

# Control de plataforma móvil de cuatro ruedas mecanum con modelo cinemático

1<sup>st</sup> Martínez, Brayan Steven  
Ingeniería Mecatrónica  
Universidad EIA  
Medellín, Colombia  
brayan.martinez@eia.edu.co

2<sup>nd</sup> Gongora, Juan Pablo  
Ingeniería Mecatrónica  
Universidad EIA  
Medellín, Colombia  
juan.gongora@eia.edu.co

3<sup>rd</sup> Isaza, Luis Miguel  
Ingeniería Mecatrónica  
Universidad EIA  
Medellín, Colombia  
luis.isaza@eia.edu.co

4<sup>th</sup> Jiménez, Sebastian David  
Ingeniería Mecatrónica  
Universidad EIA  
Medellín, Colombia  
sebastian.jimenez10@eia.edu.co

5<sup>th</sup> aaa, aaaa aaaa  
Ingeniería Mecatrónica  
Universidad EIA  
Medellín, Colombia  
email

**Abstract**—El movimiento de plataformas móviles está compuesto de estrategias de control, modelos matemáticos que describen el movimiento del mismo, planificación de trayectorias y planificación de tareas, además de sensores que deben complementarse para lograr el objetivo deseado.

**Index Terms**—robotica, trayectoria, cinemática, navegación

## I. INTRODUCCIÓN

Se presentan las simulaciones asociadas a un modelo cinemático para un robot omnidireccional con cuatro ruedas estilo mecanum teniendo de cuenta los parámetros dimensionales del robot, más no, los análisis eléctricos o de leyes de control aplicables, se da una posible ruta para continuar con el trabajo a fin de contribuir a la construcción de un robot de servicio.

## II. MATERIALES Y MÉTODO

### A. Materiales

**Matlab** para realizar matemática computacional, programación de algoritmos y gráficos a partir de los resultados.

**CoppeliaSim** para llevar a cabo simulaciones en un entorno virtual teniendo a disposición robots predeterminados, sensores y un API para conectarse con otros lenguajes de programación.

**Odroid-XU4** como tarjeta de desarrollo para ser implementada de forma tentativa en el robot. Características de buen rendimiento y tamaño la convierten en una opción para integrar todos los sistemas.

**ROS (Robot Operating System)** como framework para complementar varios sistemas en uno solo y permitir una fluida comunicación entre los mismo. Se instala sobre una distribución de Linux.

**Ubuntu** como sistema operativo para implementar en el robot aprovechando las ventajas de ROS y la compatibilidad con sistemas de desarrollo como la ODROID-XU4 y sensores.

**Modelo cinemático** [1] para predecir el comportamiento de la plataforma que posee cuatro ruedas suecas (mecanum)

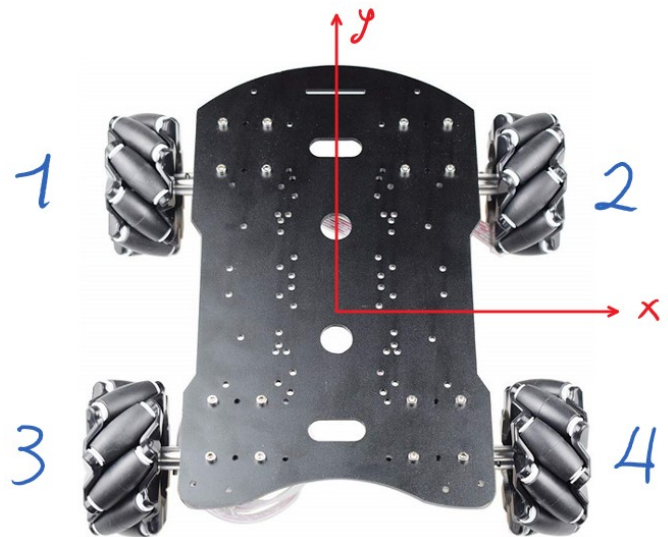


Fig. 1: Plataforma mecánica con motores y ruedas

**Plataforma móvil** con cuatro motores y ruedas suecas (mecanum), la configuración para este trabajo es como sigue en la Figura 1.

### B. Método

1) *Simulación Matlab*: Se analiza el modelo cinemático ya conocido:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & 1 & -(L+l) \\ -1 & 1 & (L+l) \\ -1 & 1 & -(L+l) \\ 1 & 1 & (L+l) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix} \quad (1)$$

En base a la Figura 2, Los parámetros  $L$ ,  $l$  y  $r$ , pertenecen al robot, para esta simulación, al KUKA YouBot [2], donde:  $r = 100$  mm,  $L = 235.5$  mm,  $l = 150$  mm.

A este modelo matemático se le ingresan las velocidades en  $X$ ,  $Y$  y rotación angular del robot, para obtener la velocidad

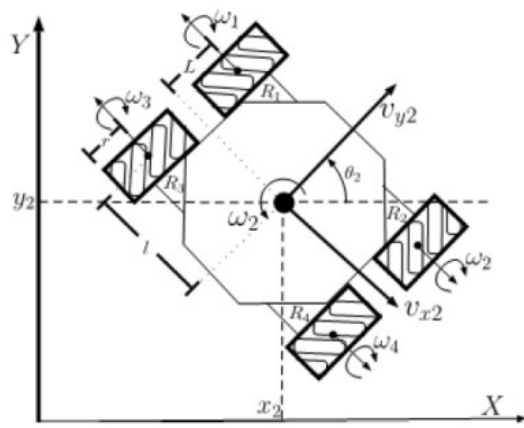


Fig. 2: Robot omnidireccional a modelar

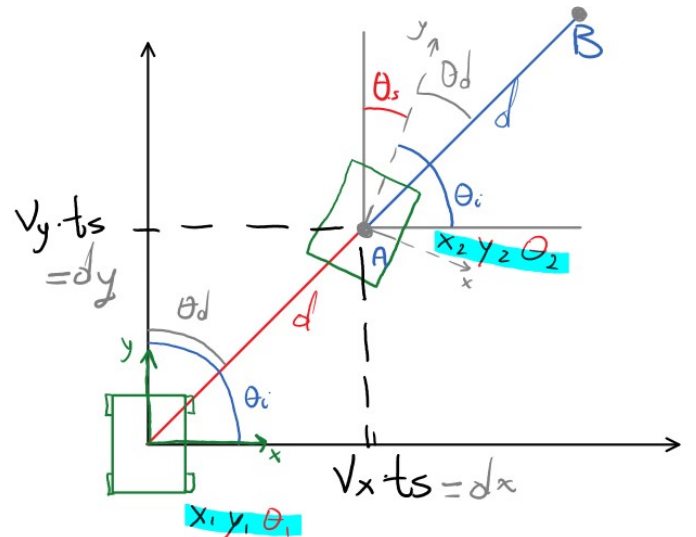


Fig. 4: Desplazamiento del robot en el plano cartesiano

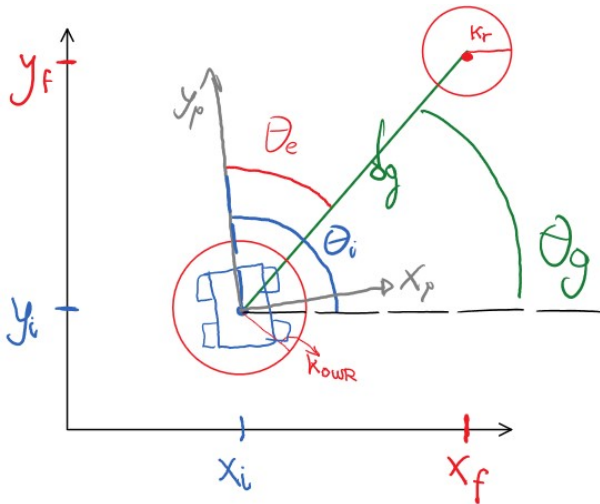


Fig. 3: Robot en el plano con campos potenciales

angular de cada rueda. Se programa la función que representa este modelo cinámico en Matlab y además, otra función para convertir las posiciones deseadas en velocidad lineal usando las ecuaciones 3, 2 y 4 teniendo en cuenta una metodología de control por campos potenciales [3] visibles en la Figura 3

$$V_x = \begin{cases} V_{x_{max}} \sin \theta_e & \text{si } d_g > K_r + K_{owr} \\ \frac{d_g}{K_r + K_{owr}} \sin \theta_e & \text{si } d_g \leq K_r + K_{owr} \\ 0 & \text{si } d_g = 0 \end{cases} \quad (2)$$

$$V_y = \begin{cases} V_{y_{max}} \cos \theta_e & \text{si } d_g > K_r + K_{owr} \\ \frac{d_g}{K_r + K_{owr}} \cos \theta_e & \text{si } d_g \leq K_r + K_{owr} \\ 0 & \text{si } d_g = 0 \end{cases} \quad (3)$$

$$\omega = \omega_{max} + \sin \theta_e \quad (4)$$

Con las ecuaciones 3, 2 y 4 se pueden obtener las velocidades angulares de cada rueda gracias al modelo cinámico.

Observando la Figura 3 se nota que es necesario en todo momento conocer la posición actual del robot y así calcular los vectores y ángulos; es fácil de obtener cuando hay otro sistema externo que proporciona esas coordenadas, pero, para una simulación en Matlab, es necesario encontrar los nuevos valores cada que el objeto se desplazada con las ecuaciones 9, 10 y 11, cuyas variables se pueden observar en la Figura 4.

$$d = \sqrt{(d_x)^2 + (d_y)^2} \quad (5)$$

$$d_x = V_x T_s \quad (6)$$

$$d_y = V_y T_s \quad (7)$$

$$\theta_d = \tan^{-1} \frac{d_x}{d_y} \quad (8)$$

$$x_2 = x_1 + \left( d \sin \left( \theta_d + \frac{\pi}{2} - \theta_i \right) \right) \quad (9)$$

$$y_2 = y_1 + \left( d \cos \left( \theta_d + \frac{\pi}{2} - \theta_i \right) \right) \quad (10)$$

$$\theta_{i_2} = \theta_{i_1} + \omega T_s \quad (11)$$

Se recomienda ir al código para analizar la secuencia lógica.

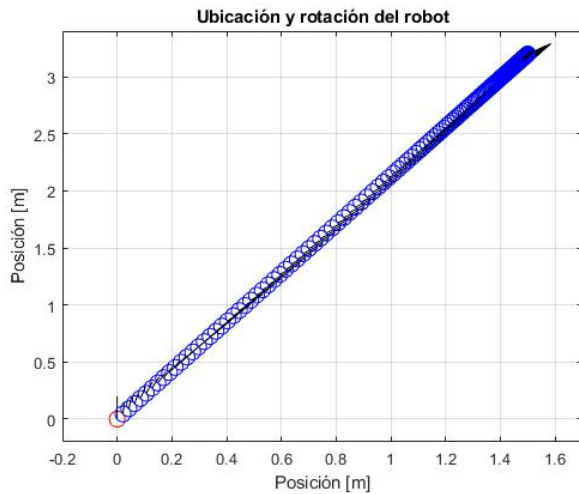


Fig. 5: Registro de movimiento del robot

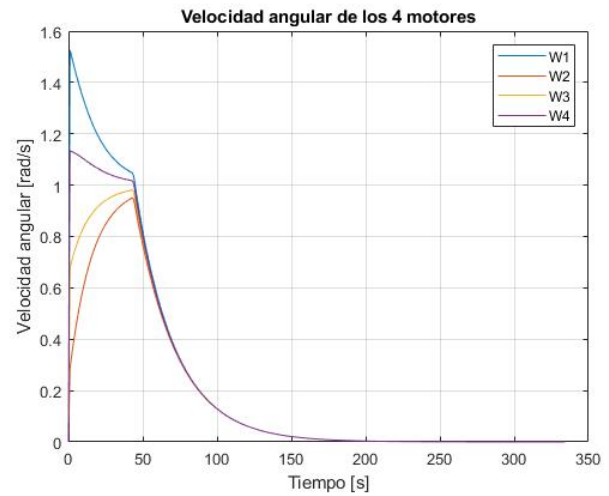


Fig. 6: Velocidad angular de los motores en el tiempo

2) *Simulación con CoppeliaSim*: Es necesario usar la API de CoppeliaSim en Matlab para llevar a cabo la simulación aprovechando que ya se tiene el modelo cinemático programado y funcionando en Matlab.

Buscar los archivos del API en la ruta de CoppeliaSim : "CoppeliaRobotics CoppeliaSimEdu programming remoteApiBindings matlab matlab", para copiar únicamente los siguientes archivos en la carpeta con el main de Matlab para controlar el robot:

- remApi
- remoteApiProto
- simpleText

En la misma carpeta, copiar otro archivo llamado remoteApi.dll que se encuentra en: "CoppeliaRobotics CoppeliaSimEdu programming remoteApiBindings lib lib Windows"

El archivo 'simpleText' sirve como guía para usar el API con Matlab.

### III. RESULTADOS

#### A. Simulación Matlab

Se obtiene una primera simulación numérica usando Matlab. El robot comienza en (0,0) a  $90^\circ$  de la horizontal, se desea que llegue a (1.5, 3.2). El resultado se puede ver en la Figura 5. La orientación del robot se va alineando con el vector definido por el punto inicial y final.

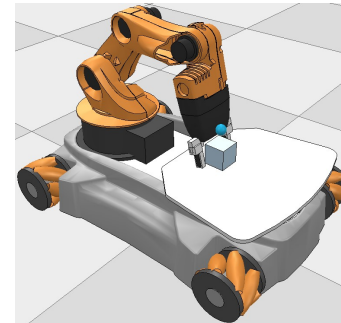
Aprovechando el modelo cinemático, se puede graficar la velocidad angular de los motores, como se observa en la Figura 6, la velocidad se reduce hasta 0, llegando así a la posición final.

#### B. Simulación CoppeliaSim

Se obtiene también, una simulación virtual en base al modelo cinemático y un robot con el cual se prueba el modelo. En CoppeliaSim se dispone del KUKA YouBot que viene por defecto, una plataforma con ruedas omnidireccionales.



(a) En la realidad



(b) En CoppeliaSm

Fig. 7: KUKA YouBot

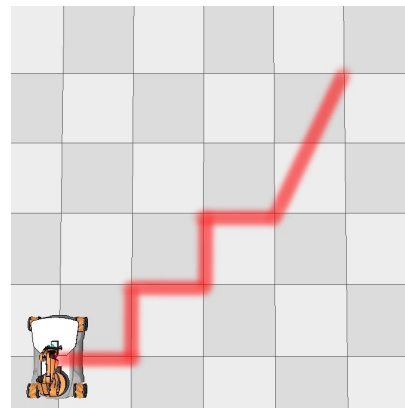


Fig. 8: Ruta programada para el KUKA YouBot y posición inicial

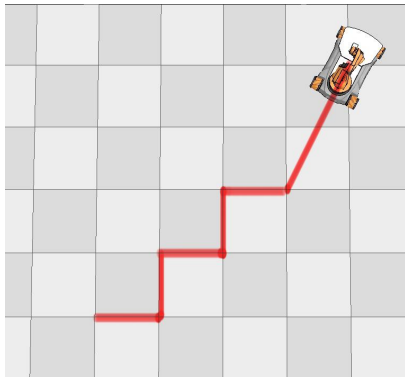


Fig. 9: Posición final KUKA YouBot

Se programó un desplazamiento como en la Figura 8 donde al final se incluye una rotación y se puede visualizar en la Figura 9.

Se conocen alternativas más cercanas al entorno real de implementación como Gazebo [4], que no se logró probar para esta iteración del semillero; se sabe que se puede complementar con ROS (Robot Operating System) y eso significa que se puede aprovechar para elementos o dispositivos electrónicos físicos.

### C. Revisión Odroid

Se instaló satisfactoriamente la distribución Ubuntu de Linux en sus versiones 16 y 18 sobre la tarjeta de desarrollo Odroid-XU4 presentando algunos problemas iniciales de apagado indeseado cuando se le cargaba al sistema operativo con más procesos, solucionable con una fuente de mejor calidad o también con un arreglo de diodo, capacitor y resistencia para evitar los cortes súbitos pero momentáneos de energía.

### D. Aprendizajes en ROS

En relación con el punto anterior, se logró instalar ROS (Robot Operating System) sobre Ubuntu 16 y correr algunos de sus ejemplos predeterminados como la TurtleBot, además, entender brevemente aunque sin profundizar, en la estructura de este sistema operativo.

### E. Cámara Orbbec Astra

Se realizaron pruebas con la cámara de profundidad Orbbec Astra concluyendo que es un excelente complemento para determinar el entorno tridimensional que rodea al robot de servicio, este dispositivo permite conocer la profundidad de cada punto de la imagen, el programa se corre en Linux y la cámara es USB.

### F. LiDAR

Se sabe de paquetes de ROS que son una gran alternativa para realizar mapeo con ROS como son:

- Google Cartographer
- Hector Slam

## IV. DISCUSIÓN O ANÁLISIS DE RESULTADOS

La simulación en CoppeliaSim no se pudo realizar más compleja porque los ángulos de rotación entregados por el API a Matlab no son cartesianos.

La velocidad angular de cada motor corresponde efectivamente con el movimiento que debería hacer el robot en la realidad.

El movimiento simulado en CoppeliaSim no tuvo ningún tipo de problema, además el fin de cada trayectoria se veía suavizado gracias al enfoque de campos potenciales [3].

El punto de inicio y el punto de final de una trayectoria definen un vector al cual el robot se intenta alinear si se incluye una medida de error entre la orientación del robot y el mencionado vector.

En la simulación no es posible determinar el ángulo final de robot, no se implementó.

La ruta aproximada para continuar con el proyecto es, en consecuencia con lo que ya se vió, formar un mapa virtual partiendo de los sensores como cámara de profundidad y LiDAR usando técnicas de SLAM con el fin de ubicar el robot en un espacio real de trabajo y prescindir del control desde Matlab para llevarlo a cabo desde la tarjeta de desarrollo; posteriormente, mentiras en algoritmos de control para los motores y así generar el movimiento deseado.

El penúltimo bloque se compone de la navegación segura para el robot, la fundamental para guardar su integridad y por último, gestión tareas de movimiento.

## V. CONCLUSIÓN

- La simulación con Matlab y CoppeliaSim en conjunto es muy útil, dado que el segundo tiene muchos robots, sensores y objetos para crear entornos de simulación, además se aprovecha la potencia de Matlab para la computación sin programar el childScript en CoppeliaSim que resulta ser poco práctico.
- Se validó de manera aceptable por medio de simulaciones que el modelo cinamático es funcional.
- Se pueden combinar la cámara de profundidad con el LiDAR para obtener una representación más fiel del entorno que rodea el robot.
- Matlab es un buen comienzo para planear el algoritmo de control y luego migrarlo a alternativas más útiles con la actual aplicación Robótica, usar ROS desde etapas tempranas si se tiene todo el hardware puede resultar provechoso.
- Las simulaciones son muy aceptables pero en la realidad también hay consideraciones mecánicas y eléctricas, los problemas realmente están en el hardware.
- Es muy importante elegir un algoritmo o ley de control adecuada para los motores de la forma correcta, el fallo en la actuación de estos elementos compromete todo el robot.
- Es muy crítico obtener la posición absoluta del robot en un entorno, esta variable de posición es entrada del modelo cinemático, es por esto que se piensa en cámara

de profundidad y/o LiDAR para tener un mapeo general donde el robot se pueda ubicar.

## REFERENCES

- [1] "Modelo Cinemático 4R."
- [2] "KUKA youBot." [Online]. Available: <https://www.kuka.com/>
- [3] V. Gonzalez-Villela, A. Sánchez-Balpuesta, and S.-A. Adan, "Cinemática de una plataforma móvil omnidireccional con llantas Mecanum en configuración AB," Ph.D. dissertation, Universidad Nacional Autónoma de México, Sep. 2015.
- [4] "Gazebo." [Online]. Available: <http://www.gazebosim.org/>