CEA Comité Español de Automática

XLVI JORNADAS AUTOMÁTICA CARTAGENA 2025

UNIVERSIDADE DA CORUÑA

Jornadas de Automática

Implementación de MPC para el seguimiento de trayectorias en un robot móvil omnidireccional

Villalba-Aguilera, Elena[a,b,*], Blesa, Joaquim[a,b,c,e], Ponsa, Pere[a,b,d]

[a]*Department of Automatic Control, Universitat Politècnica de Catalunya Barcelona Tech, 08019 Barcelona, Spain; elena.villalba@upc.edu (E.V.-A.); joaquim.blesa@upc.edu (J.B.); pedro.ponsa@upc.edu (P.P.)*
[b]*Safety and Automatic Control Research Center (CS2AC), Serra Hunter Fellow; 08222 Terrassa, Spain*
[c]*Institut de Robòtica i Informàtica Industrial, CSIC-UPC, 08028 Barcelona, Spain*
[d]*Intelligent Control Systems, 08019 Barcelona, Spain*
[e]*Department of Automatic Control, Serra Húnter Fellow, Universitat Politècnica de Catalunya Barcelona Tech, 08019, Barcelona, Spain*

To cite this article: Villalba-Aguilera, Elena, Blesa, Joaquim, Ponsa, Pere. 2025. Implementation of MPC for tra-jectory tracking in an omnidirectional mobile robot. Jornadas de Automática, 46. https://doi.org/10.17979/ja-cea.2025.46.12237

Resumen

Este trabajo presenta una arquitectura de control para un robot móvil omnidireccional de tres ruedas, validada experimentalmente en la plataforma Robotino 4. La solución evita el uso de ROS (*Robot Operating System*) al delegar el cálculo del controlador predictivo a una estación externa con MATLAB, la cual se comunica con el robot mediante TCP/IP (*Transmission Control Protocol/Internet Protocol*). El robot ejecuta un programa en C++ que recibe consignas de velocidad y envía la odometría a la estación de control. Se ha implementado un control LPV-MPC (*Linear Parameter Varying Model Predictive Control*) que regula simultáneamente la posición (x, y) y la orientación (θ), considerando restricciones sobre velocidad, aceleración y posición. El modelo se linealiza en tiempo real, lo que permite el seguimiento de trayectorias bajo condiciones realistas. La solución se ha validado mediante una trayectoria de referencia ovalada que muestra un buen rendimiento y errores acotados. La arquitectura es modular y escalable, y puede extenderse con planificación en entornos dinámicos o integración futura con ROS 2.

Palabras clave: Sistemas lineales de parámetros variables (LPV), Sistemas de control embebidos, Robots móviles, Control predictivo basado en modelo (MPC), Seguimiento de trayectorias.

Implementation of MPC for trajectory tracking in an omnidirectional mobile robot

Abstract

This work presents a control architecture for a three-wheeled omnidirectional mobile robot, experimentally validated on the Robotino 4 platform. The solution avoids the use of ROS (*Robot Operating System*) by delegating the computation of the predictive controller to an external MATLAB workstation, which communicates with the robot via TCP/IP (*Transmission Control Protocol/Internet Protocol*). The robot runs a C++ program that receives velocity commands and sends odometry data to the control station. An LPV-MPC (*Linear Parameter Varying Model Predictive Control*) scheme is implemented to simultaneously regulate the position of the robot (x, y) and orientation (θ), while explicitly considering constraints on velocity, acceleration and position. The model is linearized in real time, enabling trajectory tracking under realistic conditions. The solution has been validated using an oval reference trajectory, demonstrating good performance and bounded tracking errors. The proposed architecture is modular and scalable, and can be extended to include dynamic environment planning or future integration with ROS 2.

Keywords: Linear parameter varying (LPV), Embedded control systems, Mobile robots, Model predictive control (MPC), Trajectory tracking.

*Autor para correspondencia: elena.villalba@upc.edu

1. Introducción

Plataformas móviles holonómicas, como Robotino 4 de Festo (ver Figura 1), proporcionan una base accesible a nivel de hardware y software para implementar y validar estrategias avanzadas de control, tanto en investigación como en docencia (Villalba-Aguilera, 2024).

Figura 1: Robotino 4, robot móvil omnidireccional de tres ruedas.

En la mayoría de configuraciones estándar, Robotino es operado mediante entornos de desarrollo cerrados o a través de ROS (*Robot Operating System*) (Niemueller et al., 2013). Si bien este último ofrece gran versatilidad, su implementación conlleva una elevada complejidad técnica y computacional, especialmente al integrar nodos personalizados, gestionar sincronización entre las diferentes capas de las arquitecturas o ejecutar algoritmos avanzados en tiempo real. En este sentido, trabajos como (Mercorelli et al., 2017) han propuesto integrar MATLAB con ROS para aprovechar el modelado de alto nivel con la infraestructura de ROS. Sin embargo, esta aproximación sigue requiriendo una integración con el ecosistema ROS y depende de capacidades de cómputo a bordo, lo cual puede resultar inviable en entornos docentes o experimentales con recursos limitados.

Con el objetivo de superar estas limitaciones, este trabajo propone una arquitectura que permite ejecutar en tiempo real un controlador predictivo para plataformas holonómicas como Robotino, sin necesidad de utilizar ROS. La arquitectura se apoya en una comunicación TCP/IP (*Transmission Control Protocol/Internet Protocol*) entre la estación de control externa, que calcula las consignas de velocidad en MATLAB, y el programa en C++ ejecutado en el robot, encargado de aplicar dichas consignas y adquirir los datos de los sensores.

En este contexto, el *Model Predictive Control* (MPC) ha demostrado ser una estrategia de control eficaz para la regulación simultánea de posición y orientación de robots móviles omnidireccionales de tres ruedas, al manejar restricciones explícitas y anticipar la dinámica del sistema (Villalba-Aguilera et al., 2025). A diferencia de enfoques que fijan la orientación como constante (El-Sayyah et al., 2022) (Cáceres Flórez et al., 2018), este trabajo propone una estrategia que regula los tres estados del robot: posición (x, y) y orientación θ. Esta capacidad resulta crucial en tareas como navegación autónoma, logística de almacenes o interacción humano-robot, donde la orientación precisa permite al robot alinearse correctamente con estaciones de carga, estanterías o usuarios.

El controlador implementado es LPV-MPC (*Linear Parameter Varying Model Predictive Control*), formulado como un problema cuadrático con restricciones de velocidad, aceleración y posición. El problema se resuelve en MATLAB, obteniendo tiempos de cómputo adecuados para operación en tiempo real. La arquitectura se valida experimentalmente en Robotino 4, mediante el seguimiento de una trayectoria ovalada que permitió evaluar el desempeño del control en condiciones reales.

En las siguientes secciones se describe la arquitectura propuesta del sistema, se detallan los módulos desarrollados, se presentan los resultados experimentales, se discuten los hallazgos y, finalmente, se exponen las conclusiones.

2. Arquitectura del sistema de control

La Figura 2 presenta la arquitectura del sistema de control desarrollada para el robot móvil omnidireccional de tres ruedas Robotino 4. Esta se organiza en tres bloques: *planificador de trayectoria*, *control de movimiento* (subdividido en *control de alto nivel* y *control de bajo nivel*), y *hardware*.

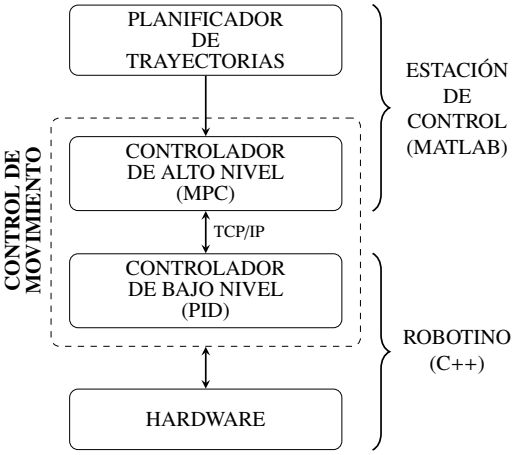

Figura 2: Arquitectura del sistema de control.

Cada uno de estos bloques se ejecuta en uno de los dos componentes físicos que conforman el sistema: la estación de control, un ordenador externo encargado de la planificación de la trayectoria y el cálculo de las consignas de control mediante el uso MATLAB; y el propio Robotino, que ejecuta las consignas y adquiere los datos de los sensores para la retroalimentación, mediante un programa en C++ desarrollado.

La comunicación entre ambos dispositivos se realiza mediante TCP/IP sobre Ethernet, permitiendo el intercambio bidireccional de datos en tiempo real. De este modo, el sistema opera en un bucle cerrado, donde se intercambian constantemente consignas de control y datos de estado del robot.

Cada ciclo de control sigue la siguiente secuencia.

1. El programa en MATLAB genera las referencias de trayectoria y calcula las velocidades óptimas en coordenadas locales del robot (v_x, v_y, ω) mediante el controlador MPC.
2. Estas consignas se codifican en formato binario y se envían a Robotino mediante la conexión en red y el protocolo Ethernet/IP.
3. El programa en C++, ejecutado en Robotino, interpreta los comandos y los aplica a través de la API nativa proporcionada por Festo.

4. Paralelamente, el mismo programa lee la odometría estimada del robot (x, y, θ).
5. Los datos de estado se envían a MATLAB, que actualiza la estimación de estado y repite el ciclo de control.

Esta arquitectura ha sido diseñada con un enfoque modular, lo que permite modificar, reemplazar o añadir nuevos componentes al sistema sin alterar el funcionamiento global. Por ejemplo, pueden integrarse módulos adicionales para diagnóstico o tolerancia a fallos.

Cabe destacar que el software ha sido desarrollado independientemente a ROS. Esto reduce las dependencias externas, mejora la eficiencia en tiempo real y facilita la depuración. Aun así, la arquitectura está preparada para una futura migración a ROS 2.

En las siguientes secciones se describe en detalle los módulos desarrollados en la estación de control, el módulo embebido en Robotino, y el protocolo de comunicación implementado para la sincronización entre ambos.

2.1. Estación de Control

La estación de control ejecuta la aplicación desarrollada en MATLAB. Esta integra dos componentes de la arquitectura definida: el *planificador de trayectorias* y el *controlador de alto nivel*.

El planificador permite definir la trayectoria de referencia que debe seguir el robot. Esta trayectoria se genera de forma discreta y proporciona, en cada instante de tiempo, una referencia temporal para las variables del espacio de estado: x, y y θ. Esta secuencia de referencias se utiliza como entrada para el *controlador de alto nivel*.

El *controlador de alto nivel* se basa en un enfoque MPC con formulación LPV. Su objetivo es calcular las consignas de velocidad en coordenadas locales (v_x, v_y y ω) que minimice el error de seguimiento, respetando al mismo tiempo las restricciones físicas del robot, como los límites de velocidad y aceleración. Este enfoque modela la dinámica del robot como un sistema lineal con parámetros variables, donde la orientación θ se considera como un parámetro exógeno que se actualiza en tiempo real.

Las consignas de velocidad resultantes se codifican y se envían a Robotino mediante la conexión TCP/IP.

2.2. Robotino

El módulo de ejecución embebido en Robotino 4 ha sido desarrollado por los autores en C++ y se encuentra disponible en (Villalba-Aguilera, 2025). Este se ejecuta sobre la distribución oficial del sistema operativo del robot, basada en Ubuntu 20.04 LTS con el siguiente conjunto de paquetes, necesarios para la abstracción del hardware:

- **robotino-daemon**: conjunto de servicios que gestionan la comunicación con los sensores y actuadores del robot.
- **robotino-dev**: biblioteca para la interacción entre los diferentes componentes internos del robot.
- **rec-rpc**: biblioteca para la comunicación remota entre procesos.
- **robotino-api2**: interfaz de programación en C/C++ que proporciona acceso directo al control de movimiento y a las lecturas de sensores.

El programa desarrollado cumple dos funciones: (i) recibir las consignas de velocidad generadas por la estación de control y aplicarlas a los actuadores del robot, y (ii) leer la odometría estimada del robot y enviarla de vuelta a la estación de control.

2.3. Gestión de la comunicación

La comunicación entre la estación de control y Robotino se implementa mediante sockets TCP/IP, donde el programa en C++ ejecutado en Robotino actúa como servidor, y MATLAB opera como cliente.

El sistema es bidireccional y se integra dentro del ciclo de control en tiempo real. En cada iteración, la estación de control genera y envía un paquete binario que contiene las consignas de velocidad lineal y angular en coordenadas locales del robot (v_x, v_y, ω), codificadas en formato `float32`.

El programa en Robotino recibe este paquete, lo decodifica y aplica los valores de velocidad al sistema de actuadores mediante la API nativa. Posteriormente, obtiene una estimación de su estado actual (x, y, θ) a través de su sistema de fusión sensorial, empaqueta esta información y la envía de vuelta a MATLAB, donde se actualiza el modelo del estado y se calcula la siguiente acción de control, cerrando así el lazo de control.

Para asegurar una sincronización estable entre el envío y la recepción de datos, se han incorporado mecanismos de control temporal y gestión de errores, entre los que se incluyen: la detección automática de pérdida de conexión, la monitorización de la latencia de transmisión y la verificación de integridad de los paquetes de datos recibidos.

3. Implementación

El sistema de control propuesto ha sido validado experimentalmente mediante la ejecución de una trayectoria de referencia en forma de óvalo con la plataforma Robotino 4 en un entorno controlado en laboratorio.

3.1. Estación de control

El programa de la estación de control ha sido desarrollado en MATLAB 2024b e integra tanto el módulo de *planificación de trayectorias* como el *controlador de alto nivel*.

La trayectoria de referencia se generó paramétricamente mediante las siguientes ecuaciones:

$$x_{ref}(t) = r_x \cdot \cos\left(\frac{2\pi}{T} \cdot t\right) + c_x \tag{1}$$

$$y_{ref}(t) = r_y \cdot \sin\left(\frac{2\pi}{T} \cdot t\right) + c_y \tag{2}$$

La orientación de referencia $\theta_{ref}(t)$, se obtuvo a partir del cálculo de la derivada de la trayectoria, mediante:

$$\theta_{ref}(t) = \arctan(\dot{y}_{ref}(t), \dot{x}_{ref}(t)) \tag{3}$$

$$\dot{x}_{ref}(t) = -\left(\frac{2\pi}{T}\right) \cdot r_x \cdot \sin\left(\frac{2\pi}{T} \cdot t\right) \tag{4}$$

$$\dot{y}_{ref}(t) = \left(\frac{2\pi}{T}\right) \cdot r_y \cdot \cos\left(\frac{2\pi}{T} \cdot t\right) \tag{5}$$

donde $T = 110$ s es el tiempo que el robot tarda en completar una vuelta, $r_x = 2$ m y $r_y = 0{,}8$ m corresponden a los radios horizontal y vertical, respectivamente, y $c_x = 2{,}4$ m, $c_y = 1$ m definen el centro del óvalo.

El controlador se formuló como un problema de optimización basado en MPC (Villalba-Aguilera et al., 2025), cuya expresión es:

$$\begin{aligned} &\min_{\boldsymbol{u}(i|k)} \sum_{i=1}^{H_p} \sum_{j=1}^{2} J_j(i|k) \\ &\text{s.t.} \\ &\quad \boldsymbol{x}(i+1|k) = \boldsymbol{x}(i|k) + B(\hat{\theta}(i|k))\boldsymbol{u}(i|k) \\ &\quad \boldsymbol{u}^{min} \le \boldsymbol{u}(i|k) \le \boldsymbol{u}^{max} \\ &\quad \Delta\boldsymbol{u}^{min} \le \Delta\boldsymbol{u}(i|k) \le \Delta\boldsymbol{u}^{max} \\ &\quad \boldsymbol{x}^{min} \le \boldsymbol{x}(i+1|k) \le \boldsymbol{x}^{max} \end{aligned} \tag{6}$$

donde el vector de estado es $\boldsymbol{x}(k) = [x(k) \quad y(k) \quad \theta(k)]^T$, mientras que el vector de control es $\boldsymbol{u}(k) = [v_x(k) \quad v_y(k) \quad \omega(k)]^T$. Las funciones de coste se definen como:

$$\begin{aligned} J_1(i|k) &= \left(\boldsymbol{x}(i|k) - \boldsymbol{x}_{ref}(k)\right)^T Q \left(\boldsymbol{x}(i|k) - \boldsymbol{x}_{ref}(k)\right) \\ J_2(i|k) &= \boldsymbol{u}(i|k)^T P \boldsymbol{u}(i|k) \end{aligned} \tag{7}$$

donde las matrices Q y P ponderan el error de seguimiento y el esfuerzo de control, respectivamente. Las variables $\boldsymbol{u}^{\text{mín}}$, $\boldsymbol{u}^{\text{máx}}$ y sus derivadas definen los límites físicos de velocidad y aceleración, mientras que $\boldsymbol{x}^{\text{mín}}$ y $\boldsymbol{x}^{\text{máx}}$ restringen la posición del robot dentro del área operativa. La matriz $B(\hat{\theta}(i|k))$ se define como:

$$B(\hat{\theta}(i|k)) = T_s R(\hat{\theta}(i|k)) \tag{8}$$

donde T_s es el tiempo de muestreo, $\hat{\theta}(i|k)$ es una estimación exógena de la evolución de orientación del robot θ y la matriz de rotación $R(\theta)$ transforma las velocidades desde el marco local al global, y se define como:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = R(\theta) \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{9}$$

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{10}$$

El modelo LPV permite aproximar sistemas no lineales mediante modelos lineales cuyos parámetros varían con el tiempo. En este caso, la no linealidad del modelo cinemático se concentra en la orientación θ, de la cual se predice su evolución en los instantes futuros $\hat{\theta}(i|k) \quad i = 1, ..., H_p$ en el instante k considerando el error actual en la orientación y las consignas para los valores futuros. Esta predicción se obtiene a partir del ángulo de referencia. Esto permite usar un modelo que, aunque cambia con el estado del sistema, conserva la estructura lineal en cada instante. Gracias a ello, el problema de control se puede resolver como una optimización cuadrática (QP), lo cual es computacionalmente eficiente y adecuado para implementaciones en tiempo real. En otras palabras, el controlador se adapta dinámicamente a la orientación actual del robot, predice su evolución futura y calcula las entradas óptimas que minimizan el error de seguimiento, respetando las restricciones físicas del sistema. Este enfoque ya ha sido validado en aplicaciones móviles como se discute en Alcalá et al. (2020); Atoui et al. (2022); Fényes et al. (2020).

El problema se implementó en MATLAB utilizando la toolbox YALMIP. Las restricciones físicas se modelaron como cotas explícitas dentro del esquema de optimización cuadrática. Para la resolución del problema se utilizó el solver Gurobi.

El controlador LPV-MPC se configuró con un horizonte de predicción $H_p = 5$ y un tiempo de muestreo $T_s = 0{,}5$ s. Las restricciones del sistema fueron:

- Límites de velocidad:

$$\boldsymbol{u}^{\text{mín}} = -\begin{bmatrix} 0{,}3 \text{ m/s} \\ 0{,}3 \text{ m/s} \\ 0{,}4 \text{ rad/s} \end{bmatrix}, \quad \boldsymbol{u}^{\text{máx}} = \begin{bmatrix} 0{,}3 \text{ m/s} \\ 0{,}3 \text{ m/s} \\ 0{,}4 \text{ rad/s} \end{bmatrix} \tag{11}$$

- Límites de aceleración:

$$\Delta\boldsymbol{u}^{\text{mín}} = -\begin{bmatrix} 0{,}15 \text{ m/s}^2 \\ 0{,}15 \text{ m/s}^2 \\ 0{,}2 \text{ rad/s}^2 \end{bmatrix}, \quad \Delta\boldsymbol{u}^{\text{máx}} = \begin{bmatrix} 0{,}15 \text{ m/s}^2 \\ 0{,}15 \text{ m/s}^2 \\ 0{,}2 \text{ rad/s}^2 \end{bmatrix} \tag{12}$$

- Límites de posición:

$$\boldsymbol{x}^{\text{mín}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ m}, \quad \boldsymbol{x}^{\text{máx}} = \begin{bmatrix} 5 \\ 2 \end{bmatrix} \text{ m} \tag{13}$$

- Matrices de pesos:

$$Q = \text{diag}(1, 1, 1), \quad P = \text{diag}(1, 1, 1) \tag{14}$$

- Horizonte de predicción: $H_p = 5$
- Estado inicial: $\boldsymbol{x}_0 = \begin{bmatrix} 2{,}4 \text{ m} & 1 \text{ m} & 0 \text{ rad} \end{bmatrix}'$

El programa se ejecutó en un portátil con procesador Intel Core i7 y 16 GB de RAM, obteniendo tiempos de cómputo promedio de 20 ms por iteración. Esto permite operar en tiempo real con márgenes suficientes respecto al periodo de muestreo $T_s = 0{,}5$ s.

3.2. *Robotino*

El ordenador a bordo de Robotino incorpora un procesador Intel Core i5 de 8ª generación (2.5 GHz) y 8 GB de RAM, encargado de ejecutar el módulo en C++ que aplica las consignas de control y mantiene la comunicación con la estación externa.

Para aplicar las velocidades deseadas en el marco local del robot, se utiliza la clase `OmniDrive` de la API `robotino-api2`, mediante la función `setVelocity(`v_x`,` v_y`,` ω`)`. Internamente, el software nativo de Robotino convierte estas velocidades en órdenes individuales para cada rueda, las cuales son reguladas por controladores PID integrados, preconfigurados por el fabricante (Festo) Villalba-Aguilera et al. (2025).

La lectura de la odometría se realiza mediante la clase `Odometry`, cuya función `readings` proporciona la posición y orientación actuales del robot en el marco global (x, y, θ). Esta estimación se obtiene mediante la fusión de datos provenientes de los encoders incrementales de las ruedas y del giróscopo. Cabe destacar que el algoritmo de fusión sensorial es implementado y gestionado internamente por el fabricante.

El programa desarrollado en C++ opera en un entorno multihilo, utilizando las bibliotecas estándar (`std::thread`, `std::mutex`), permitiendo la ejecución concurrente de los siguientes procesos:

- Recepción de las consignas de velocidad en formato binario mediante la conexión TCP/IP.
- Aplicación de las consignas al sistema de locomoción mediante la función `setVelocity(vx, vy, ω)`.
- Lectura periódica de la odometría estimada mediante la función `readings`.
- Envío periódico del estado actual del robot a la estación de control para retroalimentar el bucle de control.

4. Resultados

La validación experimental del sistema de control se llevó a cabo en un entorno real utilizando la plataforma Robotino 4, siguiendo la trayectoria de tipo oval descrita en el apartado anterior. Las pruebas permitieron evaluar la capacidad del controlador LPV-MPC para seguir referencias suaves bajo restricciones físicas reales de velocidad, aceleración y espacio operativo.

En las siguientes figuras se presentan diferentes aspectos del comportamiento del sistema durante la ejecución de la trayectoria. La Figura 3 muestra el seguimiento espacial de la trayectoria de referencia en forma de óvalo. La Figura 4 presenta la evolución temporal de los estados x, y y θ, comparando la referencia con la respuesta real del sistema. Por último, la Figura 5 muestra el error de seguimiento en posición y orientación a lo largo del tiempo.

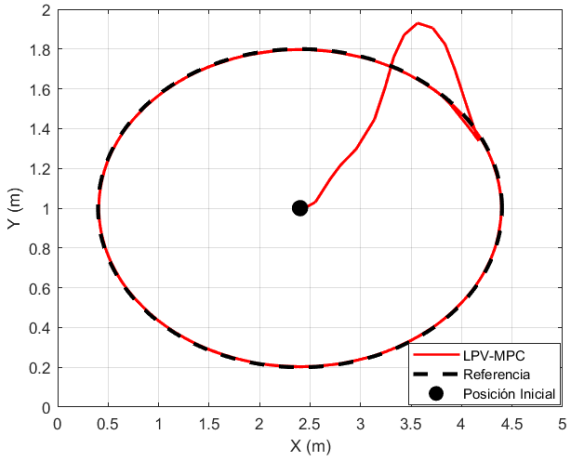

Figura 3: Seguimiento de la trayectoria en forma de óvalo. La línea discontinua negra representa la trayectoria de referencia, mientras que la línea roja muestra la trayectoria seguida por el robot mediante el controlador LPV-MPC. El punto negro indica la posición inicial del robot.

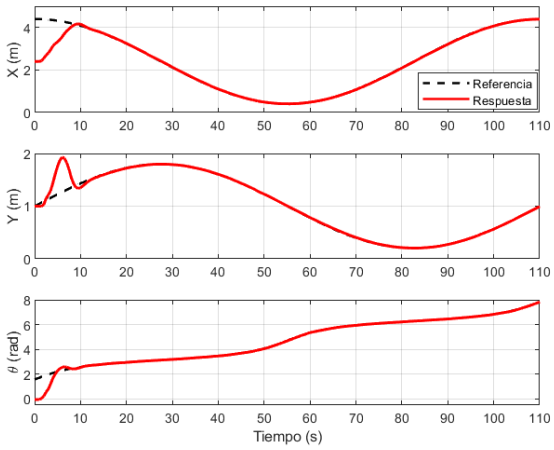

Figura 4: Evolución temporal de los estados x, y y θ. Las líneas discontinuas negras indican las señales de referencia, y las líneas rojas corresponden a la respuesta del sistema con control LPV-MPC.

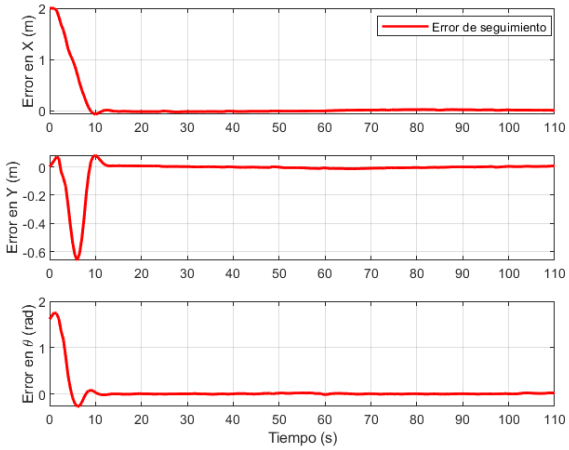

Figura 5: Errores de seguimiento en x, y y θ a lo largo del tiempo.

Para cuantificar el rendimiento del seguimiento, se ha calculado el error cuadrático medio (MSE) para cada una de las variables de estado durante la ejecución experimental. Los resultados se presentan en la Tabla 1.

Tabla 1: Error cuadrático medio (MSE) para cada variable de estado en condiciones reales

Variable de estado	MSE
x (m)	$4{,}85 \times 10^{-2}$
y (m)	$3{,}40 \times 10^{-3}$
θ (rad)	$2{,}81 \times 10^{-2}$

5. Discusión

Los resultados experimentales obtenidos validan la efectividad del sistema de control propuesto en condiciones reales. Como se observa en las figuras presentadas, el robot es capaz de seguir con precisión la trayectoria de referencia, manteniendo los errores de seguimiento acotados durante la mayor parte del recorrido.

Los mayores errores se producen durante el régimen transitorio, es decir, en los primeros segundos de la trayectoria. Esto se debe a que el robot parte desde una posición inicial

fuera de la trayectoria de referencia, en concreto, desde el centro del óvalo, lo que implica una fase inicial de convergencia. A ello se suman las limitaciones impuestas en velocidad y aceleración, que restringen la capacidad del controlador para realizar correcciones rápidas. Sin embargo, una vez superado este régimen transitorio, el error disminuye progresivamente hasta estabilizarse en valores bajos y consistentes, lo que evidencia la capacidad del sistema para recuperar y mantener el seguimiento con precisión.

En lo que respecta a la capa de *controlador de alto nivel* de la arquitectura presentada, el uso del MPC presenta múltiples ventajas. En primer lugar, permite incorporar de forma explícita las limitaciones físicas del sistema, tanto en velocidad como en aceleración. Esta característica es fundamental para garantizar que el comportamiento del sistema no comprometa la seguridad ni exceda las capacidades dinámicas del hardware.

Por otra parte, la adopción del LPV facilita una implementación eficiente del esquema MPC original. Al considerar la orientación del robot como un parámetro exógeno y realizar una linearización en cada instante de muestreo, se logra un buen equilibrio entre precisión de control y coste computacional. Esta aproximación es especialmente útil en aplicaciones embebidas en tiempo real, como la ejecutada en el sistema Robotino, donde los recursos de procesamiento son limitados y es necesario mantener tiempos de respuesta consistentes.

Adicionalmente, este enfoque permite trabajar con trayectorias en las que la orientación varía a lo largo del tiempo. Esta capacidad mejora el desempeño del seguimiento frente a métodos que asumen una orientación constante.

6. Conclusiones

En este trabajo se ha presentado una arquitectura de control modular y distribuida para un robot móvil omnidireccional, validada experimentalmente sobre la plataforma Robotino 4. El sistema combina planificación de trayectorias, control predictivo en alto nivel y control PID en bajo nivel, implementados respectivamente en MATLAB y C++, y conectados mediante comunicación TCP/IP en tiempo real.

La implementación del controlador LPV-MPC ha demostrado ser efectiva para el seguimiento de trayectorias bajo restricciones físicas, manteniendo errores acotados incluso en presencia de condiciones iniciales desfavorables. El uso de una formulación LPV ha permitido reducir la carga computacional manteniendo un buen rendimiento de control, lo cual es especialmente adecuado para sistemas embebidos con recursos limitados. Los resultados experimentales han validado tanto la estrategia de control como la arquitectura de software planteada.

Como líneas de trabajo futuro, se plantean varias extensiones para aumentar la robustez y autonomía del sistema:

- **Detección y evitación de obstáculos**: se incorporarán sensores externos para identificar obstáculos en tiempo real, integrando esta información como restricciones dinámicas dentro del problema de optimización del MPC.
- **Diagnóstico y tolerancia a fallos**: se desarrollarán módulos específicos para el diagnóstico de fallos y la aplicación de estrategias de tolerancia, con especial atención a fallos en actuadores, como pérdidas de funcionalidad en motores. La arquitectura MPC permitirá reconfigurar el controlador en tiempo real mediante la actualización del modelo y/o las restricciones, permitiendo que el robot mantenga su misión de forma degradada o, en caso crítico, transfiera la tarea a otro robot de la flota.
- **Migración a entornos ROS 2 y Gazebo**: con el objetivo de mejorar la escalabilidad del sistema, se plantea portar la arquitectura actual a ROS 2, así como desarrollar escenarios de simulación realistas en Gazebo, permitiendo una evaluación previa más completa antes de las pruebas físicas.

Agradecimientos

La autora principal agradece sinceramente el apoyo financiero proporcionado por la Universitat Politècnica de Catalunya y Banco Santander a través de la beca predoctoral FPI-UPC.

Este trabajo ha sido financiado parcialmente por la Agencia Española de Investigación (AEI) mediante el proyecto SaCoAV (ref.MINECOPID2020-114244RB-I00).

Referencias

Alcalá, E., Puig, V., Quevedo, J., Sename, O., 2020. Fast zonotope-tube-based lpv-mpc for autonomous vehicles. IET Control Theory & Applications 14 (20), 3676–3685.
URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cta.2020.0562
DOI: https://doi.org/10.1049/iet-cta.2020.0562

Atoui, H., Sename, O., Milanés, V., Martinez, J. J., 2022. Lpv-based autonomous vehicle lateral controllers: A comparative analysis. IEEE Transactions on Intelligent Transportation Systems 23 (8), 13570–13581.
DOI: 10.1109/TITS.2021.3125771

Cáceres Flórez, C., Rosario, J., Amaya, D., 04 2018. Design, simulation, and control of an omnidirectional mobile robot. International Review of Mechanical Engineering (IREME) 12, 382.
DOI: 10.15866/ireme.v12i4.13974

El-Sayyah, M., Saad, M., Saad, M., 01 2022. Enhanced mpc for omnidirectional robot motion tracking using laguerre functions and non-iterative linearization. IEEE Access PP, 1–1.
DOI: 10.1109/ACCESS.2022.3220240

Fényes, D., Hegedűs, T., Németh, B., Gáspár, P., Koenig, D., Sename, O., 2020. Lpv control for autonomous vehicles using a machine learning-based tire pressure estimation. In: 2020 28th Mediterranean Conference on Control and Automation (MED). pp. 212–217.
DOI: 10.1109/MED48518.2020.9183106

Mercorelli, P., Voss, T., Strassberger, D., Sergiyenko, O., Lindner, L., 2017. Optimal trajectory generation using mpc in robotino and its implementation with ros system. In: 2017 IEEE 26th International Symposium on Industrial Electronics (ISIE). pp. 1642–1647.
DOI: 10.1109/ISIE.2017.8001493

Niemueller, T., Lakemeyer, G., Ferrein, A., mar 2013. Incremental task-level reasoning in a competitive factory automation scenario. In: AAAI Spring Symposium 2013 on Designing Intelligent Robots: Reintegrating AI II. Stanford University, CA, USA.

Villalba-Aguilera, E., 2024. Creación de escenarios para la navegación de robot móvil. Final Master Report, ETSEIB, Universitat Politècnica de Catalunya.
URL: https://upcommons.upc.edu/handle/2117/414014

Villalba-Aguilera, E., 2025. Programa c++ para robotino. https://github.com/elena-villalba/robotino-lowlevel-interface, accedido el 29 de mayo de 2025.

Villalba-Aguilera, E., Blesa, J., Ponsa, P., 2025. Model-based predictive control for position and orientation tracking in a multilayer architecture for a three-wheeled omnidirectional mobile robot. Robotics 14 (6).
URL: https://www.mdpi.com/2218-6581/14/6/72
DOI: 10.3390/robotics14060072