UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño e implementación del controlador punto a punto del sistema Robotat por medio de visión de computadora y fusión de sensores

Trabajo de graduación presentado por José Eduardo Molina Calzia para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2020

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño e implementación del controlador punto a punto del sistema Robotat por medio de visión de computadora y fusión de sensores

Trabajo de graduación presentado por José Eduardo Molina Calzia para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2020

Vo.Bo.:

(f) _____

MSc. Miguel Enrique Zea Arenales

Tribunal Examinador:

(f) _____

MSc. Miguel Enrique Zea Arenales

(f) _____

MSc. Pablo Oliva

(f) _____

Ing. Pablo Mazariegos

Fecha de aprobación: Guatemala, 18 de enero de 2021.

Prefacio

En primer lugar le agradezco a mis padres, Rita y Juan José, quienes no solo son mi mayor ejemplo a seguir en todo aspecto, sino también son los principales promotores de mis sueños y metas. Sin su apoyo no estaría en donde me encuentro actualmente. Agradezco también el apoyo recibido por mi hermana, Calu, quien me ha apoyado con amor y paciencia en cada momento de mi vida. Agradezco también el apoyo recibido por mis queridos amigos, sus palabras de ánimo y apoyo incondicional han sido fundamentales en mi vida, específicamente a lo largo de la carrera y en la realización de este trabajo. A ellos, mi familia y amigos más cercanos, les dedico los logros de mi carrera y este trabajo.

Agradezco especialmente a mi asesor, Miguel Zea, por el apoyo y el consejo, quien a pesar de las circunstancias me exigió a trabajar con calidad y excelencia. Gracias a lo anterior este trabajo alcanzó los objetivos planteados. Por último quiero desear éxitos a mis compañeros y amigos de la carrera, quienes también están próximos a culminar sus estudios y trabajos de graduación.

Índice

Lista de figuras

Lista de cuadros

Resumen

Este trabajo se desarrolló como la segunda fase del proyecto Robotat-Bitbot. Este consistió en el desarrollo de las capacidades de sensado y los controladores de pose para los robots diferenciales, Bitbot. Este proyecto se estructuró en tres partes principales. Cada parte fue complementaria a la siguiente, teniendo como resultado general el desarrollo y comparación de una metodología de control y sensado de pose de los robots. Para todas las partes del proyecto se realizó un experimento, los cuales responden directamente a los objetivos específicos.

La primera parte consistió en el desarrollo e implementación de un sistema de visión por computadora. Este sistema, al igual que la fase pasada del proyecto, se ejecutó externamente a cada robot. A partir de este algoritmo se obtuvo las varianzas utilizadas en la parte de estimación de pose. En esta segunda parte del proyecto se desarrolló exitosamente un algoritmo de odometría y fusión de sensores. Para esta parte se evaluó la eficacia del algoritmo implementado en comparación con las mediciones de la cámara, la cual se utilizó como un GPS, y la posición real del robot.

Por último se realizó la prueba de controladores punto a punto para los robots. Para esta parte del proyecto se utilizó la estimación de pose desarrollada en la sección anterior. Se desarrolló un experimento para evaluar la eficacia de distintos esquemas de control en distintas situaciones. También se escogió un esquema de control óptimo para las necesidades de los Bitbots. Cabe resaltar que todas las pruebas de odometría y control se realizaron por medio del simulador de código abierto de robótica Webots.

Abstract

This work was developed as the second phase of the Robotat-Bitbot project. It consisted in the development of the sensing capacities and the pose controllers for the differential robots, Bitbot. This project was structured in three main parts. Each part was complementary to the next, having as a general result the development and comparison of a control and posture-sensing methodology for the robots. An experiment was carried out for all the parts of the project, which directly respond to the specific objectives.

The first part of the project consisted of the development and implementation of a computer vision system. This system was executed externally to each robot. The variances used in the pose estimation part were obtained from this algorithm. An odometry and sensor fusion algorithm was successfully developed at the second phase of the project. The efficiency of the implemented algorithm was evaluated in comparison with the measurements of the camera, which was used as a GPS, and the real position of the robot.

Finally, the point-to-point controller test for the robots was carried out. For this part of the project the pose estimation developed in the previous section was used. An experiment was developed to evaluate the effectiveness of different control schemes in different situations. An optimal control scheme was also chosen for the needs of the Bitbots. It should be noted that all the odometry and control tests were carried out using the open-source Webots robotics simulator.

CAPÍTULO 1

Introducción

El proyecto Bitbots y Robotat busca fabricar una mesa de pruebas y micro-robots para la implementación y evaluación de robótica en enjambre. La primera fase de este, sin embargo, presentó problemas de eficiencia y cadencia en la transmisión de datos desde el servidor hacia los micro-robots. Según las recomendaciones de la fase anterior se implementará un sistema diferente, pero manteniendo las bases de lo trabajado. Se trabajará con el mismo equipo que la fase pasada del proyecto, es decir, no se cambiará la cámara ni la mesa de pruebas.

En la segunda fase del proyecto se implementa el algoritmo de visión por computadora por medio de Python y se evita el uso de un servidor intermedio entre el envío de datos y la captura de imágenes. También se utiliza un microcontrolador más potente, pues el control de posición se realiza localmente en el microcontrolador. Adicionalmente a esto, se aproxima su posición y orientación utilizando sensores de posición angular en cada rueda de los micro-robots. Se plantean varias opciones para el control de posición de los robots y se elegirá la que mejor se adapte al modelo y capacidades de estos. La implementación de la visión por computadora se realiza, al igual que la fase pasada, por medio de la librería de código abierto OpenCV. La diferencia radica en que esta fase busca la implementación de multiprocesamiento mediante Python.

Posterior a la implementación del algoritmo de visión por computadora, el controlador y la obtención de pose, se trabajará en conjunto con los estudiantes José Andrés Castañeda Forno y Pablo Roberto Prado Cruz para desarrollar la plataforma de pruebas con los micro-robots y el software necesario. La implementación de la continuación del proyecto Robotat y Bitbots permitirá realizar pruebas de algorítmos de robótica en enjambre dentro del campus de la Universidad. También permitirá realizar investigación en el área de robótica móvil sin incurrir en costos tan elevados a comparación de otras mesas de pruebas similares.

CAPÍTULO 2

Antecedentes

2.1. Robotarium

El Robotarium es una plataforma accesible de forma remota, creada por estudiantes de Georgia Institute of Technology en el año 2016. Su objetivo principal es resolver los problemas principales de los entornos de investigación y desarrollo para robots en enjambre. Los problemas que resuelve son: caros de desarrollar, complejos de operar, muy exigentes de tiempo en cuanto a su operación, desarrollo y mantenimiento. Todo esto se resuelve al proveeer acceso remoto a los estudiantes para que puedan probar sus diseños desde una plataforma simulada y luego en la plataforma física. [1]

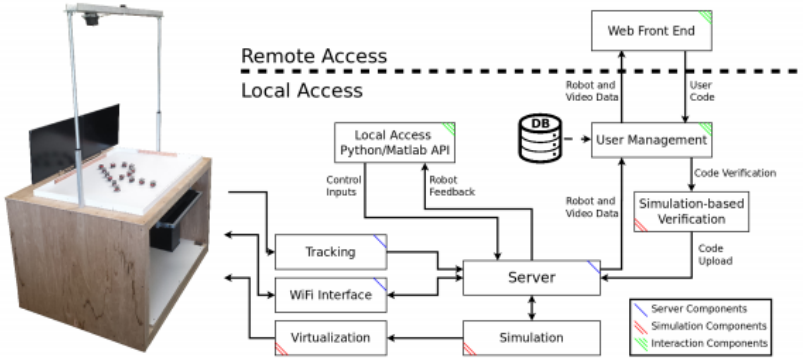


Figura 1: Diagrama general del Robotarium [2].

2.2. Fase I de proyecto de Robotat y BitBots

En la fase anterior del protecto se generó un algoritmo de reconocimiento de agentes en el banco de pruebas Robotat. Esto fue logrado al utilizar identificadores planares identificados

por una webcam y un algoritmo realizado en C++. Para la comunicación con los agentes se realizó un sistema WiFi con los módulos ESP-01. Para el reconocimiento de los agentes se utilizó un código *QR: personalizado*. Esta fase del proyecto presentó, sin embargo, problemas de rendimiento. Algunos de los problemas se deben a la forma en que enviaban los datos a cada agente, pues de primero se cargaban los datos de cada agente a una base de datos en un servidor y desde la computadora del servidor se enviaban los datos. Esto provocaba una latencia mayor en el envío de datos. C++ tiende a ser un lenguaje más eficiente que Python, sin embargo por medio de Pyhton se pueden implementar algoritmos por medio de multiprocesamiento más fácilmente. [3][4]

Los problemas de rendimiento de la primera fase del proyecto Robotat se derivan de la visión por computador, pues el algoritmo implementado y en general el análisis de la imagen en tiempo real tienden a ser computacionalmente pesados. Esto significa que se producía un *cuello de botella* en el flujo de datos. Se tenía contemplada una frecuencia de muestreo de 10Hz, sin embargo esta condición se satisfacía solamente para 3 agentes sobre la mesa. Otro factor limitante para el rendimiento del proyecto fue la implementación de una base de datos intermedia. Si bien es cierto esto facilita y aisla el control del sistema de visión por computadora, la base de datos intermedia suma al tiempo de ejecución de estos. Los microcontroladores que se utilizaron fueron los *PIC16F887* de Microchip, estos son microcontroladores de 8 bits de gama baja. Los micronoctroladores no realizaban ningún tipo de control local, todo el control de pocisión se realizaba en el servidor y era enviado al microcontrolador para que este solo ejecute. [3][4]

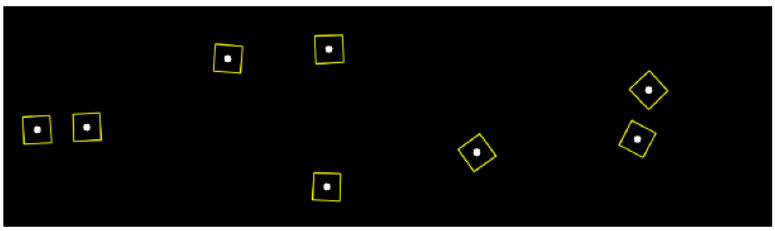


Figura 2: Resultados de identificación de agentes [3].

2.3. Zooids

Los Zooids consisten en una interface humano-cumputador por medio de robótica en enjambre. El objetivo principal de los Zooids es crear una interface de usuario de mesa por medio de micro-robots. La plataforma consiste en varios micro-robots guiados por un sistema óptico de álta velocidad. Los Zooids contribuyen a acercar más a la tecnología el concepto de una habitación en donde la materia está controlada por una computadora.[5]
Las limitaciones que presenta la plataforma son:

- Rastreo de robots solamente dentro de un área definida.
- Resolución.
- Escalabilidad.

- Potencia de alimentación, decrece entre más agentes existan.

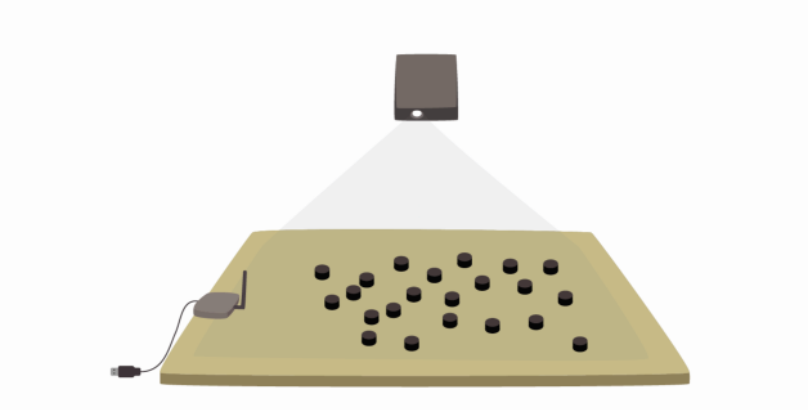


Figura 3: Diagrama general del sistema Zooids [5].

CAPÍTULO 3

Justificación

Actualmente en Latinoamérica no existe una mesa de pruebas multirobot dedicada al estudio de robótica en enjambre. Estos sistemas tienden a ser muy complejos y caros, por lo que no son replicables ni fáciles de adquirir. La necesidad de una mesa de pruebas multirobot radica en su importancia para la investigación en el área de robótica en ejambre. Por lo tanto, se propone una mesa de pruebas con sensado de posición global y local de cada robot. El sistema propuesto consta de una mesa física, una cámara, una lámpara y una computadora que servirá como servidor y será en donde estará alojado todo el software necesario. Los robots propuestos tendrán sensores de posición, más no tendrán capacidad de sensado a distancia como aquellos implementados en otros centros de investigación; esto minimiza costos y da la oportunidad al sistema a tener un rendimiento mayor.

La mesa de pruebas forma parte de la segunda fase del proyecto **Robotat** y **BitBots**. En la primera fase se lograron avances considerables, sin embargo no se cumplieron todos los objetivos planteados. En la fase actual del proyecto se optimizará lo realizado en la fase anterior, también se agregarán nuevas funcionalidades como una plataforma de simulación en el software *Webots*. Así mismo se utilizarán microcontroladores más potentes para realizar un mejor seguimiento de trayectorias.

La robótica en enjambre es un área relativamente nueva si se le compara con los estudios de robótica previos a esta. Esta ha tomado gran relevancia pues presenta oportunidades de desarrollo muy amplias. Dentro de estos estudios convergen ingenierías como la mecatrónica y la electrónica y ciencias puras como la biología. Por medio de la robótica en enjambre se pueden estudiar los patrones de comportamiento colectivo presentados por insectos o animales sociales.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Diseñar e implementar las capacidades de sensado de orientación y posición por medio de visión por computadora y odometría.

4.2. Objetivos específicos

- Comparar y determinar la metodología de control de posición óptima para las necesidades de los robots.
- Desarrollar un algoritmo para determinar la posición y orientación a partir de visión por computadora, específicamente por medio de la librería *OpenCV para Python*.
- Implementar el controlador en un modelo simulado del robot por medio de la plataforma *Webots*.

CAPÍTULO 5

Alcance

Este trabajo se centra exclusivamente en el diseño e implementación del controlador de posición y capacidades de sensado por medio de visión de computadora y odometría. Los resultados obtenidos formarán parte de la segunda fase del proyecto Robotat-Bitbot del departamento de Ingeniería Electrónica, Mecatrónica y Biomédica de la Universidad. El proyecto se trabaja en conjunto con los estudiantes José Andrés Castañeda y Pablo Prado. Castañeda se encarga del diseño de la plataforma para recolección de datos en la computadora y del bootloader, Pablo Prado trabaja el diseño mecánico, de potencia y el firmware en general de los robots Bitbot.

Debido a la situación socio-económica provocada por la pandemia COVID-19 del año 2020 no se pudo utilizar el equipo en la Universidad para realizar las pruebas correspondientes. A la hora adaptar el sistema al equipo utilizado en la Universidad se debe de adaptar el algoritmo de visión por computadora a las dimensiones de la mesa Robotat y a la cámara a utilizar.

CAPÍTULO 6

Marco teórico

6.1. Homografía

Homografía se refiere a la transformación planar entre dos contornos o áreas. Esta se puede aplicar solamente si se cumplen las siguientes condiciones:

1. Las imágenes son capturadas por la misma cámara.
2. Las imágenes muestran el mismo plano pero desde diferentes perspectivas. [6]

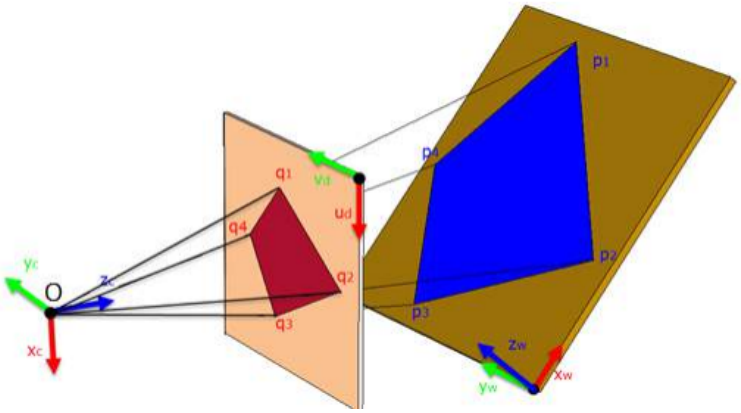


Figura 4: Ejemplo de homografía aplicada a dos planos [7].

Si se tienen dos puntos x y x', x se refiere al punto original y x' al punto luego de realizar la homografía

$$sx = Hx'. \tag{1}$$

Nótese que s es un factor de escala y H es la matriz de homografía. Esta es una matriz de 3×3 que posee 8 grados de libertad (DOF). Usualmente se normaliza imponiendo $h_{33} = 1$

o $h_{11} + h_{12} + h_{13} + h_{21} + h_{22} + h_{23} + h_{31} + h_{32} + h_{33} = 1$. [7] La matriz de homografía se define como:

$$H = \begin{bmatrix} h_{11} & h_{21} & h_{31} \\ h_{12} & h_{22} & h_{32} \\ h_{33} & h_{23} & h_{33} \end{bmatrix} \tag{2}$$

Esta matriz asegura que no se pierde información al realizar la transformación. Si se definen los vectores aumentados:

$$x = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad x' = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \tag{3}$$

la matriz de homografía H se puede calcular al ingresar 4 juegos de puntos x y x' en el siguiente sistema de ecuaciones.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & -u_1x_1 & -u_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & -v_1x_1 & -v_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & -u_2x_2 & -u_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & -v_2x_2 & -v_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & -u_3x_3 & -u_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & -v_3x_3 & -v_3y_3 \\ x_4 & y_4 & 1 & 0 & 0 & -u_4x_4 & -u_4y_4 \\ 0 & 0 & 0 & x_4 & y_4 & -v_4x_4 & -v_4y_4 \end{bmatrix} \cdot \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{bmatrix} \tag{4}$$

$$h_{33} = 1$$

La homografía posee muchas aplicaciones. Siendo relevante para el proyecto la eliminación de la distorsión de perspectiva. [7][6]

6.2. Reconocimiento de contornos

La detección de bordes de Canny es un algoritmo multi-etapa que fue desarrollado por John Canny en 1986. El algoritmo de Canny presenta las siguientes etapas. [8]

1. Aplicar un filtro de Gauss para eliminar el ruido.
2. Encontrar los gradientes de intensidad de color.
3. Aplicar una supresión *no-máxima* para eliminar falsos positivos.
4. Aplicar doble *threshold* para determinar los posibles bordes.
5. Encontrar los bordes por histéresis y eliminar los bordes sin conectar. [9]

Para aplicar la detección de Canny por medio de **OpenCV** basta con ejecutar la función *cv.Canny*. Esta función recibe como argumentos la fuente de la imagen y los valores de umbral para aplicar el doble filtrado y la histéresis. [8]

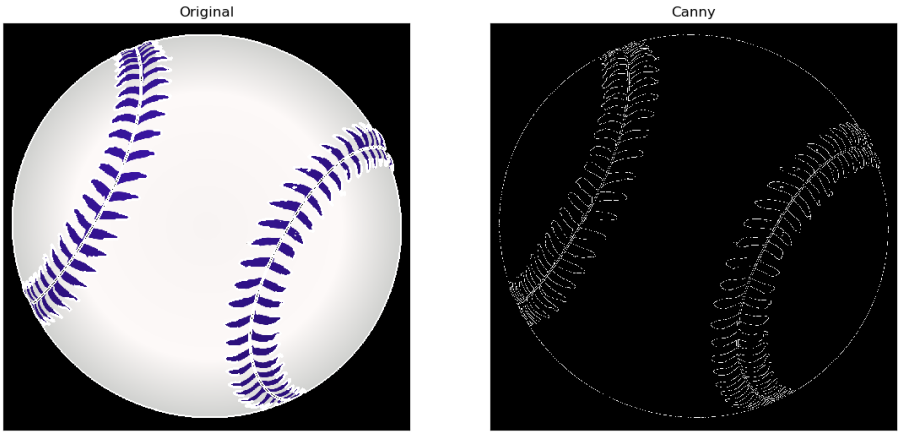


Figura 5: Demostración del funcionamiento del algoritmo de reconocimiento de contornos de Canny

6.3. Modelo uniciclo

El robot con ruedas más simple que se puede modelar es un uniciclo o bien una rueda de radio r. Se asume que esta rueda no presenta ningún tipo de deslizamiento y que el plano sobre el que rueda no presenta irregularidades. La configuración de un uniciclo de radio r es $q = (\Phi, x, y, \theta)$, en donde (x, y) es la ubicación de su punto de contacto sobre el plano, Φ es la dirección a la que rueda y θ es el ángulo de rotación de la rueda.[10] (Ver Figura 6)

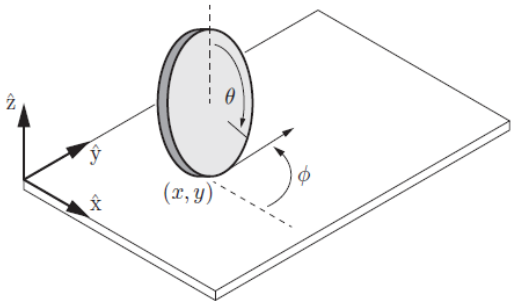


Figura 6: Configuración de un uniciclo [10].

Las ecuaciones cinemáticas del uniciclo son según [11]:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{\Phi} \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ r\cos\Phi & 0 \\ r\sin\Phi & 0 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = G(q)\mathbf{u} = g_1(\mathbf{q})u_1 + g_2(\mathbf{q})u_2 \tag{5}$$

Estas ecuaciones reciben como entrada de control la velocidad hacia delante o atrás (u_1) y la velocidad de giro respecto al plano (u_2). Es importante notar que estas velocidades

están restringidas a las capacidades del robot. [11]

$$-u_{1,max} \leq u_1 \leq u_{1,max}, \tag{6}$$

$$-u_{2,max} \leq u_2 \leq u_{2,max}. \tag{7}$$

A las columnas de la matriz $G(\mathbf{q}) \in \mathbb{R}^4$ se les denomina campos vectoriales de velocidad. Cabe mencionar que todos los modelos cinemáticos de robots *no holonómicos* tendrán la forma: $\dot{\mathbf{q}} = G(\mathbf{q})\mathbf{u}$. Las consideraciones más importantes en estos robots son según [10]:

1. Si las entradas de control son 0, $u = [0, 0]^T$, no existe velocidad, es decir no existe deslizamiento.
2. Los campos vectoriales $g_1(\mathbf{q})$ y $g_2(\mathbf{q})$ son funciones del vector de configuración $\mathbf{q}$.
3. $\dot{\mathbf{q}}$ es lineal en el control.

6.4. Robot diferencial

Un robot diferencial consiste de dos ruedas controladas independientemente de radio r, los cuales rotan sobre el mismo eje. Estos robots presentan un soporte adicional para asegurar que siempre se mantenga en posición horizontal. [10]

Cada rueda estará alejada d unidades del centro del robot, por lo que las ruedas estarán $2d$ unidades alejadas una de otra. El vector de configuración de los robots diferenciales es $\mathbf{q} = (\Phi, x, y, \theta_L, \theta_R)$. [10]

En la Figura 7 se puede observar que (x, y) son las coordenadas de su centro, θ_L, θ_R son los ángulos de rotación de cada rueda. De esta forma se obtienen las ecuaciones cinemáticas: [12]

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{\Phi} \\ \dot{x} \\ \dot{y} \\ \dot{\theta_L} \\ \dot{\theta_R} \end{bmatrix} = \begin{bmatrix} -\frac{r}{2d} & \frac{r}{2d} \\ \frac{r}{2}\cos\Phi & \frac{r}{2}\cos\Phi \\ \frac{r}{2}\sin\Phi & \frac{r}{2}\sin\Phi \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_L \\ u_R \end{bmatrix} = G(\mathbf{q})\mathbf{u}. \tag{8}$$

Las entradas de control de estos robots son las velocidades angulares en cada una de sus ruedas. $[u_L, u_R]^T$. Generalmente no interesa el ángulo de rotación de las ruedas, por lo que se puede simplificar su modelo.

Las ventajas de este modelo de robots es su simpleza, ya que se puede conectar directamente el motor al eje de cada rueda. Estos robots presentan una maniobrabilidad superior a otros modelos, ya que se puede generar una rotación sobre su centro al aplicar velocidades contrarias en las ruedas.

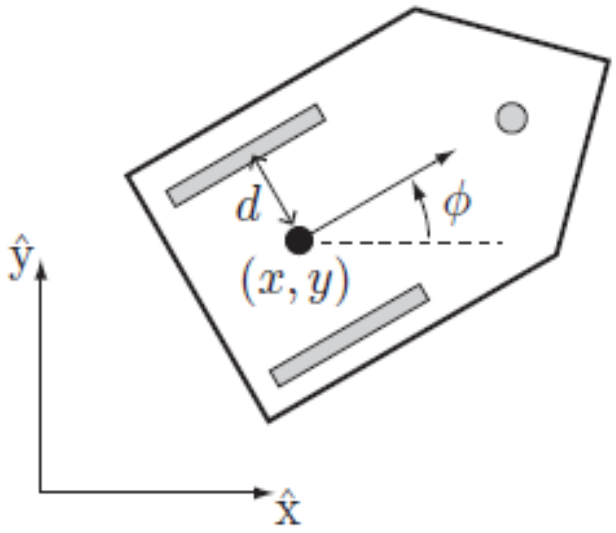


Figura 7: Configuración de un robot diferencial [10].

6.4.1. Estrategias de control para un robot diferencial

Para aplicarle control a un robot diferencial se debe de modelar como un uniciclo, ya que este es más fácil de controlar. Seguido a esto ya se puede ajustar la salida del controlador al modelo complejo de un robot diferencial. El modelo uniciclo básico indica que:

$$\begin{aligned} \dot{x} &= v\cos\theta, \\ \dot{y} &= v\sin\theta, \\ \dot{\theta} &= \omega. \end{aligned} \tag{9}$$

A partir de este modelo se derivan los modelos para un uniciclo real y un robot diferencial. En la Figura 8 se pueden apreciar los parámetros necesarios para describir el modelo real de un robot diferencial.[10]

$$\begin{aligned} \dot{x} &= \frac{1}{2}r(\dot{\phi_L} + \dot{\phi_R})\cos\theta, \\ \dot{y} &= \frac{1}{2}r(\dot{\phi_L} + \dot{\phi_R})\sin\theta, \\ \dot{\theta} &= \frac{r}{2l}(\dot{\phi_R} - \dot{\phi_L}). \end{aligned} \tag{10}$$

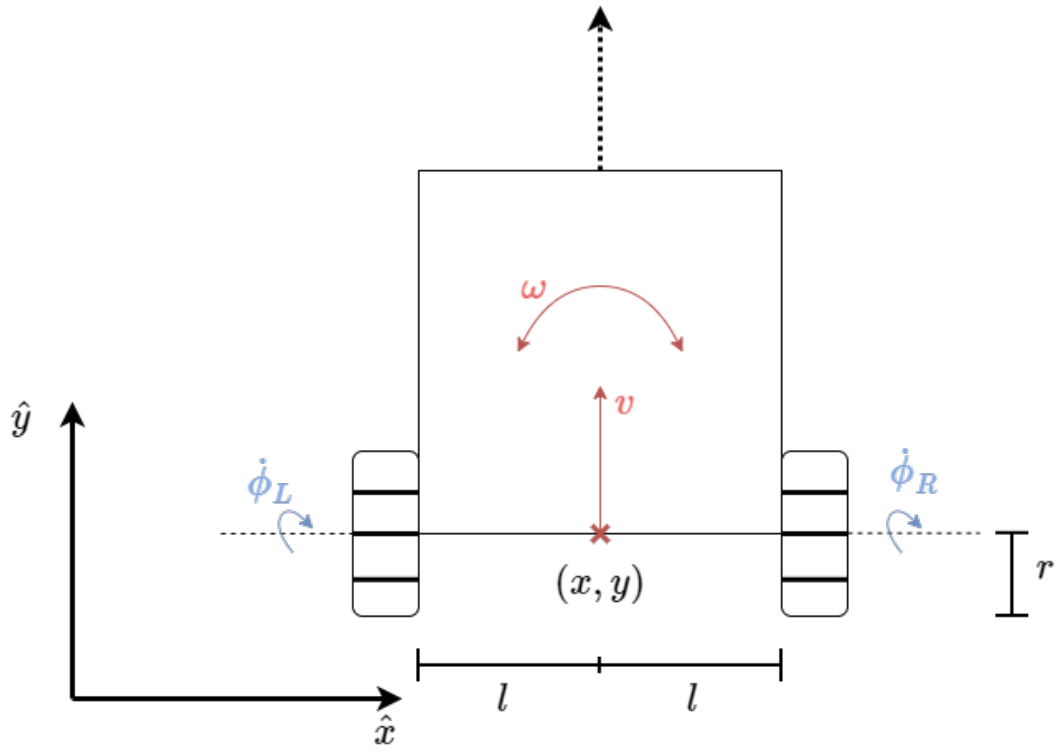


Figura 8: Modelo de robot diferencial y sus parámetros

Los problemas de control que se resolverán son: **estabilización y rastreo de trayectorias**. En ambos casos se intentará alcanzar una posición (x_g, y_g) o $(x_g(t), y_g(t))$, respectivamente. [13]

Control clásico

Para aplicar control clásico se implementará un controlador PID:. Se definen dos errores: error de posición y error de orientación. El error de posición responde a la siguiente ecuación

$$e_p = \left\| \begin{matrix} x_g - x \\ y_g - y \end{matrix} \right\| = \sqrt{(x_g - x)^2 + (y_g - y)^2}, \tag{11}$$

También se define un error angular. Para esto se necesita acotar el ángulo de giro entre $[-\pi, \pi]$. El ángulo se calcula por medio de:

$$\theta_g = \text{atan2}\left(\frac{y_g - y}{x_g - x}\right)$$

Seguido de esto se define el error de orientación como:

$$\begin{aligned} e_o &= \theta_g - \theta, \\ e_o &= \text{atan2}\left(\frac{\sin\theta_g - \theta}{\cos\theta_g - \theta}\right). \end{aligned} \tag{12}$$

Las variables de control (la velocidad lineal y angular) responden a las siguientes ecua-

ciones

$$\begin{aligned} v &= K_{Pp}e_p + K_{Ip}\int_0^t e_p(\tau)d\tau + K_{Dp}\dot{e_p}, \\ w &= K_{Po}e_o + K_{Io}\int_0^t e_o(\tau)d\tau + K_{Dp}\dot{e_o}. \end{aligned} \tag{13}$$

Este controlador tiende a provocar un movimiento en espirales cerca del punto final, por lo que se puede implementar un acercamiento exponencial en el controlador de velocidad lineal y así obtener

$$\begin{aligned} v &= \frac{v_0(1-e^{-\alpha e_p^2})}{e_p}, \\ w &= K_{Po}e_o + K_{Io}\int_0^t e_o(\tau)d\tau + K_{Dp}\dot{e_o}. \end{aligned} \tag{14}$$

En donde v_0 es la velocidad lineal máxima y α es el coeficiente de ajuste del controlador.

Control moderno LTI:
Para aplicar control moderno LTI: se debe de definir un vector de estados:

$$\boldsymbol{\xi} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{15}$$

A partir de esto se puede definir un sistema dinámico no lineal como: [11]

$$\dot{\boldsymbol{\xi}} = f(\boldsymbol{\xi}, v, w) \tag{16}$$

Al definirse un punto de operación $\boldsymbol{\xi}_{ss}$, tal que $\dot{\boldsymbol{\xi}} = 0$, surgen problemas de linealización. Por esta razón se aproxima la posición del uniciclo a través de un punto cercano a la posición del robot. Este punto tendrá las coordenadas $(\tilde{x}, \tilde{y})$ y estará l unidades alejado de la posición real. Nótese que $l \approx 0$. [10]

$$\begin{aligned} \tilde{x} &= x + l\cos\theta, \\ \tilde{y} &= y + l\sin\theta. \end{aligned} \tag{17}$$

La dinámica del punto aproximado será, por lo tanto:

$$\begin{aligned} \dot{\tilde{x}} &= \dot{x} - l\dot{\theta}\sin\theta = v\cos\theta - lw\sin\theta, \\ \dot{\tilde{y}} &= \dot{y} + l\dot{\theta}\cos\theta = v\sin\theta + lw\cos\theta. \end{aligned} \tag{18}$$

La velocidad del punto $(\tilde{x}, \tilde{y})$ será:

$$\begin{bmatrix} \dot{\tilde{x}} \\ \dot{\tilde{y}} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{19}$$

Al resolver para la velocidad del uniciclo se obtiene:

$$\begin{aligned} \begin{bmatrix} v \\ w \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & 1/l \end{bmatrix} {}^{B}R_I(\theta) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \\ \begin{bmatrix} v \\ w \end{bmatrix} &= M(l,\theta) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \end{aligned} \tag{20}$$

La matriz M convierte las velocidades lineales $[u_1, u_2]^T$ a las velocidades del sistema de un uniciclo $[v, \omega]^T$. Una consideración importante es que la distancia l debe de tender a 0 para que el *difeomorfismo:* funcione correctamente. De esta forma se obtiene el siguiente sistema dinámico: [14]

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{21}$$

Este sistema responde a la forma estándar de un sistema dinámico:

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$

Para aplicarle control **LQR:** al sistema se debe de definir un punto de operación:

$$\mathbf{x_{ss}} = \begin{bmatrix} x_g \\ y_g \end{bmatrix}. \tag{22}$$

Nótese que las matrices Q y R deben de cumplir con la condición de ser positivas definidas. A partir de esto se puede definir una entrada óptima:

$$\begin{aligned} \mathbf{u}^* = -\mathcal{K}_{LQR}(\mathbf{x} - \mathbf{x}_{ss}) + \mathbf{u}_{ss} \\ \mathbf{u}_s s = 0 \end{aligned} \tag{23}$$

Después de obtener la entrada óptima se procede a realizar la conversión de velocidades de un sistema a otro por medio de:

$$\begin{bmatrix} v \\ w \end{bmatrix} = M(l, \theta)\mathbf{u}^* \tag{24}$$

También se puede implementar un controlador **LQI:** al definir una señal de referencia y una salida, tal que:

$$\begin{aligned} \mathbf{r} &= \begin{bmatrix} x_g \\ y_g \end{bmatrix}, \\ \mathbf{y}_r &= \begin{bmatrix} x \\ y \end{bmatrix}. \end{aligned} \tag{25}$$

El error, por lo tanto será:

$$\dot{\sigma} = \mathbf{y}_r - \mathbf{r} = \begin{bmatrix} x - x_g \\ y - y_g \end{bmatrix} \tag{26}$$

Se procede a definir las matrices aumentadas del sistema necesarias para calcular el controlador LQI:.

$$\begin{aligned} \mathcal{A} &= \begin{bmatrix} A & 0_{n\times p} \\ C_r & 0_{p\times p} \end{bmatrix}, \\ \mathcal{B} &= \begin{bmatrix} B \\ B_r \end{bmatrix}. \end{aligned} \tag{27}$$

Antes de implementar el controlador se debe de tener en cuenta que: $Q \in \mathbb{R}^4$ y $R \in \mathbb{R}^2$. Luego de implementar el controlador se obtiene una entrada óptima $\mathbf{u}^*$, la cual está definida por:

$$\mathbf{u}^* = -\mathcal{K}_{LQI} \begin{bmatrix} \mathbf{x} \\ \sigma \end{bmatrix} = \mathcal{K}_{PD}\mathbf{x} - \mathcal{K}_I \boldsymbol{\sigma} \tag{28}$$

Nótese que σ se implementa como integradores en un rango de $[0, t]$:

$$\boldsymbol{\sigma} = \int_0^t \begin{bmatrix} x(\tau) - x_g(\tau) \\ y(\tau) - y_g(\tau) \end{bmatrix} d\tau \tag{29}$$

6.5. Filtro de Kalman

El filtro de Kalman es un algoritmo de procesamiento de data recursivo que estima el estado de un sistema lineal ruidoso [15]. El ruido presentado por el sistema lineal debe de ser ruido blanco gaussiano aditivo. Esto significa que el ruido en un momento determinado no tiene influencia en el ruido producido en un instante diferente en el tiempo. Al hablar de un *estado* de un sistema lineal, se hace referencia a un vector $\mathbf{x}$ de n variables o dimensiones. Estas n variables describen las propiedades del sistema lineal en cuestión. Un ejemplo claro es el vector de estados de un robot diferencial, $\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$. Este está compuesto por su posición sobre el plano x, y y por su orientación θ. [15]

El problema con el estado de un sistema es que no siempre es observable directamente. Sin embargo, el filtro de Kalman (KF por sus siglas en inglés) puede estimar el estado de un sistema si este tiene acceso a las mediciones del mismo. Estas mediciones deben de estar linealmente relacionadas al estado. Las mediciones, por lo general, se encuentran contaminadas por por ruido. Si el ruido responde a una distribución Gaussiana, el KF es estadísticamente óptimo para realizar la estimación. [16]

Las aplicaciones más comunes del KF involucran control y predicción de sistemas dinámicos. Cuando el KF se utiliza en control, como lo es el caso de este trabajo, este se utiliza para estimar el estado de los sistemas a través de mediciones indirectas y probablemente ruidosas. Por otra parte, cuando se utiliza el filtro para la predicción de sistemas, este intenta predecir el estado futuro de un sistema que para el humano es imposible de controlar.

Dado que los sistemas dinámicos se consideran ruidosos, se definen de la siguiente manera: [15]

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}(t)\mathbf{x} + \mathbf{B}(t)\mathbf{u} + \mathbf{F}(t)\boldsymbol{\omega}(t), \\ \mathbf{y} &= \mathbf{C}(t)\mathbf{x} + \mathbf{v}(t). \end{aligned} \tag{30}$$

[17] siendo $\boldsymbol{\omega}(t)$ y $\mathbf{v}(t)$ la perturbación en los actuadores y el ruido de los sensores, respectivamente. Como se menciona anteriormente, $\boldsymbol{\omega}(t)$ y $\mathbf{v}(t)$ deben de ser ruido blanco no relacionado, es decir: [18]

$$\begin{aligned} \boldsymbol{\omega} &\sim \mathcal{N}(0, \mathbf{Q}_\omega(t)), \\ \mathbf{v} &\sim \mathcal{N}(0, \mathbf{Q}_v(t)). \end{aligned} \tag{31}$$

En las ecuaciones (31) se observa que la distribución debe de ser Gaussiana, de media cero. Las matrices $\mathbf{Q}_\omega$ y $\mathbf{Q}_v$ se denominan matrices de covarianza de proceso y observación, respectivamente. Estas matrices se definen formalmente como el valor esperado de las varianzas correspondientes a los actuadores y a los sensores.

$$E\{\boldsymbol{\omega}(t)\boldsymbol{\omega}(t)^T\} = Q_\omega(t)\delta(t-\tau), \tag{32}$$

$$E\{\mathbf{v}(t)\mathbf{v}(t)^T\} = Q_\mathbf{v}(t)\delta(t-\tau). \tag{33}$$

Sin embargo, en la práctica se implementan como:

$$Q_{\boldsymbol{\omega}} = \begin{bmatrix} \sigma^2_{\omega_1} & & & 0 \\ & \sigma^2_{\omega_2} & & \\ & & \ddots & \\ 0 & & & \sigma^2_{\omega_n} \end{bmatrix} \tag{34}$$

$$Q_{\mathbf{v}} = \begin{bmatrix} \sigma^2_{\mathbf{v}_1} & & & 0 \\ & \sigma^2_{\mathbf{v}_2} & & \\ & & \ddots & \\ 0 & & & \sigma^2_{\mathbf{v}_n} \end{bmatrix} \tag{35}$$

Cuando se utiliza el KF como observador de estados, este se encuentra como la solución a un problema de minimización: [19]

$$\begin{aligned} \dot{\hat{\mathbf{x}}} &= \mathbf{A}(t)\hat{\mathbf{x}} + \mathbf{B}(t)\mathbf{u} + \mathbf{L}(t)(\mathbf{y} - \mathbf{C}(t)\hat{\mathbf{x}}), \\ \mathbf{L}(t) &= \underset{\mathbf{L}(t)}{\operatorname{argmin}}\ E\{(\mathbf{x}-\hat{\mathbf{x}})^T(\mathbf{x}-\hat{\mathbf{x}})\}. \end{aligned} \tag{36}$$

La solución óptima encontrada responde a:

$$\mathbf{L}^* = \mathbf{P}(t)\mathbf{C}^T(t)\mathbf{Q}_\mathbf{v}^{-1}(t), \tag{37}$$

En donde $\mathbf{P}(t) = E\{(\mathbf{x}-\hat{\mathbf{x}})(\mathbf{x}-\hat{\mathbf{x}})^T\}$. A esta matriz se le denomina **matriz de covarianza de estimación**. Nótese que esta matriz es la solución a la Ecuación de Riccatti. [15]

Cuando se trabaja el filtro de Kalman sobre un sistema LTI: y se asumen varianzas estáticas en el tiempo, se obtiene el filtro de Kalman en estado estacionario. El cual responde a las siguientes ecuaciones.[19]

$$\mathbf{L}^* = \mathbf{P}\mathbf{C}^T\mathbf{Q}_\mathbf{v}^{-1}. \tag{38}$$

Este filtro satisface la ecuación algebráica de Riccati:

$$0 = \mathbf{A}\mathbf{P} + \mathbf{P}\mathbf{A}^T - \mathbf{P}\mathbf{C}^T\mathbf{Q}_\mathbf{v}^{-1}\mathbf{C}\mathbf{P} + \mathbf{F}\mathbf{Q}_\omega\mathbf{F}^T, \tag{39}$$

el observador, por lo tanto presenta la siguiente forma:

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}^*(\mathbf{y} - \mathbf{C}\hat{\mathbf{x}}). \tag{40}$$

6.5.1. Filtro de Kalman discreto

Los sistemas dinámicos lineales se presentan, por lo general como sistemas contínuos en el tiempo. Sin embargo las mediciones de los sensores y en los actuadores se realiza en tiempo discreto. Por esta razón surge la necesidad de discretizar el observador para realizar la estimación del estado actual de los sistemas. A diferencia del KF en tiempo contínuo, en el cual se resuelve la ecuación de Riccati para obtener $\mathbf{L}$, el KF en tiempo discreto presenta etapas de procesamiento para obtener el estimado de estado. Las dos etapas que presenta el filtro de Kalman en tiempo discreto son: **predicción** y **corrección**. El filtro de Kalman en tiempo discreto responde a la siguiente forma. [16], [17]

$$\hat{\mathbf{x}}_k = \mathbf{A}_k\hat{\mathbf{x}}_{k-1} + \mathbf{B}_k\mathbf{u}_k + \boldsymbol{\omega}_k, \tag{41}$$

$$\mathbf{y_k} = \mathbf{C}_k\hat{\mathbf{x}}_k + \mathbf{v}_k \tag{42}$$

Como se observa en la Figura 9, el algoritmo del filtro de Kalman discreto presenta dos tipos de estimaciones, $\hat{\mathbf{x}}_{k|k-1}$ y $\hat{\mathbf{x}}_{k|k}$. Estas estimaciones del vector de estado reciben el nombre de **a-priori** y **a-posteriori**, respectivamente. Adicionalmente se define la **innovación** como

$$\mathbf{z}_k = \mathbf{y}_k - \mathbf{C}_k\hat{\mathbf{x}}_{k|k-1}. \tag{43}$$

Esta representa la diferencia de las cantidades medidas por los sensores con las cantidades estimadas *a-priori*. La innovación se utiliza para calcular la ganancia de Kalman $\mathbf{L}_k$ en dos pasos. De primero se define la matriz de covarianza de la innovación $\mathbf{S}_k$ como: [16]

$$\mathbf{S}_k = \mathbf{Q}_\mathbf{v}[k] + \mathbf{C}[k]\mathbf{P}_{k|k-1}\mathbf{C}[k]^T. \tag{44}$$

$\mathbf{L}_k$ se calcula, por lo tanto como:

$$\mathbf{L}_k = \mathbf{P}_{k|k-1}\mathbf{C}[k]^T S^{-1}. \tag{45}$$

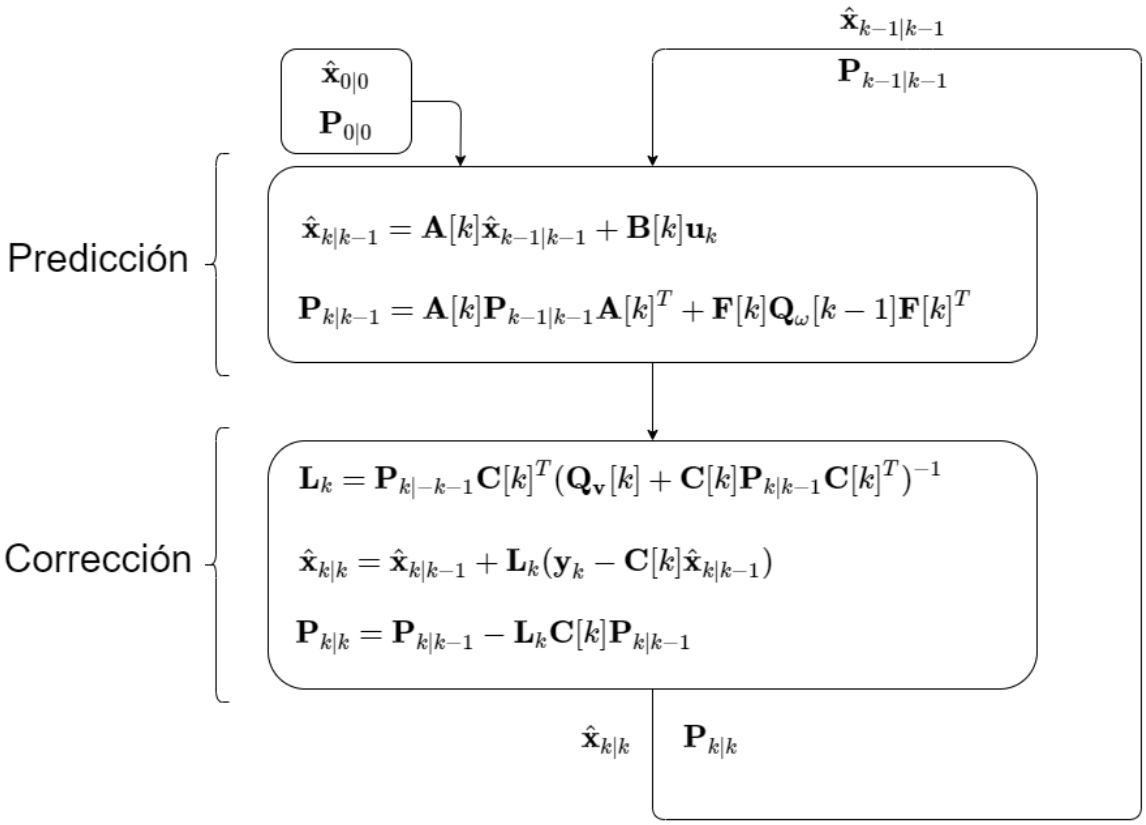


Figura 9: Algoritmo del filtro de Kalman en tiempo discreto

6.5.2. Filtro de Kalman extendido

En el estudio de estimación, el filtro de Kalman extendido (EKF por sus siglas en inglés) es la versión no lineal del filtro de Kalman. En el caso del EKF se linealiza el sistema utilizando la estimación actual. Este también suele llamarse filtro de Kalman-Schmidt. Las aplicaciones principales del EKF son en sistemas de navegación, en donde las mediciones no necesariamente son lineales. Los modelos de transición de estados y de observación no deben de ser, en este caso funciones, lineales del estado sino basta con que sean diferenciables. En las ecuaciones (46) y (47) se modela un sistema discreto no lineal. [20]

$$\mathbf{x}_{k+1} = \mathcal{F}(\mathbf{x}_k,\, \mathbf{u}_k,\, \omega_k) \tag{46}$$

$$\mathbf{y}_k = \mathcal{H}(\mathbf{x}_k) + \mathbf{v}_k \tag{47}$$

En este caso se sigue manteniendo la condición que $\boldsymbol{\omega}_k$ y $\mathbf{v}_k$ sean el ruido de proceso y observación distribuidos Gaussianamente, con media cero y no correlacionados. La función $\mathcal{F}$ se utiliza para calcular el estado predicho. Las funciones $\mathcal{F}$ y $\mathcal{H}$ no pueden ser aplicadas para las matrices de covarianza, por esta razón se calculan los jacobianos de estas para poder ser aplicadas. [20]

Para utilizar el EKF con el algoritmo mostrado en la Figura 9 se deben de redefinir las matrices $\mathbf{A}[k]$, $\mathbf{B}[k]$, $\mathbf{C}[k]$ y $\mathbf{F}[k]$ como los jacobianos de los modelos de trancisión y de observación. [15], [20], [16]

$$\mathbf{A}[k] = \frac{\partial \mathcal{F}(\hat{\mathbf{x}}_{k-1|k-1},\, \mathbf{u}_{k-1},\, 0)}{\partial \mathbf{x}}, \tag{48}$$

$$\mathbf{B}[k] = \frac{\partial \mathcal{F}(\hat{\mathbf{x}}_{k-1|k-1},\, \mathbf{u}_{k-1},\, 0)}{\partial \mathbf{u}}, \tag{49}$$

$$\mathbf{C}[k] = \frac{\partial \mathcal{H}(\hat{\mathbf{x}}_{k|k-1})}{\partial \mathbf{x}}, \tag{50}$$

$$\mathbf{F}[k] = \frac{\partial \mathcal{F}(\hat{\mathbf{x}}_{k-1|k-1},\, \mathbf{u}_{k-1},\, 0)}{\partial \omega}. \tag{51}$$

6.6. OpenCV

OpenCV (Open Source Computer Vision Library en inglés) es una librería de programación de código abierto enfocada a la visión por computadora en tiempo real. Esta librería está desarrollada por medio de C++, por lo que su interfaz principal es esta. Sin embargo, OpenCV se puede utilizar en los lenguajes Python, Java y Matlab. Las últimas versiones de esta librería presentan una limitada cantidad de instrucciones utilizables en JavaScript para el desarrollo web. Esta librería empezó como un desarrollo de Intel, por lo que implementa optimizaciones de procesamiento propias de la marca. Un ejemplo de esto es la librería de multiprocesamiento de Intel IPP (Integrated Performance Primitives), pues OpenCV utiliza esta librería automáticamente si el procesador de la computadora lo soporta. [21]

La librería OpenCV no solo posee capacidades de procesamiento de imágenes, sino esta se utiliza también para aplicaciones de aprendizaje automático (machine learning). [21]

Las aplicaciones más comunes de OpenCV incluyen:

- Reconocimiento facial.
- Reconocimiento de gestos.
- Interacción humano-computador.
- Robótica móvil.
- Identificación de objetos.
- Realidad aumentada.

Esta librería se puede encontrar en el siguiente repositorio de Github: `https://github.com/opencv/opencv`. Por otra parte toda la documentación necesaria para utilizarla se puede encontrar en la página oficial de documentación: `https://docs.opencv.org/`.

6.7. Webots

Webots es un software de código abierto enfocado en la simulación de robótica en tres dimensiones. Este simulador es ampliamente utilizado en la industria y en la academia. El desarrollo del simulador empezó en 1996 por Dr. Oliver Michel y en 1998 Cyberbotics Ltd. lo lanza como software bajo licencia de propiedad. En el 2018 se libera el software para volverlo de código abierto. [22]

Webots utiliza la librería ODE (Open Dynamics Engine) para detectar y simular colisiones y la dinámica de los cuerpos rígidos. Esta librería permite simular con gran exactitud propiedades físicas tales como velocidad, inecia y fricción. Una ventaja que presenta Webots en la simulación es la compatibilidad con los sensores y actuadores más utilizados en la robótica. Otra facilidad que presenta Webots es la capacidad de programar los controladores de los robots en diferentes lenguajes de programación. Estos pueden ser: C, C++, Python, ROS, Matlab y Java. El repositorio y la documentación se pueden encontrarn en los enlaces `https://github.com/cyberbotics/webots` y `https://cyberbotics.com/doc/reference/index`, respectivamente. [22] Un ejemplo del entorno de simulación puede encontrarse en la Figura 10.

Las aplicaciones más comunes del simulador incluyen:

- Prototipado rápido de robots móviles.
- Simulaciones multirobot en ejambre.
- Simulaciones de aplicaciones de visión por computadora.

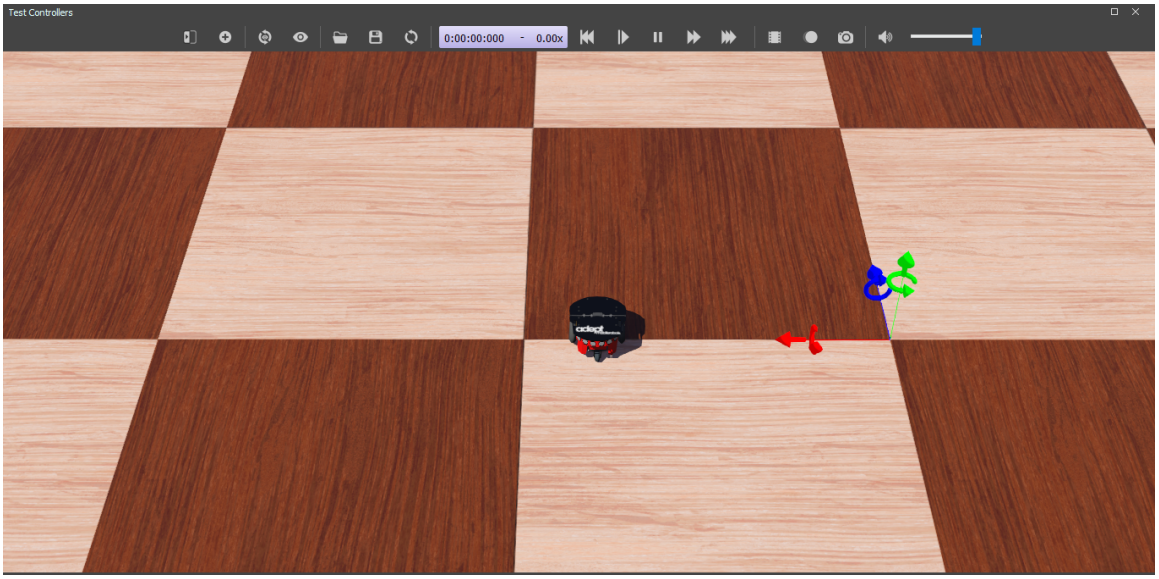


Figura 10: Entorno de simulación de Webots.

CAPÍTULO 7

Materiales y métodos

7.1. Materiales

Como se menciona en el alcance, no fue posible utilizar el equipo ubicado en las instalaciones de la Universidad, por lo que se utilizó una mesa de trabajo escalada y una cámara web diferente.

7.1.1. Visión por computadora

- Mesa de trabajo casera con dimensiones de $33cm$ x $44cm$.
- Cámara de celular Huawei P30 Pro utilizado como Webcam.
- Cinta métrica.
- Computadora ASUS X555LB.
  - Procesador: Intel Core i7 5500U de 2.40GHZ y 4MB de memoria caché.
  - Sistema operativo: Windows 10 Pro.
  - Memoria RAM: 12GB.
  - Tarjeta gráfica: NVIDIA GeForce 940M de 2GB.

7.1.2. Controladores para Bitbot

- Simulador de robótica OpenSource Webots.
- Computadora ASUS X555LB.

- Procesador: Intel Core i7 5500U de 2.40GHZ y 4MB de memoria caché.
- Sistema operativo: Windows 10 Pro.
- Memoria RAM: 12GB.
- Tarjeta gráfica: NVIDIA GeForce 940M de 2GB.

- Matlab R2019a.
- Mesa de trabajo casera con dimensiones de $33cm \times 44cm$.

7.1.3. Fusión de sensores

- Simulador de robótica OpenSource Webots.
- Computadora ASUS X555LB.
  - Procesador: Intel Core i7 5500U de 2.40GHZ y 4MB de memoria caché.
  - Sistema operativo: Windows 10 Pro.
  - Memoria RAM: 12GB.
  - Tarjeta gráfica: NVIDIA GeForce 940M de 2GB.
- Matlab R2019a.

7.2. Métodos

7.2.1. Visión por computadora

Objetivo

Determinar la eficacia del sistema de visión por computadora.

Pregunta experimental

¿El algoritmo de visión por computadora medirá correctamente distancias sobre un plano luego de haber realizado la homografía?

Hipótesis

El algoritmo de visión por computadora medirá correctamente las distancias sobre la mesa de trabajo con un error mínimo.

Metodología

Se colocarán puntos sobre la mesa y se realizarán dos mediciones desde el origen hasta ellos. Las mediciones a realizar serán: una medición física utilizando como guía la cuadrícula dibujada en la mesa de pruebas, esta será la referencia para comparar la segunda medición; la segunda medición se realizará por medio del algoritmo de visión por computadora, esta será la medición experimental. Se dividirá la mesa de trabajo en sectores para medir el error en cada sector.

7.2.2. Controladores para Bitbot

Objetivo

Comparar y determinar la metodología de control óptima para los robots Bitbot.

Pregunta experimental

¿Cuál será la metodología de control óptima para los robots Bitbot?

Hipótesis

La metodología de control óptimo LTI: como LQR: y LQI: serán los controladores óptimos para la aplicación planteada.

Metodología

Se utilizará un modelo generado en Webots para probar y validar los esquemas de control posibles para un robot diferencial, específicamente el Bitbot. Se probará su comportamiento alcanzando un punto y su comportamiento siguiendo una ruta. Se validarán los datos obtenidos por medio de Webots al probar los esquemas de control en el robot físico.

7.2.3. Fusión de sensores

Objetivo

Determinar la eficacia del algoritmo de reconocimiento de pose por medio de fusión de sensores.

Pregunta experimental

¿El algoritmo de obtención de pose por medio de fusión de sensores medirá correctamente la pose del robot Bitbot?

Hipótesis

El algoritmo implementado obtendrá correctamente la posición y orientación del robot Bitbot.

Metodología

Se utilizará un modelo generado en Webots para probar y validar el algoritmo de obtención de pose por medio de fusión de sensores. El robot se moverá hasta una posición conocida y se medirá el error obtenido por la medición interna.

CAPÍTULO 8

Visión por computadora

8.1. Algoritmo de reconocimiento de pose y agentes

Se desarrolló un algoritmo de reconocimiento de agentes por medio de visión por computadora. Este fue desarrollado por medio de la librería OpenCV en Python 3. Asimismo se utilizó un identificador de agentes basado en los códigos QR:. Los identificadores fueron realizados específicamente para los BitBots, pues tienen un tamaño de 5x5cm. Estos funcionan por medio de una codificación binaria, en donde cada cuadro del identificador corresponde a un bit de un Byte. Un ejemplo de los identificadores se puede encontrar a en la Figura 11. Los identificadores están diseñados tal que fuesen fáciles de imprimir, fáciles de leer y entendibles a simple vista. Su codificación binaria empieza en el cuadro a la derecha del cuadro blanco. Este corresponde al bit 0, los siguientes se leen de izquierda a derecha y de forma descendente. Esto significa que el cuadro en la esquina inferior derecha es el bit 7 de la codificación. Esta codificación permite un máximo de 255 agentes sobre la mesa. Estos identificadores permiten determinar eficientemente su centroide y su orientación, pues se sabe que el cuadro blanco corresponde siempre a la esquina superior izquierda del identificador. También se debe de tener en cuenta que deben de estar colocados paralelamente a la mesa y la superficie de la mesa debe de poder provocar un contraste con la orilla negra de los identificadores. Esto significa que la mesa debe de ser de un color claro como blanco o un gris muy pálido. En el algoritmo 1 se puede observar el proceso para realizar una correcta lectura de los identificadores. Esta lectura se debe de realizar luego de haber filtrado los contornos correspondientes a los agentes activos.

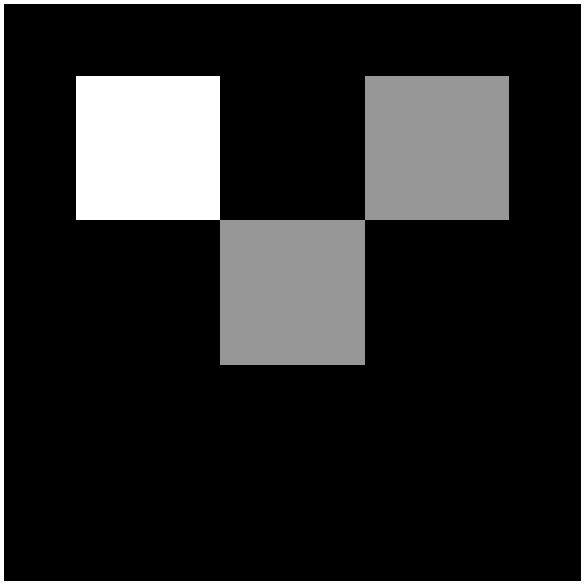


Figura 11: Ejemplo de indentificador BitBot número 10

Algorithm 1: Lectura de identificadores

input : Array de contornos y matriz de rotación del contorno
output: ID del agente y pose del identificador

Inicialización de variables
for $i \leftarrow 0$ **to** *(tamaño(array)-1)* **do**
  Extraer ángulo de rotación de la matriz de rotación $\rightarrow \theta'$
  Enderezar contorno[i]
  Aislar contorno enderezado[i]
  Guardar contorno aislado y enderezado[i]

  Leer esquinas del contorno
  if *Esquina Leída* **then**
    Aplicar rotación según esquina leída
  end
  Extraer centroide del contorno[i]
  Pose $\leftarrow (x_{cent.}, y_{cent.})$, Ángulo

  Leer celdas grises del contorno enderezado, $gris \rightarrow 1$, $negro \rightarrow 0$
  Guardar lecturas en un array
  Factor $\leftarrow 0$
  ID $\leftarrow 0$
  for $i \leftarrow 0$ **to** 7 **do**
    Factor $\leftarrow 2^{lectura[i]}$
    ID $\leftarrow$ ID + Factor
  end
end

El algoritmo encargado de determinar la posición y orientación de los identificadores utiliza el método de Canny para reconocimiento de contornos y *thresholding:* adaptativo. Aunque el método de Canny es computacionalmente pesado se logró obtener una frecuencia de lectura de 11.37Hz con 7 agentes activos. El algoritmo está conformado de tres secciones principales: Aplicación de homografía, reconocimiento de contornos de interés, lectura y cálculo de pose a partir de los identificadores. La lectura de pose y de ID del identificador se obtiene por medio del algoritmo 1. La sección encargada de eliminar la distorsión provocada por la pose de la cámara respecto al plano de la mesa es la homografía. Esta es una sección relativamente corta en comparación a las otras, sin embargo, es la más importante pues asegura una correcta lectura de los agentes. La segunda sección del algoritmo está encargada del filtrado de contornos erróneos o duplicados. Esta sección devuelve solamente los contornos correspondientes a los identificadores que se encuentran activos en la mesa de pruebas. Esta se debe de calibrar según la mesa de pruebas y cámara a utilizar, pues el proceso de filtrado requiere conocer las dimensiones del identificador relativas a la mesa de trabajo. En la Figura 12 se puede encontrar un ejemplo de los resultados del algoritmo completo de visión por computadora con 7 agentes activos en la mesa. A continuación se puede encontrar el algoritmo en forma de pseudocódigo.

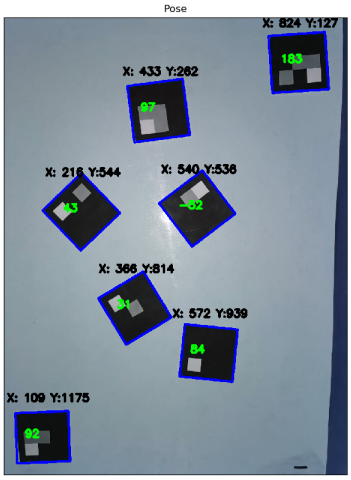
Algorithm 2: Visión por computadora

input : Lectura de cámara
output: Pose de agentes y IDs

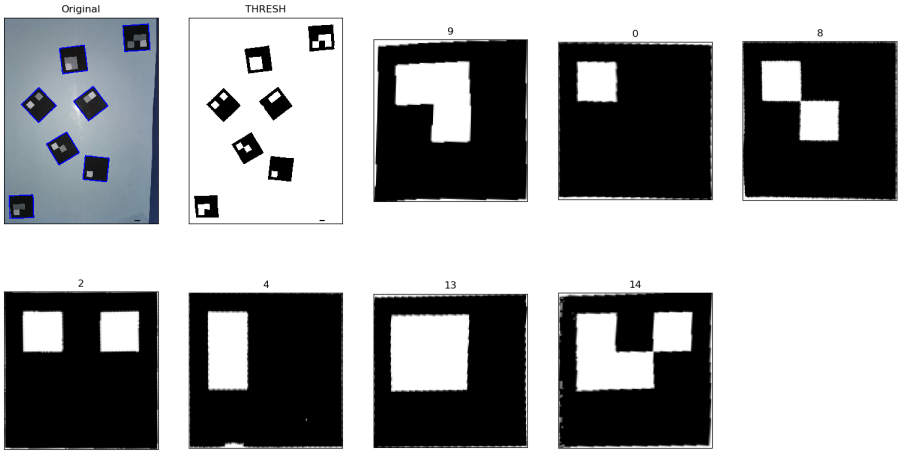
Inicialización de variables
Esperar la inicialización de la cámara →100 ms
Obtener una captura de la cámara
Obtener matriz de homografía con esa captura → Escoger 4 esquinas
while *Verdadero* **do**
    Leer cámara
    Aplicar transformación lineal de homografía
    Convertir imagen a escala de grises
    Aplicar 2 difuminados gaussianos
    Aplicar *thresholding:* adaptativo → cv2.adaptiveThreshold
    Obtener bordes → cv2.Canny
    Obtener contornos desde la lectura de bordes → cv2.findContours
    Filtrar contornos repetidos y reflejos → Centro y rotación de correctos
    Leer ID y pose. Por medio de algoritmo 1
    Mostrar imágenes
    Esperar comando para finalizar lectura → ESC
    if *tecla presionada == ESC* **then** break
end

8.2. Prueba de mediciones

Para asegurar que el algoritmo de medición de pose funciona correctamente se realizó una prueba para determinar el error inducido por la cámara y la homografía. Las cámaras inducen una distorsión de imagen según el lente y la posición desde la cual se realiza la toma, estos dos factores influyen en acumulación de error en ciertas regiones de la mesa de trabajo. Por esta razón se midió 25 puntos con coordenadas conocidas sobre la mesa de trabajo utilizando la visión por computadora. Los puntos fueron distribuidos de manera



(a) Medición de pose por medio de OpenCV.



(b) Identificación de agentes.

Figura 12: Respuesta del algoritmo de visión por computadora.

que se observara el error en cada cuadrante del plano y sobre cada eje. La distribución de

puntos se puede encontrar en la Figura 13, cada punto rojo representa un punto de prueba. El lente de la cámara se encuentra sobre el punto $(22, 24.9)$. Adicionalmente a esto se realizó mediciones de orientación sobre 9 puntos en la mesa de trabajo. Estos están distribuidos de forma similar a los puntos de medición de posición.

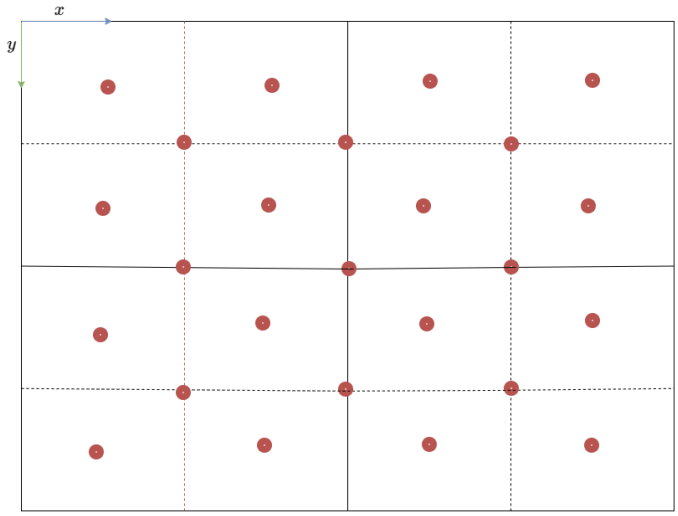


Figura 13: Distribución de puntos de prueba.

Para obtener una idea más acertada sobre la distribución de error de posición y orientación sobre la mesa se generó dos superficies tridimencionales de error. El eje Z de las superficies representa el error obtenido en cada punto del plano. Para la generación de superficie se utilizó la interpolación estándar de Matlab utilizada por el comando *surf()*. Adicionalmente a esto se realizó un análisis estadístico para obtener la media de error y su respectiva varianza. Las superficies de error de posición y orientación se pueden encontrar en las Figuras 14 y 15, respectivamente.

Parámetro	Error medio (cm)	Desviación estándar (cm)
x	0.12 cm	0.07
y	0.09 cm	0.06
θ	0.56°	1.03°

Cuadro 1: Media de error de posición y orientación.

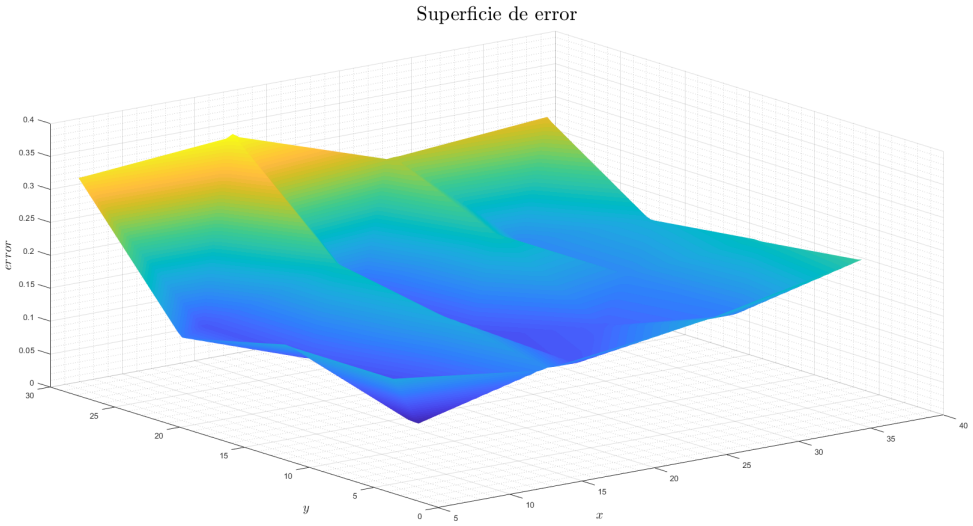


Figura 14: Error de posición sobre la mesa de trabajo.

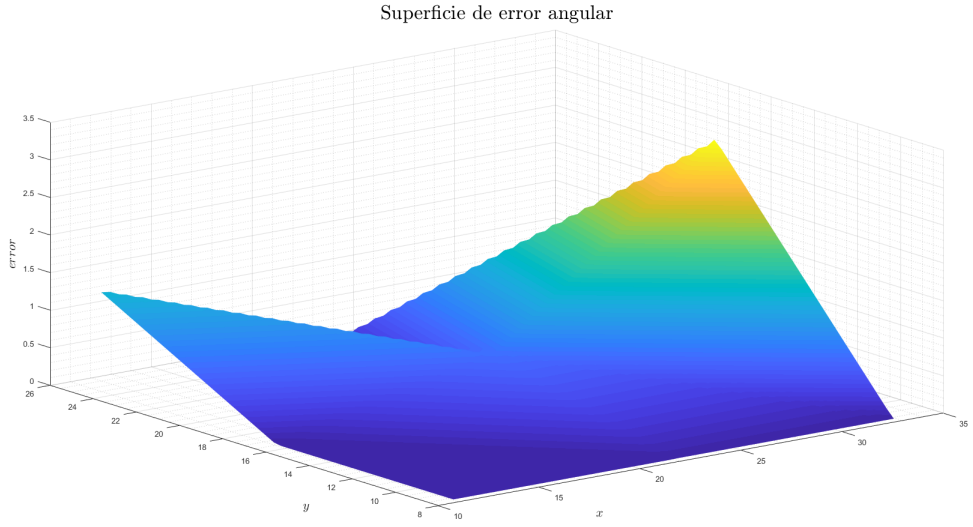


Figura 15: Error de orientación sobre la mesa de trabajo.

A continuación se pueden encontrar imágenes captadas por el algoritmo de visión por computadora durante el proceso de lectura de posición y orientación. El primer paso para llevar a cabo las pruebas es realizar una homografía de la imágen capturada por la cámara. Esto se puede encontrar en la Figura 16. El algoritmo de homografía obtiene los 4 puntos de las esquinas para luego calcular la matriz de transformación homográfica. En la Figura 17 se puede encontrar los puntos medidos por el algoritmo de visión por computadora. El algoritmo pinta sobre cada punto medido un círculo rojo. También se puede encontrar en la Figura 18 el resultado de la imagen captada durante la medición de ángulo.

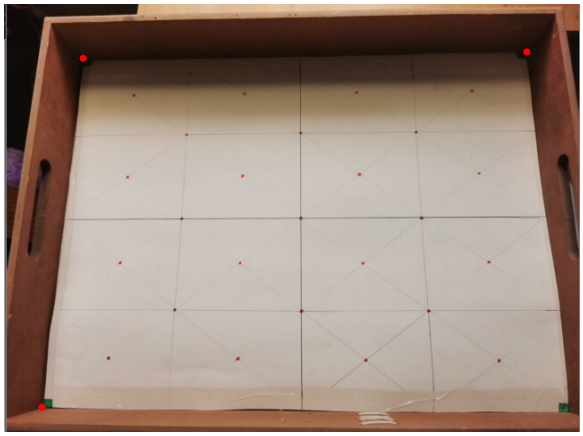


Figura 16: Calibración de mesa de trabajo.

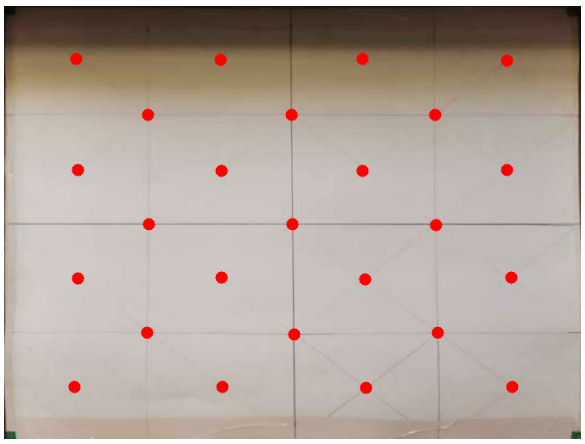


Figura 17: Realización de las mediciones.

El objetivo de esta prueba fue asegurar que las mediciones realizadas por el algoritmo de visión por computadora 9 funcionaran correctamente y no indujeran errores considerables en el sistema. Como se menciona anteriormente, la forma en que se realizó la prueba fue definiendo puntos de medición con coordenadas conocidas y colocando identificadores con orientación previamente medida sobre la mesa de trabajo. Estas fueron medidas una segunda vez por medio de la cámara y el algoritmo generado. Estas mediciones se realizaron tres veces a lo largo del día para asegurar una correcta medición con diferentes niveles de luz. Las tomas fueron tomadas a las 09:00, 13:00 y 17:00.

Como era esperado, el algoritmo de visión por computadora realizó las mediciones con cierto error en cada punto. Como se puede observar en las Figuras 14 y 15 el error fue

Figura 18: Mediciones de orientación.

máximo en la parte de la mesa más alejada a la cámara. Adicionalmente esta parte de la mesa es la que presenta más sombras a lo largo del día. Esto se puede observar en las Figuras 17 y 18. Las dimensiones de la mesa de pruebas también se presentaron como limitaciones para la prueba, pues no se pudo realizar mediciones de orientación con más identificadores. Estos, en comparación con la mesa, tienen un tamaño exagerado y no se pueden distribuir libremente sobre el espacio de trabajo. En ambos casos, medición de posición y orientación, se presentó la misma distribución de error. Esto indica que el error inducido por la cámara y el algoritmo es predecible a lo largo de la mesa de pruebas.

Las pruebas se realizaron exitosamente, tomando en cuenta las limitantes en dimensiones y de equipo. Como se puede observar en el Cuadro 1, el error máximo presentado por el sistema de visión por computadora en la prueba de posición fue de $4mm$, mientras que el error medio fue de $1.7mm$ con una desviación estándar de $0.7mm$. El eje que más error presentó fue el eje horizontal, x, con un error medio de $1.2mm$, mientras que el error del eje vertical, y, presentó una media de $0.9mm$. Al tomar en cuenta las dimensiones de la placa *PCB* realizada por Pablo Prado, las cuales fueron de $52 \times 55mm$, el error máximo obtenido ($4mm$) representó un 7.70 % del largo y 7.27 % del ancho del robot. El error medio representó, por su parte, un 3.27 % del largo del agente. Por otra parte el error angular medio fue de 0.56° con una varianza de 1.03°.

Una fuente de error considerable en el experimento fue la iluminación dispareja sobre el plano de imagen. Esto provoca que el *thresholding:* variable tuviese más dificultad para realizar el proceso de filtrado. Otro aspecto que influyó negativamente en el rendimiento del sistema fue la posición y distorsión provocada por el lente de la cámara. De preferencia se busca que el lente de la cámara se coloque exactamente sobre el centro del plano de imagen, de esa forma el error se concentra en las esquinas del plano. Sin embargo, la fuente de error más significativa para el experimento fue la incertidumbre de los instrumentos de medición utilizados para realizar las mediciones físicas sobre el plano de la mesa de pruebas. Los instrumentos utilizados fueron una cinta métrica y un transportador. Esto representa un error sistemático en el experimento, ya que incertidumbres de los instrumentos son: $0.5mm$ y 0.5°, respectivamente.

Al analizar los resultados del Cuadro 1 se puede determinar que el error de posición y de

orientación medido, el cual representó menos del 10 % de las dimensiones de los agentes, fue adecuado para luego ser utilizado en combinación con otro tipo de sensado, de esta forma se puede obtener la posición y orientación reales de cada robot. Si se utiliza una cámara con suficiente resolución el algoritmo tiene la capacidad para realizar mediciones más exactas que aquellas realizadas por medio de instrumentos físicos, tales como la cinta métrica y el transportador utilizados. Al tomar todo lo anterior en cuenta se acepta la hipótesis planteada en el diseño experimental de la prueba de visión por computadora en la sección 7.2.1 del capítulo 7. Sin embargo, se debe de tomar en cuenta que el sistema debe de operar bajo condiciones óptimas de iluminación y que la distorsión provocada por el lente de la cámara debe de ser mínima.

CAPÍTULO 9

Estimación de pose

En este capítulo se explica el desarrollo de la odometría: correspondiente a los Bitbots. La aproximación de la posición de los robots es una parte fundamental para el desarrollo del proyecto, pues se integran los dos sensores de pose utilizados en el proyecto, la cámara y los encoders. Cada robot Bitbot cuenta con dos motores stepper en cada rueda, estos se pueden utilizar como encoders digitales al tener en cuenta que el movimiento se produce través de *pasos* producidos por el software del microcontrolador. El software debe de ser capaz de contar los pasos realizados en cada uno de los motores. El otro sensor utilizado es la cámara, la cual se utiliza en este caso como un GPS:. Ambos sensores retornan una medición de pose, sin embargo la cámara presenta mediciones lentas pero exactas y los encoders mediciones defectuosas pero rápidas. La estimación de pose pretende integrar las mediciones de ambos sensores y aprovecharse de las características positivas de cada uno. Para realizarla se planteó el uso de un filtro de Kalman extendido en tiempo discreto, este combina los estimados de pose obtenidos por la odometría y el algoritmo de visión por computadora. Las mediciones obtenidas por la prueba de visión por computadora del capítulo 8 serán utilizadas en el desarrollo del filtro de Kalman, pues este necesita conocer las varianzas de cada una de las mediciones retornadas por los sensores. Estas se pueden encontrar en el Cuadro 1.

En la Figura 19 se puede observar el diagrama de bloques correspondiente al control de posición, incluyendo la fusión de sensores para la aproximación del vector de pose $\hat{\boldsymbol{\xi}} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{\theta} \end{bmatrix}$.

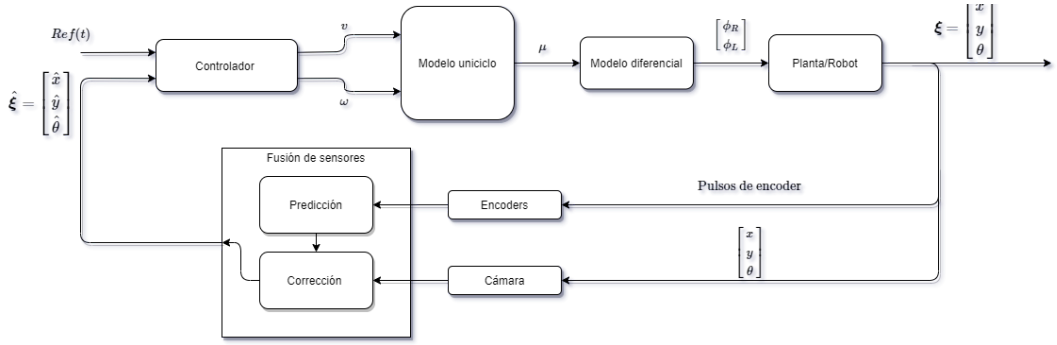


Figura 19: Diagrama de bloques Bitbot.

9.1. Algoritmo del filtro de Kalman extendido

Para obtener el vector de pose aproximado $\hat{\boldsymbol{\xi}}$ se implementa un filtro de Kalman extendido en tiempo discreto. De primero se deben de identificar las mediciones correspondientes a cada una de las partes del filtro. Este observador se separa típicamente en dos etapas: predicción y corrección. La etapa de predicción se realiza utilizando las mediciones obtenidas de los encoders, mientras que la etapa de corrección se lleva a cabo utilizando las mediciones de la cámara. Estas corresponden a las mediciones de proceso y observación, respectivamente. El algoritmo implementado corresponde al algoritmo planteado en la sección 6.5 del marco teórico. Sin embargo en el algoritmo 3 se presenta el pseudo-código para la implementación del mismo.

El algoritmo requiere conocer las varianzas de las lecturas de proceso y de observación. Las varianzas de proceso fueron calculadas de la siguiente forma:

$$\sigma_{v1} = \Delta t \cdot r_{llantas} \cdot \omega_{max}, \tag{52}$$

$$\sigma_{v2} = \frac{2^{\circ}\pi}{180}. \tag{53}$$

Se debe de tener en cuenta que se sobreestima el la varianza del error angular medido por los encoders, pues se desconoce cuánto varía el error angular medido. Esto se debe a la forma en que los encoders/motores stepper obtienen sus mediciones. Por otra parte, las desviaciones estándar utilizadas para las lecturas de observación fueron obtenidas a partir de los resultados de la prueba de visión por computadora. Estas se pueden encontrar en el Cuadro 1. Los valores utilizados son:

$$\sigma_{w1} = 7mm, \tag{54}$$

$$\sigma_{w2} = 6mm, \tag{55}$$

$$\sigma_{w3} = 1.03^{\circ}. \tag{56}$$

A partir de estas desviaciones se obtienen las matrices de covarianza para las mediciones de proceso y de observación, las cuales se implementaron de la siguiente forma:

$$Q_v = \begin{bmatrix} \sigma_{v1} & 0 \\ 0 & \sigma_{v2} \end{bmatrix} = \begin{bmatrix} 7.02 \times 10^{-3} & 0 \\ 0 & 3.49 \times 10^{-2} \end{bmatrix} \tag{57}$$

$$Q_w = \begin{bmatrix} \sigma_{w1} & 0 & 0 \\ 0 & \sigma_{w2} & 0 \\ 0 & 0 & \sigma_{w3} \end{bmatrix} = \begin{bmatrix} 7 \times 10^{-3} & 0 & 0 \\ 0 & 6 \times 10^{-3} & 0 \\ 0 & 0 & 17.97 \times 10^{-3} \end{bmatrix} \tag{58}$$

Por último, se debe de tener en cuenta que se deben de inicializar la matriz de covarinza de error de estimación,$\mathbf{P}_{k-1|k-1}$ en $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$. Mientras que $\mathbf{P}_{k|k}$ se debe de definir un error según la seguridad en la condición inicial. En este caso se definió $\sigma_e = 0.001$. Este error se utilizará para definir $\mathbf{P}_{k|k} = \sigma_e^2 \mathbf{I}_{3x3}$.

Algorithm 3: Implementación filtro de Kalman extendido

input : Lectura de cámara, lectura de encoders
output: Aproximación del vector de pose

`// Inicialización de variables`
$\hat{\mathbf{x}}_0 \leftarrow \begin{bmatrix} x_0 & y_0 & \theta_0 \end{bmatrix}$
$\hat{\mathbf{x}}_{k-1|k-1} \leftarrow \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
$\hat{\mathbf{x}}_{k|k} \leftarrow \hat{\mathbf{x}}_0$
$y_{k-1} \leftarrow 0$
while *True* **do**
  `// Lectura de sensores`
  Leer de encoder `// Medición de proceso`
  Calcular incrementos de posición angular y lineal $\rightarrow \delta\rho, \delta\theta$
  Calcular jacobianos del modelo de odometría: $\rightarrow A_d, F_d$
  $\hat{\mathbf{x}}_{k-1|k-1} \leftarrow \hat{\mathbf{x}}_{k|k}$
  $\mathbf{P}_{k-1|k-1} \leftarrow \mathbf{P}_{k|k}$
  `/* Predicción */`
  $\hat{\mathbf{x}}_{k|k-1} \leftarrow \mathcal{F}(\hat{\mathbf{x}}_{k-1|k-1}, \delta\rho, \delta\theta)$
  $\mathbf{P}_{k|k-1} \leftarrow A_d \mathbf{P}_{k-1|k-1} A_d^T + F_d Q_w F_d^T$
  if *Nueva lectura de cámara* **then**
    Leer de cámara `// Medición de observación`
    `/* Corrección */`
    $y \leftarrow \begin{bmatrix} x_{cam} \\ y_{cam} \\ \theta_{cam} \end{bmatrix}$
    $\mathbf{L}_k \leftarrow \mathbf{P}_{k|k-1} C_d^T \left(Q_v + C_d \mathbf{P}_{k|k-1} C_d^T\right)^{-1}$
    $\hat{\mathbf{x}}_{k|k} \leftarrow \hat{\mathbf{x}}_{k|k-1} + \mathbf{L}_k \left(y - C_d * \hat{\mathbf{x}}_{k|k-1}\right)$
    $\mathbf{P}_{k|k} \leftarrow \mathbf{P}_{k|k-1} - L_k C_d \mathbf{P}_{k|k-1}$
    $y_{k-1} \leftarrow y$
  else
    $y \leftarrow y_{k-1}$
    $\mathbf{L}_k \leftarrow \mathbf{P}_{k|k-1} C_d^T \left(Q_v + C_d \mathbf{P}_{k|k-1} C_d^T\right)^{-1}$
    $\hat{\mathbf{x}}_{k|k} \leftarrow \hat{\mathbf{x}}_{k|k-1} + \mathbf{L}_k \left(y - C_d * \hat{\mathbf{x}}_{k|k-1}\right)$
    $\mathbf{P}_{k|k} \leftarrow \mathbf{P}_{k|k-1} - L_k C_d \mathbf{P}_{k|k-1}$
  end
  $\hat{\boldsymbol{\xi}}_k \leftarrow \hat{\mathbf{x}}_{k|k}$
end

9.2. Prueba de odometría:

La prueba planteada en la sección 7.2.3 se realizó en el simulador Webots con el modelo del robot diferencial **Pioneer 3dx**. Se utilizó este robot, pues al ser un robot diferencial se controla de la misma manera que los Bitbots. En el capítulo 10 se discute más en detalle las similitudes. La hipótesis planteada fue: el algoritmo implementado obtendrá correctamente la posición y orientación del robot Bitbot.

La simulación en Webots tiene un tiempo de corrida de $t_f = 25s$ y un período de muestreo de $\Delta t = 50ms$. La cantidad de puntos muestreados es $Puntos = \frac{tf}{\Delta t} = 500$.

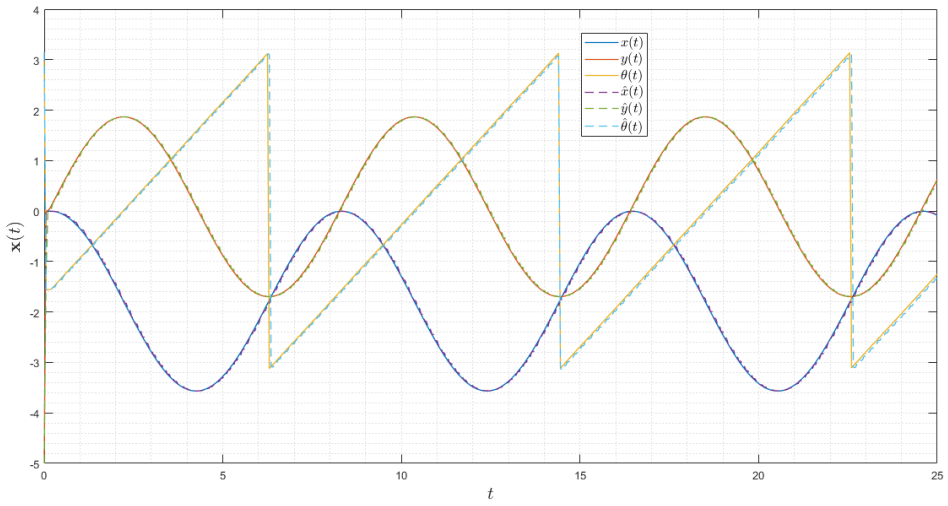


Figura 20: Salida del observador de estados.

Se observa en la Figura 20 que la aproximación de pose retornada por el algoritmo implementado es casi idéntica a la pose verdadera del robot. Según el Cuadro 2 la medición con un mayor error acumulado durante los 500 muestreos es la orientación del robot. Esta presentó un error acumulado de $0.5589rad$ o $31.98°$. Los errores acumulados obtenidos a partir de las posiciones del robot fueron mínimos, pues un error acumulado de $1.3m$ en 500 muestreos da como resultado $2.64 \times 10^{-3}m = 2.64mm$ de error en cada muestreo. Este error por muestreo equivale a un 3.8 % respecto a las dimensiones del robot ($52x55mm$).

Parámetro	Error acumulado
$x(t)$	$0.0391m$
$y(t)$	$1.3219m$
$\theta(t)_{rad}$	$0.5583rad$
$\theta(t)_{deg}$	$31.98°$

Cuadro 2: Error acumulado en mediciones.

Se observa en las Figuras 20 y 21 que el observador de estados de Kalman presenta un correcto seguimiento de la pose real y de las lecturas de la cámara. Esto se debe a que la cámara presenta varianzas pequeñas, lo cual indica que el sensor tiende a ser ideal.

Las predicciones entregadas por las mediciones de proceso son suficientemente buenas para entregar una aproximación de pose buena como salida del filtro.

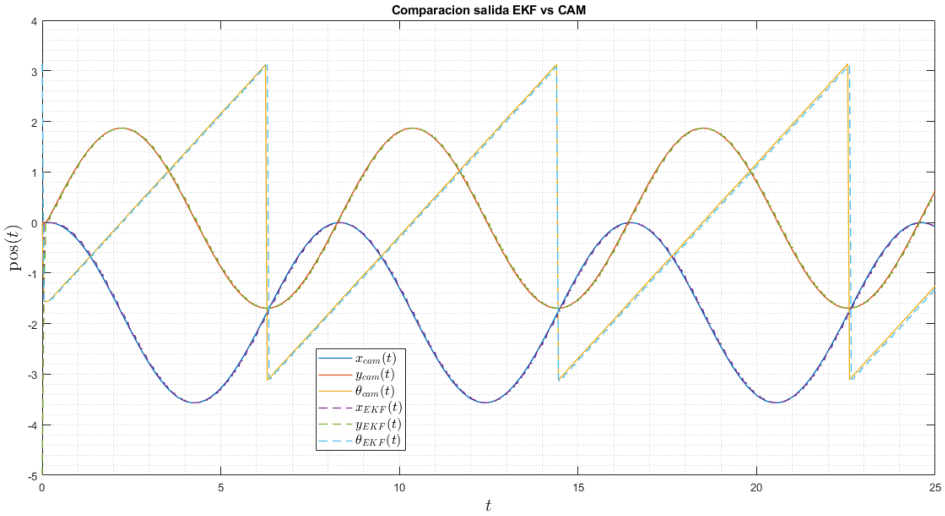
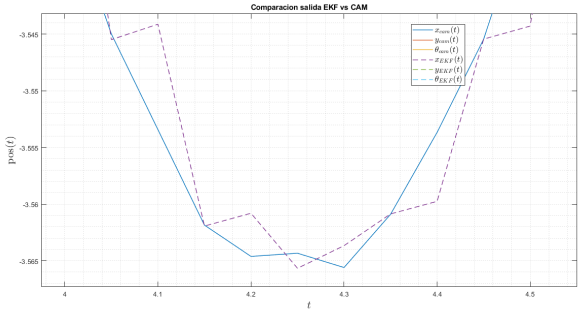
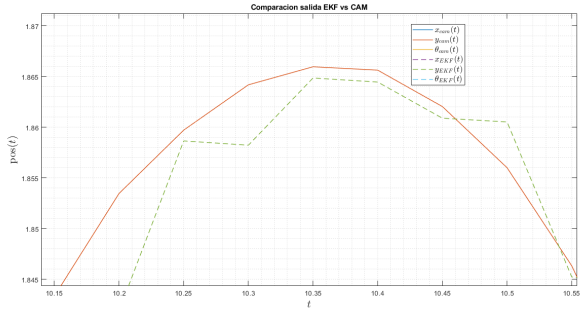


Figura 21: Comparación lectura de cámara y aproximación de Kalman.



(a) Comparación lectura de cámara y aproximación de pose, eje x.



(b) Comparación lectura de cámara y aproximación de pose, eje y.

Figura 22: Comparación lecturas de cámara y aproximaciones de pose en cada eje.

En la Figura 22 se puede observar dos aspectos importantes de la prueba de odometría:. En primer lugar, se observa que el algoritmo realiza una aproximación *errónea* en aquellos ciclos en los cuales no se tiene una lectura de la cámara. Seguido de esta medición *errónea* se realiza una corrección cuando se vuelve a obtener la lectura de la cámara. Segundo, se observan las pequeñas varianzas en las lecturas de la cámara. Este fenómeno se observa mejor en la Figura 22a, pues las mediciones de la cámara respecto al eje x presenta irregularidades. Esto se debe a que, aunque pequeña, la varianza de las lecturas respecto al eje x es mayor a la varianza sobre el eje y.

Según los resultados presentados anteriormente se puede afirmar que la pose encontrada por el algoritmo de fusión de sensores es suficientemente buena para ser utilizada en el algoritmo de control de los Bitbots. Sin embargo, se debe de tomar en cuenta que las lecturas de la cámara deben de tener una frecuencia de aproximadamente 10Hz y que no debe de existir mucho deslizamiento en las llantas del robot. Aunque el algoritmo toma en cuenta cierta cantidad de deslizamiento, si este llegara a ocurrir en exceso las aproximaciones cuando no se tengan las lecturas de la cámara presentarán aún más error que el error actual. También se debe de tomar en cuenta que la cámara a utilizar debe de presentar varianzas similares a la utilizada durante el experimento para poder asegurar que el algoritmo funciona como se presenta en las Figuras anteriormente mostradas.

El algoritmo del filtro de Kalman es una opción robusta para realizar la fusión de sensores de un robot móvil diferencial. Es una excelente forma de combinar dos señales ruidosas y obtener una aproximación suficientemente buena como para luego aplicarle control a la misma. Por otra parte este observador de estado presenta una fuerte carga computacional para el microcontrolador del robot. Esto se debe a la gran cantidad de operaciones matriciales que el algoritmo exige. Este requiere gran capacidad de procesamiento como también gran cantidad de memoria volátil, pues en las matrices se trabaja con variables tipo *float* o *double*, según la exactitud con la que se requiera. Por esta situación se recomienda utilizar un microcontrolador de 32 bits con procesador de punto flotante. El microcontrolador utilizado en el proyecto se ajusta perfectamente a los requerimientos del algoritmo, pues puede ser programado utilizando *Micropython*, también tiene un tamaño de registro de 32 bits al igual que procesador de punto flotante.

CAPÍTULO 10

Control de posición

En este capítulo se explica el desarrollo de los controladores utilizados en el proyecto Bitbots para asegurar un correcto control punto a punto. El control punto a punto de los robots es básico para realizar cualquier tarea con los robots. El algoritmo de control debe de ser capaz de trasladar el robot desde su posición inicial hasta la posición final correctamente. En la Figura 23 se presenta el diagrama de bloques correspondiente al lazo de control de los robots. Los controladores fueron diseñados y probados utilizando Matlab y luego fueron probados en la plataforma de simulaciones Webots.

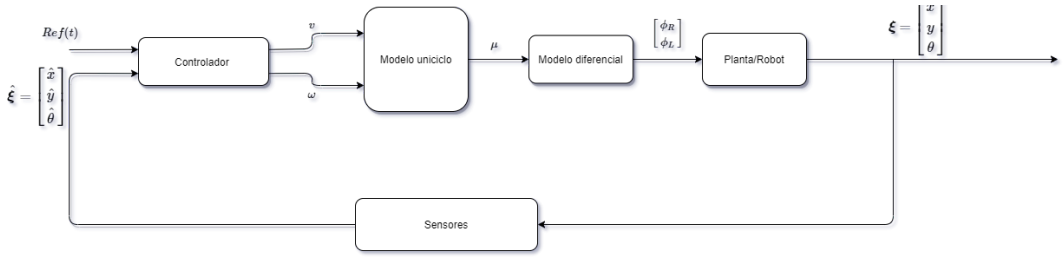


Figura 23: Lazo de control de Bitbots

10.1. Controladores utilizados

En la prueba de control de pose se plantó utilizar diferentes esquemas de control para luego determinar qué controlador se adapta mejor a las necesidades del proyecto. Se analizó el comportamiento de cinco controladores diferentes:

1. PID:.
2. PID: con acercamiento exponencial.
3. LQR:.
4. LQI:.
5. Controlador no lineal.

La implementación de los diferentes tipos de controladores, al igual que los parámetros utilizados se describen en los próximos algoritmos. Todos los controladores retornan como salida la velocidad lineal $v(t)$ y la velocidad angular $\omega(t)$. Estos utilizaron como entrada el vector de estados del robot $\boldsymbol{\xi}$ y el punto de referencia $\boldsymbol{\xi}_g$. Para calcular las velocidades de cada motor, ϕ_R, ϕ_L, se utilizaron las ecuaciones (59) y (60). Adicionalmente se encuentra un parámetro L en las ecuaciones el cual se refiere a la distancia entre la rueda y el centro del robot.

$$\phi_R = \frac{v + \omega L}{R}, \tag{59}$$

$$\phi_L = \frac{v - \omega L}{R}. \tag{60}$$

10.1.1. PID:

Para implementar un controlador PID: (*Proportional, Integral, Derivative*) a un robot diferencial, se debe de controlar por separado la posición y la orientación. Por esa razon se implementaron dos controladores PID: en paralelo. La implementación de este controlador responde a las ecuaciones (61) y (62). Los parámetros utilizados se encuentran en los Cuadros 3a y 3b. El pseudocódigo, en donde se muestra la implementación del controlador se puede encontrar a detalle en el algoritmo 4. El comportamiento del controlador se puede observar en la Figura 24.

Parámetro	Valor utilizado
K_P	1
K_I	1×10^{-3}
K_D	0.1

(a) Parámetros PID: de posición.

Parámetro	Valor utilizado
K_P	1
K_I	1×10^{-3}
K_D	0.1

(b) Parámetros PID: de orientación.

Cuadro 3: Parámetros para controladores PID:.

$$v(t) = 1e_p + 1 \times 10^{-3} \int_0^t e_p(\tau)d\tau + 0.1\dot{e}_p, \tag{61}$$

$$\omega(t) = 1e_o + 1 \times 10^{-3} \int_0^t e_o(\tau)d\tau + 0.1\dot{e}_o. \tag{62}$$

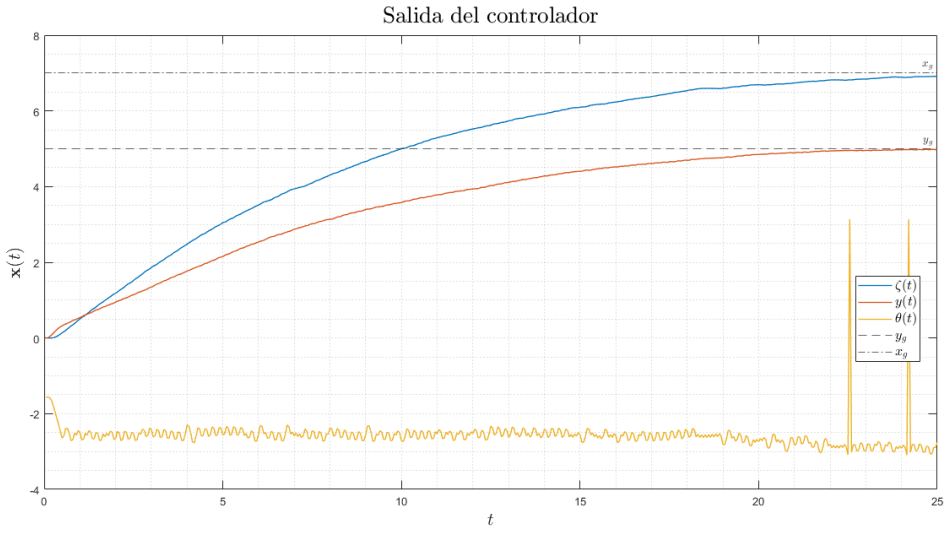


Figura 24: Comportamiento del controlador PID:.

Algorithm 4: Implementación de controladores PID:

input : Lectura de pose, meta
output: Velocidades en cada motor
// Inicialización de variables
$e_P \leftarrow 0$ // Error actual de posición
$E_P \leftarrow 0$ // Error acumulado de posición
$\dot{e}_P \leftarrow 0$ // Derivada del error de posición
$e_{P-1} \leftarrow 0$ /* Error anterior de posición */

$e_O \leftarrow 0$ // Error actual de orientación
$E_O \leftarrow 0$ // Error acumulado de orientación
$\dot{e}_O \leftarrow 0$ // Derivada del error de orientación
$e_{O-1} \leftarrow 0$ /* Error anterior de Orientación */
while *Verdadero* **do**
Aproximación de pose $\rightarrow \hat{\boldsymbol{\xi}}_i$
$x \leftarrow \boldsymbol{\xi}_x$
$y \leftarrow \boldsymbol{\xi}_y$
$\theta \leftarrow \boldsymbol{\xi}_\theta$
/* Cálculo de error */
$e \leftarrow \begin{bmatrix} x_g - x \\ y_g - y \end{bmatrix}$
$\theta_g = \text{atan2}\left(\frac{x_g - x}{y_g - y}\right)$
/* PID: Posición */
$e_P \leftarrow \|e\|$
$\dot{e}_P \leftarrow e_P - e_{P-1}$
$E_P \leftarrow E_P + e_P$
$e_{P-1} \leftarrow e_P$

$v \leftarrow K_{PP} e_P + K_{IP} E_P + K_{DP} \dot{e}_P$
/* PID: Orientación */
$e_O \leftarrow \theta_g - \theta$
$\dot{e}_O \leftarrow e_O - e_{O-1}$
$E_O \leftarrow E_O + e_O$
$e_{O-1} \leftarrow e_O$

$\omega \leftarrow K_{PO} e_O + K_{IO} E_O + K_{DO} \dot{e}_O$

$u \leftarrow \begin{bmatrix} v \\ \omega \end{bmatrix}$

$\phi_R \leftarrow \frac{v + \omega L}{R}$
$\phi_L \leftarrow \frac{v - \omega L}{R}$
end

10.1.2. PID: con acercamiento exponencial

El controlador PID: puede presentar problemas a la hora de alcanzar la meta. Este tiende a hacer espirales alrededor de la meta, por esta razón se implementa un controlador PID: para la orientación y para la posición se utiliza un acercamiento exponencial dado por la ecuación (63). El controlador de orientación responde a la ecuación (62). En la Figura 25 se puede encontrar la respuesta del controlador al imponer como referencia una meta fija.

$$v(t) = \frac{1.2(1 - e^{-0.5e_p^2})}{e_p}. \tag{63}$$

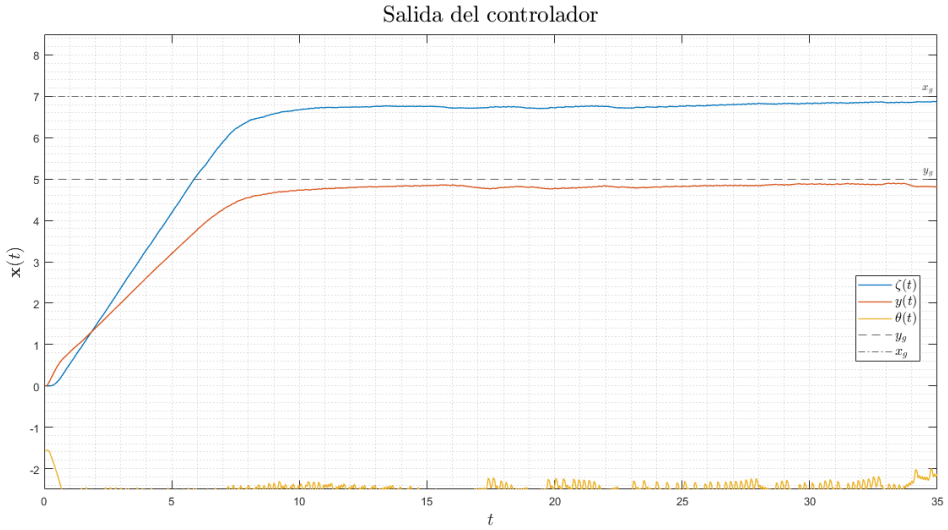


Figura 25: Respuesta del controlador PID: con acercamiento exponencial.

Algorithm 5: Implementación de controlador PID: y acercamiento exponencial

input : Lectura de pose, meta
output: Velocidades en cada motor

`/* Inicialización de variables */`
$e_O \leftarrow 0$ `// Error actual de orientación`
$E_O \leftarrow 0$ `// Error acumulado de orientación`
$\dot{e}_O \leftarrow 0$ `// Derivada del error de orientación`
$e_{O-1} \leftarrow 0$ `/* Error anterior de Orientación */`
$\alpha \leftarrow 0.5$ `/* Constante de acercamiento exponencial */`
while *True* **do**
  Aproximación de pose $\rightarrow \hat{\boldsymbol{\xi}}$
  $x \leftarrow \hat{\boldsymbol{\xi}}_x$
  $y \leftarrow \hat{\boldsymbol{\xi}}_y$
  $\theta \leftarrow z\hat{e}ta_\theta$
  `/* Cálculo de error */`
  $e \leftarrow \begin{bmatrix} x_g - x \\ y_g - y \end{bmatrix}$
  $\theta_g = \text{atan2}\left(\frac{x_g - x}{y_g - y}\right)$
  `/* PID: Orientación */`
  $e_O \leftarrow \theta_g - \theta$
  $\dot{e}_O \leftarrow e_O - e_{O-1}$
  $E_O \leftarrow E_O + e_O$
  $e_{O-1} \leftarrow e_O$
  $\omega \leftarrow K_{PO} e_O + K_{IO} E_O + K_{DO} \dot{e}_O$
  `/* Acercamiento exponencial */`
  $e_P \leftarrow \|e\|$
  $K_{exp} \leftarrow v_0 \frac{1 - e^{-\alpha e_P^2}}{e_P}$
  $v \leftarrow K_{exp} e_P$
  $u \leftarrow \begin{bmatrix} v \\ \omega \end{bmatrix}$
  $\phi_R \leftarrow \frac{v + \omega L}{R}$
  $\phi_L \leftarrow \frac{v - \omega L}{R}$
end

10.1.3. LQR:

El controlador LQR: es controlador moderno LTI:. Para implementar el controlador LQR: se debe de realizar una linealización alrededor de un punto de operación, sin embargo surgen problemas cuando se realiza en la posición $\boldsymbol{\xi}_{ss} \rightarrow \dot{\boldsymbol{\xi}} = 0$. Por esta razón es importante definir una desviación de la posición del robot l. Esta desviación se utilizó $l = 0.01m = 1cm$. Adicionalmente a esto se debe de escoger una matriz $\mathcal{K}_{\mathcal{LQR}}$ para realizar el control sobre el error de posición. El controlador responde a la ecuación (64). Después de obtener la respuesta del controlador se debe de aplicar un difeomorfismo: para calcular la velocidad lineal $v(t)$ y la velocidad angular $\omega(t)$. La matriz de difeomorfismo: se denota como $\mathcal{M}$. La matriz $\mathcal{K}_{LQR}$ se implementó $\mathcal{K}_{LQR} = \begin{bmatrix} 3 & 0 \\ 3 & 0 \end{bmatrix}$. Adicionalmente se observa en la Figura 26 la respuesta del controlador cuando se le impone una meta fija.

$$\boldsymbol{\mu}(t) = -\begin{bmatrix} 3 & 0 \\ 3 & 0 \end{bmatrix} \begin{bmatrix} x - x_g \\ y - y_g \end{bmatrix}, \tag{64}$$

$$\mathbf{u} = \mathcal{M}(l, \theta)\boldsymbol{\mu}. \tag{65}$$

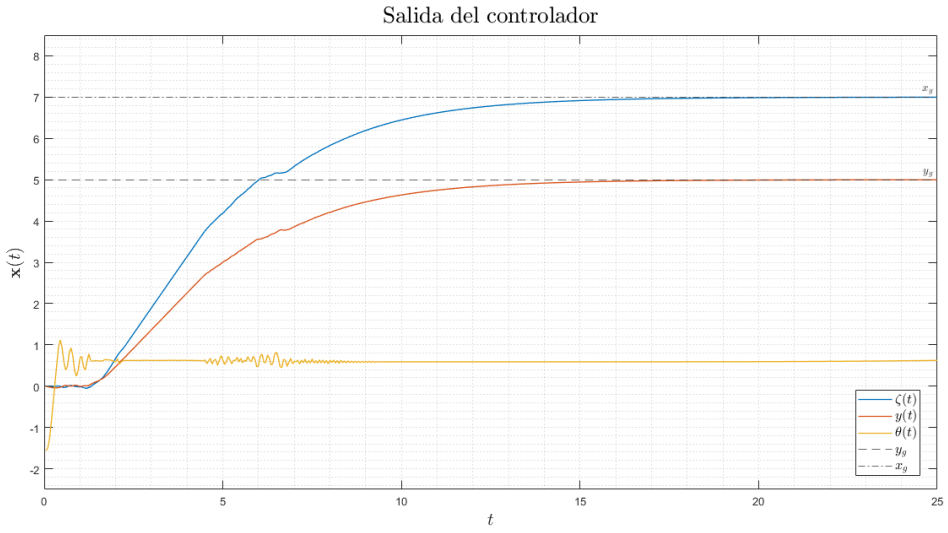


Figura 26: Controlador LQR: con meta fija.

Algorithm 6: Implementación de controlador LQR:

input : Lectura de pose, meta
output: Velocidades en cada motor

`/* Inicialización de variables */`

$\boldsymbol{\mu} \leftarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\mathcal{K}_{LQR} \leftarrow \begin{bmatrix} 3 & 0 \\ 3 & 0 \end{bmatrix}$

$\mathbf{e} \leftarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

while *Verdadero* **do**

Aproximación de pose $\rightarrow \hat{\boldsymbol{\xi}}$

$x \leftarrow \hat{\boldsymbol{\xi}}_x$

$y \leftarrow \hat{\boldsymbol{\xi}}_y$

`/* Cálculo de error */`

$e \leftarrow \begin{bmatrix} x_g - x \\ y_g - y \end{bmatrix}$

`/* Control de Posición */`

$\boldsymbol{\mu} \leftarrow -\mathcal{K}_{LQR}\mathbf{e}$

$\begin{bmatrix} v \\ \omega \end{bmatrix} \leftarrow \mathcal{M}\boldsymbol{\mu}$

$\mathbf{u} \leftarrow \begin{bmatrix} v \\ \omega \end{bmatrix}$

$\phi_R \leftarrow \frac{v+\omega L}{R}$

$\phi_L \leftarrow \frac{v-\omega L}{R}$

end

10.1.4. LQI:

El controlador LQI: también pertenece a la familia de controladores modernos LTI:. Este controlador resuelve los problemas en estado estable presentados por el controlador LQR:. A diferencia del controlador LQR:, este sí presentó una corrección respecto al error acumulado. Este se representa por medio de $\boldsymbol{\sigma}$, descrito en la ecuación (66). La implementación de este controlador involucra la matriz $\mathcal{K}_{LQI}$, la cual multiplicará el vector de error aumentado $\begin{bmatrix}\mathbf{e}\\ \boldsymbol{\sigma}\end{bmatrix}$. A la salida de este controlador, al igual que el controlador LQR:, se le debe de aplicar un difeomorfismo:, $\mathcal{M}$, para obtener el vector de control $\mathbf{u} = \begin{bmatrix}v\\ \omega\end{bmatrix}$. La respuesta del controlador al imponer una meta fija se puede encontrar en la Figura 27.

$$\sigma = \int_0^t \begin{bmatrix} x(\tau) - x_g(\tau) \\ y(\tau) - y_g(\tau) \end{bmatrix} d\tau, \tag{66}$$

$$\mu(t) = -\begin{bmatrix} 5\sqrt{3} & 0 & 1.5 \\ 5\sqrt{3} & 0 & 1.5 \end{bmatrix} \begin{bmatrix} \mathbf{e}_p \\ \sigma \end{bmatrix}. \tag{67}$$

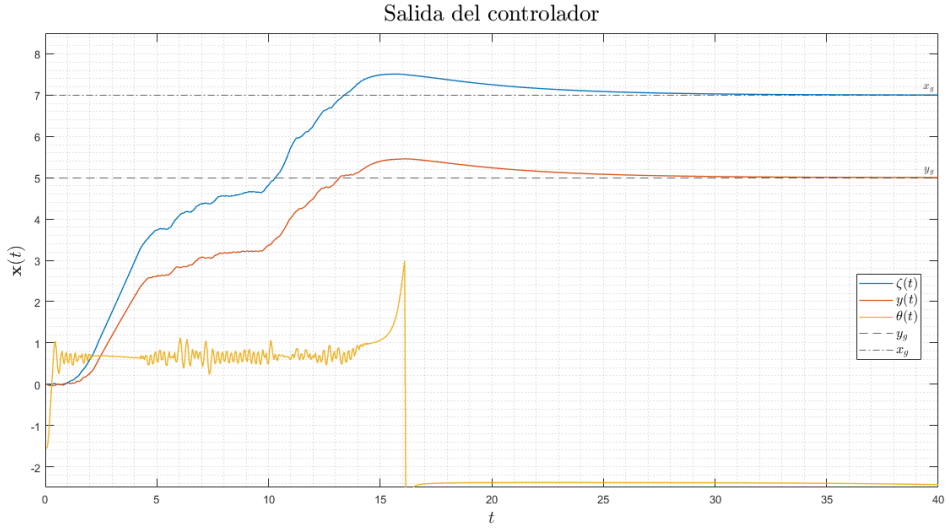


Figura 27: Respuesta del controlador LQI: con meta fija

Algorithm 7: Implementación de controlador LQI:.

input : Lectura de pose, meta
output: Velocidades en cada motor

`/* Inicialización de variables */`

$\mu \leftarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$K_{LQI} \leftarrow \begin{bmatrix} 5\sqrt{3} & 0 & 1.5 \\ 5\sqrt{3} & 0 & 1.5 \end{bmatrix}$

$e \leftarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$E_x \leftarrow 0$

`/* Error acumulado medición en` x `*/`

$E_y \leftarrow 0$

`/* Error acumulado medición en` y `*/`

while *Verdadero* **do**

Aproximación de pose $\rightarrow \hat{\boldsymbol{\xi}}$

$x \leftarrow \hat{\boldsymbol{\xi}}_x$

$y \leftarrow \hat{\boldsymbol{\xi}}_y$

`/* Cálculo de error */`

$e_x \leftarrow x_g - x$

$E_x \leftarrow E_x + e_x$

$e_y \leftarrow y_g - y$

$E_y \leftarrow E_y + e_y$

`/* Control de Posición */`

$\boldsymbol{\mu} \leftarrow \begin{bmatrix} -5\sqrt{3}x - 1.5E_x\Delta t \\ -5\sqrt{3}y - 1.5E_y\Delta t \end{bmatrix}$

$\begin{bmatrix} v \\ \omega \end{bmatrix} \leftarrow \mathcal{M}\mu$

$\mathbf{u} \leftarrow \begin{bmatrix} v \\ \omega \end{bmatrix}$

$\phi_R \leftarrow \frac{v+\omega L}{R}$

$\phi_L \leftarrow \frac{v-\omega L}{R}$

end

10.1.5. Controlador no lineal

El controlador no lineal presentó una corrección de posición y una corrección angular. La ventaja de este controlador sobre los otros es que este asegura llegar al objetivo y estacionarse sobre él. A diferencia del controlador LQR:, este no presentó error en estado estable. Para implementar correctamente el controlador no lineal, se deben de definir tres constantes: $K_\rho = 10$,$K_\alpha = 20$,$K_\beta = -10$. Se debe de tener en cuenta que se debe de calcular el error de posición $\rho = \left|\left|\begin{bmatrix} x_g - x \\ y_g - y \end{bmatrix}\right|\right|$. También se debe de calcular $\alpha = \arctan 2\left(\frac{y_g - y}{x_g - x}\right)$ y $\beta = -\theta - \alpha$. Habiendo calculado lo anterior, se puede encontrar la forma del controlador en las ecuaciones (68) y (69). La respuesta del controlador no lineal simulada en Matlab se puede encontrar en la Figura 28.

$$v(t) = K_\rho \rho, \tag{68}$$

$$\omega = K_\alpha \alpha + K_\beta \beta. \tag{69}$$

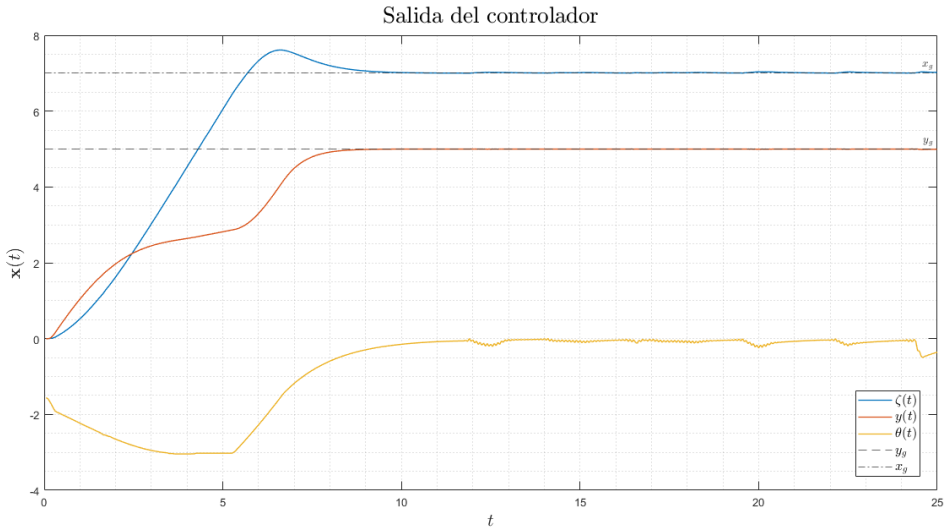


Figura 28: Respuesta del controlador no lineal.

Algorithm 8: Implementación de controlador no lineal

input : Lectura de pose, meta
output: Velocidades en cada motor
`/* Incialización de variables */`
$K_\rho \leftarrow 10$
$K_\alpha \leftarrow 20$
$K_\beta \leftarrow -10$
$\mathbf{e} \leftarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
while *Verdadero* **do**
  Aproximación de pose $\rightarrow \hat{\boldsymbol{\xi}}$
  $x \leftarrow \hat{\boldsymbol{\xi}}_x$
  $y \leftarrow \hat{\boldsymbol{\xi}}_y$
  $\theta \leftarrow \hat{\boldsymbol{\xi}}_\theta$
  `/* Cálculo de error */`
  $\mathbf{e} \leftarrow \begin{bmatrix} x_g - x \\ y_g - y \end{bmatrix}$
  `/* Control de posición */`
  $\rho \leftarrow \|e\|$
  $\alpha \leftarrow \arctan 2 \left(\frac{y_g - y}{x_g - x} \right)$
  $\beta \leftarrow -\theta - \alpha$
  $v \leftarrow K_\rho \rho$
  $\omega \leftarrow K_\alpha \alpha + K_\beta \beta$
  $\mathbf{u} \leftarrow \begin{bmatrix} v \\ \omega \end{bmatrix}$
  $\phi_R \leftarrow \frac{v + \omega L}{R}$
  $\phi_L \leftarrow \frac{v - \omega L}{R}$
end

10.2. Controlador óptimo para Bitbots

La selección de controladores se realizó por medio de una simulación en la plataforma Webots. El robot diferencial utilizado fue el mismo utilizado en la prueba de odometría: en el capítulo 9. A diferencia de la simulación realizada en el capítulo de estimación de pose, en esta simulación se utilizó un controlador de posición para alcanzar una meta fija y, posteriormente, realizar un seguimiento de trayectoria. Este experimento involucra todo lo trabajado en las pruebas anteriores. Para conocer la pose del robot, $\boldsymbol{\xi}$, se utiliza el algoritmo desarrollado en el capítulo 9, se utilizan también las varianzas determinadas en el capítulo 8. De esta forma se trabaja con la pose estimada por los sensores del robot, $\hat{\boldsymbol{\xi}}$. El algoritmo general de estimación de pose y control de pose de los Bitbots responde al pseudocódigo presentado en el algoritmo 9.

Algorithm 9: Algoritmo general de estimación y controlador de pose

input : Meta, lectura de cámara y encoders
output: Velocidades en cada motor
`/* Incialización de variables */`
Inicialización constantes del controlador
Inicialización variables del filtro de Kalman
while *Verdadero* **do**
  `/* Aproximación de pose` $\rightarrow \hat{\boldsymbol{\xi}}$ `*/`
  Predicción de pose $\rightarrow \hat{\boldsymbol{\xi}}_{k|k-1}$
  if *Lectura nueva de la cámara* **then**
    Corrección de estimación de pose $\hat{\boldsymbol{\xi}}_{k|k}$
  else
    $\hat{\boldsymbol{\xi}}_{k|k} \leftarrow \hat{\boldsymbol{\xi}}_{k|k-1}$
  end
  $\hat{\boldsymbol{\xi}} \leftarrow \hat{\boldsymbol{\xi}}_{k|k}$

  `/* Cálculo de error` $\rightarrow e$ `*/`
  $\mathbf{e} \leftarrow \begin{bmatrix} x_g - \hat{x} \\ y_g - \hat{y} \end{bmatrix}$

  `/* Control de posición` $\rightarrow \begin{bmatrix} v \\ \omega \end{bmatrix}$ `*/`
  Control de posición $\rightarrow v$
  Control de orientación $\rightarrow \omega$
  $\mathbf{u} \leftarrow \begin{bmatrix} v \\ \omega \end{bmatrix}$

  `/* Velocidad en las llantas` $\begin{bmatrix} \phi_R \\ \phi_L \end{bmatrix}$ `*/`
  $\phi_R \leftarrow \frac{v+\omega L}{R}$
  $\phi_L \leftarrow \frac{v-\omega L}{R}$
end

10.2.1. Meta fija

El primer experimento realizado fue imponer una meta fija para que el robot la alcance utilizando el algoritmo en donde se integran todos los resultados anteriores. Se realizó la prueba con cada uno de los controladores propuestos y una meta $\boldsymbol{\xi}_g = \begin{bmatrix} 7 \\ 5 \\ \pi \end{bmatrix}$. En las Figuras 29 a 33 se puede apreciar el comportamiento de cada uno de los controladores propuestos. Las condiciones iniciales del robot son: $\boldsymbol{\xi}_0 = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$, $\phi_R = 0$ y $\phi_L = 0$. Se puede observar en las Figuras que todos los controladores alcanzan la meta. La diferencia entre los controladores es el tiempo que les toma alcanzar la meta. Esto se debe al comportamiento natural de cada controlador. Por ejemplo, el controlador PID:, Figura 29, y el controlador PID: con acercamiento exponencial, Figura 30, presentan un tiempo para alcanzar la meta similar. Esto se debe a su similitud en estructura. Sin embargo, la diferencia en tiempo presentada por los controladores modernos LTI:, LQR: y LQI:, con el resto es significativa. En el caso del controlador LQI:, esto se debe a la corrección del overshoot:. En la Figura 33 se encuentra la respuesta del controlador no lineal con la meta descrita anteriormente. Este es el controlador que más rápido alcanza la meta.

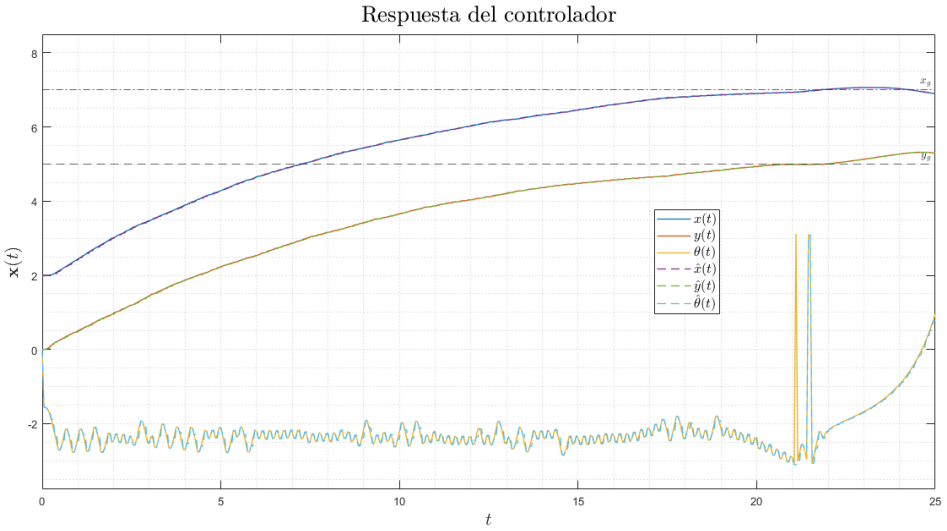


Figura 29: Respuesta controlador PID: con meta fija.

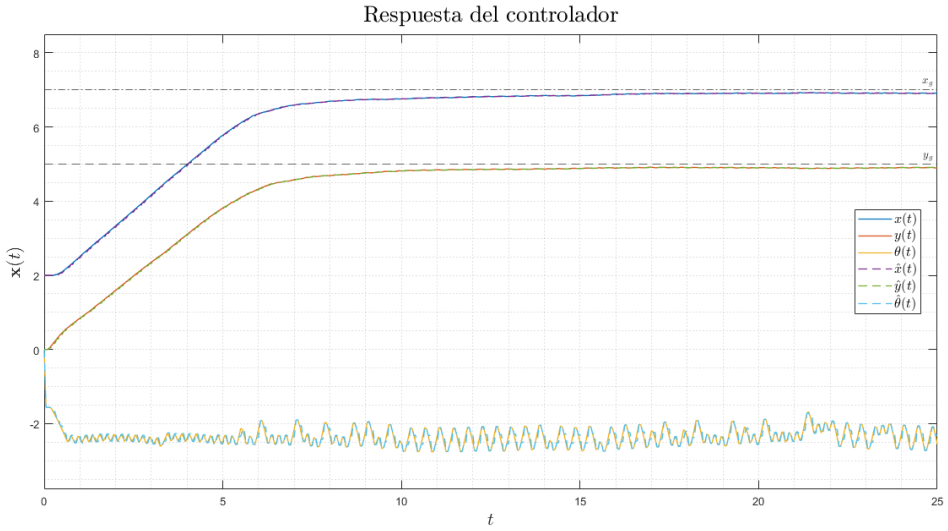


Figura 30: Respuesta controlador PID: con acercamiento exponencial con meta fija.

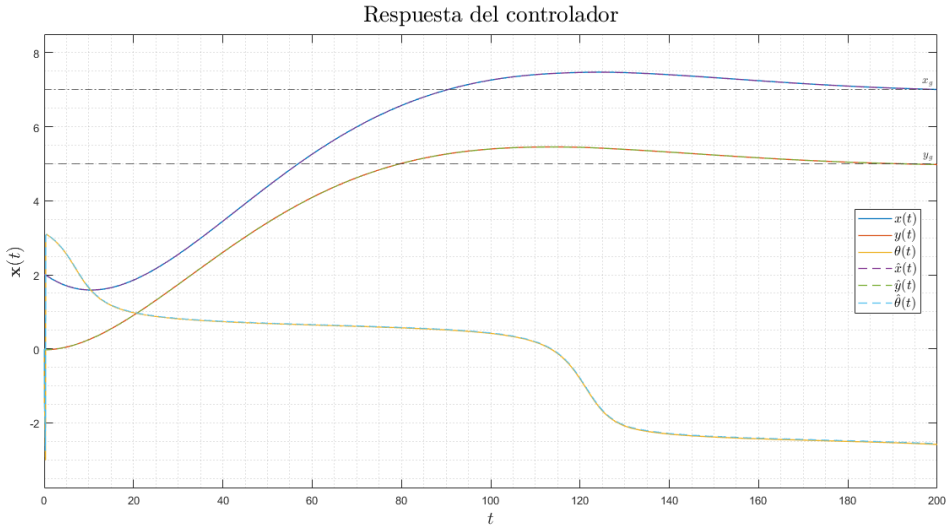


Figura 31: Respuesta controlador LQI: con meta fija.

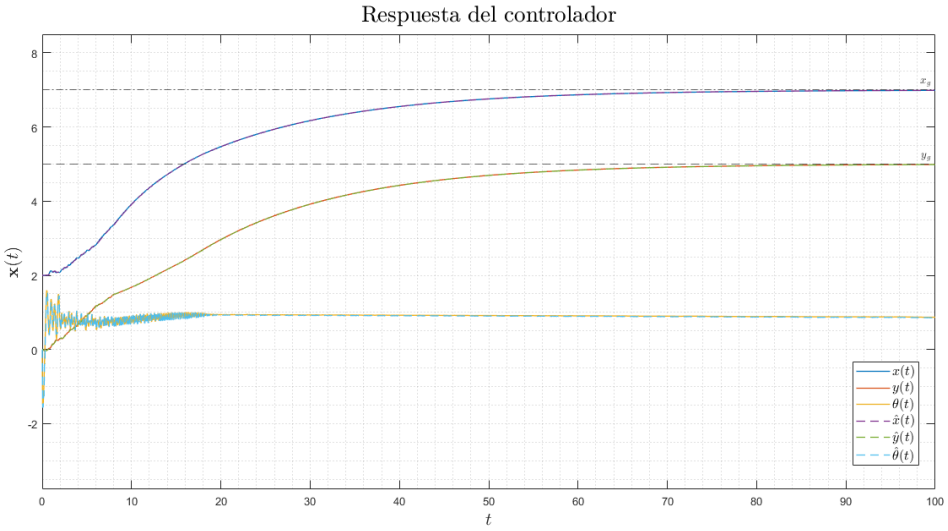


Figura 32: Respuesta controlador LQR: con meta fija.

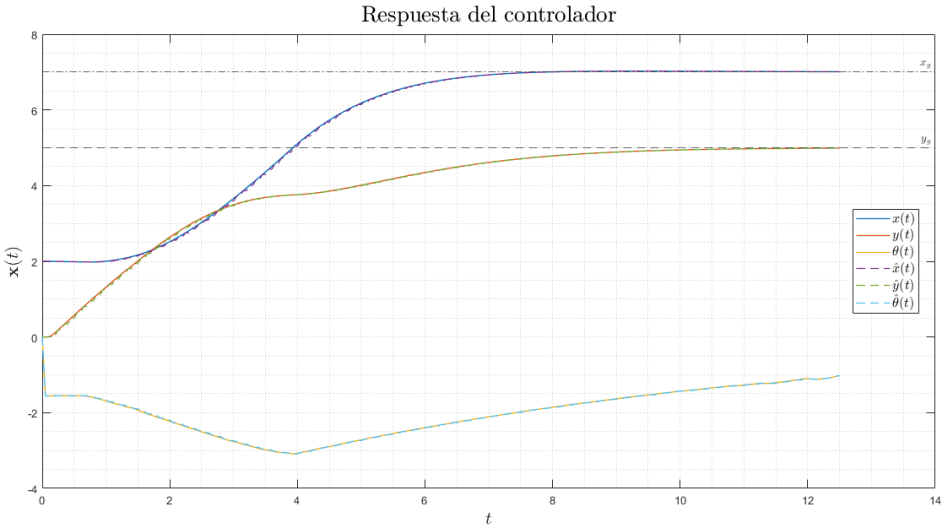


Figura 33: Respuesta controlador no lineal con meta fija.

En las Figuras 29 a 33 se muestran las trayectorias del vector de estados. Al observar solamente el vector de estados se puede obtener una buena idea sobre la respuesta del robot. Sin embargo, para comprender mejor el comportamiento sobre el plano xy se grafica también el recorrido del robot hasta alcanzar el objetivo $\boldsymbol{\xi}_g$.

Al analizar las trayectorias realizadas por los diferentes controladores, se puede observar que el controlador LQI:, Figura 36, se aproximó en una trayectoria casi directa al objetivo. Las únicas desviaciones que presenta se deben al overshoot: asociado al controlador. El controlador LQR: presentó una trayectoria directa a la meta, sin embargo al inicio se puede observar una etapa de vibraciones en la trayectoria. Esto se debe a que en esa etapa el controlador devuelve una velocidad en las ruedas ϕ_L y ϕ_R mayor a la velocidad angular máxima de los motores, lo que se observa en la Figura 37.

Por otra parte se observa en la Figura 34 la trayectoria realizada por el robot diferencial cuando se utiliza un controlador PID:. En la siguiente Figura, 35, se observa la trayectoria retornada por el controlador PID: con acercamiento exponencial. Ambas trayectorias se acercan directamente al objetivo, sin embargo la trayectoria obtenida al utilizar el controlador PID: con acercamiento exponencial nunca llegó a la meta. Esto responde a la naturaleza de un acercamiento exponencial, el cual es asintótico a su objetivo.

Por último la trayectoria realizada por el robot al utilizar el controlador no lineal es la menos directa hacia el objetivo. Esta se puede analizar en a Figura 38. Este controlador, sin embargo fue el más veloz de todos con un tiempo aproximado de convergencia a la referencia de $12.5s$. Estos tiempos aproximados pueden encontrarse en el Cuadro 4.

Según los resultados obtenidos en el experimento utilizando un objetivo fijo, se determinan dos controladores óptimos para alcanzar la meta dependiendo de las necesidades de la aplicación. En primer lugar, si se necesita que el controlador se estacione en el objetivo y sea veloz; el controlador óptimo para este caso es el controlador no lineal. En segundo lugar, si se necesita que el robot se acerque a su objetivo de manera directa, se recomienda utilizar un controlador LQI:. Adicionalmente a esto, ninguno de los controladores presenta una carga computacional elevada para el microcontrolador. Como se puede observar en el algoritmo 9, el microcontrolador debe de realizar dos tareas: la aproximación y el control de pose. De estas tareas, la aproximación de pose es la que requiere más poder computacional.

Controlador	Tiempo de convergencia
PID:	22s
PID: acercamiento exp.	25s
LQI:	200s
LQR:	100s
No lineal	21.5s

Cuadro 4: Tiempo de convergencia hasta la meta

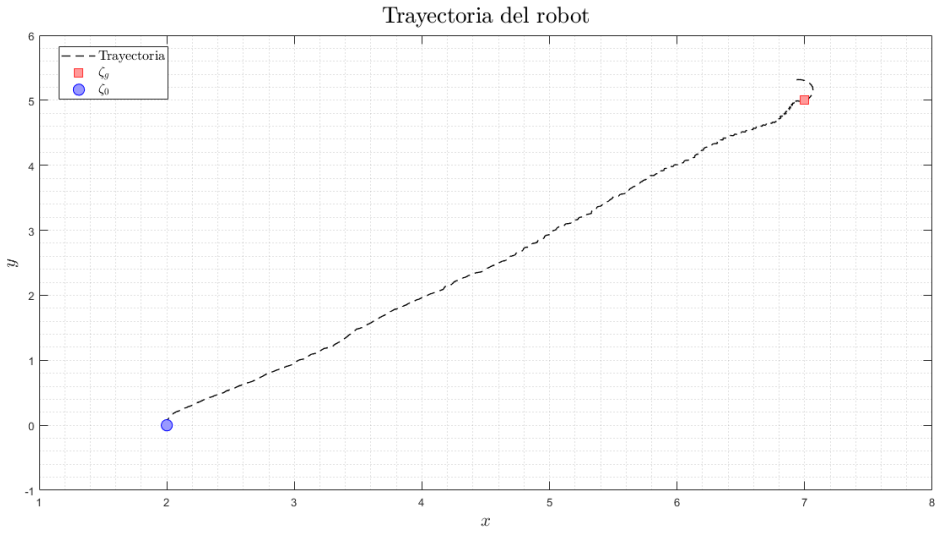


Figura 34: Trayectoria del robot utilizando un PID: con meta fija.

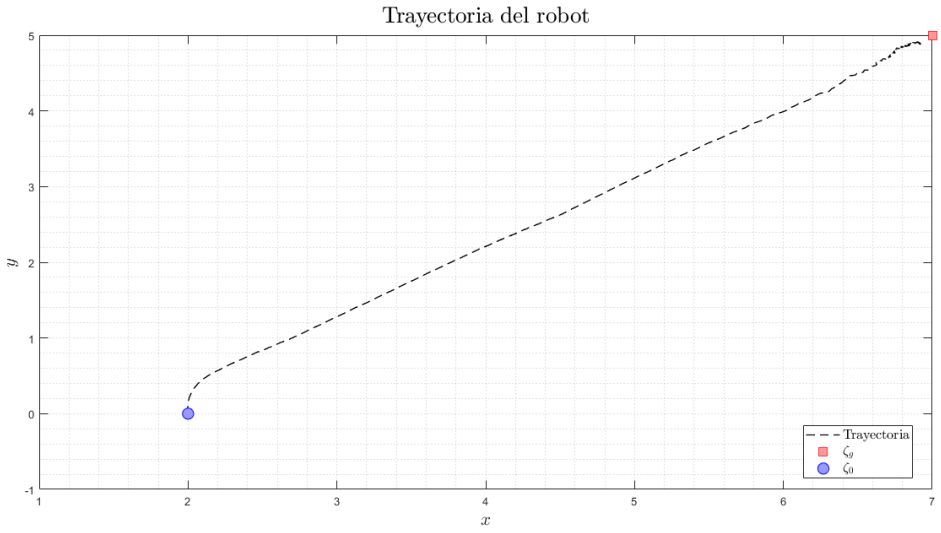


Figura 35: Trayectoria del robot utilizando un PID: con acercamiento exponencial con meta fija.

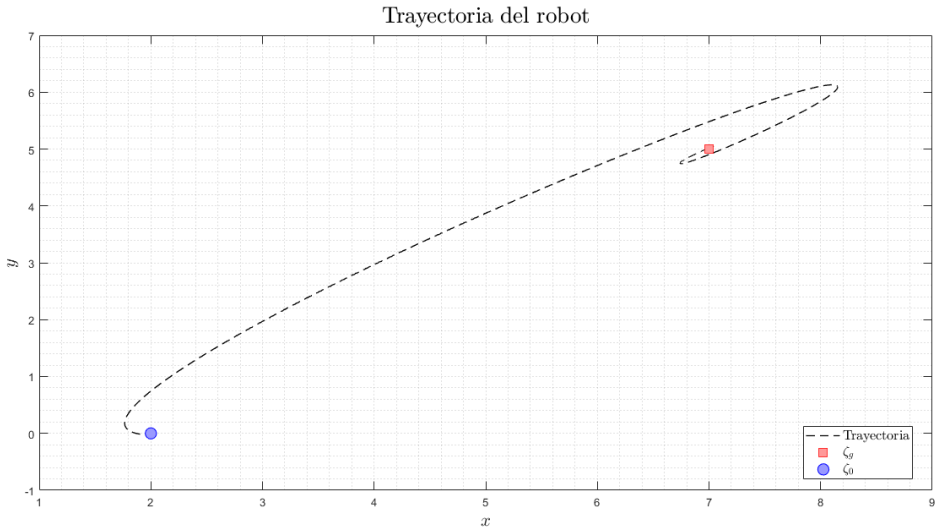


Figura 36: Trayectoria del robot utilizando un LQI: con meta fija.

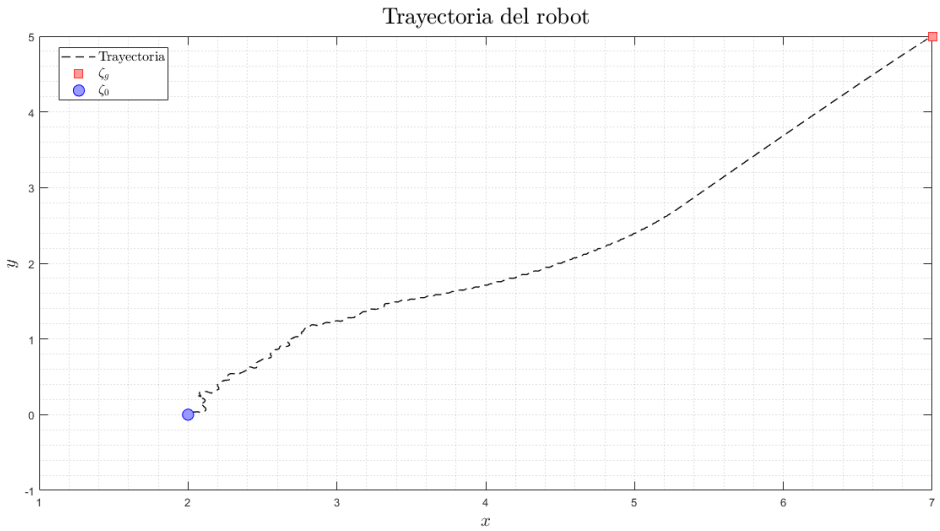


Figura 37: Trayectoria del robot utilizando un LQR: con meta fija.

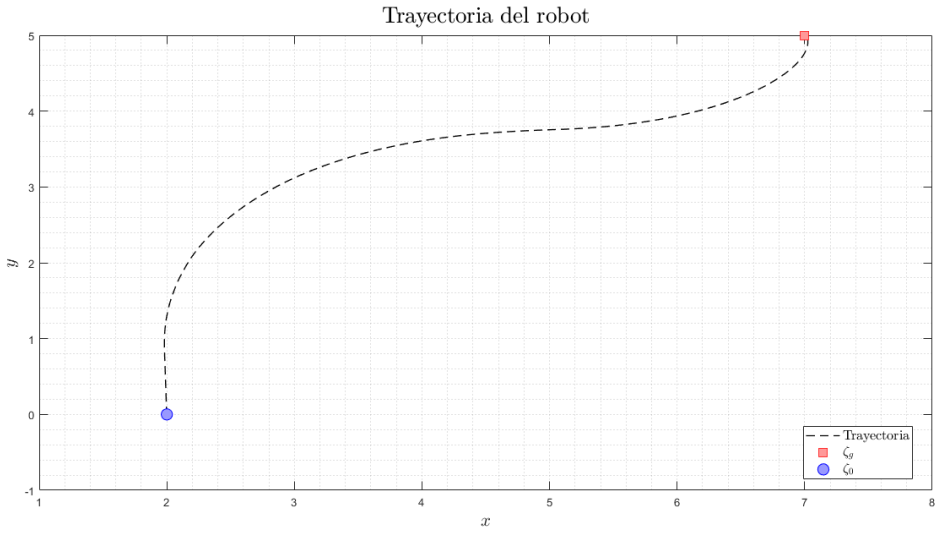


Figura 38: Trayectoria del robot utilizando un controlador no lineal con meta fija.

10.2.2. Seguimiento de trayectoria

El segundo experimento realizado respecto al control de los Bitbots fue el seguimiento de una trayectoria. La trayectoria se definió por medio de dos vectores, x_{traj} y y_{traj}. El algoritmo general para el control de los robot cambia ligeramente en este caso, pues ahora se requiere que cambie de objetivo al haber alcanzado la meta anterior. El pseudocódigo actualizado se puede encontrar en el algoritmo 10. Al igual que en la prueba pasada, se integran todos los resultados anteriores en este experimento. La estimación de pose y las varianzas obtenidas por la prueba de visión por computadora.

En las Figuras 39 a 43 se puede observar el seguimiento de los controladores de la trayectoria establecida por los Cuadros rojos. Como se observa en estas Figuras, todos los controladores lograron realizar un correcto seguimiento de la trayectoria establecida. Estas se obtienen luego de realizar la simulación en Webots con el robot Pioneer 3dx. A continuación se realiza un análisis de cada uno de los controladores.

En primer lugar se puede encontrar el seguimiento realizado por el controlador PID: en la Figura 39. Este controlador presentó irregularidades notorias alcanzando los primeros puntos de la trayectoria. En los siguientes puntos también se pueden encontrar irregularidades menores en el trayecto del robot. Este mismo fenómeno se puede observar en la Figura 40. Estos comportamientos concuerdan con lo observado en la prueba con objetivo fijo, pues ambos controladores presentaron estas vibraciones durante su recorrido hacia la meta.

Seguido de los controladores PID: se puede encontrar en las Figuras 42 y 41 el seguimiento por medio de controladores modernos LTI:. Estos realizaron un excelente seguimiento, pues alcanzaron de manera satisfactoria todos los puntos de la trayectoria. El único inconveniente respecto al controlador LQI: es la curva extra realizada al inicio. Como se puede observar en la prueba anterior, el controlador LQI: presenta overshoot: durante su recorrido hacia la meta. Esto concuerda con el comportamiento observado en la Figura 41. El seguimiento posterior a esta curva al inicio es satisfactorio. Sin embargo, el controlador LQR: presentó en este caso un mejor seguimiento de la trayectoria establecida. Por último se encuentra el seguimiento de trayectoria realizado mediante un controlador no lineal en la Figura 43. Este controlador realizó un seguimiento satisfactorio, pues alcanzó cada punto. Las rutas utilizadas, sin embargo, presentan pequeñas curvas hasta llegar al objetivo para luego realizar otra curva similar.

Según los resultados obtenidos en este experimento, se determinó que los controladores modernos LTI: y el controlador no lineal son ideales para utilizar en el caso de rastreo de trayectorias. Los controladores modernos LTI: (LQR: y LQI:) presentaron la trayectoria más limpia de todos. Al tomar en cuenta la desviación inicial del controlador LQI:, se determina que el controlador LQR: es el óptimo para una aplicación de seguimiento de trayectorias. Cabe resaltar que el controlador no lineal tuvo un comportamiento satisfactorio en ambas pruebas realizadas.

Algorithm 10: Algoritmo general de estimación y controlador de pose para seguimiento de trayectorias

input : Trayectoria, lectura de cámara y encoders
output: Velocidades en cada motor
`/* Incialización de variables */`
Inicialización constantes del controlador
Inicialización variables del filtro de Kalman
Definición de trayectorias $\rightarrow \mathbf{traj} = \begin{bmatrix} \mathbf{x}_{traj} \\ \mathbf{y}_{traj} \end{bmatrix}$
while *Verdadero* **do**
  `/* Seguimiento de trayectoria */`
  $\mathbf{e}_{traj} \leftarrow \begin{bmatrix} x_g - \hat{x} \\ y_g - \hat{y} \end{bmatrix}$
  if $\|\mathbf{e}_{traj}\| \leq 0.1$ **then**
    $x_g \leftarrow x_{traj}[k+1]$
    $y_g \leftarrow y_{traj}[k+1]$
  end
  `/* Aproximación de pose` $\rightarrow \hat{\boldsymbol{\xi}}$ `*/`
  Predicción de pose $\rightarrow \hat{\boldsymbol{\xi}}_{k|k-1}$
  if *Lectura nueva de la cámara* **then**
    Corrección de estimación de pose $\hat{\boldsymbol{\xi}}_{k|k}$
  else
    $\hat{\boldsymbol{\xi}}_{k|k} \leftarrow \hat{\boldsymbol{\xi}}_{k|k-1}$
  end
  $\hat{\boldsymbol{\xi}} \leftarrow \hat{\boldsymbol{\xi}}_{k|k}$
  `/* Cálculo de error` $\rightarrow \mathbf{e}$ `*/`
  $\mathbf{e} \leftarrow \begin{bmatrix} x_g - \hat{x} \\ y_g - \hat{y} \end{bmatrix}$
  `/* Control de posición` $\rightarrow \begin{bmatrix} v \\ \omega \end{bmatrix}$ `*/`
  Control de posición $\rightarrow v$
  Control de orientación $\rightarrow \omega$
  $\mathbf{u} \leftarrow \begin{bmatrix} v \\ \omega \end{bmatrix}$
  `/* Velocidad en las llantas` $\begin{bmatrix} \phi_R \\ \phi_L \end{bmatrix}$ `*/`
  $\phi_R \leftarrow \frac{v+\omega L}{R}$
  $\phi_L \leftarrow \frac{v-\omega L}{R}$
end

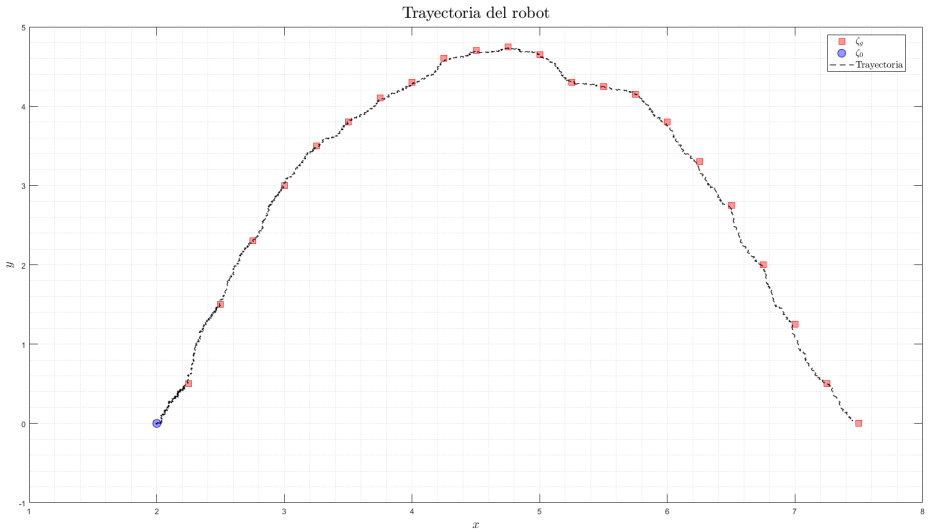


Figura 39: Trayectoria del robot con PID: con seguimiento de trayectorias.

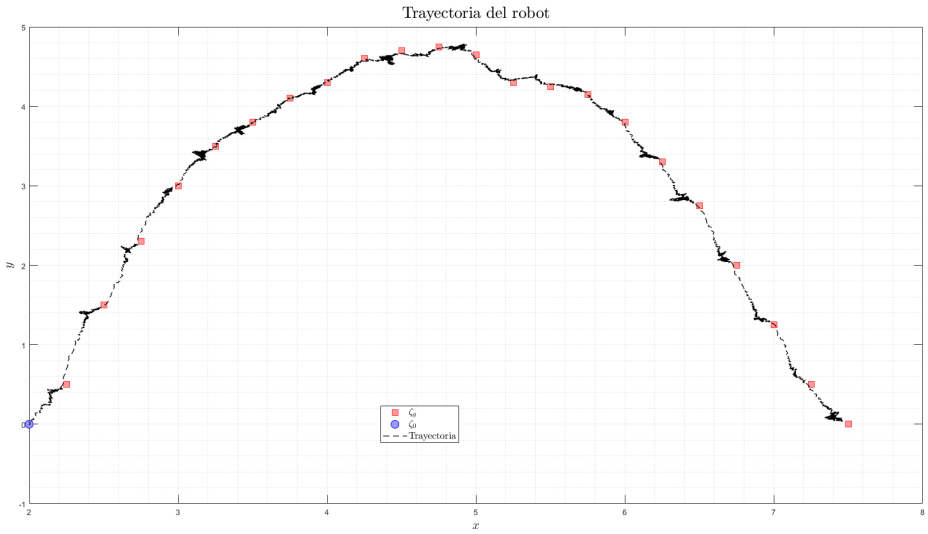


Figura 40: Trayectoria del robot con PID: y acercamiento exp. con seguimiento de trayectorias.

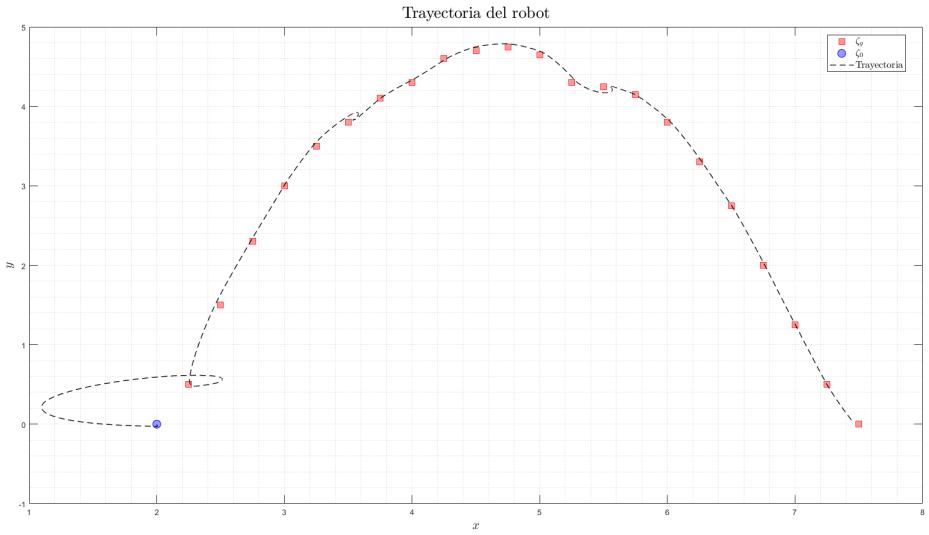


Figura 41: Trayectoria del robotcon LQI: con seguimiento de trayectorias.

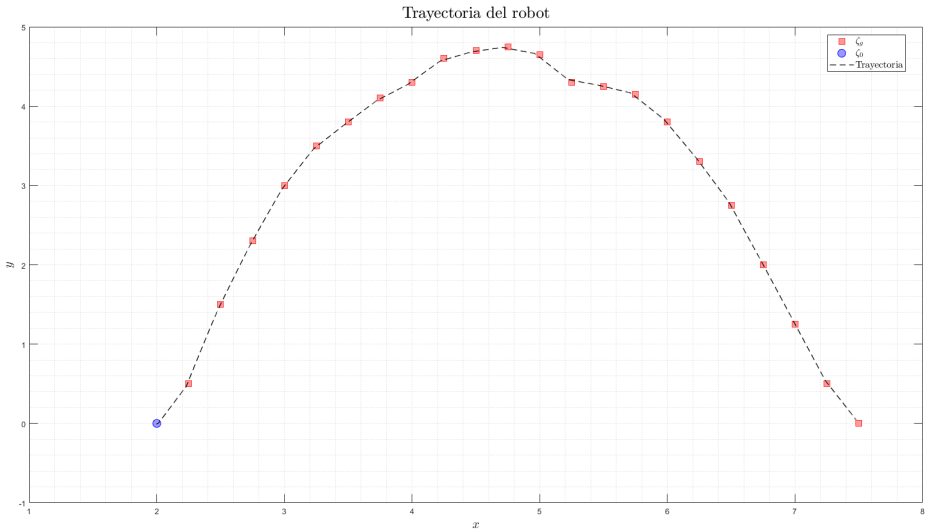


Figura 42: Trayectoria del robot con LQR: con seguimiento de trayectorias.

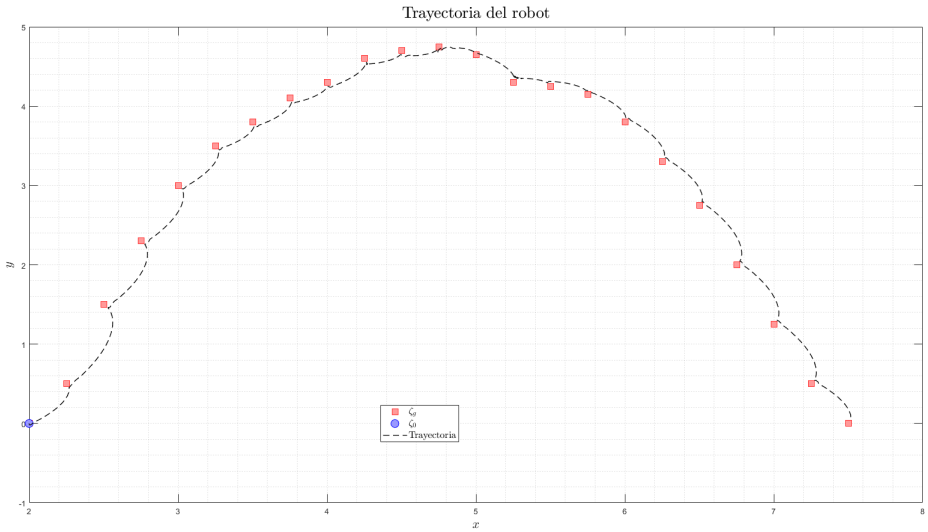


Figura 43: Trayectoria del robot con controlador no lineal con seguimiento de trayectorias.

CAPÍTULO 11

Conclusiones

1. El algoritmo efectivo para el sensado de pose y el control de posición de los robots utiliza el controlador no lineal. Este obtuvo resultados consistentes en ambas pruebas realizadas, con meta fija y rastreo de trayectorias. El controlador no lineal se escoge como controlador óptimo para el control de pose de los Bitbots.

2. El algoritmo de fusión de sensores implementado presentó un correcto seguimiento de la pose del agente. Las desviaciones presentadas por la intermitencia de la lectura de la cámara son despreciables, pues la predicción es suficientemente buena.

3. El rendimiento del algoritmo de visión por computadora es aceptable, pues es capaz de realizar lecturas de pose de hasta siete agentes con una frecuencia de muestreo de 11Hz. Adicionalmente el error medio presentado no supera el 10 % de la dimensión mínima del robot. El error aumenta hasta 3.3cm cuando el agente se aleja del lente.

4. La simulación en Webots presentó un correcto funcionamiento de cada uno de los sistemas integrados.

CAPÍTULO 12

Recomendaciones

1. Se recomienda implementar el algoritmo de Kalman en un microcontrolador que soporte una cálculos matriciales, pues las operaciones necesarias para llevar a cabo el algoritmo y computacionalmente pesadas.

2. Se recomienda realizar las pruebas físicas correspondientes a cada experimento realizado en el proyecto y luego compararlas con los resultados obtenidos en Webots.

3. Se recomienda colocar la cámara exactamente sobre el centro de la mesa de pruebas, de esa forma el error máximo se encontrará en las esquinas de la mesa.

4. Dado que el algoritmo de visión por computadora es sensible a los cambios de luz y a las sombras, se recomienda utilizar una fuente de luz externa para asegurar una iluminación uniforme de la mesa. Esta fuente externa puede ser un foco instalado de tal forma que el soporte de la cámara no produzca sombras.

5. Se recomienda disminuir a un mínimo el deslizamiento entre las llantas del robot y la mesa, pues el algoritmo de predicción de pose es sensible a estos deslizamientos.

6. A la hora de la implementación física, se recomienda ajustar los controladores para evitar saturaciones en los motores.

CAPÍTULO 13

Bibliografía

[1] D. Pickem, L. Wang, P. Glotfelter, Y. Diaz-Mercado, M. Mote, A. D. Ames, E. Feron y M. Egerstedt, "Safe, Remote-Access Swarm Robotics Research on the Robotarium", *CoRR*, vol. abs/1604.00640, 2016. arXiv: 1604.00640. dirección: http://arxiv.org/abs/1604.00640.

[2] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. D. Ames, E. Feron y M. Egerstedt, "The Robotarium: A remotely accessible swarm robotics research testbed", *CoRR*, vol. abs/1609.04730, 2016. arXiv: 1609.04730. dirección: http://arxiv.org/abs/1609.04730.

[3] A. J. Rodas, "Desarrollo e implementación de algoritmo de visión por computador en una mesa de pruebas para la experimentación con micro-robots móviles en robótica de enjambre", Tesis doct., Universidad del Valle de Guatemala, ene. de 2019.

[4] M. J. Castillo, "Diseñar e implementar una red de comunicación inalámbrica para la experimentación en robótica de enjambre", Tesis doct., Universidad del Valle de Guatemala, ene. de 2019.

[5] M. Le Goc, L. H. Kim, A. Parsaei, J.-D. Fekete, P. Dragicevic y S. Follmer, "Zooids: Building Blocks for Swarm User Interfaces", en *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST)*, Tokyo, Japan, oct. de 2016. DOI: 10.1145/2984511.2984547. dirección: https://hal.inria.fr/hal-01391281.

[6] E. G. Alegre, M. Pajares y E. H. A. d. la, *Conceptos y métodos en visión por computador*. s.l., 2016.

[7] *Basic concepts of the homography explained with code*. dirección: https://docs.opencv.org/master/d9/dab/tutorial_homography.html#lecture_16.

[8] *Canny Edge Detection*. dirección: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html.

[9] J. F. Canny, "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, págs. 679-698, 1986.

[10] K. Lynch y F. C. Park, *Modern robotics: mechanics, planning, and control*. Cambridge University Press, 2019.

[11] T.-C. Lee, K.-T. Song, C.-H. Lee y C.-C. Teng, "Tracking control of unicycle-modeled mobile robots using a saturation feedback controller", *IEEE Trans. Contr. Sys. Techn.*, vol. 9, págs. 305-318, 2001.

[12] R. Dhaouadi y A. A. Hatab, "Dynamic Modelling of Differential-Drive Mobile Robots using Lagrange and Newton-Euler Methodologies: A Unified Framework", en *ICRA 2013*, 2013.

[13] R. B. Fierro y F. L. Lewis, "Control of a nonholonomic mobile robot using neural networks", *IEEE transactions on neural networks*, vol. 9 4, págs. 589-600, 1998.

[14] S. Nurmaini, K. Dewi y B. Tutuko, "Differential-Drive Mobile Robot Control Design based-on Linear Feedback Control Law", 2017.

[15] R. Negenborn, *Robot Localization and Kalman Filters. On finding your position in a noisy world*, 2003.

[16] R. Murray, *Optimization-Based Control*. California Institute of Technology, 2010.

[17] I. Reid, *Estimation II*, 2001. dirección: `http://www.robots.ox.ac.uk/~ian/Teaching/Estimation/LectureNotes2.pdf`.

[18] T. Lacey, *Tutorial: The Kalman Filter*. dirección: `http://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%5C%20filter.pdf`.

[19] A. Kelly, "A 3D State Space Formulation of a Navigation Kalman Filter for Autonomous Vehicles", Camege Mellon University, inf. téc., mayo de 1994.

[20] S. Thrun, W. Burgard y D. Fox, *Probabilistic robotics*. MIT Press, 2005.

[21] A. Kaehler y G. Bradski, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, 2016, ISBN: 9781491938003. dirección: `https://books.google.com.gt/books?id=SKy3DQAAQBAJ`.

[22] C. Ltd., *Webots Reference Manual*. dirección: `https://cyberbotics.com/doc/guide/index`.

CAPÍTULO 14

Anexos

14.1. Simulaciones en Webots

Repositorio de las simulaciones: `https://github.com/jmolinacalzia/Simulacione-Bitbots`

Figura 44: Entorno de simulación

CAPÍTULO 15

Glosario

difeomorfismo: Función biyectiva y continua que mapea la vecindad de un punto p en un conjunto $\mathcal{M}$ a una vecindad $\mathcal{N}$.. 20, 55, 57

GPS: Sistema de posicionamiento global. **Global Positioning System** en inglés.. 41

LQI: Control lineal cuadrático-integral. **Linear Quadratic-Integral**, en inglés . 20, 29, 50, 57, 58, 62, 63, 65, 67, 69, 72

LQR: Control lineal cuadrático. **Linear Quadratic Regulator**, en inglés. 20, 29, 50, 55–57, 59, 62, 64, 65, 67, 69, 72

LTI: Lineal invariante en el tiempo. **Linear Time-Invariant**, en inglés. 19, 22, 29, 55, 57, 62, 69

odometría: Estudio de la estimación de la posición y orientación de vehículos con ruedas durante su navegación. 41, 44, 45, 47, 61

overshoot: El caso en que una señal exceda su objetivo. Término utilizado en teoría de control y procesamiento de señales.. 62, 65, 69

PID: Controlador proporcional, integral, derivativo. 18, 50–54, 62, 63, 65, 66, 69, 71

QR: Código de respuesta rápida que almacena información en una matriz de puntos bidimensional. **Quick Response** en inglés.. 4, 31

thresholding: Método de valor umbral. Los métodos de valor umbral son algoritmos utilizados para aislar objetos de interés dentro de un gráfico.. 33, 38