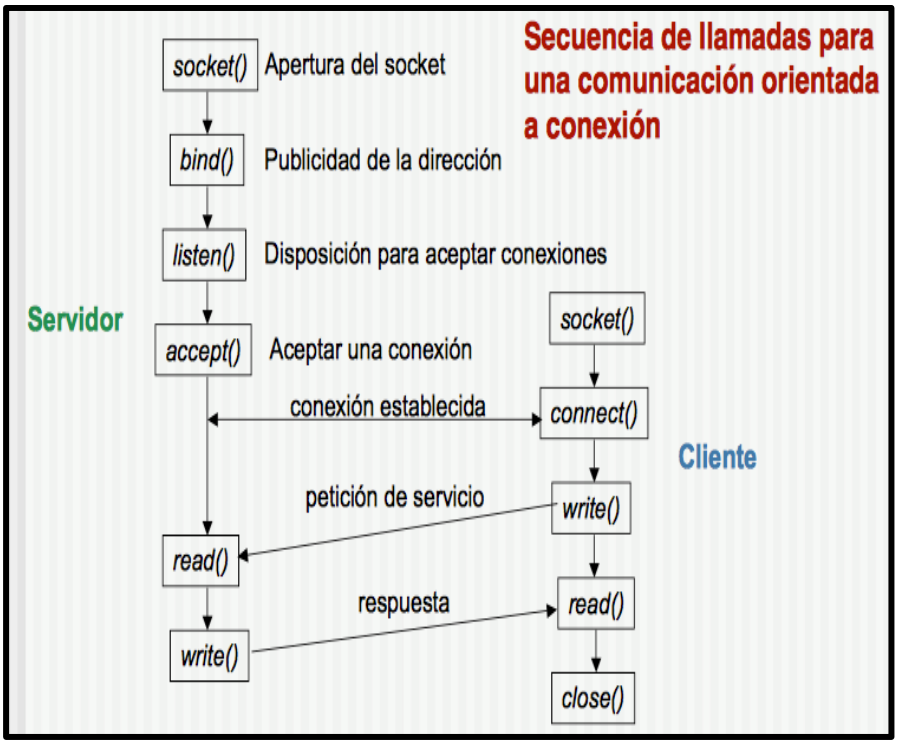

Fig. 3.8 Pasos para la conexión utilizando sockets [19]

3.6. CREACIÓN HILOS POSIX

En este proyecto se han creado hilos con el fin de evitar que la función *recv ( )*, la cual se encarga de recibir los datos enviados por el socket, produzca un paro en la parte de control del robot. Este paro se podría producir en el caso de que cuando se llame a esta función, no haya nada que recibir, de este modo al ser una función bloqueante el control se bloquearía hasta que se reciba algo.

Para empezar se tiene que tener en cuenta que se debe añadir una nueva librería, *pthread.h*

En primer lugar se declaran las funciones que se van a utilizar en el programa. En este proyecto, como ya hemos dicho, se utilizarán dos. Estas funciones devuelven un parámetro de tipo *void y tienen como argumento un parámetro de tipo void*:

```
void *funcionThread1 (void *parametro);
void *funcionThread2 (void *parametro);
```

A continuación se crean los hilos cuyas funciones de han declarado anteriormente. Para ello se utiliza la función *pthread_create ( ).* Esta función devuelve 0, en el caso de existir un error en la creación devuelve un -1. Admite cuatro parámetros:

- ***pthread_t*:*** se trata de un puntero a un identificador de thread. La función rellena este parámetro para que luego se pueda utilizar esa referencia a la hora de trabajar sobre el hilo.
- ***Pthread_attr_t*:*** son los atributos de creación del hilo. Existen varios atributos, uno de los mejores ejemplos es el de la prioridad. Normalmente se utiliza NULL, para que el hilo se cree con sus atributos por defecto.
- ***Void*(*)(void*):*** se trata de la función declarada anteriormente. Esta función se ejecutará como un hilo aparte. El hilo termina cuando la función también termina.
- ***Void*:*** parámetro que se le pasa a la función anterior cuando se ejecuta en el hilo aparte. Normalmente se utiliza NULL.

```
error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);
```

Para hacer las aplicaciones se han creado dos hilos y para que estos funcionen se necesita que se ejecute continuamente el hilo padres. Para evitar poner un *while (1),* se ha utilizado la función *pthread_join ()* que se encarga de detener el hilo que la invoca hasta que un determinado hilo termina. Esta función necesita dos parámetros: el primer argumento es el identificador del hilo que se quiere esperar y el segundo sirve para recibir parámetros del thread citado, es de tipo puntero a void*. En el caso de este proyecto, como segundo parámetros de utiliza el valor NULL ya que no se quiere recibir nada de los threads creados.

```
pthread_join(idHilo1,NULL);
pthread_join(idHilo2,NULL);
```

Para desarrollar las aplicaciones se ha utilizado el kit del LEGO EV3 y se ha montado el siguiente robot (figura 3.9) . Se ha tenido en cuenta que la parte delantera tenga un diseño que nos dé la posibilidad de empujar objetos cuando el robot se desplaza.

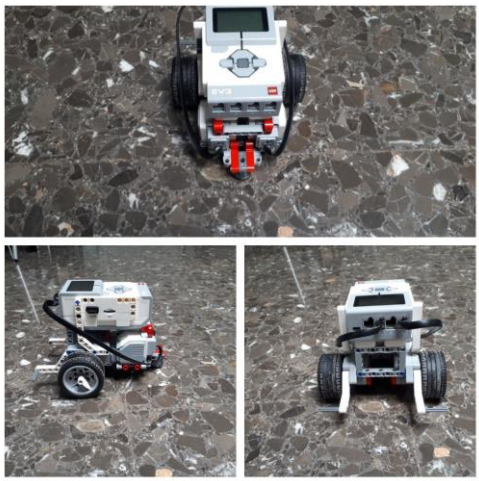

Fig. 3.9 Montaje de uno de los robots utilizados

3.7. EXPERIENCIA 1: EMPUJADOR DE OBJETOS

En algunas ocasiones se utilizan robots móviles para empujar objetos y de esta manera ahorrar tiempo y esfuerzo al ser humano. Pero puede haber ocasiones cuando los objetos que se deben empujar son tan pesados que necesitan la fuerza de más de un robot para hacerlo. Para que este empuje se haga de una manera correcta, los robots que contribuyen, deben sincronizarse entre ellos. Necesitan intercambiarse datos sobre su posición y orientación y de esta manera lograr que los dos sigan la misma trayectoria a la misma velocidad. En esta primera aplicación se ha intentado dar una solución a este problema y se han programado dos robots para que empujen, conjuntamente, un mismo objeto, en este caso, una caja.

Los dos robots parten de la misma posición en x pero separados una cierta distancia en el eje y (ver figura 3.10), es decir, tienen una posición paralela. Están separados unos 10 cm de la caja que deben empujar. Los dos robots deben empezar a recorrer una trayectoria en línea recta al mismo tiempo y a la misma velocidad con el fin llegar a la caja y empujarla los dos al mismo tiempo. La caja debe estar recta cuando se haga el empuje para así comprobar que los dos robots funcionan a la vez.

Fig. 3.10 Posición inicial

Para transmitir los datos, sobre la posición, de un robot a otro se necesitan crear un canal de comunicación, en este caso se han utilizado sockets.

A continuación se explica parte del código creado para esta aplicación (el código entero está escrito en el anexo)

Para empezar se han incluido las librerías necesarias para el proyecto, hay que tener en cuenta que para crear los sockets se necesita la librería *sys/socket.h*, para los hilos la librería *pthread.h* y para los cálculos de trigonometría *math.h.* El resto son librerías comunes que se incluyen en la mayoría de códigos de porgramación en C.

```
#include <pthread.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<math.h>
#include<math.h>
#include<time.h>
```

Tambíen se han tenido que añadir las librerías del que tiene el EV3 para poder así utilizar los motores y hacer las conexiones necesarias.

```
#include"ev3_light.h"
#include "ev3.h"
#include "ev3_port.h"
#include "ev3_tacho.h"
#include "brick.h"
```

Posteriormente se han reseteado los sockets para que cualquier valor que haya sido guardado, se borre, y de esta forma evitar que influya en el cálculo de las posiciones y en el control de los robots.

```
tacho_reset(OUTB);
tacho_reset(OUTD);
```

A continuación se crean los sockets utilizando las direcciones IP que aparecen en la parte superior izquierda de la pantalla del ladrillo del LEGO (figura 3.11). Estas mismas direcciones se utilizan para conectar vía SSH el Lego con el ordenador. Se elige como puerto de conexión el 6000.

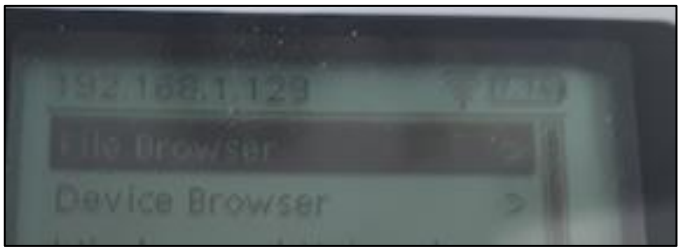

Fig. 3.11 Dirección IP

Conexión del servidor:

```
conexion = socket(AF_INET, SOCK_STREAM, 0);
if(conexion == -1){
  printf("Error al crear el socket\n");
}
puerto=6000;
bzero((char *)&server, sizeof((char *)&server));
  server.sin_addr.s_addr=inet_addr("192.168.1.129");
  server.sin_family = AF_INET;
  server.sin_port = htons(6000);
resp=bind(conexion,(struct sockaddr *)&server, sizeof(server));
listen(conexion,1);
printf("Esperando conexiones entrantes... \n");
longc = sizeof(cliente);
conexion_cliente=accept(conexion,(struct sockaddr *)&cliente,
&longc);
 if(conexion_cliente<0) {
    printf("Error al aceptar trafico\n");
    close(conexion);
    return 1;
}
```

Conexión del cliente:

```
conexion = socket(AF_INET, SOCK_STREAM, 0);
if(conexion == -1){
   printf("Error al crear el socket\n");
}
```

```
puerto=6000;
bzero((char *)&server, sizeof((char *)&server));
  server.sin_addr.s_addr=inet_addr("192.168.1.129");
  server.sin_family = AF_INET;
  server.sin_port = htons(6000);
  cliente.sin_addr.s_addr=inet_addr("192.168.1.130");
  cliente.sin_family = AF_INET;
  cliente.sin_port = htons(6000);
resp=bind(conexion,(struct sockaddr *)&cliente, sizeof(cliente));
 if(connect(conexion,(struct sockaddr *)&server, sizeof(server)) <
0){
    printf("Error conectando con el host\n");
    close(conexion);
    return 1;  }
```

Una vez programada la conexión de los sockets, dentro del programa principal *(main)* se crean los dos hilos que se utilizarán y se pone a la espera al padre hasta que los dos hilos terminen de ejecutarse para que de esta forma el hilo padre no se cierre hasta que los dos hilos no se hayan ejecutado completamente. Para la espera se utiliza la función *pthread_join ( ).*

```
pthread_t idHilo1;
pthread_t idHilo2;
int error1;
int error2;
error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);
if( error2!=0 || error1!=0 ){
            perror("No puedo crear thread");
            exit(-1);
      }
      pthread_join(idHilo1,NULL);
      pthread_join(idHilo2,NULL);
```

Toda la programación que se ha descrito hasta aquí, se utiliza y se hace de una forma idéntica en todas las aplicaciones de este proyecto, por tanto no se explicará en los apartados de las siguientes aplicaciones. Las funciones que contienen los hilos son las que varían en los distintos experimentos.

La función del primer hilo tendrá la siguiente estructura:

Se crea un bucle *while ( )* que se ejecutará un número de iteraciones incrementando una unidad el valor de i. El número de iteraciones es igual al tiempo que se quiere que dure la aplicación, entre el periodo de muestreo, que en este caso es 0.02 segundos.

Una vez dentro del bucle, se comprueba que la diferencia de posiciones entre los robots no sean superiores a 0.1 metros. En el caso de que esta diferencia sea superior, se paran los motores y de esta manera se para el robot hasta que la diferencia de posiciones sea menor que 0.1. Así se asegura el hecho de que los robots tengan posiciones aproximadamente iguales y de esta manera puedan transportar recta la caja. En el caso de que la diferencia de posición no sea mayor que 0.1 se procederá a realizar el control de la posición por punto descentralizado (explicado en el apartado 3.3), asegurando así que se sigue la trayectoria recta.

En cada iteración se envía al otro robot, la posición en el que se está y se duerme esta función hasta que se cumplan los 20 milisegundos, es decir, se mide el tiempo que se tarda en hacer el control y se hace un *sleep()* de 20 milisegundos menos ese tiempo transcurrido. De esta forma se asegura el cumplimiento del periodo de muestreo

Para enviar la posición al otro robot se escriben las siguientes funciones:

```
Coordmensajee.posxe=x;
Coordmensajee.posye=y;


send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
      posxe[i]=Coordmensajee.posxe;
      posye[i]=Coordmensajee.posye;
```

La segunda función del segundo hilo recibe los parámetros enviados por el otro robot, y se guardan en una variable para que después se pueda usar para calcular la diferencia de posiciones. Se ha creado esta función solo para las recepciones ya que, como ya se ha comentado anteriormente, la función que recibe es bloqueante .Si se utiliza en el hilo del control de la posición, la función *recv ( )* esperará a que le lleguen datos bloqueando de esta forma los motores y evitando que el control se realice de una manera correcta. Utilizando esta función en el segundo hilo evitará que esta función bloqueante afecte a la aplicación ya que solo se ejecutará cuando la primera función duerma porque ha llegado al *sleep ()* y saldrá del segundo hilo cuando el *sleep* haya pasado, por tanto aunque haya recibido algo o no, la función del control de la trayectoria sigue sin bloquearse.

La recepción se realiza de la siguiente manera:

```
recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
      xrecibido=Coordmensajer.posxr;
       posxr[i]=xrecibido;
```

```
yrecibido=Coordmensajer.posyr;
posyr[i]=yrecibido;
```

Teniendo la posición real del robot (que se envía al otro) y la posición recibida del otro robot, se calcula la diferencia entre ellos para luego decidir si parar el robot o seguir controlándolo.

Una vez terminado el bucle se paran los motores utilizando las funciones:

```
tacho_set_stop_action(OUTB,TACHO_HOLD);
tacho_set_command(OUTB, TACHO_STOP);
```

3.8. EXPERIENCIA 2: TRANSPORTE DE OBJETOS

Entre las funciones que pueden hacer los robots móviles para liberar al ser humano del trabajo duro, es transportar objetos. Pero en algunas ocasiones estos objetos son demasiado pesados como para que un solo robot lo pueda transportar y este trabajo lo deben hacer más de un robot. Para que el transporte se haga correctamente hace falta que los robots que transporten el objeto, estén sincronizados entre ellos y comuniquen su posición al otro robot para que de esta forma cada uno sepa la situación del otro y actuar, teniendo en cuenta donde está el otro.

En esta aplicación se va a transportar un vaso de plástico de forma que un robot sigue una trayectoria recta hacia delante y el otro lo hace hacia atrás sincronizándose con el que va hacia delante.

Fig. 3.12 Posición de los robots

Este problema se ha resuelto de una manera parecida a la primera aplicación. Lo primero que se ha hecho ha sido crear los sockets (explicado en el apartado 3.6) y los hilos (apartado 3.6). Se han creado dos hilos; el primero que se encarga del control de posición del robot para que siga la trayectoria deseada y el segundo hilo se encarga de recibir lo que el otro robot le ha enviado, para así posteriormente poder comparar la posición de cada uno de ellos y actuar en función de ello.

Tanto en el caso del robot que va hacia delante como el robot que va hacia atrás, el segundo hilo se ha programado igual que en el apartado 3.6 ya que es el hilo que se encarga de la recepción de datos.

El primer hilo del robot que va hacia delante entra en un bucle, cuya duración es de 12 segundos y aplica el control de la trayectoria ya explicado.

El primer hilo del robot que va hacia atrás también entra en el bucle que dura 12 segundos pero lo primero que se hace es comprobar que la distancia entre los dos robots no es mayor que la posición inicial más 0.1 metros. Esta comprobación se hace ya que si la distancia sería superior existiría la posibilidad de que el vaso se cayese y no se pudiera transportar correctamente. En el caso de que sea superior a 0.1 m el robot se para y se espera a que el otro se acerque un poco más, comprobándose continuamente la distancia entre los dos robots. Cuando la distancia es inferior a 0.1 m el robot sigue la trayectoria indicada.

Para control del robot hacia atrás, se han hecho pequeñas modificaciones al control de posición del robot que avanza hacia delante:

Se ha modificado la e cambiándola de signo (antes era 0.061 y en esta aplicación es -0.061) . Como a partir de la posición y velocidad de este punto se realiza el control, para que el robot se controle yendo hacia atrás es necesario fijar este punto en una posición negativa ya que de esta forma a la hora de aplicar los cálculos del control las acciones de control dan un resultado negativo y así las ruedas giran hacia atrás. En el caso de no cambiar esta variable, se hará el control adecuado pero las acciones de control que hay que aplicar a los motores saldrán positivas y de esta manera el robot seguirá la trayectoria girando hacia delante.

La otra modificación realizada es el cambio de los valores de referencia. En este caso la referencia será negativa.

```
refx= -(0.5+(1.500-0.500)*i/(num_itera));
```

3.9. EXPERIENCIA 3: MOVIMIENTO SINCRONIZADO

En el caso de las dos experiencias anteriores es difícil observar de una forma clara que uno de los robots se para con el propósito de esperar al otro ya que si los dos salen de una posición paralela y van a la misma velocidad existen muy pocos instantes de tiempo en el que la diferencia de posición sea mayor a la deseada y uno de los robot este obligado a parar hasta que el otro llegue. Como estos instantes de tiempo son tan pequeños no se observa muy bien a

simple vista. Por este motivo se ha decidido hacer esta experiencia y de esta forma demostrar la correcta sincronización de los dos robots.

Se coloca uno de los robots unos 40 cm por delante del otro. Tal como se puede observar en la siguiente figura

Fig. 3.13 Posición inicial de los robots

A la hora de enviar la posición del robot que va más adelantado, se le suma a la posición que tiene, los 40 cm de adelanto para que el robot más atrasado reciba la posición "real" del otro robot.

xreal=x+0.4;

Para calcular la diferencia de posiciones entre los dos robots se utiliza este valor "real "de la posición del robot adelantado.

diferencia=xreal-xrecibido;

El resto del programa es igual a la experiencia 1: se crean los sockets y los dos hilos. El primer hilo se encarga del control de la posición (si la distancia entre los dos robots es mayor que 0.1 se para el robot adelantado y si es menor se le aplica el control por punto descentralizado para que siga la referencia deseada) y el segundo hilo se encarga de la recepción de los datos enviados por el otro robot.

De esta manera, teniendo la posición inicial que se observa en la figura 3.13, el robot adelantado estará parado hasta que el otro robot esté en una posición paralela con él (aproximadamente) y una vez estando en esa posición paralela los dos robots emprenden su movimiento a la vez realizando la trayectoria indicada, empujando así la caja de una forma sincronizada.

Para que los dos terminen al mismo tiempo su trayectoria, se ha calculado, haciendo una regla de tres, el número de iteraciones que tarda el robot retrasado en recorrer 0.4 metros y se ha restado este tiempo al robot adelantado.

El número de iteraciones que recorre en 0.4 metros es de 240:

```
while(i < (num_itera-240))
```

Fig. 3.14 Posición final de los robots

3.10. EXPERIENCIA 4: AVISO DE LLEGADA

Hay ocasiones en las que se necesita liberar una zona de trabajo, para ello se pueden utilizar los robots con el fin de que "limpien" el área que se quiere dejar libre. Estos robots encuentran los objetos y los empujan en una zona destinada al almacén de estos. Pero hay veces en las que un solo robot no puede hacer la fuerza necesaria para conseguir desplazar el objeto sino que se necesita la ayuda de otro robot. El robot "ayudante" puede situarse en paralelo con el robot que ha encontrado el objeto, pero en ocasiones los objetos son tan estrechos que no permiten la posibilidad de poder ser empujados por dos robots a la vez. Si se da esa situación, el robot ayudante se tendrá que situar detrás del robot al cual quiere ayudar.

En esta experiencia se quiere solucionar el problema planteado y para ello se parte de la situación en la que un robot ha encontrado un objeto pero este no puede ser trasladado por la fuerza de un solo robot. En ese instante el robot se encuentra parado y envía su posición al otro robot para que este se sitúe detrás de él y lo ayude a empujar el objeto pesado y estrecho.

Así pues la posición inicial de los robots se puede observar en la figura 3.15; donde el robot situado a la izquierda es el que ha encontrado el objeto y está parado hasta que el otro robot llegué a posicionarse detrás de él. El robot de la derecha es el robot "ayudante "que parte del punto inicial (0, 0.8).

Fig. 3.15 Posición inicial de los robots

En esta aplicación se ha utilizado el mismo esquema que en los experimentos anteriores.

Se ha creado sockets y dos hilos: el primero que se ocupa del control de la posición y el segundo que recibe datos enviados por el otro robot.

El robot que encuentra el objeto, se trata como cliente y el que acude a ayudarle se trata como servidor.

En el caso del robot que hace de servidor dentro del primer hilo (el hilo del control), lo primero que se hace es una recepción de la posición del otro robot. A este posición se le resta en x, 0.15 metros que es la medida aproximada de un robot Lego, ya que de esta forma se puede situar el robot detrás del otro. Esta posición es el punto final de la curva de Bézier que se hace para que desde el punto inicial el robot ayudante llegue al punto final que es el punto donde el robot se encuentra situado en la parte trasera del otro.

En este caso se hace una excepción y se pone la función de recibir dentro del hilo de control pero antes de iniciar el bucle *while (),* esto se hace ya que en este caso como no está situado dentro de este bucle la función que es bloqueante no influye en el control de la trayectoria y de esta forma se asegura que se ha recibido correctamente la posición del otro robot.

Así pues se crea una curva de Bézier de 3 puntos cuyos puntos son los siguientes:

```
P0x=0.00;
P1x=0.0;
      recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
      P2x=Coordmensajer.posxr;
       P2x=P2x-0.2;
```

P2y=Coordmensajer.posyr;

```
P0y=0.8;
P1y=0.0;
P2y=Coordmensajer.posyr;
```

Se calcula el número de iteraciones necesarias para que con un periodo de muestreo de 0.02 segundo, la aplicación dure 12 segundos y se genera el bucle *while ( )* que dura dos veces el número de iteraciones calculado, esto es, el doble de tiempo (24 segundos). En los primeros 12 segundos el robot hace la curva de Bézier llegando a la posición deseada, es decir, se sitúa detrás del otro robot. En estos primero 12 segundos, se envía al robot parado el valor de *k=2* para que este robot lo reciba como una señal de que tiene que quedarse parado ya que el robot "ayudante" no ha llegado a posicionarse detrás de él.

En los 12 últimos segundos, se crea una trayectoria recta, que es igual a la trayectoria del otro robot, para que de esta forma una vez situado detrás del robot principal los dos puedan emprender la misma trayectoria empujando el objeto pesado. En este tiempo se envía al otro robot un valor de *k=1* para avisarle de que ha llegado a posicionarse en el lugar deseado y a partir de ese momento los dos pueden iniciar su trayectoria recta.

```
while(i <2*num_itera){
      if(i<=num_itera){
      parat=(i*0.02)/tfinal;
      paratcua=parat*parat;
      refx=((1.0-2.0*parat+paratcua)*P0x)+((2.0*parat-
      2.0*paratcua)*P1x)+paratcua*P2x;
      refy=((1.0-2.0*parat+paratcua)*P0y)+((2.0*parat-
      2.0*paratcua)*P1y)+paratcua*P2y;
      velxref=(2*P1x-2*P0x)+2*(P0x-2*P1x+P2x)*parat;
      velyref=(2*P1y-2*P0y)+2*(P0y-2*P1y+P2y)*parat;
      k=2;
       }
      else{
      refx=(P2x+((1+P2x)-P2x)*(i-num_itera)/num_itera);
      refy=0.0;
      velxref=(refx-refxante)/0.02;
      velyref=0;
      k=1;
```

```
}
```

En el caso del robot que hace de cliente, que es el robot principal (el que espera al otro para que le ayude). Lo primero que hace una vez entrado dentro del primer hilo es enviar su posición al robot "ayudante"

```
Coordmensajee.posxe=x;
Coordmensajee.posye=y;


send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
      posxe[i]=Coordmensajee.posxe;
      posye[i]=Coordmensajee.posye;
```

Posteriormente comprueba el valor de la k que ha recibido. Si la k recibida es igual a 2, esto significa que el otro robot todavía no ha llegado y por tanto este robot se queda parado. Si la k recibida es igual a 1, esto quiere decir que el robot ha llegado a situarse detrás y por tanto puede empezar a empujar el objeto con la ayuda del otro robot.

```
while(i < (num_itera))  {
      if ((k==2) && z==0){
            tacho_set_stop_action(OUTA,TACHO_HOLD);
            tacho_set_stop_action(OUTD,TACHO_HOLD);
            tacho_set_command(OUTA, TACHO_STOP);
            tacho_set_command(OUTD, TACHO_STOP);

 }
      if(k==1 || z==1){
             z=1;
             refx=(0.5+(1.500-0.500)*i/(num_itera));
             refy=0;
             velxref=(refx-refxante)/0.02;
             velyref=0;
```

Fig. 3.16 Posición final de los robots

3.11. EXPERIENCIA 5: SEGUIDOR DE TRAYECTORIA

En la actualidad existen varios sensores que pueden ser bastante caros y que, si se puede, es mejor prescindir de ellos ya que así se ahorra dinero y esto siempre conviene a la hora de realizar un proyecto.

Uno de los ejemplos en los que se podría ahorrar sensores es el caso en el que varios robots siguen una misma trayectoria y que presentan sensores con el fin de esquivar los obstáculos que existen en su camino. En este caso se podría ahorrar sensores ya que una opción sería que uno de los robots estuviera un poco avanzado al resto y tuviera el sensor, y al detectar el obstáculo siguiera una trayectoria para evitarlo, enviando a los demás robots (que no tienen sensor) el valor de los puntos de la trayectoria que ha seguido para que de esta forma los otros robots siguiesen su misma trayectoria y así ,sin tener el sensor, ellos también puedan evitar el obstáculo.

Así pues, en este caso solo haría falta un sensor en vez de utilizar uno para cada robot.

Para simular esto se ha realizado la siguiente aplicación.

Se sitúan los dos robots en paralelo, como se u observa en la siguiente figura 3.17 y se programa para que uno de los robots vaya más adelantado que el otro (es el robot que tendría el sensor). Este sigue una trayectoria recta hasta que se simula que se detecta un obstáculo y gira a la izquierda. En todo este momento el robot que "no tiene el sensor" va un poco más atrasado y de esta forma recibiendo los valores de posición del robot adelantado, sigue la misma trayectoria que este, evitando así el obstáculo sin tener el sensor.

Fig. 3.17 Posición inicial de los robots

Para ellos se ha seguido la misma estructura que en el resto de las aplicaciones, se han creado sockets y dos hilos. El primero que se encarga del control y el segundo que se encarga de recibir.

El robot más adelantado es el servidor y el otro es el cliente.

En el caso del servidor, la duración de la trayectoria es de 24 segundos, igual que en la experiencia anterior. En los 12 primeros segundos se sigue una línea recta en el eje x, mientras que en los 12 últimos segundos, se simula como una detección de un obstáculo, y el robot gira a la izquierda siguiendo una línea recta en el eje y.

```
if (i< num_itera)
            {
       refx=1+(2.000-1.000)*i/(num_itera);
       refy=1;
       velxref=(2.000-1.000)/(num_itera);
       velyref=0;
       }
else            {
      refx=2.000;
      refy=1+(2.000-1.0)*(i-(num_itera))/(num_itera);
      velxref=0;
      velyref=(2.000-1)/(num_itera);
      }
```

En todo este tiempo, se envía la posición de este robot al otro que está más atrasado.

```
      Coordmensajee.posxe=x;
      Coordmensajee.posye=y;
  send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
      posxe[i]=Coordmensajee.posxe;
      posye[i]=Coordmensajee.posye;
```

Mientras tanto, el robot atrasado que hace de cliente se inicializa con una posición en x desplazada -0.2 metros ya que esta es la posición inicial del robot al situarse paralelamente al otro.

La posición inicial del robot adelantado es:

```
x=1.0;
y=1.0;
```

Mientras el otro robot se inicializa en:

```
x=0.8;
y=1.0;
```

Además en este segundo robot también se inicializa la referencia en x y la referencia en y, en esos mismos valores. Esto se hace ya que en un primer instante el robot no recibe la posición del otro robot y necesita una referencia para hacer el bucle, de esta forma se le pasa la referencia que es la misma que su posición inicial y por tanto el robot se queda parado hasta el instante en el que recibe la posición del otro robot. Así es como se consigue que vaya más atrasado un poco.

```
refx=0.8;
refy=1;
```

Al entrar en el bucle primero se comprueba si se ha recibido algo, ya que existen ciertos instantes en los que a la función que recibe, no le llega ningún parámetro. Si no se ha recibido nada, se mantiene la referencia anterior y de este modo el robot no se mueve. Como el caso de que no se recibe nada se da en pocas ocasiones no se observa a simple vista que el robot se ha parado.

Si recibe la posición del otro robot, para este robot será la referencia que ha de seguir. En el caso de la referencia en x, como el robot atrasado está desplazado -0.2 metros respecto del otro, a la posición x recibida se le resta 0,2 y esta será la referencia del robot.

```
while(i < (2*num_itera)){
      if(xrecibido==0.0 || yrecibido==0.0){
                  refx=refx;
                  refy=refy;
```

```
}
else {
            refx=xrecibido-0.2;
            refy=yrecibido;
      }
```

De esta forma se consigue "imitar" la trayectoria del robot, que se "supone" que tiene el sensor, y evitar así el obstáculo sin poseer el sensor.

Fig. 3.18 Posición final de los robots

3.12. VERIFICACIÓN TIEMPO DE MUESTREO

Para que las experiencias funcionen de una forma correcta se debe cumplir el periodo de muestreo en la parte del control de la trayectoria. Como en el caso de los hilos, es el propio sistema el que le da el "turno" a cada uno de ellos, se ha considerado que sería adecuado comprobar el cumplimiento de este periodo de muestreo y de esta forma poder demostrar que el programa cede los turnos a los hilos tal como se piensa: cuando se hace el *sleep ( ),* el segundo hilo entra en acción recibiendo los datos enviados por el otro robot, cuando se termina el *sleep ( )* se sale del segundo hilo independientemente de si se ha recibido algo o no, de esta manera se evita que la función que recibe sea bloqueante y pare el sistema alterando el tiempo de muestreo.

Para asegurarse que este tiempo de muestreo se cumple, primero se ha hecho una prueba sin recibir nada, es decir sin el segundo hilo. Esto se ha hecho para asegurarse de que el control de la trayectoria de hace de una forma correcta.

Para medir tiempo se ha utilizado la función ftime() que mide la UTC actual (Coordinated Universal Time) y lo coloca en la estructura timeb.

Para ello, al principio del control (la primera línea dentro del bucle while ()) se ha guardado en una variable (t1) el valor que devuelve la función ftime() (pasándolo a milisegundos) y en otra variable(t1ante) se guarda el valor del t1 medido en la iteración anterior. Para obtener el tiempo transcurrido se resta el tiempo medido en la iteración actual menos el tiempo medido en la iteración anterior y de esta forma se averigua el tiempo que pasa entre una iteración y otra. Este tiempo debería ser 20 milisegundos, ya que este es el periodo de muestreo elegido.

```
#include<sys/timeb.h>
.
.
.
struct timeb tb;

.
.
.
while(i < num_itera){



      ftime(&tb);
      int t1=tb.millitm + ((tb.time & 0xfffff) * 1000);
      tf[i]=t1-t1ante;
      t1ante=t1;
.
.
.

i++;
      }
```

Después de hacer la medida sin los hilos, se pasa a hacer la medida con los hilos de la misma que forma que se ha explicado anteriormente. Con todos estos datos se obtiene una gráfica donde se puede observar el tiempo transcurrido entre las iteraciones del bucle.

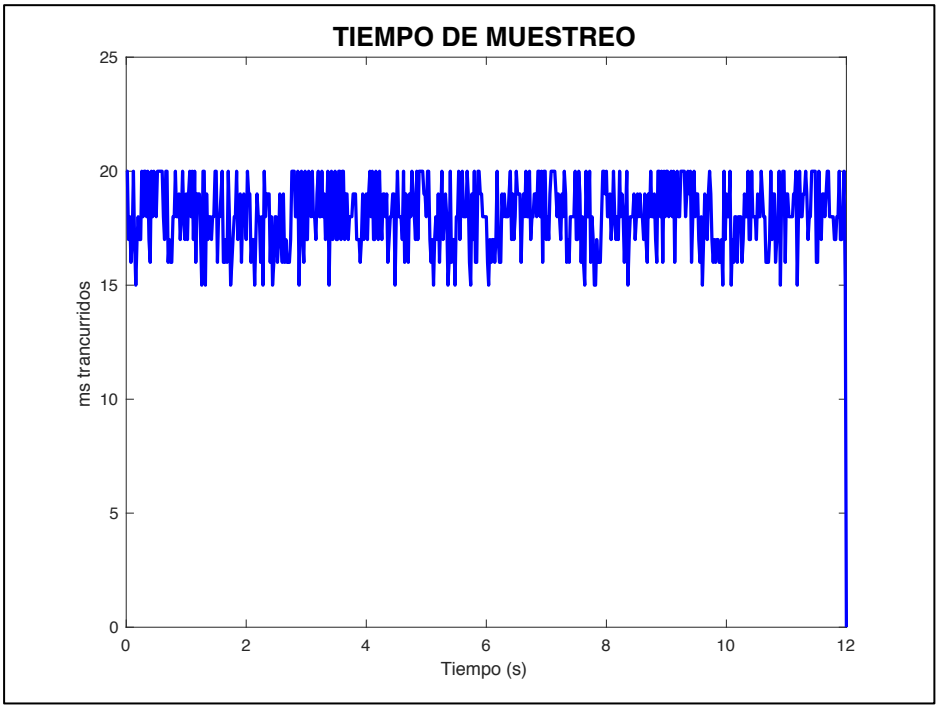

Fig. 3.19 Tiempo de muestreo sin utilizar hilos

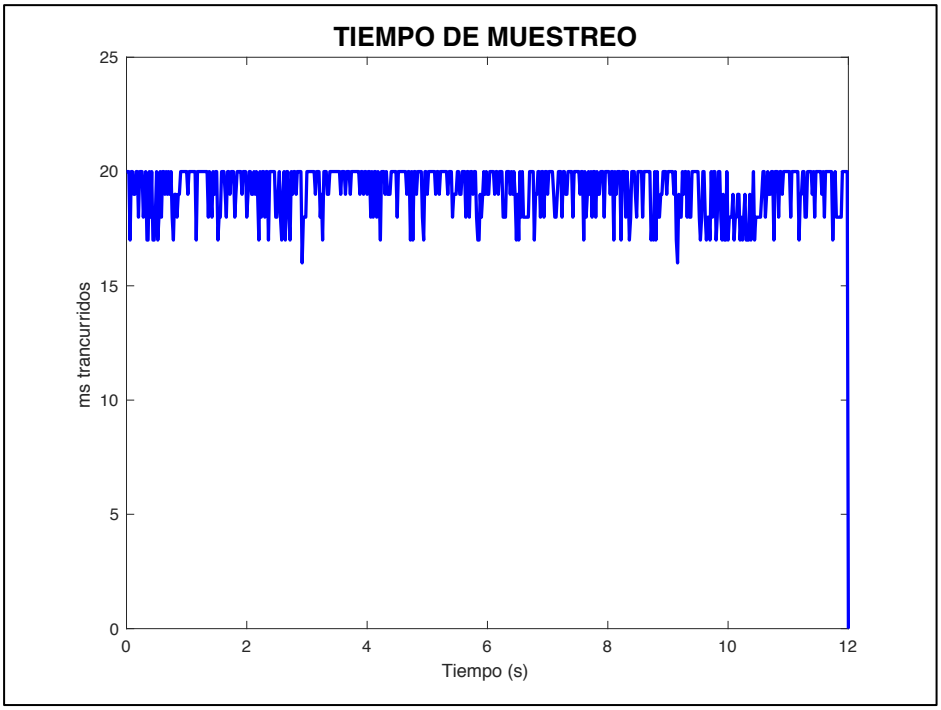

Fig. 3.20 Tiempo de muestreo utilizando hilos

Tal como se puede observar, en ambos casos se cumple el periodo de muestreo y no existe variación entre los datos obtenidos con los hilos y los datos sin utilizar los hilos. Por tanto se podría decir que funciona de una manera adecuada.

4. CONCLUSIONES Y PROYECTOS FUTUROS

A lo largo de este proyecto se han aprendido cosas nuevas que antes de iniciar este proyecto eran desconocidas y que han ayudado a alcanzar el objetivo final del proyecto.

En primer lugar se ha conocido y se ha intentado familiarizar con el funcionamiento del sistema operativo Linux que es el sistema instalado en los ladrillos de los robots utilizados

En segundo lugar, se ha aprendido a utilizar y a programar los motores de esta nueva generación de LEGO EV3 ya que se utilizan funciones que en las versiones anteriores no se pueden utilizar.

Se ha aprendido a realizar trayectorias de diferentes formas, como por ejemplo, utilizando las curvas de Bézier

Se ha entendido el funcionamiento de los sockets y el uso que tienen. Además se ha aprendido a crear sockets, en este caso utilizando el lenguaje de programación elegido, el C.

Otros de los conceptos que no se conocía era la programación multitarea, en este proyecto se ha entendido el funcionamiento de la programación multitarea y se ha aprendido a programar los hilos, que forman parte de este tipo de programación.

Finalmente, se ha aprendido a relacionar todos los conceptos citados anteriormente, con el fin de crear las aplicaciones deseadas y conseguir, de esta forma, alcanzar al objetivo final del proyecto

En conclusión, podemos decir que el objetivo final de este proyecto se ha cumplido ya que se han creado exitosamente aplicaciones en las que varios se transmiten datos para cooperar entre ellos y realizar un trabajo que uno solo no sería capaz de realizar.

Como trabajos futuros se podría extender este trabajo y utilizar más de dos robots. Así se conseguirá resolver aplicaciones mucho más complejas que las diseñadas con dos robots o también se podrán transportar objetos que ni siquiera dos robots podrían hacerlo.

Esta programación también se podría aplicar a robots industriales y de esta forma no solo simular la aplicación (como se ha hecho en este proyecto) sino poder trasladarlo a la vida real y que de esta se pueda aprovechar los avances tecnológicos utilizados en este proyecto para realizar en menor tiempo con menor esfuerzo, actividades que al ser humano le es muy pesado hacer.

5. BIBLIOGRAFÍA

5.1. Documentación

[1] Real Academia española. Definición de Robot. http://dle.rae.es/?id=WYRlhzm
[2] Tipos de Robots. http://10tipos.com/tipos-de-robots/
[3] Locomoción de robots móviles http://www.tamps.cinvestav.mx/~mgomez/Odometria/node2.html
[4] Interpolación con funciones Spline http://www4.ujaen.es/~angelcid/Archivos/An_Met_Num_INFORMATICA/Splines.pdf
[5] Curva de Bézier https://es.wikipedia.org/wiki/Curva_de_Bézier
[6] Navegación de un robot móvil de configuración diferencial basada en función sensorial
[7] https://riunet.upv.es/bitstream/handle/10251/15617/tesina_leonardomarin.pdf?sequence=1
[8] Simulación de vehículos eléctricos ligeros https://riunet.upv.es/bitstream/handle/10251/18173/Memoria.pdf?sequence=1
[9] Lego Mindstorms https://es.wikipedia.org/wiki/Lego_Mindstorms
[10] Programación del EV3 http://blog.electricbricks.com/2016/03/programacion-del-ev3/
[11] Programming Languages https://www.ev3dev.org/docs/programming-languages/
[12] ¿Cómo funciona el SSH? https://www.hostinger.es/tutoriales/que-es-ssh
[13] Down load PuTTY http://putty.org
[14] WinSCP https://es.wikipedia.org/wiki/WinSCP
[15] Sistemas de comunicación inalámbricas https://programarfacil.com/podcast/65-sistemas-de-comunicacion-inalambricas/
[16] ¿ Qué es Bluetooth y para qué sirve ? http://www.valortop.com/blog/bluetooth
[17] Protocolos UDP y TCP https://es.wikibooks.org/wiki/Redes_informáticas/Protocolos_TCP_y_UDP_en_el_nivel_de_transporte
[18] Protocolo de control de transmisión https://es.wikipedia.org/wiki/Protocolo_de_control_de_transmisión
[19] Comunicación entre procesos: sockets http://sopa.dis.ulpgc.es/ii-dso/leclinux/ipc/sockets/sockets.pdf
[20] ¿Qué es un socket ? http://www2.electron.frba.utn.edu.ar/~mdoallo/descargas/redes.pdf
[21] Arquitecturas de red http://blog.uclm.es/inocentesanchez/files/2017/02/arquitectura.pdf
[22] Servicios IP https://docs.oracle.com/cd/E24842_01/html/820-2981/ipov-6.html
[23] Sistemas Operativos: Programación de sistemas http://sopa.dis.ulpgc.es/progsis/material-didactico-teorico/tema7_1transporpagina.pdf

[24] Sockets: http://sopa.dis.ulpgc.es/ii-dso/leclinux/ipc/sockets/LEC_SOCKETS.pdf
[25] Multitarea-Conceptos básicos http://jmaw.blogspot.com/2012/07/multitarea-conceptos-basicos.html
[26] Linux, threads http://cortesfernando.blogspot.com/2011/11/linux-threads.html
[27] Características de los hilos http://www.chuidiang.org/clinux/sockets/sockets_simp.php
[28] Raúl Simarro, apuntes asignatura Control de Sistemas Mecatrónicos.
[29] Sockets en C- Linux https://www.programacion.com.py/noticias/sockets-en-c-parte-i-linux
[30] Programación de sockets en C http://www.chuidiang.org/clinux/sockets/sockets_simp.php
[31] Getting Started with ev3ev https://www.ev3dev.org/docs/getting-started/
[32] Ev3dev https://www.ev3dev.org
[33] Etcher https://etcher.io
[34] PuTTy https://putty.org
[35] WinSCP descargar https://winscp.net/eng/docs/lang:es
[36] "Unix Programación Práctica. Guía para la Concurrencia, la Comunicación y los Multihilos". Kay A. Robbins, Steven Robbins. Ed. Prentice-Hall Hispanoamericana
[37] Colegio de ingenieros técnicos industriales de Vizcaya https://www.coitibi.net/servicios-profesionales/ejercicio-profesional/calcula-precio-hora-de-tu-trabajo
[38] Colegio de ingenieros técnicos industriales de Valencia https://copitival.es
[39] Claudia Anais, desarrollo de una aplicación para el guiado automático de un vehículo eléctrico

5.2. Imágenes

[1] Robot poli articulado http://cmapspublic2.ihmc.us/rid=1QKF9KCPC-1G4V973-5WNY/poliarticulados.jpg
[2] Robot móvil https://www.robotnik.es/robots-moviles/
[3] Robot android https://www.drim.es/animales-roboticos/robot-helice-androide-121193.html
[4] Robot zoomórfico http://roboticasigloxxi.blogspot.com/2011/03/robot-zoomorfico.html
[5] Configuración Akerman https://www.xatakaciencia.com/robotica/robots-moviles-i
[6] Configuración triciclo https://www.xatakaciencia.com/robotica/robots-moviles-i
[7] Configuración omnidireccional http://www.muchotrasto.com/TiposDePlataformas.php
[8] Configuración síncrona http://www.esi2.us.es/~vivas/ayr2iaei/LOC_MOV.pdf
[9] Tracción diferencial https://www.researchgate.net/figure/Figura-3-Esquema-del-Robot-Movil-Copyright-R-Ramos-Morales-2010_fig3_316734735
[10] Interpolación https://www.taringa.net/comunidades/matlab/8594410/Aporte-Interpolacion-Metodo-de-Newton.html

[11] Curva Bézier http://w3.unpocodetodo.info/svg/cubic-bezier.php
[12] Lego RCX https://www.electricbricks.com/lego-piezas-lego-technic-electrico-9709-mindstorms-rcx-completo-con-conector-alimentacion-p-1015.html
[13] Lego NXT https://shop.lego.com/cs-CZ/NXT-Intelligent-Brick-9841
[14] Lego EV3 https://www.robotix.es/es/bloque-inteligente-ev3
[15] Navegación de un robot móvil de configuración diferencial basada en función sensorial
[16] https://riunet.upv.es/bitstream/handle/10251/15617/tesina_leonardomarin.pdf?sequence=1
[17] Todo sobre el protocolo SSH https://universo-digital.net/todo-sobre-protocolo-ssh-gnu-linux/
[18] Comunicación entre procesos: sockets http://sopa.dis.ulpgc.es/ii-dso/leclinux/ipc/sockets/sockets.pdf
[19] Sistemas Operativos: Programación de sistemas http://sopa.dis.ulpgc.es/progsis/material-didactico-teorico/tema7_1transporpagina.pdf
[20] Sockets, Dominio : http://sopa.dis.ulpgc.es/ii-dso/leclinux/ipc/sockets/LEC_SOCKETS.pdf

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado en Ingeniería Electrónica Industrial y Automática

DOCUMENTO 2 : PRESUPUESTO

Escuela Técnica Superior de Ingeniería del Diseño

Universidad Politécnica de Valencia

Valencia, Julio 2018

1. PRESUPUESTO

1.1 NECESIDAD DEL PRESUPUESTO

En la mayoría de proyectos, es necesario conocer el coste económico que supondría el hecho de poner en marcha el proyecto diseñado ya que así se puede deducir lo beneficioso que sería reproducirlo. Para sacar las conclusiones sobre el beneficio primero se debe sacar el presupuesto de lo que costaría el proyecto. A continuación se va a exponer el cálculo del presupuesto realizado para este proyecto. Para ello se va a dividir en tres partes: coste del personal, material inventariable, material fungible.

1.2 COSTE DEL PERSONAL

Para este cálculo se va a tener en cuenta el coste por hora del trabajo de un ingeniero y el número de horas que le ha dedicado al trabajo. Con el fin de obtener el coste del personal se multiplican los dos factores.

Para el coste por hora del trabajo de un ingeniero, se ha considerado una media de 35€ /h según los datos obtenidos de diferentes colegios de ingenieros técnicos industriales [37,38].

Dentro de este precio ya están incluidos los costes de la cotización a la Seguridad Social.

TRABAJO	TIEMPO (h)	COSTE (€/h)	TOTAL(€)
Instalación del sistema operativo en el ladrillo	1	35	35
Instalación WINSCP y PuTTY	1	35	35
Conexión del ladrillo con el PC	1	35	35
Programación en el PC	120	35	4200
Simulación en el Lego	50	35	1750
Redacción de los documentos	60	35	2100
TOTAL			**8155**

1.3 MATERIAL INVENTARIABLE

En este apartado se calcula la amortización de los equipos que deben ser comprados específicamente para el proyecto presupuestado, según el Centro de apoyo a la Innovación, la Investigación y la Transferencia de Tecnología [39] el periodo de amortización para material informático se considera de 5 años y para el resto de equipos 10 años.

Para estos cálculos se ha utilizado la siguiente expresión:

$$\text{Coste} = \frac{Número\ de\ meses\ de\ uso}{periodo\ de\ amortización\ (meses)} * Coste\ del\ equipo * Porcentaje\ de\ uso\ en\ el\ proyecto$$

Para calcular el porcentaje de uso del material se ha calculado teniendo en cuenta el número de horas que se usa, entre el número de horas totales del proyecto.

Equipo	Tiempo uso (meses)	Amortización (meses)	Coste(€)	%uso	TOTAL(€)
Ordenador de mesa HP	5	60	1000	100	83,333
3 Adaptadores W-FI	5	120	6,05	70	3·0,1764
Router Wi-FI	5	120	23,99	70	0,6997
2 Tarjetas MicroSD 32 GB	5	120	5,90	70	2·0,1720
Kit LEGO EV3	5	120	70	400	11,6667
TOTAL					**96,57**

1.4 MATERIAL FUNGIBLE

Se trata como material fungible a todo aquel material que se ha utilizado en el proyecto y que es de corta vida útil.

MATERIAL	PRECIO (€)
Impresión memoria	2
Encuadernación	7
Folios	3,5
TOTAL	**12,50**

1.5 RESUMEN DEL PRESUPUESTO

CONCEPTO	TOTAL (€)
Coste de personal	8155.00
Material Inventariable	96,57
Material Fungible	12,50
TOTAL SIN IVA	8264,07
IVA (21%)	1735,45
TOTAL CON IVA	**9999,52**

Así pues, el presupuesto para la realización de este proyecto es de "NUEVE MIL NOVECIENTOS NOVENTA Y NUEVE EUROS CON CINCUENTA Y DOS CÉNTIMOS"

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado en Ingeniería Electrónica Industrial y Automática

DOCUMENTO 3 : ANEXOS

Escuela Técnica Superior de Ingeniería del Diseño

Universidad Politécnica de Valencia

Valencia, Julio 2018

ANEXO I: INSTALACIÓN Y CONEXIÓN DEL LADRILLO

La información sobre la instalación del sistema operativo ev3dev en el EV3, se encuentra también en la página web de este sistema [32].

Para empezar se necesitan los siguientes elementos:

1. El LEGO MINDSTORMS EV3, que es el que se ha utilizado para este proyecto. En este caso se han utilizado dos ladrillos.
2. Dos microSD de 32GB. Una para cada ladrillo.
3. Un ordenador que tenga un adaptador para la microSD.
4. Tres adaptadores USB Wi-Fi. Uno para el ordenador y los otros dos para los ladrillos.
5. Un router que tenga conexión a internet y al cual se conectarán los tres adaptadores Wi-Fi.

El primer paso consiste en descargar la imagen que se debe flashear. La descarga se realiza a partir de la opción que existe en la página web [32]

Fig. 7.1 Botón que se pulsa para descargar la imagen [32]

Una vez descargada la imagen se pasa a flashearla. Para ello es necesario descargar y instalar el programa Etcher de su página web [33]. Cuando ya haya finalizado la descarga se abre el programa y aparece una pantalla como la de la siguiente figura.

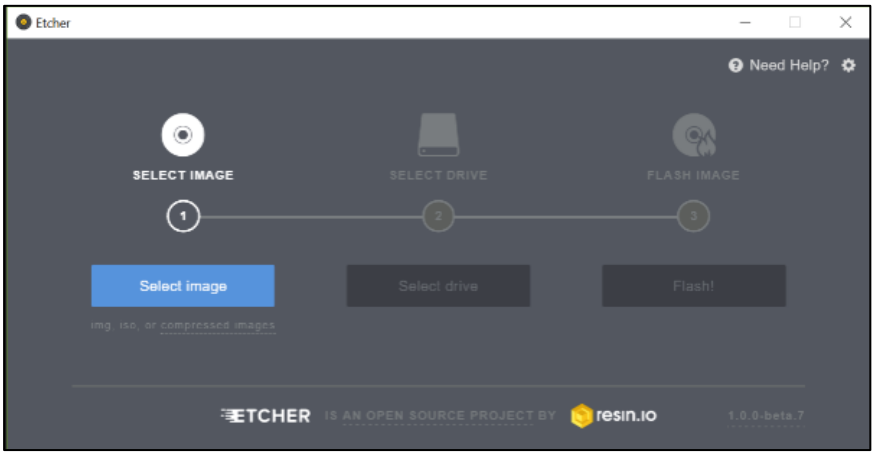

Fig. 7.2 Primer paso [32]

Se clickea sobre el botón “Select image”, se busca y se selecciona la imagen que se ha descargado en el primer paso.

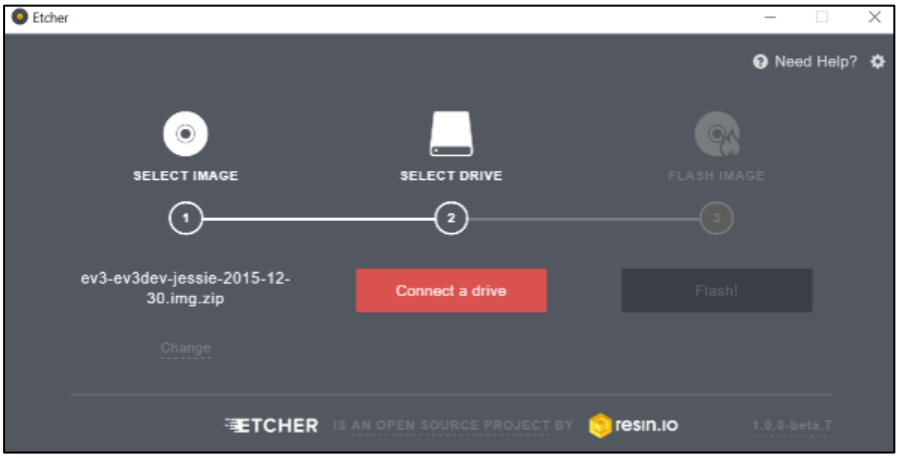

Fig. 7.3 Segundo paso [32]

Posteriormente se introduce la microSD en el ordenador y apretando el botón “Connect a drive”, se selecciona la tarjeta en la cual se va a flashear la imagen.

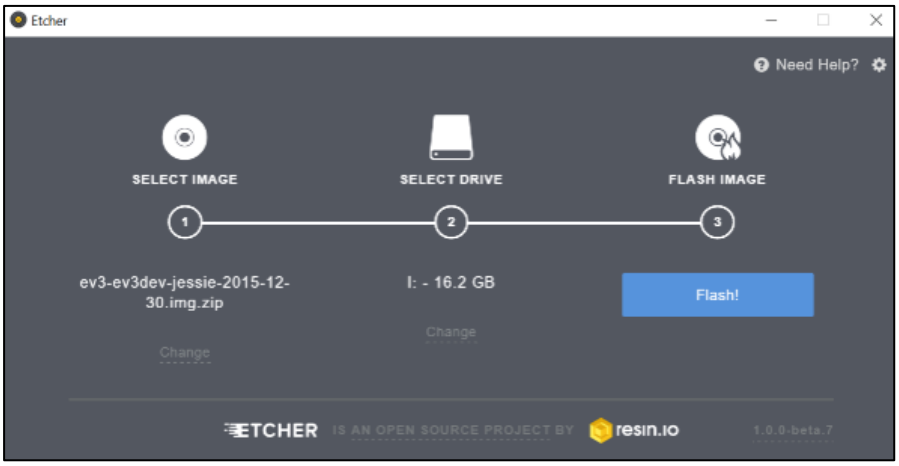

Fig. 7.4 Tercer paso [32]

Una vez seleccionada la tarjeta se hace click sobre el botón “Flash” y se espera a que termine el proceso que se ha iniciado.

Si todo se ha realizado correctamente aparecerá la siguiente pantalla:

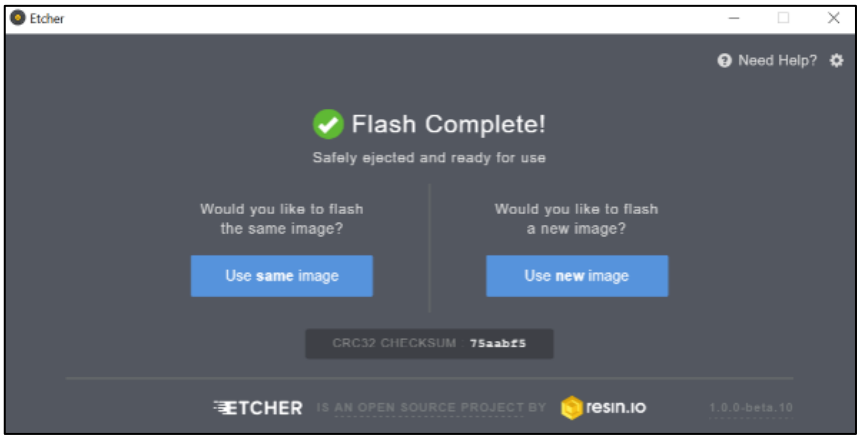

Fig. 7.5 Pantalla que aparece al terminar de flashear la imagen [32]

Se introduce la microSD en el Lego y se enciende. Al principio se encienden las luces rojas pero posteriormente pasarán a ser naranjas. Después de un minuto, la pantalla se volverá blanca y si se espera unos minutos más se observará la pantalla de carga del ladrillo. Al pasar unos minutos, si todo se ha realizado con éxito, aparecerá la siguiente pantalla y las luces serán verdes.

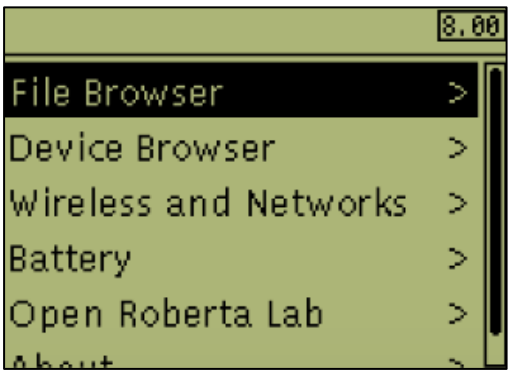

Fig. 7.6 Pantalla principal del ladrillo [32]

Se pasa a realizar la conexión a internet, de los ladrillos. Para ello se enciende el router y se introduce el adaptador de Wi-Fi en el ladrillo. A continuación, usando los botones del ladrillo, se accede al menú "Wireless and Networks -> Wi-Fi" y se selecciona la red a la que se conectarán el resto de ladrillos y el ordenador, que será la red del router.

Una vez realizados los pasos anteriores se pasa a conectar el EV3 vía SSH. Para ello se utiliza el programa PuTTY, el cual se descargará de la siguiente página [34]. La primera vez que se abre aparece el siguiente cuadro de diálogo (figura 7.7), en el que se debe seleccionar la opción "Yes". El usuario que se pide es "robot" y la contraseña "maker".

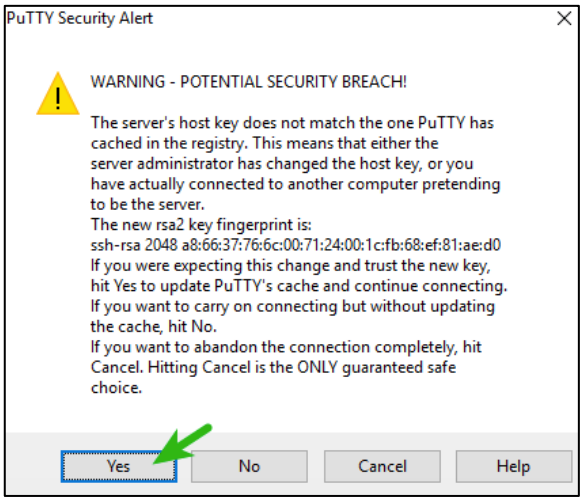

Fig. 7.7 Cuadro de diálogo que aparece la primera vez [32]

El siguiente paso es elegir el lenguaje de programación que se desea utilizar. En este caso se elige C.

En la ventana de PuTTY se escriben los siguientes comandos, para instalar el compilador y otras herramientas:

```
sudo apt-get update
sudo apt-get install build-essential
sudo apt-get install git
```

A continuación se copia las librerías y los submódulos en el directorio *home/robot/ev3de-c*

```
cd /home/robot/
git clone https://github.com/in4lio/ev3dev-c.git
cd ev3dev-c/
git submodule update --init --recursive
```

Y finalmente, se compilan y se instalan las librerías.

```
cd source/ev3/
make
sudo make install
make shared
sudo make shared-install
```

Para tener todos los programas necesarios, también hace falta descargar el WinSCP que sirve para transferir el archivo con el código, desde la carpeta de nuestro escritorio a la memoria del ladrillo.

La descarga se hace a partir de la página web [34].

ANEXO II: MANUAL DE USUARIO

Una vez instalados todos los programas necesarios, a continuación se explicarán los pasos a seguir para poner en funcionamiento las aplicaciones ya explicadas.

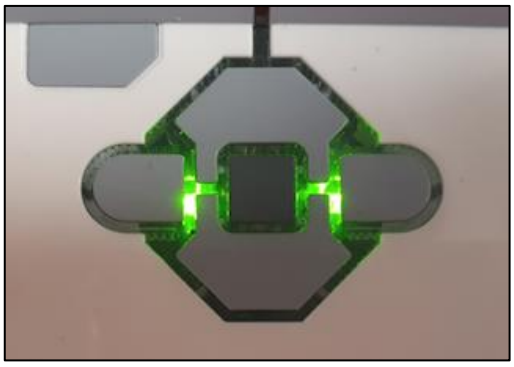

Fig. 8.1 Botones del ladrillo

Para empezar, se pulsa el botón central del ladrillo que sirve para encenderlo y se espera a que aparezca la pantalla principal (figura 7.6) y en esta pantalla, en la parte izquierda superior aparecerá la dirección IP que se utilizarán para realizar las correspondientes conexiones. En la parte derecha superior aparece el voltaje que le queda a la batería, se debe tener en cuenta que cuando este voltaje sea inferior a 5 el ladrillo se apagará.

Mientras tanto se enciende el ordenador y se introduce el adaptador Wi-Fi que se conectará a la misma red a la cual están conectados los ladrillos también.

Se abre el PuTTY y se teclea la dirección IP del ladrillo en la parte del "Host name".

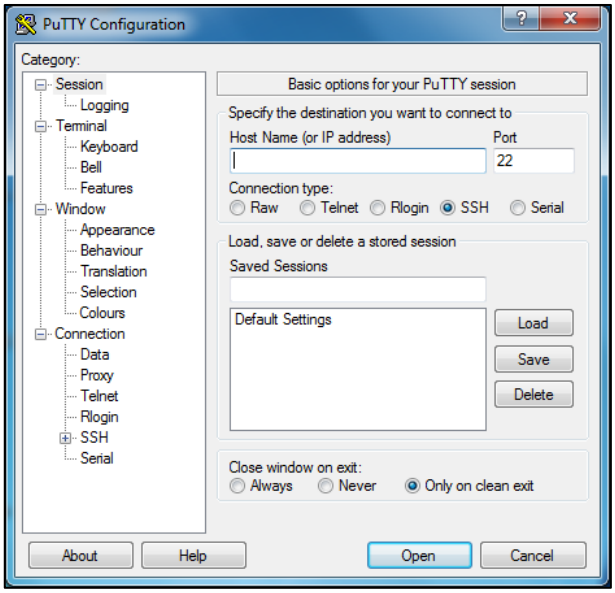

Fig. 8.2 Pantalla principal al abrir el PuTTy

A continuación se introduce como usuario “robot” y como contraseña “maker”.

Una vez realizados todos estos pasos se accede a la carpeta que contiene el archivo donde está escrita la programación (se refiere a la carpeta del ladrillo, es decir, la carpeta que está en la parte derecha de la ventana del WinSCP). Para ello se utiliza el comando `cdd` y se va introduciendo el directorio dónde está ese archivo. Así, por ejemplo, si queremos acceder al archivo `hilo_servidor.c,` que en este caso está en la carpeta servidor que está a su vez está en la carpeta `eg` y está a su tiempo está en `ev3dev-c`, se escribiría el siguiente comando.

```
cdd ev3dev-c/eg/servidor/
```

Posteriormente se pasa a compilar el archivo que está dentro de la carpeta servidor y que se denomina `hilo_servidor.c,` utilizando el siguiente comando:

```
gcc hilo_servidor.c  -lev3dev-c -lm -lpthread -o hilo_servidor
```

gcc: es el compilador que se utiliza en este caso.

hilo_servidor.c: es el nombre del archivo que se quiere compilar

–lev3dev-c –lm –lpthread: son las librerías necesarias para este caso, la primera es la librería del propio ladrillo, la segunda es la librería que contiene las funciones trigonométricas utilizadas y la tercera es la librería de los hilos creados.

–o hilo_servidor: se trata de la creación del archivo compilado, cuyo nombre será hilo_servidor. No es necesario que se denomine de la misma forma que el archivo que se compila.

En el caso de que no se presente ningún error al realizar la compilación, se pasa a la ejecución que se introduce como ***./*** seguido del nombre del archivo que se ha creado en la compilación, en este caso hilo_servidor.

```
·/hilo_servidor
```

Si todo se ha realizado correctamente el robot sigue las instrucciones introducidas en el archivo de programación.

Antes de realizar la compilación con el PuTTY, se abre el WinSCP.

En la parte de “host name” se introduce la dirección IP del ladrillo al cual se quiere conectar. Igual que en el caso del PuTTY, en la parte del usuario se introduce “robot” y en la parte de contraseña “maker”. A continuación se pulsa el botón “Login”.

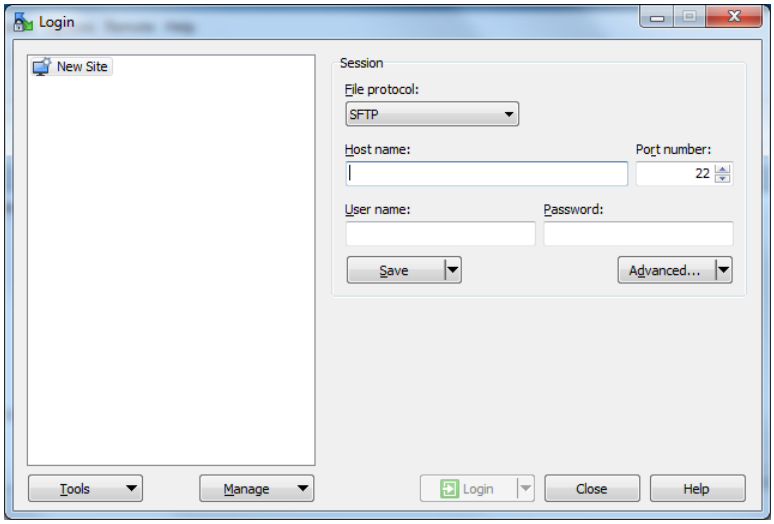

Fig. 8.3 Pantalla inicial del WINSCP

Una vez realizada la conexión con el ladrillo, en la parte izquierda, de la ventana abierta (figura 8.4), se accede al directorio del sitio donde está el archivo con el programa escrito. Debe ser un archivo .c, que para modificarse se abre con el editor de texto. Posteriormente en la parte derecha se accede al directorio donde dejaremos el archivo .c que debe ser compilado. El directorio debe ser una de las carpetas que hay en el ladrillo.

El archivo de la parte izquierda se arrastra y se pasa a la parte derecha. Una vez terminado este proceso se puede pasar a realizar la compilación con el comando `gcc` (ya explicado) en el PuTTY.

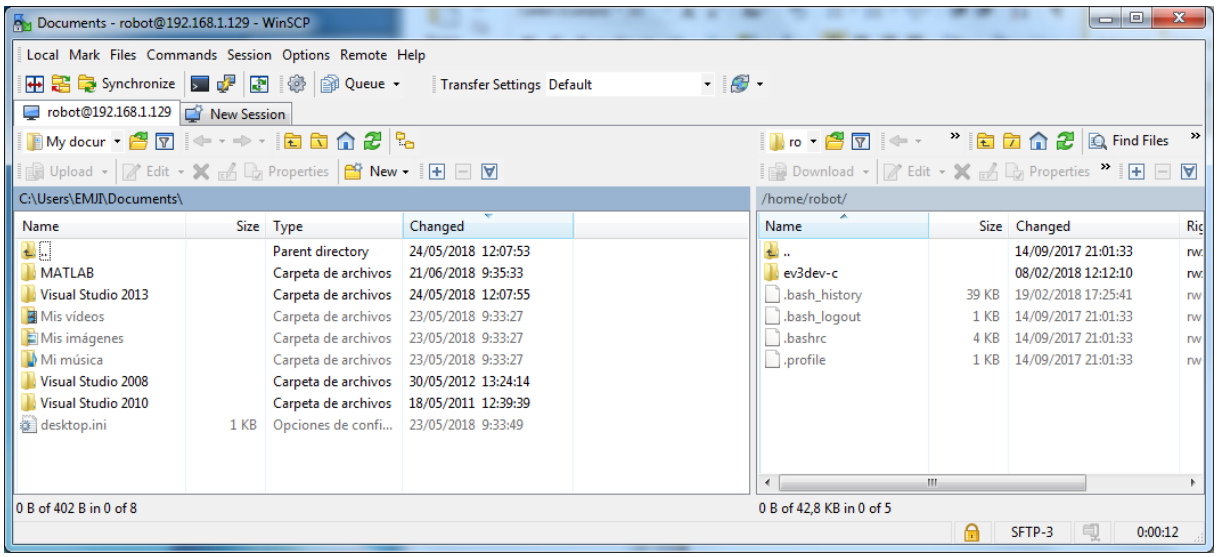

Fig. 8.4 Pantalla donde se realiza la transferencia de archivos

ANEXO III: CÓDIGO EXPERIENCIA 1

SERVIDOR

```
#include <pthread.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<math.h>
#include<time.h>

#include"ev3_light.h"
#include "ev3.h"
#include "ev3_port.h"
#include "ev3_tacho.h"
#include "brick.h"


// WIN32 ////////////////////////////////////////
#ifdef __WIN32__

#include <windows.h>

// UNIX ////////////////////////////////////////
#else

#include <unistd.h>
#define Sleep( msec ) usleep(( msec ) * 1000 )

////////////////////////////////////////////////
#endif

//VARIABLES

 struct sockaddr_in server;
 struct sockaddr_in cliente;
 struct hostent *hp;
 int puerto, conexion, resp, j, stsize, conexion_cliente ;
 char bufferposxe[100], bufferposye[100], bufferposxr[100], bufferposyr[100];
FILE *fich;
 float xrecibido,yrecibido;
 socklen_t longc;
float posxe[601];
float posxr[601];
float posye[601];
```

```
float posyr[601];


        long i = 0;
        float tfinal=12.0;
        long num_itera=0;
        float pos_d_nueva=0;
        float pos_d_anterior=0;
        float pos_i_nueva=0;
        float pos_i_anterior=0;
        float acontrol_d, acontrol_i;
        float acontrol_d1=0, acontrol_i1=0;
        float vel_ang_d, vel_ang_i;
        float vel_lin_d, vel_lin_i;
        float vref_d, vref_i;
        float wref_d,wref_i;
        float error_vel_d, error_vel_i;
        float error_vel_d1=0, error_vel_i1=0;
        float error_vel_d2=0, error_vel_i2=0;
        float vel_lin_robot, vel_ang_robot;
        float refx,refy;
        float x_deriv=0.0;
        float y_deriv=0.0;
        float velxref=0.0;
        float velyref=0.0;

        float kpx=4.0;
        float kpy=4.0;

        float kvx=0.50;
        float kvy=0.50;

        float ang=0;
        float x=0.500;
        float y=0.0;
        float b=0.061;
        float e=0.061;
        float pul2rad=0.0174533;
        float radio_rueda=0.028;
        float Ts=0.02;
        float refxante=0;
       float diferencia=0;
       clock_t t1;
       clock_t t2;
       int tf;
```

```
struct coordenviada
{
	float posxe;
	float posye;
};

struct coordrecibida
{
	float posxr;
	float posyr;
};




void *funcionThread1 (void *parametro);
void *funcionThread2 (void *parametro);

int main(void){

tacho_reset(OUTB);
tacho_reset(OUTD);


//CONEXIONES SHOCKETS
conexion = socket(AF_INET, SOCK_STREAM, 0);

if(conexion == -1){

  printf("Error al crear el socket\n");
}

puerto=6000;
bzero((char *)&server, sizeof((char *)&server));

 server.sin_addr.s_addr=inet_addr("192.168.1.129");
 server.sin_family = AF_INET;
 server.sin_port = htons(6000);

resp=bind(conexion,(struct sockaddr *)&server, sizeof(server));

listen(conexion,1);

printf("Esperando conexiones entrantes... \n");

longc = sizeof(cliente);

conexion_cliente=accept(conexion,(struct sockaddr *)&cliente, &longc);
```

```
 if(conexion_cliente<0)

  {
    printf("Error al aceptar trafico\n");
    close(conexion);
    return 1;

}

//MOTORES

#ifndef __ARM_ARCH_4T__
        /* Disable auto-detection of the brick (you have to set the correct address below) */
        ev3_brick_addr = "192.168.1.129";

#endif
        if ( ev3_init() == -1 ) return ( 1 );

#ifndef __ARM_ARCH_4T__
        printf( "The EV3 brick auto-detection is DISABLED,\nwaiting %s online with plugged
tacho...\n", ev3_brick_addr );

#else
        printf( "Waiting tacho is plugged...\n" );

#endif
        while ( ev3_tacho_init() < 1 ) Sleep( 1000 );

        printf( "*** ( EV3 ) Hello! ***\n" );

        printf( "Found tacho motors:\n" );
        for ( j = 0; j < DESC_LIMIT; j++ ) {
                if ( ev3_tacho[ j ].type_inx != TACHO_TYPE__NONE_ ) {
                        printf( "  type = %s\n", ev3_tacho_type( ev3_tacho[ j ].type_inx ));
                        printf( "  port = %s\n", ev3_tacho_port_name( j, s ));


                        }
        }

pthread_t idHilo1;
pthread_t idHilo2;
int error1;
int error2;
error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);

if( error2!=0 || error1!=0 )
        {
```

```
                perror("No puedo crear thread");
                exit(-1);
        }

        pthread_join(idHilo1,NULL);
        pthread_join(idHilo2,NULL);



 return (0);
}
void *funcionThread1(void *parametro){

struct coordenviada  Coordmensajee;

        i=0;

        num_itera=(int)((tfinal)/0.02);

        while(i < num_itera){


        t1=clock();



        if((diferencia>=0.1) && (xrecibido!=0)){
                        tacho_set_stop_action(OUTA,TACHO_HOLD);
                        tacho_set_stop_action(OUTD,TACHO_HOLD);
                        tacho_set_command(OUTA, TACHO_STOP);
                        tacho_set_command(OUTD, TACHO_STOP);
                }
else{
         refx=(0.5+(1.500-0.500)*i/(num_itera));
         refy=0;
         velxref=(refx-refxante)/0.02;
         velyref=0;

    pos_d_nueva=tacho_get_position(OUTD,0)*pul2rad;
    pos_i_nueva=tacho_get_position(OUTB,0)*pul2rad;

                    vel_ang_d=(pos_d_nueva-pos_d_anterior)/Ts;
                    vel_ang_i=(pos_i_nueva-pos_i_anterior)/Ts;

                    vel_lin_d=vel_ang_d*radio_rueda;
                    vel_lin_i=vel_ang_i*radio_rueda;

                    vel_lin_robot=(vel_lin_d+ vel_lin_i)/2;

                    vel_ang_robot=(vel_lin_d- vel_lin_i)/(2*b);
```

```
                x=x+vel_lin_robot*Ts*cos(ang);

                y=y+vel_lin_robot*Ts*sin(ang);

                ang=ang+vel_ang_robot*Ts;

            x_deriv=kvx*velxref+kpx*(refx-(x+(e*cos(ang))));
            y_deriv=kvy*velyref+kpy*(refy-(y+(e*sin(ang))));


 vref_i=(((e*cos(ang)+b*sin(ang))*x_deriv)+((e*sin(ang)-b*cos(ang))*y_deriv))/e;
vref_d=(((e*cos(ang)-b*sin(ang))*x_deriv)+((e*sin(ang)+b*cos(ang))*y_deriv))/e;

            wref_d=vref_d/radio_rueda;
            wref_i=vref_i/radio_rueda;

            error_vel_d=wref_d-vel_ang_d;
            error_vel_i=wref_i-vel_ang_i;

acontrol_d=(4*error_vel_d)-(2.8*error_vel_d1)+(0.9*error_vel_d2)+acontrol_d1;

acontrol_i=(4*error_vel_i)-(2.8*error_vel_i1)+(0.9*error_vel_i2)+acontrol_i1;


            if (acontrol_d>100)
                {
                    acontrol_d=100;
                }
            if (acontrol_d<-100)
                {
                    acontrol_d=-100;
                }
            if (acontrol_i>100)
                {
                    acontrol_i=100;
                }
            if (acontrol_i<-100)
                {
                    acontrol_i=-100;
                }


tacho_set_duty_cycle_sp(OUTB,acontrol_i);
tacho_set_duty_cycle_sp(OUTD,acontrol_d);
tacho_set_command(OUTB, TACHO_RUN_DIRECT );
tacho_set_command(OUTD, TACHO_RUN_DIRECT);


            pos_d_anterior=pos_d_nueva;
```

```
                    pos_i_anterior=pos_i_nueva;


      acontrol_d1=acontrol_d;
      acontrol_i1=acontrol_i;
      error_vel_d2=error_vel_d1;
      error_vel_i2=error_vel_i1;
      error_vel_d1=error_vel_d;
      error_vel_i1=error_vel_i;
       refxante=refx;


          Coordmensajee.posxe=x;
          Coordmensajee.posye=y;

        send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
        posxe[i]=Coordmensajee.posxe;
        posye[i]=Coordmensajee.posye;



i++;
        }
t2=clock();
tf=(int)(20-(((t2-t1)/CLOCKS_PER_SEC))*1000);
sleep_ms(tf);
diferencia=x-xrecibido;
}

                    fich=fopen("servidor.txt","w");
                    int h;
                    for(h=0;h<601;h++){
                    fprintf(fich,"%f %f %f %f\n",posxe[h],posye[h],posxr[h],posyr[h]);
                    }



                    tacho_set_stop_action(OUTB,TACHO_HOLD);
                    tacho_set_stop_action(OUTD,TACHO_HOLD);
                    tacho_set_command(OUTB, TACHO_STOP);
                    tacho_set_command(OUTD, TACHO_STOP);

                    tacho_reset(OUTB);
                    tacho_reset(OUTD);

        fclose(fich);
        ev3_uninit();
```

```
        printf( "*** ( EV3 ) Bye! ***\n" );

}

void *funcionThread2(void *parametro1){

struct coordrecibida  Coordmensajer;

                while(1){

        recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
        xrecibido=Coordmensajer.posxr;
        posxr[i]=xrecibido;

        yrecibido=Coordmensajer.posyr;
        posyr[i]=yrecibido;



                }

}
```

CLIENTE

```
#include<stdio.h>
#include <pthread.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<math.h>
#include"ev3_light.h"
#include "ev3.h"
#include "ev3_port.h"
#include "ev3_tacho.h"
#include "brick.h"
#include<time.h>


// WIN32 /////////////////////////////////////
#ifdef __WIN32__

#include <windows.h>

// UNIX //////////////////////////////////////
#else

#include <unistd.h>
#define Sleep( msec ) usleep(( msec ) * 1000 )

//////////////////////////////////////////////
#endif

//VARIABLES
 struct sockaddr_in server;
 struct sockaddr_in cliente;
 int puerto, conexion,resp,j;
 charbufferposxe[100],bufferposye[100], bufferposxr[100],bufferposyr[100];
FILE *fich;
 float refxante,xrecibido,yrecibido;
float posxe[601];
float posxr[601];
float posye[601];
float posyr[601];
        long i = 0;
        float tfinal=12.0;
        long num_itera=0;
        float pos_d_nueva=0;
        float pos_d_anterior=0;
        float pos_i_nueva=0;
```

```
        float pos_i_anterior=0;
        float acontrol_d,acontrol_i;
        float acontrol_d1=0,acontrol_i1=0;
        float vel_ang_d, vel_ang_i;
        float vel_lin_d, vel_lin_i;
        float vref_d, vref_i;
        float wref_d,wref_i;
        float error_vel_d, error_vel_i;
        float error_vel_d1=0, error_vel_i1=0;
        float error_vel_d2=0, error_vel_i2=0;
        float vel_lin_robot, vel_ang_robot;
        float refx,refy;
        float x_deriv=0.0;
        float y_deriv=0.0;
        float velxref=0.0;
        float velyref=0.0;

        float kpx=4.0;
        float kpy=4.0;

        float kvx=0.50;
        float kvy=0.50;

        float ang=0;
        float x=0.500;
        float y=0.0;
        float b=0.061;
        float e=0.061;
        float pul2rad=0.0174533;
        float radio_rueda=0.028;
        float Ts=0.02;
        float refxante=0;
       float diferencia=0;
       clock_t t1;
       clock_t t2;
       int tf;

struct coordenviada
{
        float posxe;
        float posye;
};

struct coordrecibida
{
        float posxr;
        float posyr;
};
```

```
void *funcionThread1 (void *parametro);
void *funcionThread2 (void *parametro);

int main(void){

tacho_reset(OUTB);
tacho_reset(OUTD);

//CONEXIONES SHOCKETS

conexion = socket(AF_INET, SOCK_STREAM, 0);

if(conexion == -1)
        {
  printf("Error al crear el socket\n");
        }
puerto=6000;
bzero((char *)&server, sizeof((char *)&server));

 server.sin_addr.s_addr=inet_addr("192.168.1.129");
 server.sin_family = AF_INET;
 server.sin_port = htons(6000);

 cliente.sin_addr.s_addr=inet_addr("192.168.1.130");
 cliente.sin_family = AF_INET;
 cliente.sin_port = htons(6000);

resp=bind(conexion,(struct sockaddr *)&cliente, sizeof(cliente));

 if(connect(conexion,(struct sockaddr *)&server, sizeof(server)) < 0)
 {
   printf("Error conectando con el host\n");
   close(conexion);
   return 1;
 }

//MOTORES

#ifndef __ARM_ARCH_4T__
        /* Disable auto-detection of the brick (you have to set the correct address below) */
        ev3_brick_addr = "192.168.1.130";

#endif
        if ( ev3_init() == -1 ) return ( 1 );

#ifndef __ARM_ARCH_4T__
        printf( "The EV3 brick auto-detection is DISABLED,\nwaiting %s online with plugged
tacho...\n", ev3_brick_addr );

#else
```

```
        printf( "Waiting tacho is plugged...\n" );

#endif
        while ( ev3_tacho_init() < 1 ) Sleep( 1000 );

        printf( "*** ( EV3 ) Hello! ***\n" );

        printf( "Found tacho motors:\n" );
        for ( j = 0; j < DESC_LIMIT; j++ ) {
                if ( ev3_tacho[ j ].type_inx != TACHO_TYPE__NONE_ ) {
                        printf( "  type = %s\n", ev3_tacho_type( ev3_tacho[ j ].type_inx ));
                        printf( "  port = %s\n", ev3_tacho_port_name( j, s ));


                        }
        }



pthread_t idHilo1;
pthread_t idHilo2;
int error1;
int error2;
error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);

if( error2!=0 || error1!=0 )
        {
                perror("No puedo crear thread");
                exit(-1);
        }

        pthread_join(idHilo1,NULL);
        pthread_join(idHilo2,NULL);

  return (0);
}

void *funcionThread1(void *parametro){

struct coordenviada  Coordmensajee;

        i=0;

        num_itera=(int)((tfinal)/0.02);

        while(i < num_itera)     {

        t1=clock();
```

```
                if((diferencia>=0.1) && (xrecibido!=0)){

                        tacho_set_stop_action(OUTA,TACHO_HOLD);
                        tacho_set_stop_action(OUTD,TACHO_HOLD);
                        tacho_set_command(OUTA, TACHO_STOP);
                        tacho_set_command(OUTD, TACHO_STOP)
                }
        else{

                 refx=(0.5+(1.500-0.500)*i/(num_itera));
                 refy=0;
                 velxref=(refx-refxante)/0.02;
                  velyref=0;

    pos_d_nueva=tacho_get_position(OUTD,0)*pul2rad;
    pos_i_nueva=tacho_get_position(OUTB,0)*pul2rad;

                   vel_ang_d=(pos_d_nueva-pos_d_anterior)/Ts;
                   vel_ang_i=(pos_i_nueva-pos_i_anterior)/Ts;



                   vel_lin_d=vel_ang_d*radio_rueda;
                   vel_lin_i=vel_ang_i*radio_rueda;

                   vel_lin_robot=(vel_lin_d+ vel_lin_i)/2;

                   vel_ang_robot=(vel_lin_d- vel_lin_i)/(2*b);

                        x=x+vel_lin_robot*Ts*cos(ang);

                        y=y+vel_lin_robot*Ts*sin(ang);

                        ang=ang+vel_ang_robot*Ts;



                   x_deriv=kvx*velxref+kpx*(refx-(x+(e*cos(ang))));
                   y_deriv=kvy*velyref+kpy*(refy-(y+(e*sin(ang))));

vref_i=(((e*cos(ang)+b*sin(ang))*x_deriv)+((e*sin(ang)-b*cos(ang))*y_deriv))/e;
vref_d=(((e*cos(ang)-b*sin(ang))*x_deriv)+((e*sin(ang)+b*cos(ang))*y_deriv))/e;

                   wref_d=vref_d/radio_rueda;
                   wref_i=vref_i/radio_rueda;

                   error_vel_d=wref_d-vel_ang_d;
                   error_vel_i=wref_i-vel_ang_i;
```

```
acontrol_d=(4*error_vel_d)-(2.8*error_vel_d1)+(0.9*error_vel_d2)+acontrol_d1;

acontrol_i=(4*error_vel_i)-(2.8*error_vel_i1)+(0.9*error_vel_i2)+acontrol_i1;


                    if (acontrol_d>100)
                        {
                            acontrol_d=100;
                        }
                    if (acontrol_d<-100)
                        {
                            acontrol_d=-100;
                        }
                    if (acontrol_i>100)
                        {
                            acontrol_i=100;
                        }
                    if (acontrol_i<-100)
                        {
                            acontrol_i=-100;
                        }


tacho_set_duty_cycle_sp(OUTB,acontrol_i);
tacho_set_duty_cycle_sp(OUTD,acontrol_d);
tacho_set_command(OUTB, TACHO_RUN_DIRECT );
tacho_set_command(OUTD, TACHO_RUN_DIRECT);

                    pos_d_anterior=pos_d_nueva;
                    pos_i_anterior=pos_i_nueva;

    acontrol_d1=acontrol_d;
    acontrol_i1=acontrol_i;
    error_vel_d2=error_vel_d1;
    error_vel_i2=error_vel_i1;
    error_vel_d1=error_vel_d;
    error_vel_i1=error_vel_i;
    refxante=refx;

        Coordmensajee.posxe=x;
        Coordmensajee.posye=y;

      send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
      posxe[i]=Coordmensajee.posxe;
      posye[i]=Coordmensajee.posye;

i++;
        }
t2=clock();
```

```
tf=(int)(20-(((t2-t1)/CLOCKS_PER_SEC))*1000);

sleep_ms(tf);

diferencia=x-xrecibido;

}

                fich=fopen("cliente.txt","w");
                int h;
                for(h=0;h<601;h++){
                fprintf(fich,"%f %f %f %f\n",posxr[h],posyr[h],posxe[h],posye[h]);
                }

                tacho_set_stop_action(OUTB,TACHO_HOLD);
                tacho_set_stop_action(OUTD,TACHO_HOLD);
                tacho_set_command(OUTB, TACHO_STOP);
                tacho_set_command(OUTD, TACHO_STOP);

                tacho_reset(OUTB);
                tacho_reset(OUTD);

        fclose(fich);
        ev3_uninit();
        printf( "*** ( EV3 ) Bye! ***\n" );

}

void *funcionThread2(void *parametro){

struct coordrecibida  Coordmensajer;

                while(1){

        recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
        xrecibido=Coordmensajer.posxr;
        posxr[i]=xrecibido;

        yrecibido=Coordmensajer.posyr;
        posyr[i]=yrecibido;

                }
}
```

ANEXO IV: CÓDIGO EXPERIENCIA 2

SERVIDOR

```
#include <pthread.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<math.h>
#include<time.h>

#include"ev3_light.h"
#include "ev3.h"
#include "ev3_port.h"
#include "ev3_tacho.h"
#include "brick.h"


// WIN32 /////////////////////////////////////////
#ifdef __WIN32__

#include <windows.h>

// UNIX //////////////////////////////////////////
#else

#include <unistd.h>
#define Sleep( msec ) usleep(( msec ) * 1000 )

//////////////////////////////////////////////////
#endif

//VARIABLES

 struct sockaddr_in server;
 struct sockaddr_in cliente;
 struct hostent *hp;
 int puerto, conexion, resp, j, stsize, conexion_cliente,
 char bufferposxe[100],bufferposye[100],bufferposxr[100],bufferposyr[100];
 FILE *fich;
 float xrecibido,yrecibido;
 socklen_t longc;
 float posxe[601];
 float posxr[601];
 float posye[601];
```

```
float posyr[601];


        long i = 0;
        float tfinal=12.0;
        long num_itera=0;
        float pos_d_nueva=0;
        float pos_d_anterior=0;
        float pos_i_nueva=0;
        float pos_i_anterior=0;
        float acontrol_d,acontrol_i;
        float acontrol_d1=0,acontrol_i1=0;
        float vel_ang_d, vel_ang_i;
        float vel_lin_d, vel_lin_i;
        float vref_d, vref_i;
        float wref_d,wref_i;
        float error_vel_d, error_vel_i;
        float error_vel_d1=0, error_vel_i1=0;
        float error_vel_d2=0, error_vel_i2=0;
        float vel_lin_robot, vel_ang_robot;
        float refx,refy;
        float x_deriv=0.0;
        float y_deriv=0.0;
        float velxref=0.0;
        float velyref=0.0;

        float kpx=4.0;
        float kpy=4.0;

        float kvx=0.50;
        float kvy=0.50;

        float ang=0;
        float x=0.500;
        float y=0.0;
        float b=0.061;
        float e=0.061;
        float pul2rad=0.0174533;
        float radio_rueda=0.028;
        float Ts=0.02;
        float refxante=0;
       float diferencia=0;
       clock_t t1;
       clock_t t2;
       int tf;

struct coordenviada
{
        float posxe;
        float posye;
```

```
};

struct coordrecibida
{
        float posxr;
        float posyr;
};


void *funcionThread1 (void *parametro);
void *funcionThread2 (void *parametro);

int main(void){

tacho_reset(OUTB);
tacho_reset(OUTD);


//CONEXIONES SHOCKETS

conexion = socket(AF_INET, SOCK_STREAM, 0);

if(conexion == -1) {

   printf("Error al crear el socket\n");
}

puerto=6000;
bzero((char *)&server, sizeof((char *)&server));

  server.sin_addr.s_addr=inet_addr("192.168.1.129");
  server.sin_family = AF_INET;
  server.sin_port = htons(6000);

resp=bind(conexion,(struct sockaddr *)&server, sizeof(server));

listen(conexion,1);

printf("Esperando conexiones entrantes... \n");

longc = sizeof(cliente);

conexion_cliente=accept(conexion,(struct sockaddr *)&cliente, &longc);

 if(conexion_cliente<0)

  {
    printf("Error al aceptar trafico\n");
    close(conexion);
    return 1;
```

```
}

//MOTORES

#ifndef __ARM_ARCH_4T__
	/* Disable auto-detection of the brick (you have to set the correct address below) */
	ev3_brick_addr = "192.168.1.129";

#endif
	if ( ev3_init() == -1 ) return ( 1 );

#ifndef __ARM_ARCH_4T__
	printf( "The EV3 brick auto-detection is DISABLED,\nwaiting %s online with plugged
tacho...\n", ev3_brick_addr );

#else
	printf( "Waiting tacho is plugged...\n" );

#endif
	while ( ev3_tacho_init() < 1 ) Sleep( 1000 );

	printf( "*** ( EV3 ) Hello! ***\n" );

	printf( "Found tacho motors:\n" );
	for ( j = 0; j < DESC_LIMIT; j++ ) {
		if ( ev3_tacho[ j ].type_inx != TACHO_TYPE__NONE_ ) {
			printf( "  type = %s\n", ev3_tacho_type( ev3_tacho[ j ].type_inx ));
			printf( "  port = %s\n", ev3_tacho_port_name( j, s ));


			}
	}

pthread_t idHilo1;
pthread_t idHilo2;
int error1;
int error2;
error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);

if( error2!=0 || error1!=0 )
	{
		perror("No puedo crear thread");
		exit(-1);
	}

	pthread_join(idHilo1,NULL);
	pthread_join(idHilo2,NULL);
```

```
  return (0);
}

void *funcionThread1(void *parametro){

struct coordenviada  Coordmensajee;

          i=0;

          num_itera=(int)((tfinal)/0.02);

          while(i < num_itera){

          t1=clock();

           refx=(0.5+(1.500-0.500)*i/(num_itera));
           refy=0;
           velxref=(refx-refxante)/0.02;
           velyref=0;

      pos_d_nueva=tacho_get_position(OUTD,0)*pul2rad;
      pos_i_nueva=tacho_get_position(OUTB,0)*pul2rad;

                        vel_ang_d=(pos_d_nueva-pos_d_anterior)/Ts;
                        vel_ang_i=(pos_i_nueva-pos_i_anterior)/Ts;

                        vel_lin_d=vel_ang_d*radio_rueda;
                        vel_lin_i=vel_ang_i*radio_rueda;

                        vel_lin_robot=(vel_lin_d+ vel_lin_i)/2;

                        vel_ang_robot=(vel_lin_d- vel_lin_i)/(2*b);

                              x=x+vel_lin_robot*Ts*cos(ang);

                              y=y+vel_lin_robot*Ts*sin(ang);

                              ang=ang+vel_ang_robot*Ts;

                        x_deriv=kvx*velxref+kpx*(refx-(x+(e*cos(ang))));
                        y_deriv=kvy*velyref+kpy*(refy-(y+(e*sin(ang))));

 vref_i=(((e*cos(ang)+b*sin(ang))*x_deriv)+((e*sin(ang)-b*cos(ang))*y_deriv))/e;
 vref_d=(((e*cos(ang)-b*sin(ang))*x_deriv)+((e*sin(ang)+b*cos(ang))*y_deriv))/e;

                        wref_d=vref_d/radio_rueda;
                        wref_i=vref_i/radio_rueda;


                        error_vel_d=wref_d-vel_ang_d;
```

```
                    error_vel_i=wref_i-vel_ang_i;

acontrol_d=(4*error_vel_d)-(2.8*error_vel_d1)+(0.9*error_vel_d2)+acontrol_d1;

acontrol_i=(4*error_vel_i)-(2.8*error_vel_i1)+(0.9*error_vel_i2)+acontrol_i1;

                    if (acontrol_d>100)
                        {
                            acontrol_d=100;
                        }
                    if (acontrol_d<-100)
                        {
                            acontrol_d=-100;
                        }
                    if (acontrol_i>100)
                        {
                            acontrol_i=100;
                        }
                    if (acontrol_i<-100)
                        {
                            acontrol_i=-100;
                        }


tacho_set_duty_cycle_sp(OUTB,acontrol_i);
tacho_set_duty_cycle_sp(OUTD,acontrol_d);
tacho_set_command(OUTB, TACHO_RUN_DIRECT );
tacho_set_command(OUTD, TACHO_RUN_DIRECT);

                    pos_d_anterior=pos_d_nueva;
                    pos_i_anterior=pos_i_nueva;

    acontrol_d1=acontrol_d;
    acontrol_i1=acontrol_i;
    error_vel_d2=error_vel_d1;
    error_vel_i2=error_vel_i1;
    error_vel_d1=error_vel_d;
    error_vel_i1=error_vel_i;
    refxante=refx;

        Coordmensajee.posxe=x;
        Coordmensajee.posye=y;

       send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
       posxe[i]=Coordmensajee.posxe;
       posye[i]=Coordmensajee.posye;

i++;
```

```
t2=clock();
tf=(int)(20-(((t2-t1)/CLOCKS_PER_SEC))*1000);
sleep_ms(tf);

}
                fich=fopen("servidor.txt","w");
                int h;
                for(h=0;h<601;h++) {
                fprintf(fich,"%f %f %f %f\n",posxe[h],posye[h],posxr[h],posyr[h]);
                }
                tacho_set_stop_action(OUTB,TACHO_HOLD);
                tacho_set_stop_action(OUTD,TACHO_HOLD);
                tacho_set_command(OUTB, TACHO_STOP);
                tacho_set_command(OUTD, TACHO_STOP);


                tacho_reset(OUTB);
                tacho_reset(OUTD);

        fclose(fich);
        ev3_uninit();
        printf( "*** ( EV3 ) Bye! ***\n" );

}

void *funcionThread2(void *parametro1){

struct coordrecibida  Coordmensajer;

                while(1){
        recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
        xrecibido=Coordmensajer.posxr;
        posxr[i]=xrecibido;

        yrecibido=Coordmensajer.posyr;
        posyr[i]=yrecibido;

                }

}
```

CLIENTE

```
#include<stdio.h>
#include <pthread.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<math.h>
#include"ev3_light.h"
#include "ev3.h"
#include "ev3_port.h"
#include "ev3_tacho.h"
#include "brick.h"
#include<time.h>


// WIN32 /////////////////////////////////////
#ifdef __WIN32__

#include <windows.h>

// UNIX //////////////////////////////////////
#else

#include <unistd.h>
#define Sleep( msec ) usleep(( msec ) * 1000 )

//////////////////////////////////////////////
#endif
//VARIABLES
 struct sockaddr_in server;
 struct sockaddr_in cliente;
int puerto, conexion,resp,j;
 charbufferposxe[100],bufferposye[100], bufferposxr[100],bufferposyr[100];
 FILE *fich;
 float xrecibido,yrecibido;
float posxe[601];
float posxr[601];
float posye[601];
float posyr[601];

        long i = 0;
        float tfinal=12.0;
        long num_itera=0;
        float pos_d_nueva=0;
```

```
        float pos_d_anterior=0;
        float pos_i_nueva=0;
        float pos_i_anterior=0;
        float acontrol_d,acontrol_i;
        float acontrol_d1=0,acontrol_i1=0;
        float vel_ang_d, vel_ang_i;
        float vel_lin_d, vel_lin_i;
        float vref_d, vref_i;
        float wref_d,wref_i;
        float error_vel_d, error_vel_i;
        float error_vel_d1=0, error_vel_i1=0;
        float error_vel_d2=0, error_vel_i2=0;
        float vel_lin_robot, vel_ang_robot;
        float refx,refy;
        float x_deriv=0.0;
        float y_deriv=0.0;
        float velxref=0.0;
        float velyref=0.0;

        float kpx=4.0;
        float kpy=4.0;

        float kvx=0.50;
        float kvy=0.50;

        float ang=0;
        float x=-0.500;
        float y=0.0;
        float b=0.061;
        float e=-0.061;
        float pul2rad=0.0174533;
        float radio_rueda=0.028;
        float Ts=0.02;
        float refxante=0;
       clock_t t1;
       clock_t t2;
       int tf;
       float diferencia=0;

struct coordenviada
{
        float posxe;
        float posye;
};

struct coordrecibida
{
        float posxr;
        float posyr;
};
```

```
void *funcionThread1 (void *parametro);
void *funcionThread2 (void *parametro);


int main(void){

tacho_reset(OUTA);
tacho_reset(OUTD);

//CONEXIONES SHOCKETS

conexion = socket(AF_INET, SOCK_STREAM, 0);

if(conexion == -1)
        {
  printf("Error al crear el socket\n");
        }

puerto=6000;
bzero((char *)&server, sizeof((char *)&server));

 server.sin_addr.s_addr=inet_addr("192.168.1.129");
 server.sin_family = AF_INET;
 server.sin_port = htons(6000);

 cliente.sin_addr.s_addr=inet_addr("192.168.1.130");
 cliente.sin_family = AF_INET;
 cliente.sin_port = htons(6000);

resp=bind(conexion,(struct sockaddr *)&cliente, sizeof(cliente));

 if(connect(conexion,(struct sockaddr *)&server, sizeof(server)) < 0)
 {
   printf("Error conectando con el host\n");
   close(conexion);
   return 1;
 }

//MOTORES

#ifndef __ARM_ARCH_4T__
        /* Disable auto-detection of the brick (you have to set the correct address below) */
        ev3_brick_addr = "192.168.1.130";

#endif
        if ( ev3_init() == -1 ) return ( 1 );

#ifndef __ARM_ARCH_4T__
```

```
        printf( "The EV3 brick auto-detection is DISABLED,\nwaiting %s online with plugged
tacho...\n", ev3_brick_addr );

#else
        printf( "Waiting tacho is plugged...\n" );

#endif
        while ( ev3_tacho_init() < 1 ) Sleep( 1000 );

        printf( "*** ( EV3 ) Hello! ***\n" );

        printf( "Found tacho motors:\n" );
        for ( j = 0; j < DESC_LIMIT; j++ ) {
                if ( ev3_tacho[ j ].type_inx != TACHO_TYPE__NONE_ ) {
                        printf( "  type = %s\n", ev3_tacho_type( ev3_tacho[ j ].type_inx ));
                        printf( "  port = %s\n", ev3_tacho_port_name( j, s ));


                        }
        }
pthread_t idHilo1;
pthread_t idHilo2;
int error1;
int error2;

error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);

if( error2!=0 || error1!=0 )
        {
                perror("No puedo crear thread");
                exit(-1);
        }

        pthread_join(idHilo1,NULL);
        pthread_join(idHilo2,NULL);

  return (0);
}

void *funcionThread1(void *parametro){

struct coordenviada  Coordmensajee;

        i=0;

        num_itera=(int)((tfinal)/0.02);


        while(i < num_itera)     {
```

```
t1=clock();
 if((diferencia>=0.1 ) && (xrecibido!=0)){

        tacho_set_stop_action(OUTA,TACHO_HOLD);
        tacho_set_stop_action(OUTD,TACHO_HOLD);
        tacho_set_command(OUTA, TACHO_STOP);
        tacho_set_command(OUTD, TACHO_STOP);

        }


else {
 refx=-(0.5+(1.500-0.500)*i/(num_itera));
 refy=0;
 velxref=(refx-refxante)/0.02;
 velyref=0;



pos_d_nueva=tacho_get_position(OUTD,0)*pul2rad;
pos_i_nueva=tacho_get_position(OUTA,0)*pul2rad;


        vel_ang_d=(pos_d_nueva-pos_d_anterior)/Ts;
        vel_ang_i=(pos_i_nueva-pos_i_anterior)/Ts;

        vel_lin_d=vel_ang_d*radio_rueda;
        vel_lin_i=vel_ang_i*radio_rueda;

        vel_lin_robot=(vel_lin_d+ vel_lin_i)/2;

        vel_ang_robot=(vel_lin_d- vel_lin_i)/(2*b);


            x=x+vel_lin_robot*Ts*cos(ang);

            y=y+vel_lin_robot*Ts*sin(ang);

            ang=ang+vel_ang_robot*Ts;

        x_deriv=kvx*velxref+kpx*(refx-(x+(e*cos(ang))));
        y_deriv=kvy*velyref+kpy*(refy-(y+(e*sin(ang))));
vref_i=(((e*cos(ang)+b*sin(ang))*x_deriv)+((e*sin(ang)-b*cos(ang))*y_deriv))/e;
vref_d=(((e*cos(ang)-b*sin(ang))*x_deriv)+((e*sin(ang)+b*cos(ang))*y_deriv))/e;

        wref_d=vref_d/radio_rueda;
        wref_i=vref_i/radio_rueda;

        error_vel_d=wref_d-vel_ang_d;
```

```
                    error_vel_i=wref_i-vel_ang_i;

acontrol_d=(4*error_vel_d)-(2.8*error_vel_d1)+(0.9*error_vel_d2)+acontrol_d1;

acontrol_i=(4*error_vel_i)-(2.8*error_vel_i1)+(0.9*error_vel_i2)+acontrol_i1;

                    if (acontrol_d>100)
                        {
                            acontrol_d=100;
                        }
                    if (acontrol_d<-100)
                        {
                            acontrol_d=-100;
                        }
                    if (acontrol_i>100)
                        {
                            acontrol_i=100;
                        }
                    if (acontrol_i<-100)
                        {
                            acontrol_i=-100;
                        }


tacho_set_duty_cycle_sp(OUTA,acontrol_i);
tacho_set_duty_cycle_sp(OUTD,acontrol_d);
tacho_set_command(OUTA, TACHO_RUN_DIRECT );
tacho_set_command(OUTD, TACHO_RUN_DIRECT);

                    pos_d_anterior=pos_d_nueva;
                    pos_i_anterior=pos_i_nueva;

      acontrol_d1=acontrol_d;
      acontrol_i1=acontrol_i;
     error_vel_d2=error_vel_d1;
      error_vel_i2=error_vel_i1;
      error_vel_d1=error_vel_d;
      error_vel_i1=error_vel_i;
      refxante=refx;


           Coordmensajee.posxe=x;
           Coordmensajee.posye=y;

         send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
         posxe[i]=Coordmensajee.posxe;
         posye[i]=Coordmensajee.posye;
```

```
t2=clock();
tf=(int)(20-(((t2-t1)/CLOCKS_PER_SEC))*1000);
}
sleep_ms(tf);
diferencia=-x-xrecibido
i++;
        }
                fich=fopen("cliente.txt","w");
                int h;
                for(h=0;h<601;h++){
                fprintf(fich,"%f %f %f %f\n",posxr[h],posyr[h],posxe[h],posye[h]);
                }
                tacho_set_stop_action(OUTA,TACHO_HOLD);
                tacho_set_stop_action(OUTD,TACHO_HOLD);
                tacho_set_command(OUTA, TACHO_STOP);
                tacho_set_command(OUTD, TACHO_STOP);
                tacho_reset(OUTA);
                tacho_reset(OUTD);
        fclose(fich);
        ev3_uninit();
        printf( "*** ( EV3 ) Bye! ***\n" );
}
void *funcionThread2(void *parametro){
struct coordrecibida  Coordmensajer;
                while(1){
        recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
        xrecibido=Coordmensajer.posxr;
        posxr[i]=xrecibido;
        yrecibido=Coordmensajer.posyr;
        posyr[i]=yrecibido;
                }
}
```

ANEXO V: CÓDIGO EXPERIENCIA 3

SERVIDOR

```
#include <pthread.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<math.h>
#include<time.h>

#include"ev3_light.h"
#include "ev3.h"
#include "ev3_port.h"
#include "ev3_tacho.h"
#include "brick.h"


// WIN32 /////////////////////////////////////
#ifdef __WIN32__

#include <windows.h>

// UNIX //////////////////////////////////////
#else

#include <unistd.h>
#define Sleep( msec ) usleep(( msec ) * 1000 )

//////////////////////////////////////////////
#endif

//VARIABLES

 struct sockaddr_in server;
 struct sockaddr_in cliente;
 int puerto, conexion, resp, j, stsize, conexion_cliente,k=0;
 char bufferposye[100], bufferposxr[100],bufferposyr[100],bufferposxe[100];
 FILE *fich;
 float xrecibido,yrecibido;
 socklen_t longc;
 float posxe[601];
 float posxr[601];
 float posye[601];
```

```
float posyr[601];


        long i = 0;
        float tfinal=12.0;
        long num_itera=0;
        float pos_d_nueva=0;
        float pos_d_anterior=0;
        float pos_i_nueva=0;
        float pos_i_anterior=0;
        float acontrol_d,acontrol_i;
        float acontrol_d1=0,acontrol_i1=0;
        float vel_ang_d, vel_ang_i;
        float vel_lin_d, vel_lin_i;
        float vref_d, vref_i;
        float wref_d,wref_i;
        float error_vel_d, error_vel_i;
        float error_vel_d1=0, error_vel_i1=0;
        float error_vel_d2=0, error_vel_i2=0;
        float vel_lin_robot, vel_ang_robot;
        float refx,refy;
        float x_deriv=0.0;
        float y_deriv=0.0;
        float velxref=0.0;
        float velyref=0.0;

        float kpx=4.0;
        float kpy=4.0;

        float kvx=0.50;
        float kvy=0.50;

        float ang=0;
        float x=0.500;
        float y=0.0;
        float b=0.061;
        float e=0.061;
        float pul2rad=0.0174533;
        float radio_rueda=0.028;
        float Ts=0.02;
        float refxante=0;
        float tiempo[650];
       clock_t t1;
       clock_t t2;
       int tf;

struct coordenviada
{
        float posxe;
        float posye;
```

```
};

struct coordrecibida
{
        float posxr;
        float posyr;
};

void *funcionThread1 (void *parametro);
void *funcionThread2 (void *parametro);


int main(void){

tacho_reset(OUTB);
tacho_reset(OUTD);


//CONEXIONES SHOCKETS

conexion = socket(AF_INET, SOCK_STREAM, 0);

if(conexion == -1) {

   printf("Error al crear el socket\n");

}

puerto=6000;
bzero((char *)&server, sizeof((char *)&server));

  server.sin_addr.s_addr=inet_addr("192.168.1.129");
  server.sin_family = AF_INET;
  server.sin_port = htons(6000);

resp=bind(conexion,(struct sockaddr *)&server, sizeof(server));

listen(conexion,1);

printf("Esperando conexiones entrantes... \n");

longc = sizeof(cliente);

conexion_cliente=accept(conexion,(struct sockaddr *)&cliente, &longc);

 if(conexion_cliente<0)
  {
    printf("Error al aceptar trafico\n");
    close(conexion);
    return 1;
```

```
}

//MOTORES

#ifndef __ARM_ARCH_4T__
	/* Disable auto-detection of the brick (you have to set the correct address below) */
	ev3_brick_addr = "192.168.1.129";

#endif
	if ( ev3_init() == -1 ) return ( 1 );

#ifndef __ARM_ARCH_4T__
	printf( "The EV3 brick auto-detection is DISABLED,\nwaiting %s online with plugged
tacho...\n", ev3_brick_addr );

#else
	printf( "Waiting tacho is plugged...\n" );

#endif
	while ( ev3_tacho_init() < 1 ) Sleep( 1000 );

	printf( "*** ( EV3 ) Hello! ***\n" );

	printf( "Found tacho motors:\n" );
	for ( j = 0; j < DESC_LIMIT; j++ ) {
		if ( ev3_tacho[ j ].type_inx != TACHO_TYPE__NONE_ ) {
			printf( "  type = %s\n", ev3_tacho_type( ev3_tacho[ j ].type_inx ));
			printf( "  port = %s\n", ev3_tacho_port_name( j, s ));


			}
	}

pthread_t idHilo1;
pthread_t idHilo2;
int error1;
int error2;
error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);

if(  error1!=0 )
	{
		perror("No puedo crear thread");
		exit(-1);
	}

	pthread_join(idHilo1,NULL);
	pthread_join(idHilo2,NULL);
```

```
 return (0);
}


void *funcionThread1(void *parametro){

struct coordenviada  Coordmensajee;
        i=0;

        num_itera=(int)((tfinal)/0.02);

        while(i < num_itera){
         t1=clock();
         refx=(0.5+(1.500-0.500)*i/(num_itera));
         refy=0;

         velxref=(refx-refxante)/0.02;
         velyref=0;

    pos_d_nueva=tacho_get_position(OUTD,0)*pul2rad;
    pos_i_nueva=tacho_get_position(OUTA,0)*pul2rad;

                    vel_ang_d=(pos_d_nueva-pos_d_anterior)/Ts;
                    vel_ang_i=(pos_i_nueva-pos_i_anterior)/Ts;

                    vel_lin_d=vel_ang_d*radio_rueda;
                    vel_lin_i=vel_ang_i*radio_rueda;

                    vel_lin_robot=(vel_lin_d+ vel_lin_i)/2;

                    vel_ang_robot=(vel_lin_d- vel_lin_i)/(2*b);

                        x=x+vel_lin_robot*Ts*cos(ang);

                        y=y+vel_lin_robot*Ts*sin(ang);

                        ang=ang+vel_ang_robot*Ts;

                    x_deriv=kvx*velxref+kpx*(refx-(x+(e*cos(ang))));
                    y_deriv=kvy*velyref+kpy*(refy-(y+(e*sin(ang))));


  vref_i=(((e*cos(ang)+b*sin(ang))*x_deriv)+((e*sin(ang)-b*cos(ang))*y_deriv))/e;
 vref_d=(((e*cos(ang)-b*sin(ang))*x_deriv)+((e*sin(ang)+b*cos(ang))*y_deriv))/e;

                    wref_d=vref_d/radio_rueda;
                    wref_i=vref_i/radio_rueda;
```

```
                error_vel_d=wref_d-vel_ang_d;
                error_vel_i=wref_i-vel_ang_i;

acontrol_d=(4*error_vel_d)-(2.8*error_vel_d1)+(0.9*error_vel_d2)+acontrol_d1;

acontrol_i=(4*error_vel_i)-(2.8*error_vel_i1)+(0.9*error_vel_i2)+acontrol_i1;


                if (acontrol_d>100)
                        {
                                acontrol_d=100;
                        }
                if (acontrol_d<-100)
                        {
                                acontrol_d=-100;
                        }
                if (acontrol_i>100)
                        {
                                acontrol_i=100;
                        }
                if (acontrol_i<-100)
                        {
                                acontrol_i=-100;
                        }


tacho_set_duty_cycle_sp(OUTA,acontrol_i);
tacho_set_duty_cycle_sp(OUTD,acontrol_d);
tacho_set_command(OUTA, TACHO_RUN_DIRECT );
tacho_set_command(OUTD, TACHO_RUN_DIRECT);


                pos_d_anterior=pos_d_nueva;
                pos_i_anterior=pos_i_nueva;


        acontrol_d1=acontrol_d;
        acontrol_i1=acontrol_i;
        error_vel_d2=error_vel_d1;
        error_vel_i2=error_vel_i1;
        error_vel_d1=error_vel_d;
        error_vel_i1=error_vel_i;
        refxante=refx;


                Coordmensajee.posxe=x;
                Coordmensajee.posye=y;

              send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
              posxe[i]=Coordmensajee.posxe;
```

```
        posye[i]=Coordmensajee.posye;

t2=clock();
tf=(int)(20-(((t2-t1)/CLOCKS_PER_SEC))*1000);
sleep_ms(tf);
i++;
        }
                fich=fopen("servidor.txt","w");
                int h;
                for(h=0;h<601;h++) {
                fprintf(fich,"%f %f %f %f\n",posxe[h],posye[h],posxr[h],posyr[h]);
                }
                tacho_set_stop_action(OUTA,TACHO_HOLD);
                tacho_set_stop_action(OUTD,TACHO_HOLD);
                tacho_set_command(OUTA, TACHO_STOP);
                tacho_set_command(OUTD, TACHO_STOP);

                tacho_reset(OUTB);
                tacho_reset(OUTD);

        fclose(fich);
        ev3_uninit();
        printf( "*** ( EV3 ) Bye! ***\n" );

}

void *funcionThread2(void *parametro1){

struct coordrecibida  Coordmensajer;

                while(1){

        recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
        xrecibido=Coordmensajer.posxr;
        posxr[i]=xrecibido;

        yrecibido=Coordmensajer.posyr;
        posyr[i]=yrecibido;

                }

}
```

CLIENTE

```
#include<stdio.h>
#include <pthread.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<math.h>
#include"ev3_light.h"
#include "ev3.h"
#include "ev3_port.h"
#include "ev3_tacho.h"
#include "brick.h"
#include<time.h>


// WIN32 /////////////////////////////////////
#ifdef __WIN32__

#include <windows.h>

// UNIX //////////////////////////////////////
#else

#include <unistd.h>
#define Sleep( msec ) usleep(( msec ) * 1000 )

//////////////////////////////////////////////
#endif
//VARIABLES
 struct sockaddr_in server;
 struct sockaddr_in cliente;
int puerto, conexion,resp,j;
charbufferposxe[100],bufferposye[100], bufferposxr[100],bufferposyr[100]];
 FILE *fich;
 Float yrecibido;
float posxe[601];
float posxr[601];
float posye[601];
float posyr[601];
float xrecibido=0;
float diferencia=0.0;
float xreal=0.0;
float yreal;
float irecibido=0;

        long i = 0;
```

```
        float tfinal=12.0;
        long num_itera=0;
        float pos_d_nueva=0;
        float pos_d_anterior=0;
        float pos_i_nueva=0;
        float pos_i_anterior=0;
        float acontrol_d,acontrol_i;
        float acontrol_d1=0,acontrol_i1=0;
        float vel_ang_d, vel_ang_i;
        float vel_lin_d, vel_lin_i;
        float vref_d, vref_i;
        float wref_d,wref_i;
        float error_vel_d, error_vel_i;
        float error_vel_d1=0, error_vel_i1=0;
        float error_vel_d2=0, error_vel_i2=0;
        float vel_lin_robot, vel_ang_robot;
        float refx,refy;
        float x_deriv=0.0;
        float y_deriv=0.0;
        float velxref=0.0;
        float velyref=0.0;



        float kpx=4.0;
        float kpy=4.0;

        float kvx=0.50;
        float kvy=0.50;

        float ang=0;
        float x=0.500;
        float y=0.0;
        float b=0.061;
        float e=0.061;
        float pul2rad=0.0174533;
        float radio_rueda=0.028;
        float Ts=0.02;
        float refxante=0;
       clock_t t1;
       clock_t t2;
       int tf;

struct coordenviada
{
        float posxe;
        float posye;
};

struct coordrecibida
```

```
{
        float posxr;
        float posyr;
};

void *funcionThread1 (void *parametro);
void *funcionThread2 (void *parametro);

int main(void){


tacho_reset(OUTB);
tacho_reset(OUTD);

//CONEXIONES SHOCKETS
conexion = socket(AF_INET, SOCK_STREAM, 0);

if(conexion == -1)
        {
  printf("Error al crear el socket\n");
        }
puerto=6000;
bzero((char *)&server, sizeof((char *)&server));

 server.sin_addr.s_addr=inet_addr("192.168.1.129");
 server.sin_family = AF_INET;
 server.sin_port = htons(6000);

 cliente.sin_addr.s_addr=inet_addr("192.168.1.130");
 cliente.sin_family = AF_INET;
 cliente.sin_port = htons(6000);

resp=bind(conexion,(struct sockaddr *)&cliente, sizeof(cliente));

 if(connect(conexion,(struct sockaddr *)&server, sizeof(server)) < 0)
 {
   printf("Error conectando con el host\n");
   close(conexion);
   return 1;
 }

//MOTORES

#ifndef __ARM_ARCH_4T__
        /* Disable auto-detection of the brick (you have to set the correct address below) */
        ev3_brick_addr = "192.168.1.130";

#endif
        if ( ev3_init() == -1 ) return ( 1 );
```

```
#ifndef __ARM_ARCH_4T__
	printf( "The EV3 brick auto-detection is DISABLED,\nwaiting %s online with plugged
tacho...\n", ev3_brick_addr );

#else
	printf( "Waiting tacho is plugged...\n" );

#endif
	while ( ev3_tacho_init() < 1 ) Sleep( 1000 );

	printf( "*** ( EV3 ) Hello! ***\n" );

	printf( "Found tacho motors:\n" );
	for ( j = 0; j < DESC_LIMIT; j++ ) {
		if ( ev3_tacho[ j ].type_inx != TACHO_TYPE__NONE_ ) {
			printf( "  type = %s\n", ev3_tacho_type( ev3_tacho[ j ].type_inx ));
			printf( "  port = %s\n", ev3_tacho_port_name( j, s ));


			}
	}

pthread_t idHilo1;
pthread_t idHilo2;
int error1;
int error2;
error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);

if( error2!=0 || error1!=0 )
	{
		perror("No puedo crear thread");
		exit(-1);
	}

	pthread_join(idHilo1,NULL);
	pthread_join(idHilo2,NULL);

 return (0);
}

void *funcionThread1(void *parametro){

struct coordenviada  Coordmensajee;

	i=0;

	num_itera=(int)((tfinal)/0.02);
```

```
while(i < (num_itera-240))        {

t1=clock();

        if((diferencia>=0.1) && (xrecibido!=0)){

                tacho_set_stop_action(OUTA,TACHO_HOLD);
                tacho_set_stop_action(OUTD,TACHO_HOLD);
                tacho_set_command(OUTA, TACHO_STOP);
                tacho_set_command(OUTD, TACHO_STOP);

        }



        else{

         refx=(0.5+(1.500-0.500)*i/(num_itera));
         refy=0;

         velxref=(refx-refxante)/0.02;
         velyref=0;



pos_d_nueva=tacho_get_position(OUTD,0)*pul2rad;
pos_i_nueva=tacho_get_position(OUTA,0)*pul2rad;


           vel_ang_d=(pos_d_nueva-pos_d_anterior)/Ts;
           vel_ang_i=(pos_i_nueva-pos_i_anterior)/Ts;

           vel_lin_d=vel_ang_d*radio_rueda;
           vel_lin_i=vel_ang_i*radio_rueda;

           vel_lin_robot=(vel_lin_d+ vel_lin_i)/2;

           vel_ang_robot=(vel_lin_d- vel_lin_i)/(2*b);

                x=x+vel_lin_robot*Ts*cos(ang);

                y=y+vel_lin_robot*Ts*sin(ang);

                ang=ang+vel_ang_robot*Ts;

           x_deriv=kvx*velxref+kpx*(refx-(x+(e*cos(ang))));
           y_deriv=kvy*velyref+kpy*(refy-(y+(e*sin(ang))));

vref_i=(((e*cos(ang)+b*sin(ang))*x_deriv)+((e*sin(ang)-b*cos(ang))*y_deriv))/e;
```

```
vref_d=(((e*cos(ang)-b*sin(ang))*x_deriv)+((e*sin(ang)+b*cos(ang))*y_deriv))/e;


                wref_d=vref_d/radio_rueda;
                wref_i=vref_i/radio_rueda;

                error_vel_d=wref_d-vel_ang_d;
                error_vel_i=wref_i-vel_ang_i;


acontrol_d=(4*error_vel_d)-(2.8*error_vel_d1)+(0.9*error_vel_d2)+acontrol_d1;

acontrol_i=(4*error_vel_i)-(2.8*error_vel_i1)+(0.9*error_vel_i2)+acontrol_i1;



                if (acontrol_d>100)
                        {
                                acontrol_d=100;
                        }
                if (acontrol_d<-100)
                        {
                                acontrol_d=-100;
                        }
                if (acontrol_i>100)
                        {
                                acontrol_i=100;
                        }
                if (acontrol_i<-100)
                        {
                                acontrol_i=-100;
                        }


tacho_set_duty_cycle_sp(OUTA,acontrol_i);
tacho_set_duty_cycle_sp(OUTD,acontrol_d);
tacho_set_command(OUTA, TACHO_RUN_DIRECT );
tacho_set_command(OUTD, TACHO_RUN_DIRECT);


                pos_d_anterior=pos_d_nueva;
                pos_i_anterior=pos_i_nueva;


        acontrol_d1=acontrol_d;
        acontrol_i1=acontrol_i;
        error_vel_d2=error_vel_d1;
        error_vel_i2=error_vel_i1;
        error_vel_d1=error_vel_d;
        error_vel_i1=error_vel_i;
```

```
            refxante=refx;
            xreal=x+0.4;
            yreal=y+0.4;


            Coordmensajee.posxe=xreal;
            Coordmensajee.posye=yreal;

        send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
        posxe[i]=Coordmensajee.posxe;
        posye[i]=Coordmensajee.posye;


i++;
}
t2=clock();
tf=(int)(20-(((t2-t1)/CLOCKS_PER_SEC))*1000);
sleep_ms(tf);
diferencia=xreal-xrecibido;
}

                fich=fopen("cliente.txt","w");
                int h;
                for(h=0;h<601;h++){
                fprintf(fich,"%f %f %f %f\n",posxe[h],posye[h],posxr                [h],posyr[h]);
                }

                tacho_set_stop_action(OUTA,TACHO_HOLD);
                tacho_set_stop_action(OUTD,TACHO_HOLD);
                tacho_set_command(OUTA, TACHO_STOP);
                tacho_set_command(OUTD, TACHO_STOP);

                tacho_reset(OUTB);
                tacho_reset(OUTD);

        fclose(fich);
        ev3_uninit();
        printf( "*** ( EV3 ) Bye! ***\n" );

}


void *funcionThread2(void *parametro){

struct coordrecibida  Coordmensajer;
```

```
                while(1){

        recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
        xrecibido=Coordmensajer.posxr;
        posxr[i]=xrecibido;

        yrecibido=Coordmensajer.posyr;
        posyr[i]=yrecibido;


                }


}
```

ANEXO VI: CÓDIGO EXPERIENCIA 4

SERVIDOR

```
 #include <pthread.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<math.h>
#include<time.h>

#include"ev3_light.h"
#include "ev3.h"
#include "ev3_port.h"
#include "ev3_tacho.h"
#include "brick.h"

// WIN32 /////////////////////////////////////
#ifdef __WIN32__

#include <windows.h>

// UNIX //////////////////////////////////////
#else

#include <unistd.h>
#define Sleep( msec ) usleep(( msec ) * 1000 )

//////////////////////////////////////////////
#endif

//VARIABLES
 struct sockaddr_in server;
 struct sockaddr_in cliente;
int puerto, conexion,resp, stsize,conexion_cliente;
char bufferposxe[100],bufferposye[100], bufferposxr[100],bufferposyr[100];
 FILE *fich;
 float xrecibido,yrecibido;
 socklen_t longc;
float posxe[1500];
float posxr[1500];
float posye[1500];
float posyr[1500];
float k=0;
char bufferk[100];
```

```
long j=0;
long i = 0;
float tfinal=12.0;
long num_itera=0;
float pos_d_nueva=0;
float pos_d_anterior=0;
float pos_i_nueva=0;
float pos_i_anterior=0;
float acontrol_d,acontrol_i;
float acontrol_d1=0,acontrol_i1=0;
float vel_ang_d, vel_ang_i;
float vel_lin_d, vel_lin_i;
float vref_d, vref_i;
float wref_d,wref_i;
float error_vel_d, error_vel_i;
float error_vel_d1=0, error_vel_i1=0;
float error_vel_d2=0, error_vel_i2=0;
float vel_lin_robot, vel_ang_robot;
float refx=0;
float refy=0;
float x_deriv=0.0;
float y_deriv=0.0;
float velxref=0.0;
float velyref=0.0;

float kpx=4.0;
float kpy=4.0;

float kvx=0.50;
float kvy=0.50;

float ang=-1.570796;
float x=0.0;
float y=0.8;
float b=0.061;
float e=0.061;
float pul2rad=0.0174533;
float radio_rueda=0.028;
float Ts=0.02;
float refxante=0;
float tiempo[650];
float parat=0;
float paratcua=0;
float P2x=0;
float P2y=0;
float xultimo=0;
clock_t t1;
clock_t t2;
int tf;
```

```
struct coordenviada
{
	float posxe;
	float posye;
	float kenv;
};

struct coordrecibida
{
	float posxr;
	float posyr;
};

void *funcionThread1 (void *parametro);
void *funcionThread2 (void *parametro);


int main(void){

tacho_reset(OUTA);
tacho_reset(OUTC);


//CONEXIONES SHOCKETS

conexion = socket(AF_INET, SOCK_STREAM, 0);

if(conexion == -1){

  printf("Error al crear el socket\n");
}

puerto=6000;
bzero((char *)&server, sizeof((char *)&server));

 server.sin_addr.s_addr=inet_addr("192.168.1.129");
 server.sin_family = AF_INET;
 server.sin_port = htons(6000);

resp=bind(conexion,(struct sockaddr *)&server, sizeof(server));

listen(conexion,1);

printf("Esperando conexiones entrantes... \n");

longc = sizeof(cliente);

conexion_cliente=accept(conexion,(struct sockaddr *)&cliente, &longc);
```

```
 if(conexion_cliente<0)
  {
    printf("Error al aceptar trafico\n");
    close(conexion);
    return 1;

}

//MOTORES

#ifndef __ARM_ARCH_4T__
        /* Disable auto-detection of the brick (you have to set the correct address below) */
        ev3_brick_addr = "192.168.1.129";

#endif
        if ( ev3_init() == -1 ) return ( 1 );

#ifndef __ARM_ARCH_4T__
        printf( "The EV3 brick auto-detection is DISABLED,\nwaiting %s online with plugged
tacho...\n", ev3_brick_addr );

#else
        printf( "Waiting tacho is plugged...\n" );

#endif
        while ( ev3_tacho_init() < 1 ) Sleep( 1000 );

        printf( "*** ( EV3 ) Hello! ***\n" );

        printf( "Found tacho motors:\n" );
        for ( j = 0; j < DESC_LIMIT; j++ ) {
                if ( ev3_tacho[ j ].type_inx != TACHO_TYPE__NONE_ ) {
                        printf( "  type = %s\n", ev3_tacho_type( ev3_tacho[ j ].type_inx ));
                        printf( "  port = %s\n", ev3_tacho_port_name( j, s ));


                        }
        }

pthread_t idHilo1;
pthread_t idHilo2;
int error1;
int error2;
error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);

if(  error1!=0 )
        {
                perror("No puedo crear thread");
                exit(-1);
```

```
        }

        pthread_join(idHilo1,NULL);
        pthread_join(idHilo2,NULL);



  return (0);
}

void *funcionThread1(void *parametro){

struct coordenviada  Coordmensajee;
struct coordrecibida  Coordmensajer;


        recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
        P2x=Coordmensajer.posxr;
        P2x=P2x-0.2;

        P2y=Coordmensajer.posyr;

                float P0x=0.00;
                float  P1x=0.0;

                float P0y=0.8;
                float  P1y=0.0;

        i=0;

        num_itera=(int)((tfinal)/0.02);


        while(i <2*num_itera){

        t1=clock();

                        if(i<=num_itera) {

                parat=(i*0.02)/tfinal;
                paratcua=parat*parat;
refx=((1.0-2.0*parat+paratcua)*P0x)+((2.0*parat-2.0*paratcua)*P1x)+paratcua*P2x;
 refy=((1.0-2.0*parat+paratcua)*P0y)+((2.0*parat-2.0*paratcua)*P1y)+paratcua*P2y;
                velxref=(2*P1x-2*P0x)+2*(P0x-2*P1x+P2x)*parat;
                velyref=(2*P1y-2*P0y)+2*(P0y-2*P1y+P2y)*parat;

                k=2;
                    }

                else{
```

```
                refx=(P2x+((1+P2x)-P2x)*(i-num_itera)/num_itera);
                refx=(0.35+(1.35-0.35)*(i-num_itera)/num_itera);
                refy=0.0;
                velxref=(refx-refxante)/0.02;
                velyref=0;

                k=1;

                }

    pos_d_nueva=tacho_get_position(OUTC,0)*pul2rad;
    pos_i_nueva=tacho_get_position(OUTA,0)*pul2rad;

                    vel_ang_d=(pos_d_nueva-pos_d_anterior)/Ts;
                    vel_ang_i=(pos_i_nueva-pos_i_anterior)/Ts;

                    vel_lin_d=vel_ang_d*radio_rueda;
                    vel_lin_i=vel_ang_i*radio_rueda;


                    vel_lin_robot=(vel_lin_d+ vel_lin_i)/2;

                    vel_ang_robot=(vel_lin_d- vel_lin_i)/(2*b);

                        x=x+vel_lin_robot*Ts*cos(ang);

                        y=y+vel_lin_robot*Ts*sin(ang);

                        ang=ang+vel_ang_robot*Ts;

                    x_deriv=kvx*velxref+kpx*(refx-(x+(e*cos(ang))));
                    y_deriv=kvy*velyref+kpy*(refy-(y+(e*sin(ang))));

vref_i=(((e*cos(ang)+b*sin(ang))*x_deriv)+((e*sin(ang)-b*cos(ang))*y_deriv))/e;
vref_d=(((e*cos(ang)-b*sin(ang))*x_deriv)+((e*sin(ang)+b*cos(ang))*y_deriv))/e;

                    wref_d=vref_d/radio_rueda;
                    wref_i=vref_i/radio_rueda;

                    error_vel_d=wref_d-vel_ang_d;
                    error_vel_i=wref_i-vel_ang_i;


acontrol_d=(4*error_vel_d)-(2.8*error_vel_d1)+(0.9*error_vel_d2)+acontrol_d1;

acontrol_i=(4*error_vel_i)-(2.8*error_vel_i1)+(0.9*error_vel_i2)+acontrol_i1;


                    if (acontrol_d>100)
```

```
                        {
                                acontrol_d=100;
                        }
                if (acontrol_d<-100)
                        {
                                acontrol_d=-100;
                        }
                if (acontrol_i>100)
                        {
                                acontrol_i=100;
                        }
                if (acontrol_i<-100)
                        {
                                acontrol_i=-100;
                        }


tacho_set_duty_cycle_sp(OUTA,acontrol_i);
tacho_set_duty_cycle_sp(OUTC,acontrol_d);
tacho_set_command(OUTA, TACHO_RUN_DIRECT );
tacho_set_command(OUTC, TACHO_RUN_DIRECT);


                pos_d_anterior=pos_d_nueva;
                pos_i_anterior=pos_i_nueva;

       acontrol_d1=acontrol_d;
       acontrol_i1=acontrol_i;
       error_vel_d2=error_vel_d1;
       error_vel_i2=error_vel_i1;
       error_vel_d1=error_vel_d;
       error_vel_i1=error_vel_i;
      refxante=refx;

            Coordmensajee.posxe=x;
           Coordmensajee.posye=y;
           Coordmensajee.kenv=k;

          send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
          posxe[i]=Coordmensajee.posxe;
          posye[i]=Coordmensajee.posye;


t2=clock();

tf=(int)(20-(((t2-t1)/CLOCKS_PER_SEC))*1000);

sleep_ms(tf);

i++;
```

```
	}

		fich=fopen("servidor.txt","w");
		int h;
		for(h=0;h<1400;h++){
		fprintf(fich,"%f %f %f %f\n",posxe[h],posye[h],posxr[h],posyr[h]);
		}
		tacho_set_stop_action(OUTA,TACHO_HOLD);
		tacho_set_stop_action(OUTC,TACHO_HOLD);
		tacho_set_command(OUTA, TACHO_STOP);
		tacho_set_command(OUTC, TACHO_STOP);

		tacho_reset(OUTA);
		tacho_reset(OUTC);

	fclose(fich);
	ev3_uninit();
	printf( "*** ( EV3 ) Bye! ***\n" );

}
void *funcionThread2(void *parametro1){

struct coordrecibida  Coordmensajer;

		while(1){


	recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
	xrecibido=Coordmensajer.posxr;
	posxr[i]=xrecibido;

	yrecibido=Coordmensajer.posyr;
	posyr[i]=yrecibido;

		}
}
```

CLIENTE

```
#include<stdio.h>
#include <pthread.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<math.h>
#include"ev3_light.h"
#include "ev3.h"
#include "ev3_port.h"
#include "ev3_tacho.h"
#include "brick.h"
#include<time.h>

// WIN32 /////////////////////////////////////
#ifdef __WIN32__

#include <windows.h>

// UNIX //////////////////////////////////////
#else

#include <unistd.h>
#define Sleep( msec ) usleep(( msec ) * 1000 )

//////////////////////////////////////////////
#endif
//VARIABLES
 struct sockaddr_in server;
 struct sockaddr_in cliente;
 int puerto, conexion,resp,j;
char bufferposxe[100],bufferposye[100], bufferposxr[100],bufferposyr[100];
 FILE *fich;
 float posactual[151],buffer1,x,y,ang,refx1,refx2,refy1,errorrefx,refxante,yrecibido;
float posxe[1200];
float posxr[1200];
float posye[1200];
float posyr[1200];
float xrecibido=0;
float diferenciax=0.0;
float diferenciay=0.0;
float xreal=0.0;
float yreal;
float k=0;
char bufferk[100];
```

```
float posk[1200];
float z=0;

        long i = 0;
        float tfinal=12.0;
        long num_itera=0;
        float pos_d_nueva=0;
        float pos_d_anterior=0;
        float pos_i_nueva=0;
        float pos_i_anterior=0;
        float acontrol_d,acontrol_i;
        float acontrol_d1=0,acontrol_i1=0;
        float vel_ang_d, vel_ang_i;
        float vel_lin_d, vel_lin_i;
        float vref_d, vref_i;
        float wref_d,wref_i;
        float error_vel_d, error_vel_i;
        float error_vel_d1=0, error_vel_i1=0;
        float error_vel_d2=0, error_vel_i2=0;
        float vel_lin_robot, vel_ang_robot;
        float refx,refy;
        float x_deriv=0.0;
        float y_deriv=0.0;
        float velxref=0.0;
        float velyref=0.0;

        float kpx=4.0;
        float kpy=4.0;

        float kvx=0.50;
        float kvy=0.50;

        float ang=0;
        float x=0.500;
        float y=0.0;
        float b=0.061;
        float e=0.061;
        float pul2rad=0.0174533;
        float radio_rueda=0.028;
        float Ts=0.02;
        float refxante=0;
       clock_t t1;
       clock_t t2;
       int tf;

struct coordenviada
{
        float posxe;
        float posye;
};
```

```
struct coordrecibida
{
	float posxr;
	float posyr;
	float krecv;
};

void *funcionThread1 (void *parametro);
void *funcionThread2 (void *parametro);

int main(void){

tacho_reset(OUTA);
tacho_reset(OUTD);

//CONEXIONES SHOCKETS
conexion = socket(AF_INET, SOCK_STREAM, 0);

if(conexion == -1)
	{
  printf("Error al crear el socket\n");
	}
puerto=6000;
bzero((char *)&server, sizeof((char *)&server));

 server.sin_addr.s_addr=inet_addr("192.168.1.129");
 server.sin_family = AF_INET;
 server.sin_port = htons(6000);

 cliente.sin_addr.s_addr=inet_addr("192.168.1.130");
 cliente.sin_family = AF_INET;
 cliente.sin_port = htons(6000);

resp=bind(conexion,(struct sockaddr *)&cliente, sizeof(cliente));

 if(connect(conexion,(struct sockaddr *)&server, sizeof(server)) < 0)
 {
   printf("Error conectando con el host\n");
   close(conexion);
   return 1;
 }

//MOTORES

#ifndef __ARM_ARCH_4T__
	/* Disable auto-detection of the brick (you have to set the correct address below) */
	ev3_brick_addr = "192.168.1.130";

#endif
```

```
	if ( ev3_init() == -1 ) return ( 1 );

#ifndef __ARM_ARCH_4T__
	printf( "The EV3 brick auto-detection is DISABLED,\nwaiting %s online with plugged
tacho...\n", ev3_brick_addr );

#else
	printf( "Waiting tacho is plugged...\n" );

#endif
	while ( ev3_tacho_init() < 1 ) Sleep( 1000 );

	printf( "*** ( EV3 ) Hello! ***\n" );

	printf( "Found tacho motors:\n" );
	for ( j = 0; j < DESC_LIMIT; j++ ) {
		if ( ev3_tacho[ j ].type_inx != TACHO_TYPE__NONE_ ) {
			printf( "  type = %s\n", ev3_tacho_type( ev3_tacho[ j ].type_inx ));
			printf( "  port = %s\n", ev3_tacho_port_name( j, s ));


			}
	}

pthread_t idHilo1;
pthread_t idHilo2;
int error1;
int error2;
error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);

if( error2!=0 || error1!=0 )
	{
		perror("No puedo crear thread");
		exit(-1);
	}

	pthread_join(idHilo1,NULL);
	pthread_join(idHilo2,NULL);

 return (0);
}
void *funcionThread1(void *parametro){

struct coordenviada  Coordmensajee;


	  Coordmensajee.posxe=x;
	 Coordmensajee.posye=y;
```

```
send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);

i=0;

num_itera=(int)((tfinal)/0.02);

while(i < (num_itera))  {

t1=clock();

if ((k==2) && z==0){

                tacho_set_stop_action(OUTA,TACHO_HOLD);
                tacho_set_stop_action(OUTD,TACHO_HOLD);
                tacho_set_command(OUTA, TACHO_STOP);
                tacho_set_command(OUTD, TACHO_STOP);


 }

if(k==1 || z==1){

        z=1;

        refx=(0.5+(1.500-0.500)*i/(num_itera));
        refy=0;

        velxref=(refx-refxante)/0.02;
        velyref=0;

pos_d_nueva=tacho_get_position(OUTD,0)*pul2rad;
pos_i_nueva=tacho_get_position(OUTA,0)*pul2rad;

          vel_ang_d=(pos_d_nueva-pos_d_anterior)/Ts;
          vel_ang_i=(pos_i_nueva-pos_i_anterior)/Ts;

          vel_lin_d=vel_ang_d*radio_rueda;
          vel_lin_i=vel_ang_i*radio_rueda;

          vel_lin_robot=(vel_lin_d+ vel_lin_i)/2;

          vel_ang_robot=(vel_lin_d- vel_lin_i)/(2*b);

                x=x+vel_lin_robot*Ts*cos(ang);

                y=y+vel_lin_robot*Ts*sin(ang);

                ang=ang+vel_ang_robot*Ts;

          x_deriv=kvx*velxref+kpx*(refx-(x+(e*cos(ang))));
```

```
                y_deriv=kvy*velyref+kpy*(refy-(y+(e*sin(ang))));


vref_i=(((e*cos(ang)+b*sin(ang))*x_deriv)+((e*sin(ang)-b*cos(ang))*y_deriv))/e;
vref_d=(((e*cos(ang)-b*sin(ang))*x_deriv)+((e*sin(ang)+b*cos(ang))*y_deriv))/e;

                wref_d=vref_d/radio_rueda;
                wref_i=vref_i/radio_rueda;

                error_vel_d=wref_d-vel_ang_d;
                error_vel_i=wref_i-vel_ang_i;

acontrol_d=(4*error_vel_d)-(2.8*error_vel_d1)+(0.9*error_vel_d2)+acontrol_d1;

acontrol_i=(4*error_vel_i)-(2.8*error_vel_i1)+(0.9*error_vel_i2)+acontrol_i1;


                if (acontrol_d>100)
                        {
                                acontrol_d=100;
                        }
                if (acontrol_d<-100)
                        {
                                acontrol_d=-100;
                        }
                if (acontrol_i>100)
                        {
                                acontrol_i=100;
                        }
                if (acontrol_i<-100)
                        {
                                acontrol_i=-100;
                        }


tacho_set_duty_cycle_sp(OUTA,acontrol_i);
tacho_set_duty_cycle_sp(OUTD,acontrol_d);
tacho_set_command(OUTA, TACHO_RUN_DIRECT );
tacho_set_command(OUTD, TACHO_RUN_DIRECT);


                pos_d_anterior=pos_d_nueva;
                pos_i_anterior=pos_i_nueva;

        acontrol_d1=acontrol_d;
        acontrol_i1=acontrol_i;
        error_vel_d2=error_vel_d1;
        error_vel_i2=error_vel_i1;
        error_vel_d1=error_vel_d;
        error_vel_i1=error_vel_i;
```

```
    refxante=refx;
Coordmensajee.posxe=x;
Coordmensajee.posye=y;


send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
        posxe[i]=Coordmensajee.posxe;
        posye[i]=Coordmensajee.posye;
i++;
}
t2=clock();
tf=(int)(20-(((t2-t1)/CLOCKS_PER_SEC))*1000);
sleep_ms(tf);
diferenciax=x-xrecibido;
diferenciay=yrecibido-y;
}

                fich=fopen("cliente.txt","w");
                int h;
                for(h=0;h<1200;h++){
fprintf(fich,"%f %f %f %f\n",posxe[h],posye[h],posxr[h],posyr[h],posk[h]);
                }

                tacho_set_stop_action(OUTA,TACHO_HOLD);
                tacho_set_stop_action(OUTD,TACHO_HOLD);
                tacho_set_command(OUTA, TACHO_STOP);
                tacho_set_command(OUTD, TACHO_STOP);

                tacho_reset(OUTA);
                tacho_reset(OUTD);

        fclose(fich);
        ev3_uninit();
        printf( "*** ( EV3 ) Bye! ***\n" );

}

void *funcionThread2(void *parametro){

struct coordrecibida  Coordmensajer;

                while(1){

        recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
```

```
xrecibido=Coordmensajer.posxr;
posxr[i]=xrecibido;

yrecibido=Coordmensajer.posyr;
posyr[i]=yrecibido;

k=Coordmensajer.krecv;
posk[i]=k;

        }
```

ANEXO VII: CÓDIGO EXPERIENCIA 5

SERVIDOR

```
#include <pthread.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<math.h>
#include<time.h>

#include"ev3_light.h"
#include "ev3.h"
#include "ev3_port.h"
#include "ev3_tacho.h"
#include "brick.h"


// WIN32 /////////////////////////////////////////
#ifdef __WIN32__

#include <windows.h>

// UNIX //////////////////////////////////////////
#else

#include <unistd.h>
#define Sleep( msec ) usleep(( msec ) * 1000 )

//////////////////////////////////////////////////
#endif

//VARIABLES
 struct sockaddr_in server;
 struct sockaddr_in cliente;
int puerto, conexion, resp, stsize, conexion_cliente;
 char bufferposxe[100],bufferposye[100], bufferposxr[100],bufferposyr[100],
 FILE *fich;
 float xrecibido,yrecibido;
 socklen_t longc;
float posxe[1500];
float posxr[1500];
float posye[1500];
```

```
float posyr[1500];
float k=0;
char bufferk[100];

        long j=0;
        long i = 0;
        float tfinal=12.0;
        long num_itera=0;
        float pos_d_nueva=0;
        float pos_d_anterior=0;
        float pos_i_nueva=0;
        float pos_i_anterior=0;
        float acontrol_d,acontrol_i;
        float acontrol_d1=0,acontrol_i1=0;
        float vel_ang_d, vel_ang_i;
        float vel_lin_d, vel_lin_i;
        float vref_d, vref_i;
        float wref_d,wref_i;
        float error_vel_d, error_vel_i;
        float error_vel_d1=0, error_vel_i1=0;
        float error_vel_d2=0, error_vel_i2=0;
        float vel_lin_robot, vel_ang_robot;
        float refx=0;
        float refy=0;
        float x_deriv=0.0;
        float y_deriv=0.0;
        float velxref=0.0;
        float velyref=0.0;

        float kpx=4.0;
        float kpy=4.0;

        float kvx=0.50;
        float kvy=0.50;

        float ang=0.0;
        float x=1;
        float y=1.0;
        float b=0.061;
        float e=0.061;
        float pul2rad=0.0174533;
        float radio_rueda=0.028;
        float Ts=0.02;
        float refxante=0;
        float tiempo[650];
        float parat=0;
        float paratcua=0;
        float P2x=0;
        float P2y=0;
        float yreal=0;
```

```
	clock_t t1;
	clock_t t2;
	int tf;

struct coordenviada
{
	float posxe;
	float posye;
};

struct coordrecibida
{
	float posxr;
	float posyr;
};

void *funcionThread1 (void *parametro);
void *funcionThread2 (void *parametro);
int main(void){

tacho_reset(OUTA);
tacho_reset(OUTC);

//CONEXIONES SHOCKETS
conexion = socket(AF_INET, SOCK_STREAM, 0);

if(conexion == -1){

  printf("Error al crear el socket\n");
}

puerto=6000;
bzero((char *)&server, sizeof((char *)&server));

 server.sin_addr.s_addr=inet_addr("192.168.1.129");
 server.sin_family = AF_INET;
 server.sin_port = htons(6000);

resp=bind(conexion,(struct sockaddr *)&server, sizeof(server));

listen(conexion,1);

printf("Esperando conexiones entrantes... \n");

longc = sizeof(cliente);

conexion_cliente=accept(conexion,(struct sockaddr *)&cliente, &longc);

 if(conexion_cliente<0)
  {
```

```
   printf("Error al aceptar trafico\n");
   close(conexion);
   return 1;

}
//MOTORES

#ifndef __ARM_ARCH_4T__
        /* Disable auto-detection of the brick (you have to set the correct address below) */
        ev3_brick_addr = "192.168.1.129";

#endif
        if ( ev3_init() == -1 ) return ( 1 );

#ifndef __ARM_ARCH_4T__
        printf( "The EV3 brick auto-detection is DISABLED,\nwaiting %s online with plugged
tacho...\n", ev3_brick_addr );

#else
        printf( "Waiting tacho is plugged...\n" );

#endif
        while ( ev3_tacho_init() < 1 ) Sleep( 1000 );

        printf( "*** ( EV3 ) Hello! ***\n" );

        printf( "Found tacho motors:\n" );
        for ( j = 0; j < DESC_LIMIT; j++ ) {
                if ( ev3_tacho[ j ].type_inx != TACHO_TYPE__NONE_ ) {
                        printf( "  type = %s\n", ev3_tacho_type( ev3_tacho[ j ].type_inx ));
                        printf( "  port = %s\n", ev3_tacho_port_name( j, s ));


                        }
        }

pthread_t idHilo1;
pthread_t idHilo2;
int error1;
int error2;
error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);

if(  error1!=0 )
        {
                perror("No puedo crear thread");
                exit(-1);
        }

        pthread_join(idHilo1,NULL);
```

```
        pthread_join(idHilo2,NULL);

  return (0);
}
void *funcionThread1(void *parametro){

struct coordenviada  Coordmensajee;

        i=0;

        num_itera=(int)((tfinal)/0.02);

        while(i <2*num_itera){

        t1=clock();

        if (i< num_itera)
         {
        refx=1+(2.000-1.000)*i/(num_itera);
        refy=1;
        velxref=(2.000-1.000)/(num_itera);
        velyref=0;
      }
        else  {
        refx=2.000;
        refy=1+(2.000-1.0)*(i-(num_itera))/(num_itera);
       velxref=0;
       velyref=(2.000-1)/(num_itera);
      }

         pos_d_nueva=tacho_get_position(OUTC,0)*pul2rad;
         pos_i_nueva=tacho_get_position(OUTA,0)*pul2rad;

                  vel_ang_d=(pos_d_nueva-pos_d_anterior)/Ts;
                  vel_ang_i=(pos_i_nueva-pos_i_anterior)/Ts;

                  vel_lin_d=vel_ang_d*radio_rueda;
                  vel_lin_i=vel_ang_i*radio_rueda;

                  vel_lin_robot=(vel_lin_d+ vel_lin_i)/2;

                  vel_ang_robot=(vel_lin_d- vel_lin_i)/(2*b);

                        x=x+vel_lin_robot*Ts*cos(ang);

                        y=y+vel_lin_robot*Ts*sin(ang);

                        ang=ang+vel_ang_robot*Ts;

                  x_deriv=kvx*velxref+kpx*(refx-(x+(e*cos(ang))));
```

```
                y_deriv=kvy*velyref+kpy*(refy-(y+(e*sin(ang))));

vref_i=(((e*cos(ang)+b*sin(ang))*x_deriv)+((e*sin(ang)-b*cos(ang))*y_deriv))/e;
vref_d=(((e*cos(ang)-b*sin(ang))*x_deriv)+((e*sin(ang)+b*cos(ang))*y_deriv))/e;

                wref_d=vref_d/radio_rueda;
                wref_i=vref_i/radio_rueda;

                error_vel_d=wref_d-vel_ang_d;
                error_vel_i=wref_i-vel_ang_i;

acontrol_d=(4*error_vel_d)-(2.8*error_vel_d1)+(0.9*error_vel_d2)+acontrol_d1;

acontrol_i=(4*error_vel_i)-(2.8*error_vel_i1)+(0.9*error_vel_i2)+acontrol_i1;


                if (acontrol_d>100)
                        {
                                acontrol_d=100;
                        }
                if (acontrol_d<-100)
                        {
                                acontrol_d=-100;
                        }
                if (acontrol_i>100)
                        {
                                acontrol_i=100;
                        }
                if (acontrol_i<-100)
                        {
                                acontrol_i=-100;
                        }


tacho_set_duty_cycle_sp(OUTA,acontrol_i);
tacho_set_duty_cycle_sp(OUTC,acontrol_d);
tacho_set_command(OUTA, TACHO_RUN_DIRECT );
tacho_set_command(OUTC, TACHO_RUN_DIRECT);


                pos_d_anterior=pos_d_nueva;
                pos_i_anterior=pos_i_nueva;


        acontrol_d1=acontrol_d;
        acontrol_i1=acontrol_i;
        error_vel_d2=error_vel_d1;
        error_vel_i2=error_vel_i1;
        error_vel_d1=error_vel_d;
        error_vel_i1=error_vel_i;
```

```
      refxante=refx;



            Coordmensajee.posxe=x;
           Coordmensajee.posye=y;

          send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
          posxe[i]=Coordmensajee.posxe;
          posye[i]=Coordmensajee.posye;


t2=clock();

tf=(int)(20-(((t2-t1)/CLOCKS_PER_SEC))*1000);

sleep_ms(tf);

i++;
          }

                    fich=fopen("servidor.txt","w");
                    int h;
                    for(h=0;h<1200;h++){
                    fprintf(fich,"%f %f %f %f \n",posxe[h],posye[h],posxr[h],posyr[h]);
                    }
                    tacho_set_stop_action(OUTA,TACHO_HOLD);
                    tacho_set_stop_action(OUTC,TACHO_HOLD);
                    tacho_set_command(OUTA, TACHO_STOP);
                    tacho_set_command(OUTC, TACHO_STOP);

                    tacho_reset(OUTA);
                    tacho_reset(OUTC);

          fclose(fich);
          ev3_uninit();
          printf( "*** ( EV3 ) Bye! ***\n" );

}

void *funcionThread2(void *parametro1){

struct coordrecibida  Coordmensajer;

                    while(1){
          recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
          xrecibido=Coordmensajer.posxr;
          posxr[i]=xrecibido;
          yrecibido=Coordmensajer.posyr;
          posyr[i]=yrecibido;
```

```
                }
}
```

CLIENTE

```
#include<stdio.h>
#include <pthread.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<math.h>
#include"ev3_light.h"
#include "ev3.h"
#include "ev3_port.h"
#include "ev3_tacho.h"
#include "brick.h"
#include<time.h>

// WIN32 /////////////////////////////////////
#ifdef __WIN32__

#include <windows.h>

// UNIX //////////////////////////////////////
#else

#include <unistd.h>
#define Sleep( msec ) usleep(( msec ) * 1000 )

//////////////////////////////////////////////////
#endif

//VARIABLES
 struct sockaddr_in server;
 struct sockaddr_in cliente;
int puerto, conexion,resp,j;
char,bufferposxe[100],bufferposye[100], bufferposxr[100],bufferposyr[100];
FILE *fich;
 float refxante,yrecibido;
float posxe[1200];
float posxr[1200];
float posye[1200];
float posyr[1200];
float velrx[1200];
float velry[1200];
float xrecibido=0;
```

```
float diferenciax=0.0;
float diferenciay=0.0;
float xreal=0.0;
float yreal;
float irecibido=0;
float k=0;
char bufferk[100];
float posk[1200];
float z=0;

        long i = 0;
        float tfinal=12.0;
        long num_itera=0;
        float pos_d_nueva=0;
        float pos_d_anterior=0;
        float pos_i_nueva=0;
        float pos_i_anterior=0;
        float acontrol_d,acontrol_i;
        float acontrol_d1=0,acontrol_i1=0;
        float vel_ang_d, vel_ang_i;
        float vel_lin_d, vel_lin_i;
        float vref_d, vref_i;
        float wref_d,wref_i;
        float error_vel_d, error_vel_i;
        float error_vel_d1=0, error_vel_i1=0;
        float error_vel_d2=0, error_vel_i2=0;
        float vel_lin_robot, vel_ang_robot;
        float refx=0.8;
        float refy=1;
        float x_deriv=0.0;
        float y_deriv=0.0;
        float velxref;
        float velyref;

        float kpx=4.0;
        float kpy=4.0;

        float kvx=0.50;
        float kvy=0.50;

        float ang=0.0;
        float x=0.8;
        float y=1;
        float b=0.061;
        float e=0.061;
        float pul2rad=0.0174533;
        float radio_rueda=0.028;
        float Ts=0.02;
        float refxante=0;
         float parat=0;
```

```
            float paratcua=0;
            clock_t t1;
            clock_t t2;
            int tf;

struct coordenviada
{
          float posxe;
          float posye;
};

struct coordrecibida
{
          float posxr;
          float posyr;
};


void *funcionThread1 (void *parametro);
void *funcionThread2 (void *parametro);

int main(void){

tacho_reset(OUTA);
tacho_reset(OUTD);

//CONEXIONES SHOCKETS
conexion = socket(AF_INET, SOCK_STREAM, 0);

if(conexion == -1)
          {
  printf("Error al crear el socket\n");
          }

puerto=6000;
bzero((char *)&server, sizeof((char *)&server));

 server.sin_addr.s_addr=inet_addr("192.168.1.129");
 server.sin_family = AF_INET;
 server.sin_port = htons(6000);

 cliente.sin_addr.s_addr=inet_addr("192.168.1.130");
 cliente.sin_family = AF_INET;
 cliente.sin_port = htons(6000);

resp=bind(conexion,(struct sockaddr *)&cliente, sizeof(cliente));

 if(connect(conexion,(struct sockaddr *)&server, sizeof(server)) < 0)
 {
   printf("Error conectando con el host\n");
```

```
   close(conexion);
   return 1;
 }

//MOTORES

#ifndef __ARM_ARCH_4T__
        /* Disable auto-detection of the brick (you have to set the correct address below) */
        ev3_brick_addr = "192.168.1.130";

#endif
        if ( ev3_init() == -1 ) return ( 1 );

#ifndef __ARM_ARCH_4T__
        printf( "The EV3 brick auto-detection is DISABLED,\nwaiting %s online with plugged
tacho...\n", ev3_brick_addr );

#else
        printf( "Waiting tacho is plugged...\n" );

#endif
        while ( ev3_tacho_init() < 1 ) Sleep( 1000 );

        printf( "*** ( EV3 ) Hello! ***\n" );

        printf( "Found tacho motors:\n" );
        for ( j = 0; j < DESC_LIMIT; j++ ) {
                if ( ev3_tacho[ j ].type_inx != TACHO_TYPE__NONE_ ) {
                        printf( "  type = %s\n", ev3_tacho_type( ev3_tacho[ j ].type_inx ));
                        printf( "  port = %s\n", ev3_tacho_port_name( j, s ));


                        }
        }

pthread_t idHilo1;
pthread_t idHilo2;
int error1;
int error2;
error1=pthread_create(&idHilo1,NULL, funcionThread1, NULL);
error2=pthread_create(&idHilo2,NULL, funcionThread2, NULL);

if( error2!=0 || error1!=0 )
        {
                perror("No puedo crear thread");
                exit(-1);
        }

        pthread_join(idHilo1,NULL);
        pthread_join(idHilo2,NULL);
```

```
  return (0);
}

void *funcionThread1(void *parametro){

struct coordenviada  Coordmensajee;

        i=0;

        num_itera=(int)((tfinal)/0.02);

        t1=clock();

        while(i < (2*num_itera)){

        if(xrecibido==0.0 || yrecibido==0.0){

                        refx=refx;
                        refy=refy;
        }

        else {
                        refx=xrecibido-0.2;
                        refy=yrecibido;

                }


         if (i< num_itera){

                    velxref=(2.000-1.000)/(num_itera);
                    velyref=0;
                   }

        else{
                    velxref=0;
                    velyref=(2.000-1)/(num_itera);
                    }

    pos_d_nueva=tacho_get_position(OUTD,0)*pul2rad;
    pos_i_nueva=tacho_get_position(OUTA,0)*pul2rad;


                      vel_ang_d=(pos_d_nueva-pos_d_anterior)/Ts;
                      vel_ang_i=(pos_i_nueva-pos_i_anterior)/Ts;

                      vel_lin_d=vel_ang_d*radio_rueda;
                      vel_lin_i=vel_ang_i*radio_rueda;
```

```
                vel_lin_robot=(vel_lin_d+ vel_lin_i)/2;

                vel_ang_robot=(vel_lin_d- vel_lin_i)/(2*b);

                        x=x+vel_lin_robot*Ts*cos(ang);

                        y=y+vel_lin_robot*Ts*sin(ang);

                        ang=ang+vel_ang_robot*Ts;

                x_deriv=kvx*velxref+kpx*(refx-(x+(e*cos(ang))));
                y_deriv=kvy*velyref+kpy*(refy-(y+(e*sin(ang))));

  vref_i=(((e*cos(ang)+b*sin(ang))*x_deriv)+((e*sin(ang)-b*cos(ang))*y_deriv))/e;
  vref_d=(((e*cos(ang)-b*sin(ang))*x_deriv)+((e*sin(ang)+b*cos(ang))*y_deriv))/e;

                wref_d=vref_d/radio_rueda;
                wref_i=vref_i/radio_rueda;



                error_vel_d=wref_d-vel_ang_d;
                error_vel_i=wref_i-vel_ang_i;

acontrol_d=(4*error_vel_d)-(2.8*error_vel_d1)+(0.9*error_vel_d2)+acontrol_d1;

acontrol_i=(4*error_vel_i)-(2.8*error_vel_i1)+(0.9*error_vel_i2)+acontrol_i1;


                if (acontrol_d>100)
                        {
                                acontrol_d=100;
                        }
                if (acontrol_d<-100)
                        {
                                acontrol_d=-100;
                        }
                if (acontrol_i>100)
                        {
                                acontrol_i=100;
                        }
                if (acontrol_i<-100)
                        {
                                acontrol_i=-100;
                        }


tacho_set_duty_cycle_sp(OUTA,acontrol_i);
tacho_set_duty_cycle_sp(OUTD,acontrol_d);
tacho_set_command(OUTA, TACHO_RUN_DIRECT );
```

```
tacho_set_command(OUTD, TACHO_RUN_DIRECT);


                    pos_d_anterior=pos_d_nueva;
                    pos_i_anterior=pos_i_nueva;


    acontrol_d1=acontrol_d;
    acontrol_i1=acontrol_i;
    error_vel_d2=error_vel_d1;
    error_vel_i2=error_vel_i1;
    error_vel_d1=error_vel_d;
    error_vel_i1=error_vel_i;
    refxante=refx;

        Coordmensajee.posxe=x;
        Coordmensajee.posye=y;

      send(conexion_cliente,&Coordmensajee , sizeof(Coordmensajee), 0);
      posxe[i]=Coordmensajee.posxe;
      posye[i]=Coordmensajee.posye;


t2=clock();

tf=(int)(20-(((t2-t1)/CLOCKS_PER_SEC))*1000);

sleep_ms(tf);

i++;


}

                    fich=fopen("cliente.txt","w");
                    int h;
                    for(h=0;h<1200;h++){
fprintf(fich,"%f %f %f %f \n",posxe[h],posye[h],posxr                [h],posyr[h]);
                    }

                    tacho_set_stop_action(OUTA,TACHO_HOLD);
                    tacho_set_stop_action(OUTD,TACHO_HOLD);
                    tacho_set_command(OUTA, TACHO_STOP);
                    tacho_set_command(OUTD, TACHO_STOP);

                    tacho_reset(OUTA);
                    tacho_reset(OUTD);

      fclose(fich);
      ev3_uninit();
```

```
        printf( "*** ( EV3 ) Bye! ***\n" );

}


void *funcionThread2(void *parametro){

struct coordrecibida  Coordmensajer;

                while(1){

        recv(conexion,&Coordmensajer,sizeof(Coordmensajer), 0);
        xrecibido=Coordmensajer.posxr;
        posxr[i]=xrecibido;

        yrecibido=Coordmensajer.posyr;
        posyr[i]=yrecibido;

                }


}
```