UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

**TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES**

# DESARROLLO DE ALGORITMOS DE NAVEGACIÓN E IMPLEMENTACIÓN MEDIANTE GENERACIÓN AUTOMÁTICA DE CÓDIGO APLICADO A ROBOTS MÓVILES LEGO MINDSTORMS EV3


AUTOR: ANDRÉS FORTALEZA LOBILLO

TUTOR: ÁNGEL VALERA FERNÁNDEZ

COTUTOR: ENRIQUE JORGE BERNABEU SOLER



**Curso Académico: 2018-19**

*"A Raquel, por soportarme todos los días*

*A mi familia, por apoyarme en todo lo que hago*

*A mis compañeros, sin ellos no hubiese sido lo mismo ir todos los días a clase*

*A mis tutores, Ángel y Enrique, por toda la ayuda ofrecida estos últimos meses"*

# ÍNDICE

- Documento I: Memoria
- Documento II: Presupuesto

# ÍNDICE DE LA MEMORIA

# ÍNDICE DEL PRESUPUESTO

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# DOCUMENTO I:

# MEMORIA

# 1 INTRODUCCIÓN

## 1.1 Introducción y motivación

Hace décadas que el ser humano busca crear aparatos capaces de resolver problemas que las personas, bien son incapaces de resolver, o bien tardarían un valioso tiempo en hacerlo. Algunos de estos problemas pueden ser el control de robots que se muevan de forma autómata o la resolución de un complejo problema matemático.

Hoy en día, aunque aún se continúa investigando en este campo, existen ordenadores capaces de recibir, procesar y enviar datos en milésimas de segundo; routers que permiten establecer conexiones inalámbricas entre dos o más dispositivos; etc.

El uso conjunto de los aparatos anteriores, ha permitido el desarrollo de sectores como el de la robótica o el de la visión artificial, ya que, si miramos a nuestro entorno, cada día es más común que alguien tenga un robot de limpieza en casa o que las factorías empleen robots industriales para realizar procesos de producción.

A pesar de todos estos avances, la investigación en los campos de la robótica y la visión artificial no cesan, puesto que aún quedan muchas aplicaciones útiles en las que se pueden integrar estos dos campos de la tecnología.

En este Trabajo Fin de Grado se tratará de combinar la detección de objetos mediante visión artificial y su posterior envío y procesado con el desarrollo de algoritmos para la generación de trayectorias y control de movimiento de robots móviles. Todo ello se llevará a cabo mediante el uso de Matlab para establecer las comunicaciones y generar las trayectorias, y Simulink para generar de forma automática el código que controla el movimiento de los robots móviles.

## 1.2 Objetivos

El objetivo principal de este trabajo es comprender, diseñar, analizar y desarrollar el control de trayectorias de robots móviles. Con el fin de que sea de utilidad en futuros trabajos y proyectos, para la programación de los robots se ha optado por Matlab y Simulink, la herramienta de diseño de sistemas de control basado en computador más utilizada en el área de Ingeniería de Sistemas y Automática. En cuanto al hardware, se ha seleccionado la plataforma de Lego Mindstorms, en concreto la última versión: EV3.

Para tratar de conseguir este objetivo principal se pueden concretar los siguientes objetivos específicos:

- Estudiar los modelos cinemáticos de robots móviles y escoger el tipo de modelo que mejor se acople a las necesidades del trabajo.

- Estudiar los métodos de navegación de control de robots mediante seguimiento de trayectorias y de seguimiento de caminos.
- Desarrollar los algoritmos de navegación de control de trayectorias por punto descentralizado y de control de caminos mediante la persecución pura.
- Ajustar y validar dichos algoritmos de navegación tanto a nivel de simulación como con robots móviles reales.
- Estudiar y establecer las comunicaciones necesarias entre el robot móvil y el ordenador de control.
- Generar aplicaciones para la generación automática de trayectorias entre dos posiciones empleando técnicas de visión artificial.
- Verificar el funcionamiento de los algoritmos desarrollados en Simulink mediante pruebas experimentales con robots móviles Lego Mindstorms EV3.

# 2 DESARROLLO TEÓRICO

En este apartado se van a tratar de desarrollar y profundizar aquellos temas que son importantes para el desarrollo práctico de este trabajo.

## 2.1 Introducción a la robótica

Fue en 1921 cuando el dramaturgo checo, Karel Čapek, acuñó el término robot en su obra "R.U.R (Rossum's Universal Robots)". El término robot es la traducción al inglés de la palabra checa "robota", que en el contexto de "R.U.R (Rossum's Universal Robots)" significa máquinas trabajadoras y serviles. **[1]**

A día de hoy, la RAE define el término robot de la siguiente forma: "Máquina o ingenio electrónico programable que es capaz de manipular objetos y realizar diversas operaciones."**[2]**

Resumiendo, un robot es una máquina que tiene un procesador que analiza los datos que le suministran sensores u otros receptores y desarrolla una respuesta enviándola posteriormente a unos actuadores. Con ello, esta máquina es capaz de imitar el movimiento de seres vivos con el fin de realizar las tareas para las que ha sido creada.

Por otra parte, la robótica es la ciencia que se encarga de estudiar a los robots. Se trata de un campo que engloba a la electrónica, la informática, la física, el control y la mecánica con el fin de que dichos aparatos realicen los servicios que les son propuestos.

## 2.2 Tipologías de robots

Como se ha dicho, los robots son diseñados y programados para realizar una tarea determinada, y dependiendo del tipo de tarea, los robots se pueden clasificar en dos grupos:

Por un lado se encuentran los robots manipuladores, que como su propio nombre indica, son usados para manipular elementos. Debido a su uso, estos robots suelen estar anclados sobre una superficie y pueden presentar formas de tipo cartesianas (movimientos sobre unos ejes) o antropomórficas (brazo robótico articulado).

Cabe decir que este tipo de robots se ajusta muy bien a entornos estructurados, es por ello por lo que principalmente son empleados en el ámbito industrial.

Fig. 2.2: Robot cartesiano de 3 ejes [2]


Fig. 2.1: Brazo robótico articulado [1]

Por otro lado, están los robots móviles. Estos robots presentan unos mecanismos para la transmisión del movimiento (ruedas, patas, etc.) que permiten su desplazamiento. Esta tipología de robots, además de ser utilizada en la industria (transporte de materias primas, etc.), se emplea en sectores como el doméstico (robots de limpieza, de juguete, etc.).

Fig. 2.4: Robot aspirador [3]


Fig. 2.3: Robot móvil industrial [4]

### 2.2.1 Configuraciones cinemáticas de robots móviles

Dentro del campo de los robots móviles existen diversas configuraciones cinemáticas. A continuación se verán las cuatro configuraciones más empleadas en el diseño de robots móviles:

- *Configuración diferencial:*
  En este tipo de configuración, el robot presenta dos ruedas de tracción que giran de forma independiente y una o dos ruedas de apoyo llamadas ruedas locas.

El movimiento de este tipo de robots se realiza actuando sobre la velocidad de giro de las ruedas, es decir, si se quiere girar el robot, una rueda deberá girar a una velocidad mayor que la otra.

Como ventaja se puede destacar que se trata de un sistema barato (solo se actúa sobre dos ruedas), y como desventaja cabe decir que si las ruedas de tracción deslizan, el controlador arrastrará un error que provocará que el robot no siga las referencias marcadas.

**Fig. 2.5: Robot móvil con configuración diferencial**

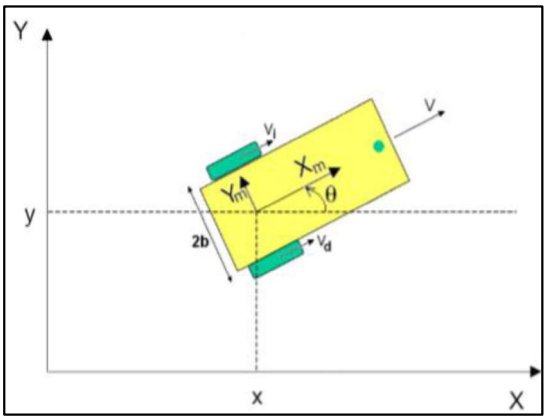


**Fig. 2.6: Representación configuración diferencial [5]**

- *Configuración de oruga:*

  En esta configuración los robots se asemejan a los tanques de guerra en el sentido de que poseen dos cadenas laterales. Para controlar el movimiento del robot basta con asemejar este tipo de configuración a la diferencial suponiendo que el robot posee dos ruedas de tracción en el centro de las cadenas laterales.

  La principal ventaja de este sistema es que es mucho menos propenso a que aparezcan deslizamientos con el suelo, y con ello, errores en el control cinemático del robot.

**Fig. 2.8: Robot con configuración de oruga [7]**

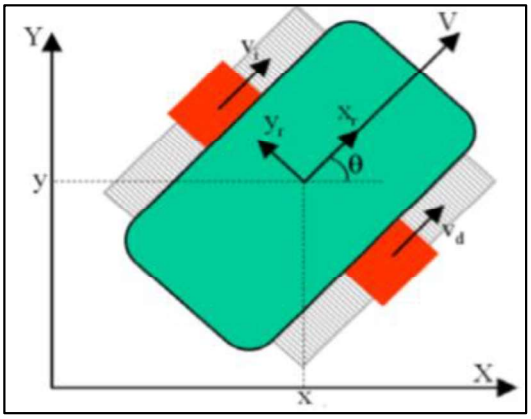


**Fig. 2.7: Representación configuración de oruga [6]**

- *Configuración triciclo:*
  Como su nombre indica, este tipo de configuración presenta tres ruedas, una de tracción, trasera o delantera, y orientable según se quiera que gire el robot, y dos ruedas de arrastre.
  Para el control de este robot solo se puede actuar sobre el giro de la rueda orientable y su velocidad de avance.
  Su ventaja respecto a las dos configuraciones anteriores es que se posee más adherencia al suelo.

**Fig. 2.10: Robot con configuración de triciclo [9]**

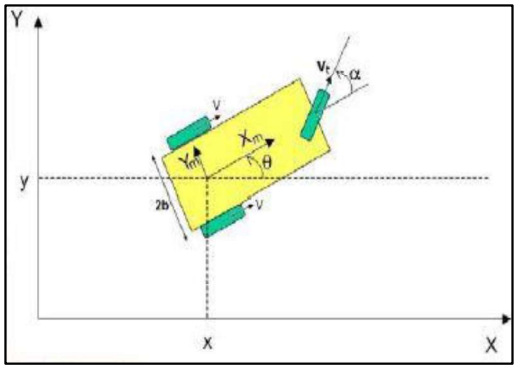


**Fig. 2.9: Esquema configuración de triciclo [8]**

- *Configuración Ackerman:*
  Se trata de una configuración en la que los robots presentan cuatro ruedas con la misma disposición que en los automóviles, las dos ruedas delanteras son las ruedas directrices y las traseras son las motrices.
  En el control cinemático de este tipo de robots se considera una rueda directriz equivalente a las dos delanteras y situada en el punto medio del eje que las une de forma que el robot pasa a "tener" tres ruedas en lugar de cuatro y así su control pasa a ser idéntico que el de configuración en triciclo.

**Fig. 2.12: Robot con configuración Ackerman [11]**

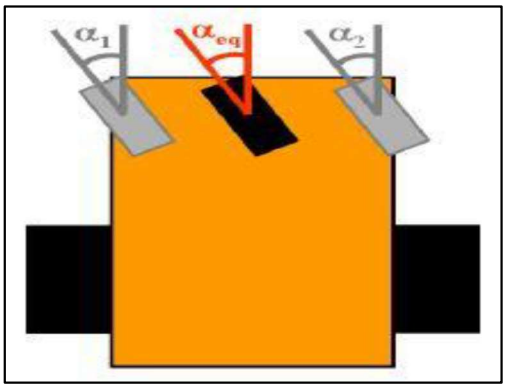


**Fig. 2.11: Esquema equivalente configuración Ackerman [10]**

## 2.3 Control de posición de robots móviles

Para que un robot móvil ejecute las acciones que se le ordena, como seguir una trayectoria, ha de tener un controlador que según las referencias estipuladas, genere unas acciones de control que hagan que los actuadores del robot sigan dichas referencias (Siegwart y Nourbakhsh, 2004).

Para ello se puede tratar de emplear controladores en bucle abierto o en bucle cerrado. Lo que ocurre es que, como es bien conocido, la utilización de controladores en bucle abierto, puede provocar que el robot no siga la referencia fijada. El hecho de no existir realimentación por parte del robot al control puede hacer que los posibles errores ocurridos durante el proceso de control no se tengan en cuenta. Estos problemas se pueden evitar empleando controladores en bucle cerrado puesto que en este caso sí que se tienen en cuenta los errores entre la posición y orientación del robot y las de referencia.

Por todo ello, se dispondrá de un esquema de control como el mostrado en la Figura 2.13. Se trata de un controlador en bucle cerrado en el que se pueden observar dos partes bien diferenciadas: el controlador cinemático y el robot en sí mismo.

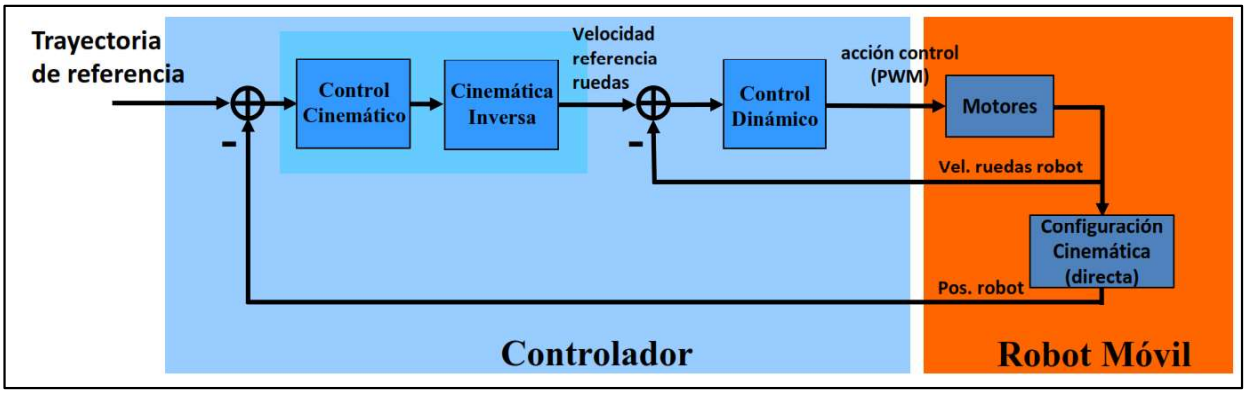


**Fig. 2.13: Esquema de control de robots móviles [12]**

El controlador (parte azul) tiene 2 bucles: un control cinemático y un control más interno dinámico. El primer controlador se encarga de generar, a partir de las referencias de movimiento, las referencias a los actuadores (en este caso motores eléctricos). El control dinámico se encarga de establecer la regulación de posición y/o velocidad de dichos actuadores.

En la parte del robot móvil (parte naranja), están contenidos los motores que hacen que éste se mueva en función del modelo cinemático del robot, en concreto la cinemática directa. En el robot móvil se tiene una realimentación de la posición de las ruedas del robot calculadas a

partir de los encoders del robot. Así se puede establecer un control en bucle cerrado tanto de velocidad como de posición, en función de la configuración cinemática del robot.

A continuación se abordará el estudio del control cinemático (cinemática directa e inversa) y del control dinámico teniendo en cuenta que el robot móvil estudiado en este trabajo presenta una configuración diferencial.

### 2.3.1 Control cinemático

Cuando se trabaja con robots se pueden distinguir dos problemas bien diferenciados: el problema de la cinemática directa y el de la cinemática inversa.

Con el problema de la cinemática directa se puede calcular las coordenadas y orientación del robot a partir de la información proporcionada por las posiciones de las articulaciones del robot. En el caso de los robots móviles, dichas articulaciones son las ruedas del robot.

Por otro lado, con el problema de la cinemática inversa se persigue obtener las velocidades de referencia que han de tener las ruedas para que el robot siga la trayectoria de referencia que se la ha fijado en el espacio cartesiano.

Cuando se trabaja con robots móviles se pueden establecer dos tipos diferentes de controladores: el control de trayectorias y el algoritmo de control de caminos. Lo único que diferencia a estos dos tipos de algoritmos es que en el control de trayectoria se debe especificar el tiempo como una variable, mientras que en el control de camino únicamente se especifica el movimiento geométrico que el robot debe hacer, independientemente del tiempo (Siegwart y Nourbakhsh, 2004).

Independientemente que se trate de un control de trayectoria o de camino, para poder abordar el desarrollo de estos controladores es necesario obtener el modelo cinemático directo e inverso del robot. En los apartados siguientes se abordará cómo obtener dichos modelos.

#### 2.3.1.1 Cinemática directa

A continuación se va a tratar de profundizar en las expresiones necesarias para establecer la cinemática directa.

La velocidad lineal, $v$, y angular, $\omega$, del robot puede ser calculada a partir de las velocidades lineales de cada rueda, $v_d$ y $v_i$, mediante las expresiones (2.1) y (2.2) respectivamente.

$$v = \frac{v_i + v_d}{2} \tag{2.1}$$

$$\omega = \frac{v_d - v_i}{2b} \tag{2.2}$$

Reescribiendo estas dos expresiones en forma matricial se obtiene la expresión (2.3):

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ -1/2b & 1/2b \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix} \tag{2.3}$$

El cálculo de las velocidades lineales de las ruedas se realiza con la siguiente expresión:

$$v_{\substack{lineal \\ rueda}} = R \times \frac{(enc_k - enc_{k-1})}{T_s} \times \frac{2\pi}{360} \tag{2.4}$$

donde R es el radio de las ruedas en metros, $T_s$ es el periodo de muestreo en segundos y $enc_k$ ,$enc_{k-1}$ son las lectura en grados sexagesimales del ángulo que ha girado la rueda en un determinado instante y justamente el anterior, respectivamente.

Por otro lado, observando la Figura 2.5 del apartado *2.2.1*, se deduce la siguiente expresión matricial:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sen(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{2.5}$$

Integrando la expresión (2.5) se obtendría tanto la posición (*X*,*Y*) del robot como su orientación (θ). El problema radica en que integrando la citada expresión el cálculo de la posición y la orientación se ralentizaría, ya que la expresión resultante de integrar la (2.5) es algo compleja. Para solventar este problema se puede simplificar la integración de la expresión anterior de tal forma que el cálculo sea mucho más simple y rápido. La expresión resultante quedaría de la siguiente forma:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} \approx \begin{bmatrix} x_k + v_k T_s \cos(\theta_k) \\ y_k + v_k T_s \text{sen}(\theta_k) \\ \theta_k + \omega_k T_s \end{bmatrix} \tag{2.6}$$

En esta última expresión se calcula la estimación de la posición como la orientación del robot para cada instante de muestreo.

Por último, cabe destacar que la simplificación llevada a cabo se puede realizar debido a que el periodo de muestreo, $T_s$, es lo suficientemente pequeño.

#### 2.3.1.2 *Cinemática inversa en el control de trayectorias*

En este apartado se va a tratar de profundizar en la expresiones que surgen de analizar la cinemática inversa cuando el control de posición empleado es el de trayectorias.

Con el control de trayectorias lo que se controla es la curva temporal de posición y la orientación del robot para cada instante de tiempo. En este tipo de control el tiempo es una variable imprescindible, ya que la curva de referencia depende de este. Dicho esto, se puede afirmar que el control está definido por:

$$\{ x(t), y(t), \theta(t) \}$$

Para el control de trayectoria se va a utilizar un algoritmo de control de posición por punto descentralizado. Como se observa en la Figura 2.14, el algoritmo se basa en situar un punto ficticio, llamado punto descentralizado, a una distancia *g* del punto medio del eje de las ruedas de tracción del robot (no hay que olvidar que se trata de una configuración diferencial).

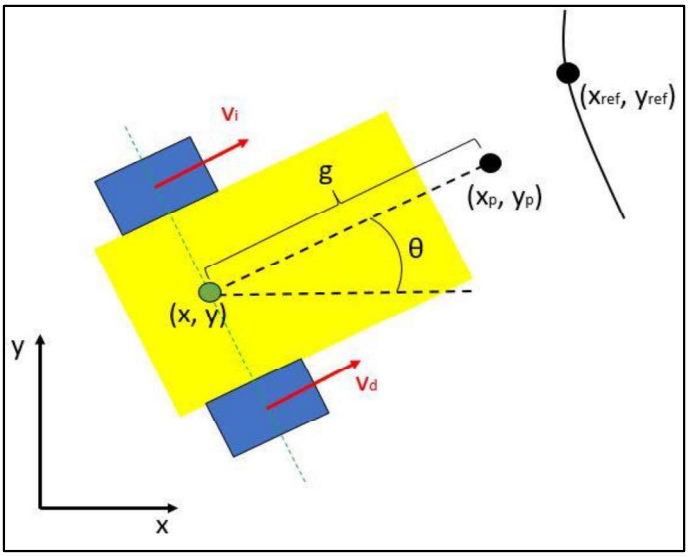


**Fig. 2.14: Esquema del control de posición por punto descentralizado**

La posición del punto descentralizado viene dada por la siguiente expresión:

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

Derivando la expresión anterior se obtiene la expresión (2.8):

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

Sustituyendo la expresión (2.3) en la (2.5) se obtiene:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sen(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2b} & \frac{1}{2b} \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix} \tag{2.9}$$

Y sustituyendo la expresión (2.9) en la (2.8) y operando se obtiene la expresión (2.10):

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \begin{bmatrix} \dot{x} - g \cdot sen(\theta) \cdot \dot{\theta} \\ \dot{y} + g \cdot cos(\theta) \cdot \dot{\theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -g \cdot sen(\theta) \\ 0 & 1 & g \cdot cos(\theta) \end{bmatrix} \begin{bmatrix} cos(\theta) & 0 \\ sen(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ -1/2b & 1/2b \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix}$$

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \frac{1}{2b} \begin{bmatrix} b \cdot cos(\theta) + g \cdot sen(\theta) & b \cdot cos(\theta) - g \cdot sen(\theta) \\ b \cdot sen(\theta) - g \cdot cos(\theta) & b \cdot sen(\theta) + g \cdot cos(\theta) \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix} \tag{2.10}$$

Por último, solamente queda despejar las velocidades lineales de las ruedas de la expresión (2.10) para así establecer las velocidades de referencia que han de tener las ruedas.

$$\begin{bmatrix} v_{i,ref} \\ v_{d.ref} \end{bmatrix} = \frac{1}{g} \begin{bmatrix} g \cdot cos(\theta) + b \cdot sen(\theta) & g \cdot sen(\theta) \ - \ b \cdot cos(\theta) \\ g \cdot cos(\theta) \ - \ b \cdot sen(\theta) & g \cdot sen(\theta) + b \cdot cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} \tag{2.11}$$

Con la expresión (2.11) se pueden obtener las velocidades lineales de referencia de las ruedas del robot, únicamente es necesario conocer $\dot{x}_p$ e $\dot{y}_p$, y para ello se emplea la ley de control propuesta por Siegwart y Nourbakhsh (2004):

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \begin{bmatrix} k_{rv} \cdot \dot{x}_{ref} \\ k_{rv} \cdot \dot{y}_{ref} \end{bmatrix} + \begin{bmatrix} k_{rp} & 0 \\ 0 & k_{rp} \end{bmatrix} \begin{bmatrix} x_{ref} - (x + g \cdot cos(\theta)) \\ y_{ref} - (y + g \cdot sen(\theta)) \end{bmatrix} \quad (2.12)$$

Finalmente, solo quedaría calcular las acciones de control que se aplican a los motores de las ruedas. Este cálculo se realiza en el bloque de control dinámico que se abordará más adelante.

### *2.3.1.3 Cinemática inversa en el control de caminos*

En el apartado anterior se ha tratado de profundizar en la cinemática inversa con control de trayectorias. Ahora se va a abordar de forma idéntica la cinemática inversa con control de caminos.

En el control de caminos la variable tiempo no influye en nada, ya que la curva de referencia no depende del mismo. Lo que se trata de controlar es la posición y orientación del robot en el espacio cartesiano. Así el control queda definido por los parámetros:

$$\{ x, y, \theta \}$$

Igual que en el control de trayectorias, en el control de caminos se va a utilizar un algoritmo de control de posición específico llamado persecución pura. Como se observa en la Figura 2.15, el algoritmo se basa en calcular geométricamente la curva que ha de seguir el robot para que vaya desde el punto donde se encuentra, ($x_r$, $y_r$), hasta un punto objetivo de la trayectoria de referencia, ($x_{ob}$, $y_{ob}$).

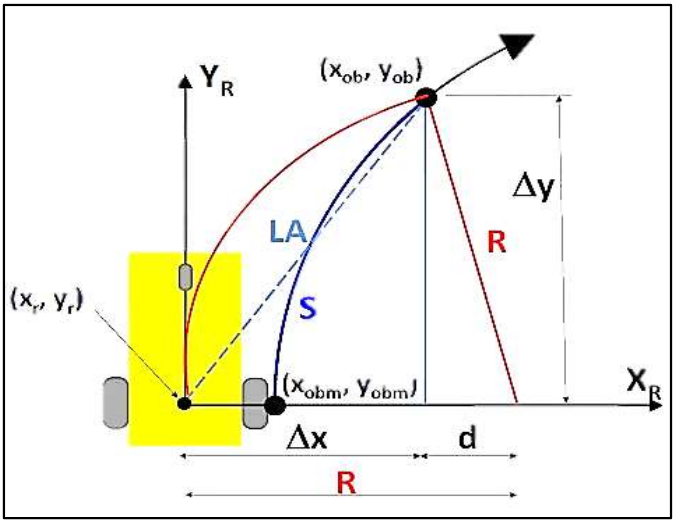


**Fig. 2.15: Esquema de control de posición por persecución pura [13]**

La distancia *LA,* llamada *Look Ahead,* es la distancia que hay hasta el punto objetivo. Además de la distancia *LA* se tiene que definir la velocidad de avance, *V,* con la que el robot avanza sobre la trayectoria.

Comentar también que, así como se fija la distancia *Look Ahead*, también se fija la velocidad de avance, *V,* con la que el robot avanza sobre la trayectoria.

Dicho esto, se va a profundizar en qué expresiones son necesarias para obtener las velocidades de referencia de cada una de las ruedas del robot, que al fin y al cabo, este el objetivo de la cinemática inversa.

Con el algoritmo de la persecución pura, las velocidades de las ruedas se obtienen aplicando una serie de expresiones geométricas y otras cinemáticas. Empezando por las expresiones geométricas, y observando la Figura 2.15, se deducen:

$$LA^2 = \Delta x^2 + \Delta y^2 \tag{2.13}$$

$$d = R - \Delta x \tag{2.14}$$

$$R^2 = d^2 + \Delta y^2 \tag{2.15}$$

$$\Delta x = (y_{ob} - y_r)\cos(\theta) - (x_{ob} - x_r)sen(\theta) \tag{2.16}$$

Sustituyendo (2.15) en (2.14) y desarrollando se obtiene:

$$R^2 = (R - \Delta x)^2 + \Delta y^2$$
$$R^2 = R^2 - 2R\Delta x + \Delta x^2 + \Delta y^2$$
$$2R\Delta x = \Delta x^2 + \Delta y^2 \tag{2.17}$$

Sustituyendo ahora (2.13) en (2.17):

$$2R\Delta x = LA^2$$
$$R = \frac{LA^2}{2\Delta x} \tag{2.18}$$

De esta manera, la curvatura, γ, se calcula como:

$$\gamma = \frac{1}{R} = \frac{2\Delta x}{LA^2} \tag{2.19}$$

Donde *Δx* se calcula mediante la expresión (2.16) y *LA* se mantiene constante.

Finalmente, solo queda obtener las expresiones que den como resultado las velocidades lineales de las ruedas. Para ello, aplicando las expresiones cinemáticas (2.20) y (2.21), sustituyendo en ellas la (2.19) y desarrollando, se pueden calcular para cada momento las velocidades lineales de las ruedas izquierda, expresión (2.22) y derecha, expresión (2.23).

$$v_{i,ref} = V - b\omega = V - b \cdot \frac{V}{R} \tag{2.20}$$

$$v_{d,ref} = V + b\omega = V + b \cdot \frac{V}{R} \tag{2.21}$$

$$v_{i,ref} = V(1 - b \cdot \gamma) \tag{2.22}$$

$$v_{d,ref} = V(1 + b \cdot \gamma) \tag{2.23}$$

### 2.3.2 Control dinámico

Para finalizar la parte del control de los robots, se desarrollará el bloque del control dinámico. Con este bloque lo que se pretende es calcular las acciones de control a aplicar en los motores acoplados a las ruedas del robot para que este siga la trayectoria fijada.

Para alcanzar el objetivo, en el control dinámico se utiliza un controlador, que puede ser de tipo proporcional (P), proporcional-derivativo (PD) o proporcional-derivativo-integrativo (PID).

En este trabajo se emplea el controlador proporcional (P) siguiente:

$$acontrol_x = k_{mp}(\omega_{x,ref} - \omega_{x,rob}) \qquad x = izda, dcha \tag{2.24}$$

La acción de control de cada rueda se calcula multiplicando el error de las velocidades angulares de las propias ruedas por la constante proporcional $K_{mp}$.

## 2.4 Generación de trayectorias de referencia

Por lo dicho hasta el momento, se puede intuir que para controlar un robot móvil resulta imprescindible comunicarle a éste la trayectoria que ha de realizar entre el punto donde se encuentra y el que tiene que alcanzar. Por ello, previamente a la puesta en movimiento del robot, resulta necesario generar una trayectoria de referencia.

Para la generación de trayectorias entre dos puntos dados son utilizados, principalmente, dos métodos: la interpolación y las curvas de aproximación. A pesar de que ambos métodos tienen una fácil implementación utilizando programas de cálculo (como Matlab), en el presente trabajo se empleará el método de las curvas de aproximación.

Con las curvas de aproximación lo que se pretende es que, dados los puntos inicial y final de la trayectoria, y otros puntos auxiliares, se genere un polígono de control al que la trayectoria generada se aproximará. Los únicos puntos por los que seguro que pasará la trayectoria generada serán el de salida y el de llegada, por el resto de puntos, puntos auxiliares, la curva se aproximará sin llegar a pasar por ellos. La expresión de estas curvas es del tipo :

$$r(t) = \sum_{i=0}^{n} P_i \cdot f_i(t)\,, \qquad t \in [0,1] \qquad (2.24)$$

Donde $P_i$ son los distintos puntos del polígono de control y $f_i$ son las llamadas funciones de ponderación.

### 2.4.1 Curvas de Bézier

En el año 1962, el ingeniero francés Pierre Bézier publicó por primera vez las curvas que llevan su nombre. Estas curvas fueron ampliamente utilizadas por la compañía Renault para el diseño de las carrocerías de sus automóviles y, posteriormente, para generar trazos en programas de diseño gráfico. **[3]**

Fundamentalmente, se trata de un tipo de curvas de aproximación en las que la función de ponderación, $f_i(t)$, pasa a ser una función conocida como polinomios de Bernstein, $B_{i,n}(t)$. Estos polinomios presentan la siguiente forma:

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}\,, \qquad t \in [0,1] \tag{2.25}$$

Visto esto, se va a obtener la expresión analítica de una curva de Bézier cúbica, es decir, se emplean 4 puntos para su generación. Se obtiene esta curva y no otra debido a que en el trabajo se emplea dicha curva.

En primer lugar, hay que desarrollar los polinomios de Bernstein para *n=3 :*

$$B_{0,3}(t) = \binom{3}{0} t^0 (1-t)^{3-0} = \frac{3!}{0!\,(3-0)!} t^0 (1-t)^3 = (1-t)^3\,, \qquad t \in [0,1]$$

$$B_{1,3}(t) = \binom{3}{1} t^1 (1-t)^{3-1} = \frac{3!}{1!\,(3-1)!} t^1 (1-t)^2 = 3t(1-t)^2\,, \qquad t \in [0,1]$$

$$B_{2,3}(t) = \binom{3}{2} t^2 (1-t)^{3-2} = \frac{3!}{2!\,(3-2)!} t^2 (1-t)^1 = 3t^2(1-t)\,, \qquad t \in [0,1]$$

$$B_{3,3}(t) = \binom{3}{3} t^3 (1-t)^{3-3} = \frac{3!}{3!\,(3-3)!} t^3 (1-t)^0 = t^3\,, \qquad t \in [0,1]$$

Una vez obtenidas las funciones de ponderación necesarias para calcular la curva, se procede a obtener la función de la curva en función de *t,* donde *t* ∈ [0,1].

$$r(t) = \sum_{i=0}^{3} P_i B_{i,3}(t) = P_0(1-t)^3 + P_1 3t(1-t)^2 + P_2 3t^2(1-t) + P_3 t^3 \tag{2.26}$$

Si se desarrolla la expresión (2.26), se puede llegar a una formulación matricial de la función de la curva. Esto es muy interesante poder alcanzarlo, ya que se va a trabajar con Matlab para generar dichas curvas, y en este programa se trabaja de una manera muy cómoda con matrices. Por tanto, desarrollando la expresión (2.26), se puede expresar *r(t)* de forma matricial, expresión (2.27).

$$r(t) = P_0(-t^3 + 3t^2 - 3t + 1) + P_1(3t - 6t^2 + 3t^3) + P_2(3t^2 - 3t^3) + P_3 t^3$$

$$r(t) = [P_0 \quad P_1 \quad P_2 \quad P_3] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}, \qquad t \in [0,1] \quad (2.27)$$

## 2.5 Conexiones inalámbricas

Actualmente hay un amplio interés en el campo de las conexiones inalámbricas. Este interés proviene de que en este tipo de conexiones, como su propio nombre indica, el uso de cables para la comunicación entre máquinas es nulo, es decir, para conectar dos o más aparatos entre sí no es necesario emplear un cable que vaya de un aparato a otro.

Este hecho permite la ejecución de muchas aplicaciones que con una conexión por cable sería imposible. Un ejemplo de ello podría ser el manejo de un robot móvil mediante el uso de un joystick o, como se va a tratar en este trabajo, el envío de datos entre el robot y el ordenador desde donde se realiza el control.

Dentro del campo de las conexiones inalámbricas hay varios tipos: conexión Bluetooth, ZigBee, WiFi, etc.

Como el robot móvil empleado en el trabajo, el Lego Mindstorms EV3, tiene la capacidad de conexión vía Bluetooth y vía WiFi, en este apartado nos centraremos en sintetizar un poco en qué consisten estos dos tipos de conexión y no lo haremos con los demás sistemas citados anteriormente.

### *2.5.1 Conexión Bluetooth*

Este tipo de conexión pertenece a las *Redes Inalámbricas de Área Personal (WPAN)*, o lo que es lo mismo, es un tipo de conexión que únicamente permite establecer conexiones entre dispositivos que estén cerca. Para transmitir datos entre dos o más dispositivos, se emplean ondas de radio de la banda ISM a 2.4 GHz. **[4]**

Por último, decir que esta tecnología presenta un inconveniente importante si se quieren realizar trabajos con comunicaciones en largas distancias, ya que el alcance de la tecnología Bluetooth es bastante corto, estando su máximo entorno a los 100 metros.

### 2.5.2 Conexión WiFi

La tecnología WiFi permite la conexión inalámbrica entre diferentes dispositivos siempre y cuando estén conectados a la misma red.

Para que los diferentes dispositivos se puedan conectar entres sí, se ha dicho que deben estar conectados a una misma red, por lo que deberá haber algún aparato que sirva como punto de acceso a la citada red. De entre los distintos aparatos existentes, en este trabajo se empleará como punto de acceso el router inalámbrico.

Por otro lado, para que los dispositivos se puedan conectar a la red tendrán que disponer de algún dispositivo diseñado para tal causa. En este trabajo se emplearán las tarjetas USB, también llamadas adaptadores, para que el robot pueda conectarse a la red.

Para finalizar, comentar que la principal ventaja de la tecnología WiFi es que su radio de alcance, para una misma fiabilidad de conexión, es mucho más elevado que el de la tecnología Bluetooth.

## 2.6 Protocolos de comunicación

La conexión entre dos dispositivos se establece con el fin de transmitir datos de uno a otro. Esta transmisión no sería posible si no se establece antes un protocolo de transporte. Dos de los protocolos más destacados son : protocolo TCP y protocolo UDP.

- *Protocolo UDP (User Datagram Protocol):* está basado en el intercambio de datagramas, permitiendo el envío de estos sin necesidad de que se haya establecido una conexión previa entre dispositivos. Este hecho, unido a que no se controla el flujo de datos transmitidos y a que ni siquiera se garantiza que los datos transmitidos lleguen en el mismo orden en el que han sido enviados, hace que no sea un protocolo fiable para la transmisión de datos.
- *Protocolo TCP (Transmission Control Protocol):* proporciona un transporte fiable de flujo de bits entre aplicaciones o dispositivos. **[5]**
  Con este protocolo se garantiza que el flujo de datos se envíe correctamente. No obstante, el precio a pagar por esta fiabilidad es el tener una menor eficiencia con respecto al protocolo UDP.

### 2.6.1 Sockets

En el presente trabajo se trabajará con aplicaciones distribuidas, es decir, con aplicaciones que se ejecutan al mismo tiempo en distintos dispositivos que están conectados a la misma red. En este caso, de los distintos tipos de modelos de aplicaciones distribuidas existentes, se va a emplear el modelo cliente-servidor, es decir, uno de los dispositivos o aplicación trabajará como cliente y el otro/a como servidor.

Dicho esto, se puede decir que el cliente es el encargado de solicitar la conexión, mediante un puerto de conexión que se encuentre libre, al servidor, y este, a su vez, es el encargado de aprobar la citada conexión. Una vez aprobada la conexión, se puede empezar a transmitir datos desde el cliente al servidor o del servidor al cliente a través de un socket.

Resumiendo, un socket sirve como mecanismo de comunicación bidireccional entre el cliente y el servidor de una comunicación, sea por el protocolo que sea.

## 2.7 Lego Mindstroms EV3

La compañía de juguetes danesa, Lego, lleva ofreciendo al mercado desde 1998 un tipo de juguete diferente a aquellos por los que la compañía es mundialmente conocida. Estos juguetes son los Lego Mindstorms, que son, nada más y nada menos, juguetes que pueden ser programados para desarrollar sistemas autómatas.

Los Lego Mindstorms van equipados con actuadores, sensores, piezas para el montaje, y lo más importante, un ladrillo, también llamado brick, inteligente programable donde se encuentra el procesador que genera las respuestas a las órdenes que se le determinan.

Debido a que el precio de estos juguetes es bastante elevado para ser tratado como un mero juguete, y que a la vez es económico para ser usado como prototipos en centros de enseñanza o de desarrollo tecnológico, estos dispositivos han pasado a ser utilizados en el campo de la educación, donde sirven para introducir a los alumnos en el amplio mundo de la robótica y la automática, o de la investigación, donde son usados para el desarrollo y testeo de ciertas aplicaciones a pequeña escala que después puedan ser extrapoladas a otros tamaños de mayor dimensión.

La primera generación de estos juguetes fue lanzada en 1998 bajo el nombre Lego Mindstorms RCX. Posteriormente, en 2006, se lanzó una nueva generación con el nombre Lego Mindstorms NXT. Esta segunda generación ya poseía capacidad para comunicarse vía Bluetooth con otros dispositivos. La última generación fue lanzada en 2013 con el nombre Lego Mindstorms EV3 (“EV” proviene del inglés, evolution, y el “3” significa que es la tercera generación). Está última generación presenta una amplia mejora en cuanto a procesador, memoria interna y conexiones con otros dispositivos se refiere.


Fig. 2.18: Ladrillo inteligente RCX [12]


Fig. 2.17: Ladrillo inteligente NXT [13]


Fig. 2.16: Ladrillo inteligente EV3 [14]

A continuación se van a comentar las principales especificaciones que presenta el ladrillo inteligente programable EV3 así como el funcionamiento de los actuadores. Acerca de los sensores, simplemente comentar que existen diferentes tipos, de distancia, de contacto, de color, etc., y que no se va a desarrollar más acerca de ellos porque en el presente trabajo no van a ser necesarios.

### 2.7.1 Ladrillo EV3

Como ya se ha comentado, el ladrillo inteligente es la parte más importante, puesto es la que contiene el procesador que ejecuta las órdenes que le llegan.

Las principales especificaciones de este son:

- 4 puertos de salida para la conexión de motores.
- 4 puertos de entrada para la conexión de sensores.
- 1 puerto USB host para el conector WiFi.
- 1 puerto USB para la conexión mediante cable al ordenador.
- 1 puerto para tarjeta microSD.
- Receptores de señales infrarrojas, bluetooth y WiFi.
- Pantalla LCD monocromática
- Altavoz integrado
- Procesador ARM9 de 32 bits, 16 MB de memoria flash, una memoria RAM de 64 MB y una frecuencia de 300 GHz.

Como se ha dicho anteriormente, con los Lego Mindstorms NXT ya se tiene la capacidad de comunicación inalámbrica entre dispositivos mediante Bluetooth, pero, con la actual generación, los Lego Mindstorms EV3, además de ofrecer el citado tipo de comunicación, también la ofrece vía WiFi.**[6]**

Esto último, junto con la mejora en la capacidad de procesamiento, son las ventajas a destacar que presentan los EV3 respecto a las dos generaciones predecesoras, y por ello, en este trabajo los robots empleados están constituidos a partir del pack Lego Mindstorms EV3.

### 2.7.2 Actuadores para Lego Mindstorms EV3

Los actuadores empleados en el trabajo son únicamente dos servo-motores conectados a las ruedas para así permitir el movimiento del robot (recordar que el robot móvil empleado presenta una configuración diferencial). Estos servo-motores presentan un par motor de 20 N·cm y una velocidad angular máxima de 170 rpm. Cabe decir que, igual que en un automóvil el eje del motor no se conecta directamente al eje de las ruedas, sino que hay una caja de cambios que ajusta el par entregado a las ruedas, en el servo-motor de Lego sucede lo mismo, existe un juego de engranajes que permiten que el motor realice sobreesfuerzos y el par motor que entregue sea lo suficientemente grande para que el robot se mueva.

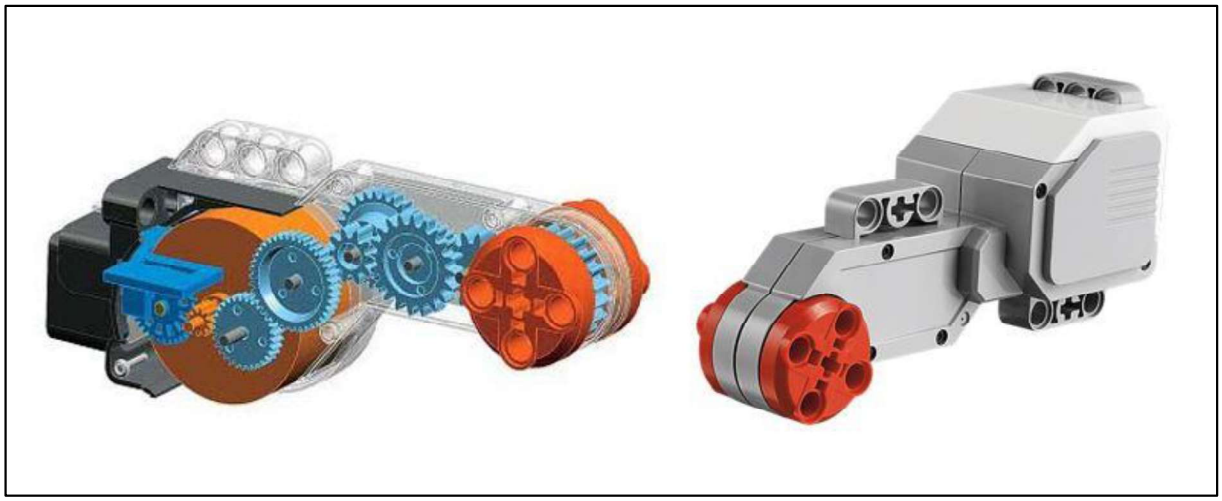

**Fig. 2.19: Vistas interior (izquierda) y exterior (derecha) de un servo-motor de Lego [15]**

Debido a que el ladrillo EV3 es un dispositivo digital, el control de la velocidad del motor se realiza mediante la Modulación de Anchura de Pulsos (PWM), es decir, el motor es alimentado únicamente con pulsos de 9 V durante un cierto intervalo de tiempo según la potencia necesaria para alcanzar la velocidad requerida. A modo ilustrativo, se pueden observar las figura 2.20.

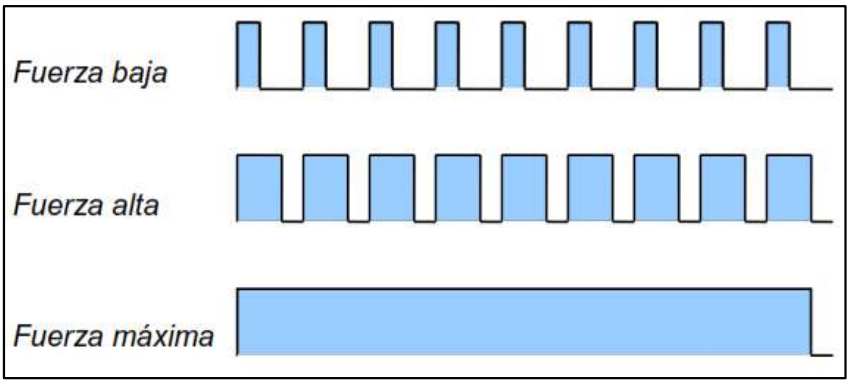


**Fig. 2.20: Señal PWM para el funcionamiento de motores DC [16]**

Con esta figura, se puede pensar que cuando la fuerza no es máxima, y como consecuencia la velocidad tampoco lo será, el robot no se mueve de forma continua. No obstante, Lego resolvió esto instalando un volante de inercia en el eje del motor que almacena la energía que hace que el motor siga girando a pesar de que la tensión de entrada al motor sea nula.

Por otro lado, aunque se ha dicho que no se hablaría de sensores, cabe destacar que los servo-motores llevan incorporados unos sensores, llamados encoders, que como ya se ha podido ver, son de muchísima utilidad, ya que sin ellos el control de posición del robot sería una tarea mucho más compleja.

Los encoders dan como señal de salida la rotación de la rueda. Presentan una precisión de 1 grado, luego, si la rueda gira una vuelta completa, la señal de salida del encoder será 360 grados.

# 3 DESARROLLO PRÁCTICO

Una vez introducida toda la parte teórica que será de utilidad para la realización del trabajo, se pasa ahora a desarrollar las tareas realizadas para el presente trabajo.

## 3.1 Montaje de los robots móviles e instalación de Matlab/Simulink

### *3.1.1 Estructura robot móvil Lego Mindstorms EV3*

Como ya se ha dicho en la parte teórica, la configuración escogida para el robot móvil empleado en el trabajo fue la diferencial, puesto que el montaje se lleva a cabo de una forma muy sencilla y el control de posición a emplear es relativamente simple de programar. El robot utilizado presenta una estructura como la de la Figura 3.1.

**Fig. 3.1: Configuración del robot empleado.**

### *3.1.2 Instalación de Matlab/Simulink*

Como se puede deducir, resultaba necesaria la instalación de Matlab y Simulink para la realización del trabajo, por lo que en primer lugar se procedió a descargar e instalar dichos programas.

Una vez instalados los programas necesarios para realizar el trabajo, era necesario configurar Simulink para poder desarrollar los programas necesarios e implementarlos posteriormente en el robot móvil Lego Mindstorms EV3.

Lo único que se tenía que hacer para configurar Simulink era instalar en esta aplicación una librería de soporte para Lego Mindstorms EV3. Para descargar la citada librería, simplemente hubo que abrir Matlab y clicar en “Add-Ons”. Dentro de “Add-ons” se seleccionó “Get Add-Ons”, ver Figura 3.2, y en la pantalla emergente que apareció, se buscó la citada librería y se procedió a descargarla. Al tener ya instalados Matlab y Simulink en el ordenador, la instalación de la librería se realizó de forma automática una vez hubo terminado su descarga.

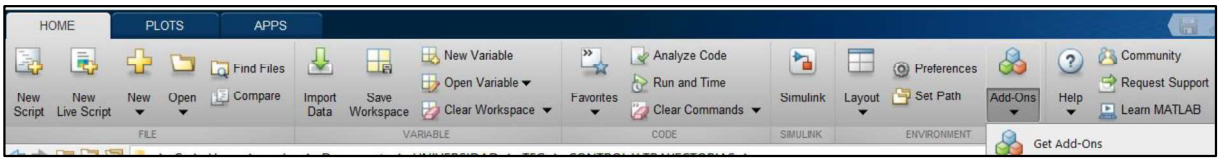


**Fig. 3.2: Captura de la localización de "Add-ons" en barra de herramientas de Matlab**

Finalmente, con el fin de que una vez que vayamos a ejecutar los esquemas de Simulink creados, estos se vean en funcionamiento sobre el robot, hay que comunicarle de alguna manera a Simulink que va a ejecutar, de forma externa, sobre el Lego Mindstorms EV3 los esquemas. Para ello simplemente hay que entrar en el configurador de parámetros de Simulink y establecer dentro del campo "Hardware board", que aparece dentro de la pestaña "Hardware Implementation", que el hardware que va a ser empleado es el Lego Mindstorms EV3.

### *3.1.3 Verificación de la conexión PC-Robot*

El sistema empleado para la conexión entre el ordenador y el robot fue la conexión WiFi de la que se ha hablado en el desarrollo teórico. Para establecer la comunicación simplemente hubo que conectar el ordenador y el robot a la misma red WiFi.

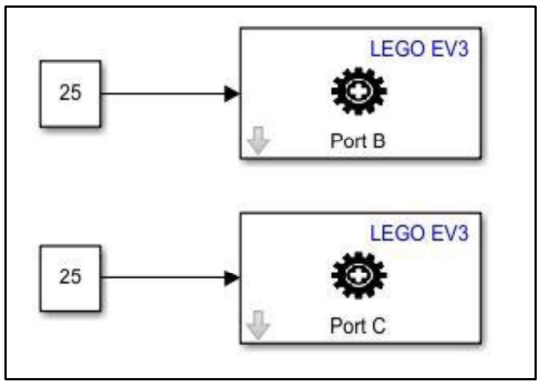


**Fig. 3.3: Esquema de prueba.**

Para empezar, en una sesión de Simulink se creó un sencillo esquema que hiciese funcionar los motores del robot, Figura 3.3. Comentar que con el esquema de la Figura 3.3 se pretende que ambos motores funcionen al 25% de su potencia.

Posteriormente, se intentó ejecutar el esquema creado para esta tarea. Para ello, se probaron diferentes adaptadores WiFi, puesto que el Lego Mindstorms EV3 necesita de este dispositivo para establecer la comunicación vía WiFi. Los modelos de estos adaptadores eran: Netgear N150, Edimax EW7611 y TL-WN725N de TP-Link.

Al inicio del trabajo fue utilizado el router CB-300RS4 de la marca Conceptronic para establecer la conexión entre el PC y el robot, pero, como se observa en la tabla 1, con este router no se consiguió establecer una comunicación robusta con ninguno de los adaptadores WiFi citados anteriormente, puesto que frecuentemente aparecían errores de conexión.

Como con el router de Conceptronic no se podía establecer una conexión robusta, se optó por utilizar el router TP-Link-Pocket-MR3020 de la marca TP-Link. Al intentar ejecutar el esquema

de la Figura 3.3 se volvieron a tener fallos de conexión. No obstante, esta vez sucedió únicamente con uno de los dos robots empleados en el trabajo, mientras que con el otro robot la comunicación no presentaba ningún fallo, era robusta.

Visto que uno de los robots se conectaba perfectamente al ordenador, se descartó la posibilidad de que el error de conexión se debiera de nuevo al router empleado, así que se planteó que esta vez el problema radicaba en el robot. Dicho esto, como el software que ejecuta el procesador del robot es el firmware del mismo, se procedió a averiguar la versión de firmware[1] que presentaba cada robot para que, de esta manera, se le pudiese instalar al robot que daba problemas la versión de aquel que se conectaba correctamente. El robot que se conectaba correctamente presentaba la versión de firmware 1.04H, mientras que la que daba error presentaba la 1.08H.

| | | ADAPTADORES WiFi | | |
|---|---|---|---|---|
| | | Netgear N150 | Edimax | Tp-link |
| Router CB-300RS4 | Ver. Firmw. 1.04H | FRECUENTES | FRECUENTES | SIEMPRE |
| | Ver. Firmw. 1.08H | SIEMPRE | SIEMPRE | SIEMPRE |
| Router TP-Link-Pocket | Ver. Firmw. 1.04H | NUNCA | NUNCA | NUNCA |
| | Ver. Firmw. 1.08H | SIEMPRE | SIEMPRE | SIEMPRE |

**Tabla 3.1: Frecuencia de aparición de errores durante la comunicación PC-Robot**

Una vez que se tenía la certeza de que los fallos de conexión no se iban a producir, se empezaron a desarrollar las diferentes tareas con el fin de conseguir el objetivo del trabajo.

[1] Para averiguar la versión del firmware del ladrillo, simplemente hay que navegar por el propio menú que presenta este. En la opción *Brick Information* aparece la versión del firmware instalada en el ladrillo.

## 3.2 Programación de funciones para la generación de trayectorias.

La generación de trayectorias de referencia es una parte imprescindible en el presente trabajo, ya que los robots móviles han de tener una referencia en la que fijarse para poder realizar con sentido el movimiento entre el punto donde se encuentran y el punto final al que han de llegar.

A partir de dos puntos, se pueden generar muchas trayectorias diferentes. En este trabajo se empleó un método de generación de curvas por aproximación a puntos desarrollado por Pierre Bézier, las llamadas curvas de Bézier. Aprovechando las expresión matricial (2.27) de las curvas de Bézier cúbicas desarrolladas en la parte teórica, se desarrolló una función en Matlab para que, dados cuatro puntos, un tiempo de duración de la trayectoria y un intervalo de tiempo entre puntos de la trayectoria, se generase una curva de Bézier.

```
1  function [xref,x_b,yref,y_b]=bezier4puntos(p0x,p0y,p1x,p1y,p2x,p2y,p3x,p3y,t_fin,Ts)
2  %Matriz de puntos:
3  P=[p0x p1x p2x p3x;p0y p1y p2y p3y];
4  %Matriz de coeficientes:
5  C=[1 -3 3 -1;0 3 -6 3;0 0 3 -3;0 0 0 1];
6  %Producto Matrices de puntos y coeficientes:
7  Maux=P*C;
8  %Generación de la curva:
9  i=1;
10 tinicial=0;
11 tfinal=t_fin;
12 taux=tinicial;
13 while taux<=tfinal
14     t=(taux-tinicial)/(tfinal-tinicial);
15     taux=taux+Ts;     %Se incrementa la variable taux en 0.02
16     T = [1;t;t^2;t^3];    %Vector con valores de t
17     Vptos= Maux*T;    %Vector con las coordenadas
18     x_b(i) = Vptos(1);
19     y_b(i) = Vptos(2);
20     i=i+1;
21 end
22 t=0:Ts:tfinal;
23 xref=[t' x_b'];
24 yref=[t' y_b'];
25 plot(x_b,y_b,p0x,p0y,'r+',p1x,p1y,'b+',p2x,p2y,'c+',p3x,p3y,'g+');
26 end
```

**Fig. 3.4: Captura código empleado para generar la curva de Bézier a partir de 4 puntos**

Además, como se puede observar en la Figura 3.4, al finalizar el bucle *while,* la función genera también las referencias que serán utilizadas más adelante en los diferentes esquemas de Simulink.

Un ejemplo de uso de esta función sería:

```
[xref,x_b,yref,y_b]= bezier4puntos(0,0,0.3,0.2,0.6,0,1,0.5,20,0.02);
```

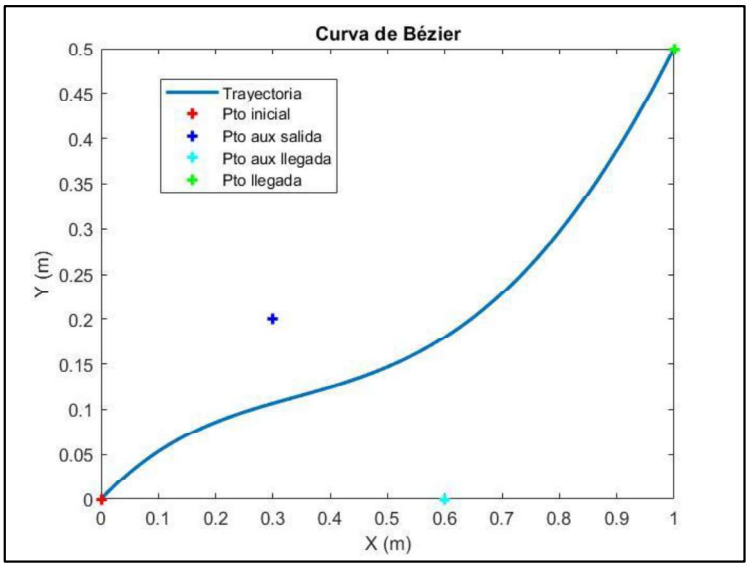


**Fig. 3.5: Curva de Bézier generada con la función *bezier4puntos*.**

No obstante, está última función tenía la limitación de que los puntos auxiliares, *P1* y *P2*, empleados, no se calculaban de forma automática, sino que había que introducirlos de forma manual.

Esto suponía una limitación en cuanto a que si se deseaba que el robot partiese con una determinada orientación desde el punto de partida, y llegase al punto de llegada con otra orientación deseada, los puntos auxiliares debían ser aquellos que, al generarse el polígono de control, la curva fuese tangente a dicho polígono tanto en el punto de llegada como en el de salida.

Para conseguir que el robot saliese del punto de salida y llegase al de llegada con la orientación deseada, se creó otra función con Matlab a la que se le han de pasar como parámetros los puntos final e inicial, la orientación de salida y llegada, el tiempo de duración de la trayectoria y el intervalo de tiempo entre puntos de la trayectoria. Esta función presentaba la siguiente estructura:

```
1   function [xref,x_b,yref,y_b]=bez_2ptos_orientacion(p0_x,p0_y,p3_x,p3_y,theta0,theta1,t_fin,Ts)
2   %% Cálculo de la distancia a la que estarán los ptos intermedios de la curva:
3   dx = abs(p0_x-p3_x);
4   dy = abs(p0_y-p3_y);
5   dis=sqrt(dx^2+dy^2);    % Distancia entre puntos
6   l_p1 =dis/3;
7   l_p2=dis/2;
8   %% Cálculo coordenadas ptos intermedios, P1 y P2:
9   % Primer punto intermedio, depende de P0 y theta0:
10  [p1_x,p1_y]=dist_pto_avanzado(p0_x,p0_y,theta0,l_p1);
11  % Segundo pto intermedio (depende de p3 y theta1):
12  p2_x = p3_x - (l_p2*cos(theta1));
13  p2_y = p3_y - (l_p2*sin(theta1));
14  %% Cálculo de la curva de Bézier:
15  [xref,x_b,yref,y_b]=bezier4puntos(p0_x,p0_y,p1_x,p1_y,p2_x,p2_y,p3_x,p3_y,t_fin,Ts);
16  end
```

**Fig. 3.6: Captura código empleado en el cálculo de la curva de Bézier con orientaciones conocidas**

En la línea 10 del código presentado en la Figura 3.6, se calculan las coordenadas del punto auxiliar de salida, *P1*, mediante una función, *dist_pto_avanzado*, creada para tal efecto. Con la citada función, se calcula la posición de *P1* teniendo en cuenta que este punto ha de estar orientado según la orientación de salida que se adopte y que puede estar en cualquiera de los cuatro cuadrantes.

Por otro lado, en las líneas 12 y 13, se calculan las coordenadas del punto auxiliar de llegada. Hay que aclarar que para este punto no se ha utilizado la función *dist_pto_avanzado* debido a que en este caso el punto auxiliar de llegada está retrasado con respecto al punto final, y con el código escrito en las líneas 12 y 13 es suficiente para el correcto cálculo del citado punto.

Un ejemplo de uso de esta función puede ser el siguiente:

```
[xref,x_b,yref,y_b]=bez_2ptos_orientacion(-0.1,-0.4,0.6,0.5,0,pi/2,20,0.02)
```

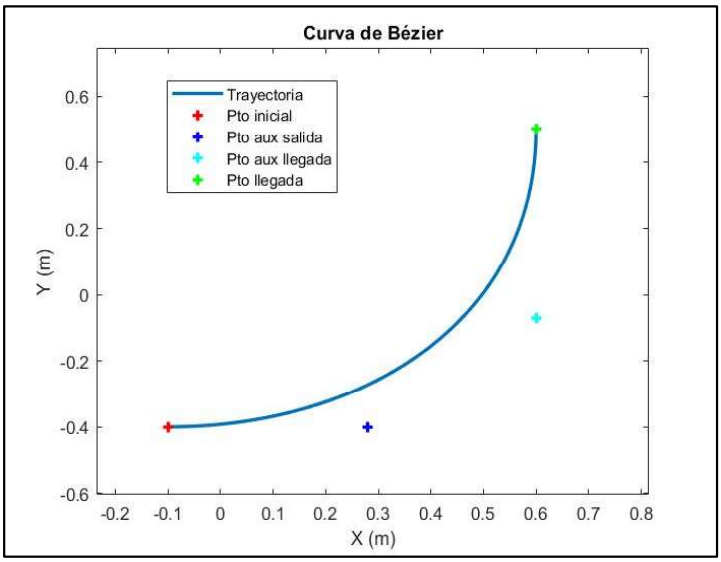


**Fig. 3.7: Curva de Bézier generada con la función *bez_2ptos_orientacion***

Como se puede observar en esta última gráfica, se parte desde el punto (-0.1,-0.4) con una orientación de 0 radianes y se llega al punto (0.6,0.5) con una orientación de pi/2 radianes. Esta función fue muy útil posteriormente cuando se quiso que el robot fuese de una posición a otra dada, y era necesario respetar tanto la orientación de salida, aunque esta no era tan importante, como la de llegada, esta sí que se debía respetar lo máximo posible.

## 3.3 Comunicación de la posición y orientación mediante el bloque TCP/IP

### *3.3.1 Generación de sockets en Matlab/Simulink*

Para la realización del trabajo, eran necesarios una serie de sockets para establecer la comunicación entre el ordenador, o la sesión de Matlab, y el robot, u otro dispositivo. El protocolo de comunicación que se empleó fue el TCP, ya que como se ha visto en el desarrollo teórico, es mucho más fiable que el UDP.

Para la creación de estos sockets, tan solo fue necesario emplear la siguiente función que se encuentra disponible en Matlab:

```
tcp = tcpip('direccion_ip',puerto,'NetworkRole','tipo')
```

Donde los parámetros a introducir son:

- *dirección_ip*: Se debe introducir la dirección IP del host, es decir, del aparato que actúe como servidor.
- *puerto*: Hay que introducir el número del puerto por el que se va a establecer la comunicación
- *tipo*: Se refiere al rol que ejerce el ordenador en la comunicación. En este campo solo se puede escribir 'server', en caso de que el ordenador funcione como servidor, o 'client', en el caso de que el ordenador haga de cliente.

Como resultado, se obtuvo un objeto, que en este caso se llama *tcp*.

Una vez creado el objeto, ya se podía establecer la comunicación. Para ello, simplemente había que abrir el objeto para conectarlo con el otro dispositivo. Cabe decir que esto se ha de realizar independientemente de si el ordenador trabaja como servidor o lo hace como cliente. La función empleada para ello fue:

```
fopen(tcp)
```

Para finalizar la comunicación, hay que emplear otra función que cierra el objeto creado para la comunicación. La función es la siguiente:

```
fclose(tcp)
```

#### *3.3.1.1 Funciones de lectura y escritura*

Una vez establecida la comunicación entre la sesión de Matlab y otro dispositivo, se podía empezar a enviar y/o recibir datos.

Las funciones disponibles para el envío de datos son:

```
fwrite(tcp,dato_a_enviar)
```

```
fpritnf(tcp,'dato_a_enviar')
```

La diferencia entre ambas expresiones radica en que con *fwrite* se envían números, sean del tipo que sean, y con *fprintf* se pueden enviar caracteres o cadenas de caracteres.

Por otro lado, las funciones disponibles para recibir datos son:

```
datos = fread (tcp, número_de_datos_a_recibir)
datos = fscanf(tcp, número_de_datos_a_recibir)
```

Con estas dos funciones sucede lo mismo que con las de envío de datos, la *fread* sirve para el recibimiento de números y la *fscanf* para el de caracteres. Además, en el segundo parámetro de la función, *número_de_datos_a_recibir*, se pueden especificar la cantidad de datos que se quieren recibir y almacenar en la variable *datos,* que cuando se recibe más de un dato es un vector. Hay que tener presente que, si ese parámetro vale, por ejemplo, 2, se recibirían únicamente los dos primeros datos que el otro dispositivo le envíe al ordenador. Si por ejemplo el ordenador necesita trabajar con el tercer y cuarto dato que se le envían, el parámetro *número_de_datos_a_recibir* deberá valer como mínimo 4.

#### 3.3.1.2 Bloques TCP/IP para Lego Mindstorms EV3

Como se ha dicho, los sockets se crean con el fin de establecer una comunicación entre el ordenador y otro dispositivo, sea cual sea este último. Si el dispositivo que con el que el que se desea establecer la comunicación es el robot empleado en el trabajo, el Lego Mindstorms EV3, aparte de crear el correspondiente socket en un script de Matlab, también hay que utilizar uno de los bloques de la librería de soporte de Simulink para Lego Mindstorms EV3.

En realidad, se emplean dos bloques, uno para enviar y otro para recibir, no obstante, la configuración se realiza de igual manera en ambos. Estos bloques, representados en las Figuras 3.8 y 3.9, dan la opción de hacer que el robot trabaje como cliente o que lo haga como servidor.


**Fig. 3.8: Bloque recibimiento**


**Fig. 3.9: Bloque de envío**

Como se tratará más adelante, en una de las tareas de este trabajo interesaba que el robot trabajase como servidor, por lo que en las figuras antes citadas aparece la configuración de servidor.

El puerto de salida del bloque de recibimiento de datos llamado *Status* puede adoptar dos valores diferentes. Adopta un 1 en el instante en que se recibe un dato y 0 el resto del tiempo. Esto puede ser de utilidad si se quieren fijar los datos que se reciben.

### 3.3.2 Control de la posición mediante sockets

En un principio, se planteó realizar el control de posición del robot enviando a un socket creado en un script de Matlab la lectura de los encoders de las ruedas. Todo esto se podía realizar debido a que Matlab tenía la función *tcpip(),* explicada anteriormente, y que la librería de soporte de Simulink para Lego Mindstorms EV3 poseía los dos bloques necesarios para establecer también la comunicación.

Una vez se llegasen las lecturas de ambos encoders al script de Matlab, la idea era aplicar las expresiones de control de posición detalladas en la parte teórica utilizando el mismo socket, es decir, se transcribirían a código en Matlab todas las expresiones del control de posición necesarias para calcular las acciones de control a aplicar en los motores del robot, para que, finalmente, estas acciones de control se enviasen al esquema de Simulink desde el socket al bloque de recibir datos.

Realizando de esta manera el control, se podría tener disponible la posición del target en el workspace de Matlab en tiempo real, cosa que será de utilidad para tareas posteriores.

El esquema de Simulink creado para realizar esta tarea es el mostrado en la Figura 3.10:

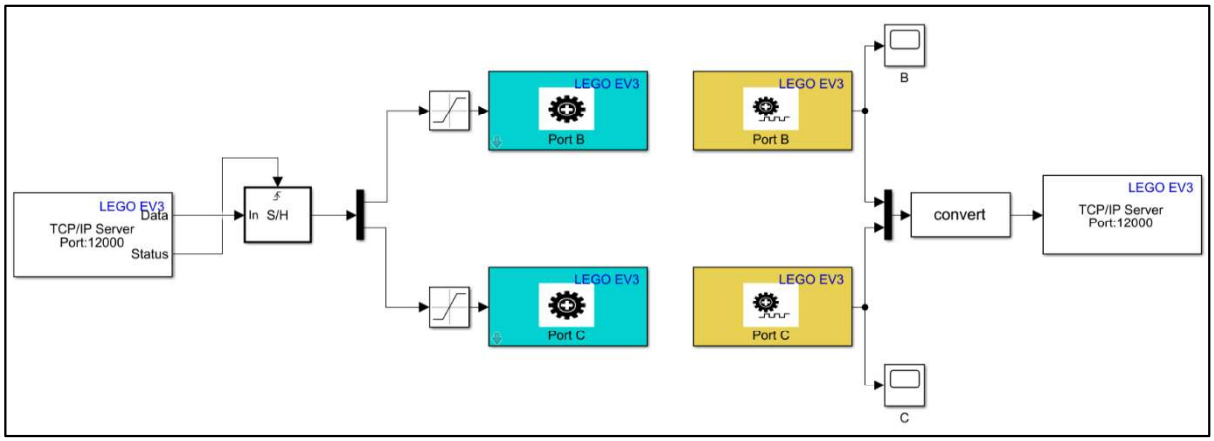


**Fig. 3.10: Captura Simulink para el envío de datos de los encoders (amarillo) y el recibimiento de las acciones de control**

Por otro lado, como se ha comentado anteriormente, el robot se tenía que comportar como servidor y el script como cliente. Esto debía ser así porque primero se tenía que ejecutar el esquema de Simulink y después el script con el socket (al revés no funcionaba porque si se ejecuta el script primero, las funciones de Simulink se bloqueaban, quedando de esta manera inhabilitada la ejecución del esquema).

A partir de la función *tcpip()* descrita anteriormente y teniendo en cuenta que la sesión de Matlab actuaba como cliente, se creó el objeto para establecer la comunicación de la siguiente manera:

```
tcp_lego=tcpip('192.168.0.102',12000,'NetworkRole','client');
```

donde la dirección IP del host era la del robot y el puerto de conexión era el mismo para ambas máquinas, ya que si no fuese el mismo no se hubiese podido establecer la comunicación.

Con tal de no realizar trabajo de más, antes de escribir todo el código necesario para realizar el control de posición del robot, se realizó una prueba para verificar que la lectura de los encoders y las acciones de control se enviarían y recibirían correctamente a través de los bloques TCP/IP. El esquema de Simulink empleado es como el mostrado en la Figura 3.11, y el código necesario para recibir los datos en Matlab es el siguiente:

```
lego = tcpip('192.168.0.102',80,'NetworkRole','Client');
fopen(lego);
i=1;
fwrite(lego,1000);  %Se envía el 1000 para lanzarlo por pantalla
while i<1001
      encoder(i) = fread(lego,1);  %Lectura del encoder
      i=i+1;
end
t=0:0.02:20-0.02;
fclose(lego);
plot(t, encoder);
```

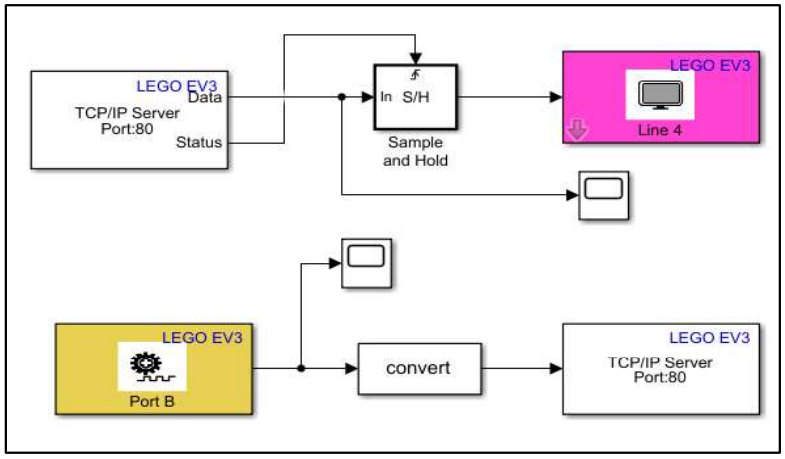


**Fig. 3.11: Esquema Simulink para la verificación del envío y recibimiento de datos con bloque TCP/IP**

La prueba consistió en ejecutar el esquema de Simulink de la Figura 3.11 y el script de Matlab con el código anterior y verificar que en la pantalla del robot aparecía el número 1000, mientras se le realizaba un giro completo, 360°, a la rueda acoplada al motor conectado al puerto B de salida.

Al finalizar la prueba, se realizó una gráfica, Figura 3.12, con los datos recibidos del encoder.

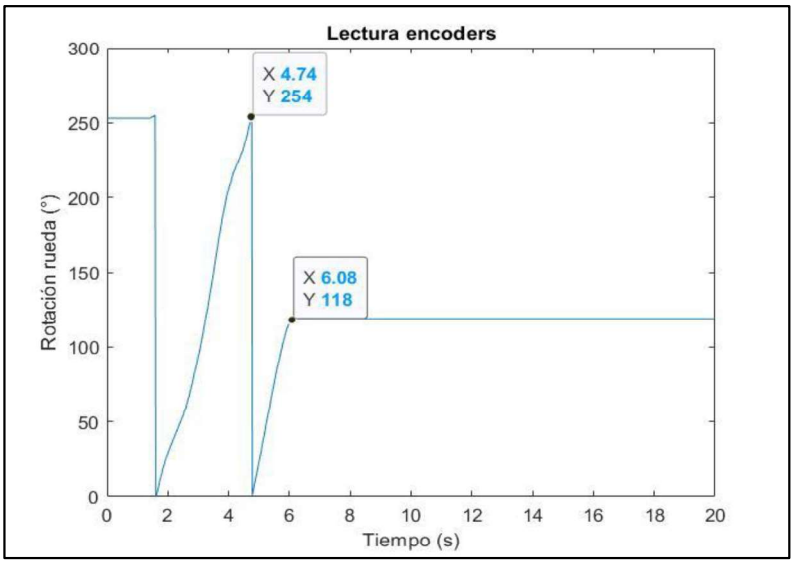


**Fig. 3.12: Gráfica con datos recibidos del bloque TCP/IP**

Como se puede observar en la gráfica anterior, el giro de la rueda empieza sobre los dos segundos y no para hasta llegar a los seis segundos aproximadamente. Además, como la precisión del encoder es de 1°, a cada intervalo de tiempo que pasa en cada iteración, el valor de la rotación será un número entero siempre. Por ello también se observa que, aunque la rueda se giró de forma continua, sobre los cinco segundos se produce un salto desde los 255° a los 0° y vuelven a subir de forma continua la rotación de la rueda hasta que, a los seis segundos, al pararse el giro, se mantiene constante en 118°. Si se realiza la suma entre 255° y 118°, el resultado es 373°, o lo que es lo mismo, aproximadamente la vuelta que se le ha dado a la rueda.

El salto producido al llegar a 255° es inadmisible si se quiere controlar la posición del robot, ya que, a partir del ángulo girado de la rueda en un instante, y el muestreado en el instante justamente anterior, se calcula la velocidad angular de la rueda y, mediante las expresiones que se han visto en el desarrollo teórico, la posición y orientación del robot. Si se produce un salto del tal magnitud, se producirá un error enorme en el cálculo de la velocidad angular de las ruedas, y consecuentemente, un error en la obtención de la posición y orientación del robot que harían que el esquema de control no funcionase.

Visto que no se puede permitir este error, se investigó acerca del bloque TCP/IP de Lego y se llegó a la conclusión de que con este bloque, lo que se envían son datos binarios cuyo tamaño máximo no excede los 8 bit, es decir, cuando Matlab realiza la conversión a decimal, como máximo podrá mostrar 255. Lo que se acaba de comentar se puede explicar con lo siguiente:

$$1111\ 1111_2 = 255_{10} \tag{3.1}$$

$$0001\ 0000\ 0000_2 = 256_{10} \quad (3.2)$$

Cuando la rueda ha girado un total de 255°, el valor binario que envía el bloque TCP/IP se corresponde a las expresión (3.1), mientras que, cuando la rueda gira un grado más, es decir, 256°, el valor en binario correspondiente debería ser una palabra de 9 bit como el mostrado en la expresión (3.2), pero, como el tamaño de palabra aceptada por el bloque TCP/IP está limitado a 8 bits, se trunca la palabra quedando una palabra binaria con ocho 0. Esta palabra, al ser traducida a su número decimal correspondiente por Matlab, pasa a ser el 0. Todo esto es precisamente lo que sucede en la gráfica de la Figura 3.12.

Como está limitación provocaba que se tuviese un error en el control, se pensó en convertir la lectura del encoder de grados a radianes. Para ello se introdujo el bloque *puls2rad*, ver Figura 3.13.

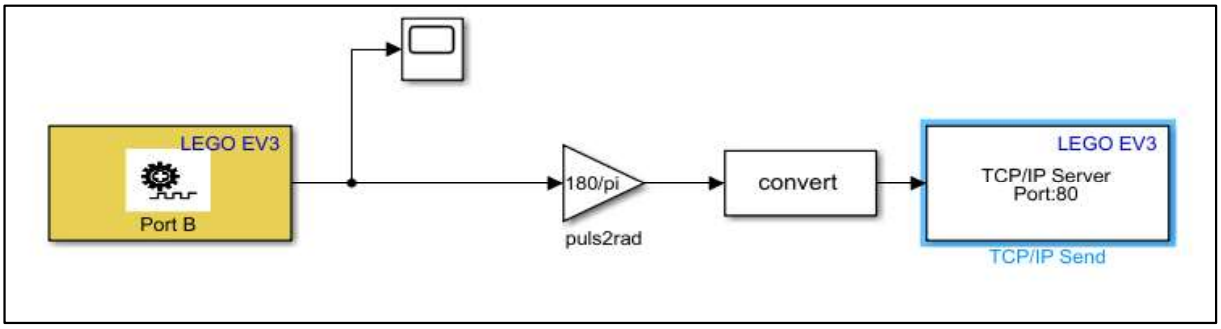


**Fig. 3.13: Esquema Simulink con bloque para la conversión de grados a radianes.**

El bloque *convert* se ha de configurar para que los datos se convirtiesen de *int64* a *double*, ya que en este caso se estaba trabajando con radianes y, por consiguiente, con números decimales.

Una vez realizada la prueba, se hizo la gráfica correspondiente a la Figura 3.14.

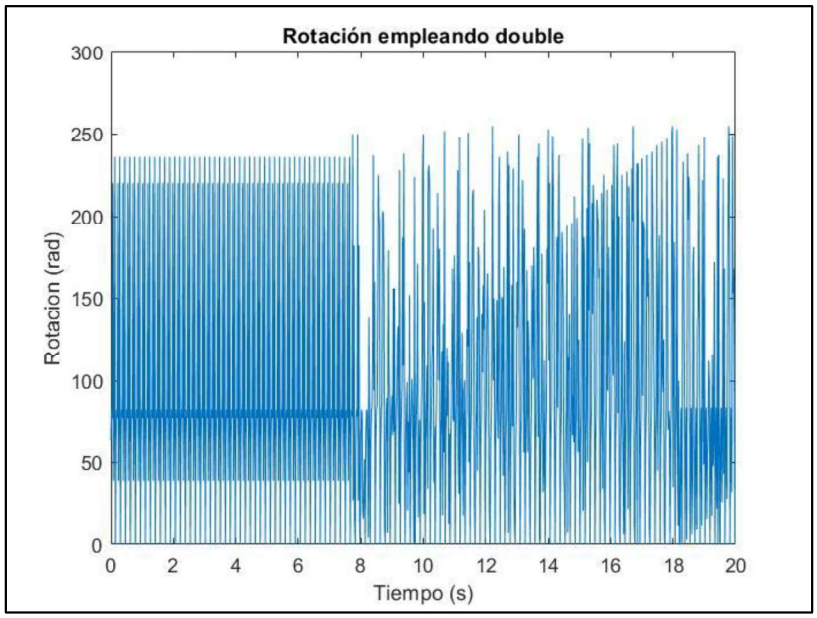


**Fig. 3.14: Gráfica con datos de tipo double recibidos del bloque TCP/IP**

Como se pude observar, convirtiendo los datos a *double* el error sería aún mayor. Además, también hay que comentar que los datos parece que no llegan como radianes, y es por ello que se volvió a poner en el bloque *convert* el tipo de dato *int64*. Con este tipo de dato, la gráfica representada es la de la Figura 3.15.

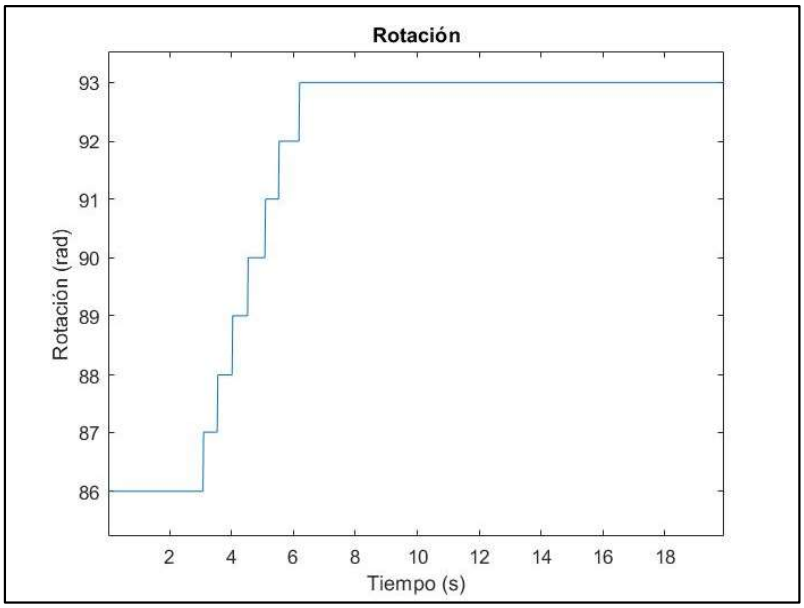


**Fig. 3.15: Gráfica con datos de tipo int64 recibidos con el bloque TCP/IP.**

Observando la anterior gráfica, se puede determinar que el bloque *puls2rad* está funcionando, ya que el salto que se produce en la rotación está ligeramente por encima de 6 radianes, o lo que es lo mismo, una vuelta completa de la rueda.

Lo que sucede ahora es que sigue existiendo error, debido a que al tratarse de datos del tipo *int64*, se trunca la parte decimal y únicamente se queda la parte entera. Por este motivo se ven los escalones de 1 radián de amplitud que empiezan en 86 radianes y acaban en 93 radianes.

A pesar de que el salto es de 1 radián, esto tampoco podía ser permitido debido a que 1 radián es el equivalente a 57.3°, es decir, que si en un determinado instante se detecta un giro de la rueda de 20° y en el instante inmediatamente posterior se detecta un giro de 23°, como ambos valores están por debajo de 57.3°, la conversión daría que en ambos instantes la rotación es de 1 radián, y como consecuencia, la velocidad de rotación de la rueda sería nula, cuando realmente no lo sería.

Llegados a este punto, debido a los errores presentes, fue descartada la posibilidad de realizar la comunicación de la posición mediante el envío de la rotación de las ruedas utilizando el bloque TCP/IP de Lego Mindstorms EV3 y la función *tcpip* de Matlab.

## 3.4 Generación de esquemas con Simulink para el control de posición del robot móvil Lego Mindstorms EV3

Debido a que no se pudo realizar el control utilizando código escrito manualmente y los bloques de comunicación TCP/IP de Lego, la siguiente opción fue la de desarrollar los controladores de posición, tanto con el método del punto descentralizado como por persecución pura, con esquemas de Simulink.

Antes de empezar con el desarrollo de los esquemas, comentar que hay una serie de parámetros constantes e idénticos en ambos tipos de controladores. Estos parámetros son:

| Parámetro | Descripción (unidades) | Valor |
|---|---|---|
| r | Radio rueda (m) | 0.028 |
| b | Dist. Centro-rueda (m) | 0.059 |

**Tabla 3.2: Parámetros constantes en ambos controladores.**

Con el fin de crear los controladores empleando Simulink, simplemente se tuvieron que utilizar las librerías de esta aplicación de Matlab e ir transcribiendo las expresiones correspondientes al control de posición que ya han sido analizadas en el desarrollo teórico.

Con el fin de ahorrar tiempo, como la cinemática directa es igual en ambos tipos de controladores, a continuación, se adjunta la captura, Figura 3.16, con su correspondiente esquema de Simulink.

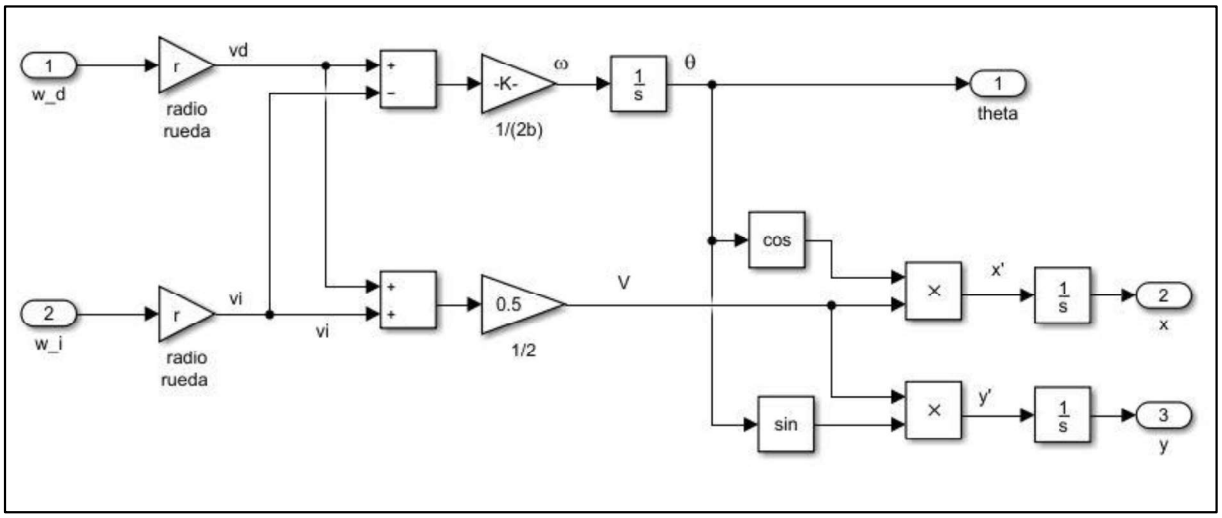


**Fig. 3.16: Esquema correspondiente a la cinemática directa.**

Por otro lado, el bloque que hay en la Figura 2.13 del apartado *2.3* correspondiente a los motores del robot móvil también es idéntico en ambos controladores, y tiene la siguiente disposición:

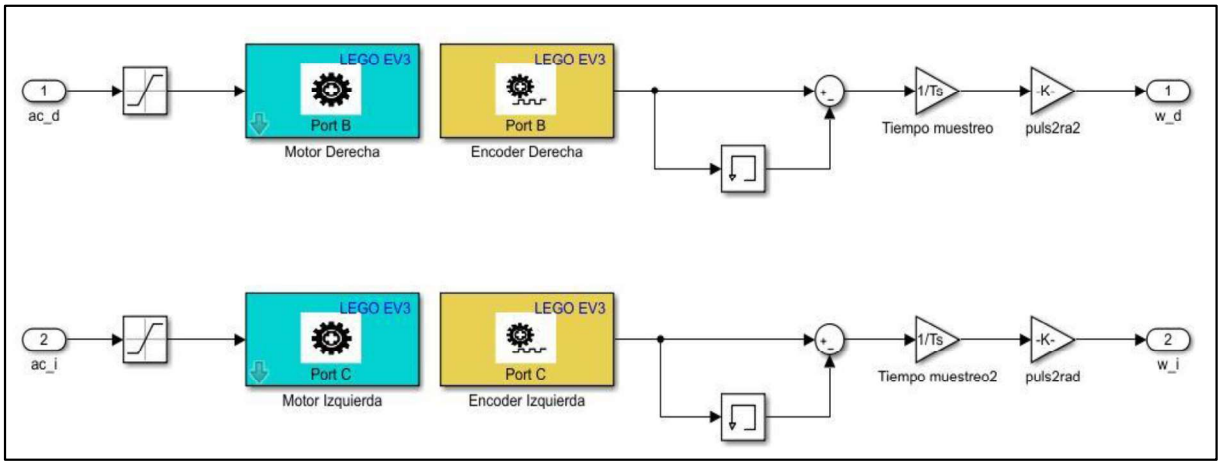


***Fig. 3.17:*** **Esquema correspondiente a aplicar las acciones de control a los motores y leer el valor de la rotación de los encoders**

Este último bloque, el de los motores, puede ser sustituido por una función de transferencia que realiza la misma función que los bloques de los motores y encoders, bloques azules y

amarillos en la Figura 3.17, respectivamente. Esta función de transferencia podía ser obtenida utilizando un método empírico que no era objeto de este trabajo, por lo que no se va a desarrollar su obtención. El bloque de simulación es muy práctico debido a que con él se puede anticipar de qué forma recorrerá el robot la trayectoria de referencia que se le asigne.

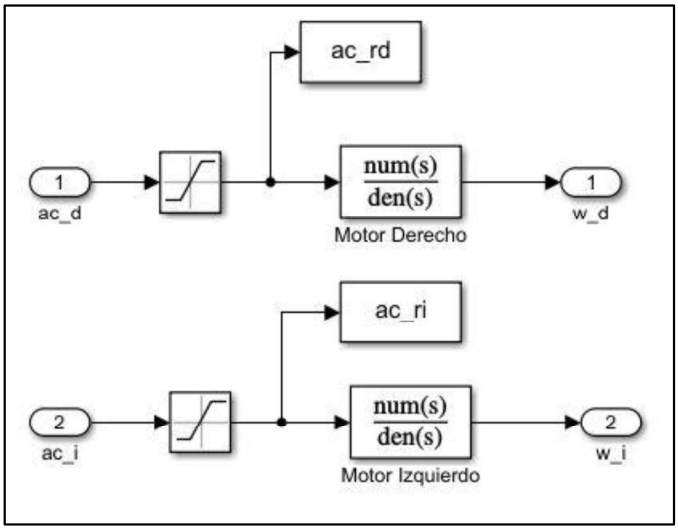


**Fig. 3.18: Esquema del bloque de simulación que sustituiría al motor**

Y, por último, el bloque del control dinámico que controla que las velocidades de rotación de las ruedas se asimilen a las de referencia tampoco cambia de forma, siendo esta como la de la Figura 3.19.

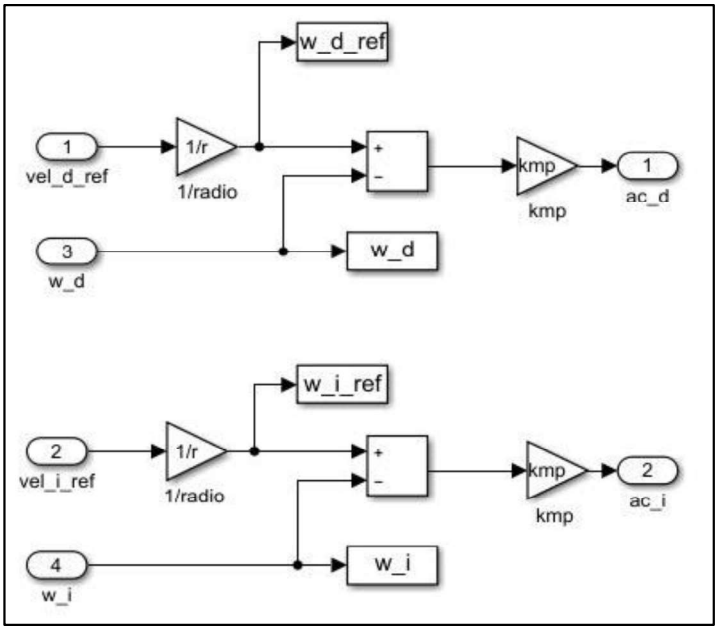


**Fig. 3.19: Esquema del control dinámico**

### 3.4.1 Esquema del control de posición por punto descentralizado

En el apartado *2.3.1.2* se han desarrollado las expresiones de la cinemática inversa necesaria para el control de posición por punto descentralizado.

En las siguientes figuras se presentarán los bloques de las referencias, cinemática inversa y control cinemático.

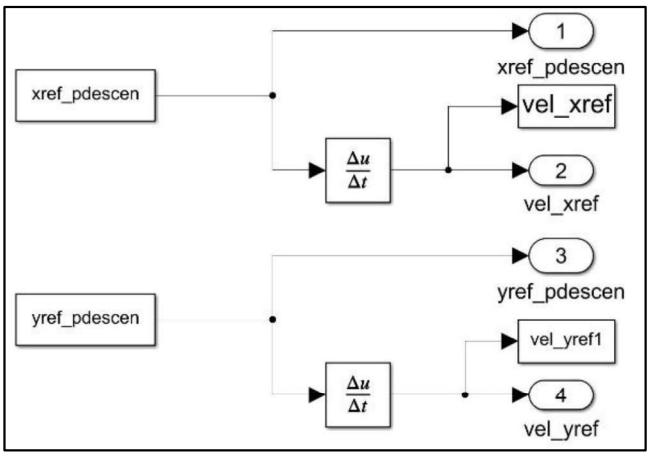


**Fig. 3.20: Esquema Simulink de las referencias.**

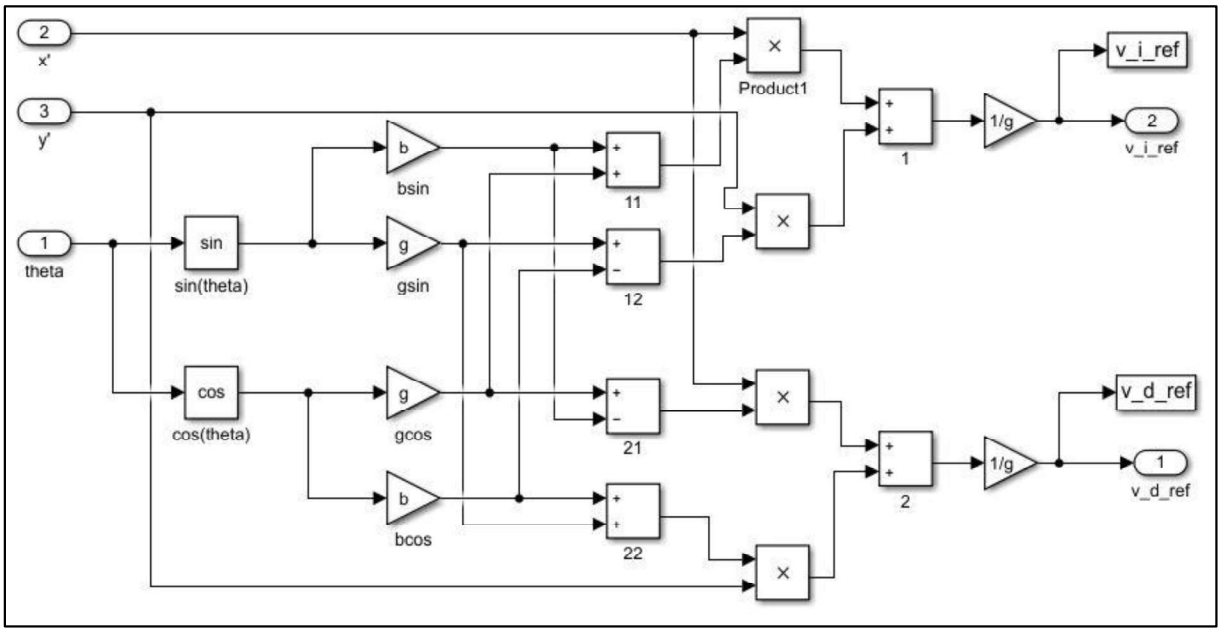


**Fig. 3.21: Esquema Simulink de la cinemática inversa por punto descentralizado**

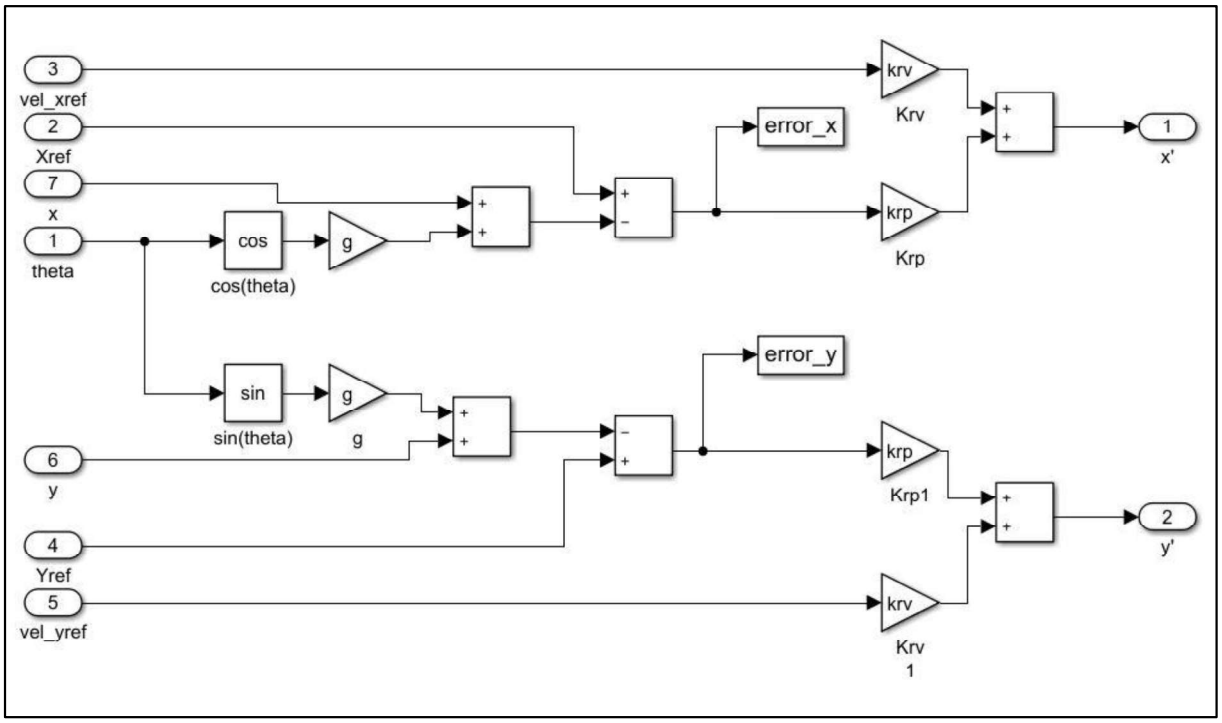


**Fig. 3.22: Esquema Simulink del control cinemático por punto descentralizado**

Como se puede observar en la Figura 3.20, en cada iteración se calcula la velocidad de referencia respecto los ejes de coordenadas. Esto se hace así porque resulta necesario para el bloque del controlador cinemático, ya que, como se vio en la parte teórica, se trata de un controlador proporcional cuyas variables de entrada son tanto las coordenadas de referencia como las velocidades de referencia.

Por otro lado, cabe decir que los parámetros $K_{rp}$ y $K_{rv}$ que aparecen en la Figura 3.22 son las constantes proporcionales del controlador cinemático, y el parámetro g de la Figura 3.21 se corresponde con la distancia a la que se encuentra el punto descentralizado empleado en el control. Todos estos valores se obtendrán más adelante por un método de prueba y error utilizando el esquema de Simulink con el bloque del simulador.

Finalmente, el esquema de Simulink con el control de posición por punto descentralizado quedó de la siguiente manera:

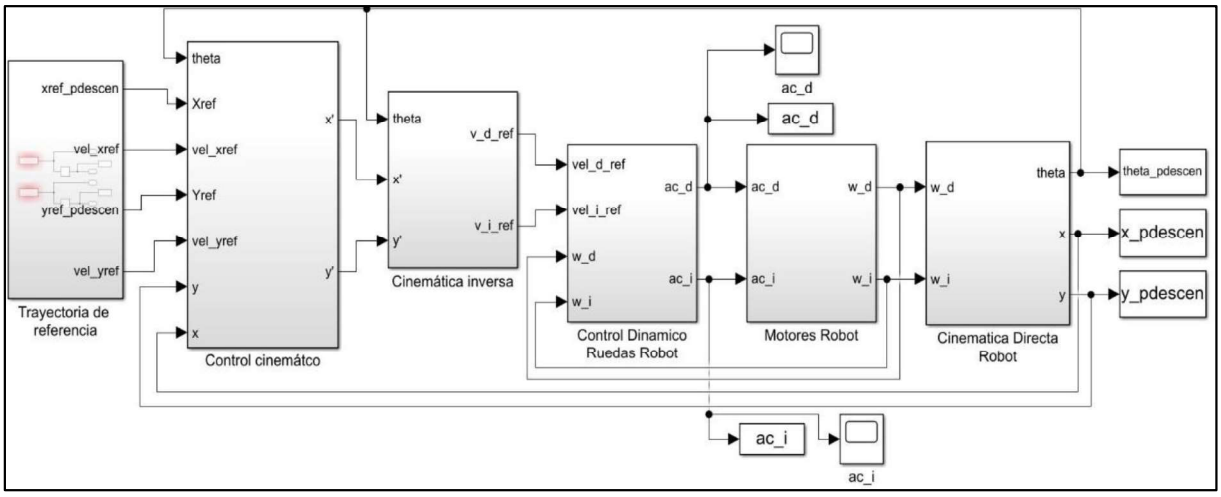


**Fig. 3.23: Esquema Simulink del control de posición por punto descentralizado**

En este esquema se pueden ver representados la conexión entre los distintos bloques que se han ido viendo de forma más detallada en las páginas anteriores.

## 3.4.2 *Esquema del control de posición por persecución pura*

Como se ha desarrollado en la parte teórica, en este tipo de controlador, el control cinemático y la cinemática inversa se agrupan en un mismo bloque, quedando este definido de la siguiente forma:

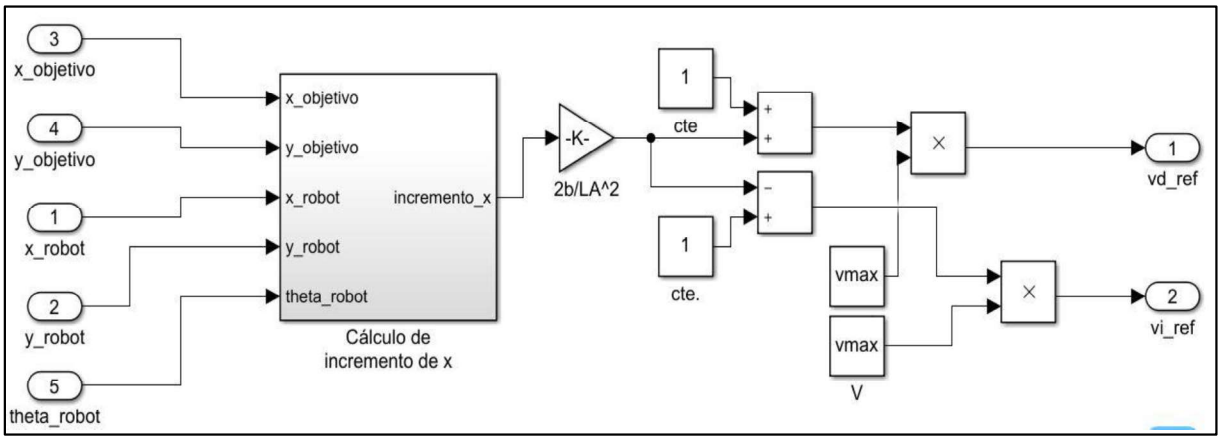


**Fig. 3.24: Esquema Simulink del control cinemático y cinemática inversa por persecución pura**

El bloque con el que se calcula *Δx* es el siguiente:

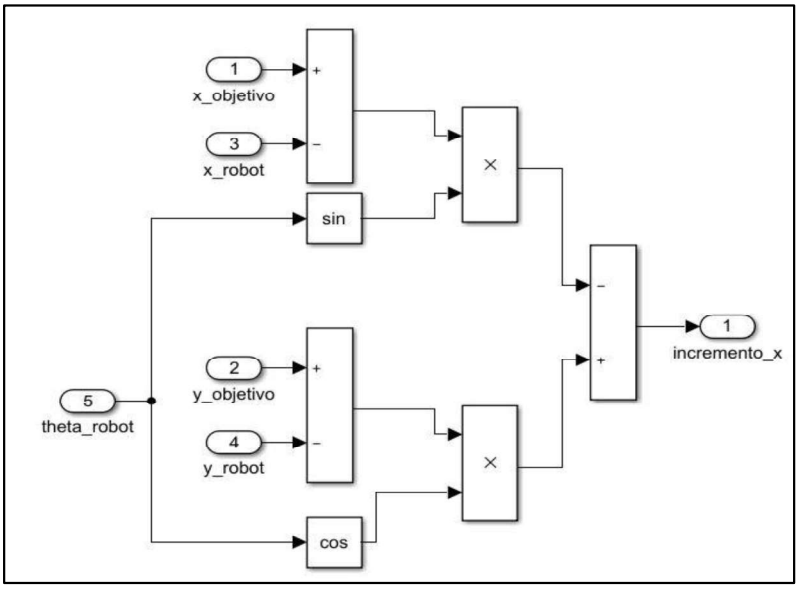


**Fig. 3.25: Esquema Simulink del cálculo del Δx**

En las Figuras 3.24 y 3.25, se han implementado con bloques de Simulink las expresiones para el control de posición que aparecen en el apartado *2.3.1.3* del desarrollo teórico.

Finalmente, de igual forma que se ha hecho con el control por punto descentralizado, se presentará el esquema completo para el control de posición por persecución pura con el fin de aclarar cómo están dispuestos los principales bloques del control.

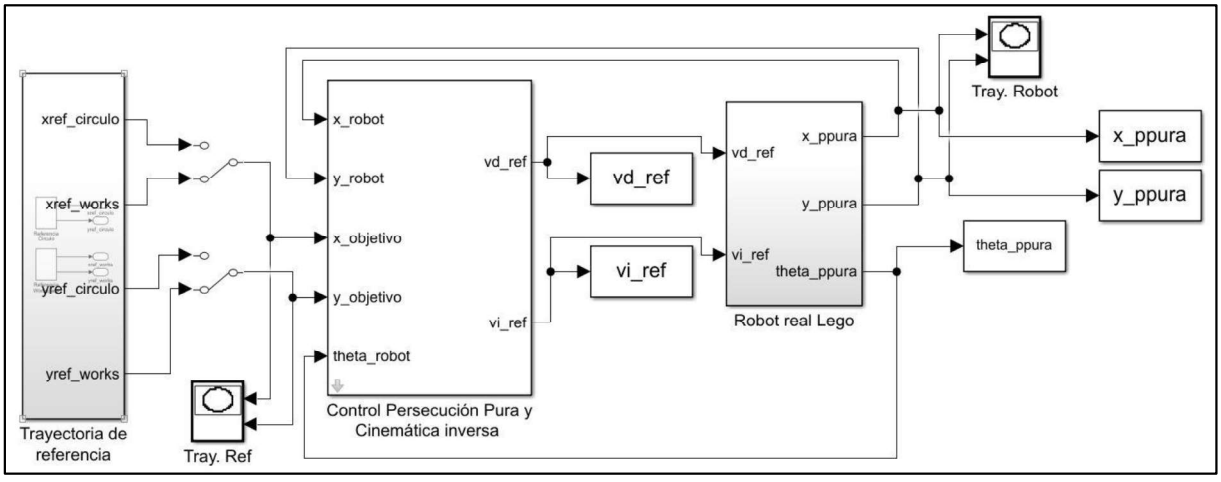


**Fig. 3.26: Esquema Simulink del control de posición por persecución pura**

En el bloque *Robot real Lego* están integrados los bloques del control dinámico, de los motores y de la cinemática directa que ya han sido mostrados con anterioridad.

## 3.5 Ajuste de constantes y validación de los esquemas

Una vez desarrollados los esquemas de ambos controladores, se aprovechó la disponibilidad del bloque de simulación para ajustar, en el caso de emplear el algoritmo del punto descentralizado, las constantes de proporcionalidad necesarias para realizar el control cinemático, $K_{rp}$ y $K_{rv}$, y la distancia a la que se sitúa el punto descentralizado, $g$, y para el caso en el que se emplee el control por persecución pura, la distancia de Look Ahead, $LA$, y la velocidad de avance, $V_{avance}$. Además, mientras se obtenían dichas variables de control, a la vez, se verificaba que los esquemas estaban realizados correctamente.

Comentar también que, la constante de proporcionalidad del control dinámico, $K_{mp}$, también tenía que ser hallada, y como el valor numérico sería el mismo tanto para el control utilizando punto descentralizado como en el que emplea la persecución pura, se obtuvo dicho valor empleando el algoritmo del punto descentralizado y posteriormente, con ese mismo valor, se obtuvieron los parámetros correspondientes al algoritmo de la persecución pura.

Para realizar el ajuste y la validación, se introdujo en ambos esquemas una trayectoria de referencia generada a partir de la función *bez_2ptos_orientacion* desarrollada en un aparatado anterior.

### *3.5.1 Ajuste de g, Kmp, Krv y Krp*

Como se ha dicho, para realizar el ajuste de estas constantes se utilizó el esquema del control de posición por punto descentralizado.

El primer parámetro que se fijó fue la distancia a la que se encuentra el punto descentralizado, *g*. Como se sabía que, si se adopta un valor para esta distancia elevado, el robot no seguiría tan bien la trayectoria, mientras que si se adopta uno relativamente bajo, el robot tendería a realizar bien la referencia fijada, se decidió fijar el valor de este parámetro en 7 centímetros, o lo que es lo mismo, aproximadamente la mitad de la longitud del robot.

El segundo parámetro a fijar fue la constante de proporcionalidad del control dinámico, ya que esta es la responsable de que la velocidad angular de las ruedas se asimile a la de referencia. Por ello, se fijó la $K_{rv}$ a 1 y la $K_{rp}$ a 1 y se empezó a simular modificando en cada simulación el valor de $K_{mp}$.

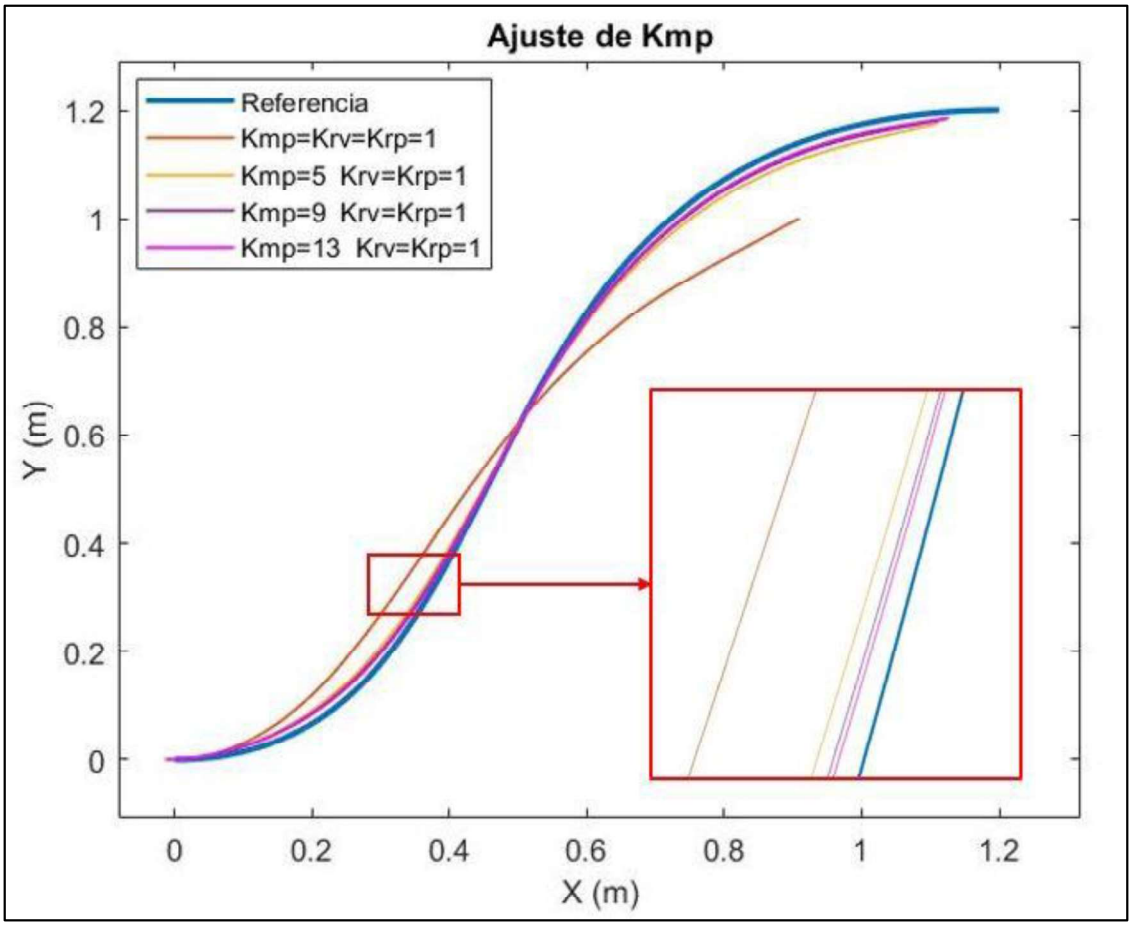


**Fig. 3.27: Gráfica obtenida tras la simulación con distintas *Kmp*.**

Como se puede observar en la Figura 3.27, a medida en que se iba aumentado la $K_{mp}$, la trayectoria seguida por el robot se acercaba cada vez más a la referencia. Finalmente, se decidió dejar $K_{mp}$ con un valor de 13 puesto que como se puede observar en la ampliación, la mejora que supone emplear una $K_{mp}$ de 13 respecto a una $K_{mp}$ de 9 no es muy notable, por lo que se pensó que no merecía la pena seguir ajustando este parámetro.

Una vez obtenida $K_{mp}$, se pasó a ajustar el valor de $K_{rp}$ y $K_{rv}$. Para ello se empezó modificando el valor de $K_{rp}$ de la misma manera en que se hizo con la $K_{mp}$. Posteriormente, se modificó de igual forma la $K_{rv}$.

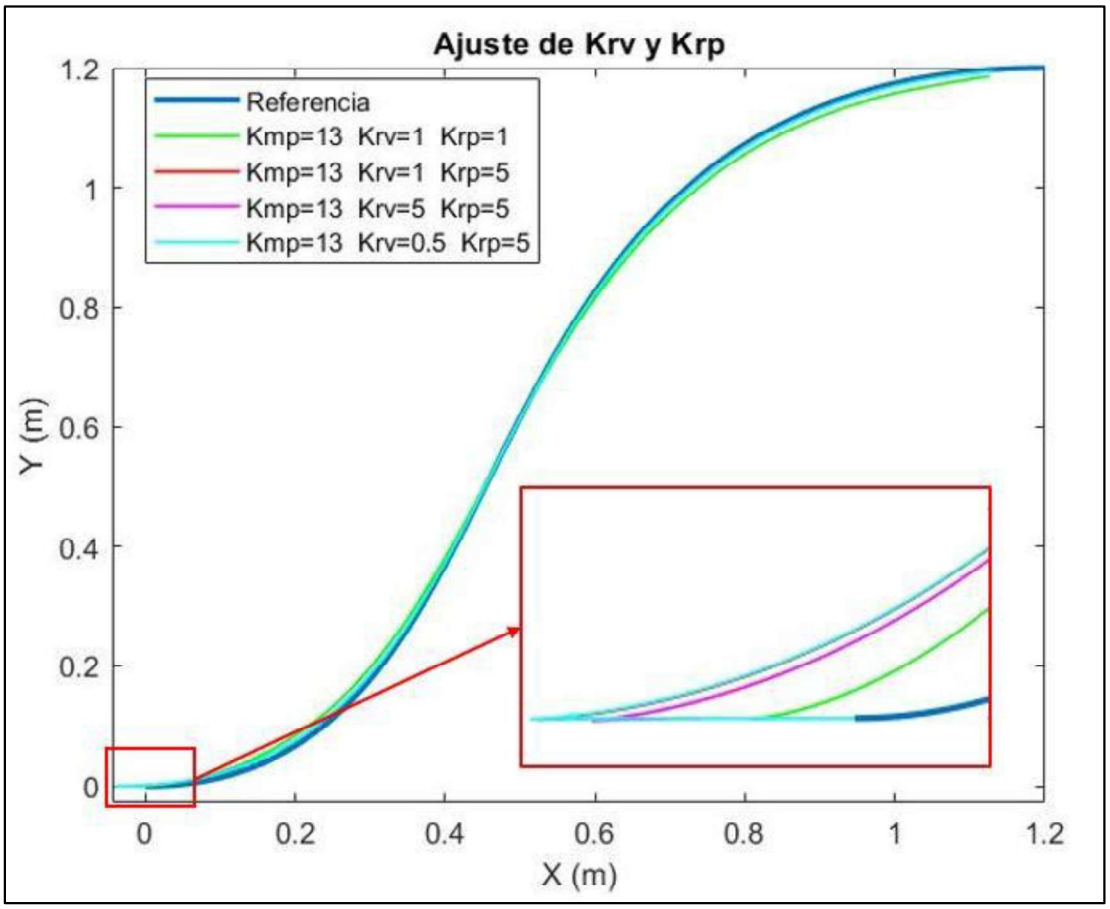


**Fig. 3.28: Gráfica obtenida tras la simulación con distintas *Krp* y *Krv***

Como se puede observar en la Figura 3.28, con una $K_{rp}$ de 5, la trayectoria del robot mejoraba en cierta medida respecto a una $K_{rp}$ de 1, lo único que sucedía era que, al inicio, el robot tendía a recular unos centímetros para después seguir sin apenas error la trayectoria de referencia.

Comentar también que, el movimiento hacia detrás que realizaría el robot al inicio de la trayectoria se debía a que el punto descentralizado no estaba sobre el punto de inicio de la trayectoria, sino que estaba algo más adelantado.

Por lo que respecta a la $K_{rv}$, apenas existía diferencia entre emplear una de 5, una de 1 o una de 0.5. Sin embargo, sí que existía una diferencia notable cuando se representaban las acciones de control, Figura 3.29. Como se puede observar, para una $K_{rv}$ de 5, las acciones de control estarían por encima del nivel de saturación, 100, por lo que las acciones de control que recibiría el robot real serían el de ese valor de saturación y obviamente el robot no realizaría la trayectoria asignada.

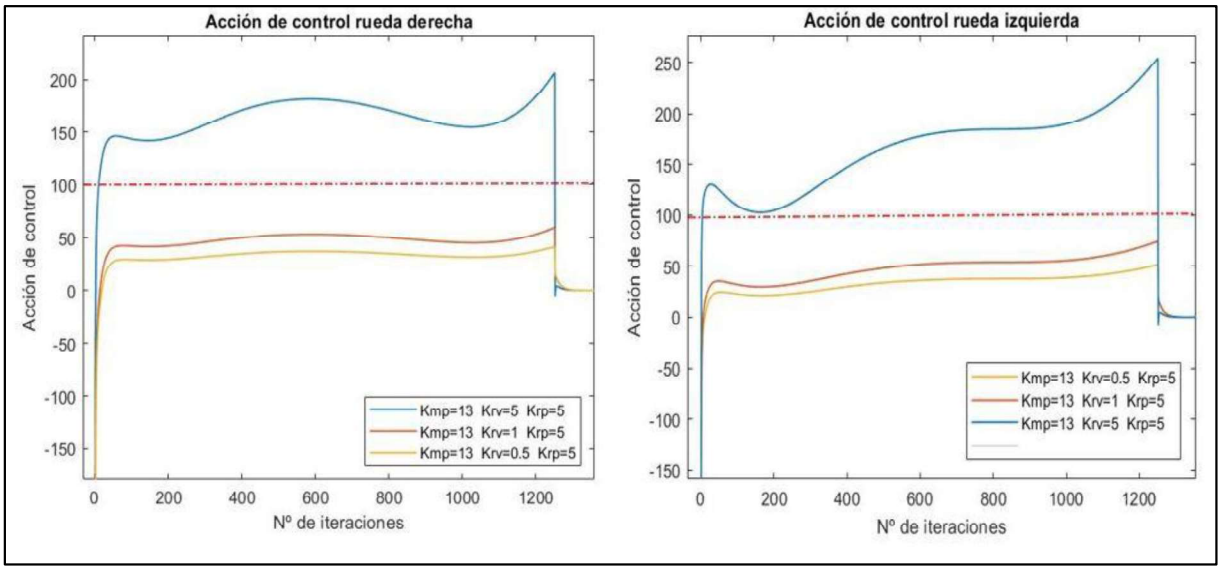


**Fig. 3.29: Gráficas con las acciones de control de cada rueda.**

Visto esto, se optó por escoger una $K_{rv}$ de 0.5 debido a que la trayectoria se seguía de forma más o menos correcta y las acciones de control eran las más suaves, por lo que el riesgo de derrape de las ruedas era menor y por ende, la probabilidad de que el robot real siguiese la trayectoria de referencia sería mayor.

Además, como se ha comentado antes, debido a que las coordenadas del punto descentralizado no se corresponden con las coordenadas del inicio de la trayectoria, el robot realiza un movimiento hacia atrás. Esto puede ser observado en las gráficas de las acciones de control, donde se observa que, al inicio, estas llegan de forma brusca a la saturación negativa, -100, lo que puede provocar el derrape de las ruedas y por consiguiente, empezar el control de posición con un error que ya no se puede corregir. Para evitar este movimiento hacia atrás, se empleó la función *dist_pto_avanzado,* creada para realizar las curvas de Bézier a partir de dos puntos y dos orientaciones, que desplaza el inicio de la trayectoria y lo sitúa justo sobre el punto descentralizado.

Llegados a este punto, y antes de realizar una nueva simulación con los parámetros ya ajustados y corrigiendo el inicio de la trayectoria, se muestra una tabla con los valores finalmente adoptados.

| | |
|---|---|
| $g$ | 0.07 (m) |
| $K_{mp}$ | 13 |
| $K_{rv}$ | 0.5 |
| $K_{rp}$ | 5 |

**Tabla 3.3: Valor de las constantes de proporcionalidad y de *g***

Finalmente, se volvió a simular la respuesta del robot, esta vez con el punto descentralizado sobre el inicio de la trayectoria. La respuesta del robot hubiese sido como la mostrada en la Figura 3.30.

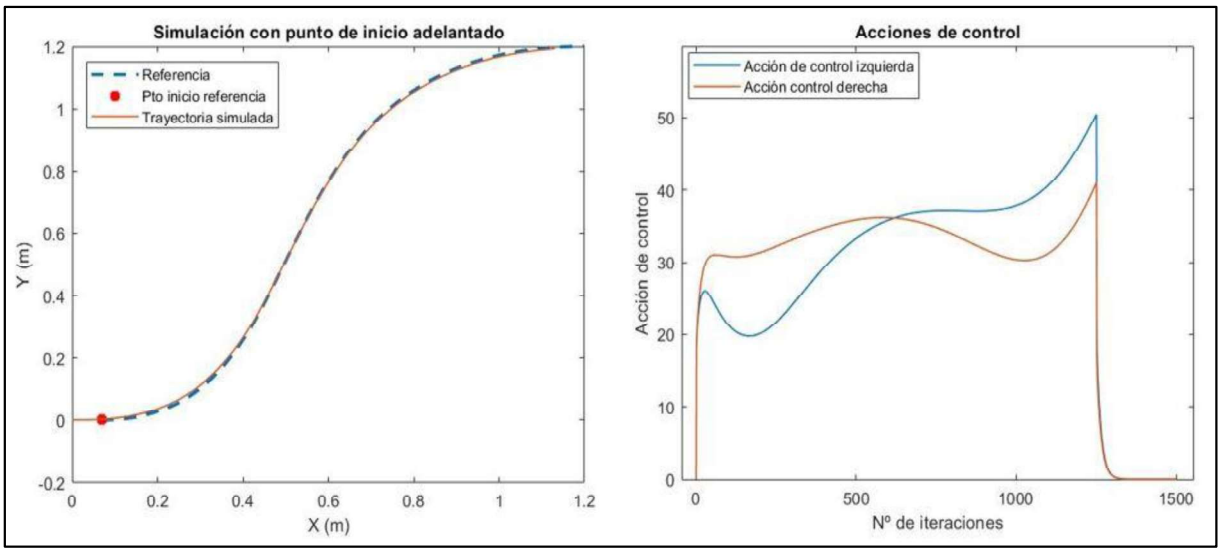


**Fig. 3.30 : Gráficas con la referencia avanzada en el inicio y las acciones de control correspondientes a dicha referencia**

Como se puede observar, el robot seguiría de manera correcta la referencia. Además, como también se puede observar, las acciones de control no llegaban a la saturación en este caso y con ello, la probabilidad de derrape, y con ello de error, se reducía notablemente.

## 3.5.2 *Ajuste de la distancia LA y de la Velocidad de avance*

Como ya se ha comentado, el valor de la constante de proporcionalidad del control dinámico sirve tanto para el control de posición por punto descentralizado como por persecución pura. Por ello, como ya se tenía el valor de $K_{mp}$, solamente era necesario ajustar los valores de *LA* y $V_{avance}$.

En realidad, la $V_{avance}$ del robot se podía ajustar sin necesidad de hacer demasiadas pruebas, ya que como es lógico, el robot presenta unas limitaciones mecánicas que no le permitirían llegar a una velocidad elevada, por mucho que se le exigiese en el control. Por ello, se decidió establecer la $V_{avance}$ en 25 cm/s, o lo que es lo mismo, 0.25 m/s.

Una vez fijada la velocidad de avance, solo quedaba ajustar el parámetro LA. Realizando una serie de simulaciones se obtuvo:

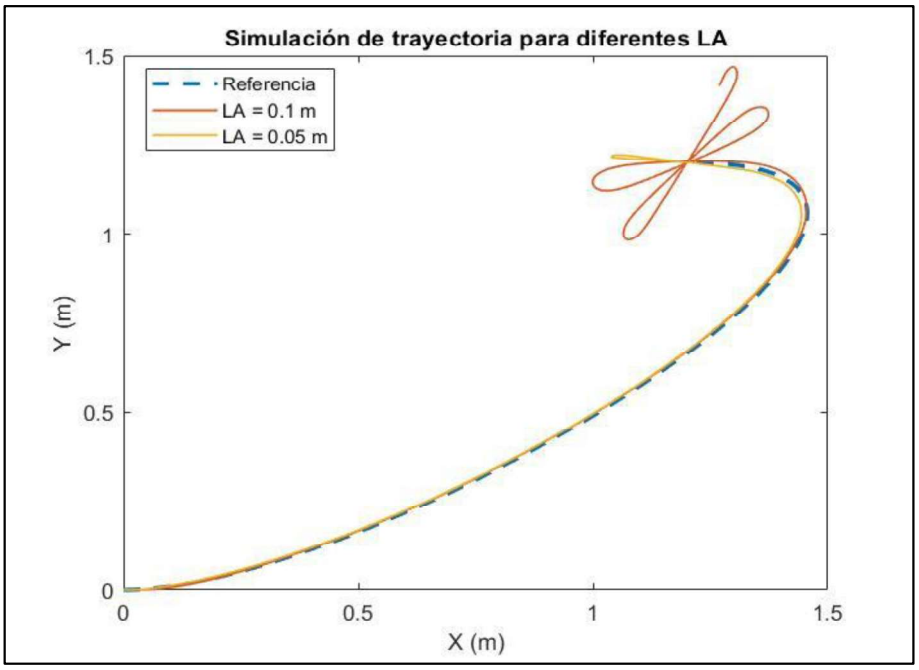


**Fig. 3.31: Gráfica con las distintas trayectorias que realizaría el robot según la distancia LA**

Como se puede observar en la Figura 3.31, según la simulación, cuando el robot llega al último punto de la trayectoria de referencia, empezaría a dar vueltas, siempre pasando por dicho punto, pero no se quedaría quieto en ningún momento.

Esto sucede debido a que, en el esquema de control, la velocidad de avance del robot se suponía constante en todo momento, por tanto, cuando el robot llegaba al final de la trayectoria, su referencia pasaría a ser ese último punto, y como la velocidad de avance no cambiaba, el robot seguía moviéndose hacia ese punto.

Para subsanar este problema, se modificó el esquema de control para que la velocidad fuese variable, de esta manera, se consiguió que el robot se detuviese al llegar al final de la referencia.

Para que la velocidad pasase a ser variable, se tuvo que poner esta en función de algún o algunos parámetros. Lo que se decidió fue ponerla en función del error entre la posición del robot y la de referencia, es decir, cuando el robot estuviese muy alejado de la referencia, el error sería elevado, y con ello la velocidad de avance también lo sería, mientras que, si el error es pequeño, la velocidad disminuye hasta que llega un momento en que podría ser nula. Este momento se da cuando el error es nulo, es decir, cuando el robot está situado sobre el punto que tiene como referencia.

La expresión utilizada para obtener una velocidad de avance variable es la siguiente:

$$V_{avance} = \sqrt{(x_{obj} - x_{rob})^2 + (y_{obj} - y_{rob})^2} \tag{3.3}$$

Esta expresión, se incorporó en el esquema de Simulink para el control por persecución pura dentro del bloque del cálculo del *Δx*. En la siguiente figura, la parte del esquema encerrada en el rectángulo rojo se corresponde con la expresión (3.3). El bloque de saturación que se observa se ha de incorporar porque, aunque no se quiera que la velocidad de avance sea constante, tampoco se puede permitir que sea muy elevada cuando el error sea elevado, por tanto, se satura la velocidad a una velocidad máxima que sea aceptable para el movimiento del robot, es decir, a la $V_{avance}$ fijada anteriormente.

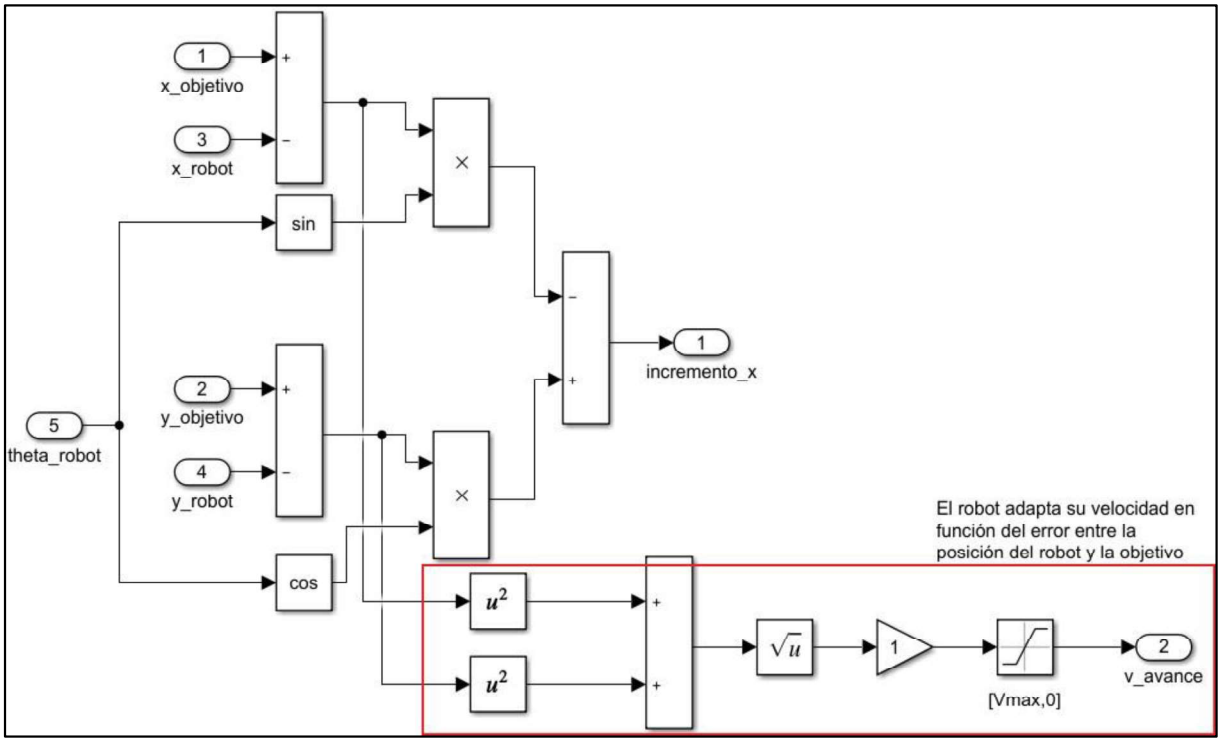


**Fig. 3.32: Esquema Simulink para el cálculo de *Δx* con velocidad de avance variable**

De esta manera, las salidas del citado bloque, *Δx* y *v_avance*, son empleadas para el posterior cálculo de las velocidades lineales de referencia que han de tener las ruedas. Esto se realiza como se puede visualizar en la Figura 3.32 dentro del bloque del control cinemático y la cinemática inversa.

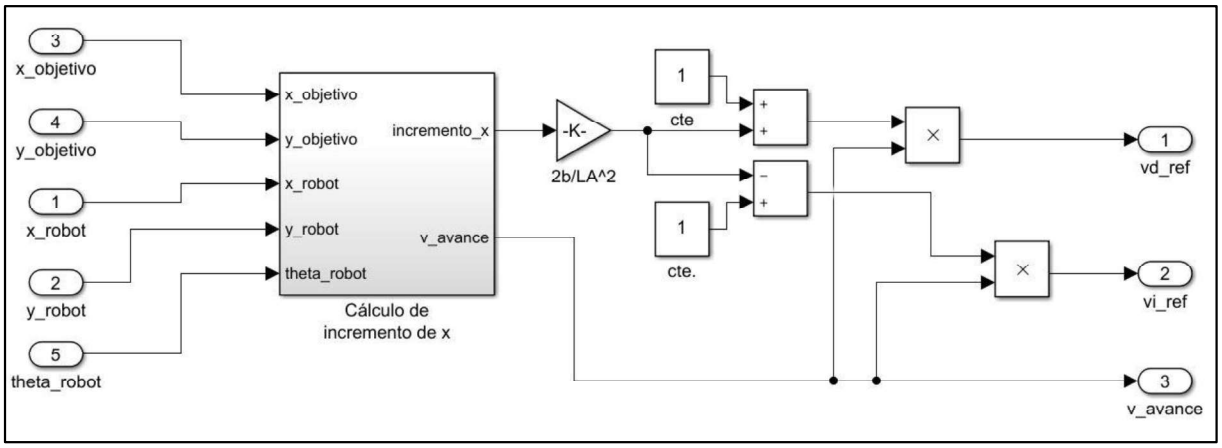


**Fig. 3.33: Esquema Simulink del bloque del control cinemático y cinemática inversa con velocidad de avance variable**

Una vez introducido este cambio dentro de los esquemas de Simulink, se volvió a simular el control de posición por persecución pura. El resultado de la simulación fue el siguiente.

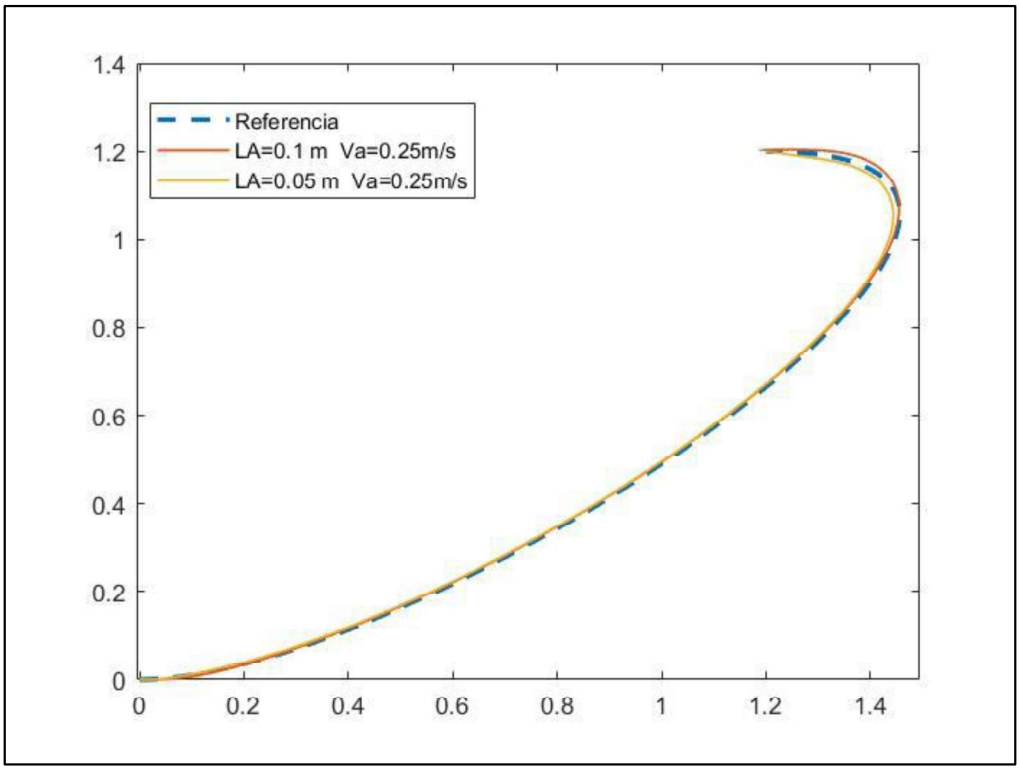


**Fig. 3.34: Gráfica con la trayectoria que realizaría el robot si la velocidad fuese variable**

Como se puede observar en la Figura 3.34, ahora el robot se detendría al llegar al final de la trayectoria, por tanto, se podía verificar que la modificación del esquema funcionaba. Por otro lado, faltaba escoger que valor de LA se escogía. Finalmente se escogió un valor de LA de 0.05

metros debido a que, aunque un valor de LA de 0.1 metros, el robot seguía prácticamente de la misma manera a la referencia, las acciones de control necesarias para el control del robot eran más suaves en algunos tramos cuando la distancia LA era de 5 centímetros, por tanto, la elección fue fijar LA a 0.05 metros.

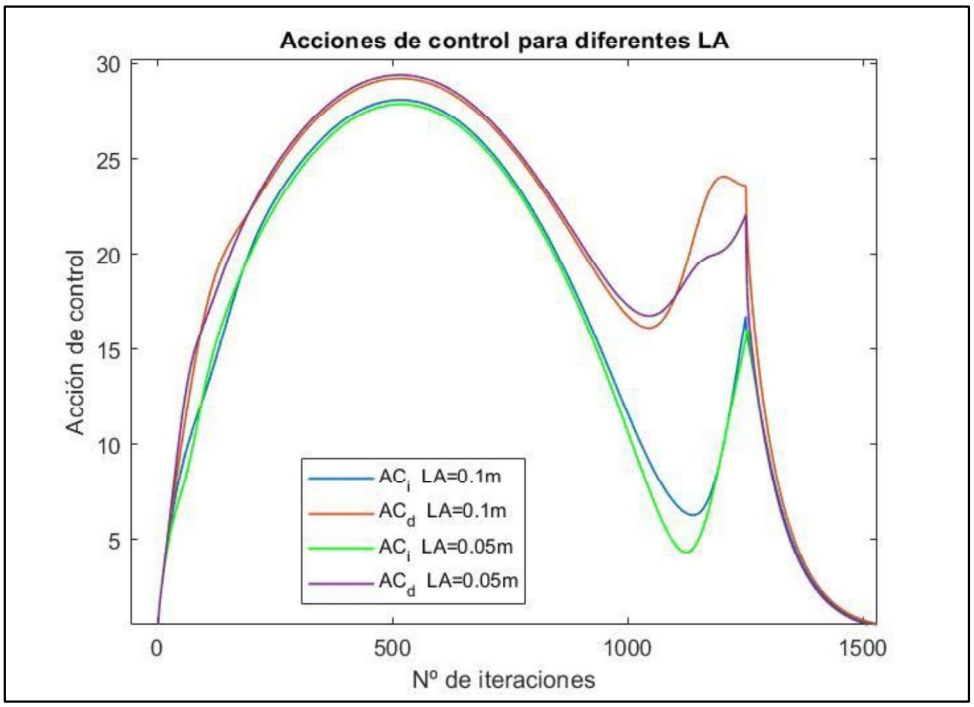


**Fig. 3.35: Gráfica con las acciones de control en función de la distancia LA.**

En la siguiente tabla se muestran los valores que se adoptaron para el control de posición empleando el algoritmo de la persecución pura.

| | |
|---|---|
| $K_{mp}$ | 13 |
| $V_{avance,\ máxima}$ | 0.25 m/s |
| $LA$ | 0.05 m |

**Tabla 3.4: Valores adoptados para el algoritmo de persecución pura**

## 3.6 Validación mediante pruebas reales

### *3.6.1 Obtención de la posición mediante visión artificial*

Llegados a este punto del trabajo, en el que todos los esquemas para el control del robot ya estaban creados y validados, volvió a ser necesaria la obtención de la posición y orientación del robot para tenerla disponible en el workspace de Matlab en tiempo real. Se dice que volvió

a ser necesaria porque ya se habían intentado, sin éxito, obtener dichos parámetros a través del envío de la lectura de los encoders mediante comunicación TCP/IP al socket creado para tal efecto en Matlab. Por ello, se procedió a implementar otro método de obtención de las coordenadas del robot en el espacio cartesiano.

Este método de obtención, se basó en la detección, por parte de una cámara cenital, de unos códigos, llamados códigos ArUco, situados sobre el robot.

Los códigos ArUco son unos códigos que pueden ser detectados mediante una cámara que es ejecutada por un programa desarrollado específicamente para realizar esta función.

Como en este trabajo no se llevó a cabo la creación del programa, ya que simplemente se utilizó como una herramienta más, no se comentarán muchos detalles acerca del mismo. No obstante, sí que se comentará alguna cosa que haya sido imprescindible para la realización de las tareas realizadas en el presente trabajo.

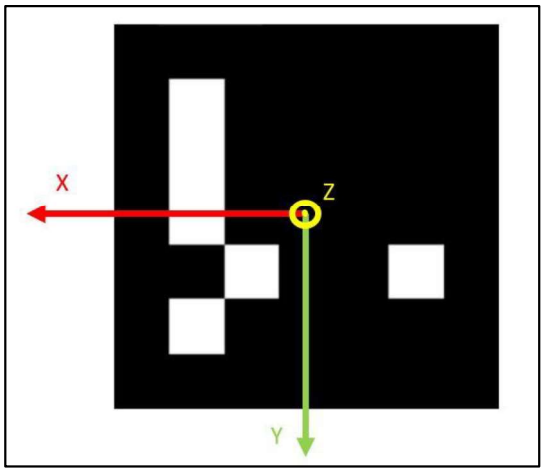


**Fig. 3.36: Marcador código ArUco con ejes**

Visto que ya se tenía la herramienta para obtener los parámetros que eran necesarios para realizar las tareas del trabajo, se procedió a desarrollar un socket que habilitase la comunicación entre Matlab y la propia cámara. La comunicación empleada volvió a ser aquella que empleaba el protocolo TCP, y como el programa de la cámara estaba configurado para que la cámara actuase como cliente, el script de Matlab donde se encontraba desarrollado el socket tenía que actuar como servidor.

```
1 -   tcp_lego=tcpip('127.0.0.1',11000,'NetworkRole','server');
2 -   fopen(tcp_lego);
3 -   pause(5);
4 -   tcp_lego.Terminator=59;
5 -   tcp_lego.Timeout=60;
6 -   i=1;
7 -   fprintf(tcp_lego,'100');
8 -   data = fscanf(tcp_lego,'%f',10);  %Se reciben los 10 datos
9 -   while data(1)~=1000
10 -      x_1(i)=data(1);
11 -      y_1(i)=data(2);
12 -      x_2=data(6);
13 -      y_2=data(7);
14 -      V = [x_1 y_1 x_2 y_2];
15 -      disp(V);
16 -      fprintf(tcp_lego,'100');
17 -      data = fscanf(tcp_lego,'%f',10);   %Se reciben los 10 datos
18 -      pause(2);
19 -  end
```

**Fig. 3.37: Captura socket para la recepción de puntos**

Con el fin de que se visualicen las conexiones entre los diferentes dispositivos y programas, se adjunta la Figura 3.38. Comentar que OpenCV es la biblioteca que emplea el programa de la cámara para detectar los marcadores con códigos ArUco.

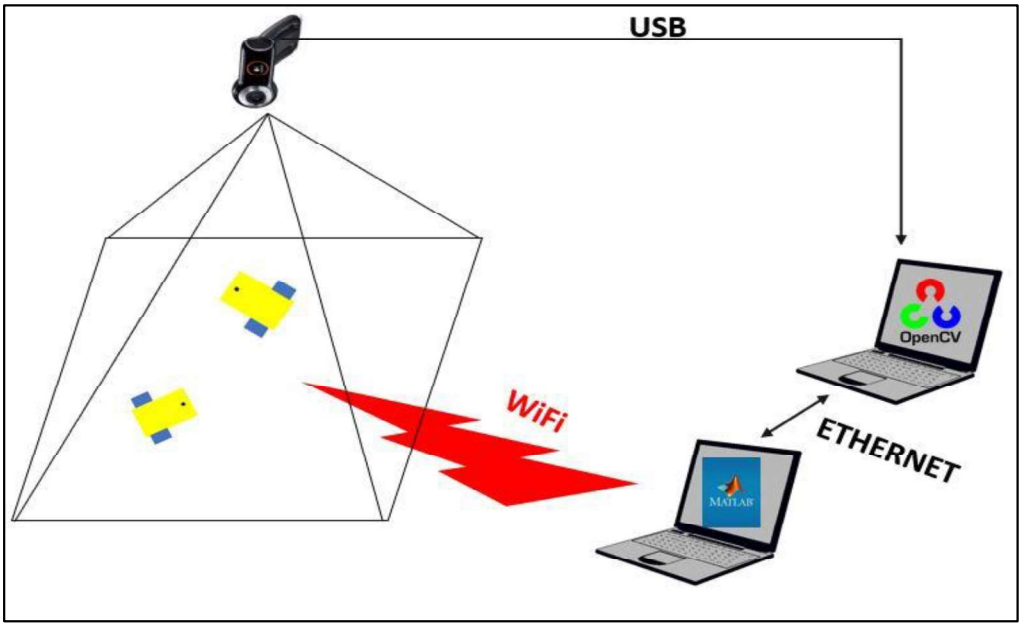


**Fig. 3.38: Esquema de la configuración para la lectura de la posición del robot**

## 3.6.2 Generación automática de trayectorias de referencia

Una vez se tenía establecida la arquitectura necesaria para la obtención de coordenadas y orientación mediante visión artificial, se pasó a realizar ciertas aplicaciones empleando esta herramienta.

### 3.6.2.1 Trayectoria de referencia entre dos posiciones

Lo primero que se trató de realizar fue el desarrollo de un socket que, a partir de las coordenadas y orientaciones de dos marcadores, crease una trayectoria de referencia basada en curvas de Bézier. Con esto lo que se perseguía era facilitar una tarea posterior en la que se iban a realizar una persecución entre robots. De esta forma, el robot que actuase como perseguidor, llegaría a ponerse detrás del robot que actuase como target en el hipotético caso de que la distancia que separase a ambos robots fuese grande. A continuación, se mostrará en forma de pseudocódigo la estructura que tiene que tener el código en Matlab del socket utilizado para esta tarea.

*Crear el objeto para establecer la comunicación entre el script de Matlab y la cámara*

*Abrir el objeto para iniciar la comunicación*

*Ordenar a la cámara que envíe datos*

*Leer datos de la cámara y asignarlos al vector* ***data***

*Asignar a* ***i*** *el valor 1*

*Asignar a* ***x_marcador1(i)*** *el primer valor de* ***data***

*Asignar a* ***y_marcador1(i)*** *el segundo valor de* ***data***

*Asignar a* ***roty_ marcador1*** *el tercer valor de* ***data***

*Asignar a* ***roty_ marcador1*** *el cuarto valor de* ***data***

*Asignar a* ***rotz_ marcador1 el*** *quinto valor de* ***data***

*Obtener* ***theta_ marcador1*** *a partir de* ***rotx_ marcador1, roty_ marcador1*** *y* ***rotz_ marcador1***

*Asignar a* ***x_marcador2*** *el sexto valor de* ***data***

*Asignar a y_****marcador2*** *el séptimo valor de* ***data***

*Asignar a* ***rotx_marcador2*** *el octavo valor de* ***data***

*Asignar a* ***roty_ marcador2*** *el noveno valor de* ***data***

*Asignar a* ***rotz_ marcador2 el*** *décimo valor de* ***data***

*Obtener* ***theta_ marcador2*** *a partir de* ***rotx_ marcador2, roty_ marcador2*** *y* ***rotz_ marcador2***

*Obtener* ***xini*** *e* ***yini*** *a partir de la función* ***dist_pto_avanzado***

*Obtener* ***xfin*** *e* ***yfin*** *a partir de la función* ***dist_seguridad***

*Obtener* ***xref*** *e* ***yref*** *a partir de la función* ***bez_2ptos_orientacion***

*Ejecutar el esquema de Simulink para que el robot se ponga en marcha*

*Mientras* ***x_marcador1(i)*** *sea distinto de 1000, hacer:*

    *Incrementar en una unidad el valor de* ***i***

    *Ordenar a la cámara que envíe datos*

    *Leer datos de la cámara y asignarlos al vector* ***data***

    *Asignar a* ***x_marcador1 (i)*** *el primer valor de* ***data***

    *Asignar a* ***y_ marcador1 (i)*** *el segundo valor de* ***data***

    *Pausar la ejecución durante 2 segundos*

*Fin Mientras*

*Cerrar la comunicación*

*Graficar* ***x_ marcador1*** *e* ***y_ marcador1***

Comentar que tanto el punto inicial, como el punto final de la trayectoria de referencia, están desplazados una cierta distancia.

Por un lado, el punto inicial se desplaza una distancia de 7 centímetros con el fin que ya ha sido comentado en el apartado de las validaciones de los esquemas, evitar que las ruedas derrapen y el robot realice de forma correcta la trayectoria de referencia.

Por otro lado, el punto final también es desplazado a una distancia de seguridad, porque si no se hiciese así, el perseguidor colisionaría con el target. Para ello se emplea una función desarrollada específicamente con tal fin llamada *dist_seguridad*.

Además de esto, comentar también que lo que se quiere decir con la línea 7 es que la posición del perseguidor es leída y almacenada para posteriormente verificar si el robot estaba realizado la trayectoria de forma correcta o no.

Con el fin de poder validar el funcionamiento real, no mediante simulación, de los esquemas desarrollados en Simulink para el control de posición del robot móvil Lego Mindstorms EV3, se realizaron distintas pruebas en las que la posición de llegada siempre era la misma y la posición y orientación de salida cambiaban.

En primer lugar, como se puede ver en las Figuras 3.39 y 3.40, la orientación de salida y de llegada eran iguales a 0°. Ejecutando el socket creado para esta tarea se obtuvieron los siguientes resultados:

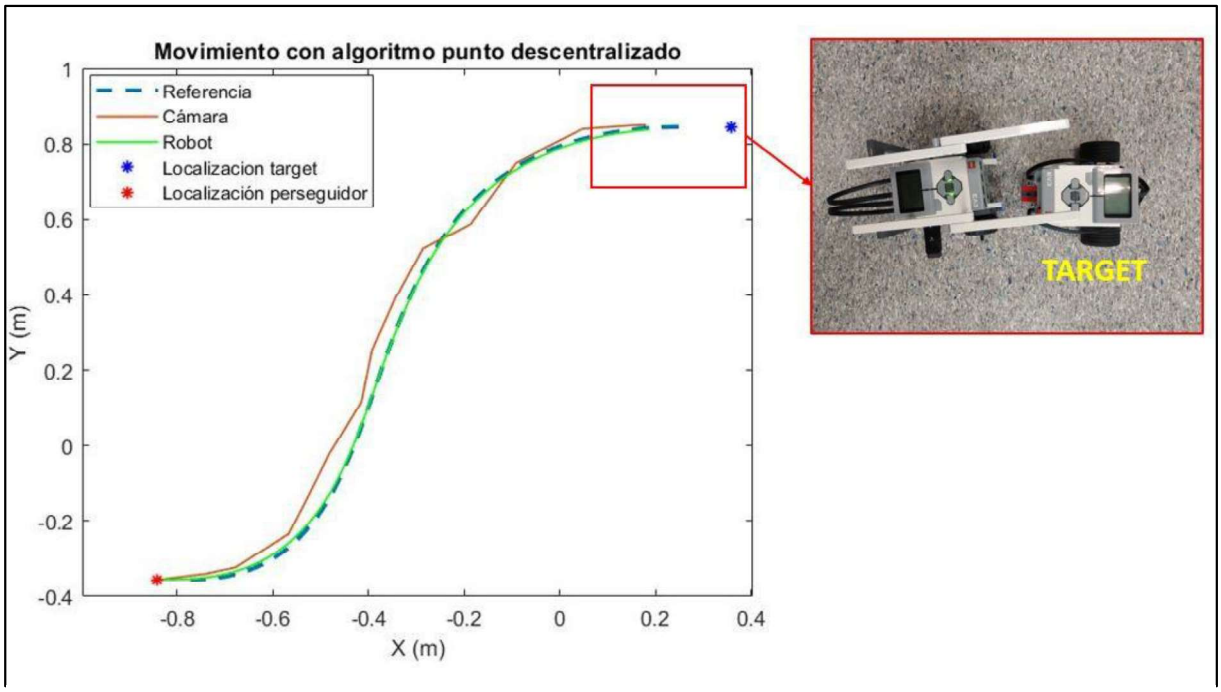


**Fig. 3.39: Gráfica del movimiento del robot perseguidor empleando el algoritmo del punto descentralizado y orientación inicial de 0°**

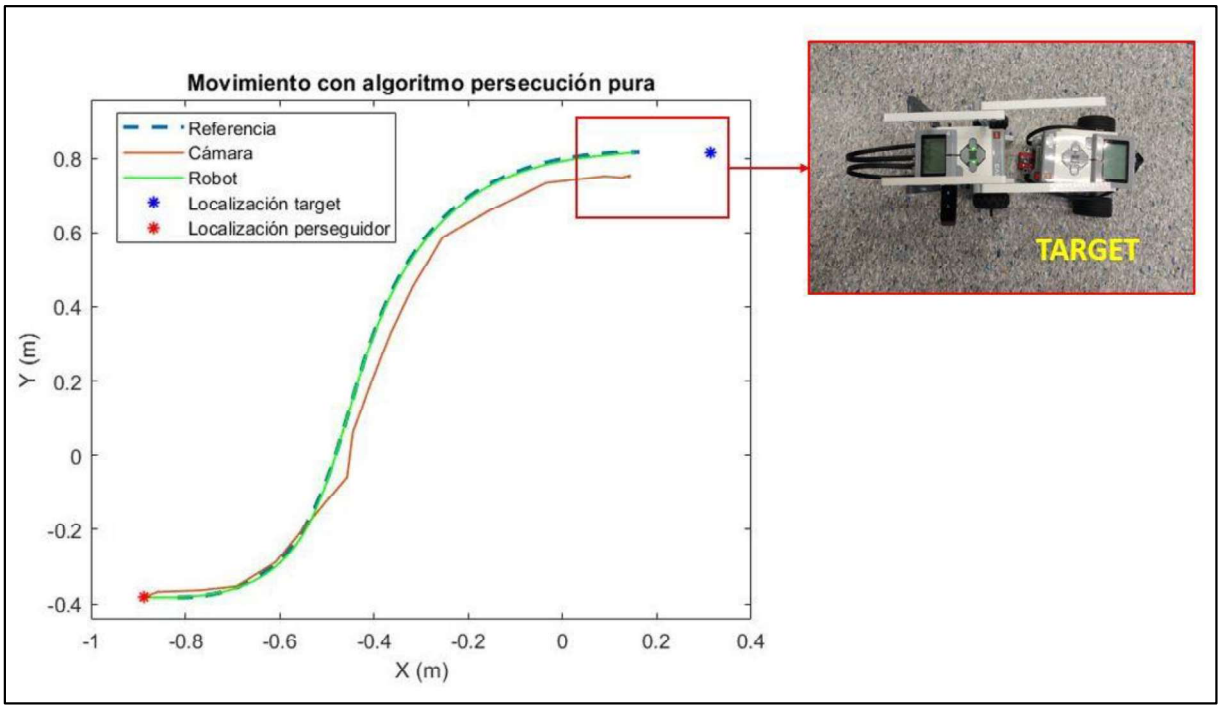


**Fig. 3.40: Gráfica del movimiento del robot perseguidor empleando el algoritmo de la persecución pura y orientación inicial de 0°**

Antes de comenzar a comentar los resultados, cabe decir que no se realizó una gráfica conjunta con el resultado para los dos algoritmos de control debido a que la cámara no tenía la suficiente precisión para enviar siempre los mismos puntos, con lo que los puntos inicial y final de cada gráfica, Figuras 3.39 y 3.40, no poseían las mismas coordenadas, y las trayectorias, aunque son similares, no empezaban y acababan en los mismos puntos. Además, en las fotografías de los robots, no aparecen los marcadores empleados para la lectura de la posición y orientación debido a que su tamaño impedía ver a los robots.

Comentando ya los resultados obtenidos, se puede ver que, a pesar de haber empleado algoritmos de control distintos, en ambos casos el robot llegó a su destino de forma correcta, sin apenas error.

Comentar también que se graficó la posición del robot perseguidor enviada por la cámara durante la ejecución del movimiento para de esta forma demostrar que la lectura de los marcadores del robot no da la posición exacta del mismo, sino que, como se podrá observar en las siguientes figuras, da una aproximación. Las razones por las que sucede esto se deben fundamentalmente a que es prácticamente imposible que la cámara de la posición sin error alguno, y a que el marcador se encuentra en movimiento con el robot, por lo que es relativamente sencillo que este cambie de posición durante la duración del movimiento del robot. Hay que tener presente que las dimensiones del robot son reducidas, y que un error de un par de centímetros tiene como consecuencia que el robot no llegue a al punto exacto que se desea, sino que llega a la posición aproximada que la cámara detecta.

En segundo lugar, como se puede ver en las Figuras 3.41 y 3.42, la orientación de salida era de 180° y de llegada de 0°. Ejecutando el socket, se obtuvieron los siguientes resultados:

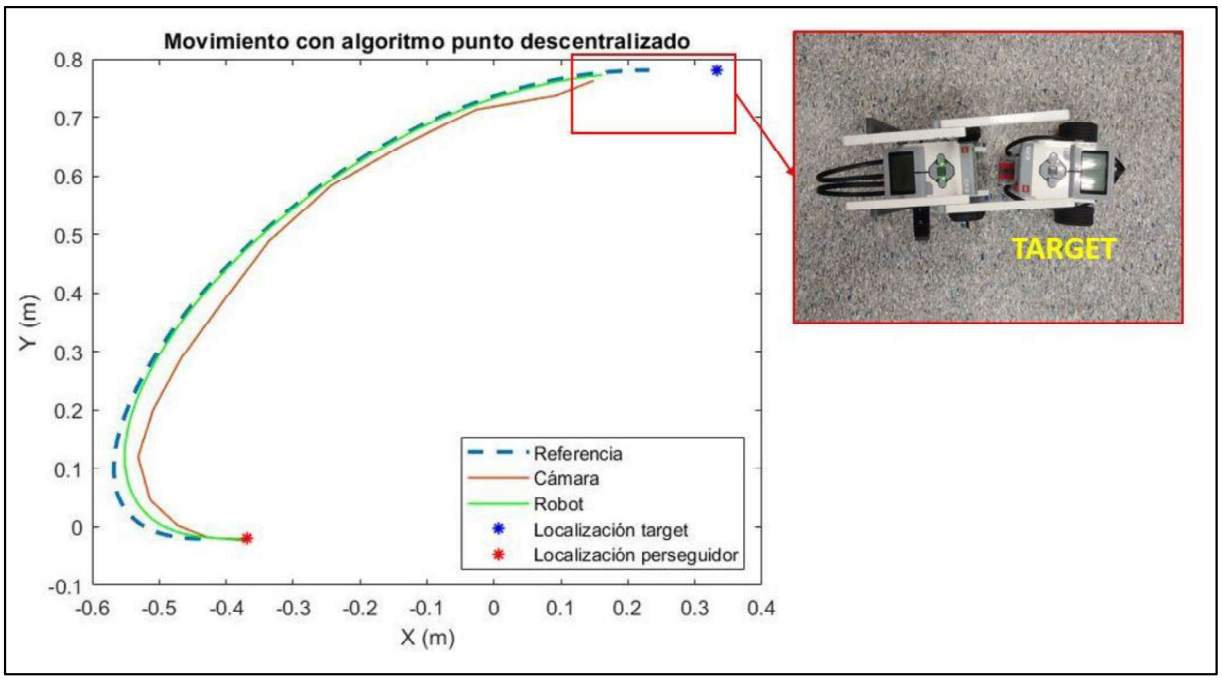


**Fig. 3.41: Gráfica del movimiento del robot perseguidor empleando el algoritmo del punto descentralizado y orientación inicial de 180°**

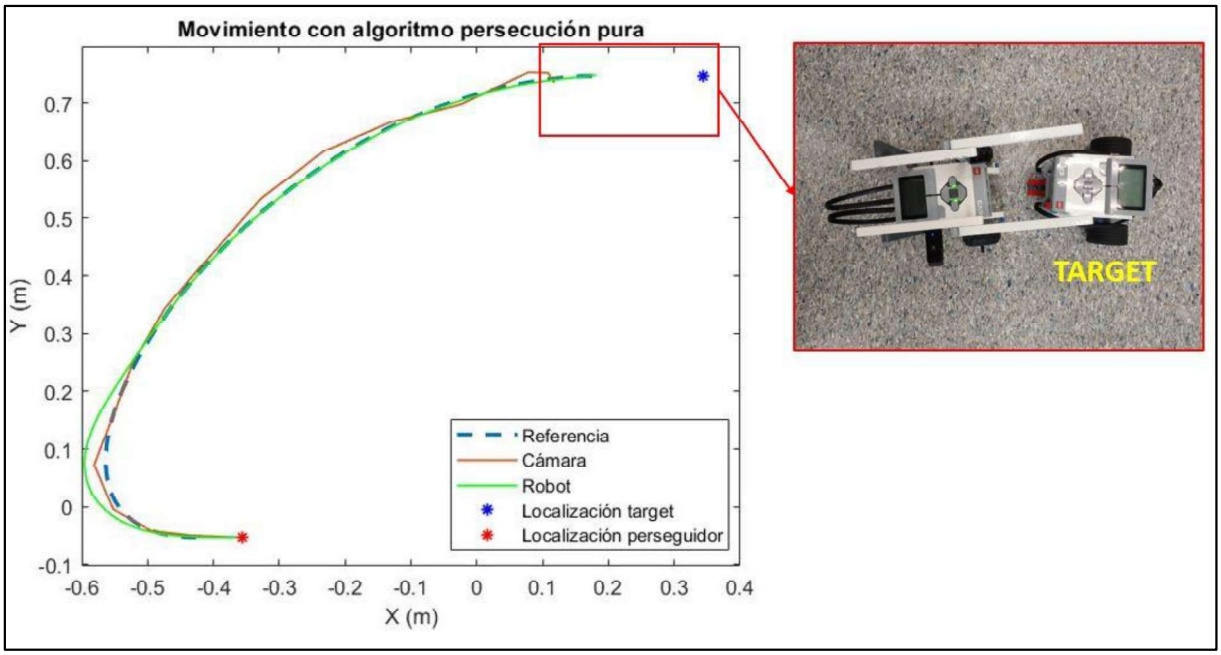


**Fig. 3.42: Gráfica del movimiento del robot perseguidor empleando el algoritmo de la persecución pura y orientación inicial de 180°**

Como se puede observar en las Figuras 3.41 y 3.42, el robot llegaba a su destino de forma correcta empleando cualquiera de los dos algoritmos de navegación desarrollados.

Por último, se cambió la orientación de salida del robot a 90° dejando la de llegada en 0°. Los resultados obtenidos fueron los siguientes:

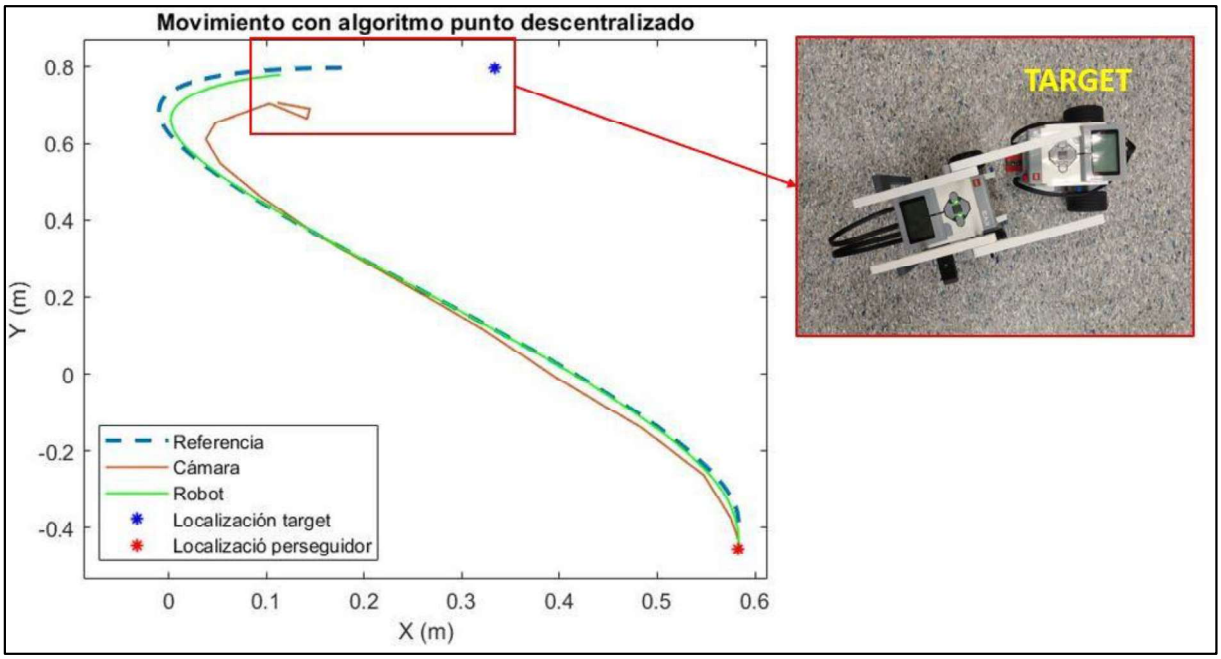


**Fig. 3.43: Gráfica del movimiento del robot perseguidor empleando el algoritmo del punto descentralizado y orientación inicial de 90°**

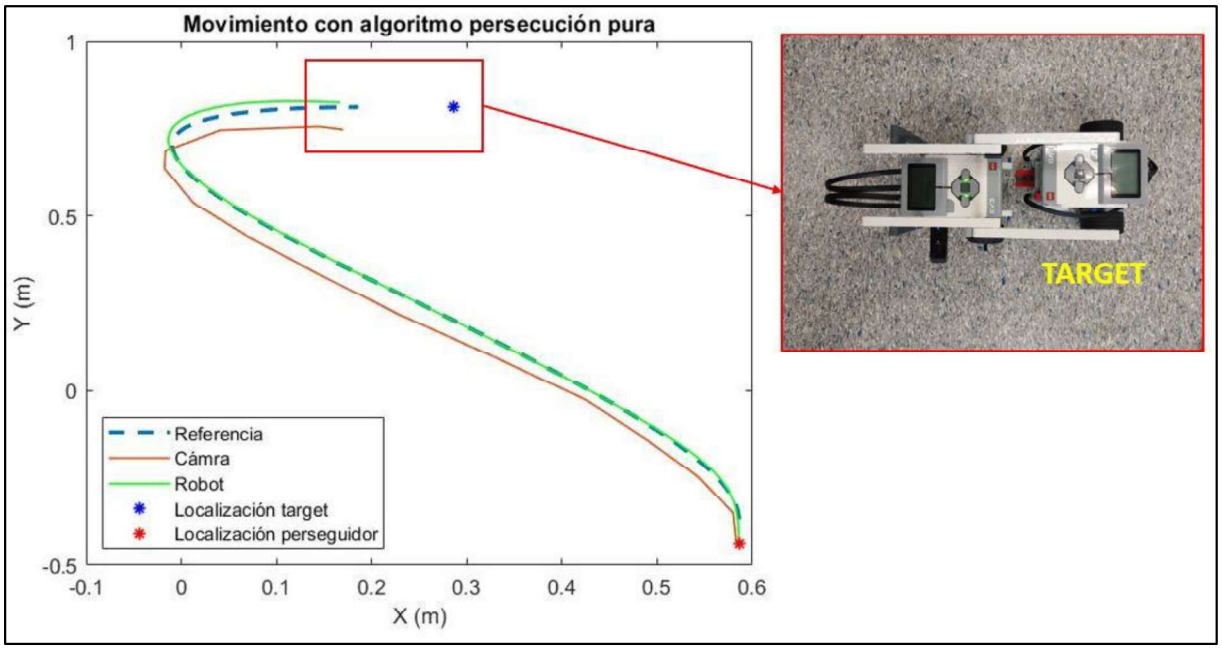


**Fig. 3.44: Gráfica del movimiento del robot perseguidor empleando el algoritmo de la persecución pura y orientación inicial de 90°**

Como se puede observar, empleando el algoritmo de persecución pura, Figura 3.44, el robot llegó con la orientación final deseada, mientras que, empleando el algoritmo del punto descentralizado, Figura 3.43, la orientación final era distinta de 0°. A pesar de esto último, se puede aceptar que el robot llegue de esta forma al destino, puesto que como se puede ver en la imagen de los robots reales de la Figura 3.43, el error cometido en lo que se refiere a la posición final del robot perseguidor es muy pequeño, prácticamente está alineado con el target.

Para finalizar, cabe decir que a pesar de que la cámara podía enviar posiciones que no eran del todo precisas, los resultados obtenidos fueron buenos, puesto que, viendo las figuras mostradas anteriormente, en la mayoría de los casos el perseguidor se situó justo detrás del target.

#### *3.6.2.2 Trayectoria de referencia evitando obstáculos*

Otra de las tareas realizadas fue la obtención de una trayectoria de referencia que evitase ciertos obstáculos presentes en el espacio de trabajo del robot. Para ello, se empleó un algoritmo basado en campos potenciales[2] que, dados el punto de partida del robot, el punto de llegada, el punto donde estaba situado cada obstáculo y las dimensiones de estos, generaba la trayectoria que haría que el robot fuese desde el punto de inicio hasta punto de llegada esquivando todos los obstáculos presentes.

Como las coordenadas de los puntos citados anteriormente se debían pasar como parámetro de forma manual a la función que generaba la trayectoria, se aprovechó que se tenía la posibilidad de utilizar la cámara para detectar la posición y se creó un socket en Matlab que, de forma automática, generase la trayectoria de referencia y ejecutase el esquema de Simulink con el control de posición del robot.

Con el fin de aclarar que era lo que debía realizar el socket generado para esta tarea, se escribe el siguiente pseudocódigo:

*Crear el objeto para establecer la comunicación entre el script de Matlab y la cámara*

*Abrir el objeto para iniciar la comunicación*

*Ordenar a la cámara que envíe datos*

*Leer datos de la cámara y asignarlos al vector* ***data***

*Asignar a* ***i*** *el valor 1*

*Asignar a* ***x_robot(i)*** *el primer valor de* ***data***

[2] Este algoritmo no fue desarrollado en el trabajo, simplemente se utilizó como una herramienta de trabajo más.

*Asignar a **y_robot(i)** el segundo valor de **data***

*Asignar a **rotx_ robot** el cuarto valor de **data***

*Asignar a **roty_ robot** el cuarto valor de **data***

*Asignar a **rotz_ robot** el quinto valor de **data***

*Obtener **theta_ robot** a partir de **rotx_ robot, roty_ robot** y **rotz_ robot***

*Asignar a **x_fin** el sexto valor de **data***

*Asignar a y_**fin** el séptimo valor de **data***

*Asignar a **x_obs1** el decimoprimer valor de **data***

*Asignar a y_**obs1** el s decimosegundo de **data***

*Asignar a **x_obs2** el decimosexto valor de **data***

*Asignar a y_**obs2** el decimoséptimo valor de **data***

*Obtener **xini** e **yini** a partir de la función **dist_pto_avanzado***

*Obtener **xfin** e **yfin** a partir de la función **dist_seguridad***

*Asignar el **largo** y **ancho** de los obstáculos*

*Obtener **xref** e **yref** a partir del algoritmo de los campos potenciales*

*Ejecutar el esquema de Simulink para que el robot se ponga en marcha*

*Mientras **x_robot(i)** sea distinto de 1000, hacer*

    *Incrementar en una unidad el valor de **i***

    *Ordenar a la cámara que envíe datos*

    *Leer datos de la cámara y asignarlos al vector **data***

    *Asignar a **x_robot (i)** el primer valor de **data***

    *Asignar a **y_ robot (i)** el segundo valor de **data***

    *Pausar la ejecución durante 2 segundos.*

*Fin Mientras*

*Cerrar la comunicación*

*Graficar **x_ robot** e **y_ robot***

En esta tarea, el punto inicial de la trayectoria también se adelantó respecto a la posición del robot por la razón que ya ha sido tratada en tareas anteriores. Además, por las mismas razones que ya han sido expuestas, el punto final se retrasa a una distancia de 20 centímetros.

Llegados a este punto, lo único que faltaba por realizar era verificar que el robot seguía la trayectoria generada y evitaba de esta manera los obstáculos.

Se realizó una prueba para verificar de nuevo que, a pesar de tratarse de una trayectoria totalmente diferente, los algoritmos de navegación desarrollados en el trabajo cumplían con su cometido:

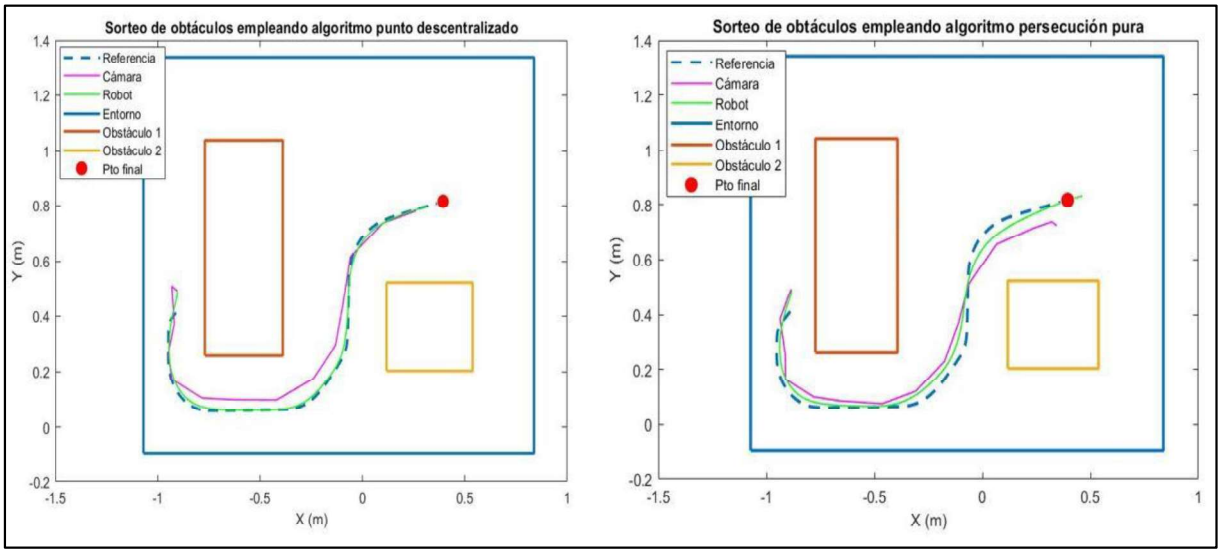


**Fig. 3.45: Gráficas con la referencia generada y la trayectoria seguida por el robot móvil empleando los diferentes algoritmos de control**

Como se puede observar en las Figura 3.45, el robot cumple prácticamente a la perfección la referencia generada evitando así los dos obstáculos que se encuentran en el espacio de trabajo. A continuación, se muestra una secuencia de imágenes del robot mientras realiza el movimiento, Figura 3.46.

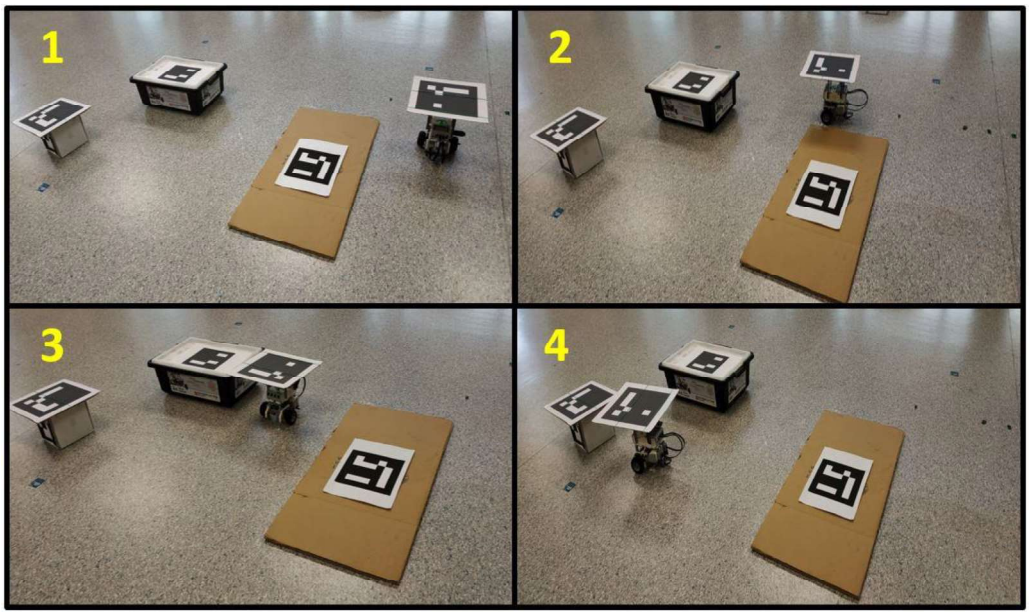


**Fig. 3.46: Secuencia del movimiento del robot móvil evitando obstáculos**

## 3.7 Persecución entre robots

Una vez se tenía la certeza de que los algoritmos de navegación desarrollados durante el trabajo cumplían con su objetivo, se procedió a implementar un algoritmo con el fin de que un robot, robot perseguidor, persiguiese a otro, target.

Para poder llevar a cabo esta tarea se tuvo que tener en cuenta que los esquemas de Simulink a ejecutar de forma externa sobre los robots necesitaban estar configurados con los parámetros de identificación de cada robot, es decir, con la dirección IP y el ID.

Por otro lado, debido a que una vez estaba en ejecución uno de los esquemas de Simulink, el otro no podía ser configurado, se decidió abrir dos sesiones de Matlab, y con ello dos sesiones de Simulink, para que, de esta manera, cuando se ejecutase el esquema que controla al robot perseguidor, se podía entrar en el otro esquema y configurarlo con la IP y el ID del target.

Una vez se tenían abiertas las dos sesiones de Simulink, la estrategia a seguir fue:

1. Generar una trayectoria de referencia basada en curvas de Bézier para que el target la siguiese.
2. Configurar el esquema de Simulink, de una de las dos sesiones, con el control por persecución pura para que se ejecutase sobre el robot perseguidor
3. Ejecutar el socket desarrollado para realizar esta tarea.
4. Configurar el esquema de Simulink de la otra sesión para ser ejecutado sobre el target.
5. Ejecutar el esquema de Simulink que controla al target. El tipo de algoritmo de control a emplear para controlar el target es indiferente, se puede emplear cualquiera de los dos desarrollados durante el trabajo.

Como se puede apreciar en la estrategia que se siguió, se ejecutó el esquema de Simulink del control por persecución pura en el robot perseguidor. Esto se hizo de esta manera porque cada cierto tiempo se le comunicaba al robot perseguidor la posición del target. El código implementado en el socket de comunicación empleado para esta tarea se describe en cierta manera con el pseudocódigo siguiente:

| |
|---|
| *Crear el objeto para establecer la comunicación entre el script de Matlab y la cámara* |
| *Abrir el objeto para iniciar la comunicación* |
| *Ordenar a la cámara que envíe datos* |
| *Leer datos de la cámara y asignarlos al vector* ***data*** |
| *Asignar a* ***i*** *el valor 1* |
| *Asignar a* ***x_target(i)*** *el primer valor de* ***data*** |

*Asignar a **y_target(i)** el segundo valor de **data***

*Asignar a **x_perseguidor(i)** el sexto valor de **data***

*Asignar a **x_perseguidor(i)** el séptimo valor de **data***

*Asignar a **rotx_target** el tercer valor de **data***

*Asignar a **roty_target** el cuarto valor de **data***

*Asignar a **rotz_target** el quinto valor de **data***

*Obtener **theta_target** a partir de **rotx_target, roty_target** y **rotz_target***

*Obtener **xref** e **yref** a partir de la función **dist_seguridad***

*Asignar a **xref_ppura** e **yref_ppura** el valor de **xref** e **yref** respectivamente*

*Actualizar el esquema de Simulink*

*Ejecutar el esquema de control por persecución pura*

*Mientras **x_target(i)** sea distinto de 1000, hacer:*

    *Incrementar en una unidad el valor de **i***

    *Ordenar a la cámara que envíe datos*

    *Leer datos de la cámara y asignarlos al vector **data***

    *Asignar a **x_target(i)** el primer valor de **data***

    *Asignar a **y_target(i)** el segundo valor de **data***

    *Asignar a **x_perseguidor(i)** el sexto valor de **data***

    *Asignar a **x_perseguidor(i)** el séptimo valor de **data***

    *Asignar a **rotx_target** el tercer valor de **data***

    *Asignar a **roty_target** el cuarto valor de **data***

    *Asignar a **rotz_target** el quinto valor de **data***

    *Obtener **theta_target** a partir de **rotx_target, roty_target** y **rotz_target***

    *Obtener **xref** e **yref** a partir de la función **dist_seguridad***

    *Asignar a **xref_ppura** e **yref_ppura** el valor de **xref** e **yref** respectivamente*

    *Actualizar el esquema de Simulink*

    *Pausar la ejecución durante 1 segundo*

*Fin Mientras*

*Cerrar la comunicación*

*Graficar **x_target, y_target, x_perseguidor** e **y_perseguidor***

Cabe decir que, para la creación de la referencia empleada por el perseguidor, se empleó la función *dist_seguridad* ya empleada en otras tareas, para desplazar 15 centímetros el punto de referencia al que debía ir el perseguidor. Se empleó esta función porque así se aseguraba que el robot perseguidor no colisionase con el target una vez que este finalizase su recorrido.

Por otro lado, la referencia enviada al perseguidor constaba de dos parámetros, *xref_ppura* e *yref_ppura*, ambos tratados como constantes dentro el esquema de Simulink. Debido a que era necesario enviar estas constantes al esquema de Simulink, puesto que, sino el perseguidor no realizaría ningún movimiento, se tuvo que hacer uso de las funciones para actualizar parámetros de Simulink desde el script de Matlab. El código necesario para el cálculo y la actualización de las referencias fue el siguiente:

```
[xref(i),yref(i)]=dist_seguridad(x_target(i),y_target(i),theta_target,0.15)
xref_ppura.Value = xref(i);     %Coordenada X de referencia
yref_ppura. Value = yref(i);     %Coordenada Y de referencia
set_param('control_ppura_dife_REAL','SimulationCommand','update');
```

En la función *dist_seguridad* se introducen las coordenadas y la orientación, *x_target(i), y_target(i)* y *theta_target* leídas por la cámara, del target y una distancia que será la distancia de seguridad, en este caso de 15 centímetros. Por otro lado, de la función *set_param()* lo único que se ha de modificar es el primer parámetro, que se corresponde con el nombre del esquema de Simulink donde se quiere actualizar la referencia, *'control_ppura_dife_REAL'*.

También hay que decir que anteriormente a la ejecución del socket de comunicación, se crearon los objetos *xref_ppura* e *yref_ppura,* puesto que, si no hubiesen existido, al intentar ejecutar la función *set_param()* se hubiese obtenido un error de compilación. Para crear estas variables, se escribió lo que se muestra a continuación en la ventana de comandos de Matlab:

```
>> xref_ppura=Simulink.Parameter;
>> yref_ppura=Simulink.Parameter;
>> xref_ppura.CoderInfo.StorageClass='ExportedGlobal';
>> yref_ppura.CoderInfo.StorageClass='ExportedGlobal';
```

Finalmente, se llevaron a cabo diferentes persecuciones para comprobar que el robot perseguidor perseguía al target. Los resultados obtenidos se muestran en las siguientes figuras.

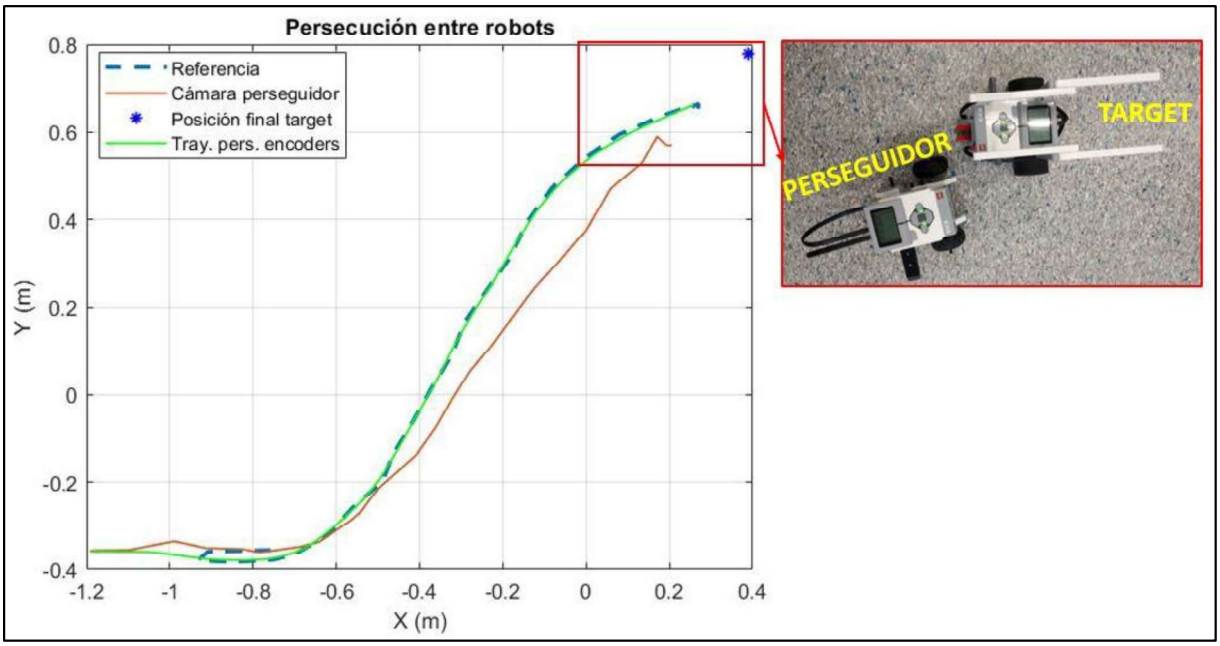


**Fig. 3.47: Gráfica con la representación de las posiciones de los robots durante la persecución**

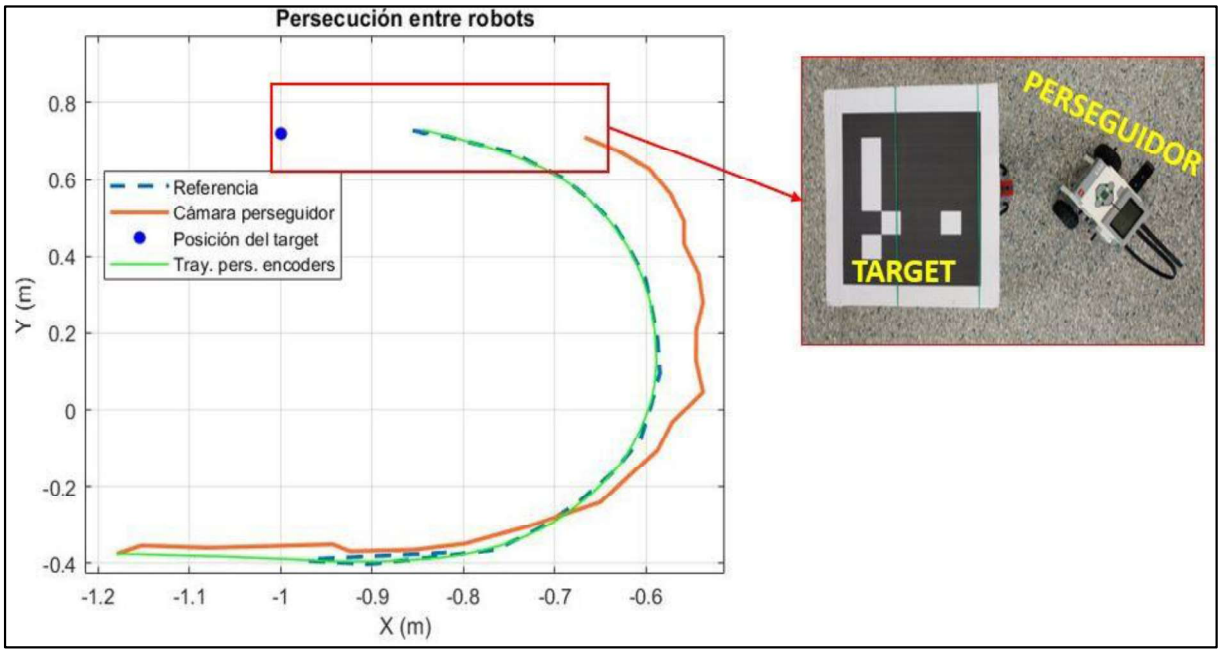


**Fig. 3.48: Gráfica con la representación de las posiciones de los robots durante la persecución**

Lo que debería suceder en las Figuras 3.47 y 3.48 es que la trayectoria del perseguidor captada por la cámara se asemeje lo máximo posible a la referencia. Sin embargo, como se puede observar, la citada trayectoria difiere unos cuantos centímetros de la referencia en ciertos puntos. Esto se debe a que la referencia se calcula a partir de las coordenadas y la orientación

del marcador que está situado sobre el target, y como ya ha sido comentado, la cámara no da estos datos de forma precisa, sino que existe un cierto error con el que hay que convivir.

Además, también hay que tener en cuenta que el robot perseguidor no realiza un movimiento continuo, sino que cada vez que recibe un nuevo punto de referencia, las acciones de control aumentan con el fin de que el robot llegue a su objetivo. Este hecho, puede ocasionar que el marcador que se encuentra sobre el perseguidor se deslice algún centímetro, con lo que se incrementaría el error en la medición.

Por último, aunque calculando la trayectoria a través de los datos de los encoders sí que se persiga a la referencia, ver Figuras 3.47 y 3.48, la realidad marca que el robot no llega realmente a situarse en un punto situado a una distancia de seguridad justo detrás del target. No obstante, para tratarse de un sistema en el que se sabe de antemano que hay un error en la obtención de la posición y orientación del target, se puede dar como válido que el perseguidor siga de la forma que sigue al target, ya que, a pesar de todo, al final consigue su cometido.

# 4 CONCLUSIONES

Como conclusión del trabajo realizado, se puede afirmar que se ha conseguido cumplir los objetivos marcados al inicio de este. Esto se dice debido a que:

- Se ha realizado un pequeño análisis de las diferentes configuraciones cinemáticas que pueden presentar los robots móviles para posteriormente escoger la que más convenía para la realización del trabajo.
- Se ha empleado el pack de Lego Mindstorms EV3, con el que se ha logrado obtener una gran familiaridad, para construir dos robots móviles con configuración diferencial.
- Se han estudiado diferentes alternativas para el control cinemático de los robots.
- Se ha empleado Simulink para implementar los algoritmos de navegación de control estudiados, control de trayectorias por punto descentralizado y control de caminos por persecución pura.
- Se han solventado los fallos presentes en la comunicación entre los robots y el ordenador de control.
- Se han generado aplicaciones con el fin de generar trayectorias de referencia basadas en curvas de Bézier.
- Se consiguió obtener la posición y orientación de los robots mediante visión artificial.
- Se han validado experimentalmente los distintos algoritmos de navegación de los robots móviles.

Todo lo enunciado anteriormente es lo que se ha logrado en este trabajo, sin embargo, esto no es más que un primer paso hacia futuros proyectos donde los robots a emplear presenten configuraciones cinemáticas diferentes o, en caso de presentar la misma configuración que los robots de este trabajo, presenten características que le permitan realizar tareas más complejas.

# 5 REFERENCIAS

## 5.1 Referencias bibliográficas del texto

**[1]** https://es.wikipedia.org/wiki/Robot (1 de junio de 2019)

**[2]** https://dle.rae.es/?id=WYRlhzm (1 de junio de 2019)

**[3]** https://es.wikipedia.org/wiki/Curva_de_B%C3%A9zier (3 de junio de 2019)

**[4]** https://es.wikipedia.org/wiki/Bluetooth (3 de junio de 2019)

**[5]** https://es.wikipedia.org/wiki/Protocolo_de_control_de_transmisi%C3%B3n (3 de junio de 2019)

Material docente asignatura Laboratorio de Automatización y Control (Ángel Valera,2019)

Introduction to Autonomous Mobile Robots (R. siegwart, I.R. Nourbakshs, 2004)

## 5.2 Referencias bibliográficas figuras

**[1]** http://www.directindustry.es/prod/euroimpianti-skilled/product-19184-479057.html

**[2]** http://www.directindustry.es/prod/schunk-gmbh-co-kg/product-7038-1841415.html

**[3]** https://www.philips.es/c-p/FC8802_01/robot-aspirador

**[4]** https://www.mobile-industrial-robots.com/es/products/mir200/

**[5]** Material docente asignatura Laboratorio de Automatización y Control (Ángel Valera,2019)

**[6]** Material docente asignatura Laboratorio de Automatización y Control (Ángel Valera,2019)

**[7]** https://electropro.pe/index.php?route=product/product&product_id=503

**[8]** Material docente asignatura Laboratorio de Automatización y Control (Ángel Valera,2019)

**[9]** https://www.feriadelasciencias.unam.mx/anteriores/feria23/feria276_01_robo_fastcar.pdf

**[10]**Material docente asignatura Laboratorio de Automatización y Control (Ángel Valera,2019)

**[11]** https://www.robotnik.es/robots-moviles/rb-car/

**[12]**https://www.electricbricks.com/lego-piezas-lego-technic-electrico-9709-mindstorms-rcx-completo-con-conector-alimentacion-p-1015.html

**[13]** https://www.steinelager.de/en/set/9841-1/nxt-intelligent-brick

**[14]** http://shop.innovadidactic.com/index.php?id_product=247&controller=product

# DOCUMENTO II:

# PRESUPUESTO

# ÍNDICE DEL PRESUPUESTO

# 1. PRECIOS MATERIALES Y MANO DE OBRA

En primer lugar, se muestra una lista con el coste de los materiales, hardware y software, necesarios para la realización del trabajo.

| | Precio (€) |
|---|---|
| Ordenador portátil HP OMEN 15-CE004NS | 799,99 |
| Ratón inalámbrico genérico | 10,00 |
| Logitech QuickCam Pro-9000 | 42,95 |
| Cable USB de 3 m de longitud | 6,86 |
| Pack education Lego Mindstorms EV3 | 492,00 |
| Router TP- Link-Pocket MR-3020 | 25,99 |
| Router Conceptronic CB300RS4 | 27,82 |
| Adaptador WiFi Netgear N150 | 12,56 |
| Adaptador WiFi Edimax EW-7611 | 13,89 |
| Adaptador WiFi TP-Link TL-WN725N | 5,72 |
| Licencia de 1 año de Matlab y Simulink | 800,00 |
| Licencia de 1 año del Pack Microsoft Office 2019 | 119,00 |

**Tabla 1: Tabla con los precios de los materiales a emplear durante el trabajo**

En segundo lugar, hay que especificar que el coste por hora de un graduado en Ingeniería en Tecnologías Industriales es de 28.34 €/h. Este coste incluye la cotización a la seguridad social.

| | Coste (€/h) |
|---|---|
| Graduado en Ingeniería en Tecnologías Industriales | 28,34 |

**Tabla 2: Precio mano de obra**

# 2. PRECIOS DESCOMPUESTOS

| Nº | | Descripción de las unidades de obra | Medición | Precio | Importe |
|---|---|---|---|---|---|
| **01** | | **Montaje de los robots móviles e instalación y configuración de Matlab/Simulink** | | | |
| | Ud. | Ordenador portátil HP OMEN 15-CE004NS | 1 | 799,99 € | 799,99 € |
| | Ud. | Ratón inalámbrico genérico | 1 | 10,00 € | 10,00 € |
| | Ud. | Licencia de 1 año de Matlab y Simulink | 1 | 800,00 € | 800,00 € |
| | Ud. | Pack education Lego Mindstorms EV3 | 2 | 492,00 € | 984,00 € |
| | Ud. | Router TP- Link-Pocket-MR-3020 | 1 | 25,99 € | 25,99 € |
| | Ud. | Router Conceptronic CB-300RS4 | 1 | 27,82 € | 27,82 € |
| | Ud. | Adaptador WiFi Netgear N150 | 1 | 12,56 € | 12,56 € |
| | Ud. | Adaptador WiFi Edimax EW7611 | 1 | 13,89 € | 13,89 € |
| | Ud. | Adaptador WiFi TP-Link TL-WN725N | 1 | 5,72 € | 5,72 € |
| | h | Graduado en Ing. en Tecnologías Industriales | 20 | 28,34 € | 570,78 € |
| | | Medios auxiliares | 2% | 3.250,75 € | 65,01 € |
| | | Costes directos complementarios | 3% | 3.315,76 € | 99,47 € |
| | | **TOTAL** | | | **3.415,23 €** |
| **02** | | **Programación de funciones para la generación de trayectorias** | | | |
| | h | Graduado en Ing. en Tecnologías Industriales | 30 | 28,34 € | 856,16 € |
| | | Medios auxiliares | 2% | 856,16 € | 17,12 € |
| | | Costes directos complementarios | 3% | 873,29 € | 26,20 € |
| | | **TOTAL** | | | **899,49 €** |
| **03** | | **Comunicación de la posición y orientación mediante el bloque TCP/IP** | | | |
| | h | Graduado en Ing. en Tecnologías Industriales | 30 | 28,34 € | 856,16 € |
| | | Medios auxiliares | 2% | 856,16 € | 17,12 € |
| | | Costes directos complementarios | 3% | 873,29 € | 26,20 € |
| | | **TOTAL** | | | **899,49 €** |

| Nº | | Descripción de las unidades de obra | Medición | Precio | Importe |
|---|---|---|---|---|---|
| **04** | | **Generación de esquemas con Simulink para el control de posición del robot móvil Lego Mindstorms EV3** | | | |
| | h | Graduado en Ing. en Tecnologías Industriales | 40 | 28,34 € | 1.141,55 € |
| | | Medios auxiliares | 2% | 1.141,55 € | 22,83 € |
| | | Costes directos complementarios | 3% | 1.164,38 € | 34,93 € |
| | | **TOTAL** | | | **1.199,32 €** |
| **05** | | **Ajuste de constantes y validación de esquemas mediante simulación** | | | |
| | h | Graduado en Ing. en Tecnologías Industriales | 25 | 28,34 € | 713,47 € |
| | | Medios auxiliares | 2% | 713,47 € | 14,27 € |
| | | Costes directos complementarios | 3% | 727,74 € | 21,83 € |
| | | **TOTAL** | | | **749,57 €** |
| **06** | | **Validación de esquemas mediante pruebas experimentales** | | | |
| | Ud. | Logitech QuickCam Pro-9000 | 1 | 42,95 € | 42,95 € |
| | Ud. | Cable USB de 3 m de longitud | 2 | 6,86 € | 13,72 € |
| | h | Graduado en Ing. en Tecnologías Industriales | 60 | 28,34 € | 1.712,33 € |
| | | Medios auxiliares | 2% | 1.769,00 € | 35,38 € |
| | | Costes directos complementarios | 3% | 1.804,38 € | 54,13 € |
| | | **TOTAL** | | | **1.858,51 €** |
| **07** | | **Persecución de robots** | | | |
| | h | Graduado en Ing. en Tecnologías Industriales | 35 | 28,34 € | 998,86 € |
| | | Medios auxiliares | 2% | 998,86 € | 19,98 € |
| | | Costes directos complementarios | 3% | 1.018,84 € | 30,57 € |
| | | **TOTAL** | | | **1.049,40 €** |

| Nº | | Descripción de las unidades de obra | Medición | Precio | Importe |
|---|---|---|---|---|---|
| **08** | | **Redacción de documentos** | | | |
| | Ud. | Licencia de 1 año del Pack Microsoft Office 2019 | 1 | 119,00 € | 119,00 € |
| | h | Graduado en Ing. en Tecnologías Industriales | 60 | 28,34 € | 1712,33 € |
| | | Medios auxiliares | 2% | 1.831,33 € | 36,63 € |
| | | Costes directos complementarios | 3% | 1.867,96 € | 56,04 € |
| | | **TOTAL** | | | **1.923,99 €** |

# 3. PRECIOS UNITARIOS

| Nº | Descripción de las unidades de obra | Precio unitario |
|---|---|---|
| 01 | Montaje de los robots móviles e instalación y configuración de Matlab/Simulink | 3.415,23 € |
| 02 | Programación de funciones para la generación de trayectorias | 899,49 € |
| 03 | Comunicación de la posición y orientación mediante el bloque TCP/IP | 899,49 € |
| 04 | Generación de esquemas con Simulink para el control de posición del robot móvil Lego Mindstorms EV3 | 1.199,32 € |
| 05 | Ajuste de constantes y validación de esquemas mediante simulación | 749,57 € |
| 06 | Validación de esquemas mediante pruebas experimentales | 1.858,51 € |
| 07 | Persecución de robots | 1.049,40 € |
| 08 | Redacción de documentos | 1.923,99 € |

# 4.PRESUPUESTO BASE DE LICITACIÓN

**Unidades de Obra:**

Montaje de los robots móviles e instalación y configuración de Matlab/Simulink……3.415,23 €

Programación de funciones para la generación de trayectorias……899,49 €

Comunicación de la posición y orientación mediante el bloque TCP/IP……899,49 €

Generación de esquemas con Simulink para el control de posición del robot móvil Lego Mindstorms EV3……1.199,32 €

Ajuste de constantes y validación de esquemas mediante simulación……749,57 €

Validación de esquemas mediante pruebas experimentales……1.858,51 €

Persecución de robots……1.049,40 €

Redacción de documentos……1.923,99 €

**Presupuesto de ejecución material……11.995,00 €**

Gastos generales (13%)……1.559,35 €

Beneficio industrial (6%)……719,70 €

**Presupuesto de ejecución por contrata……14.274,05 €**

IVA (21%)……2.997,55 €

**Presupuesto base de licitación……17.271,60 €**

Por tanto, el actual presupuesto de inversión asciende a la cantidad de:

DIECISIETE MIL DOSCIENTOS SETENTA Y UN EUROS CON SESENTA CÉNTIMOS