UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

PLATAFORMA MÓVIL OMNIDIRECCIONAL A PARTIR DE DOS ROBOTS MÓVILES DIFERENCIALES

T E S I S
Que para obtener el título de
INGENIERO MECATRÓNICO

P R E S E N T A

ARTURO MARTÍNEZ CARRILLO

DIRECTOR DE TESIS
DR. VÍCTOR JAVIER GONZÁLEZ VILLELA

MÉXICO, D. F., ENERO 2012

DEDICATORIA

A mis padres que me apoyaron incondicionalmente y que nunca dejaron de impulsarme para alcanzar todas mis metas.

A mis hermanos porque siempre han sido mi ejemplo a seguir.

AGRADECIMIENTOS

Agradezco a la Universidad Nacional Autónoma de México por brindarme todas las facilidades para mi formación profesional.

Agradezco a los profesores que me impartieron clases por todo el conocimiento que me transmitieron.

Agradezco al Dr. Víctor Javier González Villela por asesorarme en este proyecto y apoyarme en mi proceso de titulación.

Agradezco en lo que le corresponde a la DGAPA, UNAM, por el apoyo brindado para la realización de este trabajo, a través de los proyectos PAPIIT IN108308: "Investigación y desarrollo en robótica móvil, robótica paralela y sistemas mecatrónicos (2008-2010)" y PAPIIT IN115811: "Investigación y desarrollo en sistemas mecatrónicos: robótica móvil, robótica paralela, robótica híbrida y teleoperación (2011-2013)".

RESUMEN

El presente trabajo muestra el análisis cinemático de un robot móvil omnidireccional con dos robots diferenciales como elemento de tracción, mediante la implementación de una teoría unificadora. Se realiza el análisis de controlabilidad mediante álgebra de Lie y la implementación del control mediante campos potenciales. También contempla los resultados de pruebas en lazo abierto y cerrado, de seguimiento de trayectoria y de alcance de objetivo con un modelo funcional del robot que utiliza el software reacTIVision como sistema de visión para la ubicación del robot. Se programan las ecuaciones provenientes del análisis cinemático y se implementa el control mediante el software Simulink y se envían los datos mediante módulos XBee de radiofrecuencia.

En las pruebas se observa que la plataforma sí es capaz de seguir las trayectorias definidas con la orientación requerida, así como de alcanzar el objetivo solicitado. Con ello se comprueba que el análisis con la teoría unificadora es correcto y que la plataforma móvil es omnidireccional y controlable. También se prueba que con campos potenciales se puede controlar la plataforma desde su cinemática.

CONTENIDO

LISTA DE FIGURAS

CAPÍTULO I INTRODUCCIÓN

La robótica móvil es la rama que estudia a todos aquellos robots que puedan desplazarse dentro de su medio, ya sea en el aire, el agua o sobre la superficie terrestre. En esta última categoría, los que presentan un desarrollo más avanzado son aquéllos que utilizan ruedas como elemento de avance, debido a su fácil montaje y estabilidad al momento de moverse. Además, como los medios de transporte terrestre más utilizados utilizan ruedas, es natural que en robótica se trate de evolucionar con una tendencia inicial hacia un sistema ya muy bien estudiado.

Para los robots móviles (no sólo los terrestres) se tienen tres problemáticas principales a resolver para poder desplazarse: la cinemática del vehículo, que define las capacidades de movilidad propias del mismo; la definición del camino o la trayectoria a seguir y la percepción del entorno para ubicar al robot en su espacio de trabajo. En un caso totalmente manual, los últimos dos rubros son procesados por el usuario y el grado de automatización del vehículo puede ir subiendo, hasta que los tres puntos sean resueltos enteramente por el robot utilizando sensores, procesando e interpretando las señales de éstos para alcanzar el objetivo definido por el usuario. Se puede observar que la problemática que sigue presente, independientemente del nivel de automatización, es la cinemática del vehículo.

En particular, para los robots impulsados por ruedas, dicha movilidad está directamente ligada al tipo de ruedas que se utilicen y a la topología de las mismas. Éstas definen el grado de libertad de movimiento, dependiendo de cómo estén acomodadas, que se traduce a los movimientos lineales en el plano, de rotación del sistema y la relación entre ambos.

Así es como se tiene una forma de clasificar a los robots móviles con ruedas. Aquéllos que tienen control sobre uno o dos posibles movimientos son conocidos como no holonómicos, como bien puede ser el movimiento de un automóvil, que avanza y cambia de dirección, pero no puede desplazarse lateralmente. Por otro lado, se tienen los omnidireccionales u holonómicos que tienen la ventaja de poderse mover en cualquier dirección permisible por el medio. Este tipo de robot puede ir de un punto a otro cualquiera en un solo movimiento y girar de manera independiente; se puede pensar como el movimiento de una silla giratoria de oficina.

La importancia de desarrollar un vehículo omnidireccional se debe a varios factores:

- Mayor movilidad en espacios reducidos
- Cambios repentinos de dirección y sentido para evasión de obstáculos
- Desplazamiento y orientación independientes.

Lograr que un robot sea omnidireccional no es algo trivial, dado que el movimiento de avance de una rueda convencional está restringido a un solo sentido. Por ello, algunos investigadores proponen la fabricación de ruedas especiales que permiten el movimiento del vehículo en cualquier dirección, como la rueda sueca o la rueda universal mencionada por Cuéllar, F. (2006) [1].

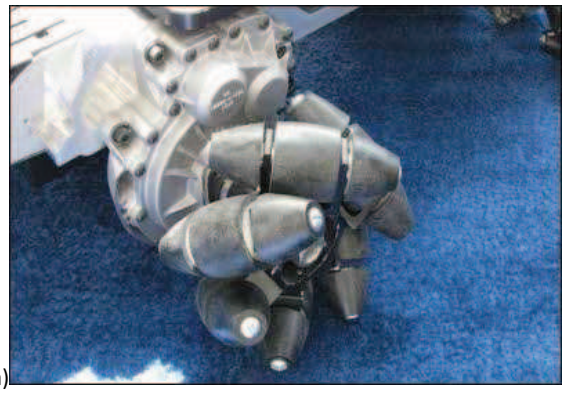
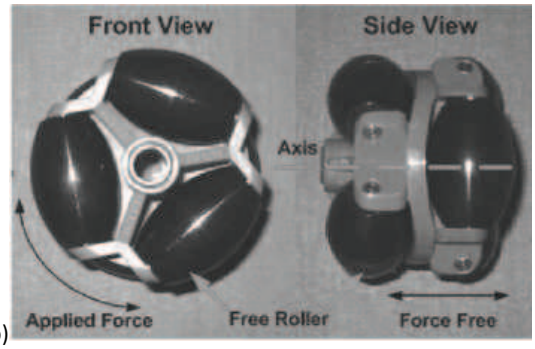
a)  b) 

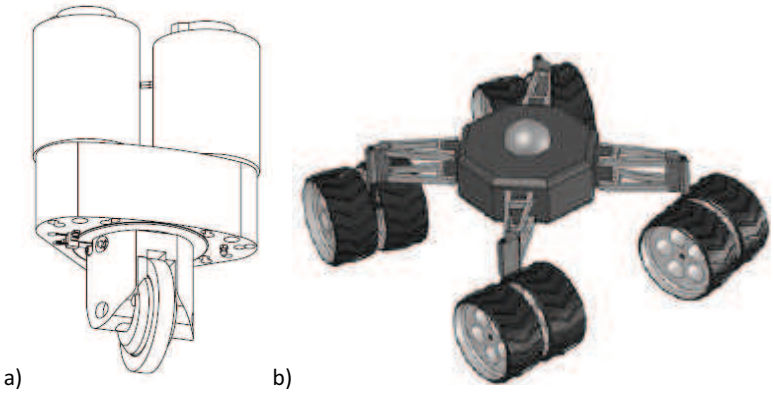
Figura I - 1 a) Rueda sueca b) Rueda universal.

La implementación de tres de éstas últimas, por ejemplo, en un arreglo triangular permite dicho movimiento, además de simplificar considerablemente el análisis

cinemático del robot. No obstante, estas ruedas son de manufactura complicada, no están pensadas para avanzar en superficies irregulares, además de que presentan inestabilidad dado que en ciertos puntos perimetrales se pierde el contacto con el suelo. Esto da lugar a que no se dejen de lado los sistemas de tracción convencionales.

A partir de analizar la cinemática de un vehículo que tenga una configuración de ruedas cualquiera que le permita girar, se puede comprobar que cualquier punto que no esté situado en el eje transversal de las ruedas es capaz de seguir cualquier trayectoria continua. Bajo este principio funcionan las *ruedas castoras*, que son las que se encuentran en los carritos del supermercado o en las sillas de oficina.

Holmberg, R. et ál. (1999) [2] y Wada, M. et ál. (1995) [3] proponen para sus respectivas aplicaciones modelos de castora automática que permita desplazar al robot de manera que éste sea holonómico, motorizando la rueda en sus rotaciones paralela y perpendicular al suelo. Udengaard, M. et ál. (2008) [4] y Yamada, T. et ál. (2001) [5] utilizan el principio de la castora con dos ruedas unidas entre ellas y con tracción independiente, que a su vez tienen una unión descentrada y libre de giro con respecto al robot. El giro en el punto de apoyo del robot se logra mediante la diferencia de velocidades en las ruedas.



a) b)

c)

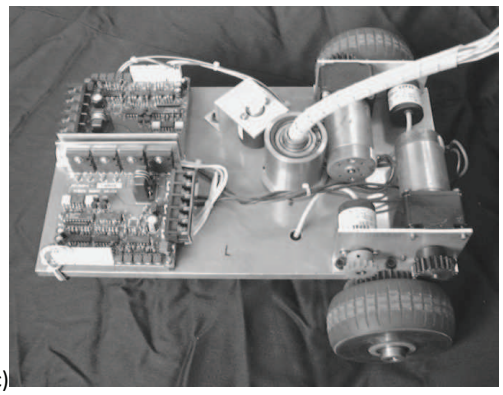


Figura I - 2 Castora automática (Holmberg) b) Robot omnidireccional todo terreno (Udengaard) c) Robot de tracción diferencial para robot omnidireccional (Yamada).

Todas estas alternativas son más factibles para aplicarse en vehículos fuera de laboratorio, particularmente la de Udengaard, M. et ál. (2008) [4] que está pensado expresamente para ello. No obstante, en los últimos dos trabajos mencionados, el análisis matemático que comprueba la holonomicidad no queda enteramente claro y sólo aplica para ese caso en particular.

Por otro lado, González Villela, V. (2006) [6] propone una teoría unificadora para el análisis cinemático y dinámico de robots con ruedas convencionales. Los trabajos propuestos por Udengaard, M. et ál. (2008) [4] y Yamada, T. et ál. (2001) [5] utilizan configuraciones no convencionales, pero parten de las mismas bases que el análisis para uno convencional.

El propósito de este trabajo es el de comprobar si es posible extender las aplicaciones de la teoría unificadora de González Villela, V. (2006) [6] a un modelo de robot móvil no convencional, en este caso una plataforma móvil omnidireccional, y con ello ser capaces de controlar el movimiento de la misma mediante un modelo físico.

Para ello, en el capítulo II se describe la plataforma y su sistema de tracción. Se define la postura del robot generalizando los parámetros, para que la extensión de la teoría unificadora pueda ser aplicable con otra configuración de robot no convencional.

Después en el capítuo III, se realiza el análisis cinemático para la propagación de las velocidades del robot a las ruedas. Esto es, que las velocidades deseadas del punto de análisis puedan ser traducidas en velocidades de giro de las ruedas, a través de la cadena cinemática formada por el eslabonamiento de los cuerpos que integran todo el robot. Posteriormente se determina el tipo de vehículo que es, para comprobar la omnidireccionalidad del mismo.

En el capítulo IV se comprueba mediante el álgebra de Lie si la plataforma es controlable. Ya con el modelo obtenido y luego de asegurar que es omnidireccional, se propone una ley para controlar a la plataforma, con la intención de que sea totalmente automatizada.

En el capítulo V se describe el algoritmo de las simulaciones, así como la constitución del modelo funcional y del sistema montado para la realización de las pruebas.

En el capítulo VI se muestran y comentan los resultados de las simulaciones y las pruebas con el modelo funcional para verificar que la respuesta sea la esperada.

Por último, en el capítulo VII se mencionan las conclusiones de los experimentos y el trabajo a futuro.

CAPÍTULO II DESCRIPCIÓN DE LA PLATAFORMA MÓVIL OMNIDIRECCIONAL

II.1 Definición de un vehículo con ruedas convencional

En general, los vehículos terrestres analizados comúnmente suponen que se cuenta con una estructura rígida, es decir, que no cuenta con partes flexibles. Se puede pensar como un bloque unido a sus ruedas, donde el único movimiento relativo existente es el giro de éstas con respecto al cuerpo del móvil. Además se asume que es un vehículo no degenerado, esto es que cuenta con las ruedas suficientes para poder controlar su posición y orientación.

A esto se le considerará como un vehículo con ruedas convencional. El robot móvil propuesto para este trabajo, se constituye del eslabonamiento de distintos elementos para generar un movimiento deseado para uno de los eslabones, mediante la combinación de los movimientos de los otros. Por lo tanto el robot a analizar será de una topología no convencional.

II.2 Configuración del robot

Para llevar a cabo la comprobación de que la teoría unificadora propuesta por González Villela, V. (2006) [6] es aplicable de manera extendida a un robot móvil no convencional, se debe explicar cuál es la configuración propuesta. En este caso, se plantea que existe un conjunto de robots móviles convencionales no holonómicos (elemento de tracción) que están unidos por un cuerpo central que será visto como el cuerpo de análisis. Las juntas que acoplan este último con los robots de tracción permiten el libre giro entre ambos elementos unidos, además de situarse en puntos donde el eje de giro no sea normal al eje de rotación de las ruedas.

El caso más sencillo de analizar es el de dos robots de tracción unidos por un cuerpo central, mismo que se utiliza para el desarrollo de este trabajo. Además, como posteriormente se observará, es el mínimo suficiente para que se compruebe que el cuerpo central es omnidireccional con el análisis cinemático.

De manera particular, en este caso se utilizan para la tracción dos robots en configuración diferencial o (2,0) según la clasificación de vehículos de Campion. Ambos están sujetos a una plataforma en dos puntos descentrados con la intención de que el cálculo no presente singularidades al estar alineados los puntos de apoyo y el punto de análisis. De esta forma se consigue que éste sea aplicable a cualquier punto del plano en el cual se mueve la plataforma.

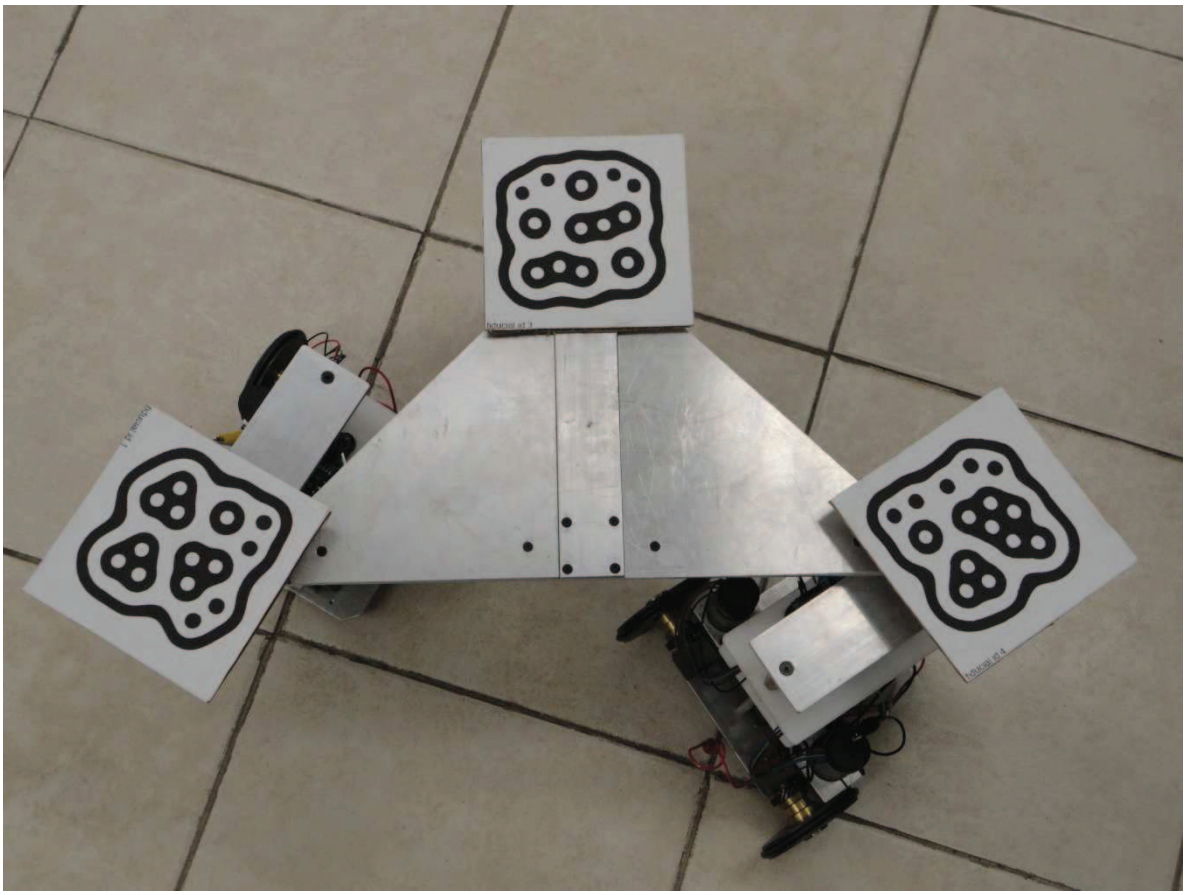


Figura II - 1 Plataforma móvil omnidireccional con dos robots diferenciales como elemento de tracción.

La figura II - 1 muestra la plataforma triangular donde en la punta superior del triángulo se encuentra el punto de análisis, mientras que sus otros dos vértices fungen como pivotes de los robots diferenciales. Las ilustraciones en cada punto son identificadores que serán descritos más adelante.

II.3 Postura del robot

Para el análisis se deben definir sistemas de coordenadas que permitan describir el desplazamiento de la plataforma en su universo, así como de los elementos internos que presenten un movimiento relativo entre ellos, por lo que se obtiene lo siguiente ilustrado en la figura II - 2.

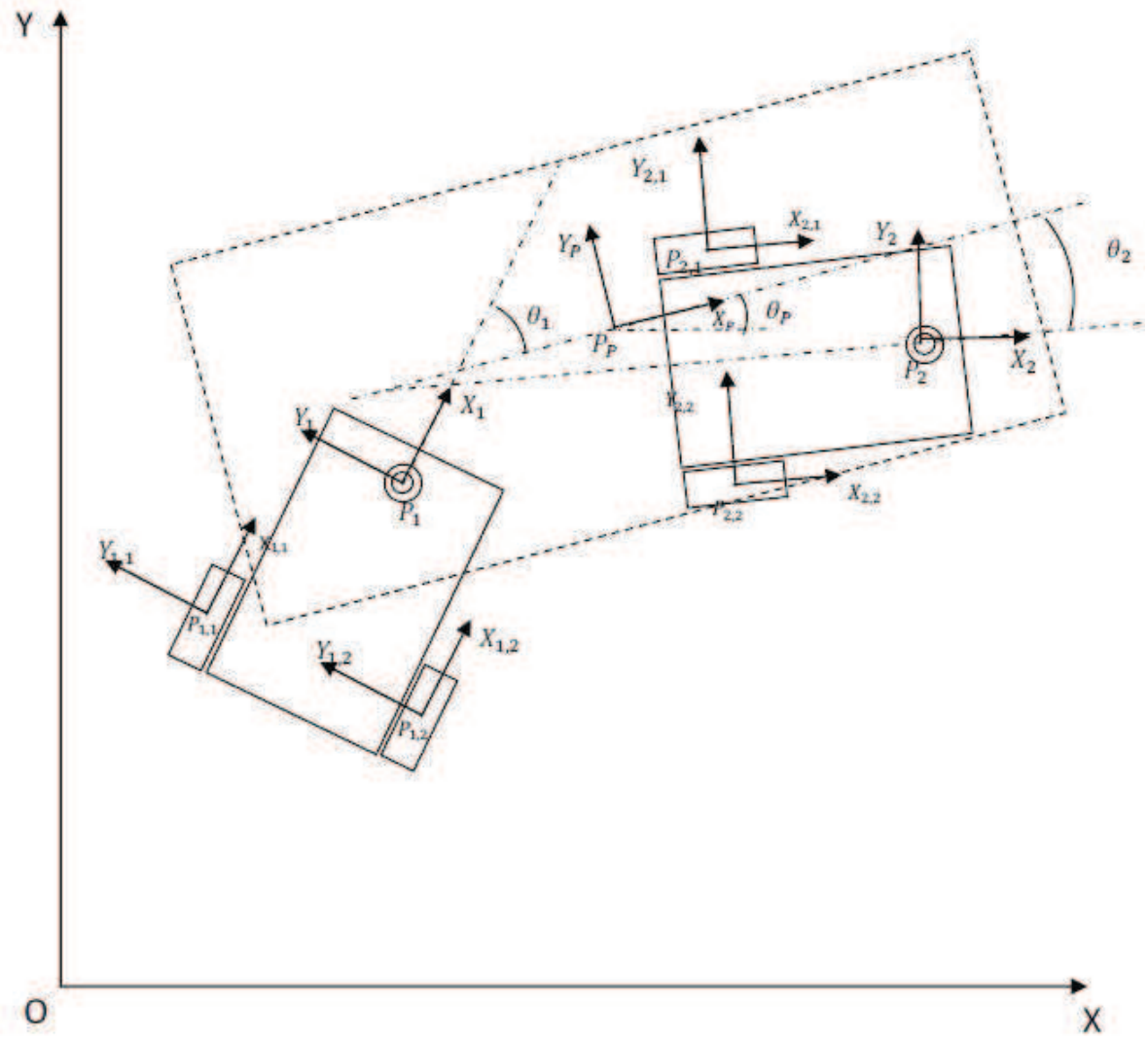


Figura II - 2 Sistemas de ejes coordenados de la plataforma móvil omnidireccional.

Donde *{X,Y}* representa el sistema coordenado del universo con *O* como su origen. $\{X_p, Y_p\}$ es el sistema coordenado de la plataforma y está anclado en el punto P_P, que es el punto de análisis. El sistema $\{X_i, Y_i\}$ situado en el punto P_i representa al sistema de coordenadas del i-ésimo robot de tracción. De igual manera, el sistema $\{X_{i,j}, Y_{i,j}\}$ situado en el punto $P_{i,j}$ representa al sistema de coordenadas de la j-ésima rueda del i-ésimo robot.

Además, el ángulo θ_p, corresponde a la orientación de los ejes coordenados $\{X_p, Y_p\}$ con respecto al sistema del universo. El ángulo θ_i hace referencia a la orientación de los ejes coordenados $\{X_i, Y_i\}$ con respecto al sistema de la plataforma.

Los sistemas coordenados de los robots de tracción y la plataforma, así como su orientación son suficientes para describir la postura del robot. No obstante, se

aprovecha desde este punto para definir la nomenclatura y los ejes coordenados de las ruedas con la intención de que se distinga de forma clara la sucesión progresiva en la numeración de todos los sistemas de coordenadas que se utilizarán para los cálculos.

En este punto, también se considera el ángulo $\theta_{i,j}$ de cada rueda con respecto al robot que pertenecen con la intención de que el análisis sea válido para cualquier otra configuración de robot de tracción. Dado que el esquema de la representa el sistema a analizar, de manera particular para este caso los ángulos $\theta_{i,j}$ tienen un valor de cero constante debido a la configuración diferencial de los robots de tracción, sin embargo este valor no se toma en consideración durante el inicio de los cálculos.

Posteriormente se definen las distancias entre los puntos de origen de los sistemas coordenados que serán necesarias para el análisis cinemático. Manteniendo nuevamente las magnitudes sin definir, la figura II - 3 muestra dichos parámetros.

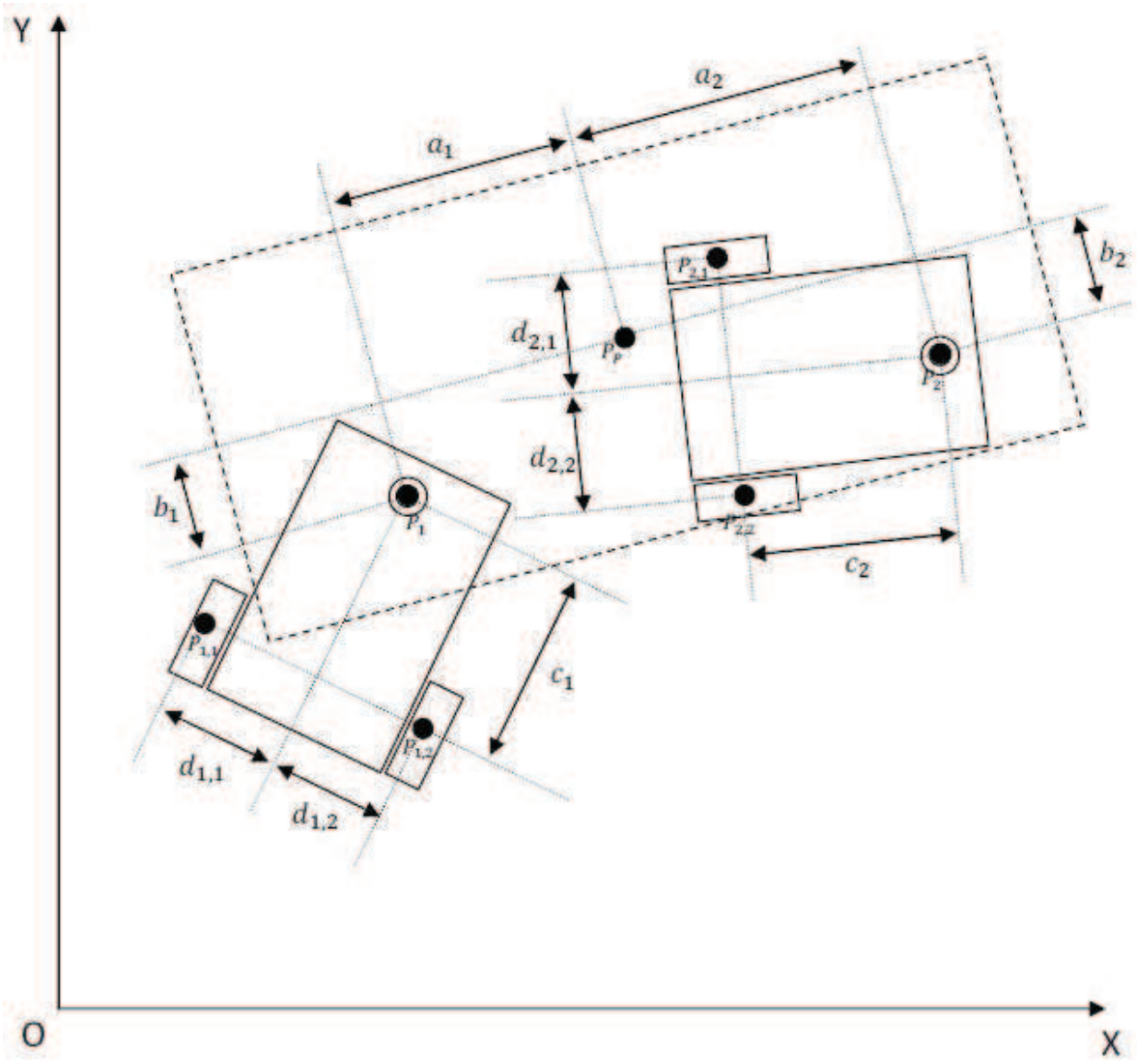


Figura II - 3 Distancia entre orígenes de los sistemas de ejes coordenados.

Se puede observar que todas las dimensiones definidas en la figura II - 3 están medidas a lo largo de los ejes coordenados definidos anteriormente.

CAPÍTULO III CINEMÁTICA INVERSA

III.1 Propagación de las velocidades

Una vez definido el sistema y declarado los parámetros necesarios, se resuelve la cinemática de la plataforma móvil. Esto se puede abordar de dos formas: a través de la cinemática directa o la inversa. En la directa se obtienen las ecuaciones que permiten conocer la velocidad y posición de la plataforma a partir de velocidades arbitrarias aplicadas a las ruedas. Este enfoque resulta impráctico, dado que el propósito del robot no es el de andar de manera aleatoria por el entorno, sino el de alcanzar un objetivo en concreto.

Es por eso que se opta por resolver la cinemática inversa, donde a partir de velocidades de entrada de la plataforma se pueden obtener las velocidades de los actuadores necesarias para cumplir dicha orden. Las ecuaciones resultantes son la expresión matemática que define las restricciones cinemáticas del robot. Para ello se utiliza el método de propagación de velocidades del eslabón *i* al eslabón *i+1*.

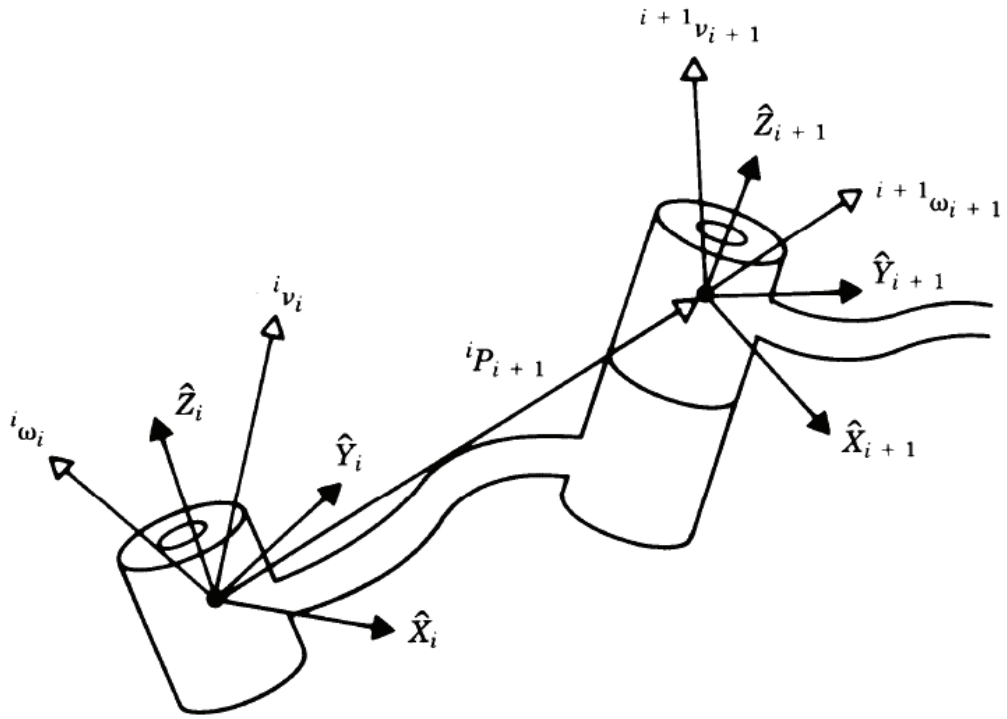


Figura III - 1 Propagación de velocidades del eslabón *i* al eslabón *i+1.*

Lo que este método sugiere es que la velocidad angular del eslabón *i+1* relativa a su propio sistema de referencia es igual a la velocidad angular del eslabón *i* con respecto al sistema *i+1*, más una componente propia del sistema *i+1*.

$$^{i+1}_{i+1}\omega = {}^{i+1}_{i}R \times {}^{i}_{i}\omega + {}^{i+1}_{i+1}\dot{\theta}$$

(Ecuación 1)

Donde ${}^{i+1}_{i}R$ es la matriz de rotación del sistema *i* al sistema *i+1*. Por lo que al aplicársela a la velocidad angula de *i* con respecto al sistema *i*, se obtiene la velocidad angular de *i* con respecto a *i+1* que se mencionó anteriormente.

Por otro lado, la velocidad lineal del eslabón *i+1* en relación con sí mismo es igual a la velocidad lineal del eslabón *i* más la velocidad provocada por el brazo de palanca ${}^{i}_{i+1}P$ y la velocidad angular de *i*. Todo esto referenciado al eje de coordenadas de *i+1*.

$$^{i+1}_{i+1}v = {}^{i+1}_{i}R \times \left({}^{i}_{i}v + {}^{i}_{i}\omega \times {}^{i}_{i+1}P\right)$$

(Ecuación 2)

Dado que el punto de análisis es el punto P_P de la plataforma, se realiza la propagación de las velocidades angular y lineal de dicho punto hasta las ruedas,

utilizando las Ecuaciones 1 y 2 respectivamente. La definición de los sistemas coordenados permite una secuenciación ordenada del proceso.

III.1.1 Propagación del sistema $\{X_p, Y_p\}$ al $\{X_1, Y_1\}$

Sea la velocidad angular del sistema *P* con respecto a sí mismo representada por el vector

$$ {}^{P}_{P}\omega = \{0 \quad 0 \quad \dot{\theta}_p\}^T $$

Y la velocidad lineal por el siguiente vector.

$$ {}^{P}_{P}v = \{{}^{P}_{P}\dot{x} \quad {}^{P}_{P}\dot{y} \quad 0\}^T $$

Y sea la matriz de rotación del sistema *P* a *1.*

$$ {}^{1}_{P}R = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

Por otro lado, sea la velocidad angular propia del sistema *1*

$$ {}^{1}_{1}\dot{\theta} = \{0 \quad 0 \quad \dot{\theta}_1\}^T $$

Y el vector de posición del punto P_1 con respecto a P_P

$$ {}^{P}_{1}P = \{a_1 \quad b_1 \quad 0\}^T $$

Sustituyendo ${}^{P}_{P}\omega, {}^{1}_{P}R, {}^{1}_{1}\dot{\theta}$ en la ecuación 1, se obtiene:

$$ {}^{1}_{1}\omega = {}^{1}_{P}R \times {}^{P}_{P}\omega + {}^{1}_{1}\dot{\theta} $$

$$ {}^{1}_{1}\omega = \{0 \quad 0 \quad \dot{\theta}_p + \dot{\theta}_1\}^T $$

Donde ${}^{1}_{1}\omega$ es la velocidad angular del robot de tracción 1 con respecto a sí mismo.

Además, se puede obtener la velocidad lineal sustituyendo las ecuaciones ${}^{P}_{P}\omega, {}^{P}_{P}v, {}^{1}_{P}R, {}^{P}_{1}P$ en la ecuación 2

$$ {}^{1}_{1}v = {}^{1}_{P}R \times ({}^{P}_{P}v + {}^{P}_{P}\omega \times {}^{P}_{1}P) $$

$$ {}^{1}_{1}v = \begin{bmatrix} {}^{P}_{P}\dot{x}\cos\theta_1 + {}^{P}_{P}\dot{y}\sin\theta_1 + \dot{\theta}_p(a_1\sin\theta_1 - b_1\cos\theta_1) \\ -{}^{P}_{P}\dot{x}\sin\theta_1 + {}^{P}_{P}\dot{y}\cos\theta_1 + \dot{\theta}_p(a_1\cos\theta_1 + b_1\sin\theta_1) \\ 0 \end{bmatrix} $$

Una vez obtenidas ambas velocidades, se repite el procedimiento para la propagación del robot a la rueda.

III.1.2 Propagación del sistema {X1,Y1} al {X1,1,Y1,1}

Sea la matriz de rotación del sistema *1* al *1,1*

$$ {}^{1,1}_{1}R = \begin{bmatrix} \cos\theta_{1,1} & \sin\theta_{1,1} & 0 \\ -\sin\theta_{1,1} & \cos\theta_{1,1} & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

El vector de velocidad angular propia de la rueda 1,1

$$ {}^{1,1}_{1,1}\dot{\theta} = \begin{Bmatrix} 0 & 0 & \dot{\theta}_{1,1} \end{Bmatrix}^T $$

Y el vector de posición del punto $P_{1,1}$ con respecto a P_1

$$ {}^{1}_{1,1}P = \begin{Bmatrix} c_1 & d_{1,1} & 0 \end{Bmatrix}^T $$

Sustituyendo nuevamente en las ecuaciones 1 y 2, se obtienen las siguientes velocidades:

$$ {}^{1,1}_{1,1}\omega = \begin{Bmatrix} 0 & 0 & \dot{\theta}_p + \dot{\theta}_1 + \dot{\theta}_{1,1} \end{Bmatrix}^T $$

$$ {}^{1,1}_{1,1}v = \begin{bmatrix} {}^{P}_{P}\dot{x}\cos(\theta_1 + \theta_{1,1}) + {}^{P}_{P}\dot{y}\sin(\theta_1 + \theta_{1,1}) + \dot{\theta}_p[a_1\sin(\theta_1 + \theta_{1,1}) - b_1\cos(\theta_1 + \theta_{1,1})] + \dot{\theta}_1(c_1\sin\theta_{1,1} - d_{1,1}\cos\theta_{1,1}) \\ -{}^{P}_{P}\dot{x}\sin(\theta_1 + \theta_{1,1}) + {}^{P}_{P}\dot{y}\cos(\theta_1 + \theta_{1,1}) + \dot{\theta}_p[a_1\cos(\theta_1 + \theta_{1,1}) + b_1\sin(\theta_1 + \theta_{1,1})] + \dot{\theta}_1(c_1\cos\theta_{1,1} + d_{1,1}\sin\theta_{1,1}) \\ 0 \end{bmatrix} $$

Siendo ${}^{1,1}_{1,1}\omega$ la velocidad angular ortogonal al plano *{X1,1,Y1,1}* y ${}^{1,1}_{1,1}v$ la velocidad lineal en el mismo plano, ambas para la rueda *1,1*.

III.2 Restricciones cinemáticas

De la expresión de la velocidad lineal, se obtienen las restricciones cinemáticas para la rueda *1,1* reescribiéndose de la siguiente manera:

Restricción ortogonal al plano de giro de la rueda {$Y_{1,1}$,$Z_{1,1}$}, suponiendo que no hay deslizamiento lateral:

$$[-\sin(\theta_1+\theta_{1,1}) \quad \cos(\theta_1+\theta_{1,1}) \quad a_1\cos(\theta_1+\theta_{1,1})+b_1\sin(\theta_1+\theta_{1,1}) \quad c_1\cos\theta_{1,1}+d_{1,1}\sin\theta_{1,1}]\dot{\xi}_1=0$$

(Ecuación 3)

Donde $\dot{\xi}_1=[{}^{p}_{p}\dot{x} \quad {}^{p}_{p}\dot{y} \quad \dot{\theta}_p \quad \dot{\theta}_1]^T$

Restricción sobre el plano de giro de la rueda {$X_{1,1}$,$Z_{1,1}$}, suponiendo rotación pura sin deslizamiento:

$$[\cos(\theta_1+\theta_{1,1}) \quad \sin(\theta_1+\theta_{1,1}) \quad a_1\sin(\theta_1+\theta_{1,1})-b_1\cos(\theta_1+\theta_{1,1}) \quad c_1\sin\theta_{1,1}-d_{1,1}\cos\theta_{1,1}]\dot{\xi}_1-r_{1,1}\dot{\phi}_{1,1}=0$$

(Ecuación 4)

Donde $r_{1,1}$ es el radio de la rueda y $\dot{\phi}_{1,1}$ es la velocidad de rotación de la misma. Este segundo término aparece debido a que la velocidad lineal para un punto en una circunferencia es igual a la velocidad angular multiplicada por el radio de giro. La ecuación 4 es el resultado de igualar las dos expresiones y agrupar los términos en un mismo lado de la ecuación.

Se aplica el mismo procedimiento de manera recursiva para las demás ruedas y se obtienen las siguientes restricciones cinemáticas:

Restricciones ortogonales al plano de giro de la rueda {$Y_{i,j}$,$Z_{i,j}$}

$$[-\sin(\theta_1+\theta_{1,2}) \quad \cos(\theta_1+\theta_{1,2}) \quad a_1\cos(\theta_1+\theta_{1,2})+b_1\sin(\theta_1+\theta_{1,2}) \quad c_1\cos\theta_{1,2}+d_{1,2}\sin\theta_{1,2}]\dot{\xi}_1=0$$

(Ecuación 5)

$$[-\sin(\theta_2+\theta_{2,1}) \quad \cos(\theta_2+\theta_{2,1}) \quad a_2\cos(\theta_2+\theta_{2,1})+b_2\sin(\theta_2+\theta_{2,1}) \quad c_2\cos\theta_{2,1}+d_{2,1}\sin\theta_{2,1}]\dot{\xi}_2=0$$

(Ecuación 6)

$$[-\sin(\theta_2+\theta_{2,2}) \quad \cos(\theta_2+\theta_{2,2}) \quad a_2\cos(\theta_2+\theta_{2,2})+b_2\sin(\theta_2+\theta_{2,2}) \quad c_2\cos\theta_{2,2}+d_{2,2}\sin\theta_{2,2}]\dot{\xi}_2=0$$

(Ecuación 7)

Donde $\dot{\xi}_2=[{}^{p}_{p}\dot{x} \quad {}^{p}_{p}\dot{y} \quad \dot{\theta}_p \quad \dot{\theta}_2]^T$.

Restricciones el plano de giro de la rueda {$X_{i,j}$,$Z_{i,j}$}:

$$\left[\cos(\theta_1+\theta_{1,2}) \quad \sin(\theta_1+\theta_{1,2}) \quad a_1\sin(\theta_1+\theta_{1,2})-b_1\cos(\theta_1+\theta_{1,2}) \quad c_1\sin\theta_{1,2}-d_{1,2}\cos\theta_{1,2}\right]\dot{\xi}_1-r_{1,2}\dot{\phi}_{1,2}=0$$

(Ecuación 8)

$$\left[\cos(\theta_2+\theta_{2,1}) \quad \sin(\theta_2+\theta_{2,1}) \quad a_2\sin(\theta_2+\theta_{2,1})-b_2\cos(\theta_2+\theta_{2,1}) \quad c_2\sin\theta_{2,1}-d_{2,1}\cos\theta_{2,1}\right]\dot{\xi}_2-r_{2,1}\dot{\phi}_{2,1}=0$$

(Ecuación 9)

$$\left[\cos(\theta_2+\theta_{2,2}) \quad \sin(\theta_2+\theta_{2,2}) \quad a_2\sin(\theta_2+\theta_{2,2})-b_2\cos(\theta_2+\theta_{2,2}) \quad c_2\sin\theta_{2,2}-d_{2,2}\cos\theta_{2,2}\right]\dot{\xi}_2-r_{2,2}\dot{\phi}_{2,2}=0$$

(Ecuación 10)

Donde $r_{i,j}$ y $\dot{\phi}_{i,j}$ es el radio y la velocidad de rotación de la j-ésima rueda del i-ésimo robot.

III.3 Coordenadas generalizadas de configuración

De acuerdo a la teoría unificadora de González Villela, V. (2006) [6], un robot móvil tiene $n_T = 3 + N_T + N_C + N_{OC}$ coordenadas de configuración $q = [q_1,\ldots,q_{nT}]^T$ y están distribuidas entre las coordenadas de postura ξ, los ángulos de las ruedas centradas α_C, los ángulos de las ruedas descentradas α_{OC} y de rotación de las ruedas ϕ.

$\xi = [{}^P_P x \quad {}^P_P y \quad \theta_P]^T$ define la postura del robot móvil *convencional*, sin embargo para este caso se puede observar en las restricciones cinemáticas que existen otras dos componentes necesarias para la definición de la postura. Éstas corresponden al ángulo de orientación de los robots de tracción θ_1 y θ_2, por lo que el número de coordenadas generalizadas aumenta dependiendo el número de robots de tracción que se tengan de manera que ahora

$$n_T = 3 + N_T + N_C + N_{OC} + N_{DR}$$

(Ecuación 11)

Donde

- N_T es el número de ruedas en general
- N_C el número de ruedas giratorias centradas
- N_{OC} el número de ruedas descentradas
- N_{DR} el número de robots de tracción.

Además, la postura del robot móvil no convencional se define ahora por ξ y por $\theta_{DR} = [\theta_1, \ldots, \theta_{N_{DR}}]^T$ que corresponde a los ángulos de orientación de los robots de tracción.

Finalmente esto da como resultado que las coordenadas generalizadas sean

$$q = [\xi \quad \theta_{DR} \quad \alpha_C \quad \alpha_{OC} \quad \phi]^T$$

(Ecuación 12)

Con las velocidades generalizadas

$$\dot{q} = [\dot{\xi} \quad \dot{\theta}_{DR} \quad \dot{\alpha}_C \quad \dot{\alpha}_{OC} \quad \dot{\phi}]^T$$

(Ecuación 13)

En particular, para la plataforma móvil, las coordenadas y velocidades generalizadas son:

$$q = [{}^P_P x \quad {}^P_P y \quad \theta_p \quad \theta_1 \quad \theta_2 \quad \phi_{1,1} \quad \phi_{1,2} \quad \phi_{2,1} \quad \phi_{2,2}]^T$$

$$\dot{q} = [{}^P_P \dot{x} \quad {}^P_P \dot{y} \quad \dot{\theta}_p \quad \dot{\theta}_1 \quad \dot{\theta}_2 \quad \dot{\phi}_{1,1} \quad \dot{\phi}_{1,2} \quad \dot{\phi}_{2,1} \quad \dot{\phi}_{2,2}]^T$$

Con estos vectores definidos, se puede agrupar toda la cinemática del sistema en una sola expresión del tipo $A_T(q)\dot{q} = 0$, de modo que si se reescriben las ecuaciones 3 a 10 de las restricciones, la expresión quedaría de la siguiente forma:

$$\begin{bmatrix}
-\sin(\theta_1+\theta_{1,1}) & \cos(\theta_1+\theta_{1,1}) & a_1\cos(\theta_1+\theta_{1,1})+b_1\sin(\theta_1+\theta_{1,1}) & c_1\cos\theta_{1,1}+d_{1,1}\sin\theta_{1,1} & 0 & 0 & 0 & 0 & 0 \\
-\sin(\theta_1+\theta_{1,2}) & \cos(\theta_1+\theta_{1,2}) & a_1\cos(\theta_1+\theta_{1,2})+b_1\sin(\theta_1+\theta_{1,2}) & c_1\cos\theta_{1,2}+d_{1,2}\sin\theta_{1,2} & 0 & 0 & 0 & 0 & 0 \\
-\sin(\theta_2+\theta_{2,1}) & \cos(\theta_2+\theta_{2,1}) & a_2\cos(\theta_2+\theta_{2,1})+b_2\sin(\theta_2+\theta_{2,1}) & 0 & c_2\cos\theta_{2,1}+d_{2,1}\sin\theta_{2,1} & 0 & 0 & 0 & 0 \\
-\sin(\theta_2+\theta_{2,2}) & \cos(\theta_2+\theta_{2,2}) & a_2\cos(\theta_2+\theta_{2,2})+b_2\sin(\theta_2+\theta_{2,2}) & 0 & c_2\cos\theta_{2,2}+d_{2,2}\sin\theta_{2,2} & 0 & 0 & 0 & 0 \\
\cos(\theta_1+\theta_{1,1}) & \sin(\theta_1+\theta_{1,1}) & -b_1\cos(\theta_1+\theta_{1,1})+a_1\sin(\theta_1+\theta_{1,1}) & -d_{1,1}\cos\theta_{1,1}+c_1\sin\theta_{1,1} & 0 & -r_{1,1} & 0 & 0 & 0 \\
\cos(\theta_1+\theta_{1,2}) & \sin(\theta_1+\theta_{1,2}) & -b_1\cos(\theta_1+\theta_{1,2})+a_1\sin(\theta_1+\theta_{1,2}) & -d_{1,2}\cos\theta_{1,2}+c_1\sin\theta_{1,2} & 0 & 0 & -r_{1,2} & 0 & 0 \\
\cos(\theta_2+\theta_{2,1}) & \sin(\theta_2+\theta_{2,1}) & -b_2\cos(\theta_2+\theta_{2,1})+a_2\sin(\theta_2+\theta_{2,1}) & 0 & -d_{2,1}\cos\theta_{2,1}+c_2\sin\theta_{2,1} & 0 & 0 & -r_{2,1} & 0 \\
\cos(\theta_2+\theta_{2,2}) & \sin(\theta_2+\theta_{2,2}) & -b_2\cos(\theta_2+\theta_{2,2})+a_2\sin(\theta_2+\theta_{2,2}) & 0 & -d_{2,1}\cos\theta_{2,1}+c_2\sin\theta_{2,1} & 0 & 0 & 0 & -r_{2,2}
\end{bmatrix}\dot{q} = 0$$

(Ecuación 14)

La ecuación 14 expresa la cinemática de la plataforma móvil de manera general, es decir, todavía contempla todas las dimensiones y los ángulos como si fueran valores distintos. Esto permitiría variar la configuración del robot sin necesidad de realizar todo el cálculo nuevamente para otro caso particular, inclusive pudiendo llegar a casos inviables para producir movimiento. No obstante, a partir de este

punto es necesario definir los parámetros de la plataforma a analizar, por lo que se obtiene lo siguiente.

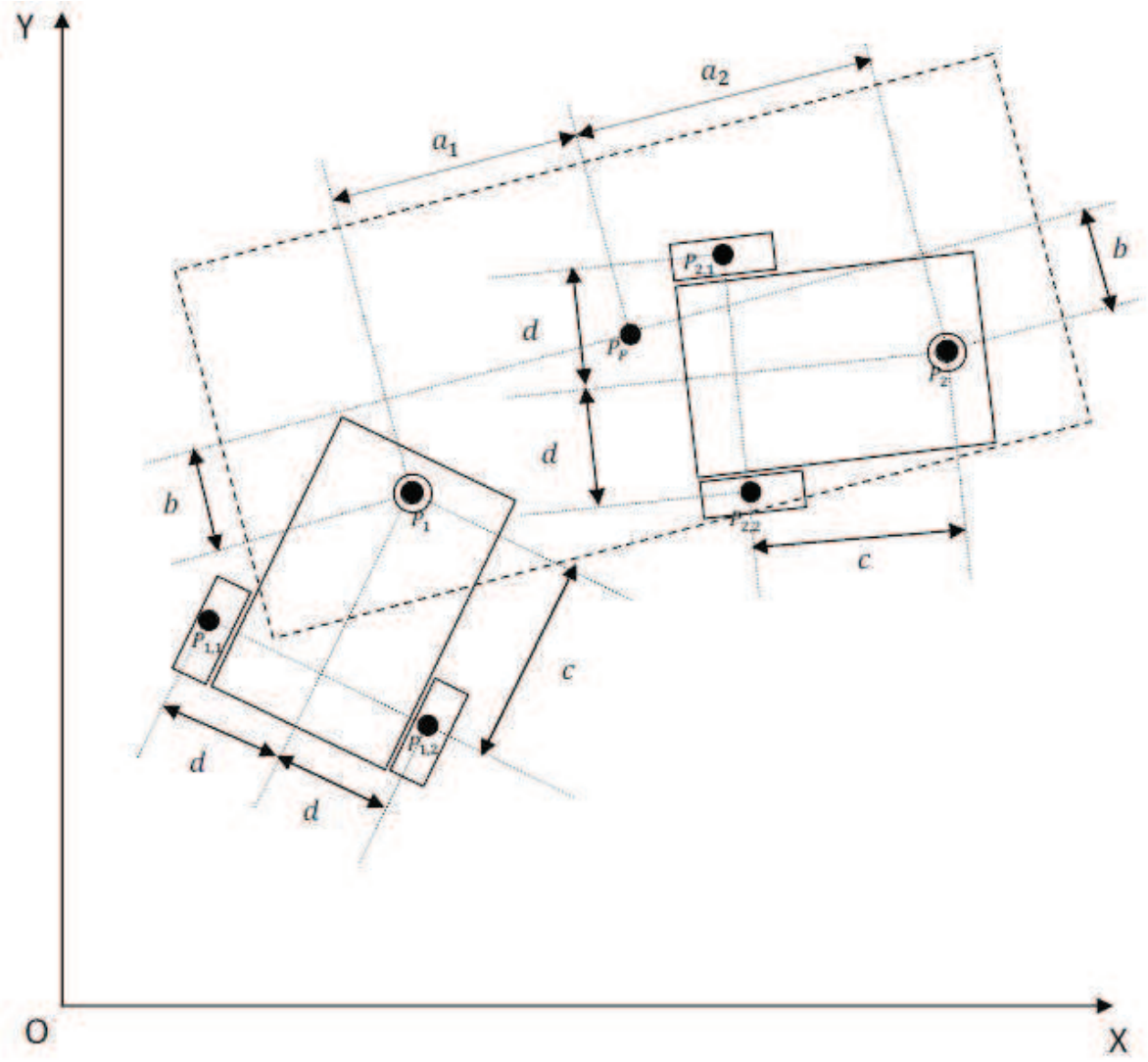


Figura III - 2 Distancia particular entre puntos.

En la figura III - 2 se observa que el único elemento que queda sin una posición fija es el punto P_p. Esto se dejó así con la intención de que la plataforma pueda analizarse desde cualquier punto del sistema de referencia *{X,Y}*; es el mismo principio del análisis de punto descentrado que se utiliza comúnmente en robótica móvil, pero integrado al punto de referencia de la plataforma.

Por otro lado, hay que recordar que los robots de tracción seleccionados son del tipo (2,0) por lo que los ángulos $\theta_{i,j}$ de las ruedas son cero y que todas las ruedas tienen el mismo diámetro. Integrando los valores de las dimensiones y de los ángulos a la ecuación 14, se obtiene la cinemática de esta configuración en particular.

$$\begin{bmatrix} -\sin(\theta_1) & \cos(\theta_1) & a_1\cos(\theta_1)+b\sin(\theta_1) & -c & 0 & 0 & 0 & 0 & 0 \\ -\sin(\theta_1) & \cos(\theta_1) & a_1\cos(\theta_1)+b\sin(\theta_1) & -c & 0 & 0 & 0 & 0 & 0 \\ -\sin(\theta_2) & \cos(\theta_2) & a_2\cos(\theta_2)+b\sin(\theta_2) & 0 & -c & 0 & 0 & 0 & 0 \\ -\sin(\theta_2) & \cos(\theta_2) & a_2\cos(\theta_2)+b\sin(\theta_2) & 0 & -c & 0 & 0 & 0 & 0 \\ \cos(\theta_1) & \sin(\theta_1) & -b\cos(\theta_1)+a_1\sin(\theta_1) & -d & 0 & -r & 0 & 0 & 0 \\ \cos(\theta_1) & \sin(\theta_1) & -b\cos(\theta_1)+a_1\sin(\theta_1) & d & 0 & 0 & -r & 0 & 0 \\ \cos(\theta_2) & \sin(\theta_2) & -b\cos(\theta_2)+a_2\sin(\theta_2) & 0 & -d & 0 & 0 & -r & 0 \\ \cos(\theta_2) & \sin(\theta_2) & -b\cos(\theta_2)+a_2\sin(\theta_2) & 0 & d & 0 & 0 & 0 & -r \end{bmatrix} \dot{q} = 0$$

(Ecuación 15)

III.4 Definición del tipo de robot

El tipo de robot se define de acuerdo a los grados de movilidad, direccionabilidad y maniobrabilidad que posea. Éstos están definidos por los movimientos permisibles, mismos que se obtienen de la cinemática del robot definida completamente en la ecuación 15.

III.4.1 Grado de movilidad

El grado de movilidad corresponde a los grados de libertad instantáneos que posee un robot y se obtiene a partir de la siguiente ecuación:

$$r_m = \dim[\xi P_p] - rank[O_{fc}]$$

(Ecuación 16)

Donde $\dim[\xi P_p]$ corresponde a la dimensión de las coordenadas generalizadas que definen la postura de la plataforma móvil y $rank[O_{fc}]$ corresponde al rango de la submatriz O_{fc} perteneciente a la matriz $A_T(q)$. Dicha submatriz corresponde a la sección de las restricciones cinemáticas ortogonales al plano de la rueda comprendida por las ruedas fijas y las ruedas centradas direccionables, que en este caso es

$$O_{fc} = \begin{bmatrix} A_T(q)_{1,1} & \cdots & A_T(q)_{1,5} \\ \vdots & \ddots & \vdots \\ A_T(q)_{4,1} & \cdots & A_T(q)_{4,5} \end{bmatrix}$$

Ya se sabe con anterioridad que $\dim[\xi P_p]$ es igual a cinco, mientras que $rank[O_{fc}]$ se observa que es igual a dos.

Por lo tanto, sustituyendo los valores en la ecuación 16, $r_m = 5 - 2 = 3$.

III.4.2 Grado de direccionabilidad

El grado de direccionabilidad define el número de elementos que cambian la dirección del vehículo y se obtiene a partir de la siguiente ecuación:

$$r_s = rank[O_c]$$

(Ecuación 17)

Donde $rank[O_c]$ corresponde al rango de la submatriz perteneciente a la matriz $A_T(q)$ definida por los elementos de las ruedas centradas direccionables de las restricciones cinemáticas ortogonales al plano de las ruedas. En este caso, dado que todas las ruedas son fijas, dicho rango es cero, por lo que $r_s = 0$.

III.4.3 Grado de maniobrabilidad

El grado de maniobrabilidad corresponde a los grados de libertad totales del robot y se define por la ecuación:

$$r_M = r_m + r_s$$

(Ecuación 18)

Por lo que se obtiene que r_M es igual a tres.

Esto significa que la plataforma móvil tiene tres grados de libertad. Utilizando nuevamente la clasificación de Campion, el tipo de robot se expresa de forma sencilla por la expresión (r_m, r_s), por lo tanto la plataforma móvil corresponde a un robot del tipo (3,0). El que tenga tres grados de libertad comprueba que el robot es omnidireccional en el plano.

CAPÍTULO IV ANÁLISIS DE CONTROLABILIDAD Y DEFINICIÓN DE LA LEY DE CONTROL

IV.1 Configuración cinemática con variables de estado

Dado que se tienen tres grados de libertad en la plataforma móvil, la siguiente cuestión es saber cuáles variables se seleccionan como variables de entrada y si el sistema es controlable. Lo que resulta más evidente es tomar directamente las variables de velocidad v_{xP} y v_{yP}, así como la velocidad angular ω_P.

Éstas serán las variables de estado del sistema, por lo que la expresión de la cinemática de la plataforma se tiene que expresar en términos de dichas variables, como en la siguiente ecuación:

$$\dot{q}_1 = \begin{bmatrix} \Sigma_\xi(\alpha_s) & 0 \\ 0 & \Sigma_c(\alpha_s) \\ \Sigma_{oc}(\alpha_s, \alpha_{oc}) & 0 \\ \Sigma_\phi(\alpha_s, \alpha_{oc}) & 0 \end{bmatrix} \begin{bmatrix} \eta \\ \zeta \end{bmatrix}$$

(Ecuación 19)

Donde Σ_{ξ} es la matriz asociada a las coordenadas de postura; Σ_{oc}, la matriz asociada a las coordenadas de las ruedas descentradas; Σ_{ϕ}, la matriz asociada a las coordenadas de rotación de las ruedas y Σ_{c}, la matriz asociada a las coordenadas de las ruedas centradas. Además, η y ζ corresponden a las componentes de movilidad y direccionabilidad.

En el caso de la plataforma, los elementos de la ecuación 19 Σ_{oc}, Σ_{c} y ζ no existen, además el vector $\eta = [v_{xP} \quad v_{yP} \quad \omega_P]^T$ corresponde claramente a las variables de entrada definidas anteriormente. Por lo tanto:

$$\dot{q}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\frac{\sin(\theta_1)}{c} & \frac{\cos(\theta_1)}{c} & \frac{a_1\cos(\theta_1) + b\sin(\theta_1)}{c} \\ -\frac{\sin(\theta_2)}{c} & \frac{\cos(\theta_2)}{c} & \frac{a_2\cos(\theta_2) + b\sin(\theta_2)}{c} \\ \frac{c\cos(\theta_1) + d\sin(\theta_1)}{cr} & \frac{c\sin(\theta_1) - d\cos(\theta_1)}{cr} & \frac{(a_1c - bd)\sin(\theta_1) - (bc + a_1d)\cos(\theta_1)}{cr} \\ \frac{c\cos(\theta_1) - d\sin(\theta_1)}{cr} & \frac{c\sin(\theta_1) + d\cos(\theta_1)}{cr} & \frac{(a_1c + bd)\sin(\theta_1) + (-bc + a_1d)\cos(\theta_1)}{cr} \\ \frac{c\cos(\theta_2) + d\sin(\theta_2)}{cr} & \frac{c\sin(\theta_2) - d\cos(\theta_2)}{cr} & \frac{(a_2c - bd)\sin(\theta_2) - (bc + a_2d)\cos(\theta_2)}{cr} \\ \frac{c\cos(\theta_2) - d\sin(\theta_2)}{cr} & \frac{c\sin(\theta_2) + d\cos(\theta_2)}{cr} & \frac{(a_2c + bd)\sin(\theta_2) + (-bc + a_2d)\cos(\theta_2)}{cr} \end{bmatrix} \begin{bmatrix} v_{xP} \\ v_{yP} \\ \omega_P \end{bmatrix}$$

(Ecuación 20)

De esta forma se tiene la cinemática interna de la plataforma expresada en términos de las variables de entrada. No obstante, el hecho de que sea interna se refiere a que aún está referenciada con respecto al sistema coordenado *{X_P, Y_P}*, por lo que se le tiene que aplicar una rotación de forma que el sistema coordenado de referencia sea el del universo *{X, Y}*. Esto para que se tenga la representación completa de acuerdo a como se observa desde fuera del robot.

Para esto se define la matriz de rotación de postura del sistema *{X_P, Y_P}* al sistema *{X, Y}*

$$^{0}_{p}R(\theta_p) = \begin{bmatrix} \cos\theta_p & -\sin\theta_p & 0 \\ \sin\theta_p & \cos\theta_p & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Y la matriz de rotación de configuración, como una extensión de la matriz anterior.

$$R_q = \begin{bmatrix} {}^{0}_{P}R(\theta_p) & 0 & 0 \\ 0 & I_{N_{DR}} & 0 \\ 0 & 0 & I_{N_c} \end{bmatrix}$$

Donde $R_q \in R^{(3+N_{DR}+N_c)\times(3+N_{DR}+N_c)}$ y tanto $I_{N_{DR}} \in R^{N_{DR}\times N_{DR}}$, como $I_{N_c} \in R^{N_c \times N_c}$ son matrices identidad. Dicha matriz de rotación, está definida por González Villela, V. (2006) [6] de forma general considerando todos los tipos de ruedas que se pueden tener, con la única diferencia que en este caso se agrega la matriz identidad $I_{N_{DR}}$ correspondiente al número de robots de tracción.

Por lo tanto, multiplicando al sistema por la matriz de rotación, se tiene la siguiente expresión:

$$\dot{q} = S_T(q)u \equiv \begin{bmatrix} \dot{\xi} \\ \dot{\theta}_{DR} \\ \dot{\phi} \end{bmatrix} = R_q \dot{q}_1$$

$$\dot{q} = \begin{bmatrix} \cos(\theta_p) & -\sin(\theta_p) & 0 \\ \sin(\theta_p) & \cos(\theta_p) & 0 \\ 0 & 0 & 1 \\ -\frac{\sin(\theta_1)}{c} & \frac{\cos(\theta_1)}{c} & \frac{a_1\cos(\theta_1)+b\sin(\theta_1)}{c} \\ -\frac{\sin(\theta_2)}{c} & \frac{\cos(\theta_2)}{c} & \frac{a_2\cos(\theta_2)+b\sin(\theta_2)}{c} \\ \frac{c\cos(\theta_1)+d\sin(\theta_1)}{cr} & \frac{c\sin(\theta_1)-d\cos(\theta_1)}{cr} & \frac{(a_1c-bd)\sin(\theta_1)-(bc+a_1d)\cos(\theta_1)}{cr} \\ \frac{c\cos(\theta_1)-d\sin(\theta_1)}{cr} & \frac{c\sin(\theta_1)+d\cos(\theta_1)}{cr} & \frac{(a_1c+bd)\sin(\theta_1)+(-bc+a_1d)\cos(\theta_1)}{cr} \\ \frac{c\cos(\theta_2)+d\sin(\theta_2)}{cr} & \frac{c\sin(\theta_2)-d\cos(\theta_2)}{cr} & \frac{(a_2c-bd)\sin(\theta_2)-(bc+a_2d)\cos(\theta_2)}{cr} \\ \frac{c\cos(\theta_2)-d\sin(\theta_2)}{cr} & \frac{c\sin(\theta_2)+d\cos(\theta_2)}{cr} & \frac{(a_2c+bd)\sin(\theta_2)+(-bc+a_2d)\cos(\theta_2)}{cr} \end{bmatrix} \begin{bmatrix} v_{xP} \\ v_{yP} \\ \omega_P \end{bmatrix}$$

(Ecuación 21)

La ecuación 21 expresa las velocidades generalizadas de la plataforma móvil en términos de las variables de entrada con respecto al sistema coordenado *{X,Y}*.

IV.2 Análisis con álgebra de control de Lie

Utilizando la explicación que da González Villela, V. (2006) [6] y el tutorial para la aplicación del álgebra de Lie para robots móviles de Coelho, P. (2003) [7], se aplicó el método para analizar la controlabilidad del sistema no lineal de la cinemática de la plataforma móvil.

Para ello se obtiene la submatriz S_{q_p} que corresponde a los elementos de las coordenadas de postura de la matriz $S_T(q)$, por lo que

$$S_{q_p} = \begin{bmatrix} \cos(\theta_p) & -\sin(\theta_p) & 0 \\ \sin(\theta_p) & \cos(\theta_p) & 0 \\ 0 & 0 & 1 \\ -\frac{\sin(\theta_1)}{c} & \frac{\cos(\theta_1)}{c} & \frac{a_1\cos(\theta_1) + b\sin(\theta_1)}{c} \\ -\frac{\sin(\theta_2)}{c} & \frac{\cos(\theta_2)}{c} & \frac{a_2\cos(\theta_2) + b\sin(\theta_2)}{c} \end{bmatrix}$$

Donde cada columna se nombrará como un vector. Por ello, $S_{q_p} = [S_{q_{p1}} \quad S_{q_{p2}} \quad S_{q_{p3}}]$.

Después se analiza la controlabilidad mediante la obtención del *corchete de Lie*, que corresponde a la expresión

$$[f, g](q) = \frac{\partial g}{\partial q} f(q) - \frac{\partial f}{\partial q} g(q)$$

(Ecuación 22)

Donde $[\boldsymbol{f}, \boldsymbol{g}](\boldsymbol{q})$ es el corchete de Lie; $\boldsymbol{f}$ y $\boldsymbol{g}$ son vectores columna $\boldsymbol{S_{q_{pn}}}$ elegidos arbitrariamente. El método consiste en obtener el corchete de Lie para todas las parejas de vectores columna de la matriz $\boldsymbol{S_{q_p}}$ y cada que se obtiene uno, agregarlo a la matriz $\boldsymbol{S_{q_p}}$; posteriormente se comprueba si la nueva columna agregada es linealmente independiente de las anteriores. De ser así, se mantiene la matriz extendida y dicha columna también se considera para el análisis de la independencia lineal de los corchetes de Lie subsecuentes; si no es linealmente independiente, se desecha y se prueba con otro par de columnas. Es claro que conforme aumenta el tamaño de la matriz, también aumentan la cantidad de pares de columnas posibles para probar.

En la ecuación 22 se puede ver que $[\boldsymbol{f}, \boldsymbol{g}](\boldsymbol{q}) = -[\boldsymbol{g}, \boldsymbol{f}](\boldsymbol{q})$, por lo que invertir las columnas resultaría en un corchete de Lie linealmente dependiente.

Una vez que ya no se obtienen más columnas linealmente independientes, el rango de la matriz extendida determina cuántas variables son controlables.

Para comprobar la independencia lineal del nuevo elemento, se hace uso de la ecuación

$$[f,g](q) = k_1 S_{q_{p1}} + \cdots + k_n S_{q_{pn}}$$

(Ecuación 23)

Si existe una solución para $k_1, \ldots, k_n$, entonces el corchete de Lie es linealmente dependiente.

De acuerdo con ello se resuelve para el caso de la plataforma.

- Para $f = S_{q_{p1}}, g = S_{q_{p2}}$

  Sustituyendo en la ecuación 22 se obtiene el siguiente resultado.

$$[f,g](q) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{c^2} \\ \frac{1}{c^2} \end{bmatrix}$$

Al tratar de resolver la ecuación 23, no se obtiene ninguna solución, por lo que el corchete de Lie es linealmente independiente y por ello se aumenta un vector columna a la matriz S_{q_p}, de tal forma que

$$S_{q_{p4}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{c^2} \\ \frac{1}{c^2} \end{bmatrix}$$

- Para $f = S_{q_{p1}}, g = S_{q_{p2}}$

  Sustituyendo en la ecuación 22:

$$[f,g](q)=\begin{bmatrix}\sin\theta_p\\-\cos\theta_p\\0\\\frac{a_1}{c^2}\\\frac{a_2}{c^2}\end{bmatrix}$$

Al tratar de resolver la ecuación 23, no se obtiene ninguna solución, por lo que:

$$S_{q_{ps}}=\begin{bmatrix}\sin\theta_p\\-\cos\theta_p\\0\\\frac{a_1}{c^2}\\\frac{a_2}{c^2}\end{bmatrix}$$

- Para $f=S_{q_{ps}}, g=S_{q_{ps}}$

  Sustituyendo en la ecuación 22:

$$[f,g](q)=\begin{bmatrix}\cos\theta_p\\\sin\theta_p\\0\\\frac{b}{c^2}\\\frac{b}{c^2}\end{bmatrix}$$

Al tratar de resolver la ecuación 23, se obtiene la siguiente solución:

$$k_1=1, k_2=k_3=\frac{c(\sin(\theta_1)-\sin(\theta_2))}{a_1-a_2+c\cos(\theta_1)-c\cos(\theta_2)}, k_3=0,$$

$$k_4=-\frac{-a_1b+a_2b-bc\cos(\theta_1)+bc\cos(\theta_2)+a_2c\sin(\theta_1)+c^2\cos(\theta_2)\sin(\theta_1)-a_1c\sin(\theta_2)-c^2\cos(\theta_1)\sin(\theta_2)}{a_1-a_2+c\cos(\theta_1)-c\cos(\theta_2)}$$

Por lo tanto, el corchete de Lie es linealmente dependiente y no se agrega a la matriz S_{q_p}.

- Para $f=S_{q_{pm}}, g=S_{q_{pn}}\ \forall\ m\in[1,n-1], n\ \in\{4,5\}|\ m,n\ \in\mathcal{Z}$

El corchete de Lie es linealmente dependiente.

Como no queda ningún par de vectores columna por probar, aquí termina el proceso iterativo de los corchetes de Lie. Claramente, se observa que la matriz S_{q_p}

ahora tiene dos columnas nuevas linealmente independientes y por ende, $S_{q_p} \in R^{5\times5}$.

Como ya se vio en la explicación del método, el rango de la matriz determina el número de variables que son controlables. En este caso, el rango de la matriz S_{q_p} es igual a cinco, por lo que las cinco variables del vector de postura son controlables. Esto significa que el robot se puede llevar a cualquier posición deseada, aunque como el número de grados de libertad es menor que el rango del vector de postura, dicha posición no puede ser alcanzada en un solo movimiento.

IV.3 Control por campos potenciales

Una vez definidas las variables a controlar y sabiendo que el sistema sí es controlable, se procede a elegir una ley de control para el movimiento de la plataforma móvil. Se decide utilizar el método de campos potenciales para el control de la plataforma móvil en lazo cerrado, con la intención de comprobar de forma rápida la omnidireccionalidad del robot, y observar las capacidades de adaptación ante el cambio de trayectoria y de objetivo.

De acuerdo a lo explicado por González Villela, V. et ál. (2004) [8], el método de campos potenciales se define por fuerzas de atracción y repulsión provocadas por el medio que interactúa con el robot, que sumadas obtienen la fuerza de atracción resultante que seguirá el robot hasta alcanzar su destino. Este método puede ser aplicado desde el punto de vista cinemático cambiando los vectores de fuerza por vectores de velocidad.

El objetivo a alcanzar representa un vector de velocidad en dirección del robot hacia éste. La magnitud de la velocidad es directamente proporcional a la distancia existente entre ambos cuerpos. Para evitar vectores de velocidad infinitos o excesivamente grandes, se define una velocidad máxima y un área de evasión alrededor del objetivo a partir del cual la velocidad comienza a disminuir, tal y como se muestra en la figura IV - 1.

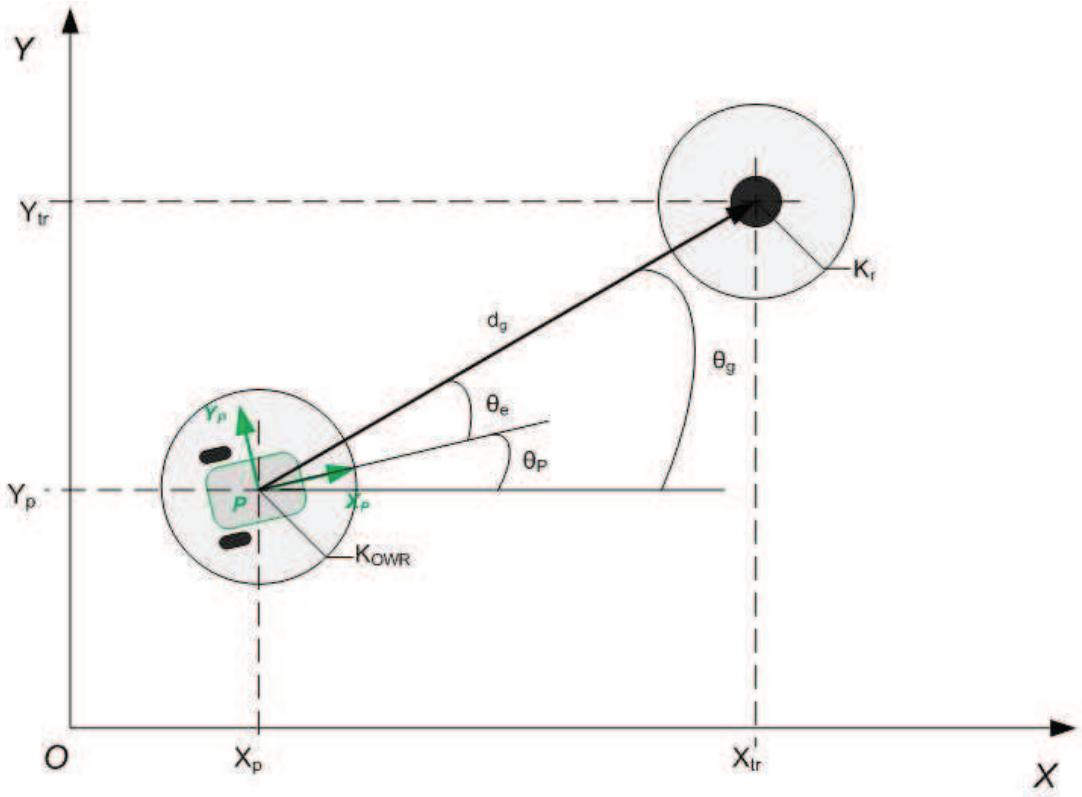


Figura IV – 1 Alcance de objetivo con campos potenciales.

En la figura IV - 1, se muestra d_g como el vector de atracción del robot al objetivo, θ_g como el ángulo de dicho vector, θ_i como el ángulo de inclinación del robot a controlar y θ_e como el error que existe entre ambos ángulos. El área de evasión de radio K_r es definida de manera arbitraria por el usuario, dependiendo de la aplicación para la que se requiera. También se define un área de seguridad en la plataforma móvil de radio K_{OWR}, dado que el punto de análisis está contenido dentro del espacio ocupado por la plataforma y existe riesgo de colisión.

A partir de esta descripción, se puede definir las velocidades lineales en X_P y en Y_P de la siguiente forma:

$$v_{xp} = \begin{cases} v_{xmax} \cdot \cos(\theta_e) & si \quad |d_g| > k_r + k_{OWR} \\ \frac{v_{xmax}}{k_r} \cdot d_g \cos(\theta_e) & si \quad |d_g| < k_r + k_{OWR} \\ 0 & si \quad |d_g| \leq k_{OWR} \end{cases}$$

(Ecuación 24)

$$v_{yp} = \begin{cases} v_{ymax} \cdot \sin(\theta_e) & si \quad |d_g| > k_r + k_{OWR} \\ \frac{v_{ymax}}{k_r} \cdot d_g \sin(\theta_e) & si \quad |d_g| < k_r + k_{OWR} \\ 0 & si \quad |d_g| \leq k_{OWR} \end{cases}$$

(Ecuación 25)

Las ecuaciones 24 y 25 muestran la ley de control para las velocidades lineales de entrada, para el alcance de un objetivo. Aunque en la figura IV - 1 se observa que el vector que conduce al robot hacia el objetivo es d_g, el ángulo contemplado para la descomposición de la velocidad necesario para que el robot alcance el objetivo no es θ_g. Esto debido a que las velocidades de entrada están definidas con respecto a los ejes coordenados de la plataforma móvil y no a los del universo. Por lo tanto, tomando como referencia el sistema de ejes coordenados de la plataforma, el vector d_g se descompone en dos elementos orientados por el ángulo de error θ_e.

La ventaja que muestra la plataforma móvil es que no se tiene que orientar de alguna manera para alcanzar el objetivo debido a sus capacidades de movilidad, por lo que no es necesario reducir el ángulo θ_e a un valor de cero. No obstante, de ser importante la orientación de la plataforma, se puede aplicar la siguiente ley de control:

$$\omega = \omega_{max} \sin(\theta)$$

(Ecuación 26)

Donde θ corresponde al ángulo variable en el tiempo que permite orientar la plataforma. En particular, para las pruebas a realizar se utilizan dos leyes de control para la velocidad angular:

$$\omega = \omega_{max} \sin(-\theta_p)$$

(Ecuación 27)

$$\omega = \omega_{max} \sin(\theta_e - \frac{\pi}{2})$$

(Ecuación 28)

La ecuación 27 permite una orientación de la plataforma de cero grados con respecto al universo, mientras que la ecuación 28 orienta a la plataforma de tal manera que la punta del triángulo apunte hacia el objetivo y se perciba de forma más clara la corrección de postura.

Con estas ecuaciones definidas, se puede tener el control de las variables de entrada del sistema para el alcance de un objetivo y por ende, el correcto desplazamiento de la plataforma.

Además del alcance de objetivos, los campos potenciales también contemplan la evasión de obstáculos. Esto implica que el obstáculo sea representado como una fuerza de repulsión que es inversamente proporcional a la distancia a la que se encuentre el robot de éste. La suma de las fuerzas de atracción del objetivo y de repulsión de los obstáculos es la que determina la resultante que tomará el robot. Debido a los objetivos de este proyecto, no se realizan pruebas con evasión de obstáculos, ya que la omnidireccionalidad de la plataforma móvil puede ser comprobada sólo con las pruebas de lazo abierto y alcance de objetivo en lazo cerrado.

CAPÍTULO V DESCRIPCIÓN DE LA SIMULACIÓN Y EL SISTEMA PARA PRUEBAS CON EL MODELO FUNCIONAL

V.1 Descripción de los elementos de la simulación

Una vez que se tiene resuelta la cinemática inversa de la plataforma y se ha determinado una ley de control, se realizan simulaciones con la intención de corroborar de forma rápida si los cálculos realizados arrojan resultados coherentes y que la plataforma (de forma teórica) se mueve de acuerdo a lo esperado.

Las simulaciones también sirven como una forma de diseñar las pruebas a realizar por el modelo real que sean más representativas para corroborar las propiedades omnidireccionales de la plataforma, así como la ley de control. De esta forma se tiene un punto de comparación no sólo en los datos y valores derivados de las ecuaciones cinemáticas y de control, sino también una referencia visual de cuál es el comportamiento esperado por la plataforma. Aunado a esto, se pueden anticipar errores de movimiento que sean inherentes a las propiedades cinemáticas y/o a la ley de control propuesta.

V.1.1 Software

Se decide utilizar la paquetería de Simulink de la compañía MathWorks, por su amplia gama de herramientas de cálculo, control, operaciones lógicas y funciones embebidas; la facilidad de implementación; la claridad para el seguimiento del programa, además de la familiaridad que ya se cuenta con este software. Además, el control del modelo funcional de la plataforma móvil puede realizarse desde este mismo programa, por lo que sólo se requiere de adaptar la simulación para la adquisición y envío de señales.

Simulink es una plataforma para simulación multidominio y diseño basado en modelos de sistemas dinámicos y embebidos. Proporciona un entorno gráfico interactivo y un conjunto de librerías de bloques personalizables que permiten diseñar, simular, implementar y probar una gran variedad de sistemas con variación temporal, entre los que se incluyen sistemas de comunicaciones, control, procesado de señales, vídeo e imagen. Simulink está integrado con MATLAB y ofrece acceso inmediato a una amplia gama de herramientas que permiten desarrollar algoritmos, analizar y visualizar simulaciones, crear series de procesado de lotes, personalizar el entorno de modelaje y definir señales, parámetros y datos de pruebas.[9]

V.1.2 Estructura de la simulación

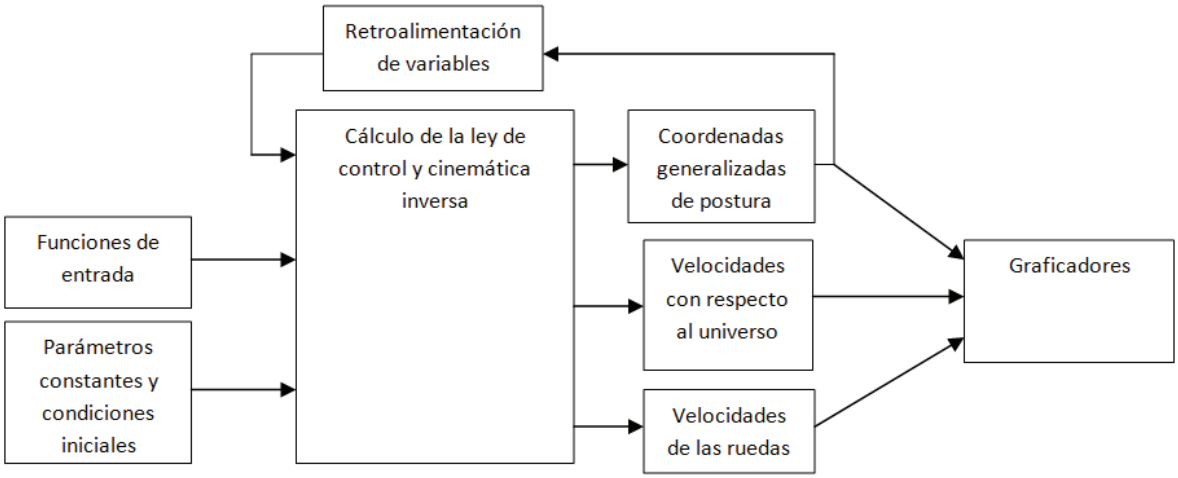


Figura V – 1 Estructura de la simulación.

La figura V - 1 muestra el diagrama de bloques general de la estructura seguida por las simulaciones. Cada una cuenta con ligeras variaciones dependiendo si es de lazo abierto o cerrado o de la trayectoria o perfil de velocidad a seguir; no obstante, los elementos mostrados siempre están presentes. Para mayor detalle de la programación y el programa de Simulink para cada simulación, revisar el Apéndice I.

Cabe señalar que previo al arranque del programa de Simulink, se debe correr un archivo de MATLAB que contiene las dimensiones de la plataforma, así como otros parámetros constantes (velocidades máximas, condiciones iniciales, posición del objetivo, radio del área de evasión).

V.2 Elementos del sistema para pruebas con el modelo funcional

La figura V - 2 muestra un diagrama con los elementos del sistema y cómo fluye la información a través de éste. Está conformado por el modelo funcional, el sistema de visión artificial, el sistema de control y el módulo de radiofrecuencia.

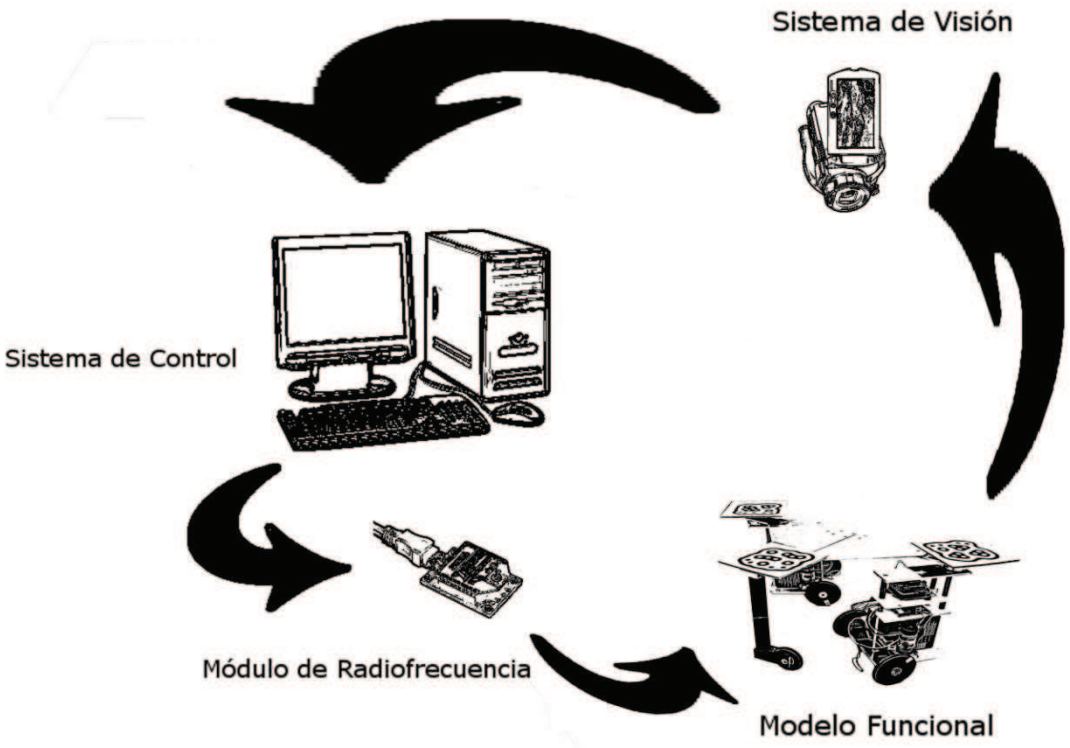


Figura V – 2 Diagrama del sistema para pruebas.

El modelo funcional es en sí el robot a controlar, con la instrumentación necesaria que le permita comunicarse con los demás elementos del sistema. El espacio de trabajo en el que se mueve la plataforma está delimitado por el área cubierta por la cámara del sistema de visión. El software utilizado por el sistema de visión otorga los datos de realimentación necesarios para el sistema de control. Éste último es el que procesa los datos de entrada para que se obtengan las velocidades de las ruedas necesarias para el desplazamiento deseado de la plataforma. Los datos se envían y reciben mediante módulos de radiofrecuencia Xbee; uno conectado al puerto serial de la computadora que emite datos y dos montados en cada uno de los robots de tracción de la plataforma que reciben los comandos de velocidad.

Cada uno de los elementos del sistema es por sí solo un subsistema que cuenta con más elementos que son necesarios de detallar para comprender el completo funcionamiento del sistema de pruebas.

V.2.1 Modelo funcional

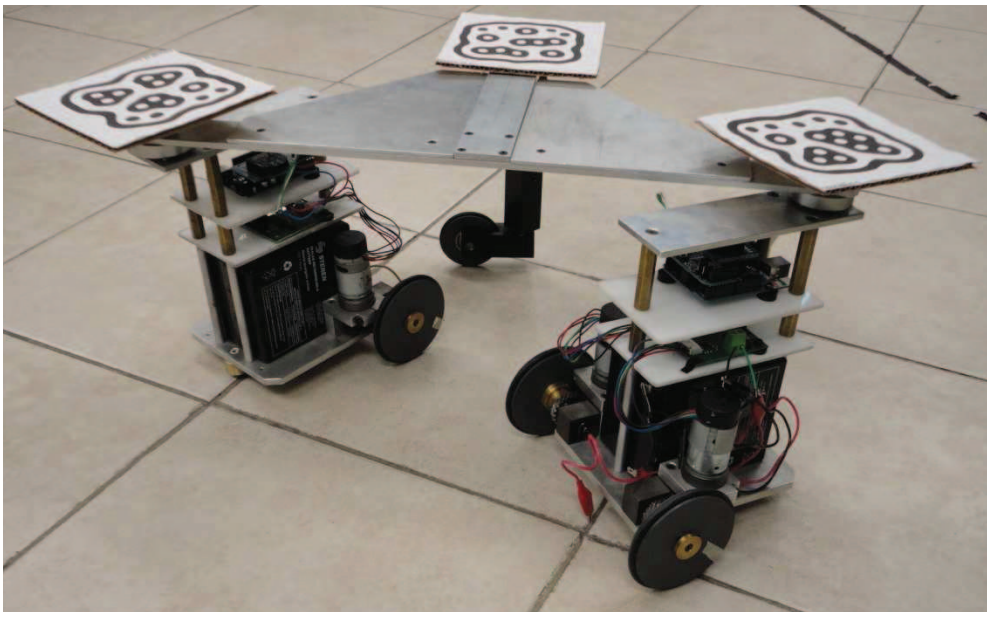


Figura V – 3 Fotografía del modelo funcional.

La figura V - 3 muestra todos los componentes montados en el modelo funcional. Éste puede ser dividido en distintos elementos para facilitar su descripción. Dichos elementos son: plataforma, robots de tracción, ruedas, motores, elementos de soporte, elementos electrónicos, alimentación e identificadores para el sistema de visión.

V.2.1.1 Plataforma

La plataforma consta de tres cuerpos unidos a un elemento de soporte que en conjunto dan una forma trapezoidal. Dicha forma acentúa el arreglo triangular de los puntos en los cuales descansan los identificadores para el sistema de visión. Además, permite una vista clara del movimiento de los robots de tracción y de la orientación de la plataforma durante sus movimientos. Las dimensiones son suficientes para evitar colisiones entre los robots de tracción al momento de girar y para desplazarse en el espacio de trabajo definido por el sistema de visión permitiendo movimientos largos y significativos para las pruebas.

V.2.1.2 Robots de tracción

Cada uno de éstos es un robot móvil convencional por sí solo, de arreglo diferencial (como se vio en el capítulo II). Cuentan con tres niveles de altura para el montaje de sus elementos. En el nivel más bajo se encuentra la batería y el sistema de tracción, el segundo nivel aloja a la tarjeta que controla los motores y en el tercer nivel descansa la tarjeta del microcontrolador que tiene adaptado el módulo Xbee para la recepción de datos.

V.2.1.3 Ruedas

El robot cuenta con distintos tipos de ruedas para su desplazamiento: Cuatro ruedas fijas necesarias para la configuración diferencial de los robots de tracción. Dos balines en la punta de cada robot de tracción para soportar y nivelar sin afectar la movilidad de los robots. Una rueda castora que da soporte al punto descentrado de la plataforma móvil y que no afecta a la cinemática de la plataforma móvil.

V.2.1.4 Motores

Figura V – 4 Motor EMG30.

La tracción de las ruedas está dada por cuatro motores EMG30 [10]. Son motores que funcionan con 12 volts de corriente directa, con encoder de efecto hall

integrado y una reducción 30:1. Su velocidad máxima sin carga, utilizando la tarjeta MD25 como controlador, es de 200 revoluciones por minuto y sus especificaciones nominales están dadas en la Tabla 1.

Tabla 1. Especificaciones del motor EMG30

Voltaje nominal	12 [V]
Torque nominal	1.5 [Kg/cm]
Velocidad nominal	170 [rpm]
Corriente nominal	530 [mA]
Velocidad sin carga	216 [rpm]
Corriente sin carga	150 [mA]
Corriente a motor bloqueado	2.5 [A]
Potencia nominal	4.22 [W]
Pulsos por vuelta del encoder	360

V.2.1.5 Elementos de soporte

Son los que unen las distintas partes de la plataforma móvil y dan rigidez a la estructura. Se tiene, como elemento principal, una barra rectangular a la que van atornilladas las placas de la plataforma, además de contar en sus extremos con juntas con rodamientos donde se acoplan los robots de tracción y permita el libre giro de éstos. Los robots de tracción también cuentan con una base con tres postes atornillados a ésta que sirven de soporte para los niveles superiores y para el eslabón que une a estos robots con la placa principal.

V.2.1.6 Tarjeta MD25

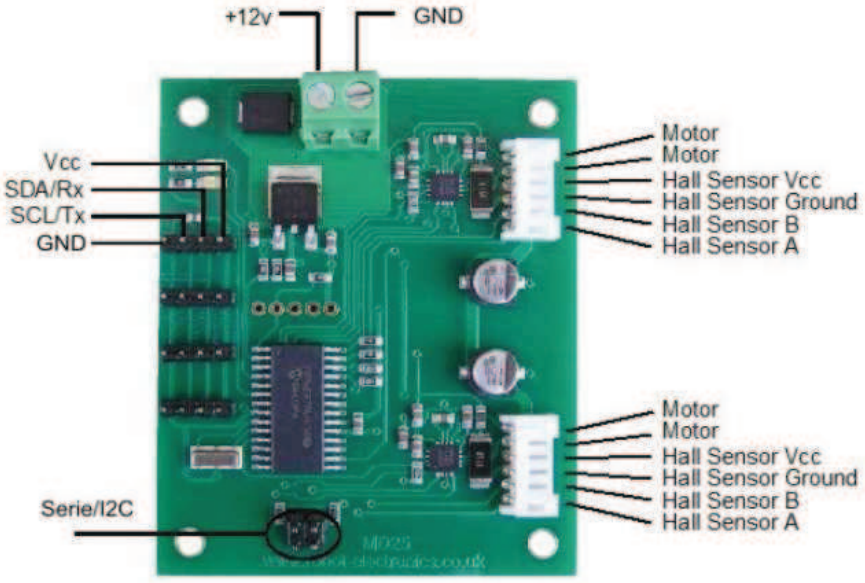


Figura V – 5 Tarjeta MD25.

La tarjeta MD25 [11] es un controlador de doble puente H desarrollado para el control de dos motores de DC modelo EMG30. Sus características principales son:

- Lectura de los encoders para determinar el desplazamiento de los motores y el sentido de giro de éstos
- Alimentación con un voltaje único de 12 volts de corriente directa
- Regulador de +5Vcc a 300mA para alimentar la circuitería externa
- Suministro de hasta 3ª de corriente por cada motor
- Interfaz serial o I^2C con posibilidad de conectar hasta 8 controladores MD25 en el mismo bus
- Control de potencia y aceleración.

Esta tarjeta facilita el control de los motores, gracias a los comandos preestablecidos que sólo requieren la activación de distintos registros. Se decide utilizar la interfaz I^2C para el control de la tarjeta, debido a que el microcontrolador a utilizar sólo cuenta con una terminal para comunicación serial, misma que se utiliza para la comunicación inalámbrica.

Para el control de los motores, existen distintas opciones configurables en la tarjeta, dependiendo de la aplicación y de la practicidad para el manejo de datos. Para este proyecto, los motores se manejan en modo independiente con datos de

entrada de velocidad que van de 0 a 255, siendo 0 la velocidad máxima en un sentido y 255 la velocidad máxima en sentido contrario.

V.2.1.7 Tarjeta Arduino Duemilanove

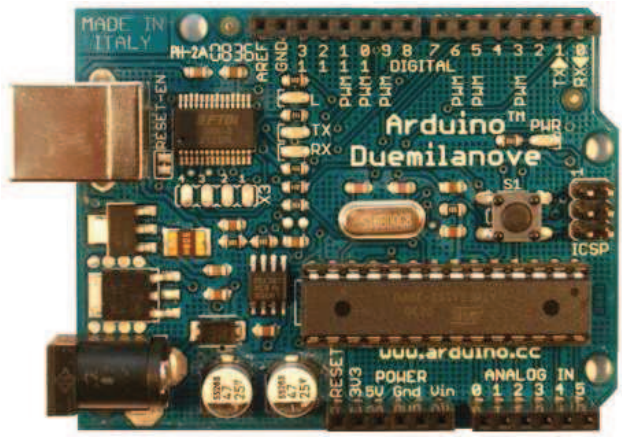


Figura V – 6 Arduino Duemilanove.

Arduino™ [12] es una plataforma de código libre basada en hardware y software flexible y fácil de usar. Cuenta con distintas versiones de tarjetas que trabajan con microcontroladores de la compañía Atmel de distintas capacidades, dependiendo de las necesidades requeridas.

En particular la tarjeta Arduino Duemilanove [13] (figura V - 6) trabaja con el microcontrolador Atmega328. Las características de la tarjeta son las siguientes:

Tabla 2. Especificaciones de la tarjeta Arduino Duemilanove

Voltaje de operación	5 V
Voltaje de entrada (recomendado)	7-12 V
Voltaje de entrada (límite)	6-20 V
E/S digitales	14 (6 permiten salida de PWM)
Entradas analógicas	6
Corriente directa por terminal E/S	40 mA
Corriente directa en terminal 3.3V	50 mA
Memoria Flash	32 KB (2KB utilizados para el bootloader)
SRAM	2 KB
EEPROM	1 KB
Velocidad de reloj	16 MHz

Además de las características especificadas en la Tabla 2, dos de los pines digitales (0 y 1) pueden utilizarse para la comunicación serial, mientras que otros dos (4 y 5) sirven para la comunicación I^2C.

Su programación es en lenguaje C, a través del software proporcionado por Arduino. Cuenta con numerosas bibliotecas para el manejo de las interrupciones del microcontrolador, la comunicación serial e I^2C, salidas de PWM, así como actuadores y funciones comunes, entre muchas otras más.

En el modelo funcional, esta tarjeta recibe de forma inalámbrica los comandos de velocidad de las ruedas ya procesados para enviarse a la tarjeta MD25. El microcontrolador se encarga de recibir datos del receptor inalámbrico a través de la comunicación serial a una velocidad de 9600 baudios, utilizando la librería *Serial.h*. También del correcto direccionamiento y envío de estos datos a la tarjeta MD25 mediante comunicación I^2C con la librería *Wire.h*. Para el correcto funcionamiento de esta librería, la conexión entre ambos dispositivos se realiza como en la figura V - 7. Aunque el protocolo dicta que se tengan resistencias de pull-up en los canales de comunicación, no es necesario colocarlas debido a que el Arduino Duemilanove ya cuenta con estas resistencias que entran en funcionamiento al momento de llamar a la librería *Wire.h*.

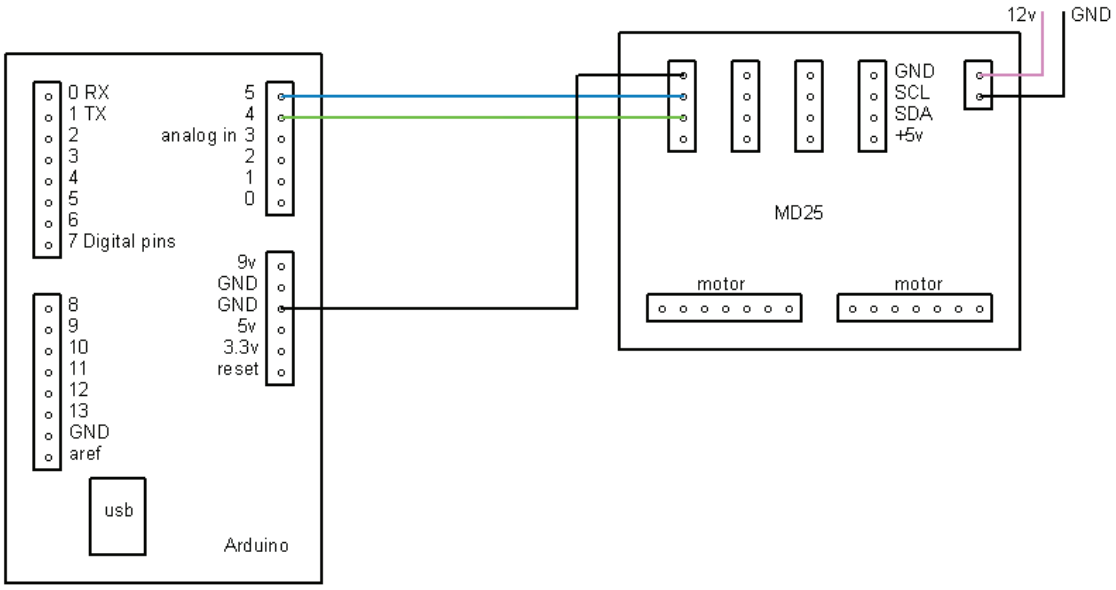


Figura V – 7 Conexión entre MD25 y Arduino Duemilanove.

Para observar el código a detalle de cada uno de los microcontroladores, véase el Apéndice II.

V.2.1.8 Xbee Shield

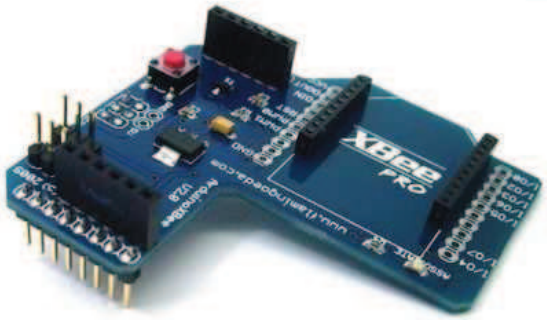


Figura V – 8 Xbee Shield.

Este es un módulo diseñado para el montaje del radio Xbee en la tarjeta Arduino Duemilanove o sus modelos posteriores [14]. Este módulo canaliza los datos recibidos por el radio a las terminales de puerto serie de la tarjeta Arduino.

V.2.1.9 Alimentación

Para la alimentación se utiliza una batería recargable de doce volts de corriente directa y cuatro ampere-hora en cada uno de los robots de tracción. Ésta se conecta a la tarjeta MD25, que a su vez alimenta a la tarjeta del microcontrolador a través de su terminal de alimentación de cinco volts para circuitería externa. El Xbee Shield se alimenta de la tarjeta del microcontrolador, debido a que cuenta con dos pines que se acoplan a las terminales de cinco volts y tierra del Arduino Duemilanove.

V.2.1.10 Identificadores para el sistema de visión

La plataforma cuenta con tres identificadores en su parte superior que son necesarias para identificar la postura de la plataforma móvil, así como del ángulo de giro de los robots de tracción. La forma particular de estas figuras es necesaria para que sean reconocidas por el software del sistema de visión artificial. Éste será explicado a detalle más adelante.

Además de los sistemas descritos anteriormente, se definen las dimensiones reales de la plataforma, que son necesarias para los cálculos de la cinemática inversa de la plataforma en la Tabla 3.

Tabla 3. Dimensiones de la plataforma móvil omnidireccional

A1	-23.3 [cm]
a2	22.7 [cm]
b	-20 [cm]
c	14.4 [cm]
d	8.8 [cm]
r	4[cm]

V.2.2 Sistema de visión

El sistema de visión permite observar desde una vista aérea a la plataforma móvil y a partir de la imagen obtenida ubicar puntos importantes de la plataforma y de los objetivos dentro del espacio de trabajo. El campo de visión que cubre la cámara utilizada es quien delimita el área donde se puede mover el robot. Si alguno de los identificadores sale del campo de visión, el sistema ya no puede reconocer su ubicación y por lo tanto no se puede tener un control de la plataforma. Además sirve para definir el sistema de ejes coordenados del universo.

V.2.2.1 Cámara

El sistema utiliza una cámara *Sony Handycam DCR-TRV361* para la captación de imágenes en tiempo real. Ésta se conecta de forma alámbrica a través del puerto firewire a la computadora que aloja al programa de reconocimiento de imagen. El

software cuenta con soporte, en Windows, para prácticamente cualquier cámara que utilice puerto USB, USB2.0 o firewire. Esta última opción es la que permite una transferencia más rápida de datos, dentro de los puertos de la computadora provista, lo que permite una velocidad más rápida de desplazamiento de la plataforma móvil. El área que alcanza a cubrir la cámara ya montada en el techo del laboratorio es de 1.96 x 1.47 [m].

V.2.2.2 Software reacTIVision

El software utilizado para el reconocimiento de imágenes para este proyecto es reacTIVision [15]. Es un software de código libre para el reconocimiento de marcadores *fiducial* (figura V - 9), así como de dedos sobre una superficie. Fue diseñado en un principio como el componente sensorial del *Reactable*, un sintetizador modular que cuenta con una serie de estándares para aplicaciones multi-touch.

La forma en que reacTIVision funciona es la siguiente: reacTIVision rastrea marcadores fiducial diseñados de forma especial, mediante video en tiempo real. La imagen fuente es convertida en una imagen en blanco y negro mediante un algoritmo de umbral adaptativo. Después esta imagen es segmentada en un árbol de regiones negras y blancas cambiantes (gráfica de región adyacente). En esta gráfica se buscan secuencias específicas que se codifican en un símbolo fiducial. Finalmente, las secuencias encontradas se relacionan mediante un diccionario de símbolos para obtener un número de identificación único. El diseño del fiducial permite el cálculo eficiente del centro del marcador, así como su orientación. Mensajes OSC que implementan el protocolo TUIO codifican la presencia del fiducial, su localización, orientación y número de identificación y se transmite la información a las aplicaciones cliente.

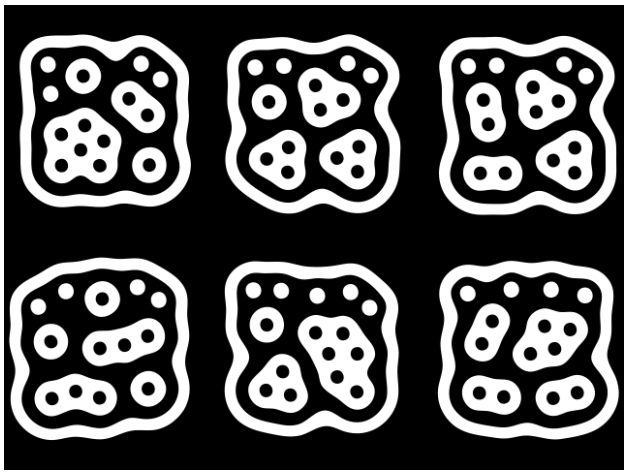


Figura V – 9 Ejemplo de marcadores fiducial.

V.2.2.3 Protocolo TUIO

ReacTIVision es una aplicación independiente, que envía mensajes TUIO a cualquier aplicación que esté habilitada con un cliente TUIO. El protocolo TUIO [16] fue implementado inicialmente en reacTIVision, aunque ha sido adoptado por otros proyectos relacionados con interacción al tacto y multi-touch.

Este protocolo provee una interfaz de comunicación general y versátil entre controladores de tacto y aplicaciones que requieran de estos datos. Fue diseñado para las necesidades de superficies multi-touch de mesas interactivas, donde el usuario es capaz de manipular una serie de objetos y trazar figuras en la superficie de la mesa con la punta de sus dedos. Los objetos son seguidos por un sistema de sensado y pueden ser identificados y localizados en cuanto a su posición y orientación en la superficie de la mesa. La figura V - 10 ilustra el diagrama de funcionamiento del sistema de mesa interactiva utilizando el protocolo TUIO.

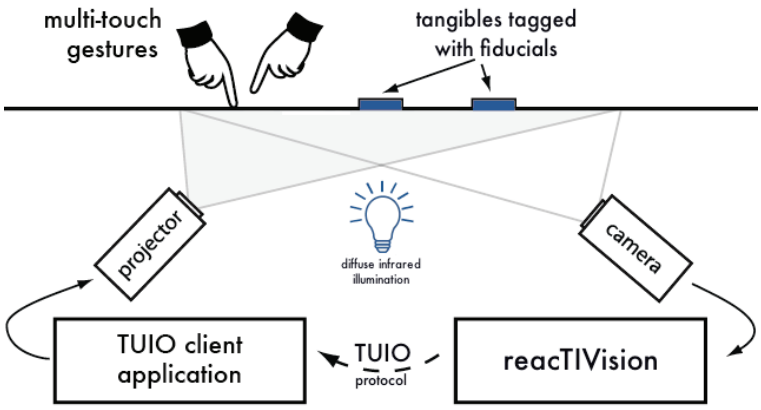


Figura V – 10 Diagrama de funcionamiento del protocolo TUIO.

V.2.2.4 Localización de la posición y orientación de un fiducial

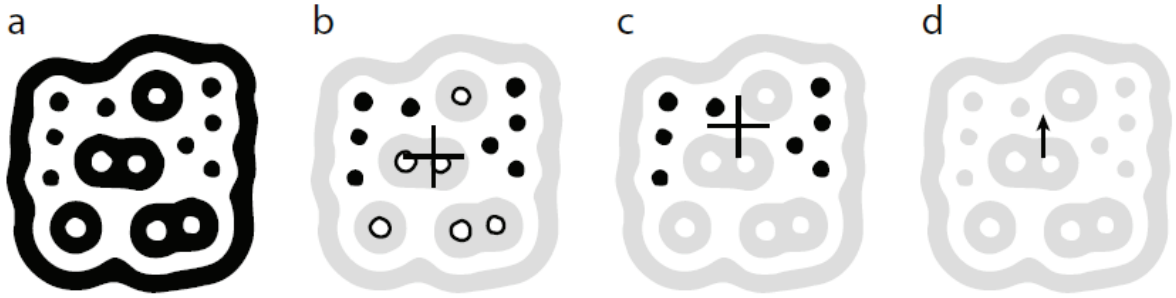


Figura V – 11 (a) Fiducial de reacTIVision (b) Hojas blancas y negras y su centroide (c) Hojas negras y su centroide (d) Vector para calcular la orientación.

La figura V - 11 ilustra de forma clara el algoritmo utilizado por el programa. Del fiducial completo se utilizan los elementos más pequeños (hojas de la gráfica de región adyacente), debido a que al ser los elementos más pequeños, es más probable que sus centros sean la información espacial más exacta que se tenga del fiducial. Después, obteniendo el centroide promedio de todas las hojas blancas y negras, se tiene el centro del símbolo y realizando el mismo procedimiento sólo para las hojas negras se obtiene un segundo centroide que genera un vector que va del centro del símbolo al centroide de hojas negras y que permite conocer la orientación del fiducial.

Aprovechando las capacidades de reconocimiento de marcadores fiducial, se coloca uno en el punto *P* (punto de análisis), otro en el punto P_1 y un tercero en el punto P_2. Del fiducial del punto *P* se obtiene la posición y orientación del sistema de ejes coordenados $\{X_P, Y_P\}$ con respecto al universo. De los otros dos puntos sólo es necesario saber la orientación, debido a que la posición con respecto a su sistema de ejes coordenados anterior es constante. En los casos en que se requiere alcanzar un objetivo mediante campos potenciales, se utiliza un fiducial más que funge como la meta a alcanzar. De este último sólo se requiere conocer su posición, dado que la orientación es irrelevante para un área de evasión circular.

Para que Simulink sea capaz de interactuar con reacTIVision, se agrega el archivo de cliente TUIO a la biblioteca de MATLAB. De esta forma, se pueden crear objetos de esta clase dentro de una función para recibir los parámetros obtenidos por reacTIVision que serán manipulados por el sistema de control. Además se realiza un ajuste de dimensiones para que los datos obtenidos concuerden con las dimensiones reales.

V.2.3 Sistema de control

El sistema de control es el que se encarga de recopilar la información del sistema de visión y los parámetros de referencia. Después, se introducen como datos de entrada para la ley de control que determina las velocidades lineal y angular requeridas para la realización del movimiento y éstos a su vez incluirlos en la cinemática inversa de la plataforma móvil para el cálculo de las velocidades de las llantas. La figura V - 12 ilustra de forma clara el proceso.

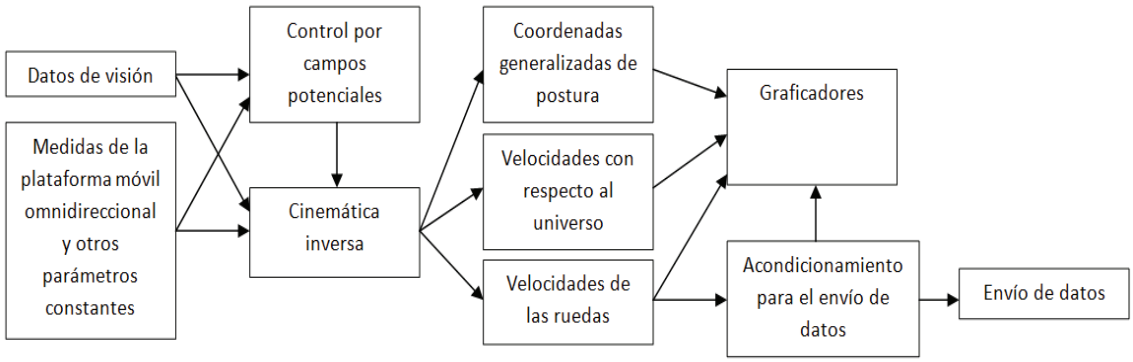


Figura V – 12 Diagrama del sistema de control.

Como se mencionó anteriormente, se adapta el programa de la simulación en Simulink para el control de la plataforma móvil en tiempo real. La retroalimentación de variables que existía en la simulación ahora se da por los datos obtenidos del sistema de visión, ya no del cálculo interno de la cinemática.

En el caso de las pruebas en lazo abierto no cuentan con la ley de control dentro del diagrama de bloques, mientras que en las demás sí influye.

Además se incluye el bloque de acondicionamiento de datos a enviarse inalámbricamente al robot, esto es, escalar las velocidades de las ruedas para tener valores entre 0 y 255, como lo descrito en la descripción de la tarjeta MD25. Además se convierten en datos enteros no signados de ocho bits, por ser el formato permisible por el bloque de comunicación serial de Simulink. Además se incluye un quinto dato que corresponde a un protocolo de corroboración de la trama a enviar (checksum), para que el receptor confirme que la información recibida es correcta.

Los graficadores sirven para almacenar los resultados obtenidos de las pruebas en variables y poder utilizarlos posteriormente para su análisis. También sirven para corroborar de forma visual y rápida el funcionamiento del modelo real y detectar errores que no saltan a primera vista al momento de observar el movimiento de la plataforma móvil omnidireccional.

Para observar la programación de cada una de las pruebas, véase el Apéndice III.

V.2.4 Módulo de radiofrecuencia

El módulo de radiofrecuencia es el que permite la comunicación inalámbrica entre el sistema de control y el modelo funcional. Esto abre la posibilidad de tener un control remoto de la plataforma móvil y de aprovechar la velocidad y versatilidad de una computadora personal para el control del robot. En robótica móvil esto resulta de gran utilidad, dado que el robot está hecho para desplazarse, por lo que resulta poco práctico conectar cables que cubran todo el espacio de trabajo y que no se enreden o entorpezcan el movimiento del robot. En particular, para la plataforma móvil omnidireccional es imprescindible contar con comunicación inalámbrica, debido a que los robots de tracción giran por debajo de la plataforma y de forma independiente, lo que imposibilita el uso de una conexión por medio de cables.

Para la comunicación inalámbrica se decide utilizar módulos de radiofrecuencia Xbee Serie 2 [17]. Esta serie fue diseñada para operar con el protocolo ZigBee y soportar las necesidades de redes inalámbricas de bajo costo y baja potencia. Los módulos consumen un nivel mínimo de potencia y proveen una entrega confiable de información entre los dispositivos remotos.

Figura V – 13 Módulo Xbee.

El protocolo Zigbee [18] fue creado por las compañías pertenecientes a la Alianza Zigbee. Corresponde a una tecnología inalámbrica desarrollada como un estándar global y abierto para cumplir con las necesidades descritas anteriormente para el módulo Xbee. Este protocolo opera bajo las especificaciones físicas del estándar IEEE 802.15.4. Provee comunicación en ambientes hostiles de radiofrecuencia que son comunes en las aplicaciones comerciales e industriales. Además soporta una amplia gama de topologías de red, como son punto a punto, punto a puntos múltiples o redes de malla.

Permite dos modos de configuración. El modo transparente (AT) donde los módulos dejan pasar toda la información enviada y el modo API, que requiere de la construcción de una trama específica como protocolo de comunicación. En este proyecto se utiliza el modo transparente por su sencillez y simplificación de código.

Una red personal (PAN) consiste de un coordinador y uno o más enrutadores y/o dispositivos finales. La red se crea cuando el coordinador elige un canal y un número de identificación de red (PAN ID). Una vez que éste inicia la red, puede dar permiso a otros enrutador o dispositivos finales para que se unan.

Cuando un enrutador o un dispositivo final se unen a la PAN, recibe una dirección de 16 bits y puede enviar o recibir información de otros dispositivos en la PAN. Los enrutadores y los coordinadores pueden otorgar permiso a otros dispositivos para que se unan a la red y pueden ayudar en el envío de información dentro de la red para asegurar que los datos sean encausados al dispositivo receptor correcto. Cuando un enrutador o un coordinador permiten la entrada de un dispositivo final, éste se convierte en un “hijo” de quien le dio acceso.

Los dispositivos finales pueden enviar o recibir información, pero no pueden encausar información de un nodo a otro ni dar acceso a otro dispositivo a la red. Sólo pueden comunicarse con el dispositivo que les dio acceso. El enrutador o coordinador “padre” pueden enviar información en representación del dispositivo final para que llegue a su destino. Estos elementos están pensados para ser alimentados por baterías y soportan modos de bajo consumo de energía.

Para configurar los módulos se utiliza el software X-CTU provisto por la empresa Digi. Éste es un programa gratuito y fue creado expresamente para la configuración de los módulos Xbee. La configuración en modo transparente se realiza de la siguiente forma:

Para el coordinador, se conecta el módulo Xbee a la computadora mediante un adaptador de USB. Dentro del programa X-CTU, en la pestaña *PC Settings* seleccionar el puerto en el que esté conectado el módulo (figura V - 14). Dar clic en el botón *Test/Query* para corroborar que esté bien la velocidad de flujo de datos, el control de flujo, número de bits, la paridad y el número de bits de parada. Si marca error, generalmente sólo es necesario cambiar la velocidad de transmisión.

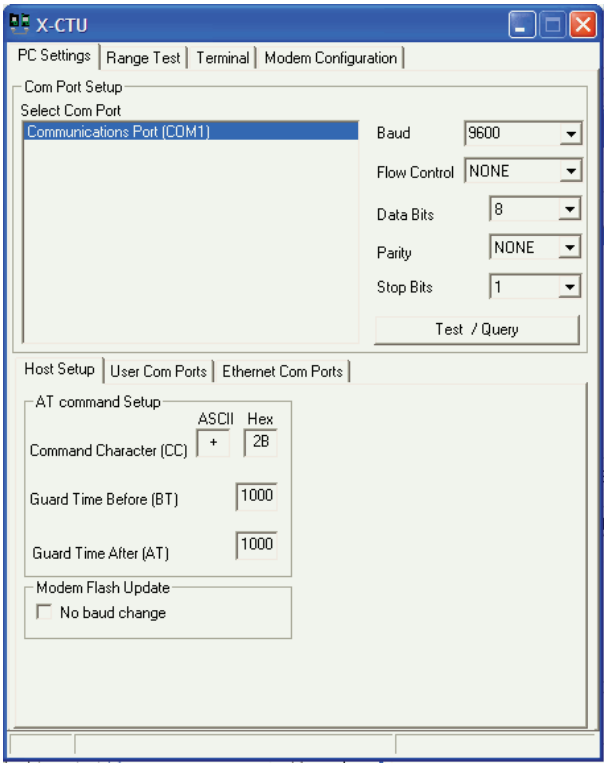


Figura V – 14 X-CTU PC Settings.

Una vez que la comunicación entre la computadora y el módulo sea correcta, ingresar a la pestaña *Modem Configuration* y presionar el botón *Read* para leer la

configuración que tenga previamente el módulo Xbee. Después, seleccionar en el menú *Function Set* la opción ***ZNET 2.5 COORDINATOR AT*** (figura V - 15).

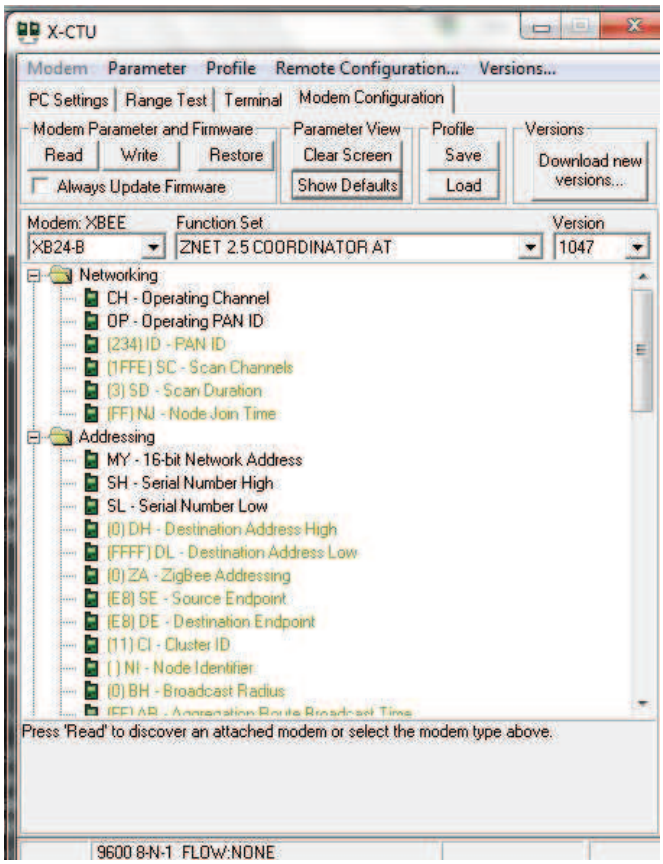


Figura V – 15 X-CTU configuración del coordinador.

Elegir el *PAN ID* de forma arbitraria y la velocidad de transmisión deseada en *Baud Rate*. Introducir la dirección destino del dispositivo a enviar información en *DH – Destination Address High* y en *DL – Destination Address Low*. El número que se introduce es el número de serie de 64 bits en formato hexadecimal que viene escrito en el módulo, en DL se escriben los bits menos significativos y en DH los más significativos. Se puede también utilizar la dirección de 16 bits que adquiere el dispositivo al unirse a la red, pero sólo se puede observar una vez que se une a la red, lo que vuelve el proceso más tardado, además de ser un número que puede cambiar con el tiempo, mientras que el número de serie nunca cambia.

Presionar el botón *Write* para que quede grabada la configuración en el módulo. Después presionar el botón *Read* nuevamente para corroborar que haya quedado bien grabada la configuración. Poner especial atención en el canal de operación *CH*; aunque no se pueda cambiar, éste es el canal al que se unirán los demás dispositivos. En caso de existir algún error de comunicación, puede ser debido a que el canal de operación y el PAN ID de los dispositivos en la red no coinciden.

Para el dispositivo final, realizar el mismo procedimiento, pero elegir en el menú *Function Set* la opción ***ZNET 2.5 ROUTER/END DEVICE AT***. Este dispositivo no tendrá el mismo canal que el coordinador hasta que se pongan en funcionamiento los dos módulos. Después de que ambos estén encendidos, el dispositivo final o el enrutador automáticamente buscan el canal de comunicación del coordinador y éste les da permiso.

En total se utilizan tres radios para el funcionamiento del sistema de pruebas:

- ✓ Un emisor conectado a la computadora mediante un módulo de comunicación vía puerto USB (coordinador).
- ✓ Un receptor y emisor en el robot de tracción 1 conectado a un Xbee Shield (dispositivo final).
- ✓ Un receptor en el robot de tracción 2 conectado a otro Xbee Shield (dispositivo final).

El coordinador envía los datos del sistema de control como se describió anteriormente al receptor del robot de tracción 1. Éste envía los datos del robot de tracción 2 al módulo receptor de dicho robot, junto con otro byte de corroboración de la trama (checksum).

Con esto finaliza la descripción de los elementos del sistema para pruebas con el modelo funcional de la plataforma móvil omnidireccional.

CAPÍTULO VI RESULTADOS DE LAS SIMULACIONES Y LAS PRUEBAS

Como se mencionó anteriormente, los experimentos a realizar fueron pensados para la clara apreciación visual de las propiedades omnidireccionales de la plataforma.

Los experimentos a realizar son los siguientes:

- ✓ Lazo abierto
  - Movimiento del punto de análisis en línea recta en diagonal sin giro de la plataforma
  - Movimiento circular del punto de análisis sin giro de la plataforma
- ✓ Lazo cerrado
  - Seguimiento de trayectoria rectilínea del punto de análisis con orientación de la plataforma a cero grados
  - Seguimiento de trayectoria rectilínea del punto de análisis con giro de la plataforma
  - Seguimiento de trayectoria circular del punto de análisis con orientación de la plataforma a cero grados
  - Seguimiento de trayectoria circular del punto de análisis con giro de la plataforma
  - Alcance de objetivo con orientación de la plataforma
  - Seguimiento de trayectoria dinámica con orientación de la plataforma

Los experimentos en lazo abierto permiten observar qué tanto afecta es controlable el robot. Esto permite conocer las causas de distintos errores que sean ajenos a la ley de control o errores en estado permanente. Por otro lado, los de lazo cerrado comprueban si el robot es estabilizable, además del funcionamiento y desempeño del mismo sistema de control. Además permite demostrar las capacidades de desplazamiento de la plataforma, dado que se pueden realizar pruebas con movimientos más complejos y controlados.

La realización de cada simulación y prueba con el modelo funcional se describe a continuación.

VI.1 Movimiento del punto de análisis en línea recta diagonal sin giro de la plataforma

Se realiza un movimiento rectilíneo con una inclinación de 45° con respecto al sistema *{X, Y}*. Para ello se introducen velocidades lineales constantes e iguales, tanto para v_{xp} como para v_{yp} y se elimina la velocidad angular ω. De forma arbitraria se selecciona la posición inicial de la plataforma y de los robots de tracción.

- Velocidades de entrada:
  - $v_{xp} = 5 \left[\frac{cm}{s}\right]$
  - $v_{yp} = 5 \left[\frac{cm}{s}\right]$
  - $\omega = 0 \left[\frac{rad}{s}\right]$

VI.1.1 Simulación

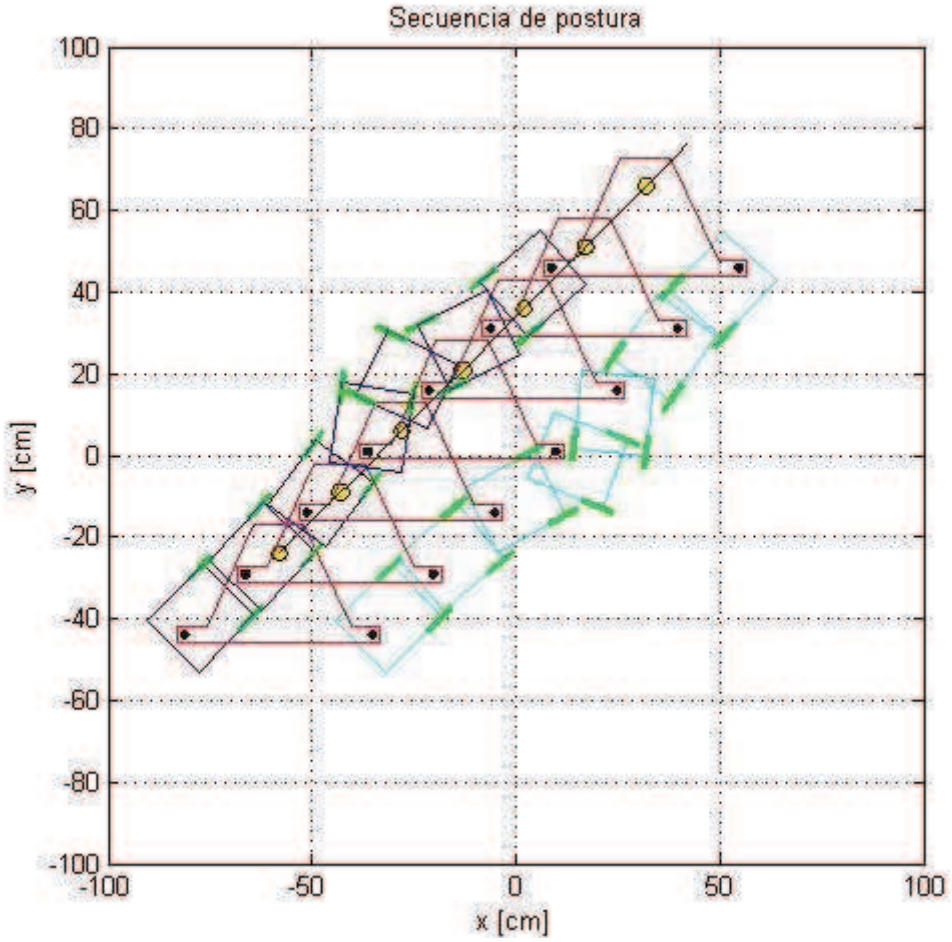


Figura VI – 1 Simulación movimiento rectilíneo sin giro en lazo abierto.

En la figura VI - 1 se observa cómo en la simulación la plataforma móvil logra realizar el movimiento rectilíneo con respecto al punto de análisis (círculo amarillo). Cabe destacar la orientación que adquieren los robots de tracción, que al final terminan alineados con la trayectoria.

VI.1.2 Modelo funcional

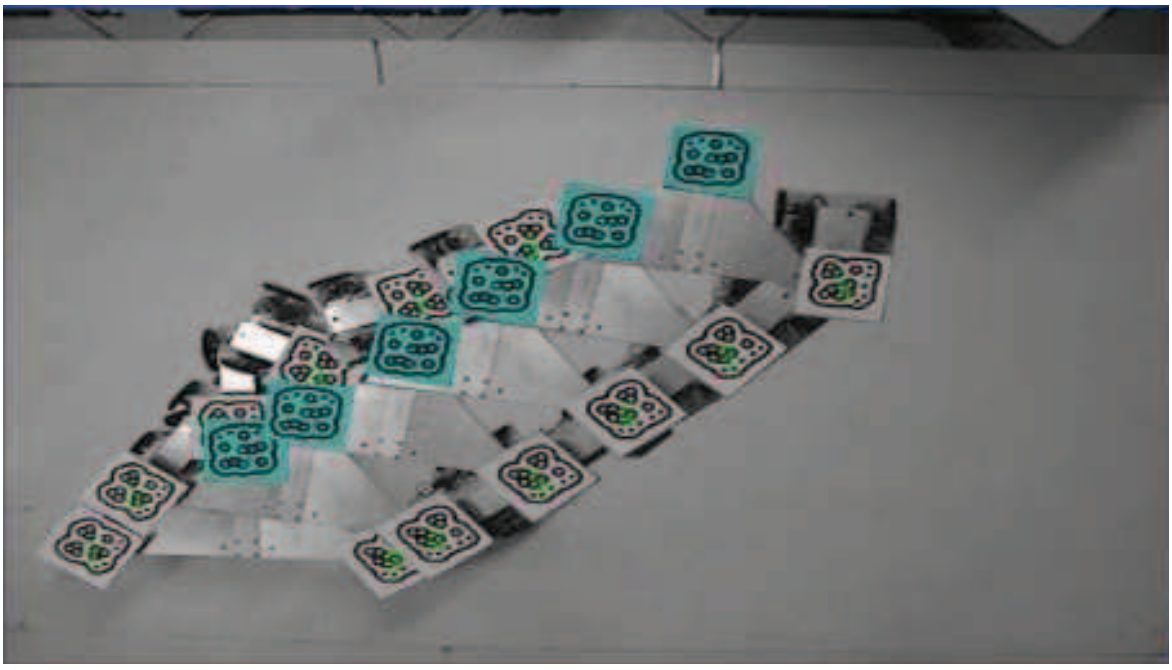


Figura VI – 2 Prueba movimiento rectilíneo sin giro en lazo abierto.

En la figura VI - 2 se observa cómo en la prueba real ya influyen las fuerzas dinámicas que no fueron contempladas. El marcador fiducial de color es el que está anclado al punto de análisis, mismo que se resalta para no perder de vista el movimiento. A diferencia de la simulación, se presenta una rotación involuntaria provocada por el derrape y/o estancamiento de las ruedas. Esto genera que la recta a 45° que tiene que describir en el movimiento, se pierda. Al principio del movimiento es más notorio este efecto, debido a una reacción ligeramente tardía del robot de tracción 2 al momento de vencer la fuerza de fricción de arranque. Esto es más visible en las siguientes gráficas.

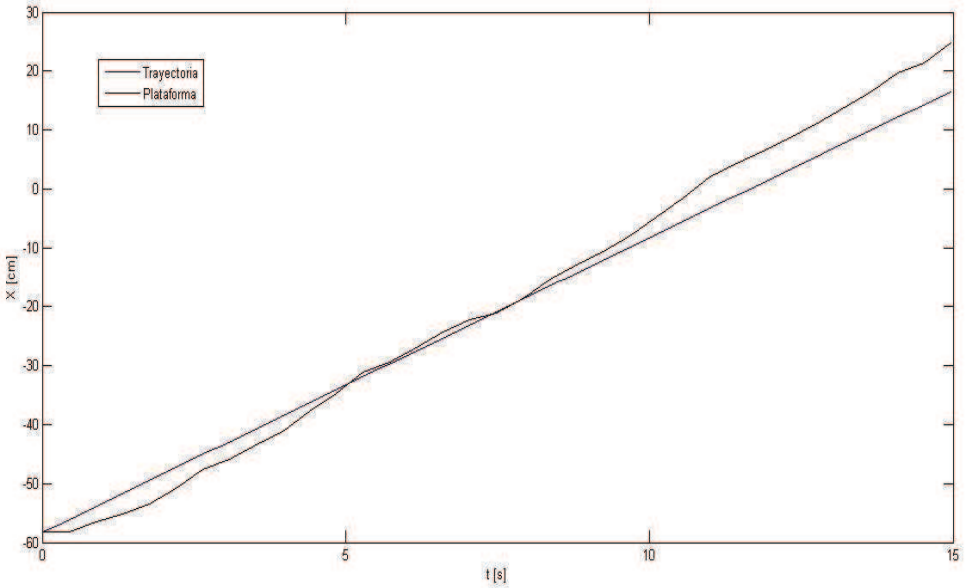


Figura VI – 3 Desplazamiento en el eje X (esperado y realizado).

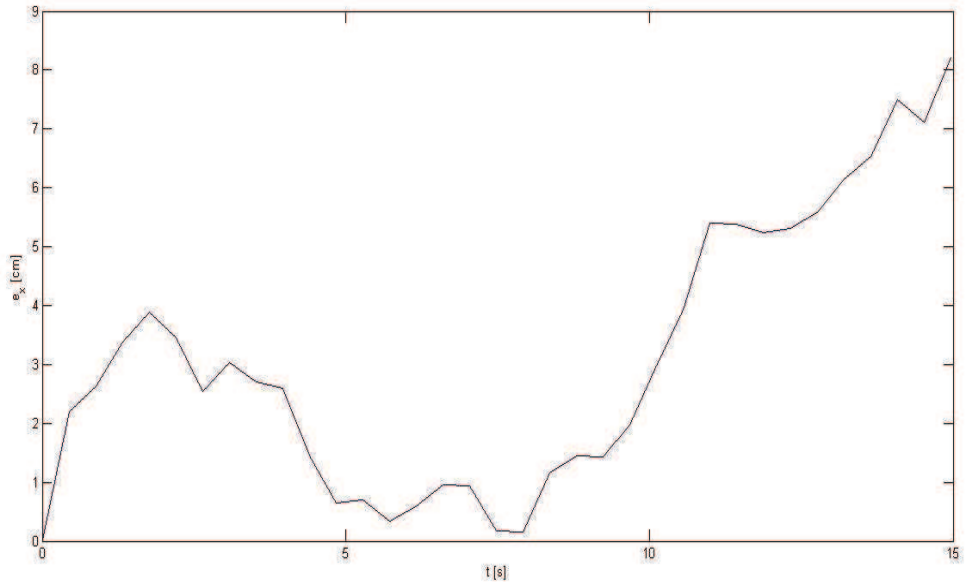


Figura VI – 4 Error absoluto de desplazamiento en el eje X.

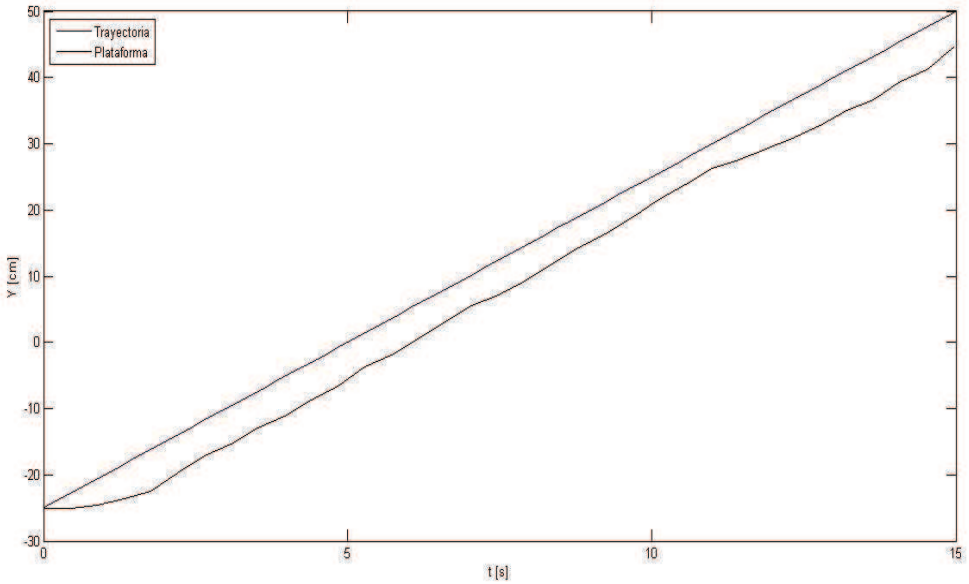


Figura VI – 5 Desplazamiento en el eje Y (esperado y realizado).

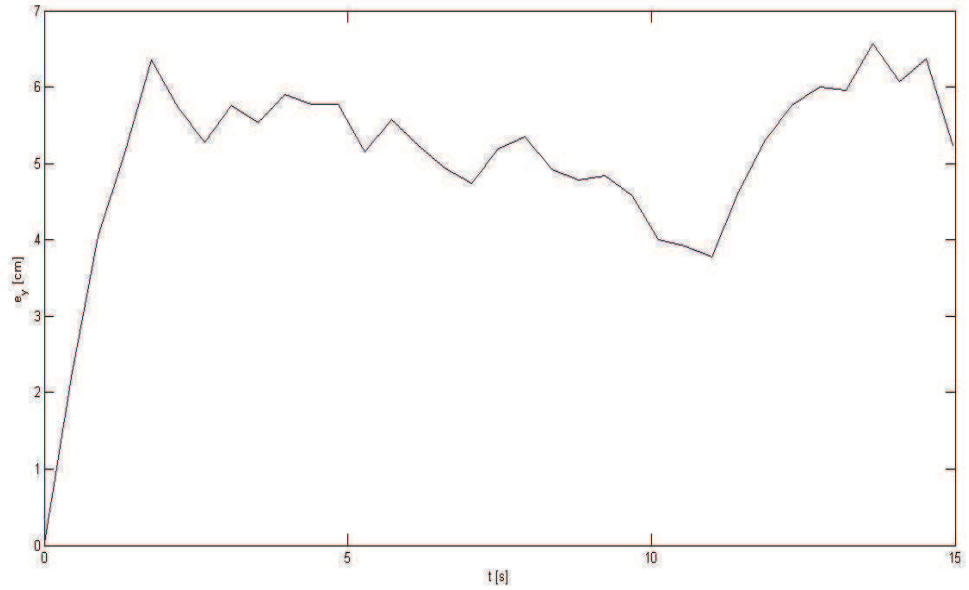


Figura VI – 6 Error absoluto de desplazamiento en el eje Y.

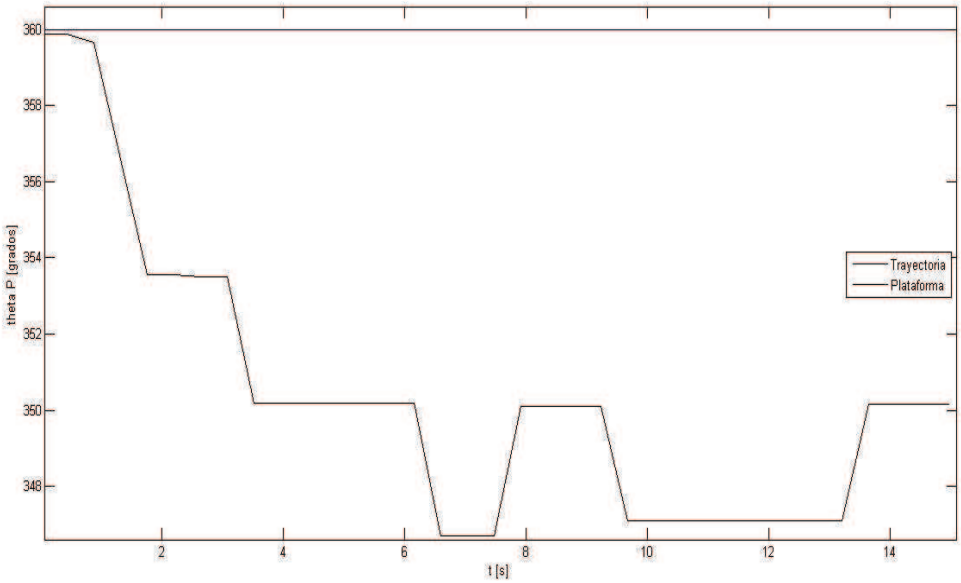


Figura VI – 7 Rotación de la plataforma (esperada y realizada).

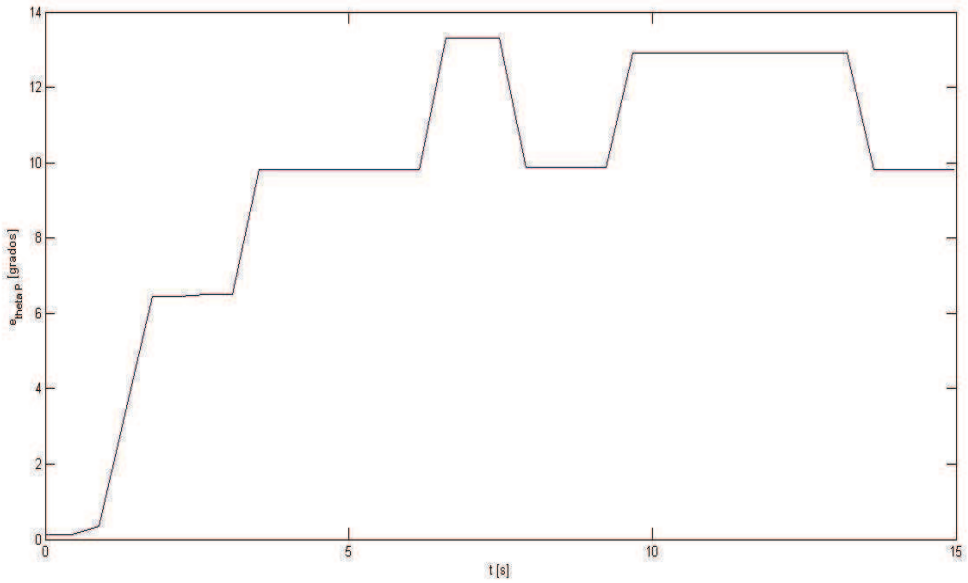


Figura VI – 8 Error absoluto de rotación.

En los datos de la prueba es más claro cómo la plataforma sí intenta describir la recta de 45°. No obstante, el principal problema es la rotación, ya que las velocidades de entrada están dadas con respecto a los ejes de la plataforma $\{X_P, Y_P\}$. Por esta razón, si la plataforma gira, seguirá avanzando a 45° con respecto a sí misma, pero no con respecto al universo.

Como era de esperarse, la variación del error durante toda la prueba es aleatoria. Esto deja en claro que pueden ser diversos los efectos que lo puedan producir. Como distinción, ninguna de las variables lineales excedió los diez centímetros de error y el ángulo después de su error inicial, termina por oscilar con una variación de tres grados.

VI.2 Movimiento circular del punto de análisis sin giro de la plataforma

Se realiza un movimiento circular del punto de análisis con respecto al sistema *{X, Y}* sin giro de la plataforma. Este movimiento permite observar cómo la velocidad lineal puede cambiar sin alterar la orientación de la plataforma móvil. Para ello se introducen perfiles sinusoidales de velocidad a la entrada, uno desfasado 90° del otro. La velocidad angular se mantiene nula.

- Velocidades de entrada
  - $v_{xp} = 2.5\sin(\omega t)\left[\frac{cm}{s}\right], \omega = \frac{\pi}{10}$
  - $v_{yp} = 2.5\cos(\omega t)\left[\frac{cm}{s}\right], \omega = \frac{\pi}{10}$
  - $\omega = 0\left[\frac{rad}{s}\right]$

VI.2.1 Simulación

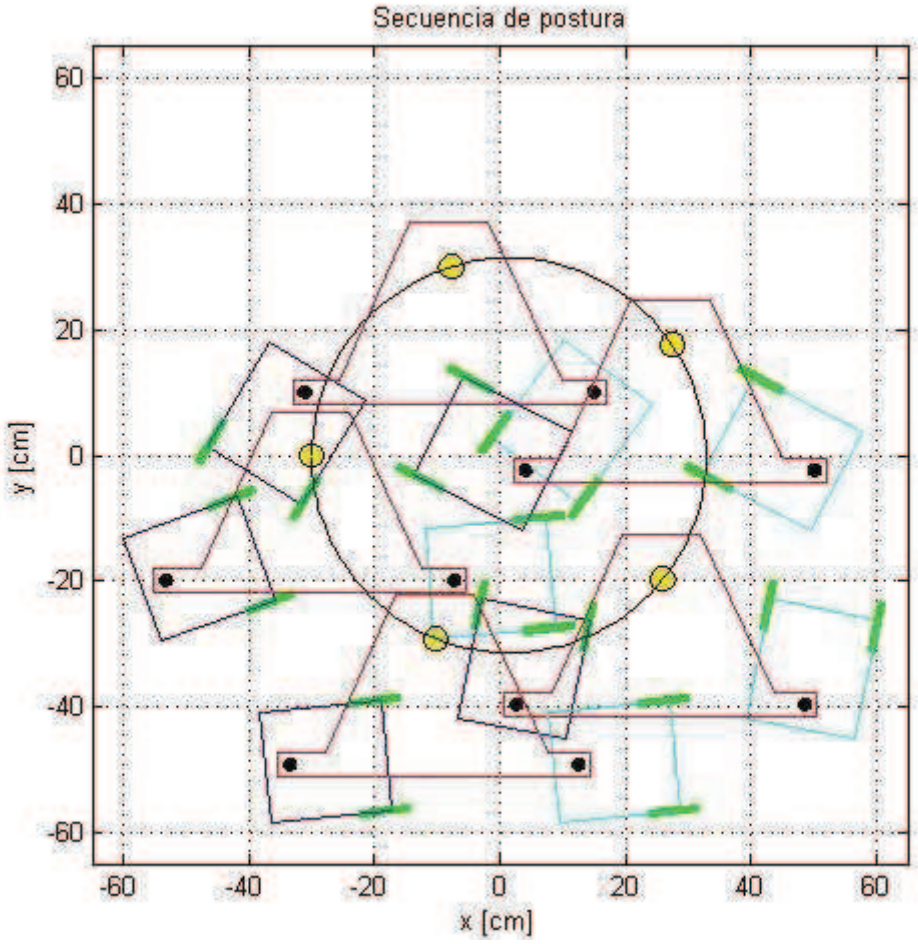


Figura VI – 9 Simulación movimiento circular sin giro en lazo abierto.

La simulación realiza el movimiento solicitado, como se puede observar en la figura VI - 9, donde éste comienza en las coordenadas (-30,0). La trayectoria señalada por la línea negra toca en todo momento al punto de análisis. Una vez más es notoria la orientación de los robots de tracción, que tienden a avanzar paralelos entre sí.

VI.2.2 Modelo funcional

Figura VI – 10 Prueba movimiento circular sin giro en lazo abierto.

Una vez más se observa en la figura VI - 10 lo mismo que en la prueba anterior: una variación de ángulo que provoca la modificación de la trayectoria. En este caso, provoca un cambio de centro instantáneo de rotación de la plataforma, lo que genera un desplazamiento del círculo dibujado por el punto de análisis. Cabe destacar que el movimiento de los robots de tracción coincide con el de la simulación. A continuación se muestran las comparaciones de la trayectoria ideal y la descrita por la plataforma y las gráficas de error.

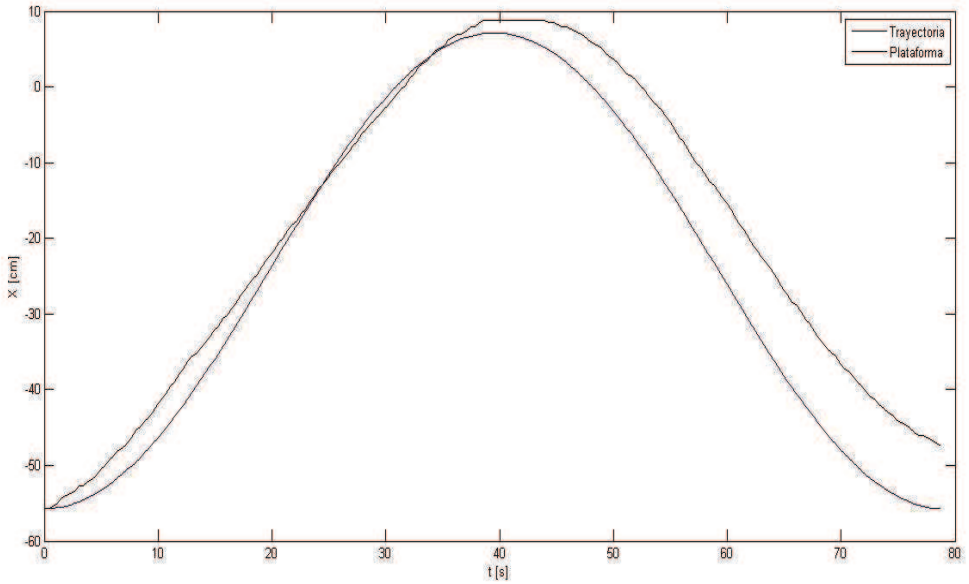


Figura VI – 11 Desplazamiento en el eje X del punto de análisis (esperado y realizado).

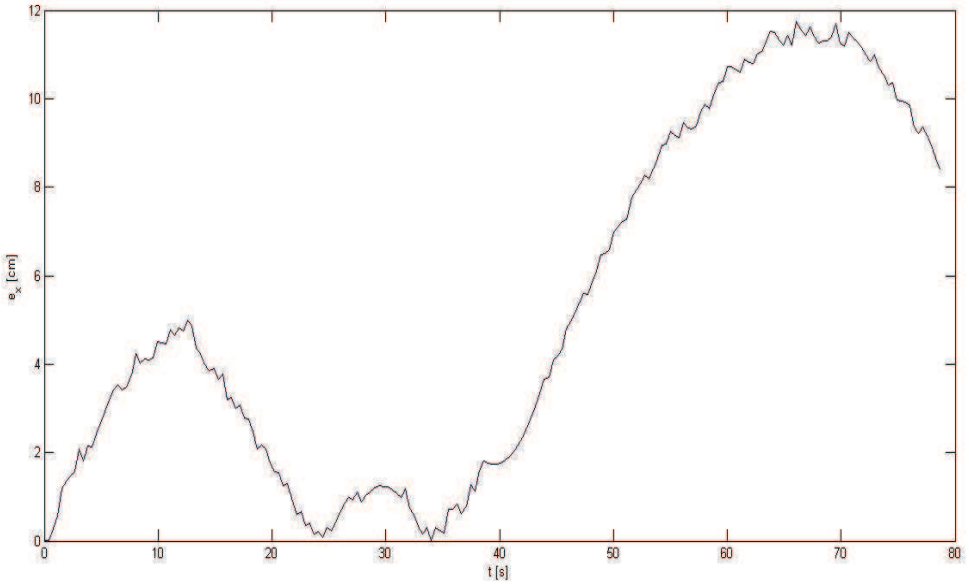


Figura VI – 12 Error absoluto de desplazamiento en el eje X.

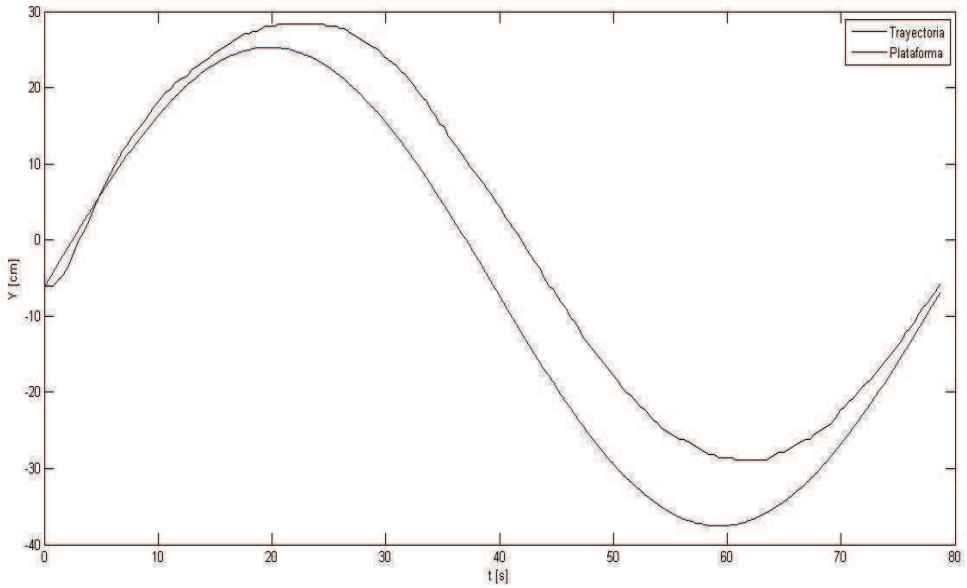


Figura VI – 13 Desplazamiento en el eje Y del punto de análisis (esperado y realizado).

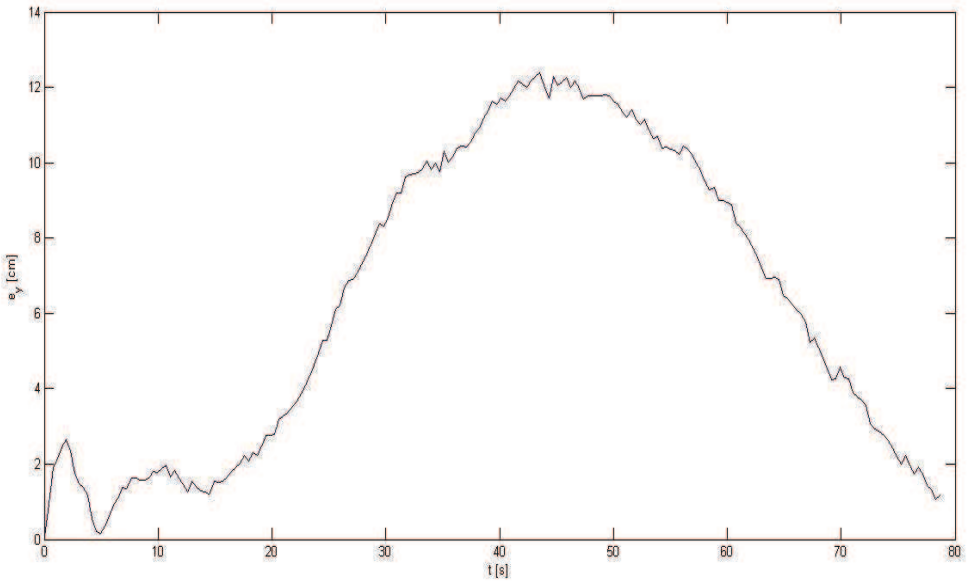


Figura VI – 14 Error absoluto de desplazamiento en el eje Y.

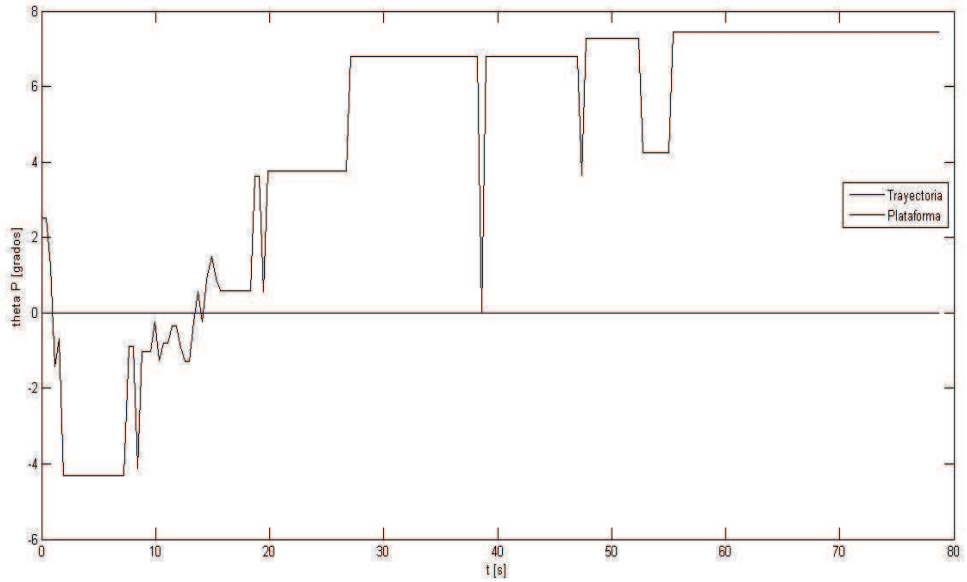


Figura VI – 15 Rotación de la plataforma (esperada y realizada).

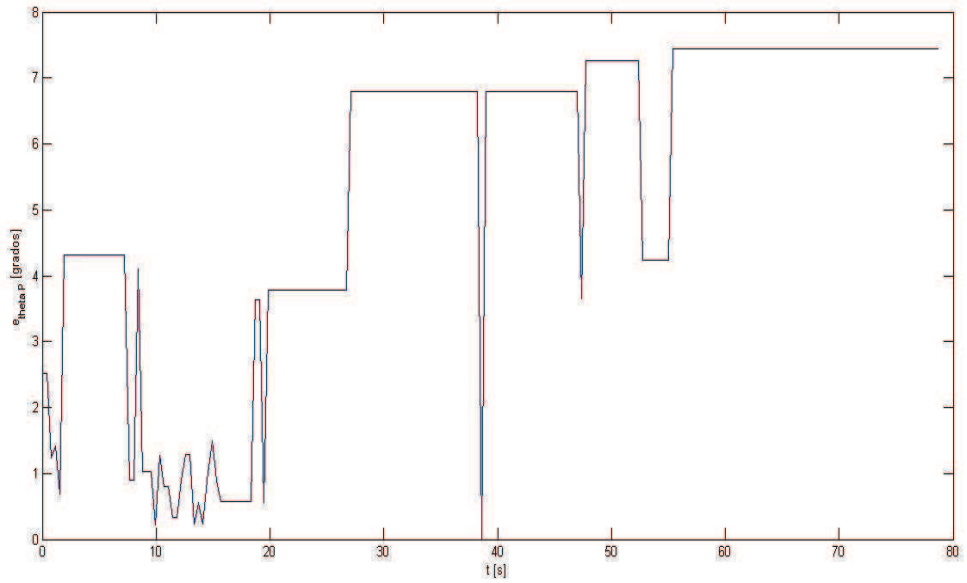


Figura VI – 16 Error absoluto de rotación.

Dado que el movimiento a realizar es más complejo, era de esperarse que el error tendiera a ser más grande. No obstante, no tiende a aumentar en demasía y en el desplazamiento lineal se observa cierta periodicidad que coincide con el tipo de movimiento que se está realizando.

En esta prueba el ángulo oscila un poco más, probablemente debido al constante cambio de dirección que se tiene en el movimiento, lo que provoca una mayor probabilidad de derrape de las llantas.

Es importante recalcar el siguiente fenómeno: la frecuencia dada a las funciones de velocidad no corresponde con la reacción real de la prueba. No obstante, la plataforma móvil sí realiza el movimiento, sólo que a una frecuencia menor. Esto es más notorio en pruebas posteriores. Después de todas ellas, se discutirá por qué sucede.

Con esto concluyen las pruebas en lazo abierto. Es importante tener en cuenta que el error no fue tan grande como para provocar una desviación excesiva del movimiento deseado. Se debe esperar que el error máximo en lazo cerrado se mantenga en niveles menores a éste.

VI.3 Seguimiento de trayectoria rectilínea del punto de análisis con orientación de cero grados de la plataforma

Se traza una trayectoria recta de 45° como en la prueba de lazo abierto, con la diferencia de que es una recta fija en el plano *XY* que comienza en el punto (-30,-30). El punto objetivo requerido por la ley de campos potenciales pertenece a la recta de la trayectoria y es cambiante con respecto al tiempo, por lo que se rige por el siguiente sistema de ecuaciones:

$$\begin{cases} x_t = 7t - 30 \\ y_t = 7t - 30 \end{cases}$$

(Ecuación 29)

Las ecuaciones de control de velocidad fueron definidas en el capítulo IV. Dado que se requiere de una orientación de la plataforma de cero grados con respecto al sistema de ejes coordenados *{X,Y}*, se utiliza la ecuación 27 para el control de velocidad angular.

- Parámetros
  - $v_{xmax} = 5\left[\frac{cm}{s}\right]$
  - $v_{ymax} = 5\left[\frac{cm}{s}\right]$

- $\omega_{max} = \frac{\pi}{30} \left[\frac{rad}{s}\right]$
- $k_v = 7\ [cm]$
- $k_{OWR} = 0\ [cm]$
- $\theta_p = 5°$

VI.3.1 Simulación

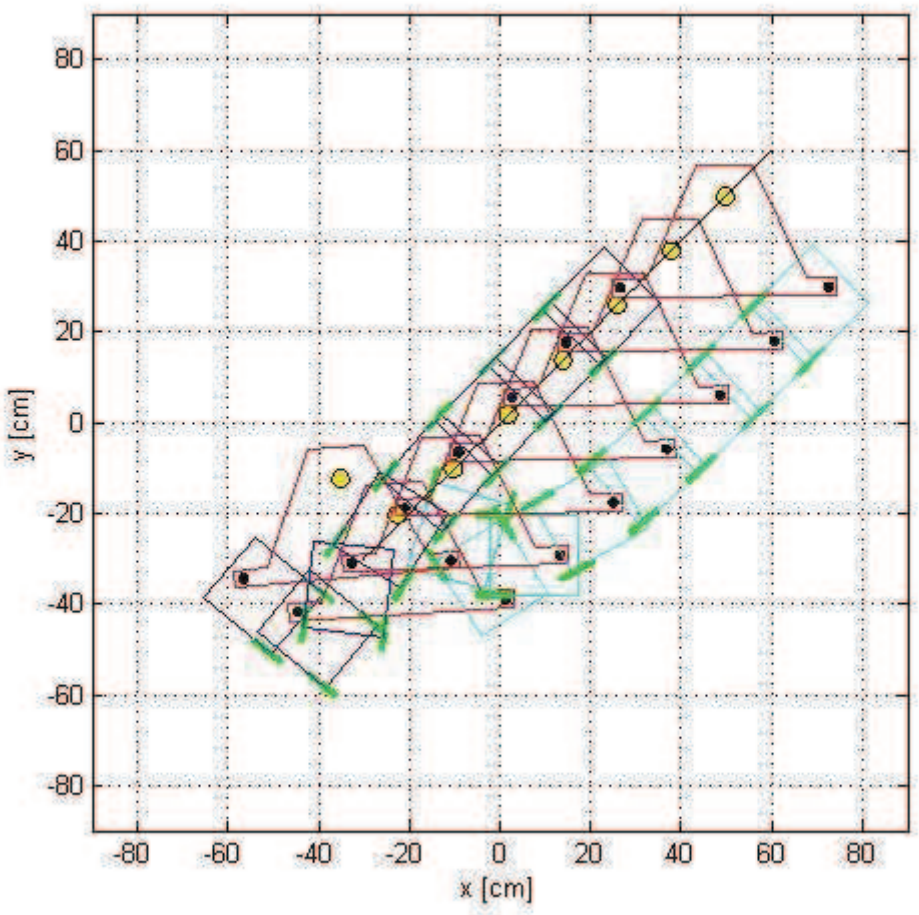


Figura VI – 17 Simulación de seguimiento de trayectoria rectilínea con orientación fija en lazo cerrado.

En comparación con la prueba de lazo abierto, la plataforma comienza en una postura cualquiera y es capaz de alcanzar la trayectoria y orientarse como se solicitó. La figura VI - 17 muestra cómo rápidamente la plataforma busca la línea de la trayectoria, conforme ésta sigue avanzando con pendiente positiva de 45°.

Es de esperarse que nunca alcance al punto que describe la trayectoria, puesto que éste avanza constantemente.

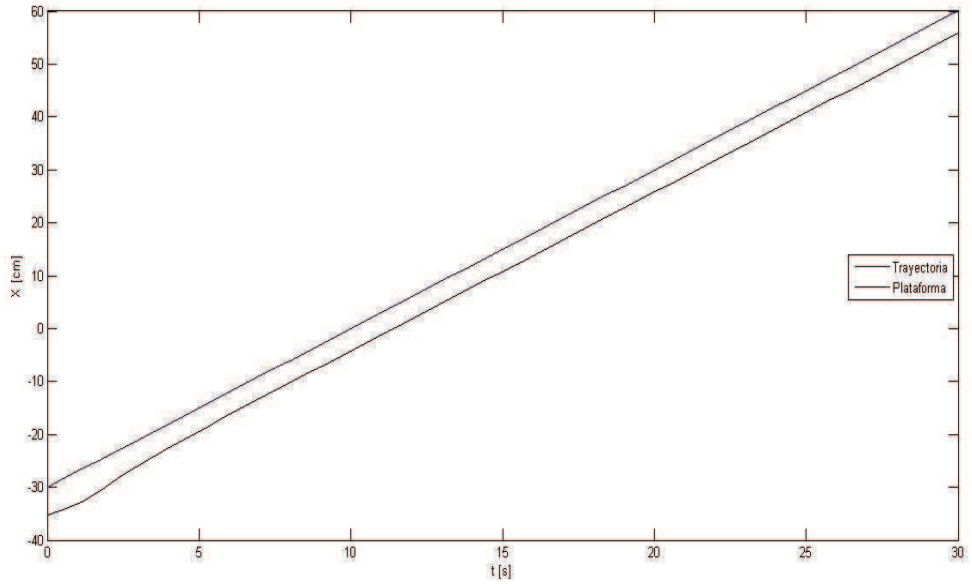


Figura VI – 18 Desplazamiento del punto de análisis en el eje X (a alcanzar y simulado).

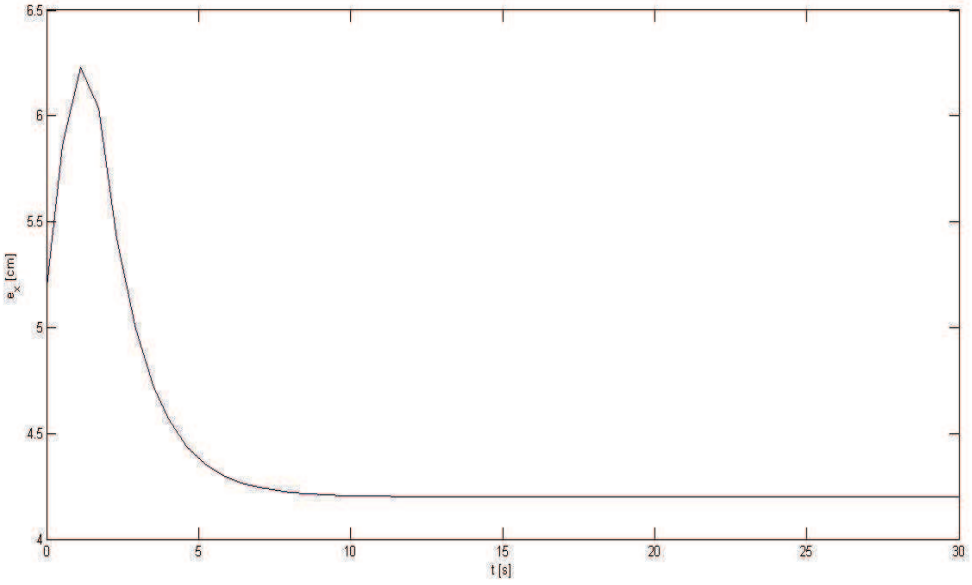


Figura VI – 19 Error de desplazamiento en el eje X.

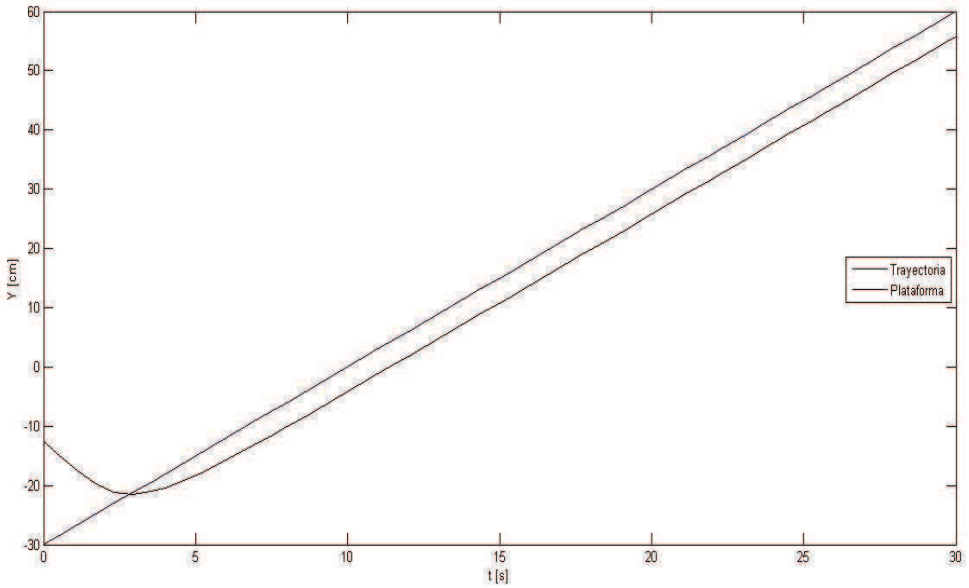


Figura VI – 20 Desplazamiento del punto de análisis en el eje Y (a alcanzar y simulado).

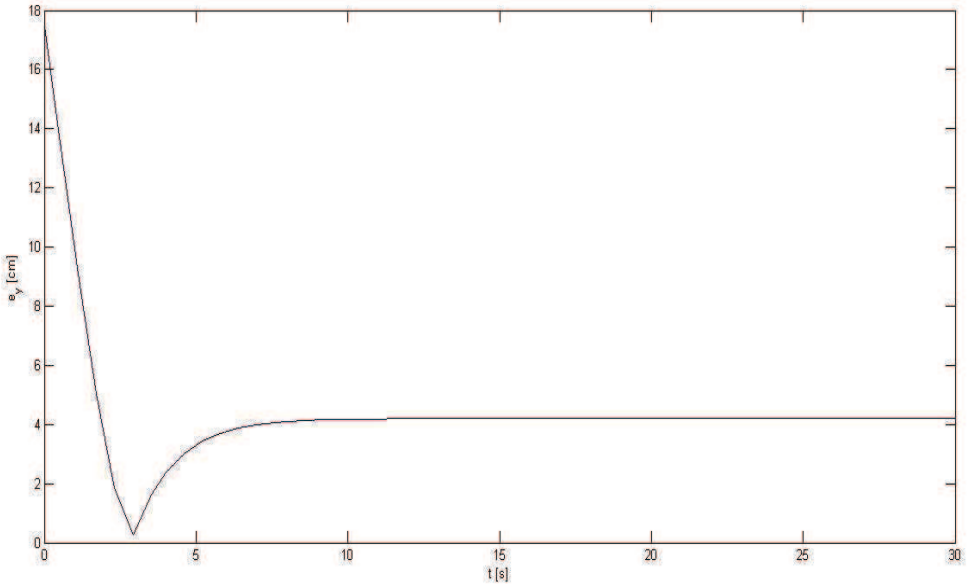


Figura VI – 21 Error de desplazamiento en el eje Y.

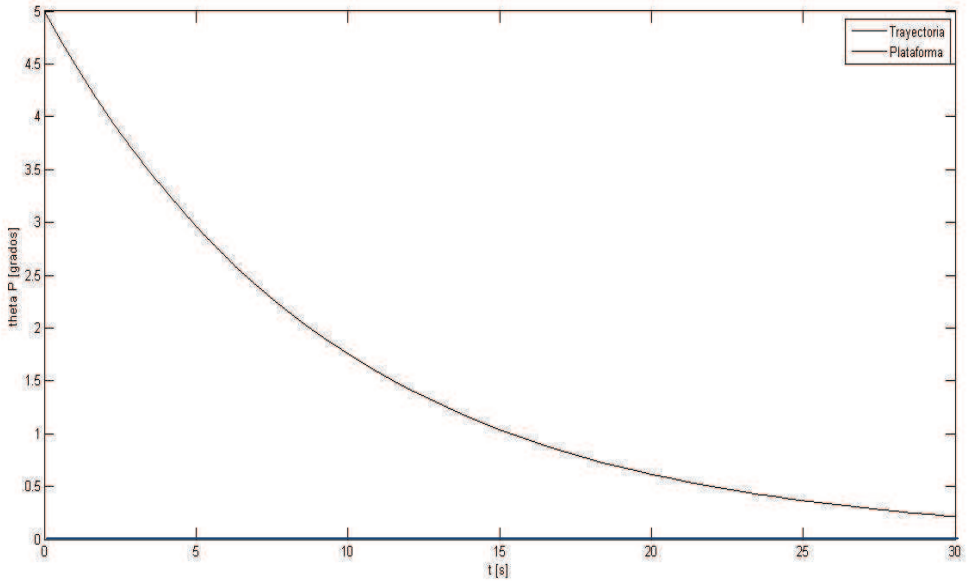


Figura VI – 22 Ángulo de rotación de la plataforma (a alcanzar y simulado).

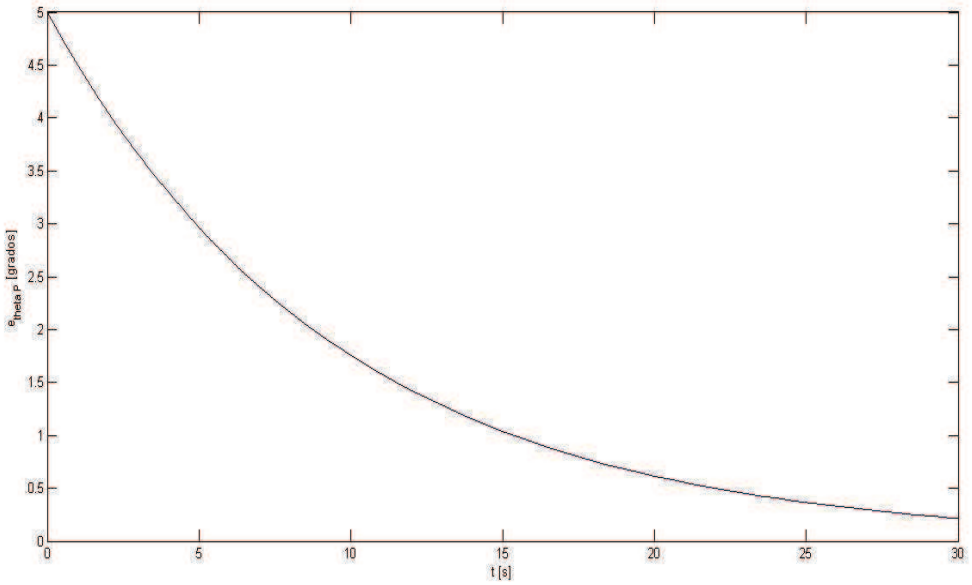


Figura VI – 23 Error del ángulo de rotación.

Es claro que la plataforma ya es capaz de corregir errores una vez que la acción de control entra en juego. Como el punto de análisis nunca alcanza al punto objetivo dentro de la trayectoria, el error en el desplazamiento lineal tiende a estabilizarse y el del ángulo de rotación de la plataforma tiende a cero, ya que éste sí es fijo. De esta forma se comprueba que de forma simulada la ley de control funciona de manera correcta.

VI.3.2 Modelo funcional

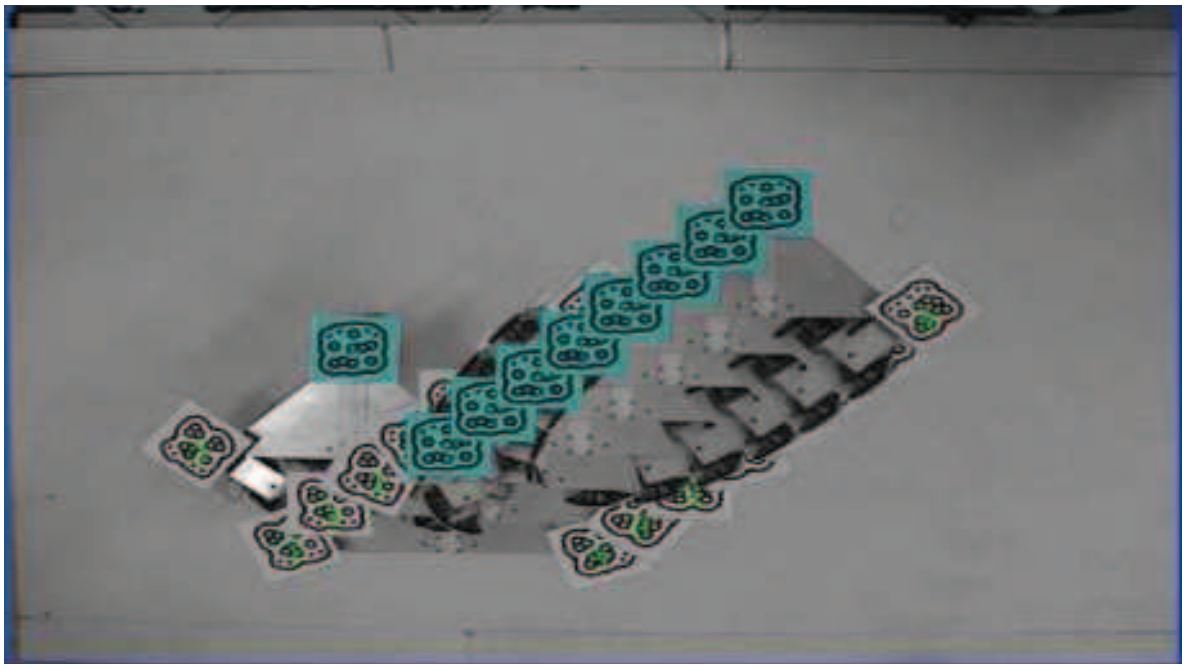


Figura VI – 24 Prueba de seguimiento de trayectoria rectilínea con orientación fija en lazo cerrado.

La figura VI - 24 muestra un movimiento bastante similar al obtenido por la simulación. No obstante, como se mencionó anteriormente, el error de posición tenderá a estabilizarse en otro valor que en la simulación, debido a la incongruencia en los tiempos de avance. En el caso de esta prueba, el avance del punto que describe la trayectoria es más lento que en la simulación, aunque la acción de control que corrige la postura de la plataforma sí reacciona de forma adecuada.

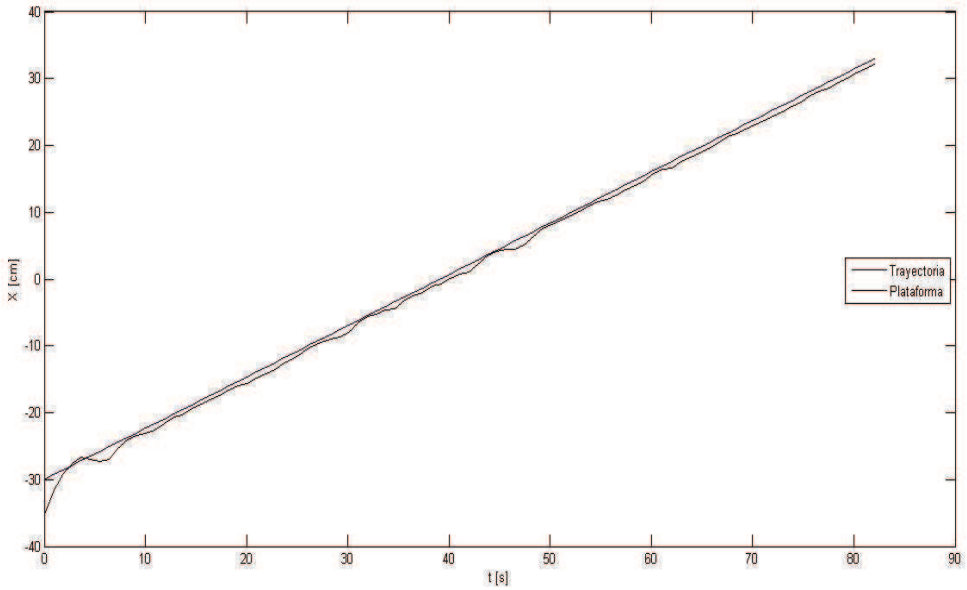


Figura VI – 25 Desplazamiento del punto de análisis en el eje X (a alcanzar y realizado).

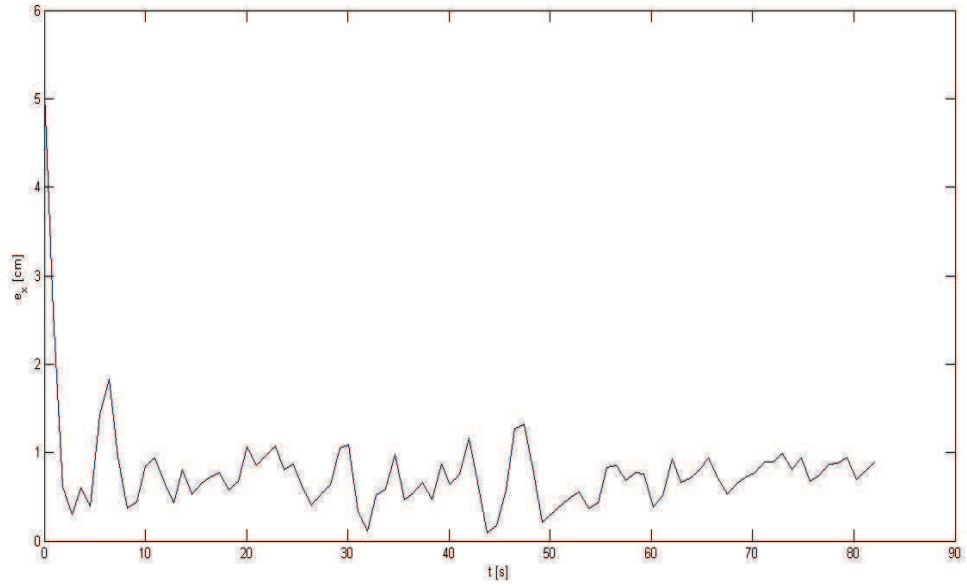


Figura VI – 26 Error de desplazamiento en el eje X.

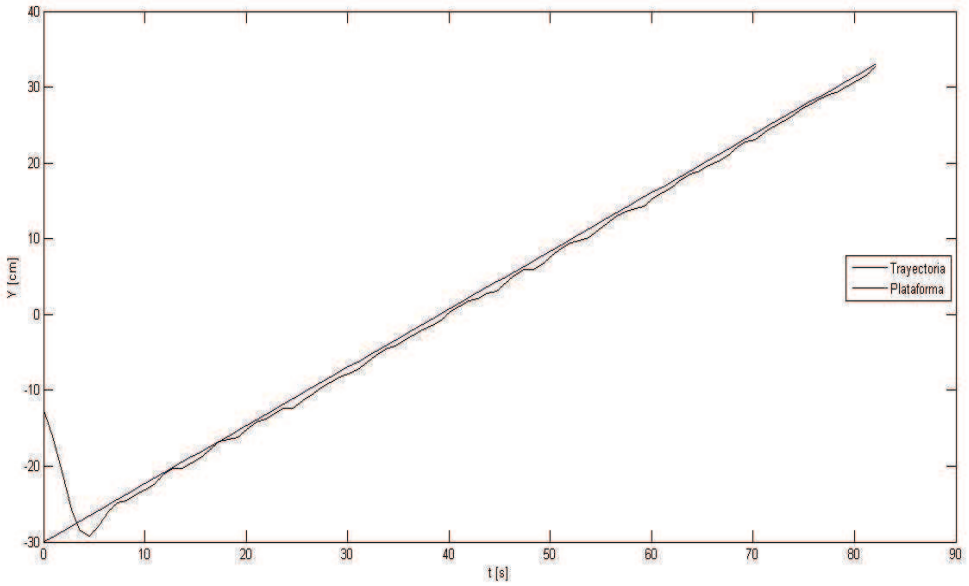


Figura VI – 27 Desplazamiento del punto de análisis en el eje Y (a alcanzar y realizado).

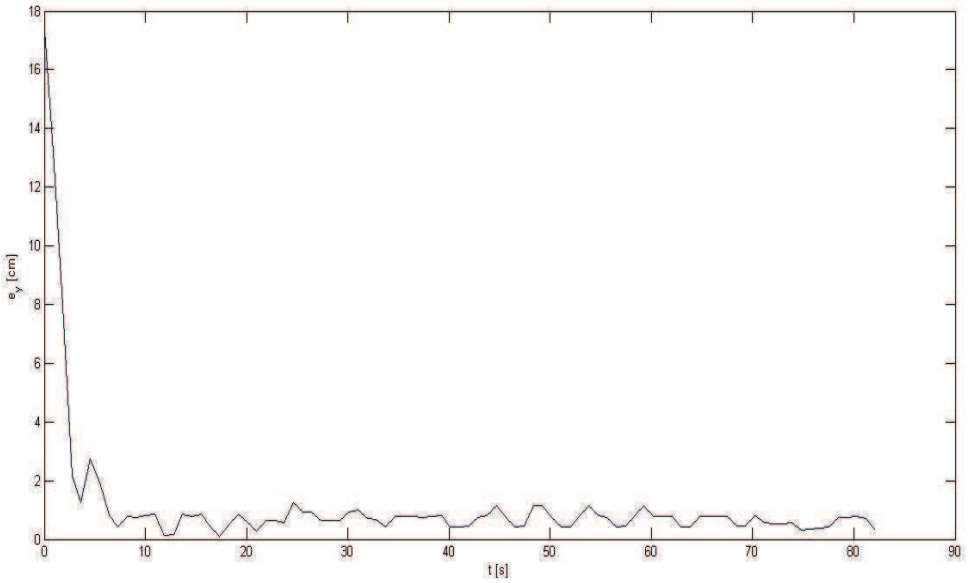


Figura VI – 28 Error de desplazamiento en el eje Y.

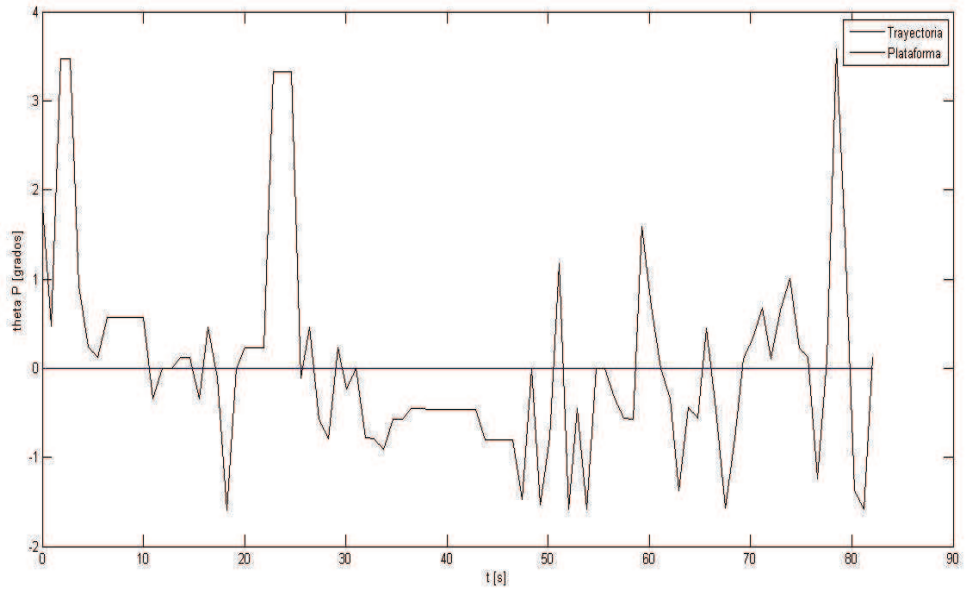


Figura VI – 29 Ángulo de rotación de la plataforma (a alcanzar y realizado).

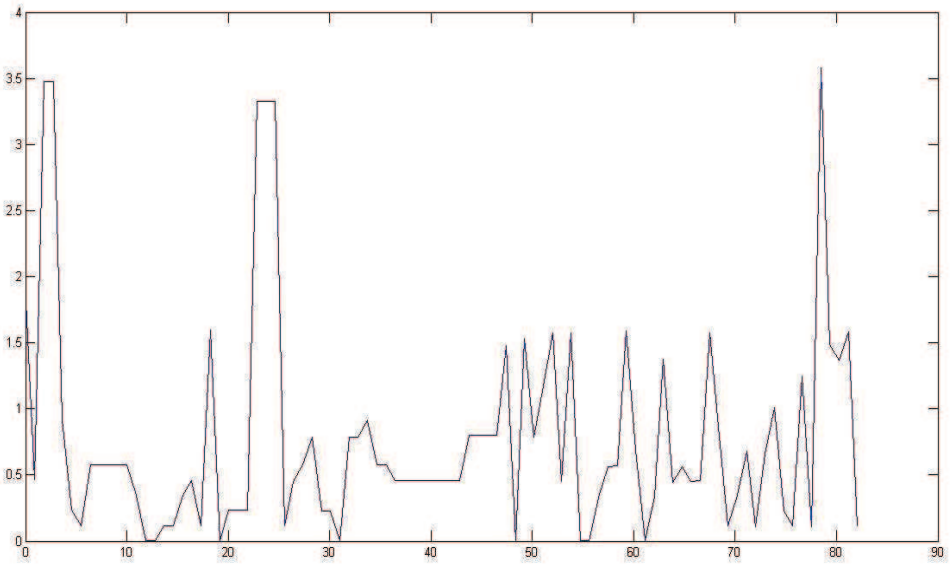


Figura VI – 30 Error del ángulo de rotación.

En cuanto a la posición se refiere, las primeras cuatro gráficas dejan muy claro cómo la ley de control sí actúa de forma adecuada, llegando a tener un error de aproximadamente un centímetro una vez que se estabiliza. Esto comprueba que los problemas que presenta Simulink, no afectan al proceso de cálculo y envío de datos, solamente al avance de la trayectoria.

Por otro lado, la acción de control de rotación reacciona rápidamente, lo que le permitió tener un error máximo de poco más de 3.5°. Esto contrasta bastante con los errores obtenidos en lazo abierto que eran tres o cuatro veces mayores.

VI.4 Seguimiento de trayectoria rectilínea del punto de análisis con giro de la plataforma

Se realiza la misma prueba que la anterior, con la excepción de que se agrega una velocidad de giro constante.

VI.4.1 Simulación

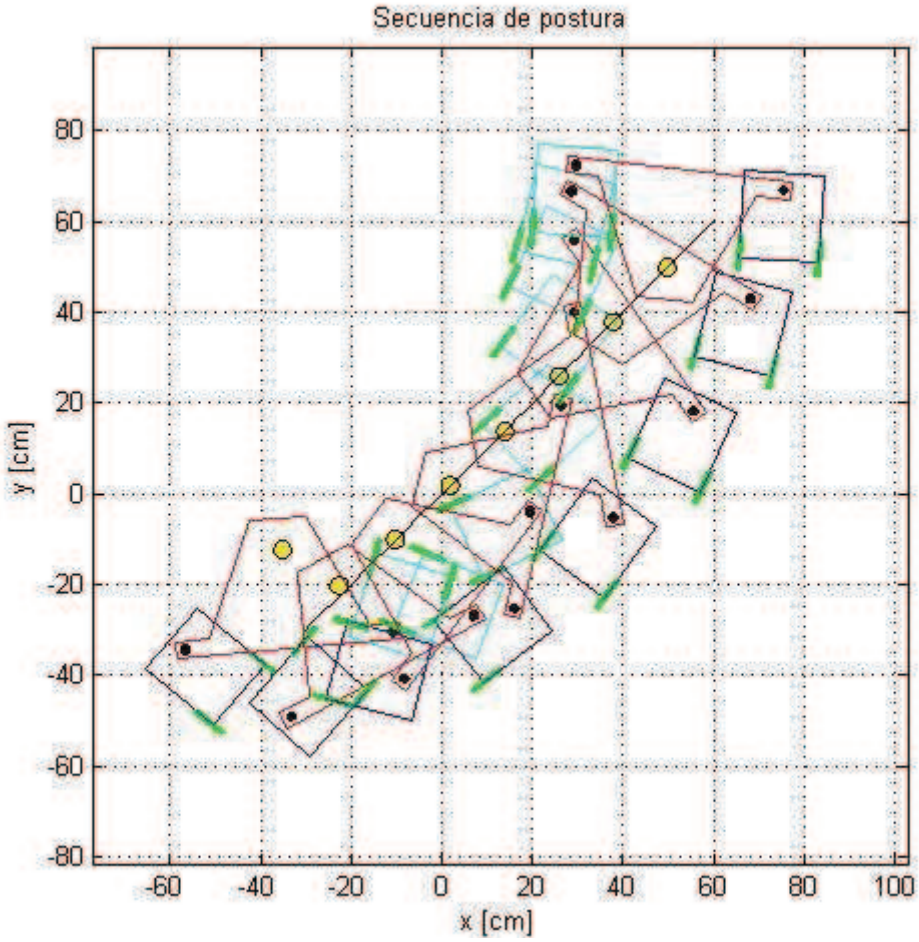


Figura VI – 31 Simulación de seguimiento de trayectoria rectilínea con giro.

La simulación muestra un movimiento más complejo que el anterior, pero el punto de análisis no deja de seguir la trayectoria de la misma forma que lo realizó anteriormente.

VI.4.2 Modelo funcional

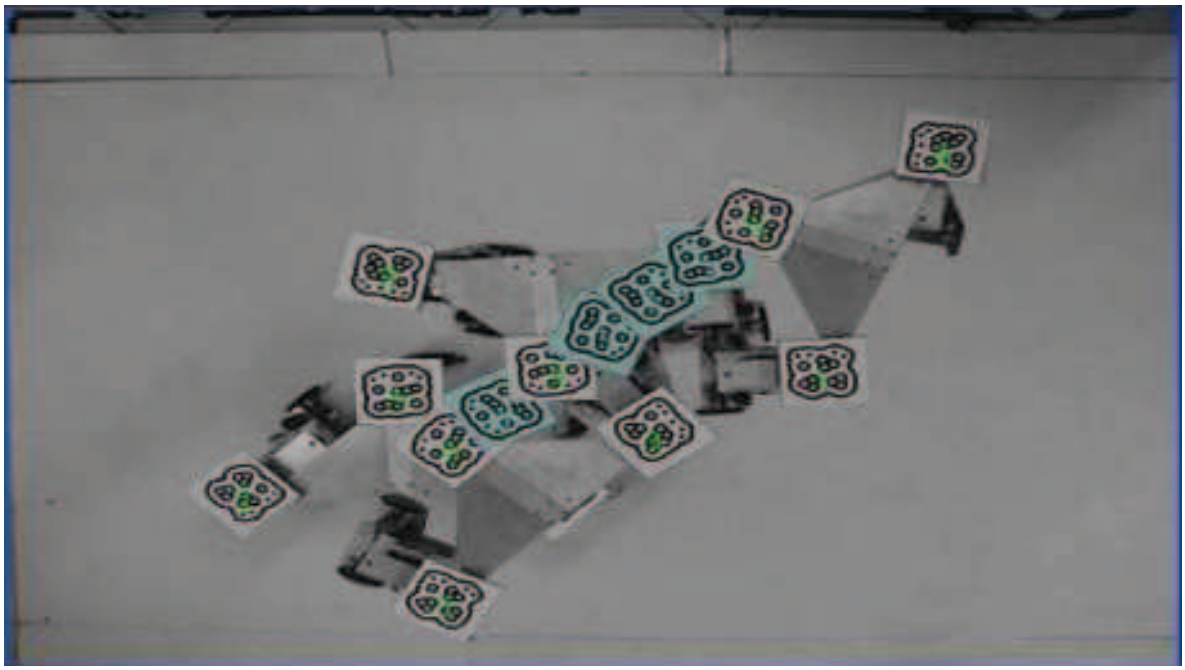


Figura VI – 32 Prueba de seguimiento de trayectoria rectilínea con giro.

La figura VI - 32 muestra la secuencia de avance del marcador fiducial que lleva la plataforma móvil en el punto de análisis, junto con algunas proyecciones del resto del robot para que se observe la rotación realizada. Los marcadores coloreados corresponden a aquellos cuyos cuerpos completos no aparecen en la imagen.

Si se observa con detenimiento el fiducial del punto de análisis, es notorio el giro que realiza la plataforma, dado que éste está girando. Por otro lado, el avance en diagonal propio de la trayectoria a seguir es claro.

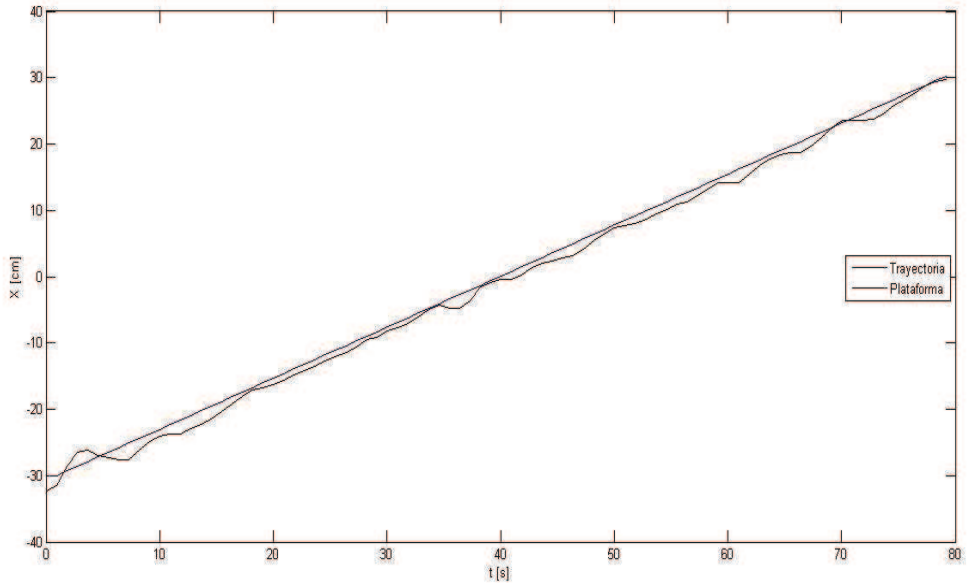


Figura VI – 33 Desplazamiento en el eje X del punto de análisis (a seguir y realizado).

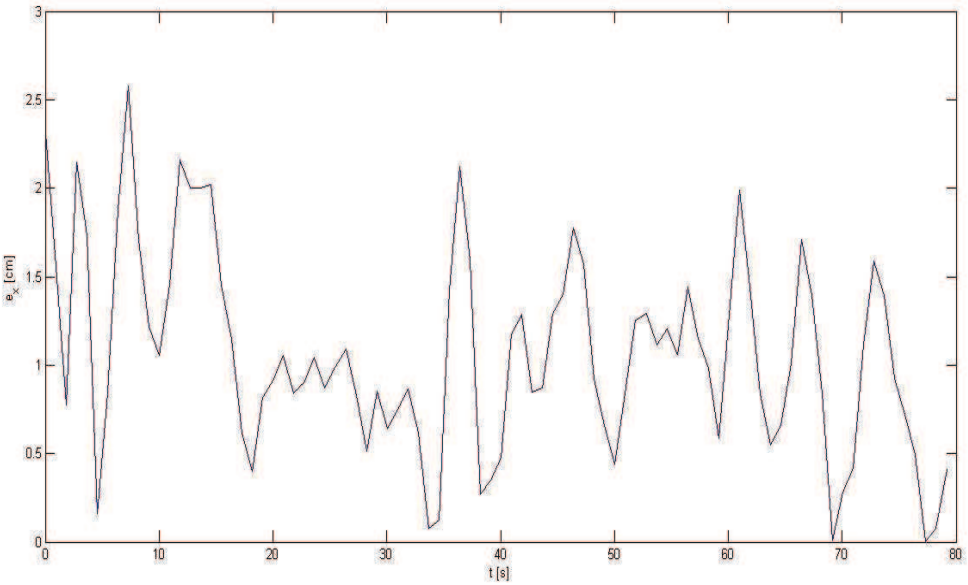


Figura VI – 34 Error de desplazamiento en el eje X.

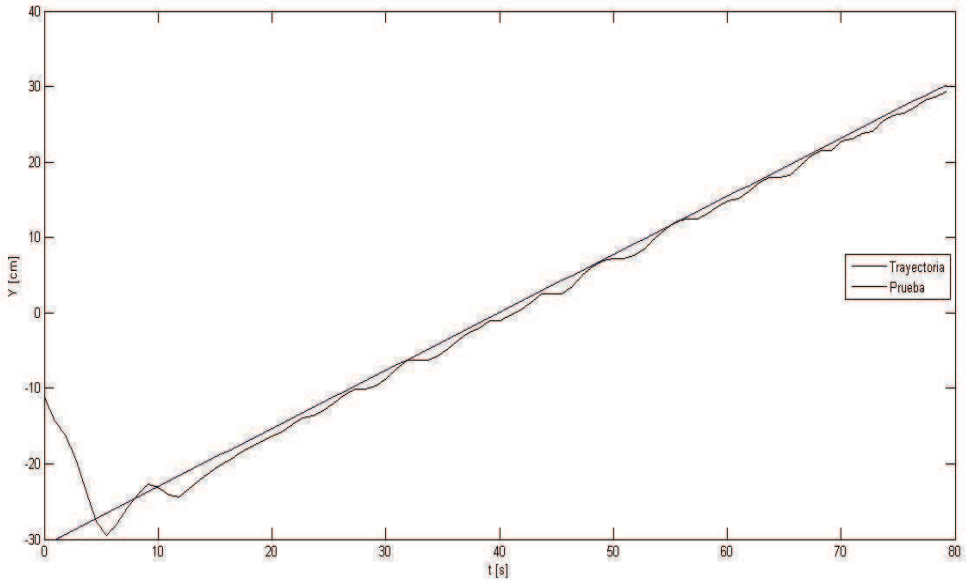


Figura VI – 35 Desplazamiento en el eje Y del punto de análisis (a alcanzar y realizado).

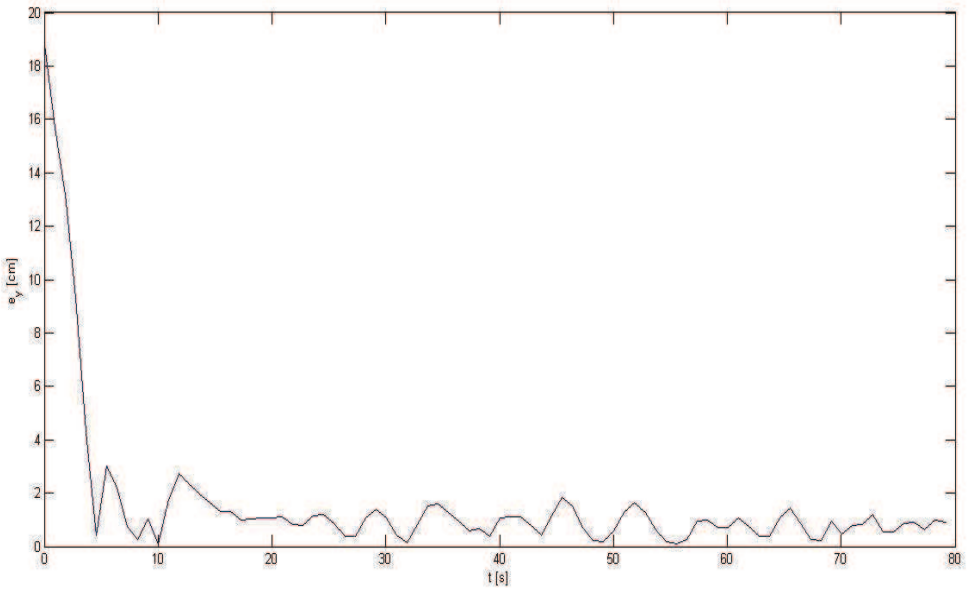


Figura VI – 36 Error de desplazamiento en el eje Y.

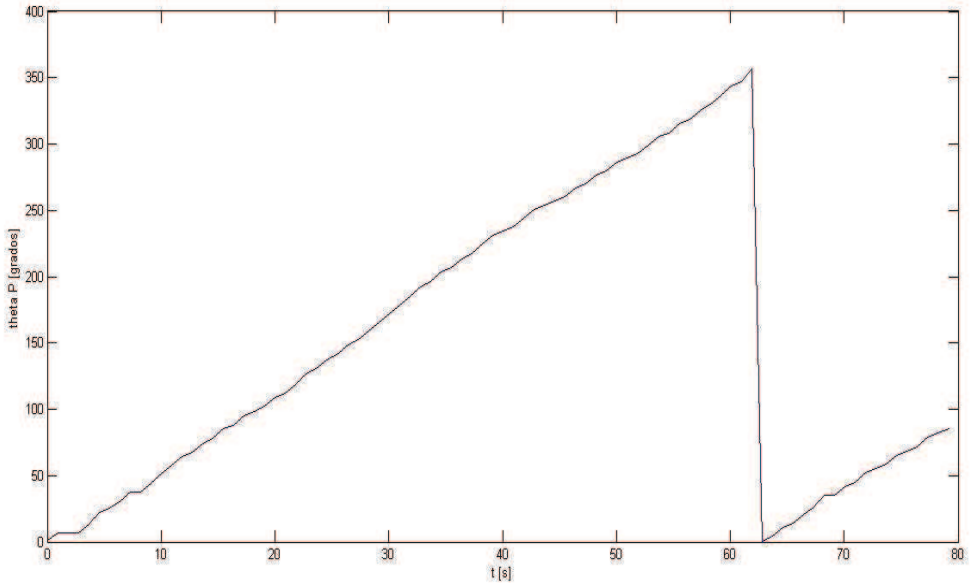


Figura VI – 37 Ángulo de rotación de la plataforma (realizado).

Una vez agregado el giro, el error de desplazamiento aumenta, aunque no de manera considerable. Por otro lado, la rotación también presenta ligeros problemas y no es tan uniforme. No obstante, sí se mantiene en aumento durante toda la prueba.

VI.5 Seguimiento de trayectoria circular del punto de análisis con orientación de la plataforma a cero grados

Se realiza lo mismo que en el seguimiento de trayectoria rectilínea, pero ahora con las siguientes ecuaciones para el avance de la trayectoria.

$$\begin{cases} x_t = 25\sin\left(\frac{\pi}{7}t\right) \\ y_t = 25\cos\left(\frac{\pi}{7}t\right) \end{cases}$$

(Ecuación 30)

VI.5.1 Simulación

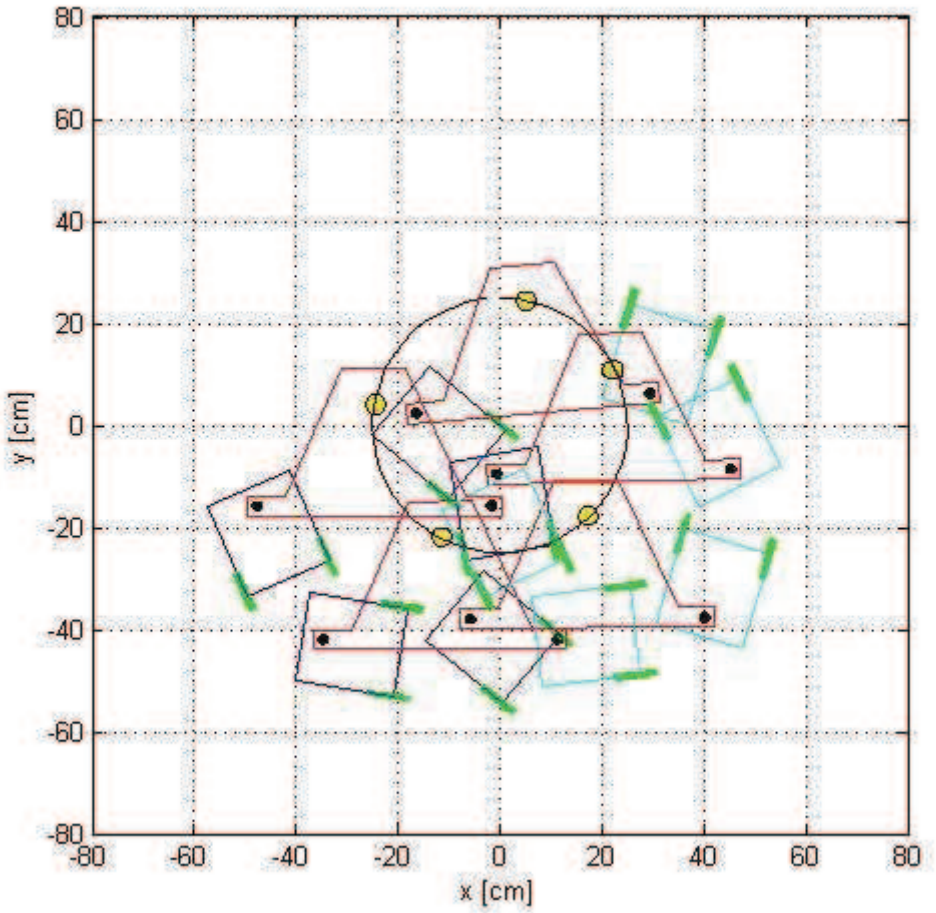


Figura VI - 38 Simulación de trayectoria circular del punto de análisis.

VI.5.2 Modelo funcional

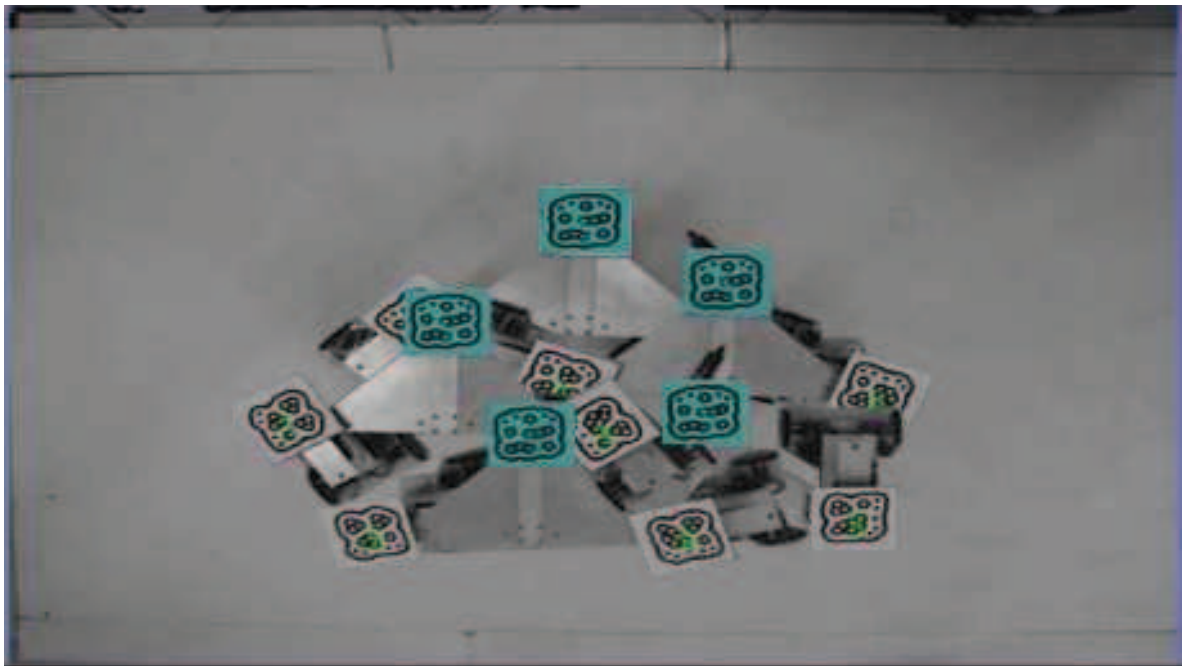


Figura VI - 39 Prueba de trayectoria circular del punto de análisis con orientación fija.

En las gráficas siguientes se observará que en la prueba la plataforma da dos vueltas a la circunferencia, aunque en la figura VI - 39 se muestre sólo una.

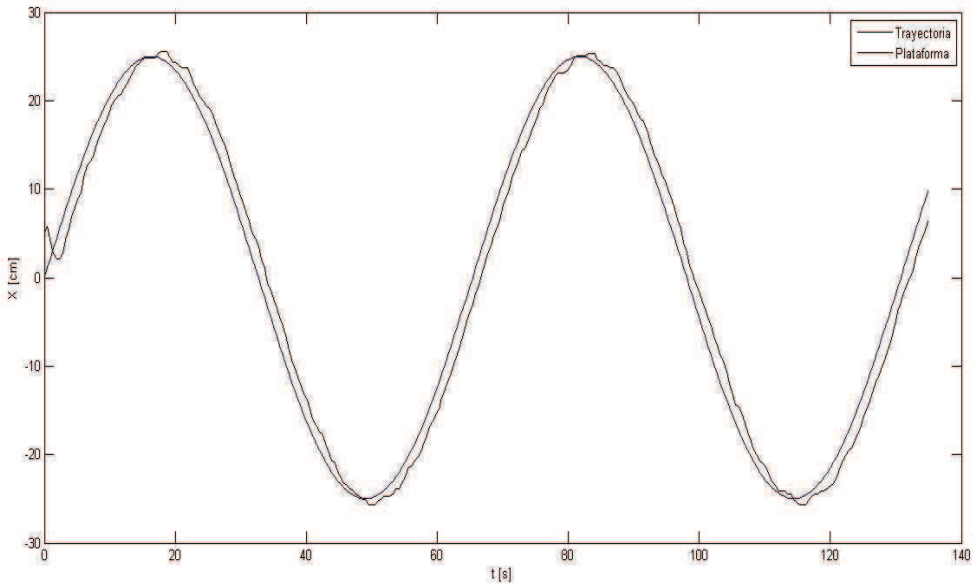


Figura VI - 40 Desplazamiento en el eje X del punto de análisis (a alcanzar y realizado).

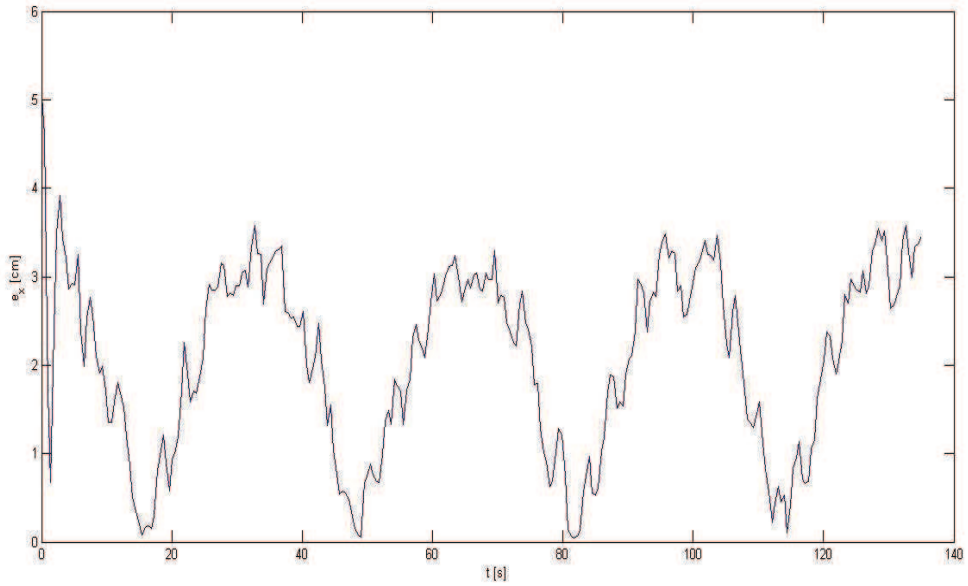


Figura VI - 41 Error de desplazamiento en el eje X.

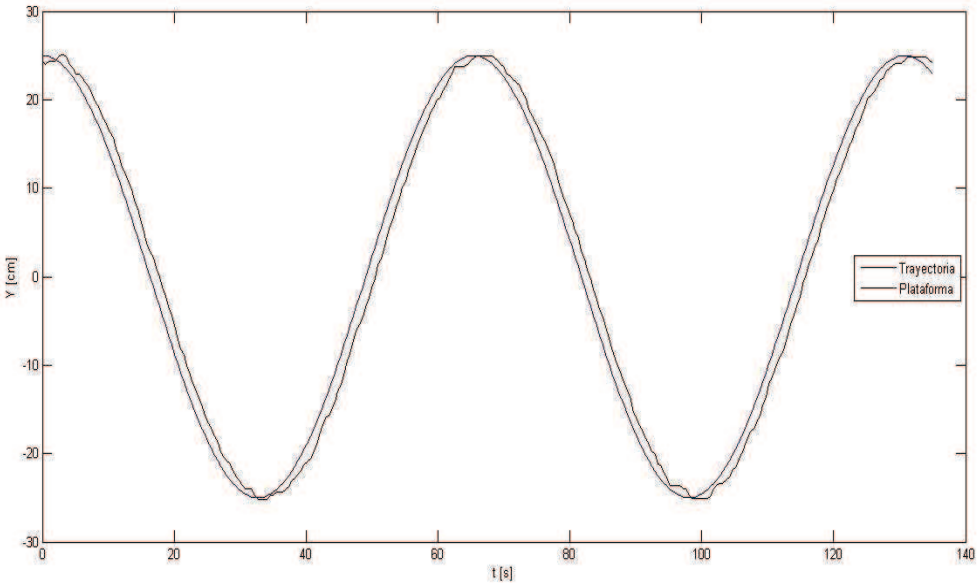


Figura VI - 42 Desplazamiento en el eje Y del punto de análisis (a alcanzar y realizado).

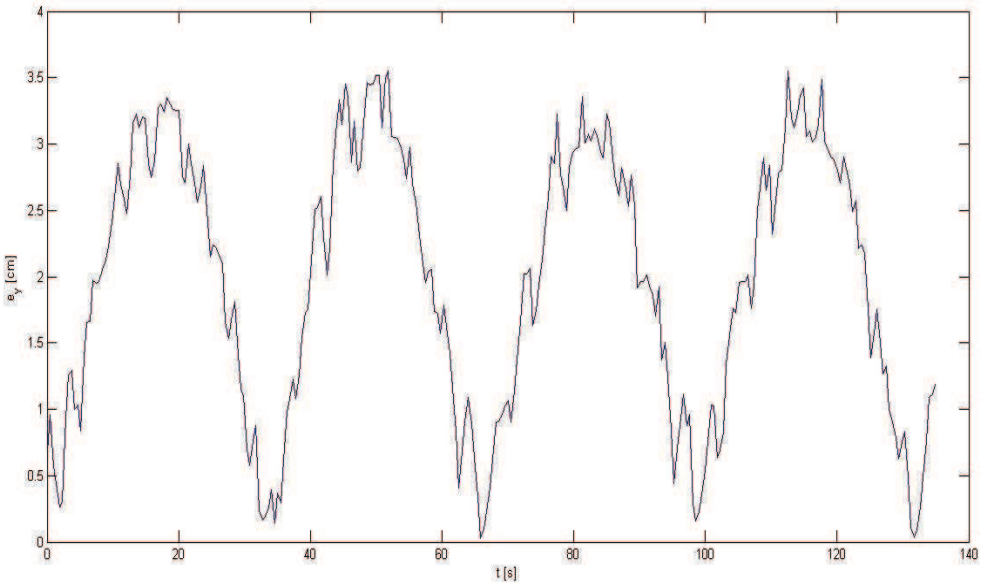


Figura VI - 43 Error de desplazamiento en el eje Y.

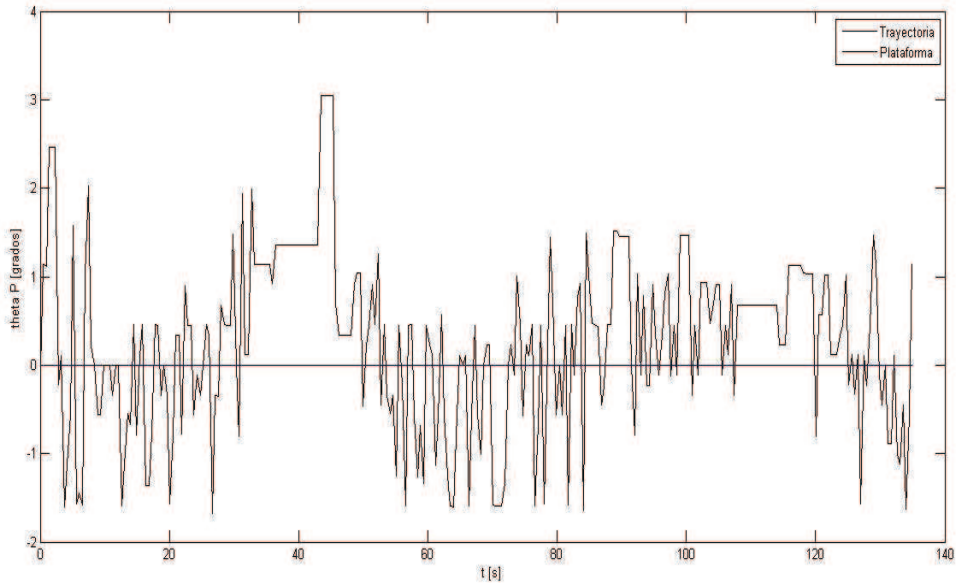


Figura VI - 44 Ángulo de rotación de la plataforma (a alcanzar y realizado).

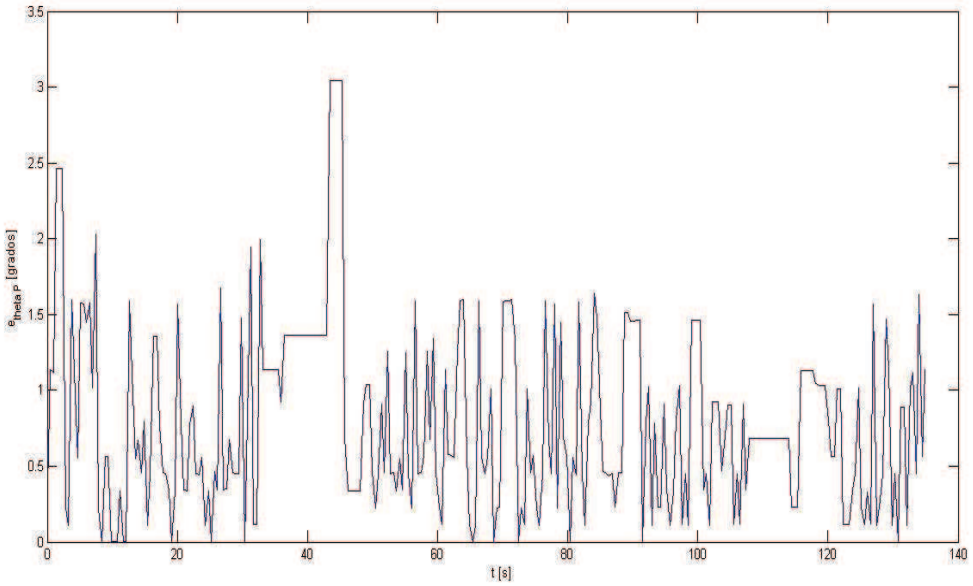


Figura VI - 45 Error de rotación.

Los errores de desplazamiento en los dos ejes muestra niveles máximos parecidos entre sí que ronda los 3.5 [cm]. El ángulo sigue oscilando de forma similar a las pruebas anteriores.

VI.6 Seguimiento de trayectoria circular del punto de análisis con giro de la plataforma

De igual forma que en la trayectoria rectilínea, se repite la prueba circular, pero con velocidad de rotación constante de $\omega = \frac{\pi}{30}\left[\frac{rad}{s}\right]$.

VI.6.1 Simulación

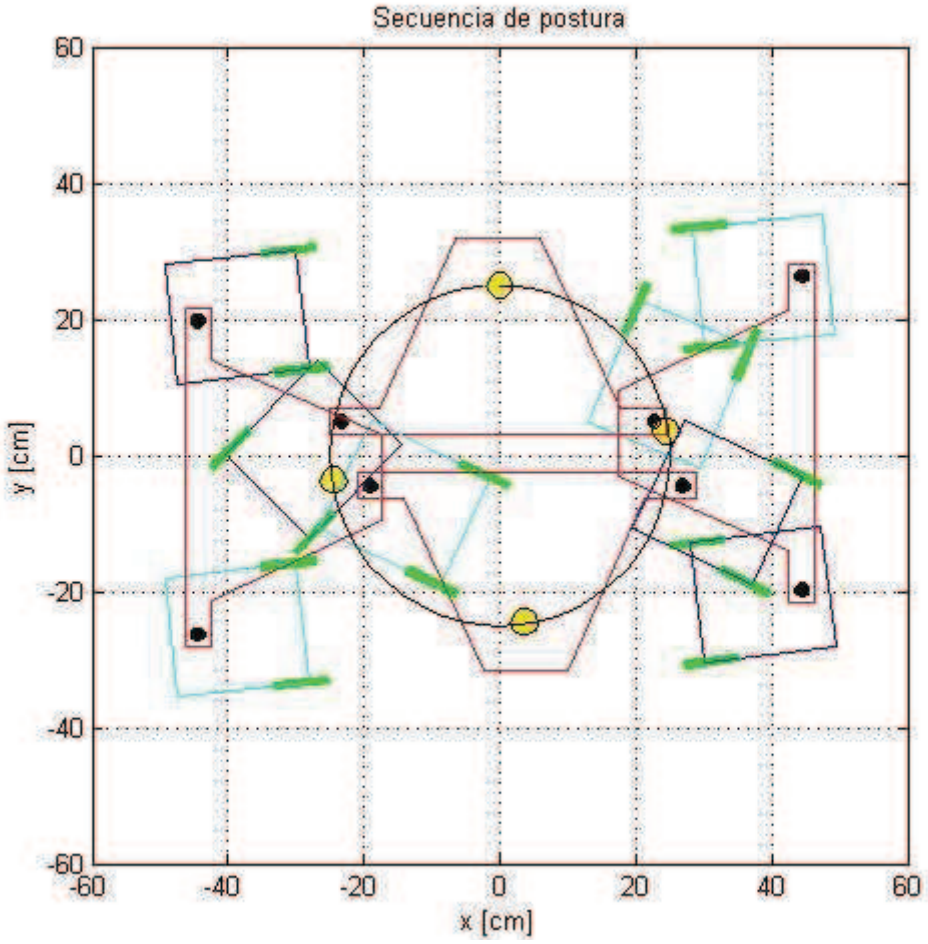


Figura VI - 46 Simulación de trayectoria circular del punto de análisis con rotación constante de la plataforma.

VI.6.2 Modelo funcional

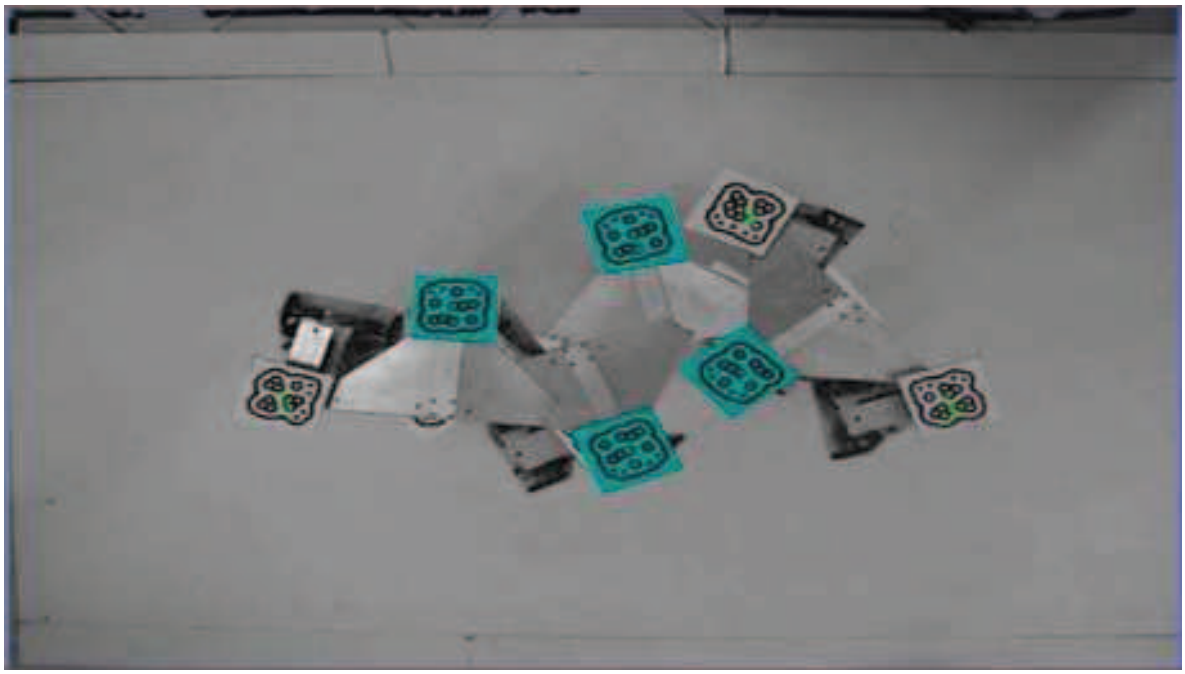


Figura VI - 47 Prueba de trayectoria circular con giro de la plataforma.

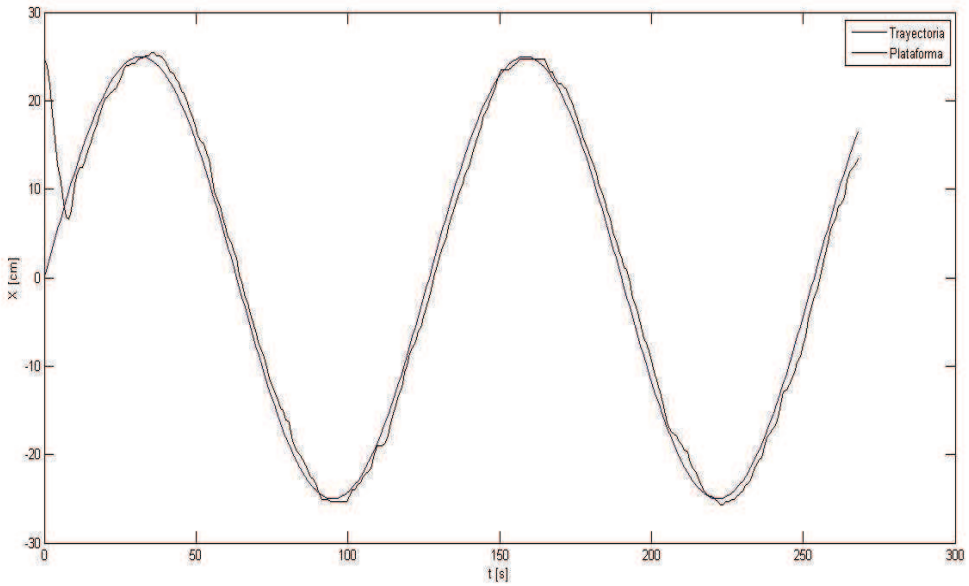


Figura VI - 48 Desplazamiento en el eje X del punto de análisis (a alcanzar y realizado).

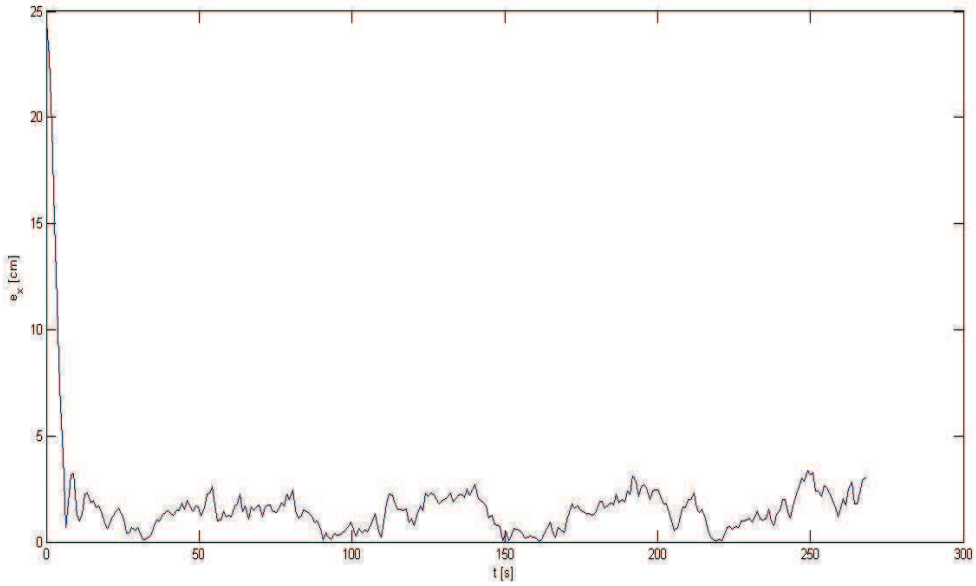


Figura VI - 49 Error de desplazamiento en el eje X.

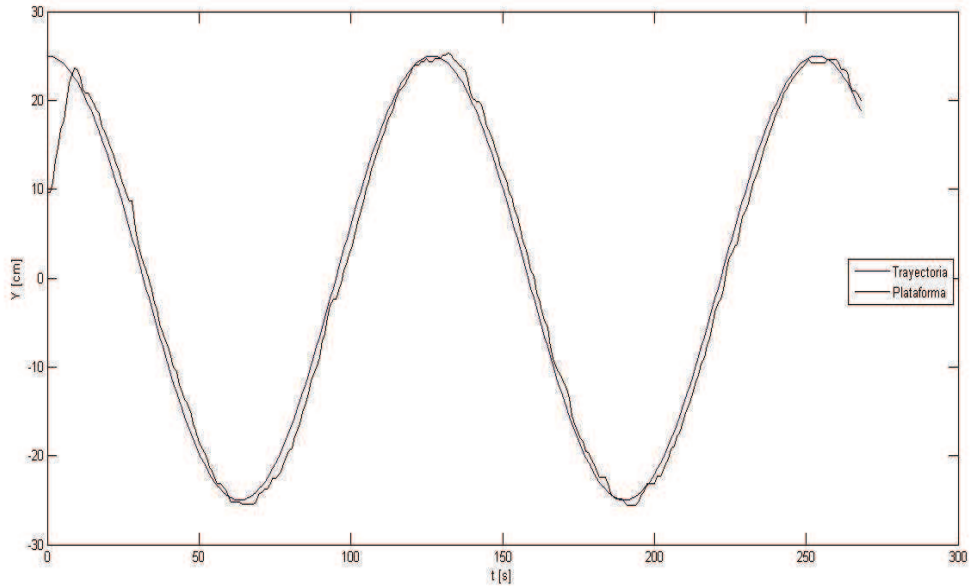


Figura VI - 50 Desplazamiento en el eje Y del punto de análisis (a alcanzar y realizado).

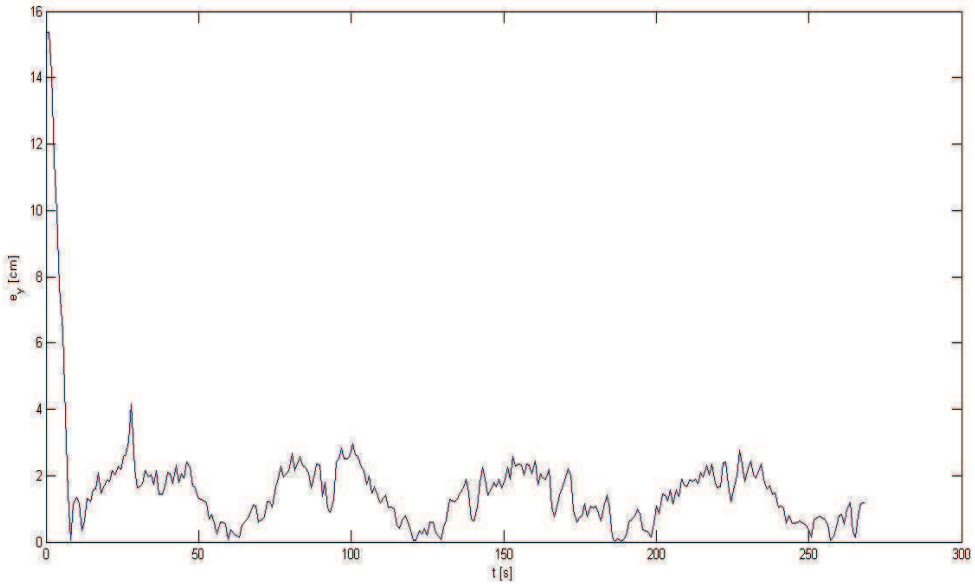


Figura VI - 51 Error de desplazamiento en el eje Y.

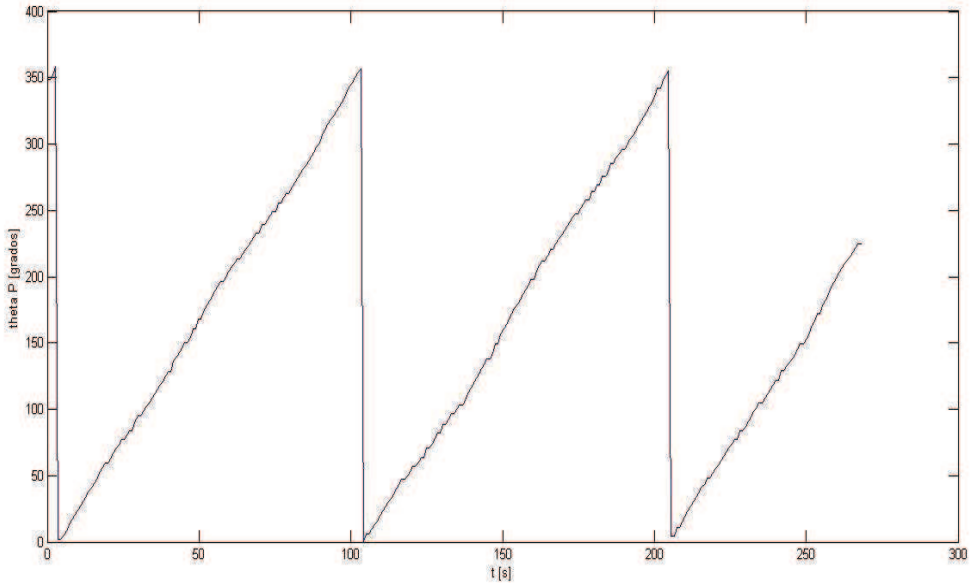


Figura VI - 52 Ángulo de rotación de la plataforma.

El desplazamiento lineal muestra un error mayor que cuando no existe giro constante. Esto coincide con lo sucedido en la prueba de seguimiento de trayectoria rectilínea.

VI.7 Alcance de objetivo con orientación de la plataforma

Para la prueba de alcance de objetivo, se coloca un marcador fiducial adicional que funciona como la meta a alcanzar. La plataforma se orienta para formar un ángulo de 90° entre el eje X_p y el vector d_g (capítulo IV) mediante la ecuación 28.

Se consideran las siguientes condiciones para las pruebas:

- $v_{xmax} = 15 \left[\frac{cm}{s}\right]$
- $v_{ymax} = 15 \left[\frac{cm}{s}\right]$
- $\omega_{max} = \frac{\pi}{10} \left[\frac{rad}{s}\right]$
- $k_r = 20\ [cm]$
- $k_{OWR} = 20\ [cm]$

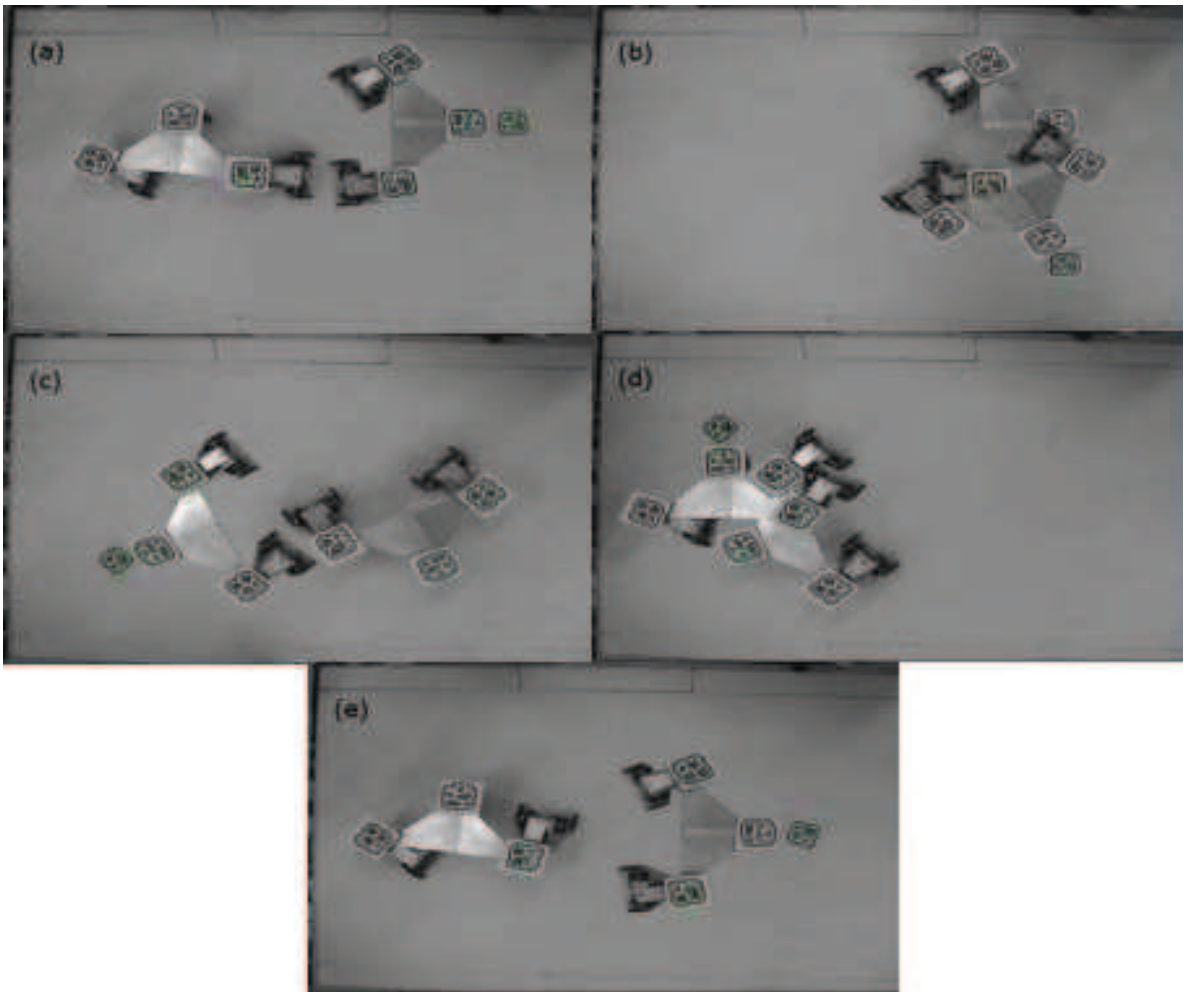


Figura VI - 53 Secuencia del alcance de objetivos.

La figura VI - 53 muestra la posición inicial y final de cada secuencia para el alcance de los objetivos. El recorrido realizado por el punto de análisis *P* al momento de la prueba se muestra en la siguiente figura.

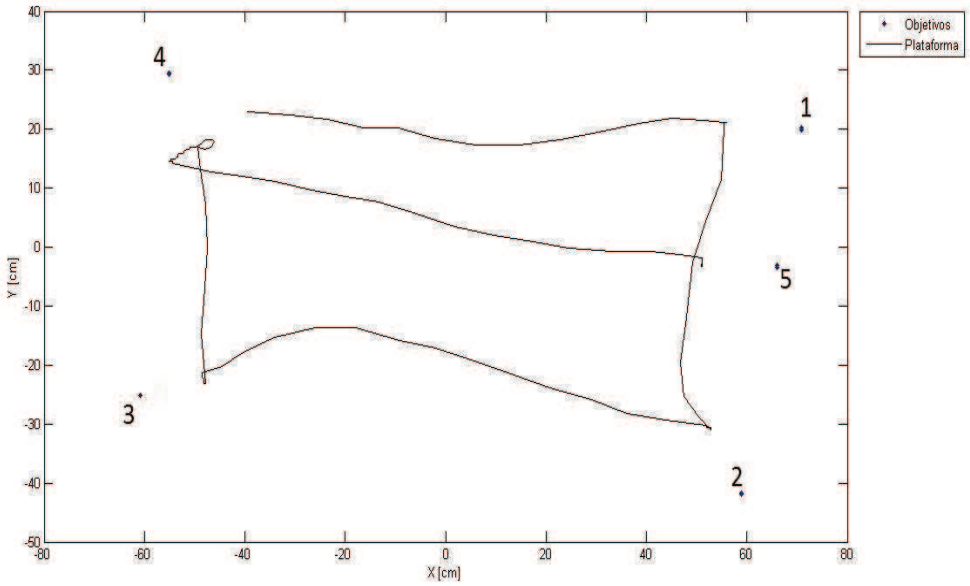


Figura VI - 54 Recorrido del punto de análisis para el alcance de objetivos.

En las cinco ocasiones, se puede observar que la plataforma se detiene dentro del radio de 20 [cm] que tiene de límite, por lo que cumple con lo señalado con la ley de control. También es importante recalcar que el punto de análisis no realizó movimientos totalmente rectos, aunque como se observó en las pruebas de seguimiento de trayectoria, el giro y avance simultáneos genera errores que puede esperarse que aumenten conforme la velocidad aumenta. En este caso las velocidades lineales máximas para ambos ejes fue del triple que en las pruebas anteriores, como también lo fue la velocidad angular máxima.

Para esta prueba, así como para la siguiente no fue necesario cerciorarse del funcionamiento de la plataforma mediante la simulación, debido a que la ley de control ya había sido probada satisfactoriamente en el seguimiento de trayectoria.

VI.8 Seguimiento de trayectoria dinámica con orientación de la plataforma

Esta prueba sigue el mismo principio que la anterior con la salvedad de que no se espera a que la plataforma alcance al objetivo y se estabilice, sino que se mantiene en movimiento constante y la plataforma lo sigue. Este es un

comportamiento común en el caso de que se requiera seguir a un líder que guía el camino o para persecución (de otro robot, otro ser vivo u otro objeto).

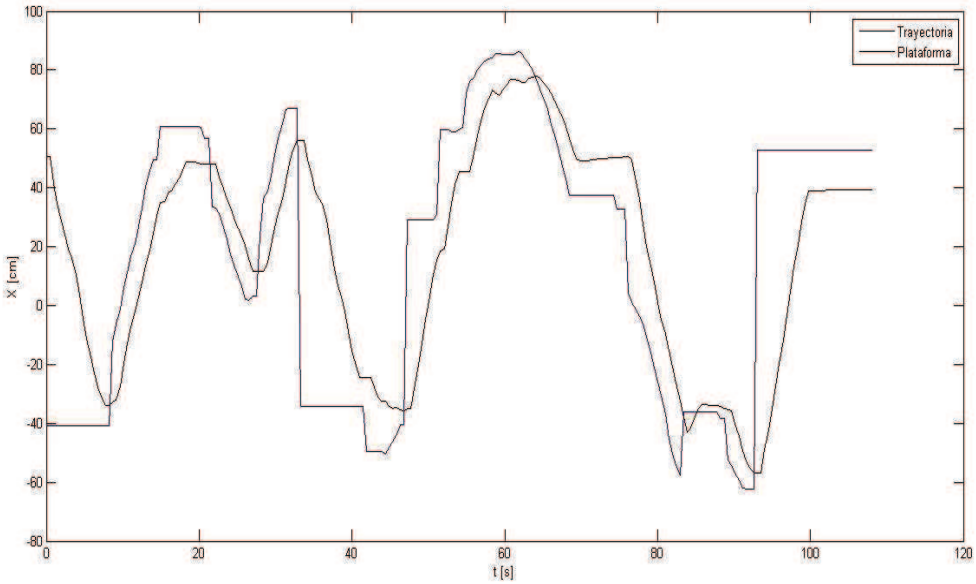


Figura VI - 55 Desplazamiento en el eje X (objetivo y plataforma).

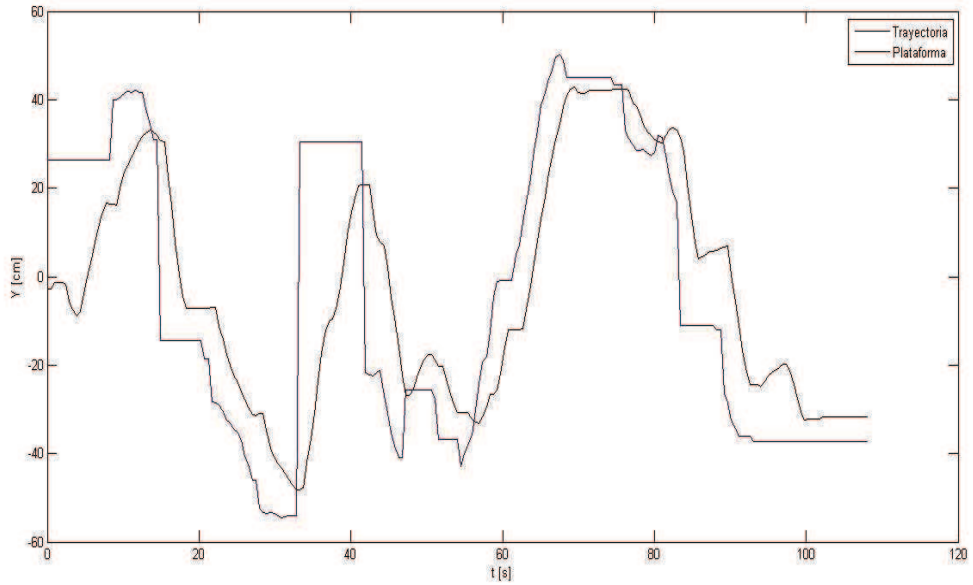


Figura VI - 56 Desplazamiento en el eje Y (objetivo y plataforma).

Se observa que el punto de análisis *P* cambia con un ligero desfase en tiempo con respecto al cambio realizado por la trayectoria. Aún así, se observan cambios de dirección bastante similares a los realizados por la trayectoria.

Con esto terminan las pruebas realizadas para la comprobación de las propiedades cinemáticas, la controlabilidad y la capacidad de estabilizarse del robot.

CAPÍTULO VII CONCLUSIONES Y TRABAJO A FUTURO

VII.1 Conclusiones

La teoría unificadora de González Villela, V. (2006) [6] funciona para el análisis de robots móviles no convencionales, como se pudo observar en los primeros tres capítulos. Esto dio lugar a que el análisis se desarrollara de forma rápida y sencilla. Sólo fue necesario definir algunos términos que no son contemplados, por tratarse de un robot móvil no convencional.

Por otro lado, la configuración propuesta sí es omnidireccional y se comprobó tanto de forma teórica como práctica. Fue posible manipular los tres grados de libertad permisibles en el plano de manera independiente y está reflejado en movimientos de traslación independientes de los de rotación en varias de las pruebas.

También es importante recalcar que la plataforma sí es controlable y que la ley de campos potenciales sí es aplicable desde el punto de vista cinemático, como se observó en las pruebas de lazo cerrado. En el seguimiento de trayectoria se observó que la aplicación de campos potenciales es viable siempre y cuando se tenga un espacio que permita desviaciones de unos cuantos centímetros. También es importante recalcar que conforme la velocidad aumenta, el control tarda más en

corregir, como se observó en las pruebas de alcance de objetivo y trayectoria dinámica.

En cuanto a lo que se ve reflejado en los resultados de las pruebas, hay varios detalles que es importante hacer notar. Primero, en lo que al modelo funcional se refiere, hubo principalmente problemas mecánicos que propiciaban vibraciones al momento de avanzar o “cascabeleo” de ciertos componentes, aunque no representó ningún contratiempo considerable más allá de provocar oscilaciones en el ángulo de orientación. La electrónica montada funcionó correctamente, de acuerdo a lo demandado y las baterías permitieron varias horas de pruebas antes de requerir una recarga, lo que refleja un bajo consumo de energía.

El sistema de visión cumple su función, aunque se presentaban errores más grandes cuando se trabajaba en los límites del campo de visión, lo que refleja que la curvatura del lente también puede ser un factor importante para la generación de errores. La resolución de la cámara y la velocidad de captura también son factores importantes que influyen principalmente en la velocidad de desplazamiento y giro del robot, así como en el tiempo de reacción ante el cambio de objetivo.

Por otro lado, el software utilizado no cumplió en su totalidad lo solicitado al momento de realizar las pruebas. En lo que a la simulación se refiere, Simulink es una herramienta adecuada, que muestra fielmente lo que debe suceder. No obstante, el hecho de que disminuya la velocidad de procesamiento drásticamente cuando se involucra la comunicación en tiempo real, da lugar a que no sea del todo confiable al momento de controlar la plataforma.

VII.2. Trabajo a futuro

En este proyecto se demostró el principio de funcionamiento de la configuración de plataforma móvil omnidireccional utilizada, desde el punto de vista cinemático. Esto da lugar a varias opciones en trabajos posteriores. En primer lugar, darle una aplicación concreta al principio de funcionamiento descrito en este trabajo. También, realizar el análisis dinámico y observar el desempeño, así como desarrollar un diseño mecánico a conciencia que se adecue a las necesidades de la aplicación.

También es conveniente buscar otras formas de ubicación de la plataforma. Si se carece de un espacio de trabajo delimitado, se requeriría que el robot por sí solo fuera capaz de ubicarse en su entorno y encontrar el camino que más le convenga o contar con un sistema de visión montado en la plataforma para su manipulación manual de forma remota.

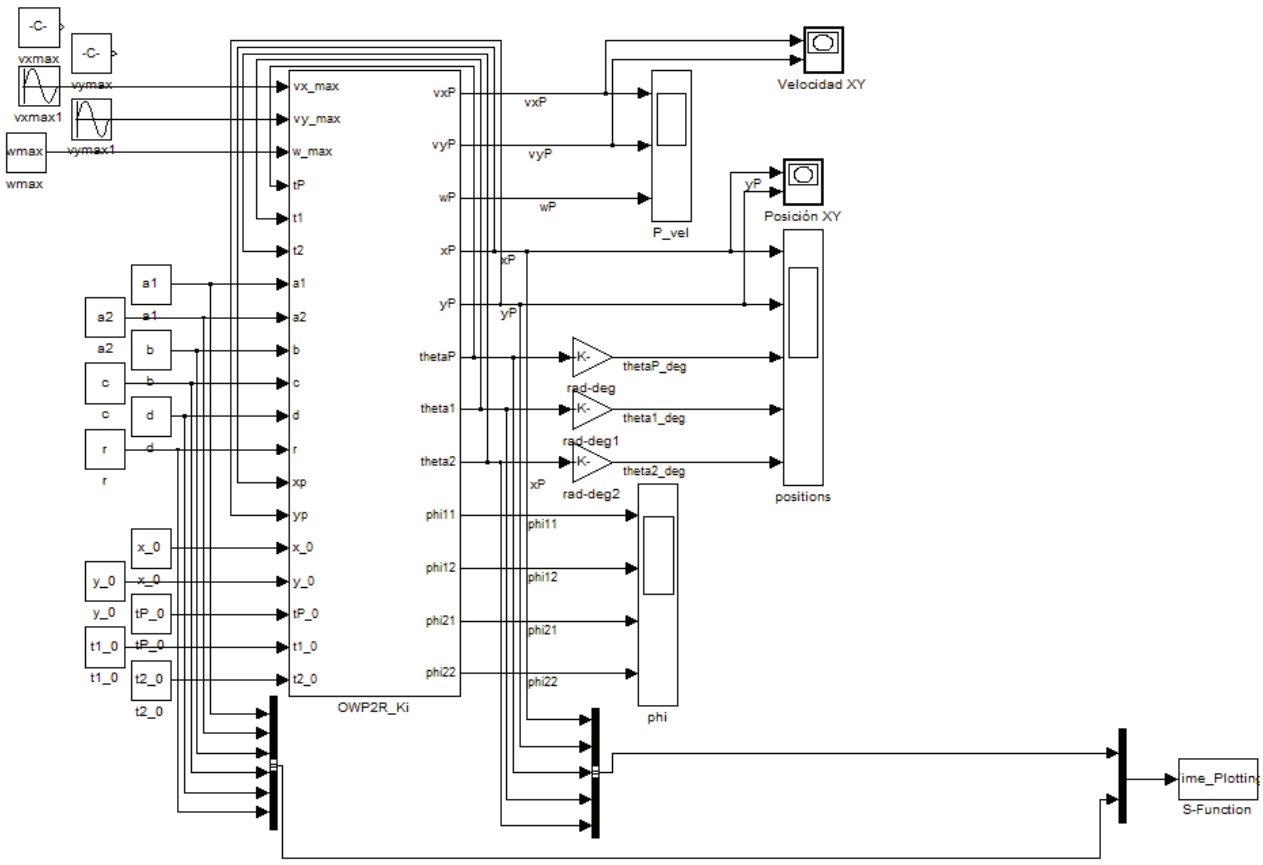
Por otro lado, sería conveniente probar otras leyes de control o combinarlas para encontrar la que optimice el desempeño, tanto en seguimiento de una referencia, como en velocidad de reacción y movilidad.

APÉNDICES

Apéndice I Diagramas de bloque y programación de las simulaciones

Lazo abierto

Diagrama de bloque



*Los bloques desconectados de valores constantes se alternan con los de funciones sinusoidales para los distintos tipos de prueba.

Programación de la función S para el cálculo de la cinemática

```
function [sys,x0,str,ts] = kinematic_model_path(t,x,u,flag)
```

```
% Programa creado por Arturo Martínez Carrillo,
% alumno de la Facultad de Ingeniería, de la Universidad Nacional Autónoma de México,
% para el proyecto de tesis "Plataforma móvil omnidireccional a partir de dos robots móviles
diferenciales"
%
% Rutina para el cálculo de la cinemática de una plataforma móvil omnidireccional, donde los
estados de entrada fueron seleccionados como:
%
%     1) La velocidad lineal del punto "P" sobre el eje "Xp".
%     2) La velocidad lineal del punto "P" sobre el eje "Yp".
%     3) La velocidad angular de la plataforma móvil.
%
% Las entradas están definidas de la siguiente manera:
%
% u(1) = vx_max   Velocidad lineal máxima en el eje "Xp"
% u(2) = vy_max   Velocidad lineal máxima en el eje "Yp"
% u(3) = w_max    Velocidad angular máxima de la plataforma móvil
% u(4) = tP       Ángulo de orientación de la plataforma con respecto al sistema {X,Y}
% u(5) = t1       Ángulo de orientación del robot de tracción 1 con respecto al sistema
{Xp,Yp}
% u(6) = t2       Ángulo de orientación del robot de tracción 2 con respecto al sistema
{Xp,Yp}
% u(7) = a1       Distancia del punto "P" al punto "P1" sobre el eje "Xp"
% u(8) = a2       Distancia del punto "P" al punto "P2" sobre el eje "Xp"
% u(9) = b        Distancia del punto "P" a los puntos "P1" y "P2" sobre el eje "Yp"
% u(10) = c       Distancia del punto "P1" a los puntos "P1,1" y "P1,2" sobre el eje "X1" y
del punto "P2" a los puntos "P2,1" y "P2,2" sobre el eje "X2"
% u(11) = d       Distancia del punto "P1" a los puntos "P1,1" y "P1,2" sobre el eje "Y1" y
del punto "P2" a los puntos "P2,1" y "P2,2" sobre el eje "Y2"
% u(12) = r       Radio de las ruedas
% u(13) = xp      Posición del punto "P" sobre el eje "X"
% u(14) = yp      Posición del punto "P" sobre el eje "Y"
%
% Las salidas están definidas de la siguiente manera:
%
% o(1) = vxP      velocidad lineal del punto "P" sobre el eje "X"
% o(2) = vyP      velocidad lineal del punto "P" sobre el eje "Y"
% o(3) = wP       velocidad angular de la plataforma móvil
% o(4) = w1       velocidad angular del robot de tracción 1
% o(5) = w2       velocidad angular del robot de tracción 2
% o(6) = phi11    velocidad angular de la rueda 1,1
% o(7) = phi12    velocidad angular de la rueda 1,2
% o(8) = phi21    velocidad angular de la rueda 2,1
% o(9) = phi22    velocidad angular de la rueda 2,2

switch flag,

  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  case 1,
    sys=mdlDerivatives(t,x,u);

  case 2,
    sys=mdlUpdate(t,x,u);

  case 3,
    sys=mdlOutputs(t,x,u);

  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  case 9,
    sys=mdlTerminate(t,x,u);

  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
```

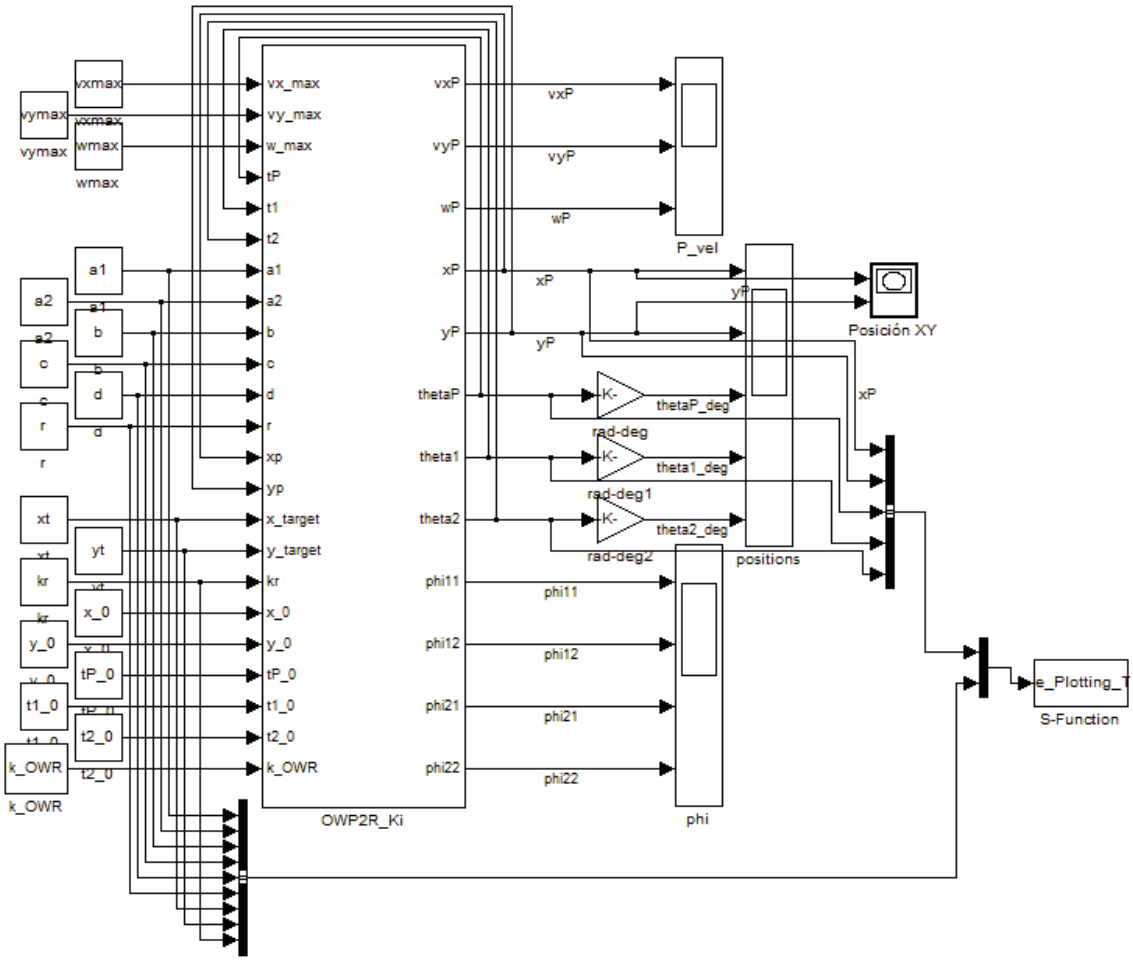
```
end

% end sfuntmpl

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts,T_samp]=mdlInitializeSizes

%
sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 9;
sizes.NumInputs      = 14;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

x0  = [];

str = [];

ts  = [0 0];

% end mdlInitializeSizes

%
%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u)

sys = [];

% end mdlDerivatives

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
function sys=mdlUpdate(t,x,u)

sys = [ ];

% end mdlUpdate

%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%
function sys=mdlOutputs(t,x,u)


% Variables de velocidad de entrada
```

```
    U = [u(1); u(2); u(3)];


% Variables de retroalimentación angular

    tP = u(4);
    t1 = u(5);
    t2 = u(6);


% Dimensiones de la plataforma

    global a1 a2 b c d r;

    a1 = u(7);
    a2 = u(8);
    b = u(9);
    c = u(10);
    d = u(11);
    r = u(12);

% Cálculo de las velocidades generalizadas

    Sq=[cos(tP),-sin(tP),0;
        sin(tP),cos(tP),0;
        0,0,1;
        -sin(t1)/c,cos(t1)/c,a1*cos(t1)/c+b*sin(t1)/c;
        -sin(t2)/c,cos(t2)/c,a1*cos(t2)/c+b*sin(t2)/c;
        (c*cos(t1)+d*sin(t1))/(c*r),-d*cos(t1)/(c*r)+sin(t1)/r,-b*cos(t1)/r-
a1*d*cos(t1)/(c*r)+a1*sin(t1)/r-b*d*sin(t1)/(c*r);
        (c*cos(t1)-d*sin(t1))/(c*r),d*cos(t1)/(c*r)+sin(t1)/r,-
b*cos(t1)/r+a1*d*cos(t1)/(c*r)+a1*sin(t1)/r+b*d*sin(t1)/(c*r);
        (c*cos(t2)+d*sin(t2))/(c*r),-d*cos(t2)/(c*r)+sin(t2)/r,-b*cos(t2)/r-
a2*d*cos(t2)/(c*r)+a2*sin(t2)/r-b*d*sin(t2)/(c*r);
        (c*cos(t2)-d*sin(t2))/(c*r),d*cos(t2)/(c*r)+sin(t2)/r,-
b*cos(t2)/r+a2*d*cos(t2)/(c*r)+a2*sin(t2)/r+b*d*sin(t2)/(c*r)];

    X= Sq*U;

    sys = [X];


% end mdlOutputs

%
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1;    %  Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
%
```

```
function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

Lazo cerrado

Diagrama de bloque



Programación de la función S para el cálculo de la cinemática

```
function [sys,x0,str,ts] = kinematic_model_target(t,x,u,flag)

% Programa creado por Arturo Martínez Carrillo,
% alumno de la Facultad de Ingeniería, de la Universidad Nacional Autónoma de México,
% para el proyecto de tesis "Plataforma móvil omnidireccional a partir de dos robots móviles
diferenciales"
%
% Rutina para el cálculo de la cinemática de una plataforma móvil omnidireccional que
alcanza un objetivo,
% donde los estados de entrada fueron seleccionados como:
%
%     1) La velocidad lineal del punto "P" sobre el eje "Xp".
%     2) La velocidad lineal del punto "P" sobre el eje "Yp".
%     3) La velocidad angular de la plataforma móvil.
%
% Las entradas están definidas de la siguiente manera:
%
% u(1) = vx_max       Velocidad lineal máxima en el eje "Xp"
% u(2) = vy_max       Velocidad lineal máxima en el eje "Yp"
% u(3) = w_max        Velocidad angular máxima de la plataforma móvil
% u(4) = tP           Ángulo de orientación de la plataforma con respecto al sistema {X,Y}
% u(5) = t1           Ángulo de orientación del robot de tracción 1 con respecto al sistema
{Xp,Yp}
% u(6) = t2           Ángulo de orientación del robot de tracción 2 con respecto al sistema
{Xp,Yp}
% u(7) = a1           Distancia del punto "P" al punto "P1" sobre el eje "Xp"
% u(8) = a2           Distancia del punto "P" al punto "P2" sobre el eje "Xp"
% u(9) = b            Distancia del punto "P" a los puntos "P1" y "P2" sobre el eje "Yp"
% u(10) = c           Distancia del punto "P1" a los puntos "P1,1" y "P1,2" sobre el eje
"X1" y del punto "P2" a los puntos "P2,1" y "P2,2" sobre el eje "X2"
% u(11) = d           Distancia del punto "P1" a los puntos "P1,1" y "P1,2" sobre el eje
"Y1" y del punto "P2" a los puntos "P2,1" y "P2,2" sobre el eje "Y2"
% u(12) = r           Radio de las ruedas
% u(13) = xp          Posición del punto "P" sobre el eje "X"
% u(14) = yp          Posición del punto "P" sobre el eje "Y"
% u(15) = x_target    Posición del objetivo sobre el eje "X"
% u(16) = y_target    Posición del objetivo sobre el eje "Y"
% u(17) = kr          Radio del área de aproximación al objetivo
%
% Las salidas están definidas de la siguiente manera:
%
% o(1) = vxP      velocidad lineal del punto "P" sobre el eje "X"
% o(2) = vyP      velocidad lineal del punto "P" sobre el eje "Y"
% o(3) = wP       velocidad angular de la plataforma móvil
% o(4) = w1       velocidad angular del robot de tracción 1
% o(5) = w2       velocidad angular del robot de tracción 2
% o(6) = phi11    velocidad angular de la rueda 1,1
% o(7) = phi12    velocidad angular de la rueda 1,2
% o(8) = phi21    velocidad angular de la rueda 2,1
% o(9) = phi22    velocidad angular de la rueda 2,2

switch flag,

  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  case 1,
    sys=mdlDerivatives(t,x,u);

  case 2,
    sys=mdlUpdate(t,x,u);

  case 3,
    sys=mdlOutputs(t,x,u);

  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
```

```
  case 9,
    sys=mdlTerminate(t,x,u);

  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts,T_samp]=mdlInitializeSizes

%
sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 9;
sizes.NumInputs      = 18;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

x0  = [];

str = [];

ts  = [0 0];

% end mdlInitializeSizes

%
%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u)

sys = [];

% end mdlDerivatives

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
function sys=mdlUpdate(t,x,u)

sys = [ ];

% end mdlUpdate

%
%=============================================================================
% mdlOutputs
```

```
% Return the block outputs.
%==============================================================================
%
function sys=mdlOutputs(t,x,u)


% Variables de velocidad de entrada

    vxmax = u(1);
    vymax = u(2);
    wmax = u(3);


% Variables de retroalimentación angular

    tP = u(4);
    t1 = u(5);
    t2 = u(6);


% Dimensiones de la plataforma

    global a1 a2 b c d r;

    a1 = u(7);
    a2 = u(8);
    b = u(9);
    c = u(10);
    d = u(11);
    r = u(12);

% Coordenadas del robot y del objetivo

    global xp yp xt yt;

    xp = u(13);
    yp = u(14);
    xt = u(15);
    yt = u(16);
    kr = u(17);
    k_OWR = u(18);


% Cálculo de las velocidades de entrada

    dg = sqrt((xt-xp)^2 + (yt-yp)^2);

    if yt==yp && xt==xp
        tg = tP;
    else
        tg = atan2((yt-yp),(xt-xp));
    end

    te = tg-tP;

    if dg <= k_OWR
        vxpp = 0;
        vypp = 0;
    elseif dg <= k_OWR+kr
        vxpp = vxmax*dg*cos(te)/(kr+k_OWR);
        vypp = vymax*dg*sin(te)/(kr+k_OWR);
    else
        vxpp = vxmax*cos(te);
        vypp = vymax*sin(te);
    end
```

```
%     wpp = wmax*sin(te-(pi/2));    %Velocidad angular para orientación de 90° apuntando
hacia el objetivo
%     wpp = wmax*sin(-tP);          %Velocidad angular para orientación a 0°
    wpp = wmax;                     %Velocidad angular constante

    U = [vxpp; vypp; wpp];


% Cálculo de las velocidades generalizadas

    Sq=[cos(tP),-sin(tP),0;
        sin(tP),cos(tP),0;
        0,0,1;
        -sin(t1)/c,cos(t1)/c,a1*cos(t1)/c+b*sin(t1)/c;
        -sin(t2)/c,cos(t2)/c,a1*cos(t2)/c+b*sin(t2)/c;
        (c*cos(t1)+d*sin(t1))/(c*r),-d*cos(t1)/(c*r)+sin(t1)/r,-b*cos(t1)/r-
a1*d*cos(t1)/(c*r)+a1*sin(t1)/r-b*d*sin(t1)/(c*r);
        (c*cos(t1)-d*sin(t1))/(c*r),d*cos(t1)/(c*r)+sin(t1)/r,-
b*cos(t1)/r+a1*d*cos(t1)/(c*r)+a1*sin(t1)/r+b*d*sin(t1)/(c*r);
        (c*cos(t2)+d*sin(t2))/(c*r),-d*cos(t2)/(c*r)+sin(t2)/r,-b*cos(t2)/r-
a2*d*cos(t2)/(c*r)+a2*sin(t2)/r-b*d*sin(t2)/(c*r);
        (c*cos(t2)-d*sin(t2))/(c*r),d*cos(t2)/(c*r)+sin(t2)/r,-
b*cos(t2)/r+a2*d*cos(t2)/(c*r)+a2*sin(t2)/r+b*d*sin(t2)/(c*r)];

    X= Sq*U;

    sys = [X];


% end mdlOutputs

%
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1;    %  Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
%
function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

Programación de la función S para graficar la simulación

```
function [sys,x0,str,ts] = Real_Time_Plotting_Path(t,x,u,flag, T_sim, Axis_Width,ON)

% Esta rutina grafica la posición de una plataforma móvil omnidireccional
%
switch flag,


  % Initialization %
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(T_sim, Axis_Width);

  % Derivatives %
  case 1,
    sys=mdlDerivatives(t,x,u);

  % Update %
  case 2,
    sys=mdlUpdate(t,x,u, Axis_Width,ON);

  % Outputs %
  case 3,
    sys=mdlOutputs(t,x,u);

  % GetTimeOfNextVarHit %
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % Terminate %
  case 9,
    sys=mdlTerminate(t,x,u);

  % Unexpected flags %
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end


%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes(T_sim, Axis_Width)

%

sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 0;
sizes.NumInputs      = 11;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

%
% initialize the initial conditions
%
x0  = [];

%
% str is always an empty matrix
%
```

```
str = [];

%
% initialize the array of sample times
%
ts  = [T_sim 0];


% Inicialización de la figura

% the handles of the disk and index mark are stored in the block's UserData.

if (isempty(findobj('UserData',gcb)));
    h_fig = figure('Position', [450 100 1000 1000], ...
                   'MenuBar', 'figure','NumberTitle','off', ...
                   'Resize', 'off', ...
                   'Name','POSTURE: Omnidirectional Mobile Platform');

    set(h_fig,'UserData',gcb) ;
    set(h_fig,'DoubleBuffer','on');
end

% end mdlInitializeSizes

%
%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
%

function sys=mdlDerivatives(t,x,u)

sys = [];

% end mdlDerivatives

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%

function sys=mdlUpdate(t,x,u, Axis_Width, ON)

sys = [];


% Dibujo de las figuras


if (ON==0)|(t==0)
    hold off;
end

figure(1)
set(1,'Position', [490 150 525 500]);

% Fijación de ejes

    axis([-Axis_Width Axis_Width -Axis_Width Axis_Width]);
    grid on;

    xlabel('x [cm]')
```

```
    ylabel('y [cm]')
    title('Secuencia de postura')


% Entradas

    xP      = u(1);
    yP      = u(2);
    thetaP  = u(3);
    theta1  = u(4);
    theta2  = u(5);
    a1      = u(6);
    a2      = u(7);
    b       = u(8);
    c       = u(9);
    d       = u(10);
    r       = u(11);


% Dimensiones extra de la plataforma

    extraP_x1=1.9;
    extraP_x2=5;
    extraP_x3=16.75;
    extraP_x4=5.6;
    extraP_y1=1.9;
    extraP_y2=28.9;

% Dimensiones extra de las ruedas

    wheel_y         = 1;

% Cálculos útiles

    cos_thP = cos(thetaP);
    sin_thP = sin(thetaP);

    cos_th1 = cos(theta1);
    sin_th1 = sin(theta1);

    cos_th2 = cos(theta2);
    sin_th2 = sin(theta2);


% Realización del robot de tracción 1

    wmr1_contour_x = [-(c+1.1); -(c+1.1); 3.9; 3.9];
    wmr1_contour_y = [-d; d; d; -d];

    wmr1_pos_x = xP + cos_thP*(a1 + cos_th1*wmr1_contour_x - sin_th1*wmr1_contour_y) -
sin_thP*(b + sin_th1*wmr1_contour_x + cos_th1*wmr1_contour_y);
    wmr1_pos_y = yP + sin_thP*(a1 + cos_th1*wmr1_contour_x - sin_th1*wmr1_contour_y) +
cos_thP*(b + sin_th1*wmr1_contour_x + cos_th1*wmr1_contour_y);

% Dibujo del robot de tracción 1

    hd_wmr1 = fill(wmr1_pos_x, wmr1_pos_y, 'b');
    hold on;
    set(hd_wmr1,'EraseMode','none','FaceColor','none','EdgeColor','b');

    axis([-Axis_Width Axis_Width -Axis_Width Axis_Width]);
    grid on;


% Realización del robot de tracción 2
```

```
    wmr2_contour_x = [-(c+1.1); -(c+1.1); 3.9; 3.9];
    wmr2_contour_y = [-d; d; d; -d];

    wmr2_pos_x = xP + cos_thP*(a2 + cos_th2*wmr2_contour_x - sin_th2*wmr2_contour_y) -
sin_thP*(b + sin_th2*wmr2_contour_x + cos_th2*wmr2_contour_y);
    wmr2_pos_y = yP + sin_thP*(a2 + cos_th2*wmr2_contour_x - sin_th2*wmr2_contour_y) +
cos_thP*(b + sin_th2*wmr2_contour_x + cos_th2*wmr2_contour_y);

% Dibujo del robot de tración 2

    hd_wmr2 = fill(wmr2_pos_x, wmr2_pos_y, 'c');
    set(hd_wmr2,'EraseMode','none','FaceColor','none','EdgeColor','c');


% Realización de la rueda

    wheel_contour_x = [-r; -r; r; r];
    wheel_contour_y = [-(wheel_y/2); (wheel_y/2); (wheel_y/2); -(wheel_y/2)];

% Dibujo de la rueda 1,1

    wheel11_pos_x = xP + cos_thP*(a1 + cos_th1*(-c + wheel_contour_x) - sin_th1*(d +
wheel_contour_y)) - sin_thP*(b + sin_th1*(-c + wheel_contour_x) + cos_th1*(d +
wheel_contour_y));
    wheel11_pos_y = yP + sin_thP*(a1 + cos_th1*(-c + wheel_contour_x) - sin_th1*(d +
wheel_contour_y)) + cos_thP*(b + sin_th1*(-c + wheel_contour_x) + cos_th1*(d +
wheel_contour_y));

    hd_wheel11 = fill(wheel11_pos_x, wheel11_pos_y, 'g');
    set(hd_wheel11,'EraseMode','none','EdgeColor','g');

% Dibujo de la rueda 1,2

    wheel12_pos_x = xP + cos_thP*(a1 + cos_th1*(-c + wheel_contour_x) - sin_th1*(-d +
wheel_contour_y)) - sin_thP*(b + sin_th1*(-c + wheel_contour_x) + cos_th1*(-d +
wheel_contour_y));
    wheel12_pos_y = yP + sin_thP*(a1 + cos_th1*(-c + wheel_contour_x) - sin_th1*(-d +
wheel_contour_y)) + cos_thP*(b + sin_th1*(-c + wheel_contour_x) + cos_th1*(-d +
wheel_contour_y));

    hd_wheel12 = fill(wheel12_pos_x, wheel12_pos_y, 'g');
    set(hd_wheel12,'EraseMode','none','EdgeColor','g');

% Dibujo de la rueda 2,1

    wheel21_pos_x = xP + cos_thP*(a2 + cos_th2*(-c + wheel_contour_x) - sin_th2*(d +
wheel_contour_y)) - sin_thP*(b + sin_th2*(-c + wheel_contour_x) + cos_th2*(d +
wheel_contour_y));
    wheel21_pos_y = yP + sin_thP*(a2 + cos_th2*(-c + wheel_contour_x) - sin_th2*(d +
wheel_contour_y)) + cos_thP*(b + sin_th2*(-c + wheel_contour_x) + cos_th2*(d +
wheel_contour_y));

    hd_wheel21 = fill(wheel21_pos_x, wheel21_pos_y, 'g');
    set(hd_wheel21,'EraseMode','none','EdgeColor','g');

% Dibujo de la rueda 2,2

    wheel22_pos_x = xP + cos_thP*(a2 + cos_th2*(-c + wheel_contour_x) - sin_th2*(-d +
wheel_contour_y)) - sin_thP*(b + sin_th2*(-c + wheel_contour_x) + cos_th2*(-d +
wheel_contour_y));
    wheel22_pos_y = yP + sin_thP*(a2 + cos_th2*(-c + wheel_contour_x) - sin_th2*(-d +
wheel_contour_y)) + cos_thP*(b + sin_th2*(-c + wheel_contour_x) + cos_th2*(-d +
wheel_contour_y));

    hd_wheel22 = fill(wheel22_pos_x, wheel22_pos_y, 'g');
    set(hd_wheel22,'EraseMode','none','EdgeColor','g');
```

```
% Realización de la plataforma

    platform_contour_x = [a1-extraP_x1; a1-extraP_x1; a2+extraP_x1; a2+extraP_x1; a2-
extraP_x2; a2-extraP_x3; a1+extraP_x3; a1+extraP_x4];
    platform_contour_y = [b+extraP_y1; b-extraP_y1; b-extraP_y1; b+extraP_y1; b+extraP_y1;
extraP_y2+(b-extraP_y1); extraP_y2+(b-extraP_y1); b+extraP_y1];

    platform_pos_x = xP + cos_thP*platform_contour_x - sin_thP*platform_contour_y;
    platform_pos_y = yP + sin_thP*platform_contour_x + cos_thP*platform_contour_y;

% Dibujo de la plataforma

    hd_platform = fill(platform_pos_x, platform_pos_y,'r');
    set(hd_platform,'EraseMode','none','FaceColor','none','EdgeColor','r');


% Realización del punto P

    Point_Sides = 10;
    Point_Sides_Angles = (0:1/Point_Sides:1-1/(2*Point_Sides))'*2*pi;

    Point_Size = 2;
    Point_Size_y = Point_Size*sin(Point_Sides_Angles);
    Point_Size_x = Point_Size*cos(Point_Sides_Angles);

    Point_y = Point_Size_y + yP;
    Point_x = Point_Size_x + xP;


% Dibujo del punto P

    hd_Point = fill(Point_x, Point_y, 'y');
    set(hd_Point,'EraseMode','none');

% Realización del punto P1

    Point_Sides = 10;
    Point_Sides_Angles = (0:1/Point_Sides:1-1/(2*Point_Sides))'*2*pi;

    Point_Size = 1;
    Point_Size_y = Point_Size*sin(Point_Sides_Angles);
    Point_Size_x = Point_Size*cos(Point_Sides_Angles);

    Point_y = Point_Size_y + yP + sin_thP*a1 + cos_thP*b;
    Point_x = Point_Size_x + xP + cos_thP*a1 - sin_thP*b;


% Dibujo del punto P1

    hd_Point = fill(Point_x, Point_y, 'k');
    set(hd_Point,'EraseMode','none');

% Realización del punto P2

    Point_Sides = 10;
    Point_Sides_Angles = (0:1/Point_Sides:1-1/(2*Point_Sides))'*2*pi;

    Point_Size = 1;
    Point_Size_y = Point_Size*sin(Point_Sides_Angles);
    Point_Size_x = Point_Size*cos(Point_Sides_Angles);

    Point_y = Point_Size_y + yP + sin_thP*a2 + cos_thP*b;
    Point_x = Point_Size_x + xP + cos_thP*a2 - sin_thP*b;
```

```
% Dibujo del punto P2

    hd_Point = fill(Point_x, Point_y, 'k');
    set(hd_Point,'EraseMode','none');


% Guardado de los objetos

    set_param(gcb,'UserData',[hd_platform; hd_wmr1; hd_wmr2; hd_wheel11; hd_wheel12;
hd_wheel21; hd_wheel22; hd_Point]);


% end mdlUpdate

%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%
function sys=mdlOutputs(t,x,u)

sys = [];



% end mdlOutputs

%
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1;    %  Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
%
function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

Apéndice II Programa de los microcontroladores

Microcontrolador del robot de tracción 1

```
#include <Wire.h>

#define md25Dir 0x58    // Dirección de la MD25
#define speed1 0x01     // Dirección para la velocidad de la rueda 1,1
#define speed2 0x00     // Dirección para la velocidad de la rueda 1,2

byte vel1=128, vel2=128, i=0;
byte rev[5]={128,128,128,128,0};
int checksum=0;

void setup(){
  Serial.begin(4800);  //Inicio de comunicación serial
  Wire.begin();       //Inicio de comunicación i2c
  com_MD25(vel1,vel2); //Detención de los motores de las ruedas
}

void loop(){
  checksum=0; //Reinicialización del corroborador de la trama

//Recepción de la trama de comunicación serial
  if(Serial.available()==5){
    for(i=0;i<5;i++){
      rev[i]=Serial.read();
    }
    Serial.flush();
// Cálculo del checksum interno
    for(i=0;i<4;i++){
      checksum=checksum+rev[i];
    }
    while(checksum>255){
      checksum=checksum-256;
    }
    checksum=255-checksum;

// Validación de la información recibida a través del checksum
    if(byte(checksum)==rev[4]){
      vel1=rev[0];
      vel2=rev[1];
    }
  }

// Envío de los comandos de velocidad a los motores de las ruedas
  com_MD25(vel1,vel2);
}

void com_MD25(byte v1, byte v2){
// Rutina de envío de los comandos de velocidad vía i2c

  Wire.beginTransmission(md25Dir);
  Wire.send(speed1);
  Wire.send(v1);// Velocidad de la rueda 1,1
  Wire.endTransmission();

  Wire.beginTransmission(md25Dir);
  Wire.send(speed2);
```

```
  Wire.send(v2);// Velocidad de la rueda 1,2
  Wire.endTransmission();
}
```

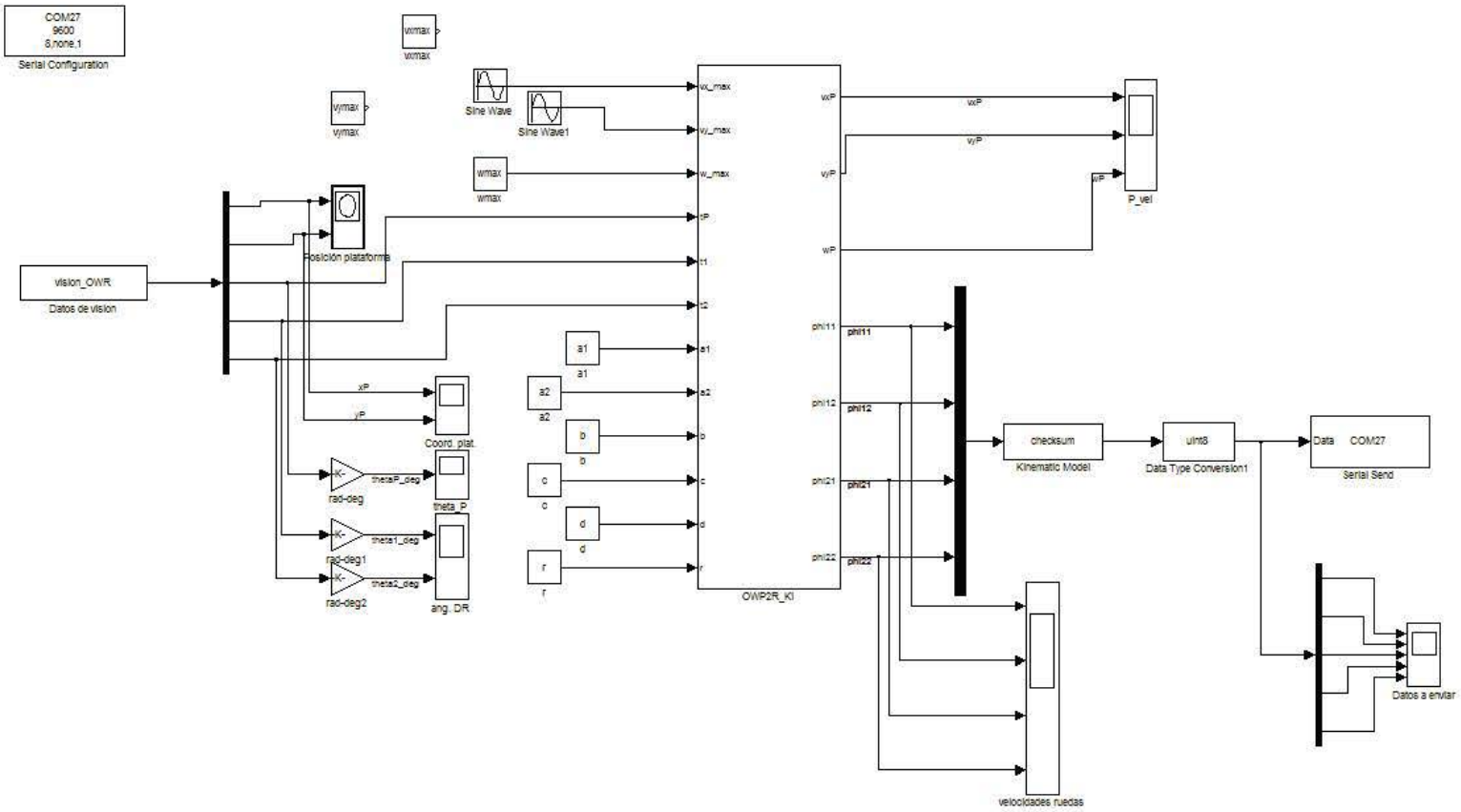
Microcontrolador del robot de tracción 2

```
#include <Wire.h>

#define md25Dir 0x58    // Dirección de la MD25
#define speed1 0x01     // Dirección para la velocidad de la rueda 2,1
#define speed2 0x00     // Dirección para la velocidad de la rueda 2,2

byte vel1=128, vel2=128, i=0;
byte rev[5]={128,128,128,128,0};
int checksum=0;

void setup(){
  Serial.begin(4800);  //Inicio de comunicación serial
  Wire.begin();        //Inicio de comunicación i2c
  com_MD25(vel1,vel2); //Detención de los motores de las ruedas
}

void loop(){
  checksum=0; //Reinicialización del corroborador de la trama

//Recepción de la trama de comunicación serial
  if(Serial.available()==5){
    for(i=0;i<5;i++){
      rev[i]=Serial.read();
    }
    Serial.flush();
// Cálculo del checksum interno
    for(i=0;i<4;i++){
      checksum=checksum+rev[i];
    }
    while(checksum>255){
      checksum=checksum-256;
    }
    checksum=255-checksum;

// Validación de la información recibida a través del checksum
    if(byte(checksum)==rev[4]){
      vel1=rev[2];
      vel2=rev[3];
    }
  }

// Envío de los comandos de velocidad a los motores de las ruedas
  com_MD25(vel1,vel2);
}

void com_MD25(byte v1, byte v2){
// Rutina de envío de los comandos de velocidad vía i2c

  Wire.beginTransmission(md25Dir);
  Wire.send(speed1);
  Wire.send(v1);// Velocidad de la rueda 2,1
  Wire.endTransmission();
```

```
  Wire.beginTransmission(md25Dir);
  Wire.send(speed2);
  Wire.send(v2);// Velocidad de la rueda 2,2
  Wire.endTransmission();
}
```

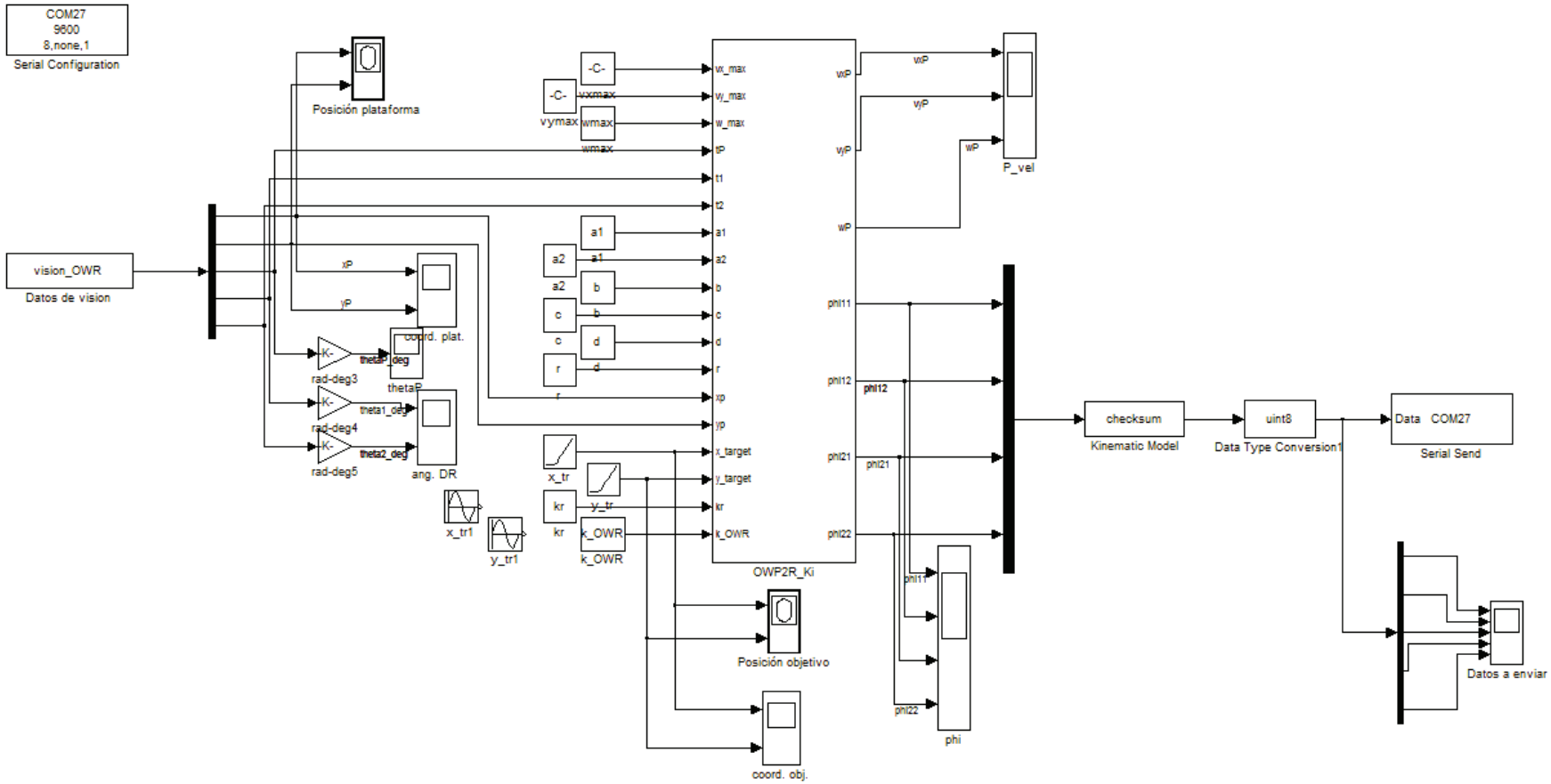
Apéndice III Diagramas de bloque y programación de las pruebas

Lazo abierto



*Los bloques desconectados de valores constantes se alternan con los de funciones sinusoidales para los distintos tipos de prueba.

Lazo cerrado: seguimiento de trayectoria



*Los bloques desconectados de funciones sinusoidales se alternan con los de funciones rampa para los distintos tipos de prueba.

Programación de la función S para la adquisición de datos de visión

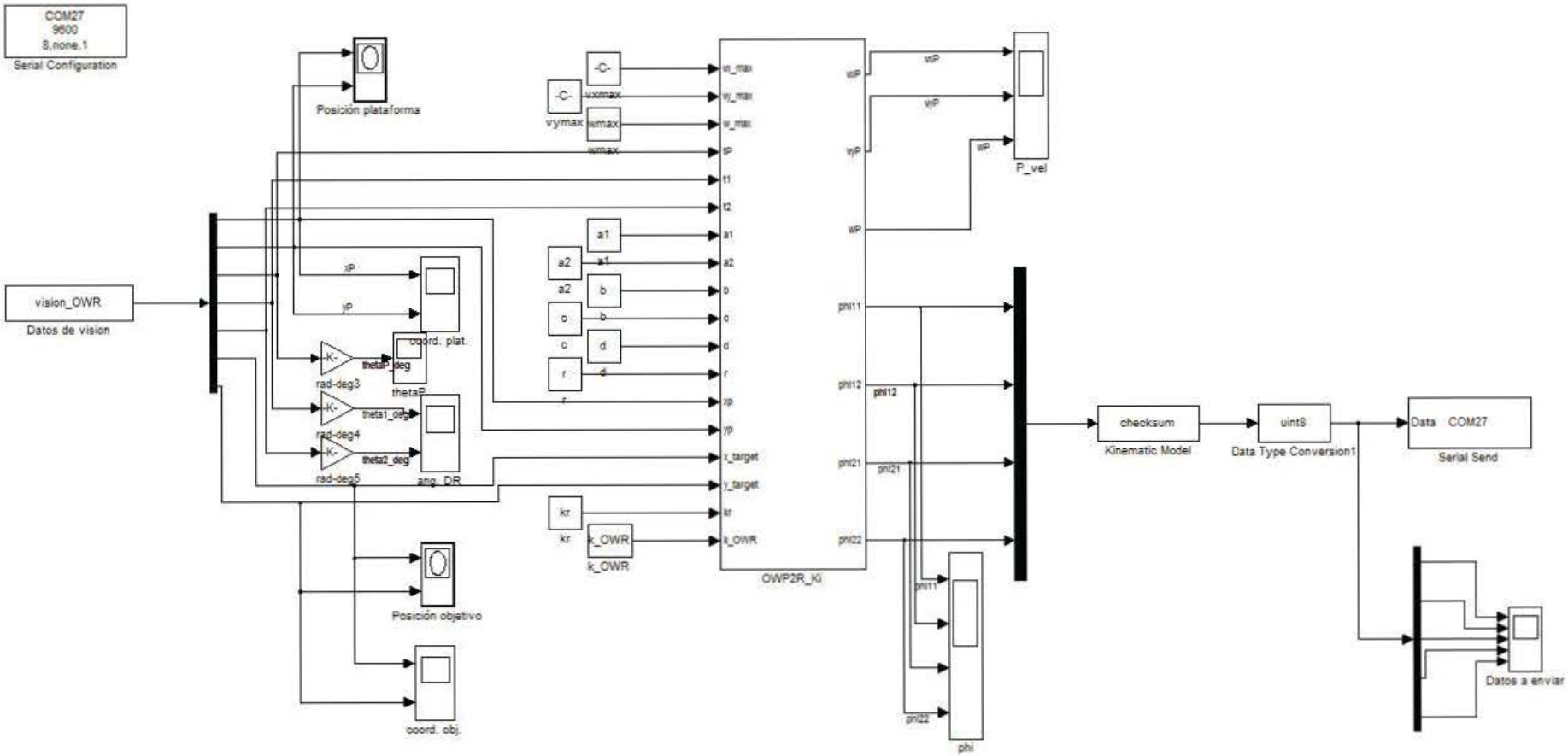
```
function [sys,x0,str,ts] = vision_OWR(t,x,u,flag)

% Programa creado por Arturo Martínez Carrillo,
% alumno de la Facultad de Ingeniería, de la Universidad Nacional Autónoma de México,
% para el proyecto de tesis "Plataforma móvil omnidireccional a partir de dos robots móviles
diferenciales"

% Rutina para la obtención de datos del sistema de visión a través del programa
reacTIVision.
% Detalles a contemplar:
%
%     1) La posición está dada en cm y ha sido ajustada para un espacio de trabajo de
196x147[cm] con el origen en el centro del espacio de trabajo.
%     2) Los ángulos obtenidos están dados en radianes y son absolutos con respecto al
sistema de referencia.

% Las salidas obtenidas son:
%
% o(1) = x_P  coordenada en X del punto de análisis P
% o(2) = y_P  coordenada en Y del punto de análisis P
% o(3) = t_P  ángulo de orientación de la plataforma móvil
% 0(4) = t_1  ángulo de orientación del robot de tracción 1
% 0(5) = t_2  ángulo de orientación del robot de tracción 2

switch flag,

  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  case 1,
    sys=mdlDerivatives(t,x,u);

  case 2,
    sys=mdlUpdate(t,x,u);

  case 3,
    sys=mdlOutputs(t,x,u,flag);

  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  case 9,
    sys=mdlTerminate(t,x,u);

  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts,T_samp]=mdlInitializeSizes

sizes = simsizes;

sizes.NumContStates  = 0;
```

```
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 5;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

x0  = [];

str = [];

ts  = [0 0];

% end mdlInitializeSizes

%
%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u)

sys = [];

% end mdlDerivatives

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
function sys=mdlUpdate(t,x,u)

sys = [ ];

% end mdlUpdate

%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%

function sys=mdlOutputs(t,x,u, flag)

%Función para obtener datos de Reactivision

%1 -> ángulo theta_P (FD 3)
%2 -> ángulo theta_1 (FD 1)
%3 -> ángulo theta_2 (FD 4)

global t_P x_P y_P t_1 t_2;

import TUIO.*;
tc = TuioClient();
tc.connect();

for i = 1 : 2

    c=tc.getTuioObjects().size();
```

```
    if (c > 0)
        f1 = tc.getTuioObjects().get(0).getSymbolID();
            switch f1
                case 3
                    t_P = tc.getTuioObjects().get(0).getAngle();
                    x_P = tc.getTuioObjects().get(0).getPosition().getX();
                    y_P = tc.getTuioObjects().get(0).getPosition().getY();
                case 1
                    t_1 = tc.getTuioObjects().get(0).getAngle();
                case 4
                    t_2 = tc.getTuioObjects().get(0).getAngle();
                otherwise
            end

        if(c > 1)
            f2 = tc.getTuioObjects().get(1).getSymbolID();
            switch f2
                case 3
                    t_P = tc.getTuioObjects().get(1).getAngle();
                    x_P = tc.getTuioObjects().get(1).getPosition().getX();
                    y_P = tc.getTuioObjects().get(1).getPosition().getY();
                case 1
                    t_1 = tc.getTuioObjects().get(1).getAngle();
                case 4
                    t_2 = tc.getTuioObjects().get(1).getAngle();
                otherwise
            end

            if(c > 2)
                f3 = tc.getTuioObjects().get(2).getSymbolID();
                switch f3
                    case 3
                        t_P = tc.getTuioObjects().get(2).getAngle();
                        x_P = tc.getTuioObjects().get(2).getPosition().getX();
                        y_P = tc.getTuioObjects().get(2).getPosition().getY();
                    case 1
                        t_1 = tc.getTuioObjects().get(2).getAngle();
                    case 4
                        t_2 = tc.getTuioObjects().get(2).getAngle();
                    otherwise
                end
            end;
        end;
    end;
    pause(0.1);
end;

tc.disconnect();

sys = [(x_P*-196)+98 (y_P*-147)+73.5 t_P t_1 t_2];

% end mdlOutputs

%
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sys = [];
% sampleTime = 1;    %  Example, set the next hit to be one second later.
% sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit
```

```
%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
%
function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

Lazo cerrado: alcance de objetivo



Programación de la función S para la adquisición de datos de visión

```
function [sys,x0,str,ts] = vision_OWR(t,x,u,flag)

% Programa creado por Arturo Martínez Carrillo,
% alumno de la Facultad de Ingeniería, de la Universidad Nacional Autónoma de México,
% para el proyecto de tesis "Plataforma móvil omnidireccional a partir de dos robots móviles
diferenciales"

% Rutina para la obtención de datos del sistema de visión a través del programa
reacTIVision.
% Detalles a contemplar:
%
%     1) La posición está dada en cm y ha sido ajustada para un espacio de trabajo de
196x147[cm] con el origen en el centro del espacio de trabajo.
%     2) Los ángulos obtenidos están dados en radianes y son absolutos con respecto al
sistema de referencia.

% Las salidas obtenidas son:
%
% o(1) = x_P  coordenada en X del punto de análisis P
% o(2) = y_P  coordenada en Y del punto de análisis P
% o(3) = t_P  ángulo de orientación de la plataforma móvil
% 0(4) = t_1  ángulo de orientación del robot de tracción 1
% 0(5) = t_2  ángulo de orientación del robot de tracción 2
% o(6) = x_tr coordenada en X del objetivo
% o(7) = y_tr coordenada en Y del objetivo

switch flag,

  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  case 1,
    sys=mdlDerivatives(t,x,u);

  case 2,
    sys=mdlUpdate(t,x,u);

  case 3,
    sys=mdlOutputs(t,x,u,flag);

  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  case 9,
    sys=mdlTerminate(t,x,u);

  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts,T_samp]=mdlInitializeSizes

sizes = simsizes;
```

```
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 7;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

x0  = [];

str = [];

ts  = [0 0];

% end mdlInitializeSizes

%
%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u)

sys = [];

% end mdlDerivatives

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
function sys=mdlUpdate(t,x,u)

sys = [ ];

% end mdlUpdate

%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%

function sys=mdlOutputs(t,x,u, flag)

%Función para obtener datos de Reactivision

%1 -> ángulo theta_P (FD 3)
%2 -> ángulo theta_1 (FD 1)
%3 -> ángulo theta_2 (FD 4)
%4 -> Objetivo (FD 0)

global t_P x_P y_P t_1 t_2 x_tr y_tr;

import TUIO.*;
tc = TuioClient();
tc.connect();

for i = 1 : 3
```

```
c=tc.getTuioObjects().size();

if (c > 0)
    f1 = tc.getTuioObjects().get(0).getSymbolID();
        switch f1
            case 3
                t_P = tc.getTuioObjects().get(0).getAngle();
                x_P = tc.getTuioObjects().get(0).getPosition().getX();
                y_P = tc.getTuioObjects().get(0).getPosition().getY();
            case 1
                t_1 = tc.getTuioObjects().get(0).getAngle();
            case 4
                t_2 = tc.getTuioObjects().get(0).getAngle();
            case 0
                x_tr = tc.getTuioObjects().get(0).getPosition().getX();
                y_tr = tc.getTuioObjects().get(0).getPosition().getY();
            otherwise
        end

    if(c > 1)
        f2 = tc.getTuioObjects().get(1).getSymbolID();
        switch f2
            case 3
                t_P = tc.getTuioObjects().get(1).getAngle();
                x_P = tc.getTuioObjects().get(1).getPosition().getX();
                y_P = tc.getTuioObjects().get(1).getPosition().getY();
            case 1
                t_1 = tc.getTuioObjects().get(1).getAngle();
            case 4
                t_2 = tc.getTuioObjects().get(1).getAngle();
            case 0
                x_tr = tc.getTuioObjects().get(1).getPosition().getX();
                y_tr = tc.getTuioObjects().get(1).getPosition().getY();
            otherwise
        end

        if(c > 2)
            f3 = tc.getTuioObjects().get(2).getSymbolID();
            switch f3
                case 3
                    t_P = tc.getTuioObjects().get(2).getAngle();
                    x_P = tc.getTuioObjects().get(2).getPosition().getX();
                    y_P = tc.getTuioObjects().get(2).getPosition().getY();
                case 1
                    t_1 = tc.getTuioObjects().get(2).getAngle();
                case 4
                    t_2 = tc.getTuioObjects().get(2).getAngle();
                case 0
                    x_tr = tc.getTuioObjects().get(2).getPosition().getX();
                    y_tr = tc.getTuioObjects().get(2).getPosition().getY();
                otherwise
            end
            if(c > 3)
                f3 = tc.getTuioObjects().get(3).getSymbolID();
                switch f3
                    case 3
                        t_P = tc.getTuioObjects().get(3).getAngle();
                        x_P = tc.getTuioObjects().get(3).getPosition().getX();
                        y_P = tc.getTuioObjects().get(3).getPosition().getY();
                    case 1
                        t_1 = tc.getTuioObjects().get(3).getAngle();
                    case 4
                        t_2 = tc.getTuioObjects().get(3).getAngle();
                    case 0
                        x_tr = tc.getTuioObjects().get(3).getPosition().getX();
                        y_tr = tc.getTuioObjects().get(3).getPosition().getY();
                    otherwise
                end
            end;
```

```
            end;
        end;
    end;
    pause(0.1);
end;

tc.disconnect();

sys = [(x_P*-196)+98 (y_P*-147)+73.5 t_P t_1 t_2 (x_tr*-196)+98 (y_tr*-147)+73.5];

% end mdlOutputs

%
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sys = [];
% sampleTime = 1;    %  Example, set the next hit to be one second later.
% sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
%
function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

Programación de la función S para el acondicionamiento de los datos a enviar

```
function [sys,x0,str,ts] = checksum(t,x,u,flag)

% Programa creado por Arturo Martínez Carrillo,
% alumno de la Facultad de Ingeniería, de la Universidad Nacional Autónoma de México,
% para el proyecto de tesis "Plataforma móvil omnidireccional a partir de dos robots móviles
diferenciales"
%
% Rutina para el acondicionamiento de datos a enviar vía puerto serial. Se contemplan los
siguientes parámetros:
%
%     1) Velocidades de las ruedas, de [rad/s] a valores de 0 a 255 como entradas para la
tarjeta MD25 (driver de los motores),
%        siendo 0 la velocidad máxima en un sentido y 255 la máxima en el otro.
%     2) Checksum para corroborar que la trama enviada sea recibida correctamente.
%
% Las entradas están definidas de la siguiente manera:
%
% u(1) = phi11      Velocidad de giro de la rueda 1,1 de la plataforma omnidireccional
% u(2) = phi12      Velocidad de giro de la rueda 1,2 de la plataforma omnidireccional
```

```
% u(3) = phi21      Velocidad de giro de la rueda 2,1 de la plataforma omnidireccional
% u(4) = phi22      Velocidad de giro de la rueda 2,2 de la plataforma omnidireccional
%
% Las salidas están definidas de la siguiente manera:
%
% o(1) = vel1,1     Valor de velocidad para la rueda 1,1
% o(2) = vel1,2     Valor de velocidad para la rueda 1,2
% o(3) = vel2,1     Valor de velocidad para la rueda 2,1
% o(4) = vel2,2     Valor de velocidad para la rueda 2,2
% o(5) = checksum   Byte para confirmar la correcta recepción de datos

switch flag,

  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  case 1,
    sys=mdlDerivatives(t,x,u);

  case 2,
    sys=mdlUpdate(t,x,u);

  case 3,
    sys=mdlOutputs(t,x,u);

  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  case 9,
    sys=mdlTerminate(t,x,u);

  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts,T_samp]=mdlInitializeSizes

%
sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 5;
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

x0  = [];

str = [];

ts  = [0.1 0];

% end mdlInitializeSizes

%
```

```
%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u)

sys = [];

% end mdlDerivatives

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
function sys=mdlUpdate(t,x,u)

sys = [ ];

% end mdlUpdate

%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%
function sys=mdlOutputs(t,x,u)


    vel(1)=6.1127*u(1)+128;
    vel(2)=-6.1127*u(2)+128;
    vel(3)=6.1127*u(3)+128;
    vel(4)=-6.1127*u(4)+128;

    if vel(1) < 0
        vel(1)=0;
    elseif vel(1) > 255
        vel(1)=255;
    end

    if vel(2) < 0
        vel(2)=0;
    elseif vel(2) > 255
        vel(2)=255;
    end

    if vel(3) < 0
        vel(3)=0;
    elseif vel(3) > 255
        vel(3)=255;
    end

    if vel(4) < 0
        vel(4)=0;
    elseif vel(4) > 255
        vel(4)=255;
    end

    vel(1)=uint8(vel(1));
    vel(2)=uint8(vel(2));
    vel(3)=uint8(vel(3));
    vel(4)=uint8(vel(4));

    sumaB = 0;
```

```
        for i=1:4
            sumaB = sumaB + vel(i);
        end
        while sumaB > 255
            sumaB = sumaB - 256;
        end
        checksumB = 255 - sumaB;

    sys = [vel, checksumB];


% end mdlOutputs

%
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)

sampleTime = 1;    %  Example, set the next hit to be one second later.
sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
%
function sys=mdlTerminate(t,x,u)

sys = [];

% end mdlTerminate
```

REFERENCIAS

[1] Cuellar, F. (2006) *Analysis and Design of a Wheeled Holonomic Omnidirectional Robot*. IEEE 3rd Latin American Robotics Symposium. Perú

[2] Holmberg, R. et al. (1999) *Development of a Holonomic Mobile Robot for Mobile Manipulation Tasks.* International Conference on Field and Service Robotics, Pittsburgh, PA. Estados Unidos

[3] Wada, M. (1995) *Omnidirectional Holonomic Mobile Robot Using Nonholonomic Wheels.* IEEE/RSJ International Conference on Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots'. Japón

[4] Udengaard, M. et al. (2008) *Design of an Omnidirectional Mobile Robot for Rough Terrain.* IEEE International Conference on Robotics and Automation Pasadena, CA. Estados Unidos

[5] Yamada, T. et al. (2001) *Dynamic Model and Control for a Holonomic Omnidirectional Mobile Robot.* Autonomous Robots 11. Japón; págs. 173 - 189.

[6] González-Villela, V.J. (2006) *IV A Unifying Theory on Conventional Wheeled Mobile Robots.* Research on a semiautonomous mobile robot for loosely structured environments focused on transporting mail trolleys. Reino Unido

[7] Coelho, P. et al. (2003) *Lie Algebra Application to Mobile Robot Control: A Tutorial.* Robotica volumen 21. Portugal; págs. 483 - 493

[8] González-Villela, V.J. et al. (2004) *A Wheeled Mobile Robot with Obstacle Avoidance Capability.* SOMIM Ingeniería mecánica tecnología y desarrollo Vol. 1 No. 5. México; págs. 159 - 166

[9] http://www.mathworks.es/products/simulink/?s_cid=global_nav

[10] http://www.robot-electronics.co.uk/htm/emg30.htm

[11] http://www.msebilbao.com/notas/downloads/Doble%20puente%20H%20para%20Motores%20MD25.pdf

[12] http://arduino.cc/en/

[13] http://arduino.cc/en/Main/ArduinoBoardDuemilanove

[14] http://www.arduino.cc/en/Main/ArduinoXbeeShield

[15] http://reactivision.sourceforge.net/

[16] http://www.tuio.org/?specification

[17] ftp://ftp1.digi.com/support/documentation/90000866_A.pdf

[18] http://www.digi.com/technology/rf-articles/wireless-zigbee