GENERACION Y PLANIFICACION DE TRAYECTORIAS PARA EL DESARROLLO DE TRABAJO COOPERATIVO CON ROBOTS MOVILES EN LA DETECCION DE MINAS ANTIPERSONALES.

ANDERSON JOSE SANCHEZ PADILLA

UNIVERSIDAD AUTONOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍA
INGENIERÍA MECATRÓNICA
BUCARAMANGA
2022

GENERACION Y PLANIFICACION DE TRAYECTORIAS PARA EL DESARROLLO DE TRABAJO COOPERATIVO CON ROBOTS MOVILES EN LA DETECCION DE MINAS ANTIPERSONALES.

ANDERSON JOSE SANCHEZ PADILLA

TRABAJO DE GRADO PARA OPTAR POR EL TITULO DE:
INGENIERO MECATRONICO

MS.C. HERNANDO GONZALEZ ACEVEDO
DIRECTOR

UNIVERSIDAD AUTONOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍA
INGENIERÍA MECATRÓNICA
BUCARAMANGA
2022

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

1. INTRODUCCIÓN

En este libro, se presentan y describen los resultados obtenidos del proyecto de generación de trayectorias para plataformas robóticas móviles de tracción diferencial. Este proyecto tiene su enfoque en los algoritmos de generación de trayectorias para robots móviles en busca de una navegación autónoma, por consiguiente se presentan análisis de forma detallada de temas tales como la cinemática de los robots móviles con tracción de tipo diferencial, cálculos del control de alto nivel (planificadores de ruta), cálculos de control de bajo nivel (control de los motores), además de mostrar los algoritmos programados para la generación y planificación de trayectorias para los robots.

Para el desarrollo de este, se partió los robots móviles desarrollados por el estudiante Daniel Felipe León Cardona en el proyecto “Diseño y construcción de una plataforma robótica para el control de formación y distribución de tareas”, dirigido por el profesor Carlos Forero González, a estos robots se le realizaron modificaciones para lograr el cumplimiento del presente proyecto.

De igual manera se muestran las oportunidades de mejora que pueden ser aplicables a este proyecto para un mejor cumplimiento de las actividades para las cuales fue aplicado.

2. OBJETIVOS

2.1. Objetivo General.

Desarrollar algoritmos para la generación y planificación de trayectorias para el desarrollo de trabajo cooperativo con robots móviles en la detección de minas antipersonales.

2.2. Objetivos Específicos.

- Establecer un sistema de comunicación bidireccional inalámbrico entre varios robots para el desarrollo de trabajos cooperativos.
- Desarrollar un algoritmo basado en campos potenciales para la exploración de terrenos.
- Desarrollar un algoritmo basado en grafos de visibilidad para la creación de mapas.
- Validar los algoritmos de generación de trayectorias en un ambiente controlado que simule un campo minado.

3. CAPITULO 1

En este capítulo se presentan las ecuaciones matemáticas que rigen a los robots móviles con tracción de tipo diferencial al igual que el algoritmo de control de velocidad para los actuadores (motores).

Los robots móviles son aquellos que, gracias a su sistema de locomoción, pueden desplazarse libremente por un área de trabajo sin definir para lograr su objetivo, sin la intervención de un operador humano. Los robots móviles se caracterizan por una conexión inteligente entre las operaciones de percepción y acción que define su comportamiento y le permite llegar a la consecución de los objetivos programados sobre entornos con cierta incertidumbre. [9]

Para el desarrollo de este proyecto se implementaron los algoritmos de generación de caminos y seguimientos de ruta y posición en un robot móvil de locomoción tipo diferencial comúnmente llamados en la literatura como robot móvil tipo uniciclo.

Los robots móviles se pueden clasificar según su locomoción, nivel de autonomía entorno de trabajo, configuración cinemática geometría entre otras, como: Terrestres Acuáticos, Aéreos, Espaciales.

3.1. Caracterización del robot móvil

El robot utilizado en el desarrollo de este proyecto es de tipo diferencial con dos motores que gobiernan las ruedas (derecha e izquierda) de forma independiente, controlado por un microcontrolador ATmega2560 montado en la tarjeta de desarrollo Arduino Mega 2560.

La operación del robot móvil se realiza en un entorno controlado y en una superficie plana buscando eliminar el deslizamiento en las ruedas.

El robot móvil cuenta con ruedas de 32 [mm] de diámetro y una separación de 82 [mm] entre las ruedas, tiene dos motores de 6 [V] y 1600 [mA], se realiza la medición de velocidad con encoders ópticos incrementales acoplados en el eje del motor, estos motores son controlados mediante un driver L298N capaz de suministrar hasta 2 [A] de corriente a cada motor, se comunica de manera inalámbrica a través de bluetooth a 2.5 GHz, para la interacción con el medio está equipado con sensores de ultra sonido que miden distancias, todo esto alimentado con una batería de tipo Lipo de 11.1 [V] y 2.2 [Ah] monitoreada mediante un sensor que indica la tensión y el estado de la batería.

A continuación, se presentan los componentes descritos:

Elemento	Cantidad	Voltaje [V]	Consumo de corriente [mA]
Micromotor	2	6	1600
Encoder	2	2.7 – 18	20
Sensor ultrasónico HC-SR04	4	5	15
Arduino Mega 2560	1	7-24	200
Modulo L298N	1	7-36	36

Tabla 1. Caracterización de la instrumentación.

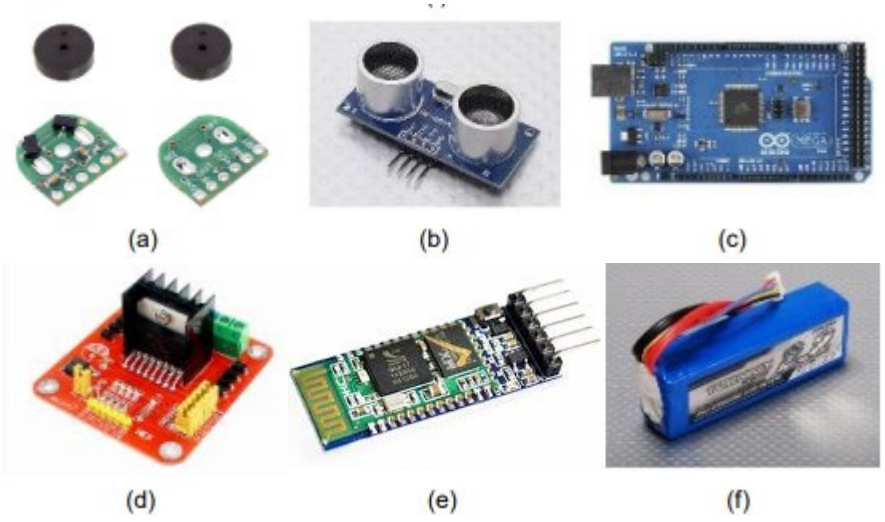


Figura 1. Componentes del robot móvil [12]

En la figura 1 se muestran los siguientes componentes: a) Encoder magnético Pololu. (b) Sensor ultrasónico HC-SR04. (c) Arduino Mega 2560 R3. (d) Módulo L298N. (e) Módulo de HC-05. (f) Batería LiPo 2200 mAh 11.1.

3.2. Medición de velocidad en las ruedas.

Para la medición de velocidad angular de cada rueda, se hace uso del encoder acoplado a cada motor, el encoder magnético incremental entrega dos señales cuadradas desfasadas 90° entre sí, comúnmente llamadas señal A y señal B [10], analizando los estados de activación de cada señal y las combinaciones entre estas, se puede obtener una tabla lógica que relaciona el valor leído en el encoder con el aumento o disminución de velocidad y el sentido de giro de la rueda con alta precisión.

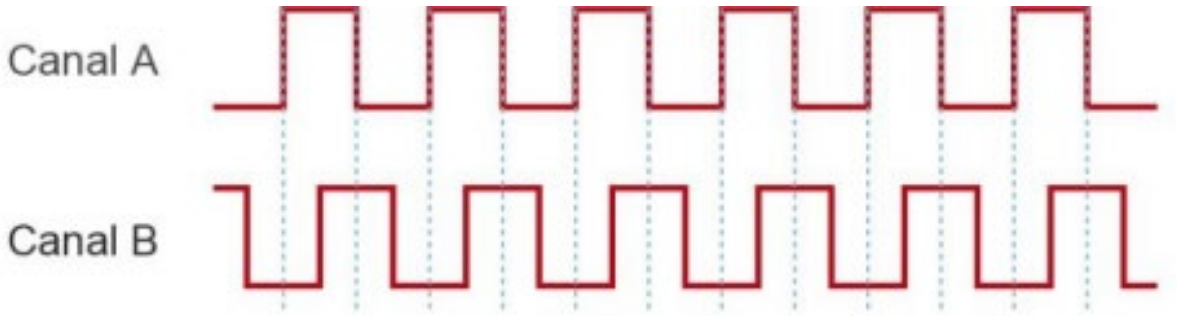


Figura 2. Señales de salida del encoder. [10]

Estado	AB
2	10
3	11
1	01
0	00

Tabla 2. Posibles estados actuales de las señales A y B.

	Anterior	0	1	2	3
Actual	AB	00	01	10	11
0	00	0	-1	+1	E
1	01	+1	0	E	-1
2	10	-1	E	0	+1
3	11	E	+1	-1	0

Tabla 3. Análisis de los estados anterior y siguiente de las señales A y B.

En la tabla 2 se analiza el estado actual de las señales A y B (en el momento de hacer la medición) la secuencia medida es ascendente (como lo indica la flecha a la izquierda de la tabla) esto indica un aumento en las cuentas de los pulsos recibidos, si la secuencia es descendente (como lo indica la flecha a la derecha de la tabla) indica una disminución en la cuenta de los pulsos recibidos.

En la tabla 3 se analiza el estado actual y el estado anterior para alcanzar una mayor precisión en la asignación de cuentas de los pulsos y determinar el sentido de giro, si se toma como el bit más significativo a la señal A se establece como sentido positivo el giro antihorario y como sentido negativo el giro horario, por ejemplo:

Si se toma como medida actual el estado 1 (A=0, B=1) y como medida anterior el estado 2 (A=1, B=0) se define el sentido de giro positivo (antihorario) y aumento de 1 en las cuentas de los pulsos, y si luego se toma la medida A = 0, B=0 estos pasan a ser el estado actual (estado 0) comparando al estado anterior estado 1) esto indica una disminución en 1 de la cuenta de los pulsos recibidos y un sentido de giro negativo (horario).

Mediante una rutina de interrupción externa se lee en cada periodo de muestreo la cantidad de pulsos recibidos, aplicando la ecuación (1) se puede determinar la velocidad a la que gira la rueda.

$$W = \frac{60*n}{t*R} \quad [RPM] \tag{1}$$

Donde W → es la velocidad angular en revoluciones por minuto.

n → es el número de pulsos que llegaron con determinada frecuencia

t → es el tiempo de generación de los pulsos en segundos

R → es la resolución del encoder (1206 para los encoders utilizados en el robot).

3.3. Sistema de control de velocidad de las ruedas.

Los encoders se encuentran acoplados directamente en el eje de cada motorreductor, los motorreductores se controlan desde la placa Arduino mediante el driver L298N, una vez determinada la medición de la velocidad de las ruedas se establece el controlador que garantice la velocidad en cada motor teniendo como entrada de control el valor del PWM entregado desde el microcontrolador a través del driver de potencia, como se especifica en el siguiente diagrama.

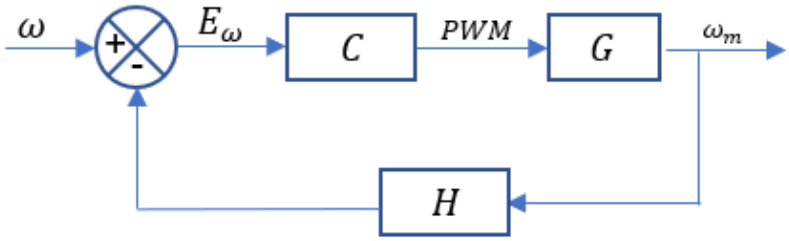


Figura 3. Lazo cerrado de control de velocidad del motor.

Donde:

ω representa la consigna de velocidad.

E_ω representa el error de velocidad

C es el microcontrolador

G es el conjunto driver y motor

ω_m es la velocidad desarrollada en el motor

H es el sensor de velocidad (encoder)

Para sintonizar el controlador es necesario conocer la función de transferencia G que describe la dinámica del motor la cual relaciona la velocidad del motor con el PWM ingresado, para esto se realiza identificación del motor para un sistema de orden reducido (de primer o segundo orden) más el tiempo muerto. La identificación

de los parámetros se hace mediante la aplicación de una entrada escalón, analizando la respuesta del sistema [11].

Se buscará aproximar el modelo hasta una función de primer orden de la forma:

$$G(s) = \frac{k_p e^{-t_m s}}{\tau s + 1} \tag{2}$$

La cual tiene una respuesta transitoria característica como la mostrada en la figura 4:

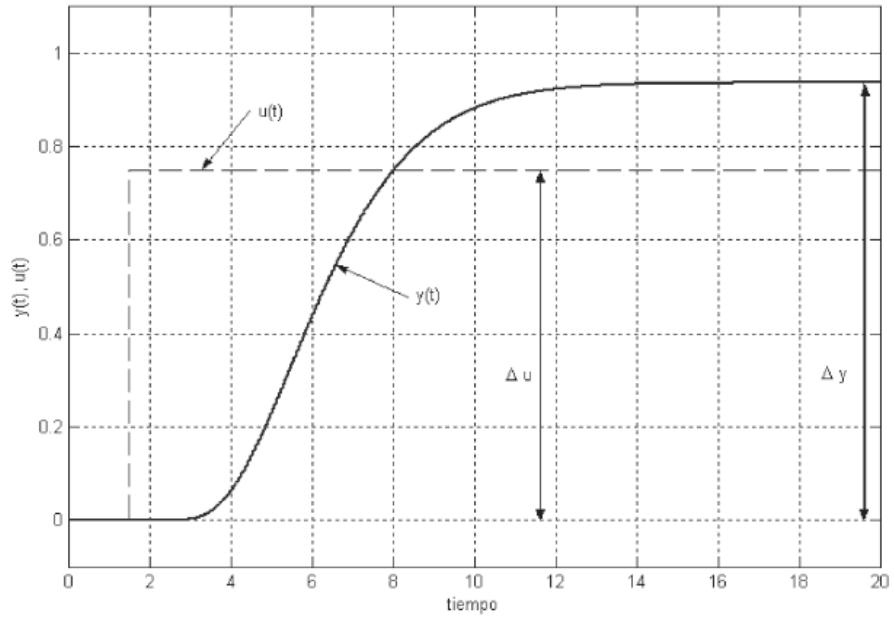


Figura 4. Respuesta característica de un sistema de 1er orden [11]

En la figura 4 se observa el comportamiento de un sistema y(t) de primer orden ante la aplicación de una entrada escalón u(t), la ganancia Kp se puede determinar cómo [11]:

$$k_p = \frac{\Delta y}{\Delta u} \tag{3}$$

La respuesta del modelo en el dominio del tiempo se representa como [11]:

$$y(t) = \begin{cases} 0 & 0 \leq t \leq t_m \\ k_p \left[1 - e^{-\frac{(t-t_m)}{\tau}}\right] \Delta u & t_m \leq t \end{cases} \tag{4}$$

A continuación, se aplica una entrada escalón al motor normalizada en porcentajes (pwm y velocidad), esto en búsqueda de la estimación mediante modelo de caja negra.

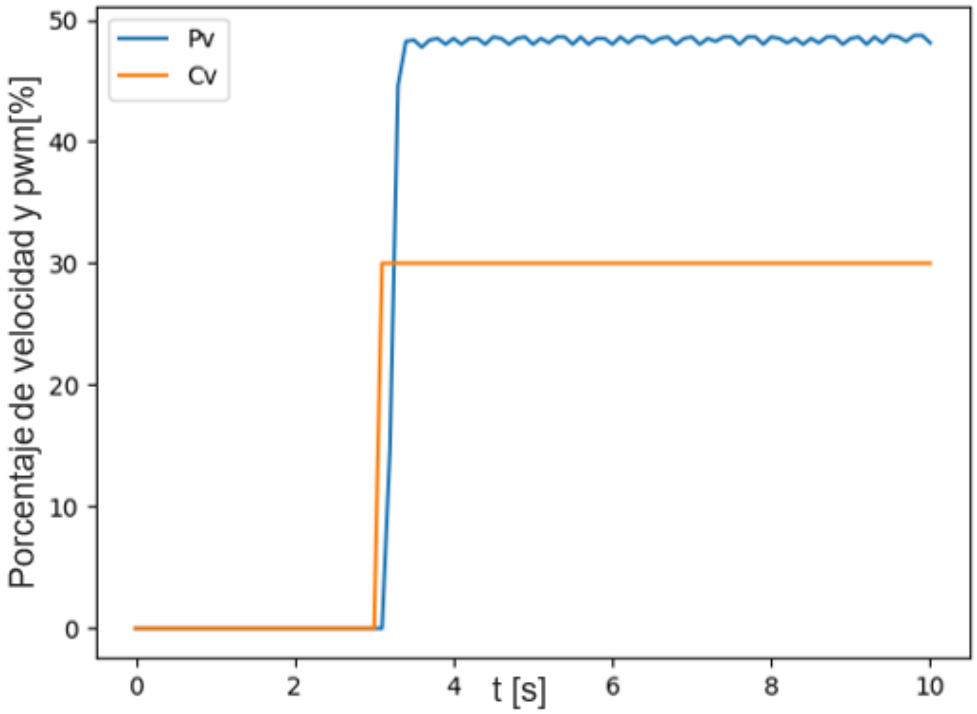


Figura 5. Respuesta escalón en lazo abierto.

Para la respuesta en lazo abierto se realiza la respectiva normalización entre el valor máximo y mínimo de velocidad que el motor desarrolla (43 [rad/s] y 0 [rad/s]) y el valor máximo y mínimo de PWM a aplicar (0 y 255) teniendo en cuenta la zona muerta del robot.

Siendo así 43 [rad/s] el 100% de velocidad que el robot puede alcanzar y 255 el 100% de PWM que se le aplicará (puesto que el microcontrolador posee una resolución máxima de 8 bits en los pines PWM).

En la figura 5 se observa la respuesta escalón del motor, se aplicó una entrada correspondiente al 30 % de PWM y el motor respondió con cerca del 50% de la velocidad que puede desarrollar en la escala normalizada previamente.

Para calcular el valor de las constantes y realizar la estimación se programó la ecuación (4) en un algoritmo Python y se usó una placa Arduino para enviar pwm y medir la velocidad, recopilando y graficando de manera comparativa estos datos (ver figura 6) obteniendo como salida las constantes del modelo aproximado.

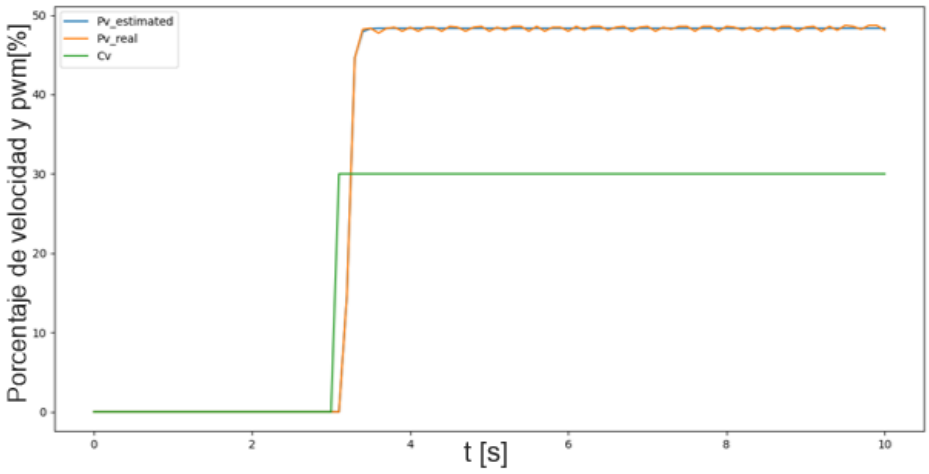


Figura 6. Estimación del sistema.

En la anterior figura se observa una muy buena respuesta por parte del modelo estimado (grafica color azul) por lo que se tomará como la función de transferencia del motor con las siguientes constantes: $k_p = 1.6116,\ \tau = 0.0453,\ t_m = 0.0837$, obteniendo el modelo del motor:

$$G(s) = \frac{1.6116e^{-0.0837s}}{1+0.0453s} \tag{5}$$

3.3.1. Sintonización del controlador.

Teniendo la función de transferencia del motor y la medición de velocidad se puede diseñar el controlador, aplicando métodos de sintonía como pueden ser Ziegler-Nichols, método Lambda entre otros.

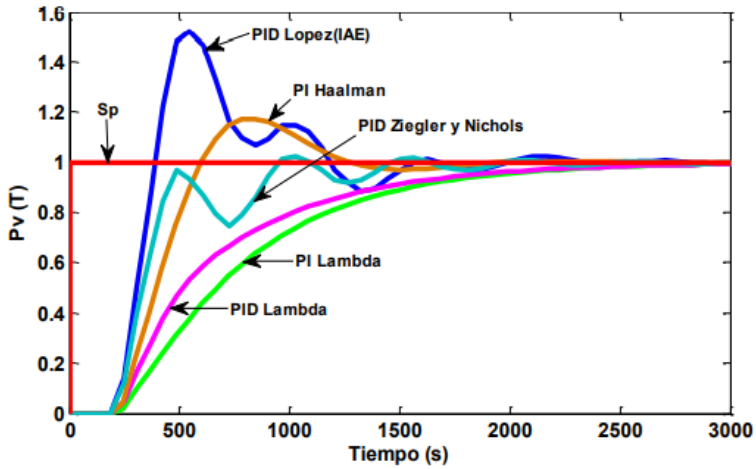


Figura 7. Comparación entre diferentes métodos de sintonización de controladores [11]

En la figura 7 se pueden observar las respuestas a entrada escalón de controladores sintonizados con diferentes métodos, con principales variaciones en características como sobrepaso y amortiguamiento, como se puede observar en dicha grafica el método de sintonía Lambda no presenta sobrepasos en ninguna de sus dos configuraciones, PI y PID, a diferencia de métodos como el de Ziegler-Nichols o Haalman que presentan sobrepasos y oscilaciones.

A continuación, se utiliza el método Lambda para sintonizar el controlador PID puesto que este presenta una respuesta más rápida que el PI y sin sobrepasos, el método Lambda es un caso especial de asignación de polos que es comúnmente usado en procesos en la industria, representado por un modelo de 1er orden [12].

3.3.2. Controlador PID y sintonía Lambda.

Para el diseño del control PID se utiliza la forma [12]:

$$C(s) = \frac{K(1+sT_i)(1+sT_d)}{sT_i} \tag{6}$$

Se aproxima el retardo del motor usando la aproximación de Pade de primer orden el cual da la función de transferencia del proceso [12].

$$G(s) = \frac{k_p e^{-t_m s}}{1+\tau s} \approx \frac{k_p k_c (1-\frac{st_m}{2})}{(1+\tau s)(1+\frac{st_m}{2})} \tag{7}$$

Para cancelar los polos del proceso se selecciona el tiempo integral como $T_i = \tau$ y el tiempo derivativo en $T_d = \frac{t_m}{2}$, adicionalmente se selecciona la ubicación del polo en lazo cerrado en $s = -\frac{1}{\lambda}$ [12] y la ecuación característica de lazo cerrado se convierte:

$$k_p k_c - \frac{\tau}{\frac{t_m}{2}+\lambda} = 0 \tag{8}$$

De esta forma se obtienen las reglas de sintonización del método de lambda:

$$k_c = \frac{\tau}{\left(\frac{t_m}{2}+\lambda\right)k_p} \tag{9}$$

$$T_i = \tau \tag{10}$$

$$T_d = \frac{t_m}{2} \tag{11}$$

Se selecciona el parámetro lambda como $\lambda = 3\tau$ para lograr controladores más robustos [11], obteniendo las siguientes constantes de acuerdo a las ecuaciones (9), (10) y (11):

$$k_c = 0.1581362$$

$$T_i = 0.0453$$

$$T_d = 0.04185$$

Para poder programar en un sistema digital el controlador se tomará la forma estándar debido a la simplicidad que implementar esta ecuación conlleva (12):

$$C(s) = k_c \left(1 + \frac{1}{T_i s} + T_d s\right) e(s) \tag{12}$$

Por lo cual se deben calcular las constantes PID relacionadas a esta configuración, esto se calcula mediante las siguientes ecuaciones [12]:

$$k_c = {k_c}' \frac{T_i' + T_d'}{T_i'} \tag{13}$$

$$T_d = \frac{T_i' T_d'}{T_i' + T_d'} \tag{14}$$

$$T_i = T_i' + {T_d}' \tag{15}$$

Aplicando las ecuaciones descritas hasta aquí, se encuentran las constantes PID del controlador resultantes de las ecuaciones 13, 14 y 15: $k_C = 0.31,\ T_d = 0.09, T_i = 0.02$, obteniendo el controlador que relaciona la velocidad en [rad/s] y el PWM interpretado en palabra digital:

$$C(s) = 0.31 \left(1 + \frac{1}{0.02s} + 0.09s\right) e(s) \tag{16}$$

Se procede a implementar dicho controlador en la placa Arduino con un periodo de muestreo de $t_s = 0.1\ [s]$ como se muestra en la figura (8) y se analiza el seguimiento al SP, para esto primero se calcula el error mediante la resta del Set-point y el valor de velocidad, se crea un vector de 3 posiciones llamado “memoria” que almacene el error, el termino derivativo y el termino integral, finalmente se programa la salida de control mediante la ecuación 16 con estos términos calculados, $k_p,\ T_d, T_i$ corresponden a las constantes PID previamente calculadas.

```
double error = setpoint - velocidad;

memoria[2] += Ts*error; // Termino integral
memoria[1] = (error-memoria[0])/Ts; // Termino derivativo
memoria[0] = error; // error[k-1]

double output = _kp*(memoria[0]+(1/_ti)*memoria[2]+_td*memoria[1]);
```

Figura 8. Implementación del controlador PID en la placa Arduino.

La respuesta en lazo cerrado se presenta en la figura 9, y en la figura 10 se presenta la salida de control de la ecuación (16).

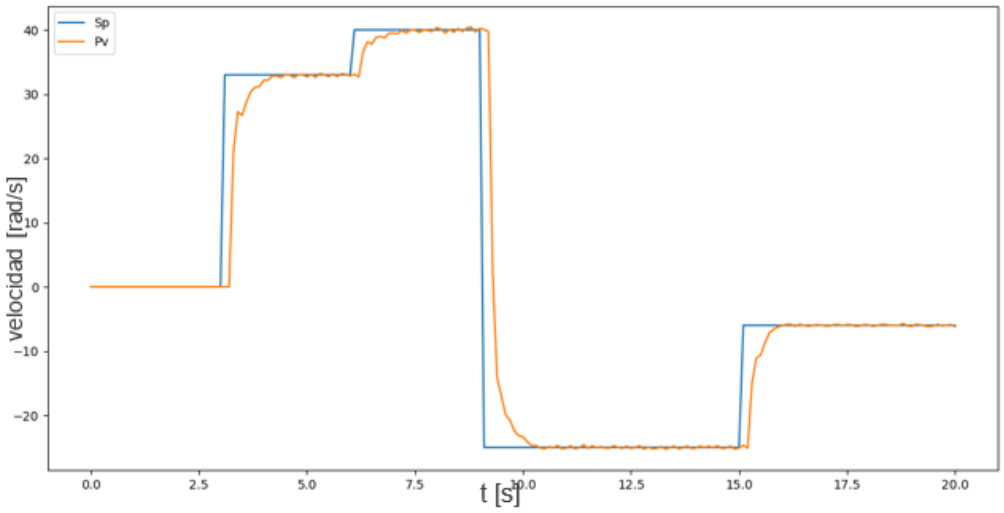


Figura 9. Respuesta en lazo cerrado.

El controlador implementado garantiza un error cero además de no tener sobrepaso, en la figura 10 se observa la acción de control generada.

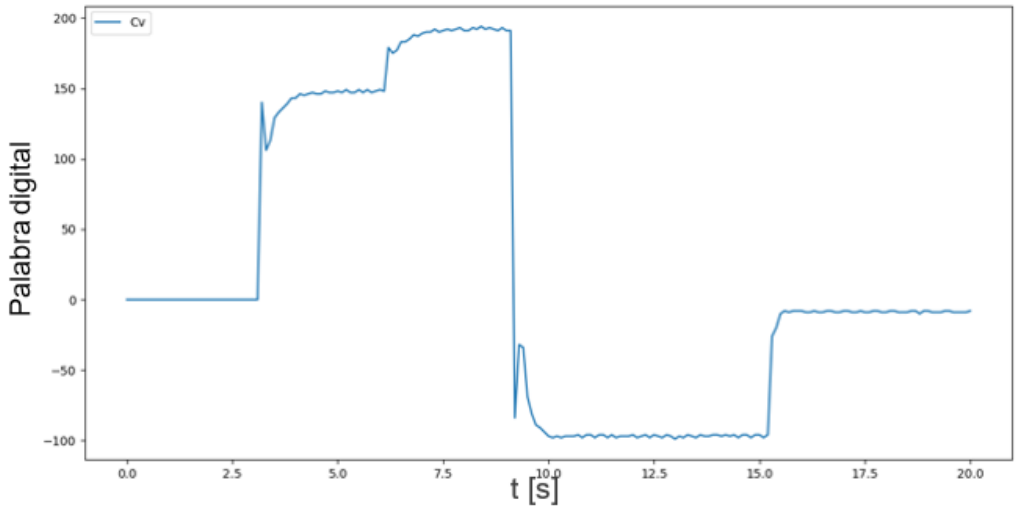


Figura 10. Acción de control en lazo cerrado

De las figuras 10 y 9 se puede determinar la correcta sintonización del controlador PID, puesto que responde al retardo, set-point con error cero y sin sobrepaso, además de tener una acción de control moderada y sin grandes variaciones, la gráfica de la acción de control (figura 9) relaciona en el eje vertical el PWM interpretado en palabra digital (de 0 a 255) y en el eje horizontal el vector de tiempo.

3.4. Sistemas de locomoción.

Para desplazarse por el entorno, los robots móviles cuentan con sistemas de locomoción que pueden ser de ruedas o cintas transportadoras (Diferencial, Síncrona, Triciclo, Omnidireccional etc.) o con patas (robots caminantes).

3.5. Locomoción diferencial.

Un robot móvil de accionamiento diferencial presenta 2-DOF donde cada actuador (motor) está ubicado en el mismo eje. Una forma simple de caracterizar su comportamiento en términos de seguimiento de trayectoria es basado en el uso del modelo cinemático del robot con el modelo dinámico de los motores, que permite diseñar el controlador con un bajo costo computacional. [6] En la figura 10 se puede observar la composición de un robot diferencial, además de especificar los ejes coordenados globales (marco de referencia inercial), un eje de referencia local (marco de referencia ubicado en el robot móvil y que se mueve con este), y un eje fijo asignado al punto destino, teniendo en cuenta los ejes descritos se pueden identificar las diferentes variables para la cinemática, como son la velocidad lineal de desplazamiento v, la velocidad angular de rotación ω, la ubicación en el espacio del robot móvil, la distancia entre el eje local y el punto destino ρ, β es el ángulo de

orientación de la posición final en coordenadas polares, y α es el ángulo de orientación de la posición final en coordenadas polares medido desde el sistema local.

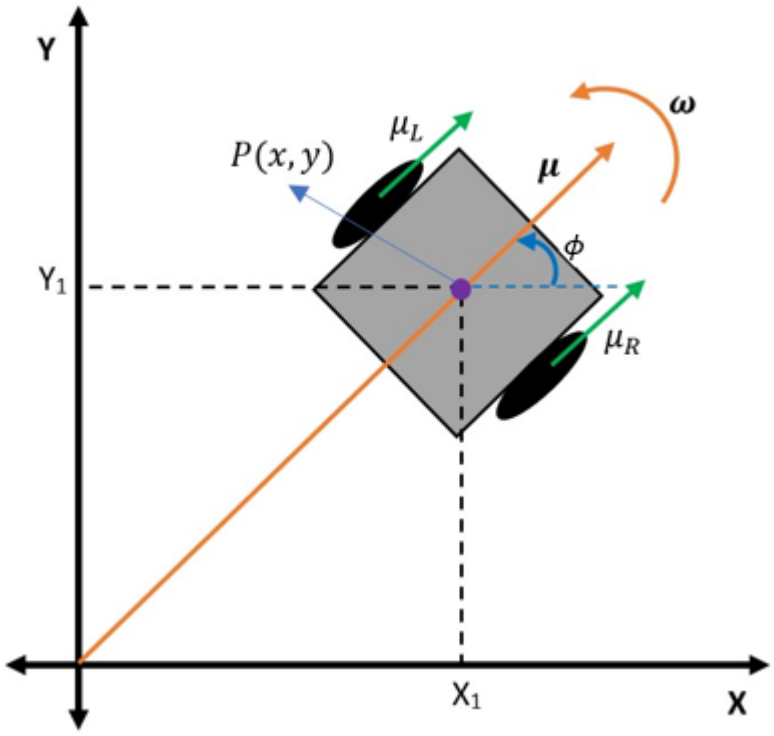


Figura 11. Marcos de referencia de un robot de locomoción diferencial.

Para el cálculo del modelo cinemático del robot tipo diferencial, se realizará el análisis de la cinemática diferencial del robot móvil partiendo de un diagrama representativo (figura 11) sobre el cual se representan las variables que intervienen en dicho modelo y que se enlistan a continuación:

$P(x,y) \rightarrow$ es el punto de referencia ubicado en un punto medio en el eje que atraviesa las llantas del robot.

$\omega \rightarrow$ es la velocidad angular del robot en [rad/s]

$\omega_L, \omega_R \rightarrow$ es la velocidad angular de la rueda izquierda y de la rueda derecha respectivamente en [rad/s]

$\mu \rightarrow$ es la velocidad lineal del robot en [m/s]

$\mu_L, \mu_R \rightarrow$ es la velocidad lineal de la rueda izquierda y de la rueda derecha respectivamente en [m/s]

$\phi \rightarrow$ es la orientación del robot en [rad]

De la figura 10 se puede identificar que el robot tendrá unas coordenadas:

$$P_x = x_1 \tag{17}$$

$$P_y = y_1 \tag{18}$$

Se hace necesario encontrar una expresión que permita conocer los valores de la velocidad lineal y angular en un tiempo t, la velocidad angular es expresada como la variación de la orientación en el tiempo, para el cálculo de las velocidades lineales se descompone el vector de velocidad μ en sus componentes cartesianas como se presenta en la figura 10

Finalmente se obtiene:

$$\dot{p}_x = \mu \cos(\phi) \tag{19}$$

$$\dot{p}_y = \mu \, sin(\phi) \tag{20}$$

$$\omega = \dot{\phi} \tag{21}$$

Representando las ecuaciones (19) y (20) en forma matricial, se obtiene el jacobiano de velocidades y el modelo matemático de velocidades.

$$h(\dot{t}) = Jq(\dot{t}) \tag{22}$$

$$\underbrace{\begin{bmatrix}\dot{p}_x\\ \dot{p}_y\\ \dot{\phi}\end{bmatrix}}_{\dot{h(t)}} = \underbrace{\begin{bmatrix}\cos(\phi) & 0\\ \sin(\phi) & 0\\ 0 & 1\end{bmatrix}}_{J} \underbrace{\begin{bmatrix}\mu\\ \omega\end{bmatrix}}_{\dot{q(t)}} \tag{23}$$

3.5.1. Cinemática diferencial del punto de interés.

Hasta ahora se calculó la cinemática directa para el punto central del robot, para este proyecto el punto de interés se encuentra en el extremo del sensor detector de metales, por lo que se debe realizar el análisis de posición de dicho punto que pasará a llamarse $h(x, y)$

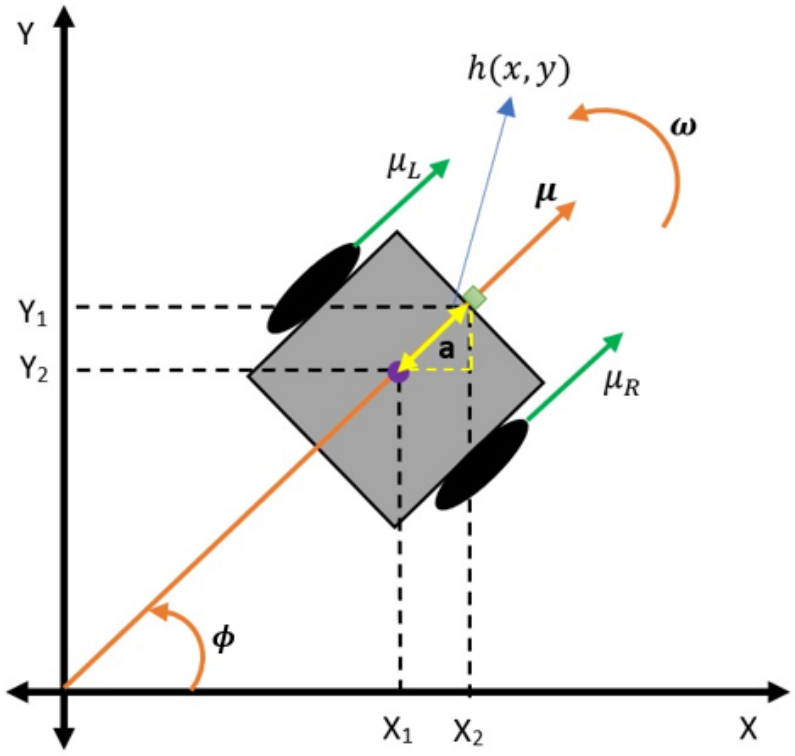


Figura 12. Análisis del punto de interés del robot

De la figura 12 se puede identificar que el robot tendrá unas coordenadas en el punto de interés h para el instante t:

$$h_x = x_1 + x_2 \tag{24}$$

$$h_y = y_1 + y_2 \tag{25}$$

Realizando el análisis vectorial a las componentes cartesianas del vector de posición proyectado desde el punto de origen del robot (como se muestra en la figura 12), se llegan a las siguientes ecuaciones:

$$x_2 = \mathrm{a}\cos(\phi) \tag{26}$$

$$y_2 = \mathrm{a}\sin(\phi) \tag{27}$$

Para encontrar el modelo de velocidades referidas al punto de interés $h(x, y)$ (el cual será el utilizado para la programación de los algoritmos de navegación a desarrollar) se realiza la derivada de las ecuaciones (24) y (25) como se presenta a continuación:

$$h_x = x_1 + \mathrm{a}\cos(\phi) \tag{28}$$

$$h_y = y_1 + \mathrm{a}\sin(\phi) \tag{29}$$

$$\frac{dh_x}{dt} = \frac{dh_x}{dx_1}\frac{dx_1}{dt} + \frac{dh_x}{d\phi}\frac{d\phi}{dt} \tag{30}$$

$$\frac{dh_y}{dt} = \frac{dh_y}{dy_1}\frac{dy_1}{dt} + \frac{dh_y}{d\phi}\frac{d\phi}{dt} \tag{31}$$

Realizando las operaciones indicadas se obtiene el modelo a utilizar representado en forma matricial.

$$\dot{h}_x = \mu \cos(\phi) - a\omega \sin(\phi) \tag{32}$$

$$\dot{h}_y = \mu \sin(\phi) + a\omega \cos(\phi) \tag{33}$$

$$\underbrace{\begin{bmatrix} \dot{h}_x \\ \dot{h}_y \end{bmatrix}}_{\dot{h(t)}} = \underbrace{\begin{bmatrix} \cos(\phi) & -a\sin(\phi) \\ \sin(\phi) & a\cos(\phi) \end{bmatrix}}_{J} \underbrace{\begin{bmatrix} \mu \\ \omega \end{bmatrix}}_{\dot{q(t)}} \tag{34}$$

Desde este punto se le hará referencia al modelo cinemático del robot mediante la ecuación (34) o (35).

$$\dot{h(t)} = J\dot{q(t)} \tag{35}$$

3.6. Odometría.

La odometría es una técnica que tiene por objeto estimar la posición y orientación de un robot móvil a partir del número de vueltas dadas por sus ruedas. La idea principal de la edometría es la integración temporal del movimiento, lo cual lleva inevitablemente a la acumulación de errores [13].

Para llevar la cuenta del número de vueltas dadas, se usan codificadores ópticos de alta precisión sobre los ejes de las ruedas, para poder estimar la posición, se requiere el registro odométrico de las dos ruedas.

Para determinar las velocidades a las que se mueven cada llanta del robot en el instante t, Si se trazan dos líneas colineales al eje de las ruedas desde el punto

inicial del movimiento hasta el punto final, estas se encuentran en un punto en el espacio ubicado a una distancia L conocido como el centro instantáneo de rotación (CIR)

En el cual se pueden dibujar los arcos ΔS_R, ΔS_L y ΔS como la trayectoria que dibuja la llanta derecha, la llanta izquierda y el punto de referencia del robot *g* respectivamente, d es la distancia entre las llantas como lo muestra la figura 13:

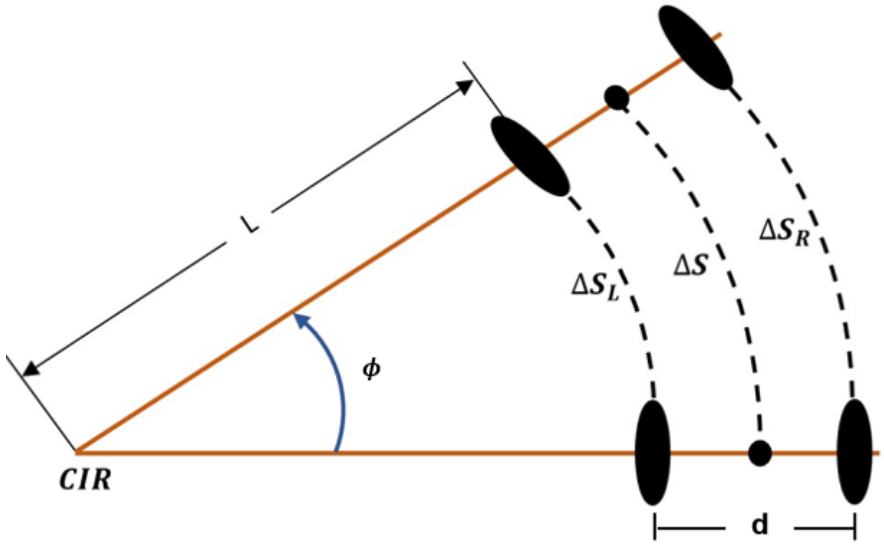


Figura 13. Centro instantáneo de rotación generado al realizar un desplazamiento curvilíneo.

Con esto en mente se puede determinar el incremento del ángulo $\Delta\phi$, con la proporción entre la longitud del arco equivalente al desplazamiento ΔS_R o ΔS_L y el radio del sector circular formado es decir la distancia al centro instantáneo de rotación L o $L + d$:

$$\frac{\Delta S_L}{L} = \Delta\phi \tag{36}$$

$$\frac{\Delta S_R}{L + d} = \Delta\phi \tag{37}$$

De lo que se puede determinar que $\Delta\phi$ queda definido como:

$$\Delta\phi = \frac{\Delta S_R - \Delta S_L}{d} \tag{38}$$

Para determinar el desplazamiento del robot en función del desplazamiento de cada una de sus llantas se establecen la siguiente ecuación:

$$\frac{\Delta S}{L + \frac{d}{2}} = \Delta\phi \tag{19}$$

Igualando los términos semejantes en las ecuaciones (38) y (19) se obtiene:

$$\frac{\Delta S}{L + \frac{d}{2}} = \frac{\Delta S_R - \Delta S_L}{d} \tag{40}$$

Finalmente, luego de despejar y operar los términos semejantes se encuentra la ecuación que determina el desplazamiento del robot en términos del desplazamiento de sus llantas.

$$\Delta S = \frac{\Delta S_R + \Delta S_L}{2} \tag{41}$$

Para este análisis se desea obtener las velocidades que es determinar el desplazamiento en cualquier instante t, por lo cual se debe tener en cuenta el radio de las ruedas *R* que en el robot móvil son de 16 [mm] de radio, la distancia entre las llantas *d*, que para este robot es de 82 [mm], se puede conocer la velocidad angular de cada llanta por medio de los encoders ubicados en estas, y a partir de la velocidad angular de cada llanta se encuentran las ecuaciones para determinar la velocidad angular y lineal del robot, aplicando las ecuaciones de análisis cinemático (42) y (47), la ecuación (43) es la ecuación característica que describe la velocidad de un punto *r* en un cuerpo rígido, conociendo otro punto de este, este caso *g* que es el punto central del robot:

$$\mu_g = R\omega \tag{42}$$

$$\mu_r = \mu_g + \mu_{r/g} \tag{43}$$

$$\mu_{r/g} = \omega * (\frac{d}{2}) \tag{44}$$

Se obtiene de la ecuación (38) y (41):

$$\omega = (\mu_R + \mu_L)/dR \tag{45}$$

$$\mu_g = \frac{R}{2}(\mu_R + \mu_L) \tag{46}$$

Se convierte la ecuación (43) en:

$$\mu_r = \mu_g + \frac{\omega d}{2} \tag{47}$$

$$\mu_L = \mu_g - \frac{\omega d}{2} \tag{48}$$

Se conoce de la ecuación (42) que:

$$\omega_r = \frac{\mu_r}{R} \tag{49}$$

$$\omega_L = \frac{\mu_L}{R} \tag{50}$$

Obteniendo las ecuaciones:

$$\omega_r = \frac{\mu_g + \frac{\omega d}{2}}{R} \tag{51}$$

$$\omega_L = \frac{\mu_g - \frac{\omega d}{2}}{R} \tag{52}$$

Mediante la ecuación (47), (48), (51) y (52) se determina la velocidad lineal y angular de la rueda derecha e izquierda respectivamente, mediante la ecuación (45) se determina la velocidad lineal del robot

Finalmente, las ecuaciones 42-47 sirven para determinar la velocidad a la que debe ir cada rueda para cumplir la velocidad lineal y angular planteada, de igual modo se puede conocer la velocidad lineal y angular del robot partiendo desde la velocidad de sus redas.

3.7. Comunicación inalámbrica

Los robots móviles son autónomos y de baterías, esto les permite tener un mayor rango de movimiento sin tropiezos a causas de cables, por lo cual se hace necesario implementarles un sistema de comunicación inalámbrica.

Para este proyecto se implementó comunicación vía bluetooth con frecuencia de banda de 2.4 GHz, mediante Python se realiza la selección del robot al cual se le enviará y/o recibirá información, por esta banda se envían al robot los valores de velocidad angular y lineal a los cuales debe moverse para lograr llegar al punto objetivo, también se recibe la velocidad lineal y angular con la cual se desplaza el robot para los cálculos del algoritmo en Python.

Se utilizó el módulo HC 05, el cual es una tarjeta capaz de comunicarse vía bluetooth a un rango de 2.5 GHz y con un alcance de hasta 10 metros, programado a 112500 bps mediante comandos seriales AT, el algoritmo de Python es capaz de recibir, interpretar y enviar información hacia cada robot móvil.

4. CAPITULO 2

Ya definido el sistema de locomoción, se necesita controlar los actuadores de tal forma que le permitan a los robots tener una velocidad lineal y angular que siga un camino o que se desplace hasta un punto objetivo, en este capítulo se presentan los algoritmos para de seguimiento de camino y controlo de posición para lograr que el robot siga un camino dado.

En robótica móvil se definen los algoritmos de control de robots de navegación autónoma bajo 3 niveles de control, estos son el control de bajo nivel, control de nivel medio y control de alto nivel [14].

El control de alto nivel lo conforma los algoritmos de generación de trayectorias para evasión de obstáculos programado, en este nivel se generan los vectores que almacenan el conjunto de puntos de la ruta asegurando un paso libre sin obstáculos.

El control de nivel medio lo conforman los algoritmos de control de posición, seguimiento de trayectorias y seguimiento de camino, en este punto el controlador calcula las velocidades a la que debe mover cada rueda para llegar al punto o pasar por cada punto definido en el control de alto nivel, es decir toma un punto como entrada y genera velocidades de salida.

El control de bajo nivel es el implementado directamente para control de los actuadores que generan el movimiento en el robot, en este nivel el controlador busca garantizar la velocidad indicada por el controlador de nivel medio con error cero de cada llanta.

Garantizar la velocidad de cada llanta llevará al robot a moverse por cada punto trazado por el algoritmo generador de trayectorias o planificador de ruta y con la velocidad indicada por el algoritmo de seguimiento de ruta cumpliendo así el objetivo de alcanzar una meta evadiendo obstáculos.

4.1. Diseño de los algoritmos de control de nivel medio.

Como se había descrito con anterioridad, el robot dispone de un planificador de rutas y de un controlador de seguimiento de rutas.

En donde el planificador de ruta encuentra la ruta (secuencia de puntos) desde la posición del robot hasta la meta, sin chocarse con los obstáculos presentes en el ambiente, el control de seguimiento de ruta realiza el cálculo de las velocidades lineales y angulares que el robot debe tener para pasar por cada punto descrito, el control de los actuadores calcula y controla la velocidad a la que cada llanta debe moverse para que el robot pase por cada punto descrito por el planificador, este

control de actuadores también es conocido en la literatura como control de bajo nivel, siendo el control de trayectoria de nivel medio, y el planificador de rutas un control de alto nivel.

Para el diseño de los algoritmos de control de nivel medio se parte del previo conocimiento de la ruta a seguir, la cual será simbolizada como h , y $\dot{q}$ representa el vector de velocidades de cada rueda para alcanzar cada punto de la trayectoria. El objetivo del algoritmo de control de bajo nivel es determinar un conjunto de velocidades $\dot{q}$ que permitan al robot desplazarse a través de la trayectoria h .

Partiendo del modelo cinemático encontrado en la ecuación (35), se analiza que $q(\dot{t})$ es el vector el cual almacena las velocidades que se desean calcular por lo tanto se debe acomodar el modelo jacobiano de velocidad hasta encontrar una expresión para determinar la velocidad de los actuadores de la ecuación:

$$h(\dot{t}) = J(q(t))q(\dot{t}) \tag{53}$$

Se puede encontrar $q(\dot{t})$ con la inversión de la matriz jacobiana multiplicada por la derivada de la trayectoria como se muestra a continuación $q(\dot{t})$:

$$q(\dot{t}) = J\big(q(t)\big)^{-1} h(\dot{t}) \tag{54}$$

Como se puede observar en la ecuación (49) se hace necesario encontrar la derivada del camino a seguir $h(\dot{t})$ por lo que se procede a diseñar el algoritmo mediante un esquema de solución basado en los errores entre el camino deseado y el camino real considerando las siguientes ecuaciones:

$$h_e = h_d - h \tag{55}$$

Donde h_e representa el error de trayectoria, h_d la trayectoria deseada y h la trayectoria real.

Para el cálculo de la derivada del error se aplica la derivada a h_d y h con respecto al tiempo, como se muestra a continuación

$$\dot{h_e} = \dot{h_d} - \dot{h} \tag{56}$$

Se realiza el despeje $\dot{h}$ y se remplaza la ecuación obtenida en la ecuación (56) y se procede al diseño del control de posición de los robots.

$$\dot{q(t)} = J\big(q(t)\big)^{-1} (\dot{h_d} - \dot{h_e}) \tag{57}$$

4.2. Control de posición

El objetivo de este control es que el robot diferencial ejecute de forma autónoma movimientos previamente planificados hasta alcanzar una posición objetivo. Se pretende que el robot móvil siga una ruta de forma autónoma. Este problema puede formularse como la obtención de las leyes de control que permitan estabilizar al robot sobre un punto de trabajo. [15]

Para el controlar la posición de robots móviles diferenciales, se pueden aplicar varias técnicas, entre las que se encuentran:

Técnica de teorías de control, mediante modelos dinámicos del sistema (del robot) se puede encontrar un controlador que lo estabilice y a la vez realice la tarea deseada.

Técnica de métodos geométricos para el seguimiento o planificación de caminos con algoritmos matemáticos.

Técnicas realizadas para sistemas inteligentes o inteligencia artificial, las cuales pueden ser algoritmos genéticos, redes neuronales, el algoritmo del vecino más cercano o lógica difusa, para el control de movimiento.

Para este proyecto se hará uso de técnicas basadas en teoría de control buscando estabilizar el modelo matemático con técnicas como el criterio de estabilidad de Lyapunov.

4.2.1. Análisis mediante el criterio de estabilidad de Lyapunov

El algoritmo de Lyapunov funciona bajo el principio de una función de energía, la cual pasa a llamarse función de Lyapunov, esta se puede aplicar a ecuaciones diferenciales o de diferencias para sistemas lineales o no lineales. En función del

signo de la función de Lyapunov y de su derivada con respecto al tiempo, se pueden obtener conclusiones sobre el tipo de estabilidad de un estado de equilibrio. [16]

Método directo de Lyapunov indica que un sistema que posea un estado de equilibro en el origen, si existe una función escalar denominada como función candidata V (x), con derivas parciales continuas y que satisfaga las siguientes condiciones [16].

$$v(x) > 0 \tag{58}$$

$$\frac{dv}{dx} < 0 \tag{59}$$

entonces el estado de equilibrio en el origen es uniforme y asintóticamente estable.

4.2.2. Diseño del algoritmo de control de posición.

Se considera la ecuación (57) para el cálculo de la velocidad de los actuadores, para el caso del control de posición la trayectoria deseada se limita a alcanzar un punto en el espacio por lo cual h_d es un valor constante así $\dot{h_d} = 0$ por lo cual se obtiene la siguiente ecuación:

$$\dot{q(t)} = -J\big(q(t)\big)^{-1}\, \dot{h_e} \tag{60}$$

Se aplicará el criterio de Lyapunov para encontrar una matriz K que asegure la estabilidad del modelo basado en el error de camino.

El criterio de estabilidad de Lyapunov conlleva proponer una función que se pasará a nombrar como función candidata, para este caso se hará uso del error cuadrático:

$$v(h_e) = \frac{h_e^T h_{ex}^2}{2} \tag{61}$$

Donde v representa la función candidata de Lyapunov en función de los errores de trayectoria, seguidamente se debe obtener la derivada con respecto al tiempo de la función candidata, para el caso del error de trayectoria h_{ex}, se debe aplicar el cuadrado al error debido a que se hará uso del error cuadrático:

$$v(h_{ex}) = \frac{h_e^T h_{ex}^2}{2} \tag{62}$$

Obteniendo:

$$\dot{v} = h_e^T \dot{h_e} \tag{63}$$

Se obtiene la ecuación (63) con la derivada de la función candidata para las componentes X y Y en forma matricial.

Finalmente se deben verificar las condiciones de estabilidad descritas según el criterio de Lyapunov (para la función candidata y su derivada).

Para un mejor y más sencillo análisis se realizará un cambio en $\dot{h_e}$ el cual pasará a estar representado con la matriz K y los errores h_e esto para no depender de valores de las derivadas que puedan inestabilizar el modelo, se tendrá:

$$\dot{h_e} = -Kh_e \tag{64}$$

Remplazando en la ecuación (63) y aplicando el criterio de estabilidad se obtiene:

$$\dot{v} = -h_e^T K h_e < 0 \tag{65}$$

De la cual podemos definir que si K es una matriz positiva el sistema será asintóticamente estable, aplicando entonces a la ecuación (60) se obtiene el algoritmo de control de posición el cual genera las velocidades para desplazarse al punto seleccionado.

$$\dot{q(t)} = J\big(q(t)\big)^{-1} K h_e \tag{66}$$

Sin embargo, este algoritmo puede presentar oscilaciones a medida que el robot se acerca al punto de meta, por lo cual conviene hacer modificaciones a la matriz K para que esta cambie su valor mientras el robot se desplaza, se puede considerar entonces una ganancia adaptativa [16]:

$$K = \begin{bmatrix} \frac{Kmax}{1+k_1 d} & 0 \\ 0 & \frac{Kmax}{1+k_1 d} \end{bmatrix} \quad (67)$$

$$d = \sqrt{h_{ex}^2 + h_{ey}^2} \quad (68)$$

Para la cual se establece un límite máximo para la matriz K aquí llamado como Kmax siendo 'd' la distancia euclidiana del error de trayectoria, y una ganancia k_1 que se ajustará para lograr que el algoritmo converja de forma más rápida a cero. Se hicieron pruebas experimentales variando el valor de las constantes K_{max} y K_1 para un punto objetivo (punto de meta) hasta determinar el valor que mejor respuesta entrega eliminando el error en estado estable, a continuación, se presentan las gráficas correspondientes a la variación de las constantes en los rangos [2, 10] para Kmax y [10, 15] para K1, con un mismo punto (x, y) objetivo.

Puntos objetivos; X=2 Y=3

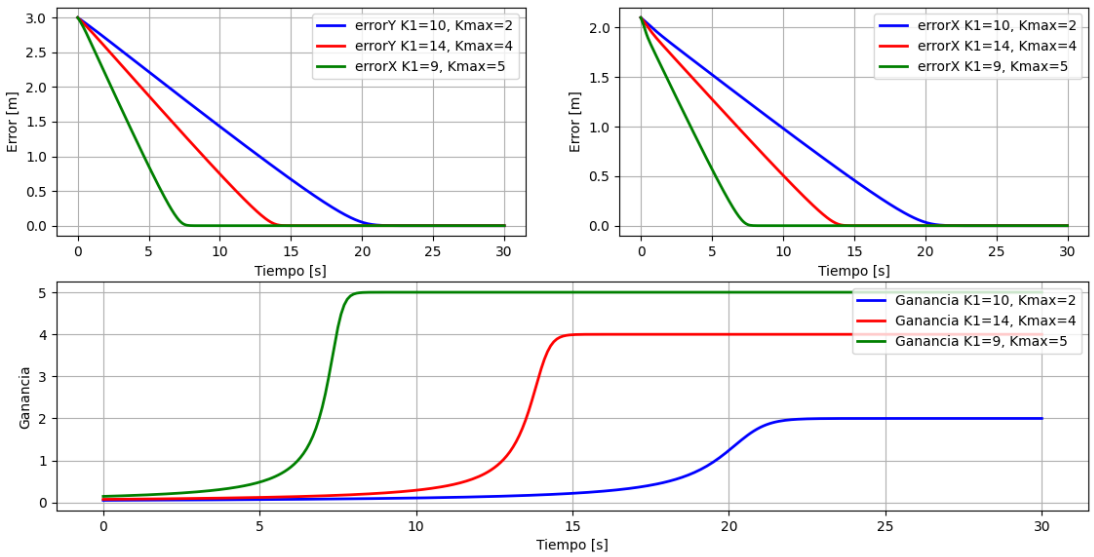


Figura 13. errores y ganancia con las constantes especificadas para el punto objetivo

Se observa en la gráfica 13 como el error en X y en Y disminuye conforme la ganancia aumenta hasta alcanzar el valor máximo asignado.

Puntos objetivos; X=-1.2 Y=2.6

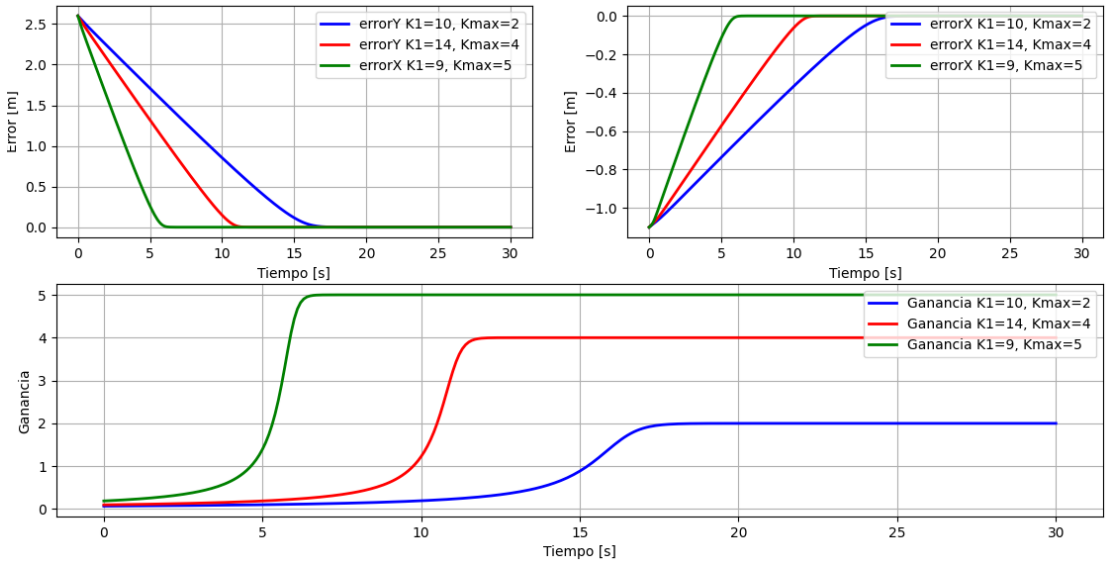


Figura 14. errores y ganancia con las constantes especificadas para el punto objetivo 2.

Se puede determinar que con este rango de valores de la matriz K (el parámetro de ganancia en las figuras 13 y 14) el robot alcanza la meta con error cero (0), por consiguiente, se determinará como parámetros para la matriz K los valores que generen la acción de control más suave y que garantice llegar al punto objetivo. Luego del análisis del comportamiento experimental del robot se determinó los parámetros de la matriz K en: $k_{max} = 2$ y $k_1 = 10$ obteniendo la matriz K:

$$K = \begin{bmatrix} \frac{2}{1+10d} & 0 \\ 0 & \frac{2}{1+10d} \end{bmatrix} \quad \textbf{(69)}$$

4.2.3. Prueba simulada del algoritmo de nivel medio (control de posición).

Para la simulación de los algoritmos se realiza la programación en Python del modelo cinemático directo diferencial, los cálculos de los errores de trayectoria, el modelo de control y el entorno de simulación como lo muestra el diagrama de flujo de la figura 12.

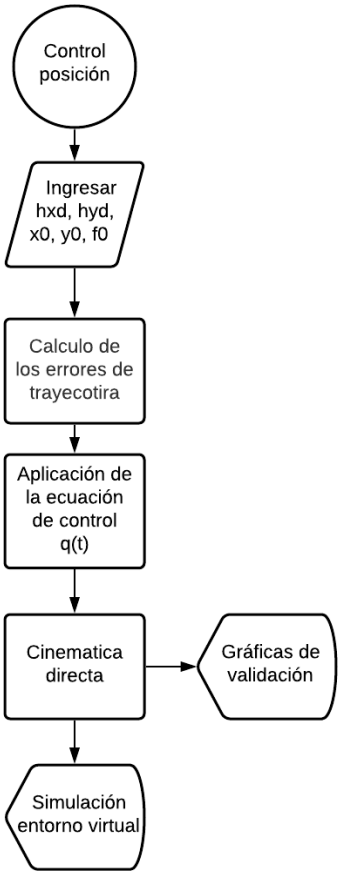


Figura 14. Diagrama de flujo para programación del control de posición.

Los parámetros hxd, hyd corresponden a las coordenadas de la meta, x0, y0, f0 corresponden a la posición inicial y orientación del robot que para este caso son (0,0,0) respectivamente, y la ecuación de control corresponde a la ecuación (66) con la matriz K aplicada, el bloque de cinemática directa representa las ecuaciones (35) encontradas en el apartado anterior y el bloque de simulación del entorno virtual representa las gráficas de validación del robot virtual.

4.2.4. Aplicación de cada Fase del diagrama de flujo del control de posición.

- **Ingresar las condiciones iniciales del robot.**

Se crean variables en el algoritmo Python donde se almacenarán las condiciones iniciales del robot y el punto objetivo o meta, para el desarrollo de este proyecto se crearon las siguientes variables, la variable h corresponde al número de puntos para la simulación:

```
x1 = Zeros(H)
y1 = Zeros(H)
phi = Zeros(H)
hx = Zeros(H)
hy = Zeros(H)
x1[0] = 0
y1[0] = 0
hx[0] = 0
hy[0] = 0
phi[0] = 0
```

- **Cálculos de los errores de trayectoria.**

Se crean vectores que almacenaran los datos del error de la trayectoria de h_{ex} y h_{ey} , estos se calculan mediante la diferencia entre la posición actual del robot h_yy h_x con la posición del punto deseado o meta p_{yd} y p_{xd} mediante la ecuación:

$$h_{ex} = p_{xd} - h_x$$

$$h_{ey} = p_{yd} - h_y$$

Estos vectores formaran la matriz de errores de trayectoria h_e de la siguiente forma:

$$h_e = [h_{ex}; h_{ey}]$$

```
hex[n] = hxd - hx[n]
hey[n] = hyd - hy[n]
he = array([[hex[n]],[hey[n]]])
```

- **Aplicación de la ecuación de control**

Se programa la ecuación (57) aplicando los errores calculados y la matriz jacobiano, remplazando a K con la determinada en la ecuación (60), como se muestra, la variable h_e corresponde a un vector de 2x1 con el error de posición en X y Y:

$$\dot{q(t)} = \begin{bmatrix} \cos(\phi) & -a\sin(\phi) \\ \sin(\phi) & a\cos(\phi) \end{bmatrix}^{-1} * \begin{bmatrix} \frac{2}{1+10d} & 0 \\ 0 & \frac{2}{1+10d} \end{bmatrix} * h_e$$

De donde se sabe que las velocidades lineales y angulares estarán dadas por:

$$\dot{q(t)} = \begin{bmatrix} \mu \\ \omega \end{bmatrix}$$

$\dot{q(t)}$ este vector de velocidades es la señal de referencia para el lazo de control PID y mediante las ecuaciones (51) y (52) se obtienen las velocidades angulares de cada rueda partiendo de las velocidades lineales y angulares del robot.

Para el desarrollo de este proyecto se programaron dichas ecuaciones en Python de la siguiente forma:

```
# Matriz Jacobiana

    J = array([[ cos(phi[n]), - a*sin(phi[n])],

               [ sin(phi[n]),  a*cos(phi[n])]])

   distancia = sqrt((hxd - hx[n])**2+(hyd - hy[n])**2)

   kmax = 2

   k1 = 10

   gain[n] = kmax/(1+k1*distance)

     K = array([[ gain[n], 0],

               [ 0,  gain[n]]])

 qpRef = linalg.pinv(J)@K@he
```

- **Cinemática directa.**

Para la programación de la cinemática directa se debe calcular la posición del robot luego de aplicarle los vectores de velocidad $q(\dot{t})$ calculados en el paso anterior para esto se usará inicialmente el método de integración de Euler hacia adelante para hallar la coordenada $p = (x, y)$ como se muestra a continuación:

$$x_1[n] = \mu * t_s + x_1[n-1]$$

$$y_1[n] = \mu * t_s + y_1[n-1]$$

$$\phi[n] = \omega * t_s + \phi[n-1]$$

Donde μ es la velocidad lineal calculada en el paso anterior y t_s es el periodo de muestreo de 0.1 segundos, y_1 y x_1 representan la posición del origen del robot, ϕ representa la orientación del robot. Para encontrar las coordenadas del punto de interés $h = (x, y)$ se programarán las ecuaciones (28) y (29):

$$h_x = x_1 + \mathrm{a}\cos(\phi)$$

$$h_y = y_1 + \mathrm{a}\sin(\phi)$$

En donde h_y y h_x representan la posición del punto de interés siendo en este caso la herramienta de detección de metal y ***a*** es la longitud desde el origen de coordenadas del robot hasta la ubicación del punto de interés.

Para el desarrollo de este proyecto se programó como se muestra a continuación:

```
x1p = uRef[n]*cos(phi[n+1])

y1p = uRef[n]*sin(phi[n+1])


x1[n+1] = x1[n] + ts*x1p

y1[n+1] = y1[n] + ts*y1p


hx[n+1] = x1[n+1]+a*cos(phi[n+1])

hy[n+1] = y1[n+1]+a*sin(phi[n+1])
```

- **Entornos de simulación**

Para la elaboración del entorno de simulación virtual, se toma el diseño CAD del robot en formato STL y en conjunto con las funcionalidades que el módulo PyVista de Python ofrece se realiza la simulación 3D mostrando gráficamente la trayectoria y el movimiento del robot.

```
ArchivoSTL = "ACTUALSTL"

color = ['white','white','white','green','red','gray','gray','black','black','white','white','white','white','white','black','white']

Robot = robotics(ArchivoSTL,color)


xmin = -2

xmax = 2

ymin = -2

ymax = 3

zmin = 0

zmax = 2


Limites_ejes = [xmin, xmax, ymin, ymax, zmin, zmax]

uniciclo.configureScene(Limites_ejes)

uniciclo.initTrajectory(hx,hy)

uniciclo.initRobot(x1,y1,phi,escala)

uniciclo.startSimulation(1,ts)
```

4.2.5. Simulación con diferentes puntos de meta

A continuación, se presentan imágenes de simulación correspondientes a diferentes puntos x,y de la meta objetivo.

Posición objetivo X = 0, Y = 2.4

Posición inicial X=0, Y=0, ϕ=0

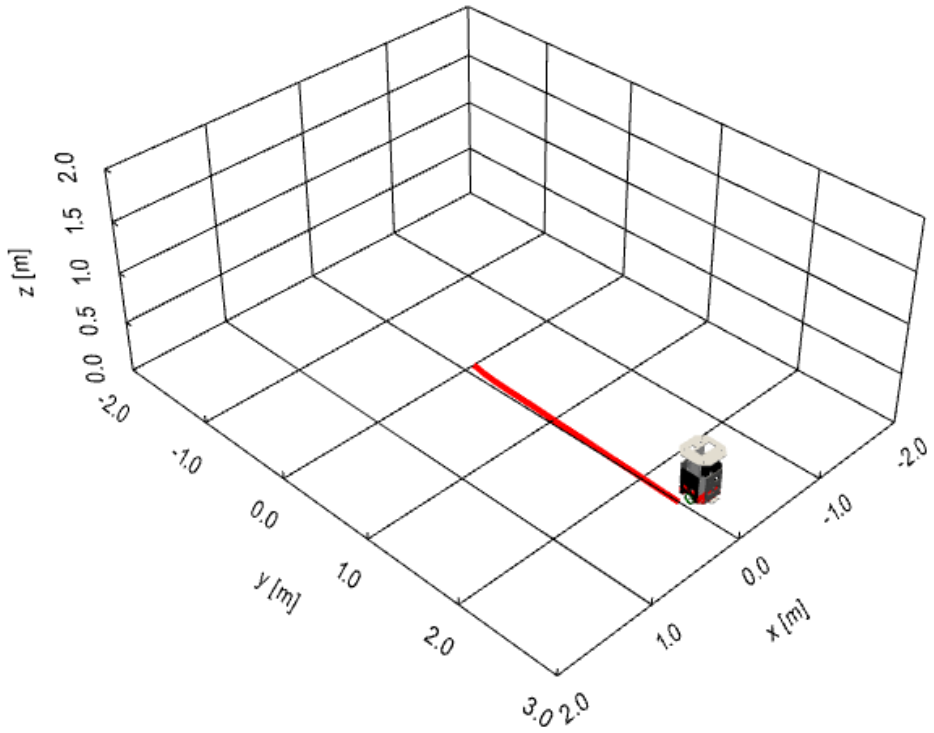


Figura 15. Entorno 3D de simulación.

En la figura 15 se observa mediante una línea roja los puntos por donde pasó el robot hasta llegar a la meta X=0 y =2.4, el robot inicia con una orientación de cero grados (0°) y se debe desplazar hasta un valor Y de posición por lo cual rota aproximadamente 90° hasta el eje Y, la línea roja se dibuja desde el punto de interés $h(x, y)$ que se encuentra a 0.07 metros de distancia al eje central que une las ruedas del robot diferencial.

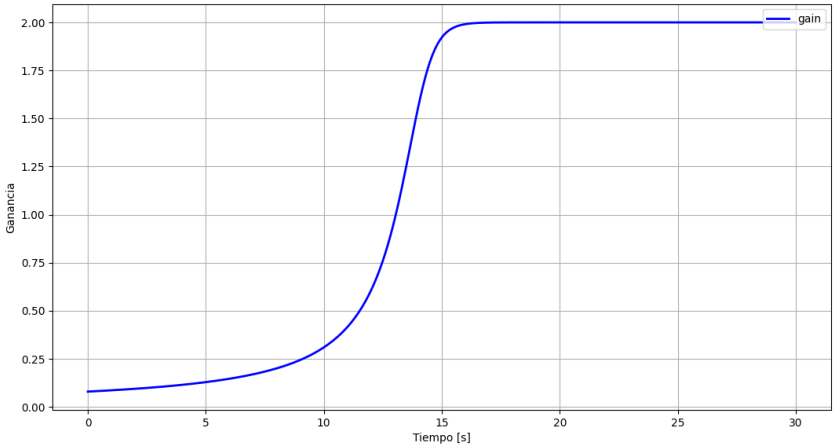


Figura 16. Ganancia generada para llegar a la meta X=0, Y=2.4, esta ganancia representa la matriz K que estabiliza el sistema determinado en la sección anterior.

En la figura 16 se observa de forma gráfica la variación del valor de la ganancia adaptativa esta se calcula mediante las ecuaciones (67) y (68) con un arranque suave y teniendo como máximo valor el parámetro definido previamente como $k_{max} = 2$, Esta ganancia representa la matriz K que estabiliza el sistema determinado en la sección anterior.

Velocidad Lineal [m/s]

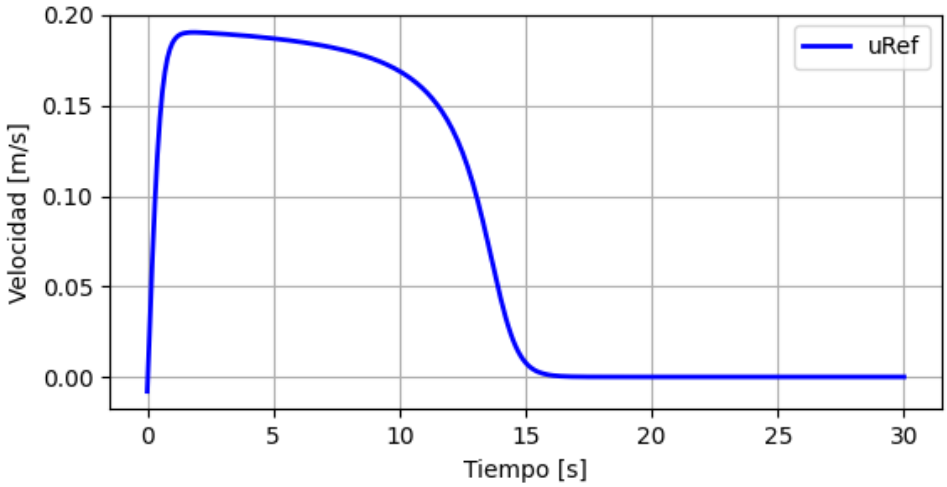


Figura 17. Velocidad Lineal desarrollada.

En la figura 17 se observa la velocidad lineal a la cual se debe mover el robot para alcanzar el punto objetivo, esta es la primera fila del vector de velocidades calculado por el algoritmo de control, con la ecuación (66) y con la matriz K de la ecuación (67) y (68).

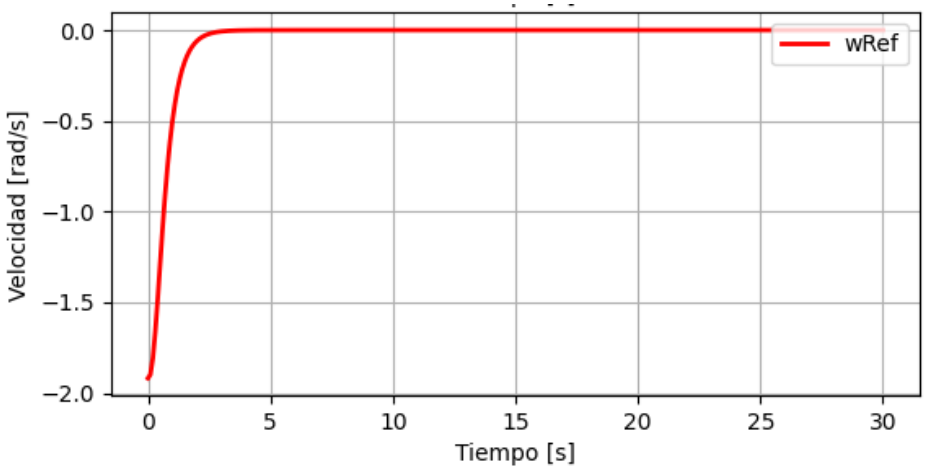


Figura 18. Velocidad angular desarrollada

la figura 18 se observa la velocidad angular con la cual se debe mover el robot para llegar al punto objetivo, esta es la segunda fila del vector de velocidades calculado por el algoritmo de control, con la ecuación (66) y con la matriz K de la ecuación (67) (68).

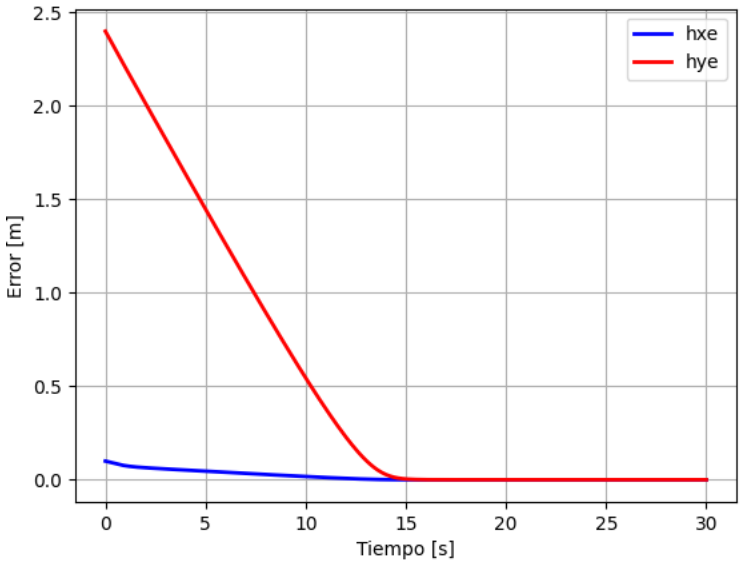


Figura 19. Error de posición

En la figura 19 se observa el error de posición en metros que converge a cero (0) indicando que se alcanza de forma satisfactoria la posición objetivo, estos se calculan mediante la resta de las componentes X y Y de la trayectoria como se especificó en la implementación de la simulación del algoritmo de posición.

Simulación con la Posición X = 2, Y = 2

Posición inicial X=0, Y=0, ϕ=0

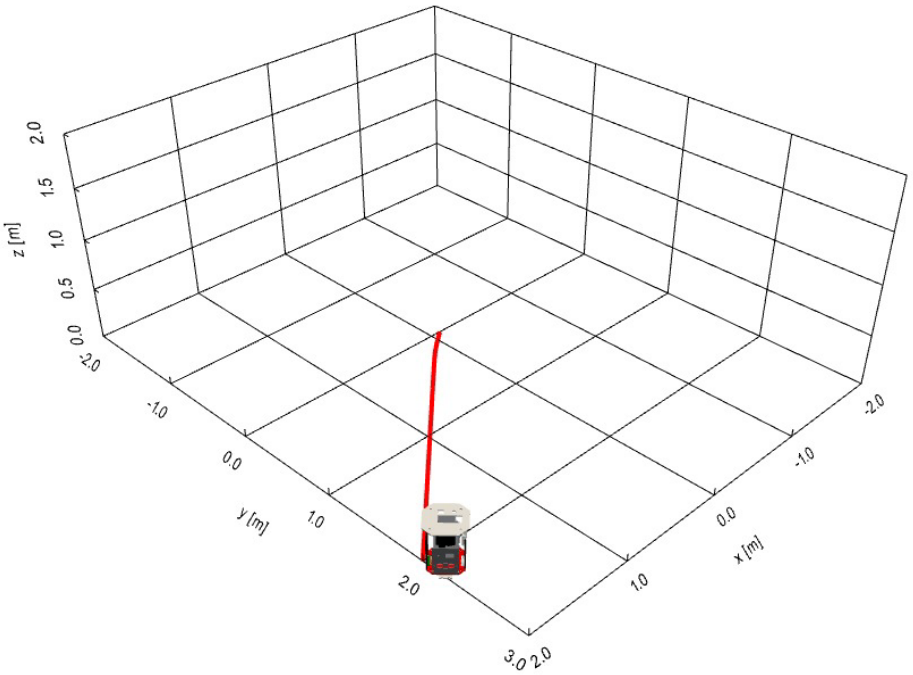


Figura 20. Simulación entorno 3D

En la figura 20 se presenta el entorno virtual y como se desplaza el robot según las condiciones iniciales y finales establecidas, Se puede observar mediante la simulación en 3D que el robot llega al punto objetivo.

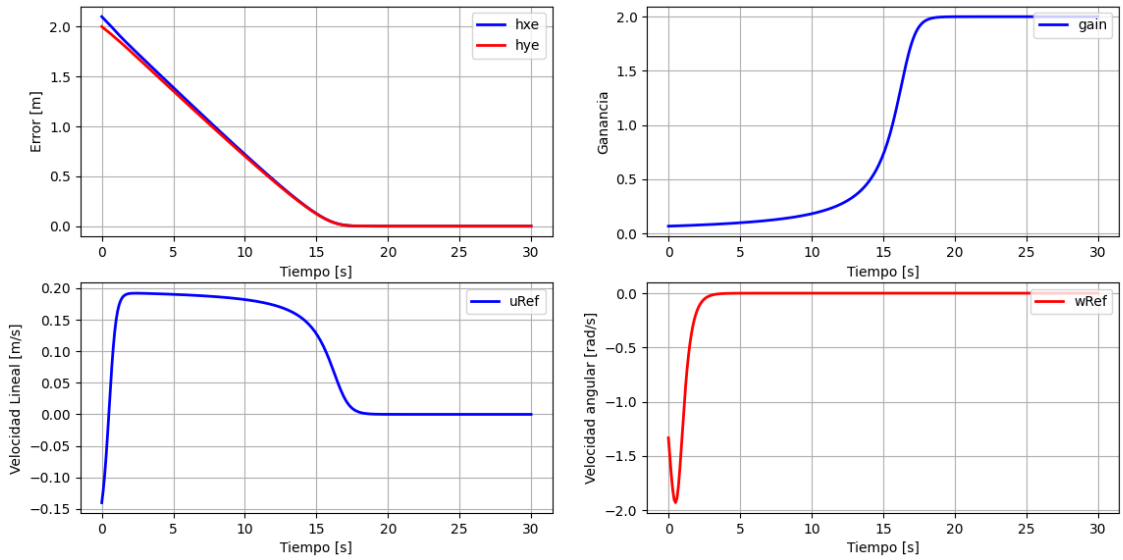


Figura 21. Errores de posición.

En la figura 21 se puede observar que los errores convergen en cero por lo cual el robot llega a la posición objetivo.

Se observa en la figura 21 la variación de la ganancia a medida que se acerca a la meta.

También se presentan las gráficas de velocidades que llevan al robot hasta el punto objetivo.

Simulación con la Posición X = -1, Y = 2.5

Posición inicial X=0, Y=0, ϕ=0

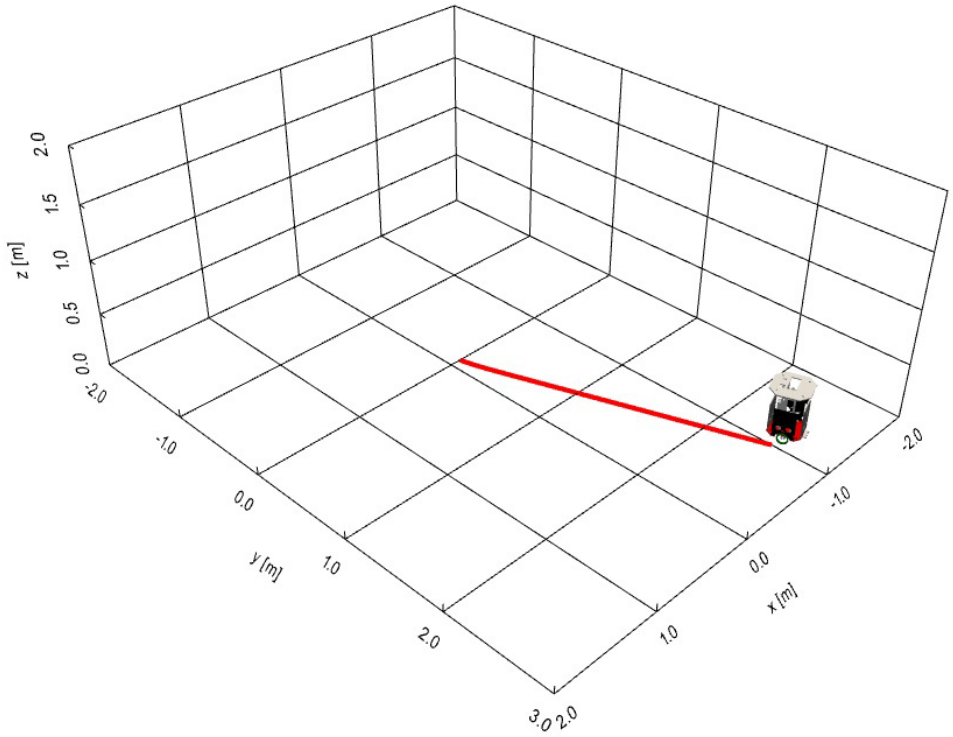


Figura 22. Simulación entorno 3D.

Se verifica en la figura 22 que el robot alcanza la posición de la meta.

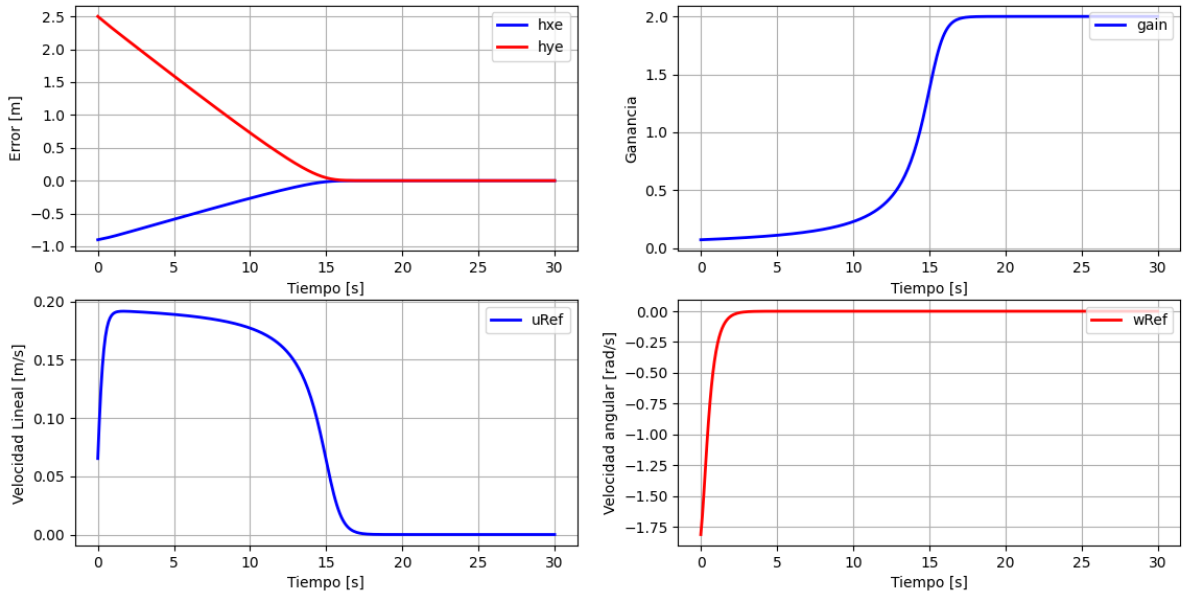


Figura 23. Error de posición.

Se sigue comprobando en la figura 23 que los errores de posición convergen en cero (0). Estas velocidades se envían al controlador de bajo nivel el cual calcula a qué velocidad debe mover cada llanta y aplica la técnica de control previamente calculada en el capítulo anterior.

4.3. Control de Seguimiento de Camino

Para el seguimiento de camino, se toma como referencia o punto de partida el control de posición en el cual el robot se mueve a un punto inicial meta y luego cuando el robot lo alcanza este punto se actualiza teniendo así un nuevo punto meta hasta llegar al final del camino (a la meta), es decir el robot sigue la secuencia de puntos que dictan el camino hasta la meta.

Para el cálculo de este algoritmo se toma en consideración el modelo descrito en la ecuación (60) y el criterio descrito en la ecuación (63):

$$\dot{q(t)} = -J\big(q(t)\big)^{-1}\,\dot{h_e} \tag{60}$$

$$\dot{v} = h_e^T \dot{h_e} \tag{63}$$

teniendo en cuenta que a diferencia del control de posición no se tiene un único punto sino un camino por lo cual $\dot{h_d} \neq 0$, por lo que el modelo para el control de camino resulta:

$$\dot{q(t)} = J\big(q(t)\big)^{-1}(\dot{h_d} + Kh_e) \tag{70}$$

Como se había descrito anteriormente, $\dot{h_d}$ es la velocidad deseada para recorrer el camino, para el análisis se realizará un cambio de variable a $\dot{p_d}$ que indica la velocidad deseada para llegar a un punto dentro del camino, de esta forma se tendrá una velocidad deseada para cada punto según el robot se acerque a este, entonces $\dot{h_d} \neq \dot{p_d}$, el algoritmo de control de camino calcula la velocidad que lleva al robot a través de estos puntos, por lo cual al considerar esto aparece un nuevo termino, el error de velocidades deseadas δ_e que es la resta entre el vector de velocidad deseada para recorrer el camino $\dot{h_d}$ y el vector de velocidad de un punto especifico $\dot{p_d}$, teniendo como resultado el modelo de velocidades:

$$\dot{q(t)} = J\big(q(t)\big)^{-1}(\dot{p_d} + Kh_e) \tag{71}$$

$$\delta_e = \dot{h_d} - \dot{p_d} \tag{72}$$

Se puede expresar $\dot{h}$ tomando la ecuación (57) y (63):

$$\dot{q(t)} = J\big(q(t)\big)^{-1}(\dot{h_d} - \dot{h_e}) \tag{57}$$

$$\dot{h_e} = -Kh_e \tag{64}$$

como:

$$\dot{h} = \dot{p_d} + K\dot{h_e} \tag{73}$$

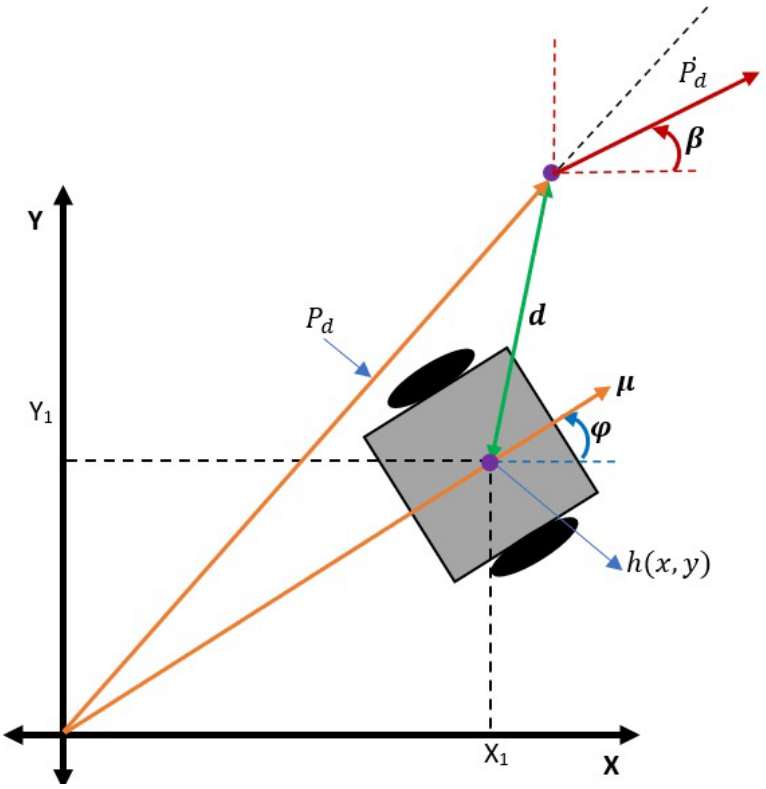


Figura 24. Análisis del seguimiento de camino.

De la figura 24 se puede determinar las componentes rectangulares del vector $\dot{p_d}$ siendo este la derivada en el punto p_d (cualquier punto dentro del camino) y un vector tangente al camino, se obtiene:

$$\dot{p}_x = |\dot{p_d}| \cos(\beta) \tag{74}$$

$$\dot{p}_y = |\dot{p_d}| \sin(\beta) \tag{75}$$

$$\beta = \tan^{-1}\left(\frac{\Delta p_{dy}}{\Delta p_{dx}}\right) \tag{76}$$

Al igual que para el control de posición se aplicará el criterio de Lyapunov para asegurar la estabilidad del modelo obtenido en la ecuación (70).

Se toma la ecuación (62) la misma ecuación candidata de Lyapunov y su derivada (63):

$$v(h_e) = \frac{h_e^T h_e^2}{2} \tag{62}$$

$$\dot{v} = h_e^T \dot{h_e} < 0 \tag{63}$$

Para encontrar una matriz K que cumpla con la estabilidad se analizará mediante la función candidata partiendo del modelo cinemático del robot (35) y las ecuaciones de errores encontradas hasta ahora (73, 56 y 72):

$$\dot{h} = J\dot{q} \tag{35}$$

$$\dot{h} = \dot{p_d} + K\dot{h_e} \tag{73}$$

$$\dot{h_e} = \dot{h_d} - \dot{h} \tag{56}$$

$$\delta_e = \dot{h_d} - \dot{p_d} \tag{72}$$

De estas ecuaciones se puede encontrar la ecuación:

$$\dot{h_d} - \dot{h_e} = \dot{p_d} + Kh_e \tag{77}$$

Despejando $\dot{p_d}$ en (77) se obtiene (78) y luego despejando $\dot{h_d}$ se obtiene (79):

$$\delta_e = \dot{p_d} + Kh_e \tag{78}$$

$$\dot{h_e} = \delta_e - Kh_e \tag{79}$$

Remplazando en la ecuación candidata (63) la ecuación (79):

$$\dot{v} = h_e^T(\delta_e - Kh_e) \tag{80}$$

Aplicando el criterio de estabilidad $\dot{v}(h_e) < 0$ y resolviendo la inecuación se obtienen los valores de K que garantizan estabilidad.

$$|h_e^T K h_e| > |h_e^T \delta_e| \tag{81}$$

$$K > \frac{|\delta_e|}{|h_e|} \tag{82}$$

De la ecuación (82) se identifica que la constante K debe ser definida positiva y mayor al cociente entre el error de velocidades en el punto deseado y el error de trayectoria.

Para el diseño del control de camino se estableció la matriz K como una matriz diagonal con los siguientes valores:

$$K = \begin{bmatrix} 12 & 0 \\ 0 & 12 \end{bmatrix} \tag{83}$$

Se hicieron pruebas de simulación variando el valor de la matriz K para un determinado camino, hasta establecer el valor que mejor respuesta entrega eliminando el error en estado estable y que genere velocidades sin oscilaciones, a continuación, se presentan las gráficas resultantes y el camino aplicado.

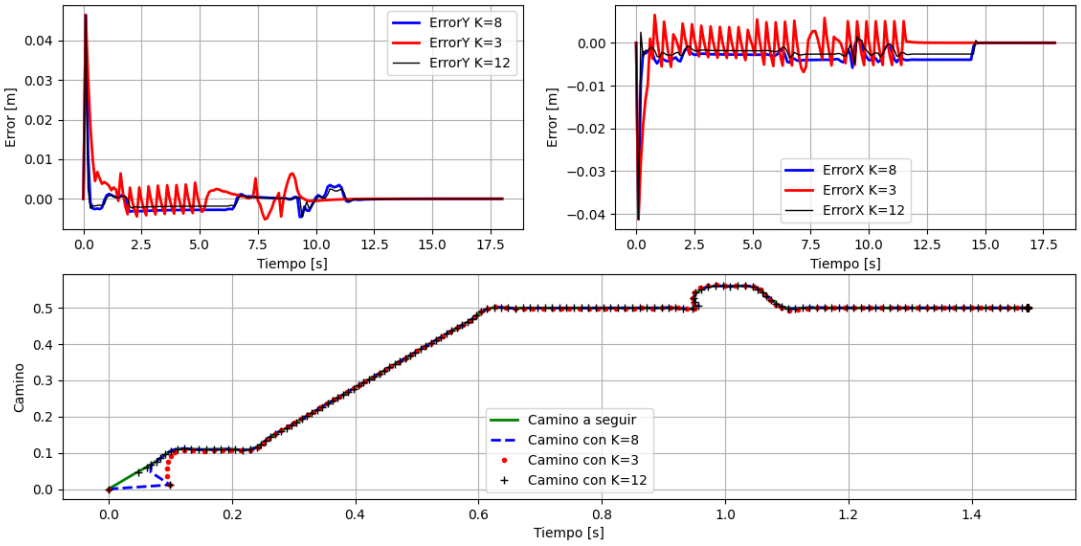


Figura 25. Variación del error y el seguimiento del camino según el cambio de la matriz K.

En la figura 25 se evidencia el comportamiento del error de trayectoria, así como el seguimiento de la misma, se puede observar que con el valor K=12 y K=8 se

presenta menor amplitud y menores oscilaciones en los errores, al aumentar la ganancia K las oscilaciones disminuyen y el robot toma más tiempo en completar la ruta, es decir el tiempo de establecimiento aumenta, con el valor de ganancia K=3 el tiempo que toma el error en X en llegar a cero (0) es 12 segundos, y el error en Y 9 segundos, en cuanto se sube el valor de la ganancia el tiempo de establecimiento aumenta, para los valores K=12 y K=8 los tiempos de establecimiento son aproximadamente el mismo.

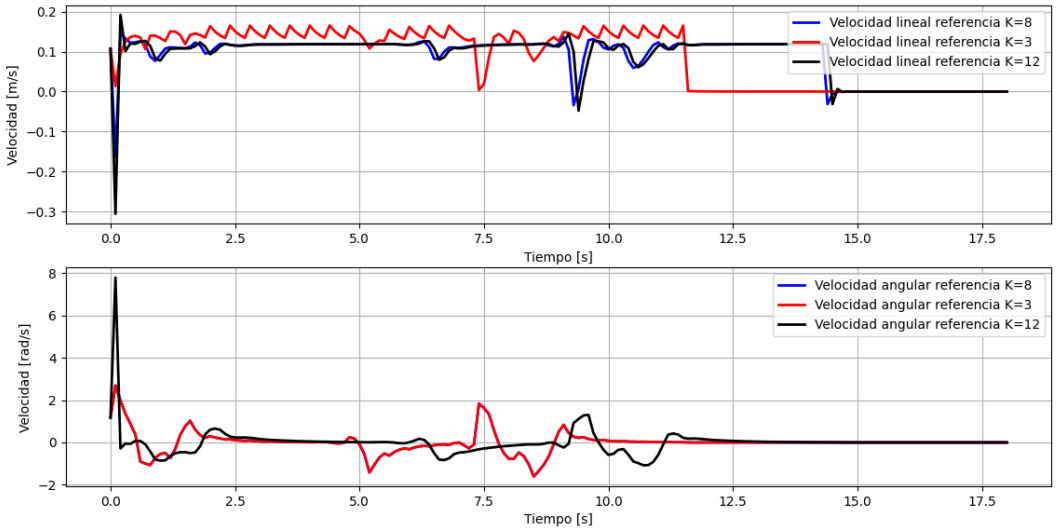


Figura 26. Variación de las velocidades lineal y angular según cambio de la matriz K.

En la figura 26 se observa que las velocidades angulares generadas, con el valor K=12 y K=8 tienen el mismo comportamiento, y la velocidad lineal generada no posee oscilaciones.

Debido a que la respuesta del error generada con K=12 tuvo menor amplitud además de estabilizarse en cero (0), se tomó este valor como parámetro para la matriz K.

4.3.1. Prueba simulada del algoritmo de control de camino.

Para la simulación del algoritmo de control de caminos se realiza la programación en Python del modelo cinemático directo diferencial, se aplica una interpolación de grado 3 a los puntos del camino generado, con la finalidad de obtener una trayectoria suavizada para que la locomoción del robot diferencial sea capaz de realizarla, de igual modo se calculan los errores de trayectoria, se programa el modelo de control y el entorno de simulación como lo muestra el diagrama de flujo de la figura 24.

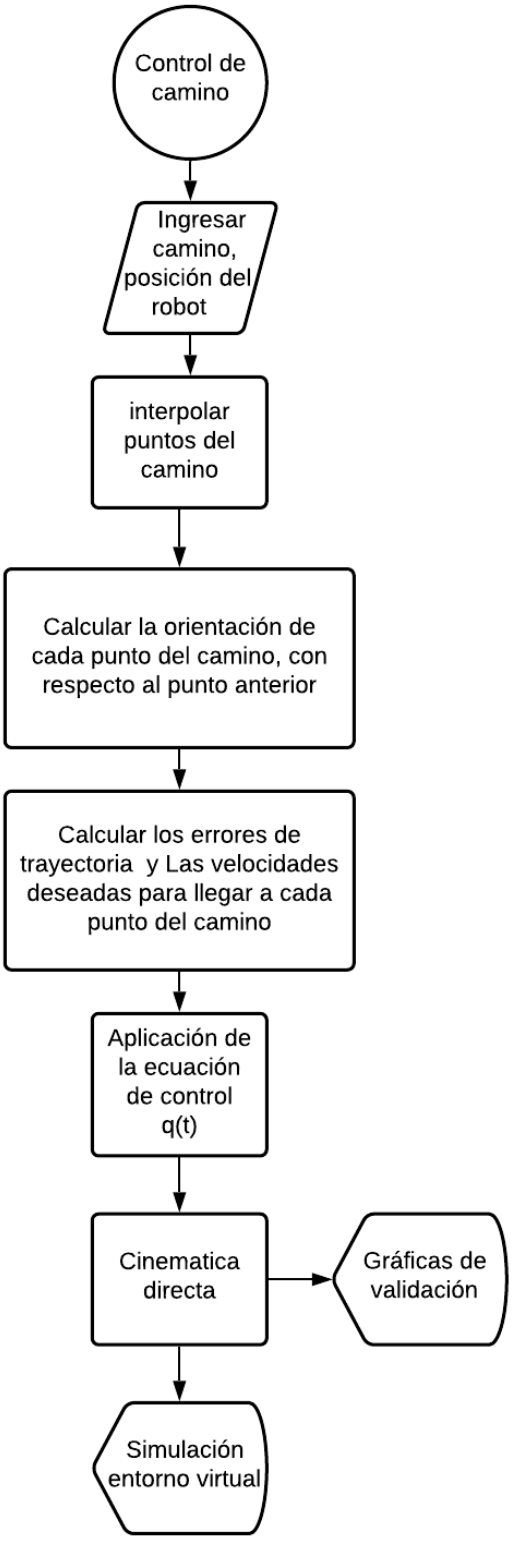


Figura 27. Diagrama de flujo para programación del control de camino.

4.3.2. Aplicación de cada Fase del diagrama de flujo del control de camino.

- **Ingresar las condiciones iniciales del robot y los vectores con el camino.**

Se crean variables en el algoritmo Python donde se almacenarán las condiciones iniciales del robot y el punto objetivo o meta, para el desarrollo de este proyecto se crearon las siguientes variables, en donde N representa el número de puntos para la simulación:

```
x1 = Zeros(H)

y1 = Zeros(H)

phi = Zeros(H)


hx = Zeros(H)

hy = Zeros(H)


x1[0] = 0

y1[0] = 0


hx[0] = 0

hy[0] = 0

phi[0] = 0


pointX = [0.0, 0.019999999999999907, 0.04999999999999993, 0.07999999999999996.....
```

pointY = [0.0, 0.019999999999999907, 0.04999999999999993, 0.07999999999999996.....

- **Interpolar puntos del camino.**

Se debe realizar la interpolación de los puntos del camino, buscando obtener una trayectoria curva sin giros a 90° que la geometría y construcción del robot le permita seguir, para este caso se utilizó la interpolación de grado 3 programándose de la siguiente manera, Se obtienen los nuevos puntos en las variables p_{xd} y p_{yd}:

```
pts = array([pointX,pointY])


tck, u = splprep(pts, u=None, s=0.0, per=0, k = 3 )

u_new = linspace(u.min(), u.max(), div)

pxd, pyd = splev(u_new, tck, der=0)
```

Mediante la función Linspace se genera un vector espaciado linealmente desde un punto u.min a u.max, este vector será la cantidad de puntos que tendrá el nuevo camino luego de la interpolación y con la función splev se genera una representación B-spline de un conjunto de puntos relacionándolos entre sí, este genera las coordenadas X y Y del camino interpolado.

- **Cálculos de la orientación de los puntos del camino.**

Se calculan la orientación de cada punto respecto al punto inmediatamente anterior, esto para que el robot siga la ruta con la orientación correcta

- **Cálculos de los errores de trayectoria y de las velocidades para cada punto del camino.**

Se crean vectores que almacenaran los datos del error de la trayectoria de h_{ex} y h_{ey} , estos se calculan mediante la diferencia entre la posición actual del robot h_yy h_x con la posición del punto del camino p_{yd} y p_{xd} mediante la ecuación:

$$h_{ex} = p_{xd} - h_x \tag{83}$$

$$h_{ey} = p_{yd} - h_y$$

Estos vectores formaran la matriz de errores de trayectoria h_e de la siguiente forma:

$$h_e = [h_{ex}; h_{ey}]$$

```
hex[n] = pxd[pos] - hx[n]

hey[n] = pyd[pos] - hy[n]


he = array([[hex[n]],[hey[n]]])
```

Para calcular la velocidad con la que debe pasar por cada punto se utiliza la ecuación (74), (75) y (76), que relacionan un valor de velocidad máxima definido como 0.2 [m/s], y el ángulo β calculado en el paso anterior, para esto se programó la siguiente línea:

```
# Velocidad deseada

pxdp = vMax*cos(beta[pos]);

pydp = vMax*sin(beta[pos]);
```

En el cual el vector pdp almacena las dos componentes del vector de velocidad calculado.

- **Aplicación de la ecuación de control de camino**

Se programa la ecuación (71) aplicando los errores calculados y la matriz jacobiano, remplazando a K con la determinada en la ecuación (74), como se muestra:

$$q(\dot{t}) = \begin{bmatrix} \cos(\phi) & -a\sin(\phi) \\ \sin(\phi) & a\cos(\phi) \end{bmatrix}^{-1} * (pdp + \begin{bmatrix} 12 & 0 \\ 0 & 12 \end{bmatrix} * h_e)$$

De donde se sabe que las velocidades lineales y angulares estarán dadas por:

$$q(\dot{t}) = [\begin{matrix} \mu \\ \omega \end{matrix}]$$

Para el desarrollo de este proyecto se programaron dichas ecuaciones en Python de la siguiente forma:

```
# Parametros de control
```

```
K = array([[ 12, 0],
           [ 0,  12]])

# Velocidad
pxdp = vMax*cos(beta[pos]);
pydp = vMax*sin(beta[pos]);


qpRef = linalg.pinv(J)@(pdp+K@he)
```

- **Cinemática directa.**

Para la programación de la cinemática directa se debe calcular la posición del robot luego de aplicarle los vectores de velocidad $q(\dot{t})$ calculados en el paso anterior para esto se usará inicialmente el método de integración de Euler hacia adelante para hallar la coordenada $p = (x, y)$ como se muestra a continuación:

$$x_1[n] = \mu * t_s + x_1[n-1]$$

$$y_1[n] = \mu * t_s + y_1[n-1]$$

$$\phi[n] = \omega * t_s + \phi[n-1]$$

Donde μ es la velocidad lineal calculada en el paso anterior y t_s es el periodo de muestreo 0.1 segundos, y_1 y x_1 representan la posición del origen del robot, φ representa la orientación del robot. Para encontrar las coordenadas del punto de interés $h = (x, y)$ se programarán las ecuaciones (28) y (29):

$$h_x = x_1 + \mathrm{a}\cos(\phi)$$

$$h_y = y_1 + \mathrm{a}\sin(\phi)$$

En donde h_y y h_x representan la posición del punto de interés siendo en este caso la herramienta de detección de metal y a es la longitud desde el origen de coordenadas del robot hasta la ubicación del punto de interés.

Para el desarrollo de este proyecto se programó como se muestra a continuación:

```
x1p = uRef[n]*cos(phi[n+1])

    y1p = uRef[n]*sin(phi[n+1])


    x1[n+1] = x1[n] + ts*x1p

    y1[n+1] = y1[n] + ts*y1p


    hx[n+1] = x1[n+1]+a*cos(phi[n+1])

    hy[n+1] = y1[n+1]+a*sin(phi[n+1])
```

- **Entornos de simulación**

Para la elaboración del entorno de simulación virtual, se toma el diseño CAD del robot en formato STL y en conjunto con las funcionalidades que el módulo PyVista de Python ofrece se realiza la simulación 3D mostrando gráficamente la trayectoria y el movimiento del robot.

```
ArchivoSTL = "ACTUALSTL"

color                                                                    =
['white','white','white','green','red','gray','gray','black','black','white','white','white',
'white','white','black','white']


Robot = robotics(ArchivoSTL,color)


xmin = -2

xmax = 2

ymin = -2

ymax = 3
```

```
zmin = 0

zmax = 2

Limites_ejes = [xmin, xmax, ymin, ymax, zmin, zmax]

uniciclo.configureScene Limites_ejes)


uniciclo.initTrajectory(hx,hy)

uniciclo.initRobot(x1,y1,phi,escala)

uniciclo.startSimulation(1,ts)
```

4.3.3. Simulación del controlador con diferentes caminos

A continuación, se generarán de manera aleatoria caminos en 2 dimensiones y serán aplicados al algoritmo de control de camino para validar su funcionamiento simulado.

Camino 1.

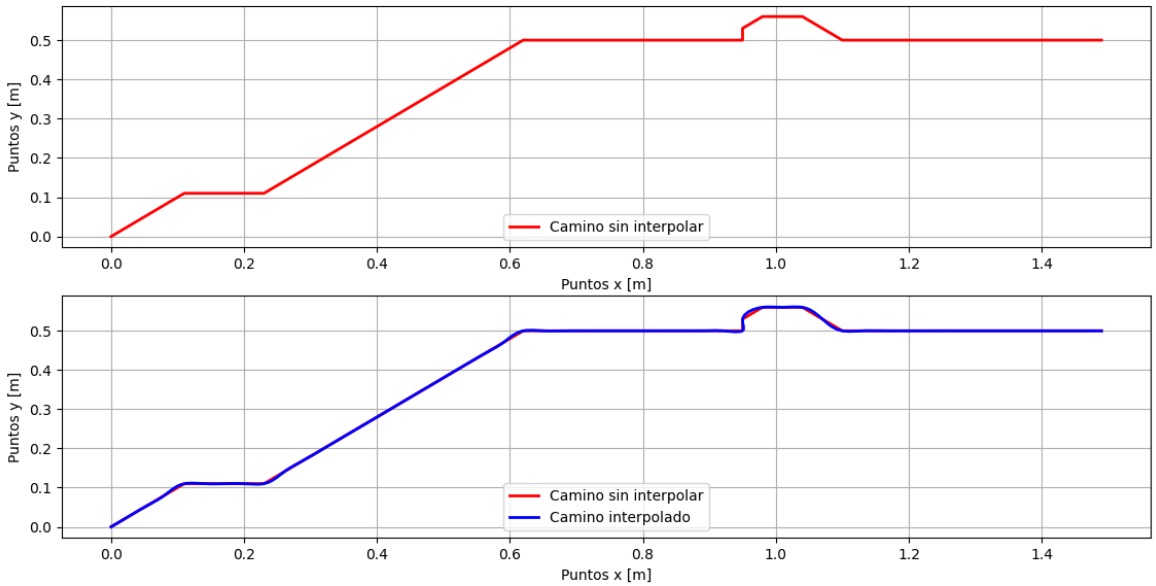


Figura 28. Camino aplicado para prueba de simulación

En la figura 28 se observa en la gráfica superior el camino sin interpolar que será aplicado para la simulación, se observa como al cambiar de dirección se generan ángulos que la construcción diferencial del robot no permite seguir, en la gráfica inferior se presenta el camino interpolado en el cual se evidencia el seguimiento al camino original además de redondear las esquinas en los cambios de dirección, este es el camino que se aplicará como entrada al algoritmo de control de camino para verificar el correcto seguimiento de este.

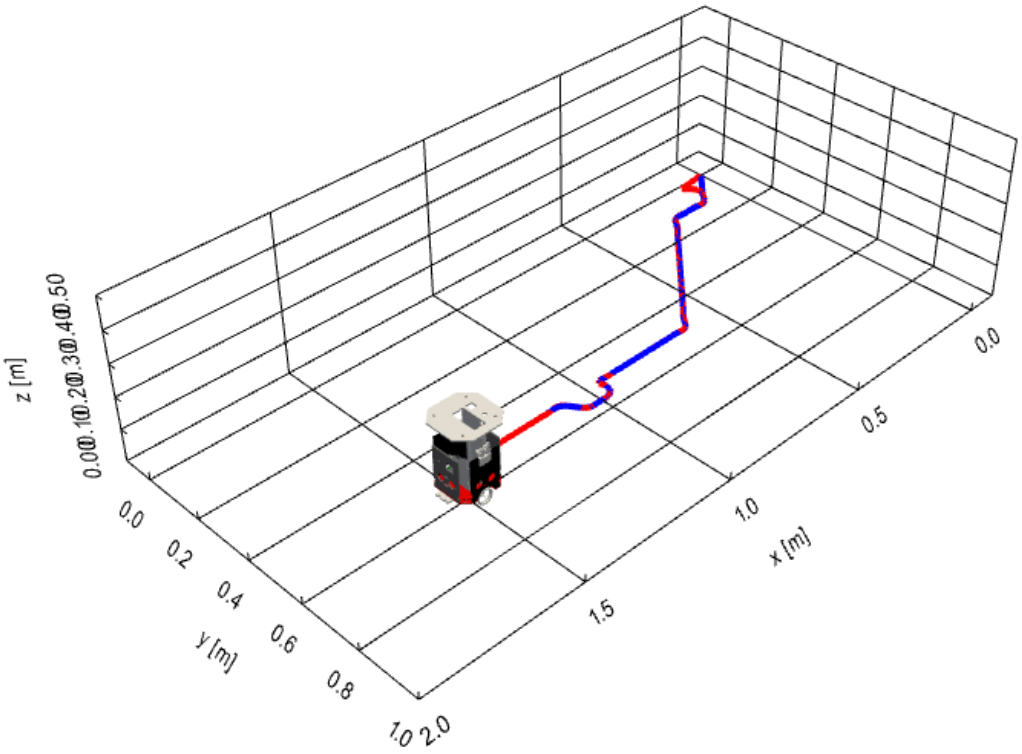


Figura 29. Simulación en entorno 3D del movimiento de trayectoria

En la figura 29 se observa de color azul el camino aplicado, y en color rojo el camino seguido por el robot, se puede determinar un correcto seguimiento a la trayectoria.

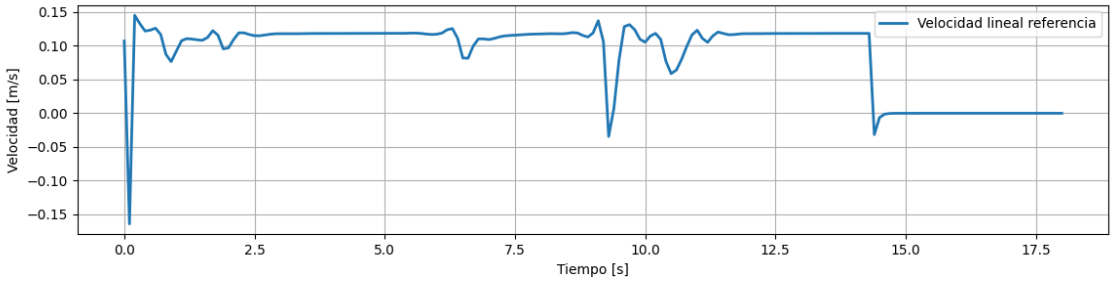


Figura 30. Velocidad lineal calculada

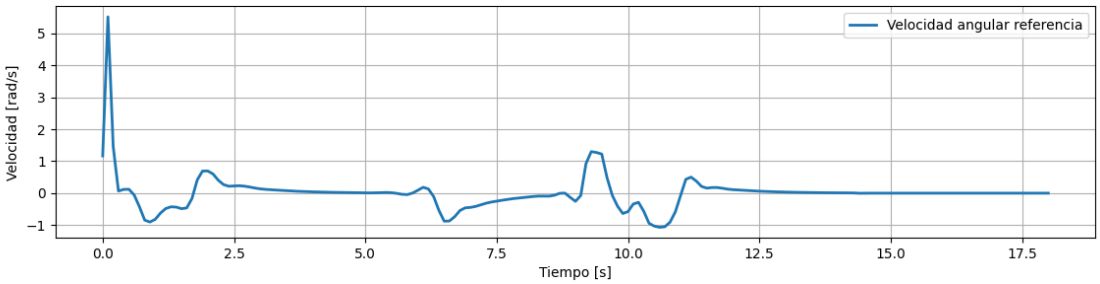


Figura 31. Velocidad angular calculada.

En la figura 30 y figura 31 se observa la velocidad lineal en [m/s] y la velocidad angular en [rad/s] a la cual debe moverse el robot para lograr seguir el camino dichas graficas fueron calculadas con la ecuación (71) aplicando la matriz K determinada en la ecuación (83).

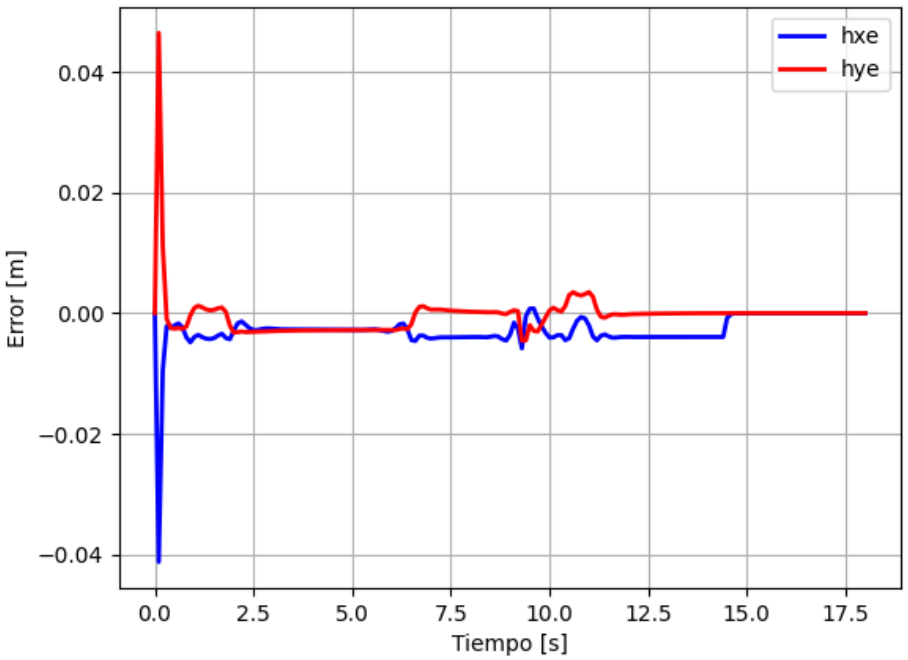


Figura 32. Errores resultantes del algoritmo de caminos.

De la figura 32 se observa el comportamiento del error de trayectoria, se presentan dos graficas para el error que son el error en X y el error en Y, estas se calculan con la ecuación (14) que relaciona la diferencia entre la posición del robot y el punto del camino.

Hay que notar que el error se mantiene presente, puesto que al ser un set-point de camino este corrige el error al llegar a un punto de la ruta, sin embargo, de inmediato tendrá que seguir a otro punto de esta, teniendo así un nuevo valor de error de trayectoria, llegando a cero (0) solo en el último punto de la ruta.

Camino 2.

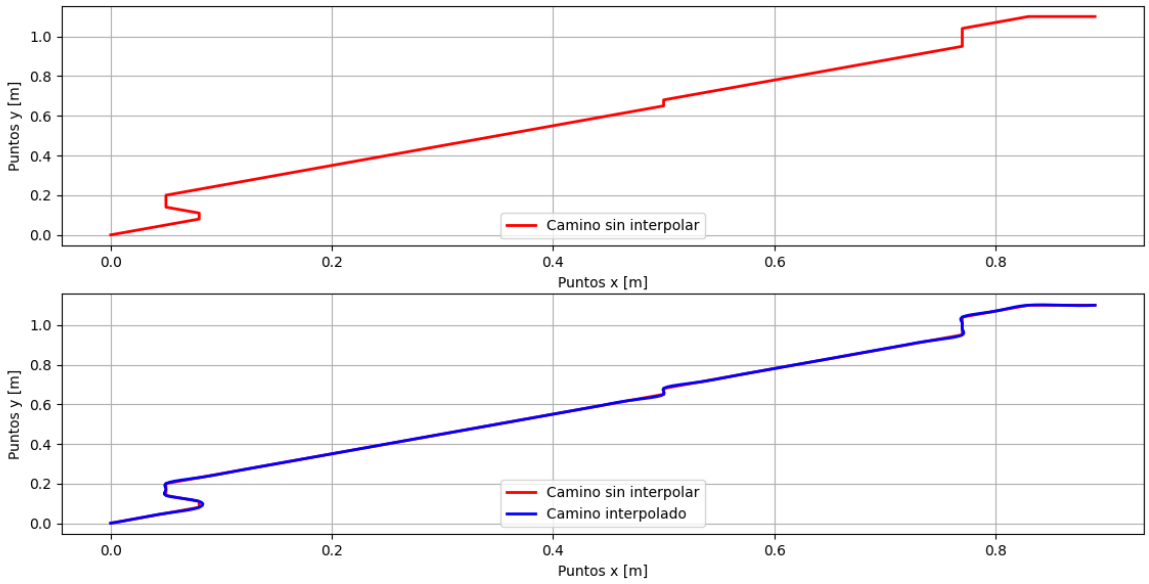


Figura 33. Camino para probar en simulación

Se establece el camino mostrado en la figura 33 como el Set-Point a aplicar para la segunda prueba de simulación

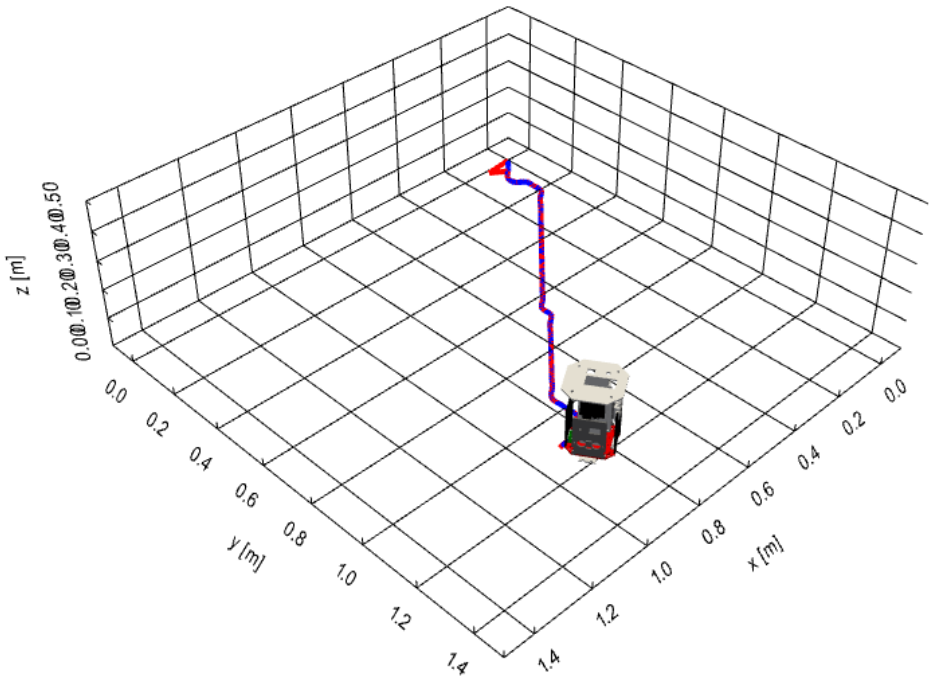


Figura 34. Simulación en entorno 3D.

Se observa en la figura 34 un correcto seguimiento del camino aplicado.

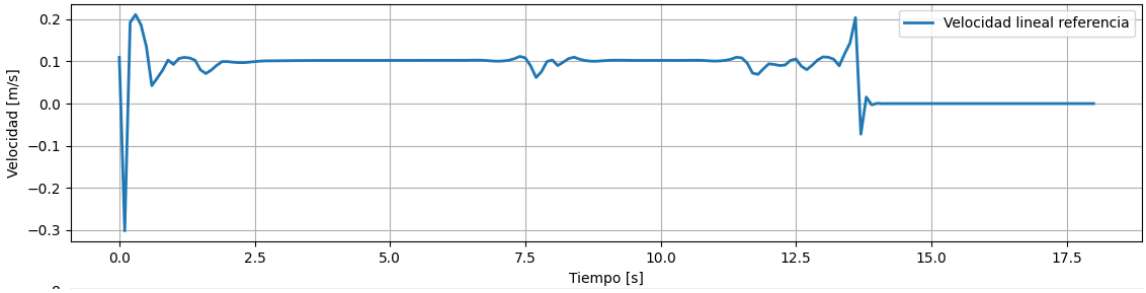


Figura 35. Velocidad lineal calculada

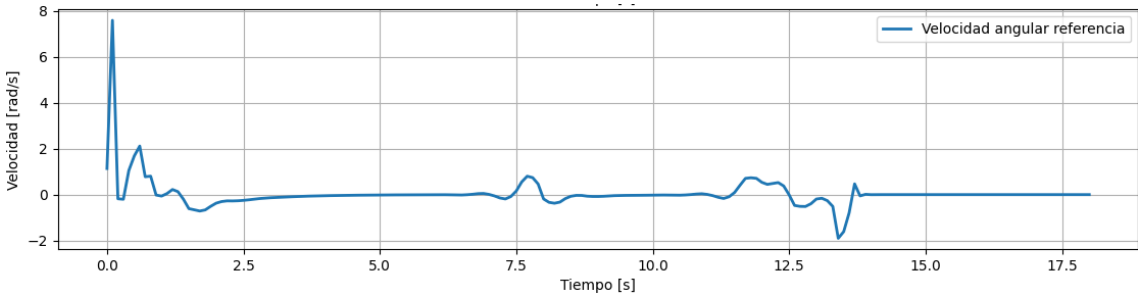


Figura 36. Velocidad angular calculada.

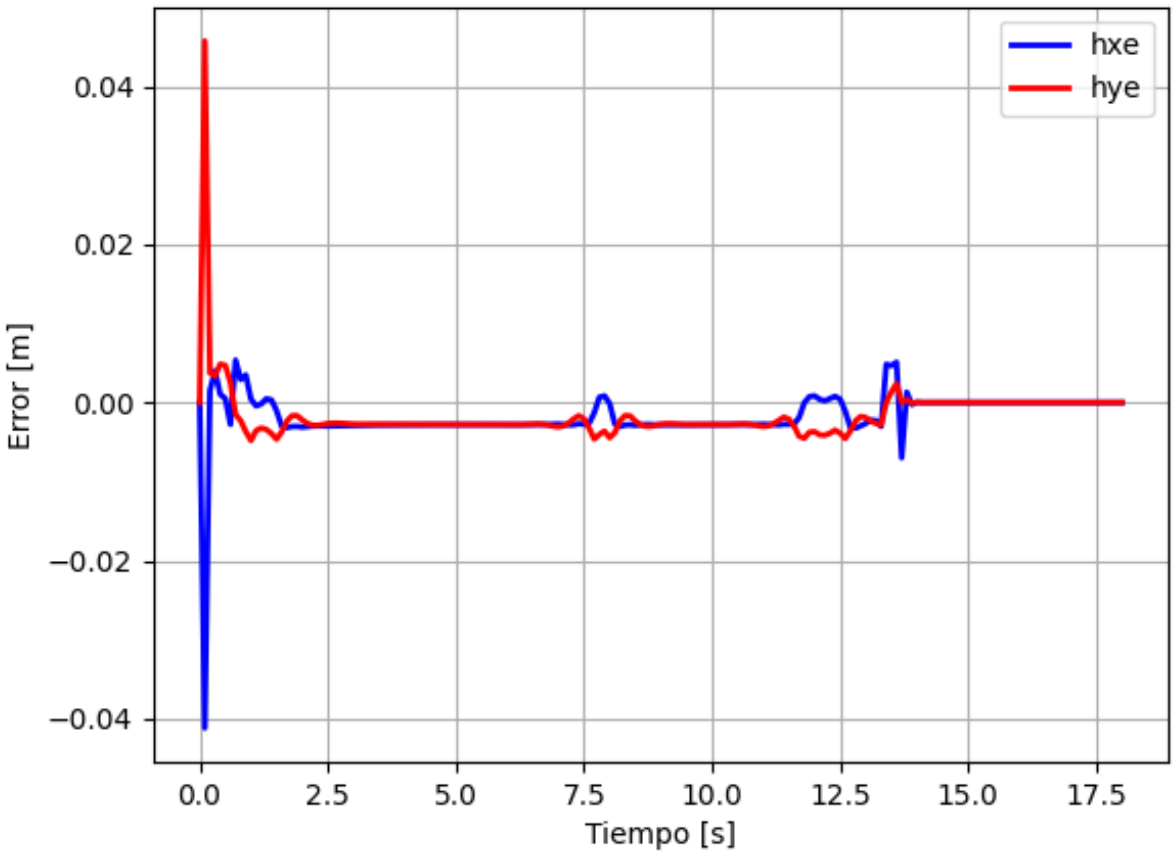


Figura 37. Errores calculados

En las figuras anteriores 35,36,37 se presentan la velocidad lineal, angular y los errores en X y Y para el camino aplicado.

Se puede observar que el error en estado estable es cero (0), lo cual indica que el robot luego de realizar el seguimiento a la trayectoria este se detiene en el punto objetivo.

Camino 3.

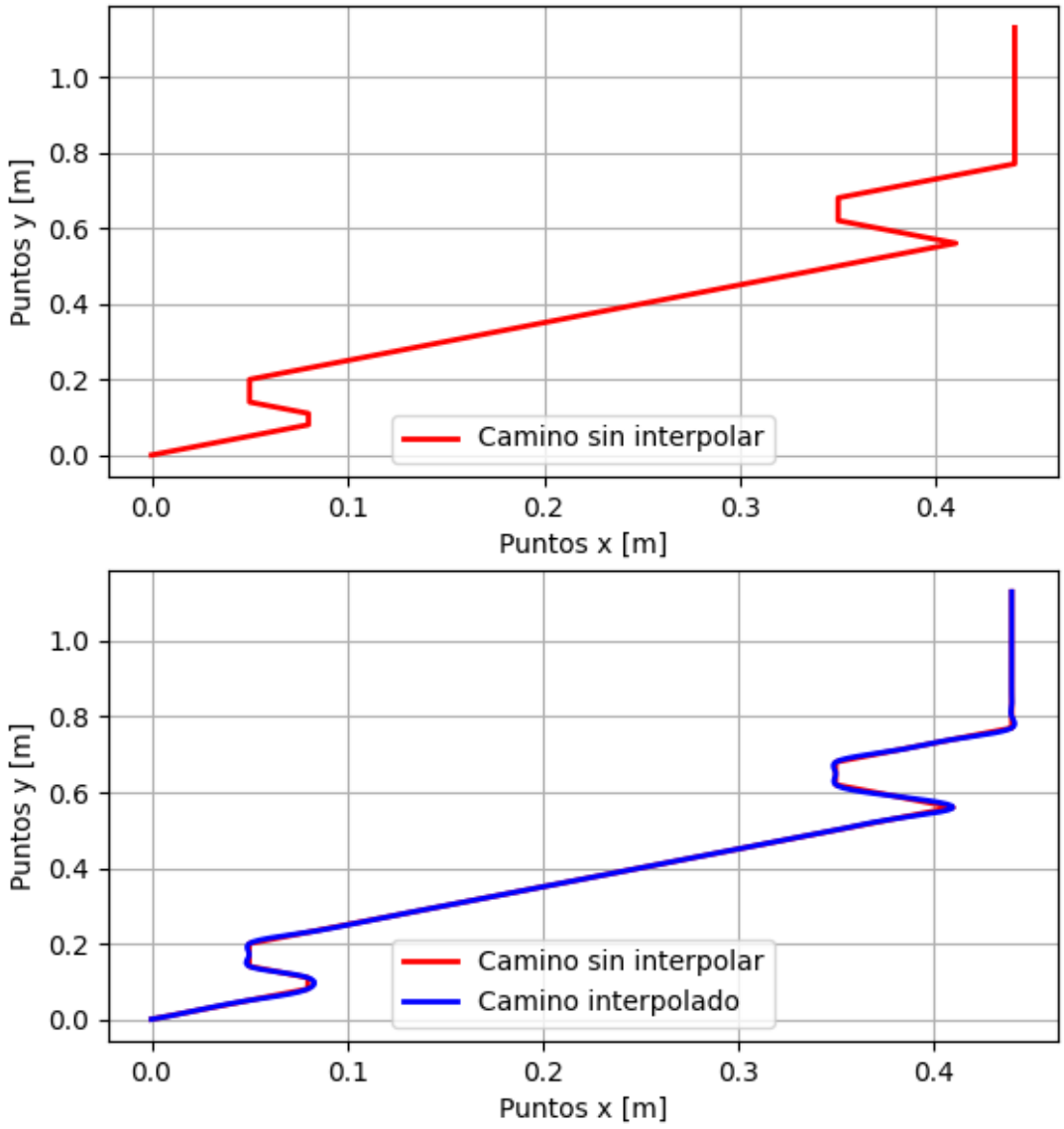


Figura 38. Camino aplicado para simulación.

En la figura 38 se presenta el tercer camino con su equivalente interpolado, para probar mediante simulación el algoritmo de seguimiento de camino programado.

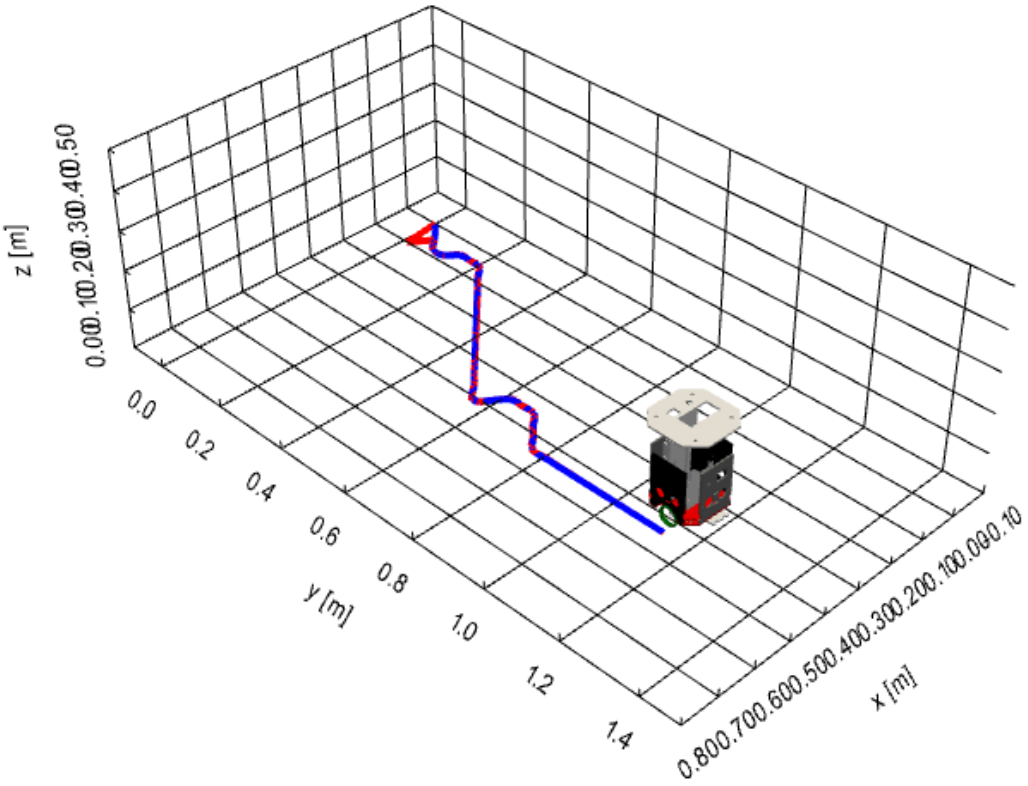


Figura 39. Entorno de simulación 3D.

En la figura 39 se aprecia que el robot se desplaza por el camino confirmando así el funcionamiento del controlador de camino.

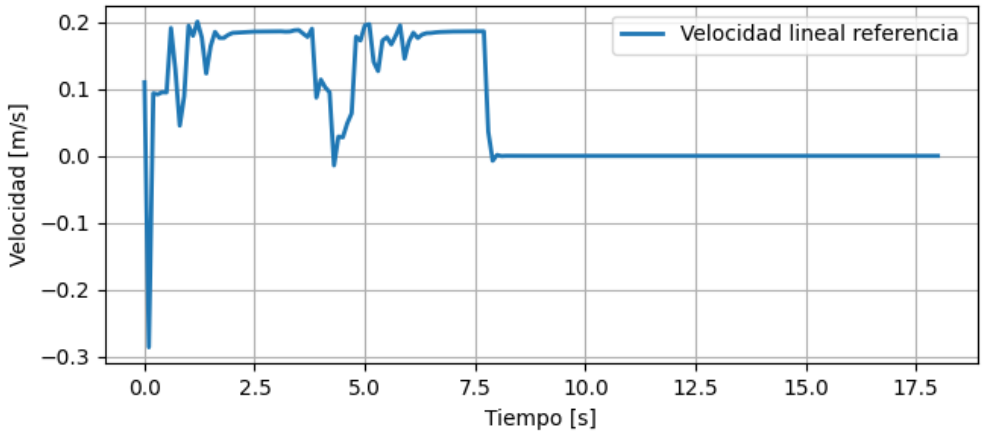
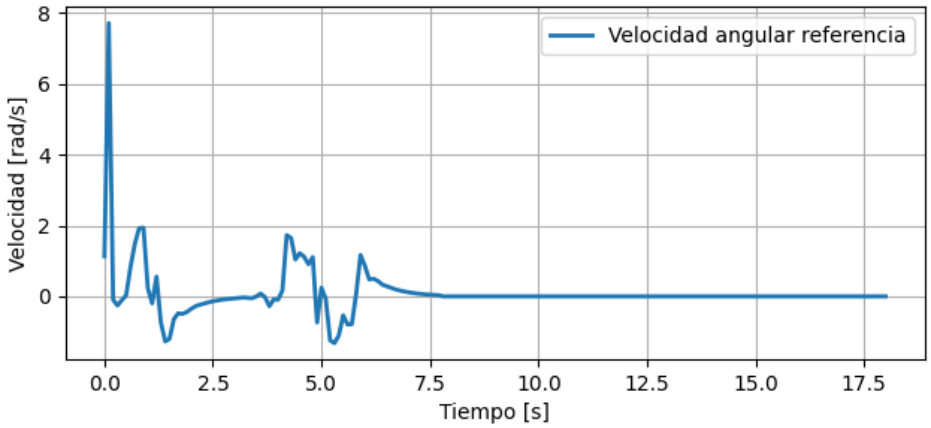


Figura 40. Velocidad lineal calculada.



Errores [rad/s]

Figura 41. Velocidad angular calculada.

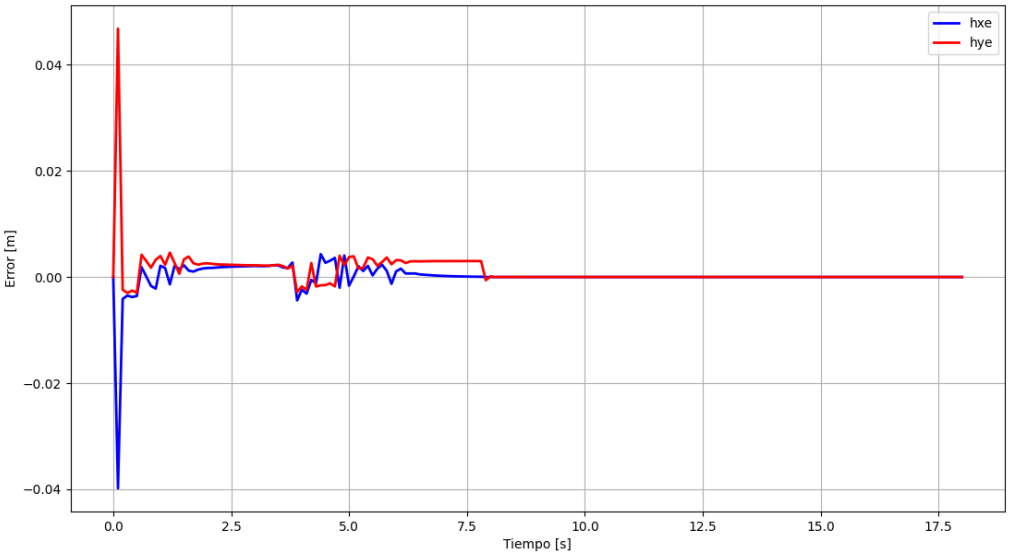


Figura 42. Errores de trayectoria calculados.

Se puede observar de igual forma en la figura 42 como los errores de trayectoria X y Y convergen a cero (0) en estado estable.

5. Diseño de los algoritmos de control de alto nivel.

Ya se ha descrito los algoritmos que generan las velocidades para seguimiento de camino y control de posición, a continuación, se describen los algoritmos de alto nivel que generan los caminos libres de obstáculos hasta la meta. Para la ejecución de estos algoritmos se hace necesario conocer el mapa en el cual se desplaza el robot, esto incluye la ubicación de los obstáculos para la posterior modelización mediante el algoritmo planificador.

5.1. Algoritmo basado en campos potenciales

Campos potenciales es el nombre que recibe un método de generación y planificación de trayectorias para navegación autónoma de robots móviles, este método calcula campos imaginarios de atracción o repulsión que emanan de los obstáculos, dichos campos pueden variar según la geometría o distancia de los obstáculos, este proceso consiste en encontrar un camino que se encuentre tan alejado de los obstáculos como sea posible que lleve al robot desde un punto inicial hasta otro final [17].

Como ya se definió, el algoritmo de campos potenciales modela campos repulsivos generados por obstáculos siendo más fuertes cerca a estos, y más débiles o nulos alejados de los obstáculos, del mismo modo el punto objetivo o meta genera un campo potencial atractivo, es así como el algoritmo determina la ruta a seguir libre de obstáculos hacia la meta.

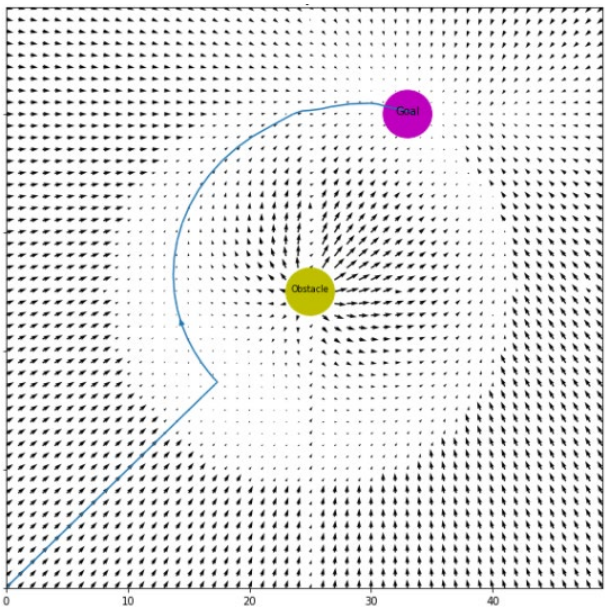


Figura 43. Ejemplo de funcionamiento del algoritmo de campos potenciales [19]

En la figura 43 se observa un ejemplo de algoritmo de campos potenciales, una partícula amarilla que simula un obstáculo, genera flechas en dirección opuesta al objeto, bajo la interpretación del algoritmo, estas flechas son equivalentes al campo repulsivo, y la meta genera flechas hacia su dirección, lo que interpreta como campos atractivos, de esta forma la partícula se desplaza a través del mallado de flechas siguiendo la dirección indicada por cada una hasta lograr alcanzar la meta, generando así un camino libre de obstáculos

La idea de un campo potencial se toma de la naturaleza. Por ejemplo, una partícula cargada que navega por un campo magnético o una pequeña bola rodando en una colina. La idea es que dependiendo de la fuerza del campo o de la pendiente de la colina, la partícula o la bola pueden llegar a la fuente del campo, al imán o al valle en este ejemplo.

En robótica, se puede simular el mismo efecto creando un campo de potencial artificial que atraerá al robot hacia la meta. [18].

- **Ecuaciones para el algoritmo de campos potenciales**

El algoritmo de campos potenciales calcula los valores de fuerza inducida cada vez que el robot avanza, como ya se especificó antes los obstáculos se modelan como una partícula que tienen una posición y un tamaño especifico, estos generan campos repulsivos son representados mediante la ecuación [19]:

$$U_o = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{d}-\frac{1}{q}\right)^2, & d \leq q \\ 0, & d > q \end{cases} \tag{84}$$

En la cual el potencial repulsivo es representado mediante U_o, η representa la ganancia repulsiva, d representa la distancia a la meta y q es la distancia al objeto.

La meta que es el punto objetivo genera un campo potencial atractivo, que impulsa al robot a moverse sobre este, el campo atractivo se representa mediante la ecuación [19]:

$$U_g = \frac{1}{2}\gamma d(q, q_{goal}) \tag{85}$$

En la cual el potencial atractivo es representado mediante U_g, γ representa la ganancia atractiva, d representa la distancia a la meta en función de q_{goal} la distancia a la meta y q el valor potencial de la casilla en la cual se encuentra el robot dentro de la matriz previamente modelada para el algoritmo.

Finalmente se calcula la fuerza inducida en cada paso del robot sumando las fuerzas repulsivas y las fuerzas atractivas esto aplica para el resto del escenario que no está ocupado por la meta o algún obstáculo [19]:

$$U_f = U_g + U_o \tag{86}$$

Se debe realizar un mapeo del área (dentro del entorno virtual) donde se aplicará algoritmo para calcular la fuerza potencial de cada zona del mapa.

- **Programación del algoritmo de campos potenciales.**

El algoritmo de campos potenciales será implementado en el lenguaje Python, para esto es necesario representar el robot, los obstáculos y el ambiente para simulación, se usarán las ecuaciones de las fuerzas atractivas (85) las fuerzas repulsivas (84) y la fuerza total calculada en cada punto del mapa (86), También se debe considerar que cada paso que el robot avance se evaluarán los valores de la fuerza potencial en la posición del robot y este se mueve gracias a la fuerza inducida por el campo potencial, de esta forma el algoritmo puede determinar el punto siguiente de la trayectoria.

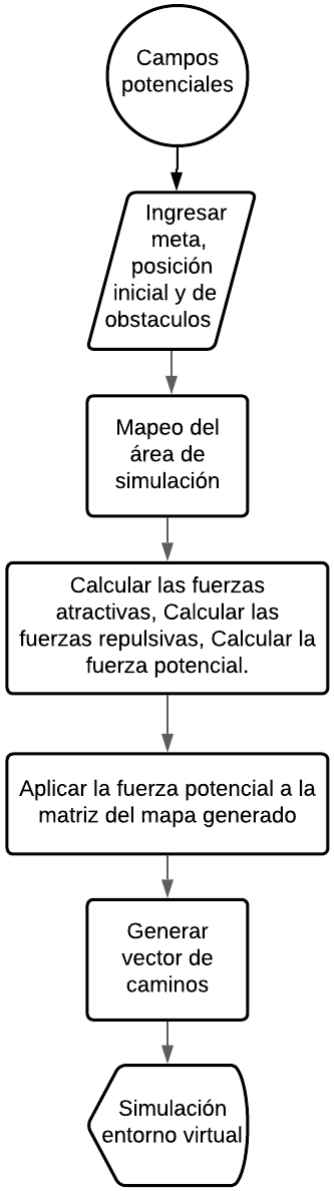


Figura 44. Diagrama de flujo para programación del algoritmo campos potenciales.

5.1.1. Aplicación de cada Fase del diagrama de flujo para la programación del algoritmo de campos potenciales.

- **Ingresar meta, posición inicial y de obstáculos.**

Se crean variables en el algoritmo Python donde se almacenarán las condiciones iniciales del escenario tales como la posición inicial del robot, la coordenada del punto objetivo, y los vectores de los obstáculos.

- **Mapeo del área de simulación.**

El algoritmo modela el entorno como una matriz en la cual cada cuadricula es del tamaño especificado al inicio del programa, posteriormente realiza un mapeo para establecer la ubicación de la meta y los objetos sobre dicho espacio matricial y calcular el campo potencial correspondiente para cada casilla, se almacena en una variable *pmap* es una matriz que contiene todos los puntos con su respectivo potencial.

- **Cálculos de las fuerzas atractivas, repulsivas y potencial.**

Se calculan mediante las ecuaciones (84) y (85) para el potencial atractivo se utiliza la ecuación (85) calculando la distancia euclidiana hasta la meta multiplicada por la ganancia atractiva, esta ganancia determina que tan amplio es el rango de atracción de la meta, un valor muy grande genera un área amplia de atracción, sin embargo, al tener obstáculos cerca se puede encontrar el robot con un mínimo local, para este caso esta ganancia tiene un valor de 5.

El potencial repulsivo se calcula mediante la ecuación (84) calculando la distancia euclidiana entre la ubicación actual del robot con cada obstáculo, es decir primero se encuentra al obstáculo más cercano a la posición del robot y se almacena en la variable “minid”, luego se encuentra al obstáculo más alejado y se almacena en la variable “maxd” con esto se mapea en el área de simulación las posiciones de la matriz que serán fuerzas repulsivas y de acuerdo a las dimensiones de los obstáculos se determina la distancia hasta donde el potencial repulsivo tendrá efecto partiendo desde el obstáculo.

Para el cálculo final del potencial se usa la ecuación (86) tal como se explicó en el paso anterior.

```
def calc_atractivo_potencial(x, y, mx, my):

    return 0.5 * kp *hypot(x - mx, y - my)

def calc_repulsivo_potencial(x, y, ox, oy, rr):

    # buscando el obstaculo más cercano

    min= -1
```

```
    d_min = float("inf")
    for i, _ in enumerate(ox):
        aux = hypot(x - ox[i], y - oy[i])
        if d_min >= aux:
            d_min = aux
            min = i
    # potencial repulsivo
    auxq = hypot(x - ox[min], y - oy[min])
    if auxq <= rr:
        if auxq <= 0.1:
            auxq = 0.1
        return 0.5 * eta * (1.0 / auxq - 1.0 / rr) ** 2
    else:
        return 0.0
```

- **Aplicar la fuerza potencial a la matriz del mapa generado.**

Se toma la función programada en pasos previos para calcular la fuerza potencial de cada casilla de la matriz, para establecer el alcance del potencial repulsivo de cada obstáculo se toma la resolución o tamaño de la cuadricula y las condiciones mínimas y máximas calculadas mediane la función programa en paso previo, el cual para este caso tiene el nombre “cal_potencial_campo” para esto se programaron las siguientes líneas de código:

```
def potencial_campo_planificador(sx, sy, mx, my, ox, oy, reso, rr):
    # calculo del campo potencial
    pmap, minx, miny = calc_potencial_campo(mx, my, ox, oy, reso, rr, sx, sy)
```

```
#Busca el camino
daux = hypot(sx - mx, sy - my)
ix = round((sx - minx) / reso)
iy = round((sy - miny) / reso)
mix = round((mx - minx) / reso)
miy = round((my - miny) / reso)
```

Ix e iy almacenan la posición inicial del robot, mix y miy la posición de la meta, daux es la distancia desde el robot hasta la meta.

- **Generar vector de caminos.**

Mediante cilcos while, for, if anidados, se establece el potencial para cada punto con coordenadas X, Y y se agrupan en dos vectores los cuales contienen la ruta X, Y que lleva hasta la meta.

- **Simulación.**

Para la simulación basta con graficar los puntos encontrados y aplicarle una pequeña pausa de tiempo al algoritmo plotter.

5.1.2. Pruebas de simulación del algoritmo de campos potenciales.

Las pruebas de simulación se realizarán para diferentes puntos meta y variando la ubicación de los obstáculos para analizar la variación del camino calculado.

Prueba 1:

Mx= 1.05 m, My = 0.44 m, obx = [40,80,21] cm, oby = [40,47,34] cm.

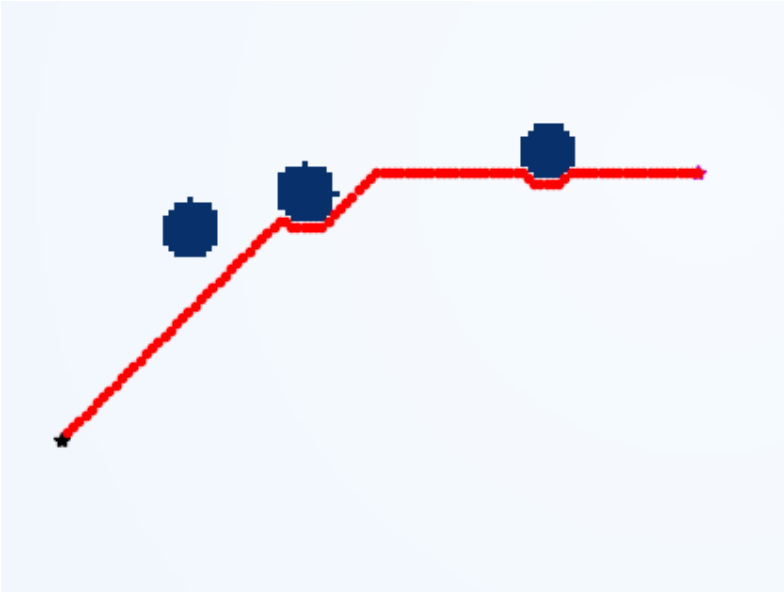


Figura 45. Simulación del algoritmo de campos potenciales.

En la figura 45 se observa la simulación del algoritmo de campos potenciales, en donde el camino se ve representado en color rojo, color azul solido representa el campo repulsivo emanado de los obstáculos, y las estrellas indican el punto de inicio y meta.

Prueba 2:

Mx= 0.94 m, My = 0.66 m, obx = [40,80,21] cm, oby = [40,47,34] cm

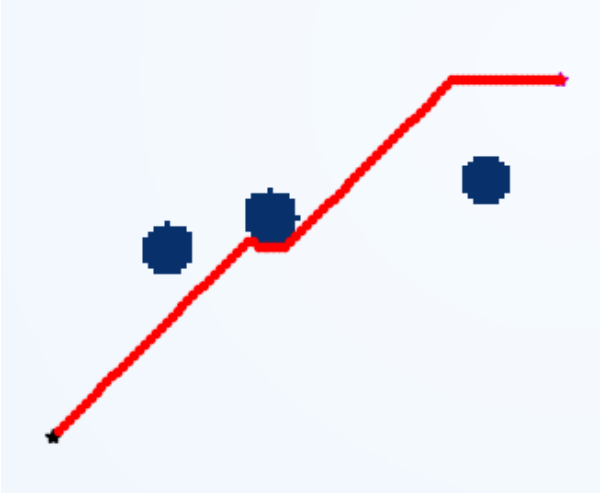


Figura 46. Prueba 2 de simulación del algoritmo de campos.

En la figura 46, se observa el camino generado por el algoritmo cambiando la posición de meta y manteniendo la posición de los obstáculos.

Prueba 3:

Mx= -0.94 m, My = 0.66 m, obx = [-40,-80,-21] cm, oby = [30,63,34] cm

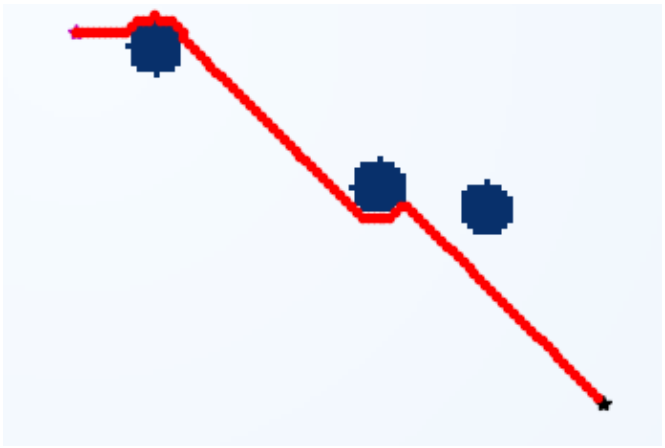


Figura 47. Prueba 3 de simulación del algoritmo de campos.

En la figura 47, se observa la meta con coordenadas negativas y dos obstáculos hacia esta, se logra observar como el algoritmo es capaz de esquivar cada obstáculo.

Prueba 4:

Mx= -0.94 m, My = 0.66 m, obx = [-40,-80,-21,-65] cm, oby = [30,63,34,56] cm

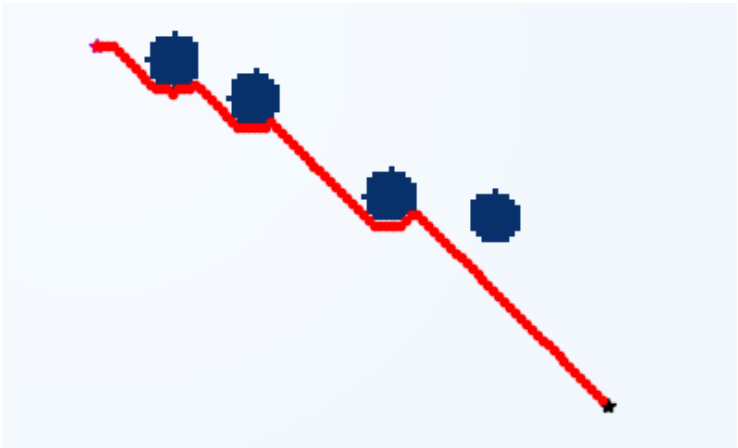


Figura 48. Prueba 4 de simulación del algoritmo de campos.

Para la prueba 4 se usaron las mismas coordenadas que la prueba 3 y se agregó otro objeto que atraviesa el camino inicial, en la figura 62 se observa como el algoritmo evade el nuevo obstáculo y llega hasta la meta.

5.2. Algoritmo basado en grafos de visibilidad: Grafos de visibilidad.

Los grafos permiten relacionar los objetos contenidos en un conjunto. Formalmente, un grafo G es un conjunto de V (vértices) y de A (aristas) tomado de la colección de subconjuntos de dos elementos de V. Una arista de G es un subconjunto como $\{a, b\}$, con $a, b \in V$, $a \neq b$. De forma general un grafo es una colección de vértices que se unen por aristas, los puntos en el plano son los vértices y las aristas son líneas que unen estos puntos [11]

- **Grafos: Algoritmos de búsqueda.**

Dentro de la teoría de grafos existen algoritmos importantes con los cuales se pueden resolver diferentes problemas como la síntesis de circuitos secuenciales, trayectos como líneas de autobuses etc. [20] para este proyecto se utilizará el algoritmo de búsqueda conocido como A*.

- **Algoritmo A***

Como todo algoritmo de búsqueda en amplitud, A* es un algoritmo completo: en caso de existir una solución, siempre dará con ella, el algoritmo A* utiliza una función de evaluación [20]:

$$\mathrm{f}(n) = \mathrm{g}(\mathrm{n}) + \mathrm{h}'(\mathrm{n}) \qquad (27)$$

Donde h'(n) representa el valor heurístico del nodo a evaluar desde el actual n, hasta el final, g(n) es el coste real del camino recorrido para llegar a dicho nodo.

Para garantizar la admisibilidad del algoritmo, la función h(n) debe ser heurística admisible, esto es, que no sobrestime el coste real de alcanzar el nodo objetivo, es decir, h(n) debe ser menor que h*(n) para todo nodo no final [20].

5.2.1. Programación del algoritmo de A*.

Para la programación y el desarrollo de este algoritmo se utiliza la ecuación (87), determinada en el apartado anterior, además se hace necesario conocer el entorno, la posición del robot, la posición de los obstáculos y la posición de la meta, con todo esto presente se hará uso de la lógica del algoritmo A* basada en grafos de visibilidad.

Se modelará una matriz que almacena las posiciones x e y del entorno, aquí ubicaremos la posición inicial, la ubicación de los obstáculos y la meta.

Teniendo en cuenta las limitaciones de movimiento de la locomoción diferencial se parte al análisis del algoritmo, se usarán las ecuaciones (87)

Este algoritmo navega a través de la matriz del entorno verificando que casillas están libres de obstáculos hasta llegar a la casilla objetivo entonces se calculan las casillas con menor coste que conduzcan hasta la casilla objetivo.

Para esto se encuentran el conjunto de nodos ‘f’ con menor coste donde ‘g’ son los posibles nodos por donde puede pasar la ruta, y ‘h’ es la heurística que calcula el coste de los nodos definiendo la ruta que tenga menor coste.

A continuación, se presenta el resultado de la implementación del algoritmo con los siguientes puntos de inicio y meta. Como ya se describió al ser este un algoritmo de búsqueda, su criterio de planificación se expande hacia todas las direcciones hasta encontrar la meta.

Prueba 1:

Mx= 1.9 m, My = 1.8 m, obx = [0.77, 0.2, 1.18, 1.5, 2, 1.5, 1.6] m, oby = [1.0, 0.2, 1, 1.4, 2, 1.2, 1.5] m

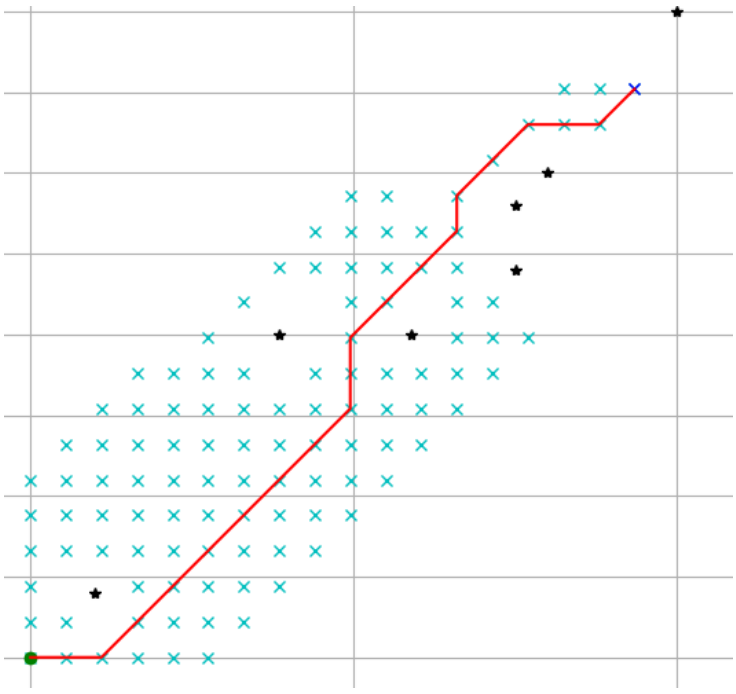


Figura 49. Simulación del algoritmo A*

En la figura 49 se observa el funcionamiento del algoritmo A*, el cual explora hacia todas las casillas libres hasta encontrar la casilla objetivo, y al alcanzarla se identifica mediante la heurística la ruta a seguir.

Prueba 2:

Mx= 1.61 m, My = 1.25 m, obx = [0.77, 0.2, 1.18, 1.2, 0.7, 1.5, 1.6] m, oby = [1.0, 0.2, 1, 0.6, 0.5, 1.2, 1.5] m

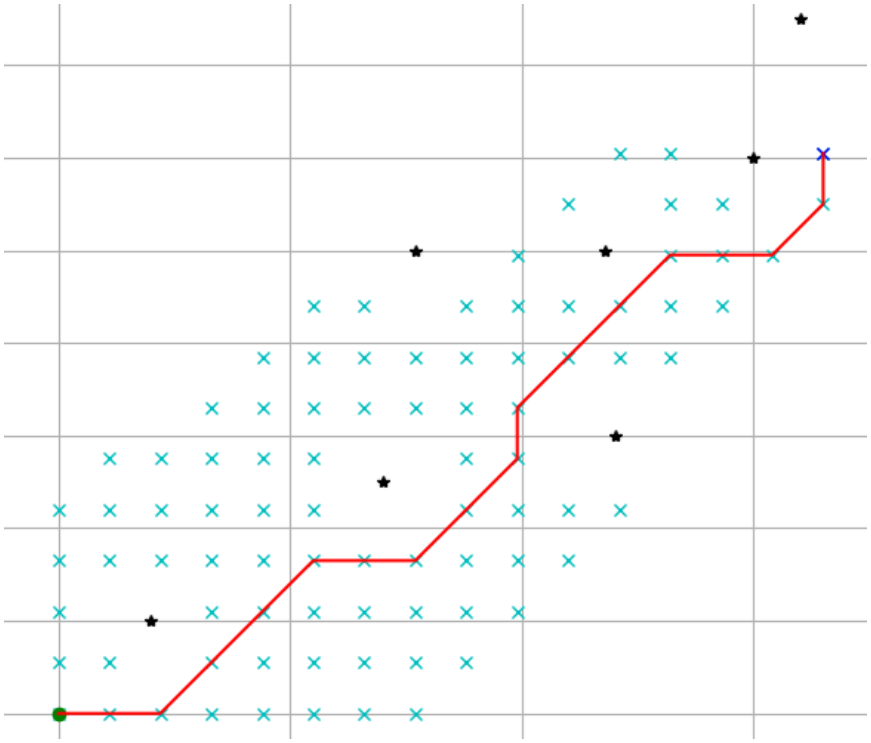


Figura 50. Prueba 2 de simulación del algoritmo A*

Prueba 3.

Mx= 0.61 m, My = 1.25 m, obx = [0.5, 0.2, 0.29] m, oby = [1.0, 0.2, 0.5] m

Figura 51. Prueba 3 de simulación del algoritmo A*

Se puede observar en la figura 51 que, al alcanzar la meta objetivo, el algoritmo detiene la búsqueda y por consiguiente la exploración de casillas libres.

5.3. Comparación entre los algoritmos de generación de trayectorias: Algoritmo basado en grafos A* y Algoritmo de campos potenciales.

Para la comparación de los algoritmos de generación de trayectorias se asignarán los mismos obstáculos y meta para encontrar la ruta hasta el punto objetivo mediante ambos algoritmos, determinando la ruta con menor distancia y que sea factible para la configuración cinemática del robot.

Prueba 1.

Mx= 1.18 m, My = 0.88 m, obx = [0.78, 0.2, 1.18, 1.2, 0.29, 0.55] m,

oby = [0.58, 0.2, 1, 0.6, 0.5, 0.22] m

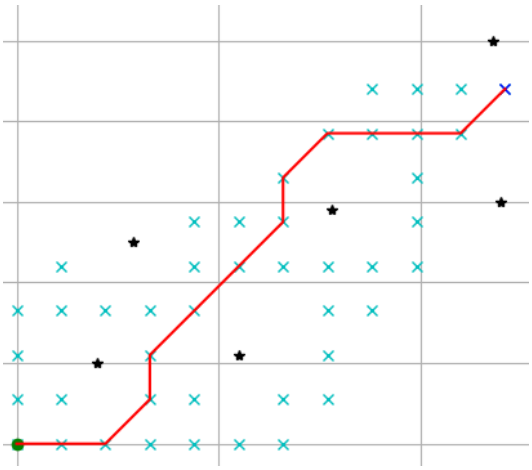
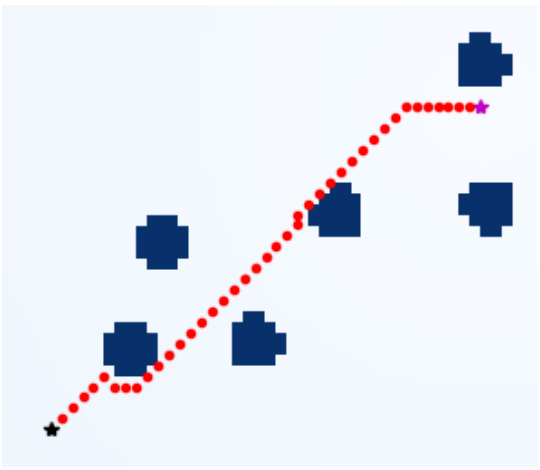
Algoritmo A*	Campos potenciales
	
Distancia recorrida: 1.71 m	Distancia recorrida: 1.57 m

Tabla 4. Comparación de los algoritmos de generación de trayectorias

Prueba 2.

Mx= 0.6 m, My = 1.14 m, obx = [0.3, 0.2, 0.44, 0.29, 0.0] m, oby = [1.17, 0.2, 0.89, 0.5, 0.66] m

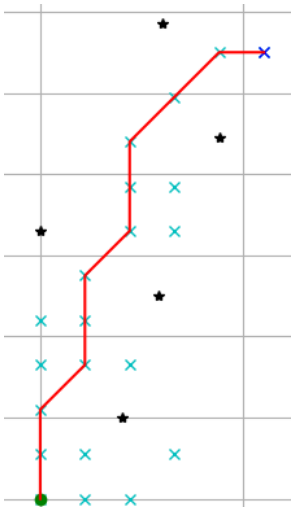
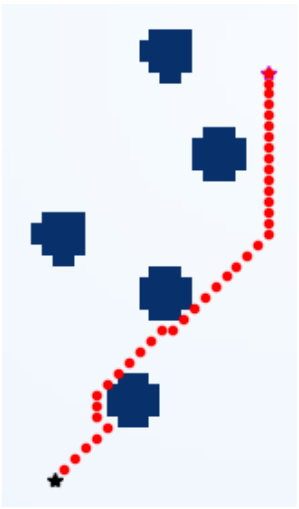
Algoritmo A*	Campos potenciales
	
Distancia recorrida: 1.392 m	Distancia recorrida: 1.4 m

Tabla 5. Comparación de los algoritmos de generación de trayectorias, prueba 2

Prueba 3.

Mx= 0.9 m, My = 0.3 m, obx = [0.4, 0.2, 0.66, 0.42, 0.22] m, oby = [0.12, 0.2, 0.3, 0.4, 0.0] m

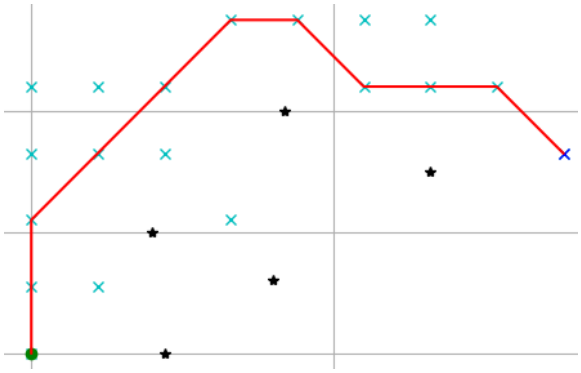
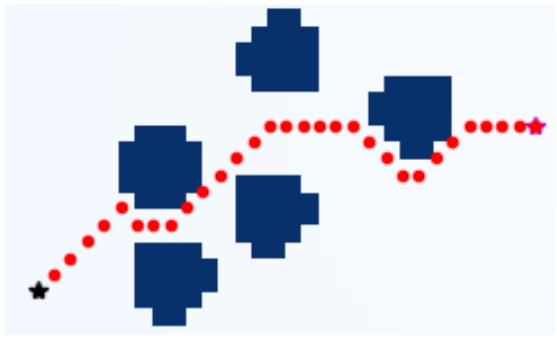
Algoritmo A*	Campos potenciales
 Distancia recorrida: 1.33 m	 Distancia recorrida: 1.1 m

Tabla 6. Comparación de los algoritmos de generación de trayectorias, prueba 3

Tomando en consideración las comparaciones realizadas para ambos algoritmos en un mismo escenario, se puede terminar que el algoritmo más eficiente es el de campos potenciales, puesto que este genera menos desviaciones en la trayectoria (rizados) teniendo tramos mayormente rectos y en la mayoría de las ocasiones el algoritmo de campos potenciales encuentra la ruta más corta.

La principal limitación del algoritmo de campos es que se pueda topar con un mínimo local, esto ocurre si en la ruta se encuentra a un objeto de geometría no convexa (forma de U etc.), entonces el algoritmo no habrá encontrado una ruta libre de obstáculos hasta la meta. Los mínimos locales no representan un problema para el algoritmo de A*, puesto que funciona extendiéndose en todas las direcciones en donde encuentra una casilla disponible y al encontrar la coordenada de la meta calcula la ruta más corta que pase por las casillas libres, debido a esto sólo puede trazar líneas a 90°, 0° y 45° en dirección a la casilla objetivo, cabe notar también que el algoritmo continua expandiéndose hasta las casillas en dirección contraria al objetivo realizando esfuerzo computacional.

6. CAPÍTULO 3.

En este capítulo se desarrollan las diferentes pruebas de simulación de las tareas asignadas a los robots móviles (exploración y navegación), mediante el uso de una interfaz gráfica de usuario.

A continuación, se describe la forma en que se integran los 3 algoritmos presentados (Algoritmo de generación de trayectoria, algoritmo de control de seguimiento de trayectoria, algoritmo de control de velocidad de cada rueda)

Se ejecuta el algoritmo de generación de caminos para obtener la ruta hasta el objetivo, este calcula una ruta libre de obstáculos, se obtiene entonces las coordenadas X y Y del camino como matriz de 2xn, donde n es el número total de puntos del camino, seguido se ejecuta el algoritmo de control de seguimiento de camino teniendo como entrada (SP) la matriz previamente descrita.

Previo al control de seguimiento de camino se aplica interpolación a los puntos de la matriz de entrada, y se calcula la velocidad lineal y angular a la que se debe desplazar el robot para seguir el camino SP, estos son la entrada para el algoritmo de control de velocidad de los motores del robot.

Se ejecuta el control de velocidad calculando la velocidad para cada llanta y asegurando el correcto seguimiento al SP de velocidad de cada una de las ruedas, mediante los encoders se calcula las velocidades lineales y angulares a las cuales se desplaza el robot, y se envían al algoritmo de seguimiento de camino para el cálculo del error y para el cálculo de las siguientes velocidades lineales y angulares para el desplazamiento del robot.

6.1. Pruebas de simulación para tareas de exploración y tareas de navegación.

Para las pruebas de simulación y experimentales se programó una interfaz gráfica (GUI) en Python. En la cual se ubican las tareas de navegación y las tareas de exploración.

La cuales consisten en:

- **Tareas de Exploración:**

El robot inicia en el escenario hacia objetivos cambiantes y durante el movimiento lee los sensores (distancia, detector de objeto metálico), para identificar la ubicación de obstáculos en el escenario y presentar una gráfica que identifique la posición de los obstáculos u objetos metálicos encontrados.

En la ejecución de esta tarea se ubica el robot en punto (0,0), y se establece un punto objetivo, teniendo en cuenta que para esta tarea no se tiene conocimiento previo del entorno o la ubicación de los obstáculos.

Se ejecuta el algoritmo de campos potenciales para obtener una ruta hasta la meta este calcula una ruta directa hasta el objetivo (debido a que no se cuenta con información de los obstáculos), con la ruta obtenida el algoritmo de seguimiento de camino calcula las velocidades para el desplazamiento, el algoritmo de control de velocidades de las ruedas inicia el desplazamiento del robot.

Durante este desplazamiento el robot lee constantemente los sensores de distancia y de metales, para detectar los objetos o metales que se pueda encontrar durante su avance hacia la meta, si el robot detecta algún obstáculo registra la posición encontrada, y mediante el algoritmo de generación de caminos recalcula la ruta hasta el objetivo esquivando el o los obstáculos encontrados durante la exploración.

Previamente se desarrolla un mapa 2D en el cual se ubican mediante una representación cartesiana los objetos en las ubicaciones encontradas identificando si eran metales u obstáculos.

- **Tareas de Navegación:**

Se indica a los robots las coordenadas de los obstáculos en el escenario y mediante algoritmos de generación y planificación de trayectorias se genera una ruta que evada los obstáculos hasta la meta, el cual el robot seguirá gracias al control de seguimiento de camino diseñado en el capítulo 2.

Para ejecutar la tarea de navegación se ubica el robot en punto (0,0), y se establece un punto objetivo, y se indica al algoritmo la ubicación de los obstáculos.

Para realizar la tarea de navegación se utiliza el algoritmo de Grafos A*, debido a que, por sus ecuaciones matemáticas de funcionamiento, ecuación (87), el algoritmo debe conocer la ubicación de los obstáculos en el entorno para obtener una ruta hasta la meta.

con la ruta obtenida el algoritmo de seguimiento de camino al igual que para la tarea de exploración, calcula las velocidades para el desplazamiento, el algoritmo de control de velocidades de las ruedas inicia el desplazamiento del robot.

Durante el desplazamiento el robot registra las posiciones por donde avanza, estas se registran en un mapa 2D en el cual se evidencia la correcta navegación del robot esquivando los obstáculos hasta el objetivo.

A continuación, se presenta la ventana principal de la interfaz GUI programada para las pruebas de simulación y experimentación:

Figura 52. Ventana principal de la GUI

En la ventana principal se puede navegar mediante 2 botones:

- Tareas de Navegación
- Tareas de exploración.

6.2. Pruebas de simulación de las tareas de navegación.

A continuación, se presenta la ventana de la GUI para simulación de las tareas de navegación para *n* robots móviles.

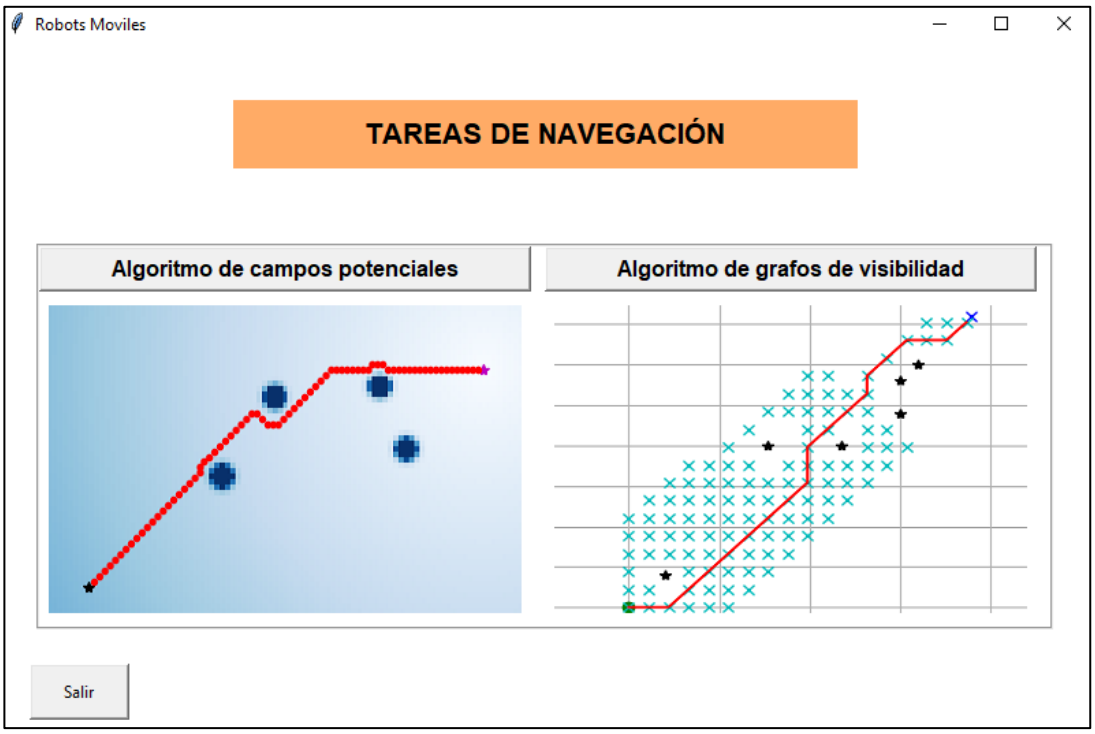


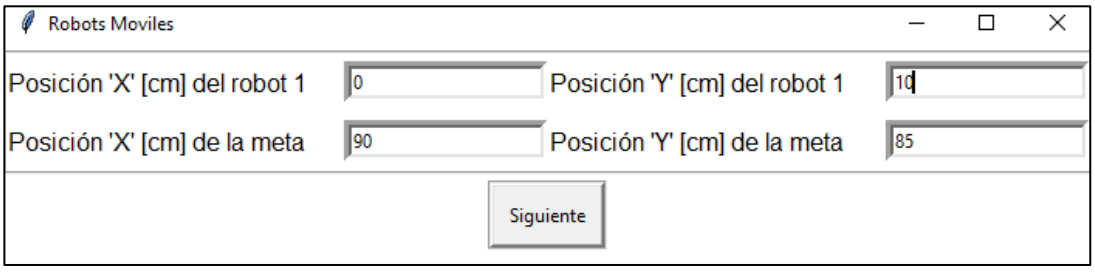
Figura 53. Ventana Tareas de simulación

Para las tareas de navegación se utilizará el algoritmo de grafos de visibilidad para obtener una ruta que llega a la meta esquivando los obstáculos.

Durante la ejecución de esta tarea se debe conocer la posición de los obstáculos en el entorno de simulación.

6.2.1. Pruebas de simulación para 2 robots.

Se utilizarán los siguientes puntos de inicio y meta correspondiente para cada robot.



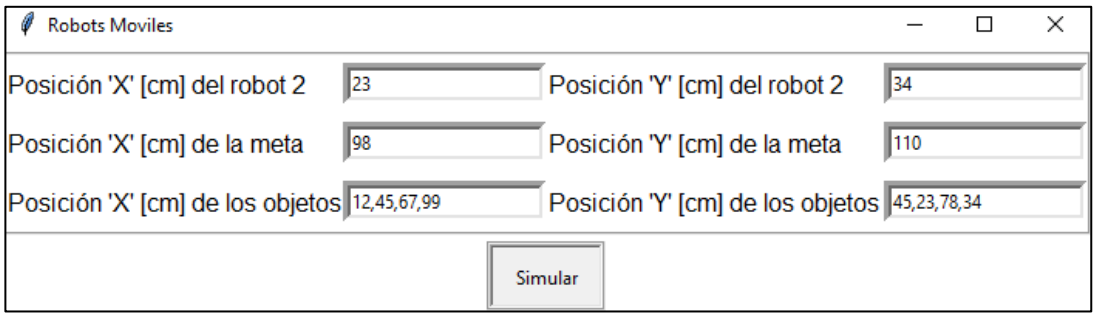


Figura 54. Condiciones iniciales de los robots y el escenario.

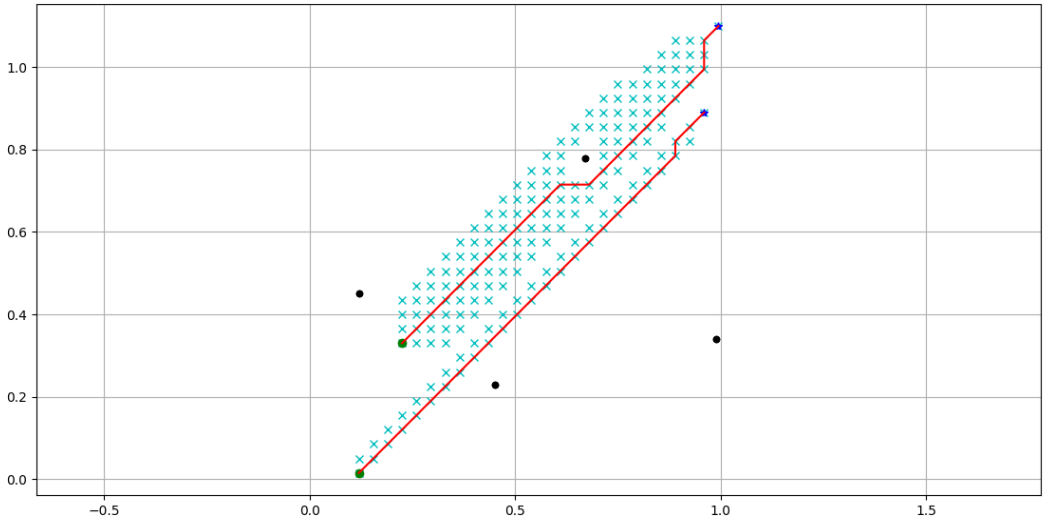


Figura 55. Ruta calculada por el algoritmo.

Se evidencia la ruta calculada por el algoritmo de grafos explorando casillas libres hasta llegar al objetivo.

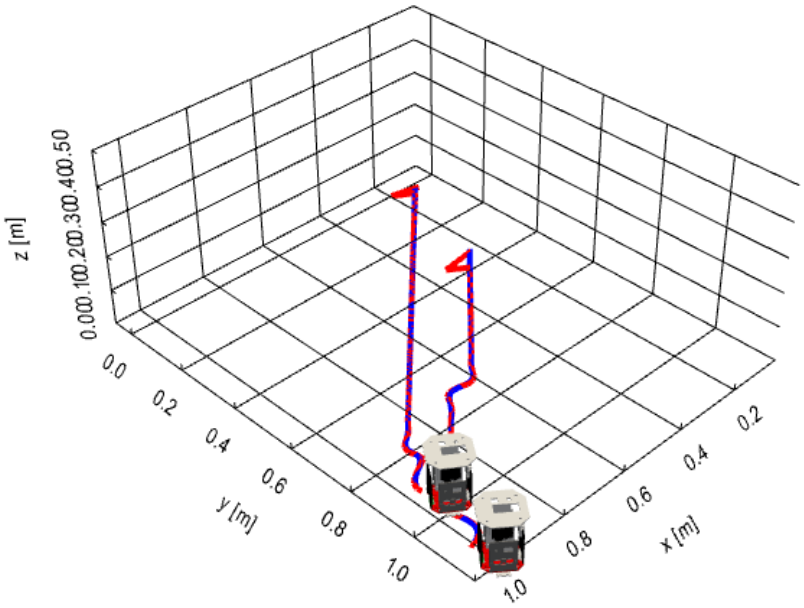


Figura 56. Simulación en entorno virtual del algoritmo para 2 robots móviles.

En la figura 56 se observa los dos robots en el ambiente de simulación, se puede evidenciar el correcto seguimiento de la trayectoria generada

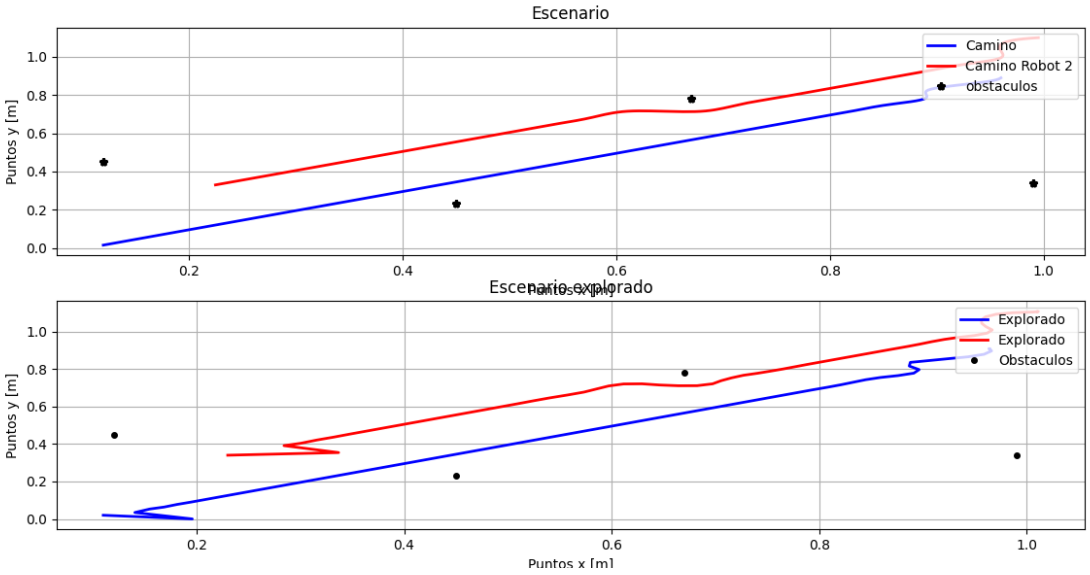


Figura 57. Simulación del robot en tareas de navegación.

En la gráfica superior de la imagen se observa la ruta calculada por el algoritmo y en la gráfica inferior se encuentra la ruta seguida por los robots, corroborando que estas rutas llegan a la meta sin pasar por los objetos en el escenario.

6.2.2. Prueba de simulación número 2 para 3 robots.

Para la simulación de las tareas de navegación para 3 robots se usaron las siguientes coordenadas correspondientes a los puntos de inicio y meta de cada robot móvil.

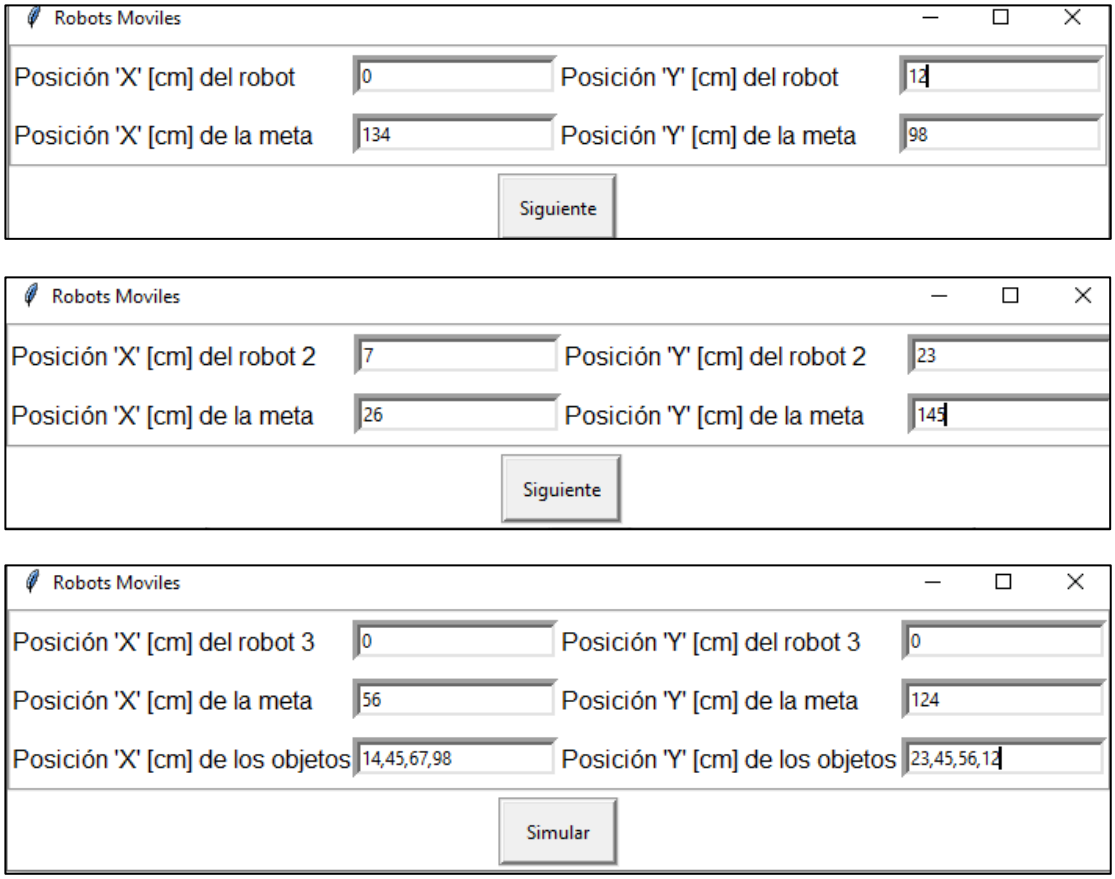


Figura 58. Condiciones iniciales de los robots y el escenario.

A continuación, se muestran las gráficas resultantes de la simulación del algoritmo para los puntos especificados.

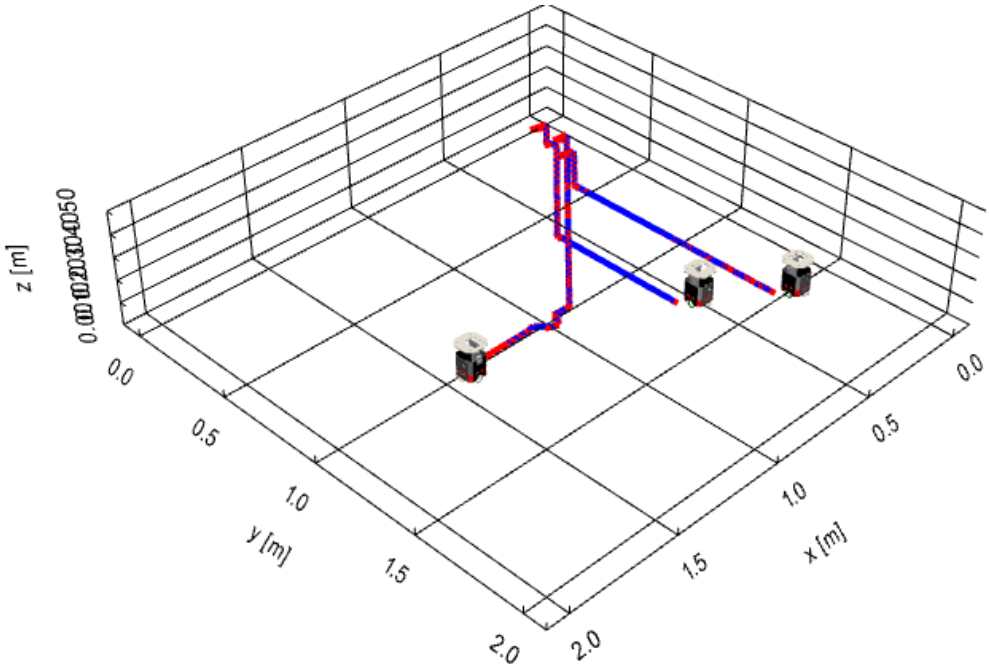


Figura 59. Simulación en entorno virtual del algoritmo para 3 robots móviles

se puede evidenciar el seguimiento de los 3 robots según la ruta calculada por el algoritmo de navegación.

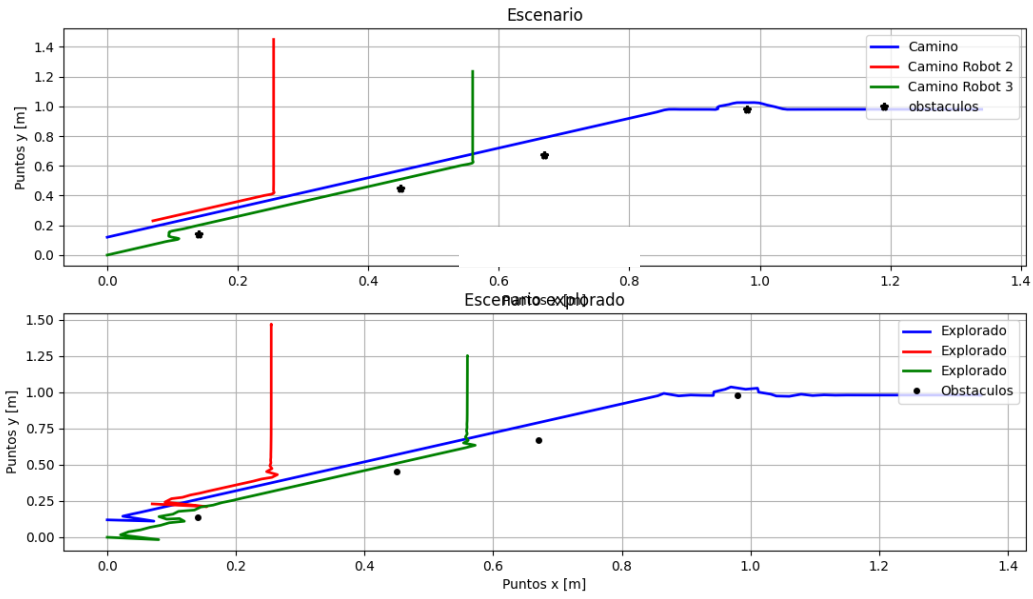


Figura 60. Simulación del robot en tareas de navegación.

Se puede observar la ruta dada por el algoritmo de navegación que corresponde a la gráfica superior de la imagen y la ruta seguida por los robots en la simulación en la gráfica inferior

6.3. Pruebas de simulación de las tareas de exploración.

Como se describió en la viñeta anterior para esta tarea los robots inician en el escenario hacia diferentes metas y durante el movimiento lee los sensores (distancia, detector de objeto metálico), para identificar la ubicación de obstáculos en el escenario y presentar un mapa cartesiano 2D que identifique la posición de los obstáculos u objetos metálicos encontrados.

Los algoritmos presentados se pueden emplear en *n* número de robots en un mismo entorno, y de esta forma se tendrá una ruta diferente para cada robot, y mayor rango de exploración, puesto que cada robot explora en cuanto se desplaza y con esto se puede generar un mapa con la recopilación de los datos obtenidos mediante cada robot.

Para la simulación con 2 robots se les asignan puntos de meta y puntos objetivo para cada uno dentro de un mismo entorno.

No sé indican posiciones de los obstáculos o artefactos metálicos puesto que para las simulaciones estos se ubicarán de forma aleatoria, el objetivo como ya se mencionó es determinar la posición de estos obstáculos en cuanto el robot los encuentre durante el desplazamiento, en la siguiente figura se presenta una simulación realizada para 2 robots.

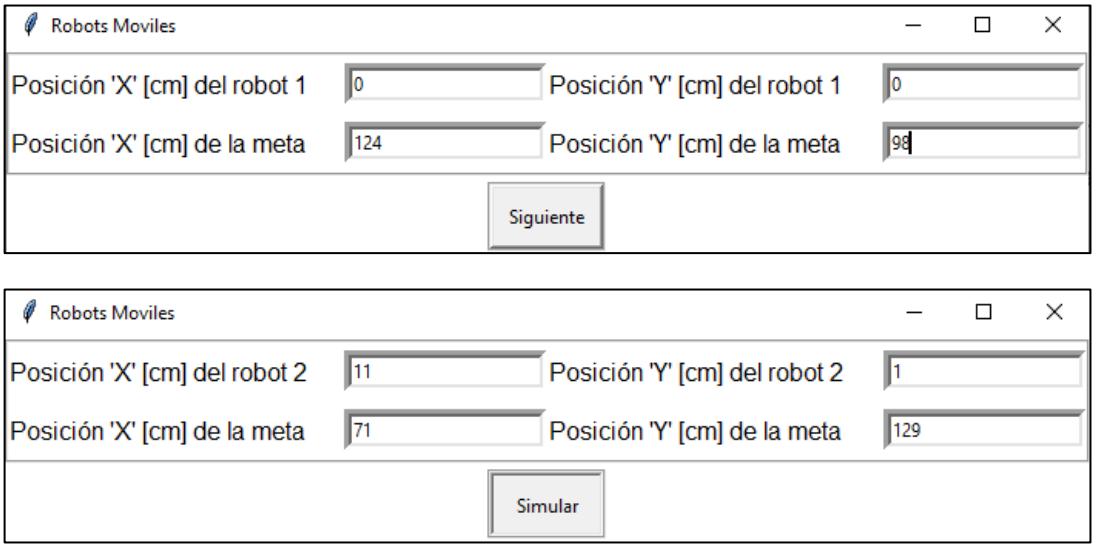


Figura 61. Condiciones iniciales de los robots.

En la figura 61 se presentan las coordenadas X y Y de inicio para cada robot al igual que las coordenadas objetivo para cada uno.

Mediante la ejecución de la simulación se obtiene la representación del entorno virtual para los dos robots móviles como se muestra en la siguiente figura:

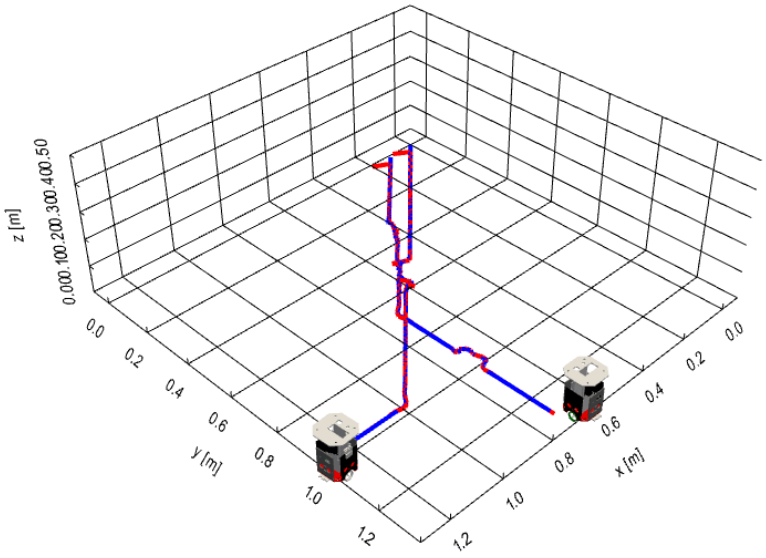


Figura 62. Simulación en entorno virtual del algoritmo para 2 robots móviles.

En la figura 62 se observa la ruta calculada por el algoritmo para cada robot, igualmente se evidencia el correcto seguimiento del robot a la ruta correspondiente.

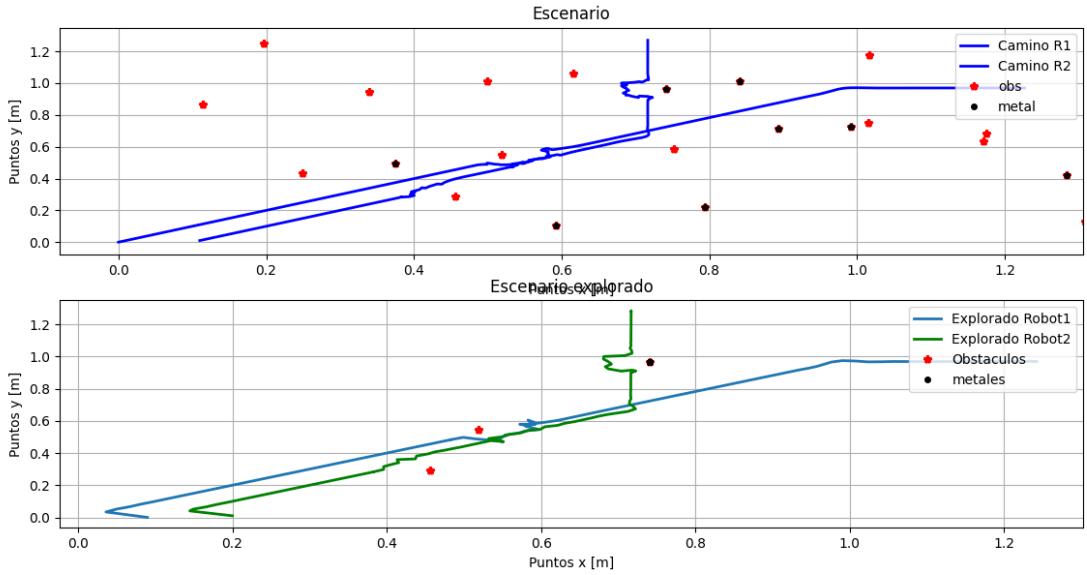


Figura 63. Escenario explorado.

Se observa en la gráfica superior de figura la 63 el escenario recreado con todos los metales y obstáculos ubicados virtualmente, en azul se observa la ruta que el robot toma hasta llegar a la meta correspondiente, se puede notar en las discontinuidades de los tramos de la ruta cerca a los objetos que el robot recalcula la ruta al encontrar un obstáculo en el camino.

En la gráfica inferior de la figura 63 se observa el camino por donde pasaron los robots durante la simulación, se puede evidenciar el mapa que crea identificando solo los obstáculos que encontraron en su desplazamiento.

6.3.1. Simulación número 2 para 3 robots

Se presenta una segunda simulación del algoritmo de exploración, para 3 robots para los cuales la metodología sigue siendo la ya descrita, se asignan puntos de meta para cada robot, en el entorno de simulación y se ejecuta ejecutan los algoritmos de generación y seguimiento de caminos obteniendo el siguiente resultado:

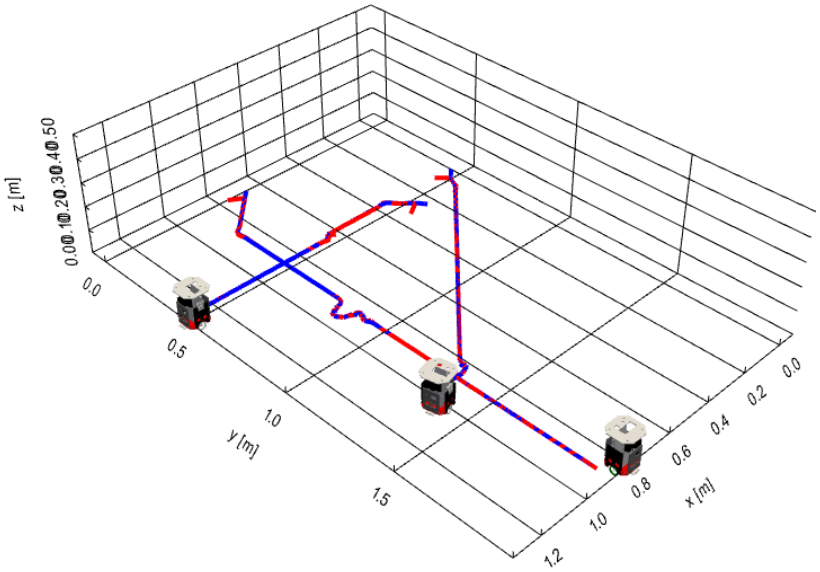


Figura 64. Simulación en entorno virtual del algoritmo para 3 robots móviles

En la figura 64 se observa el recorrido para 3 robots en el entorno de simulación, evidenciando el correcto seguimiento del camino.

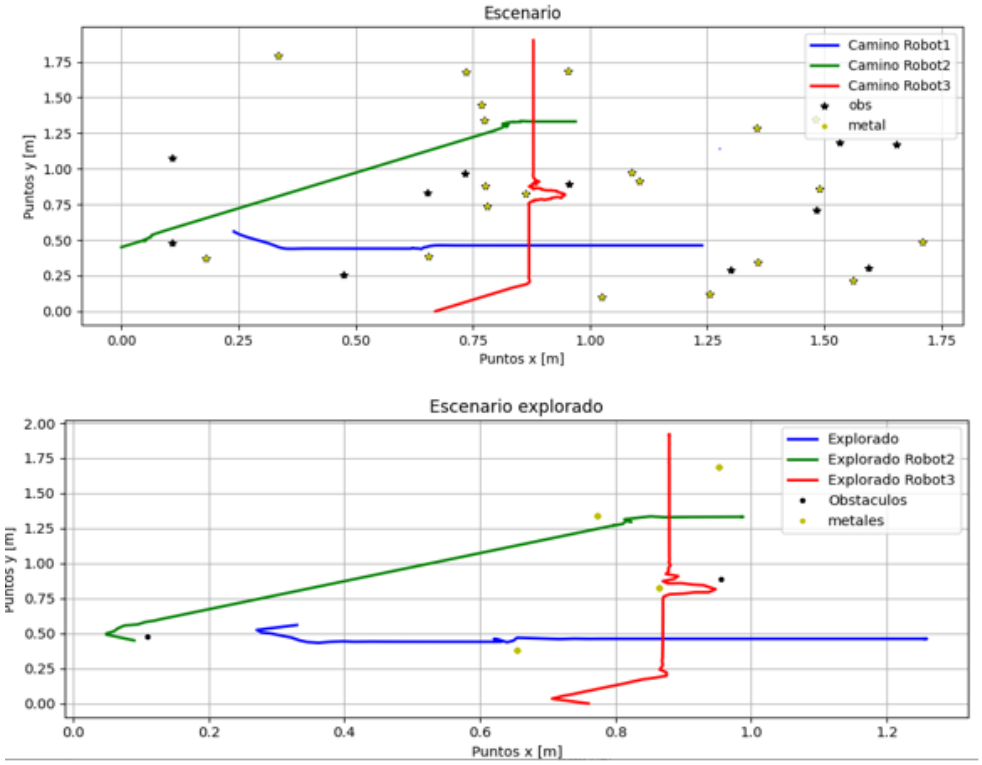


Figura 65. Escenario explorado 3 robots móviles.

En la figura 65 se observa el escenario para con todos los obstáculos ubicados de forma aleatoria en el escenario, se evidencia la ruta para que cada robot alcance la meta respectiva, en la gráfica inferior se evidencia los objetos y artefactos metálicos con los que se encuentras los robots durante la exploración.

6.3.2. Graficas para diferentes muestras simuladas

El principio de funcionamiento descrito hasta ahora para 2 y 3 robots se puede replicar de igual forma para n robots.

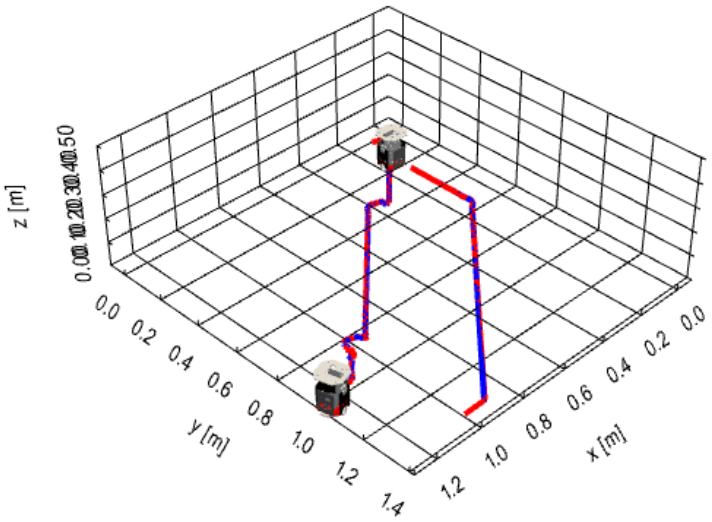


Figura 66. Simulación en entorno virtual.

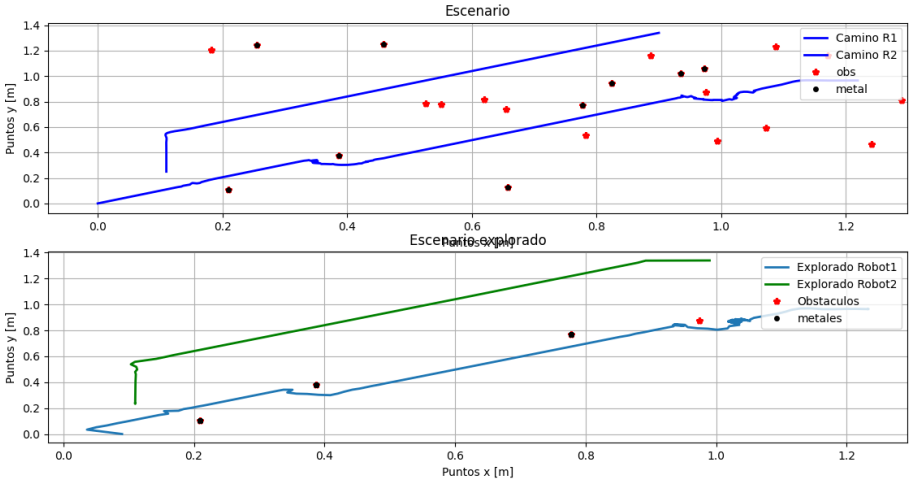


Figura 67. Escenario explorado

7. CAPÍTULO 4.

En este capítulo se presenta la validación de los modelos determinados en el capítulo 2 y la implementación experimental de las tareas de navegación y exploración descritos en el capítulo 3

Para la validación de los modelos de cinemática diferencial, control de camino, control de posición, control de caminos y algoritmos de planificación en entorno real.

Se probarán los algoritmos de nivel medio en el robot móvil en una superficie plana de material madera tipo MDF buscando que el robot pueda avanzar sin problemas con la geografía del terreno y tratando de disminuir el deslizamiento. El espacio para pruebas de validación es de 1.8 [m] x 1.5 [m], lo que componen un área total de 2.7 [m2] y se encuentra cuadriculado con cuadros de 10 [cm] x 10 [cm]

7.1. Pruebas experimentales para el algoritmo de control de posición

Prueba 1.

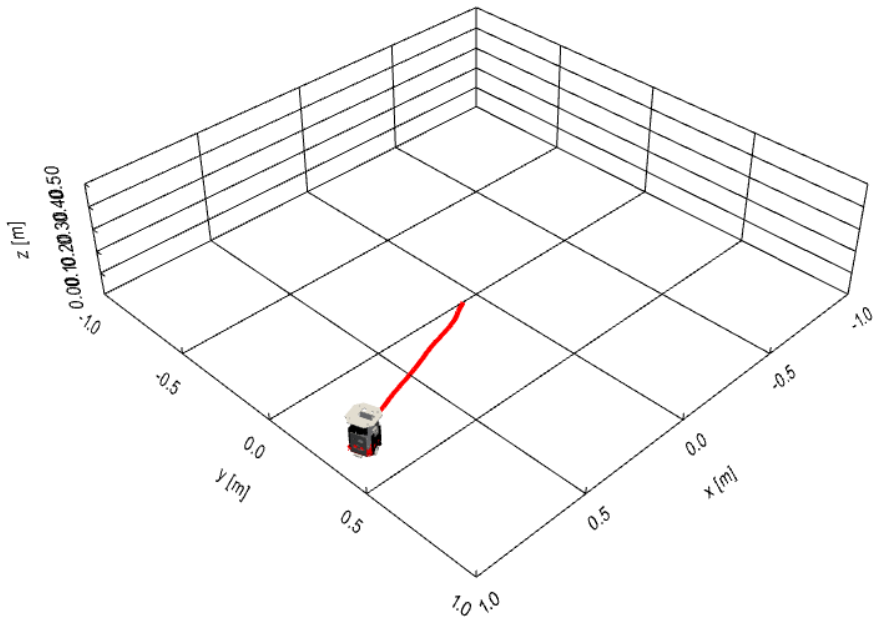


Figura 68. Reconstrucción virtual de la implementación

En la figura 68 se visualiza que el robot en el entorno virtual realiza el mismo movimiento que en el entorno real.

Para el cálculo de la velocidad lineal y angular del robot partiendo de la velocidad medida por los encoder en cada rueda se usan las ecuaciones (45) y (46)

$$\omega = (\mu_R + \mu_L)/dR \quad \textbf{(45)}$$

$$\mu_g = \frac{R}{2}(\mu_R + \mu_L) \quad \textbf{(46)}$$

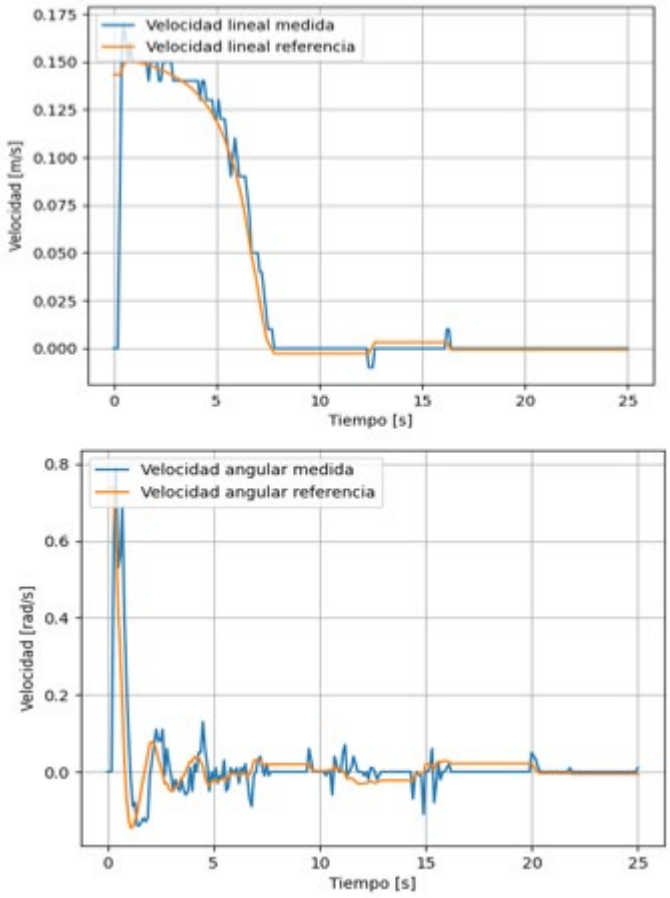


Figura 69. Velocidad lineal y angular desarrollada en el robot.

Se observa en la figura 69 la velocidad lineal y angular con la que se desplaza el robot comparado con la referencia se evidencia un seguimiento óptimo de la señal de SP.

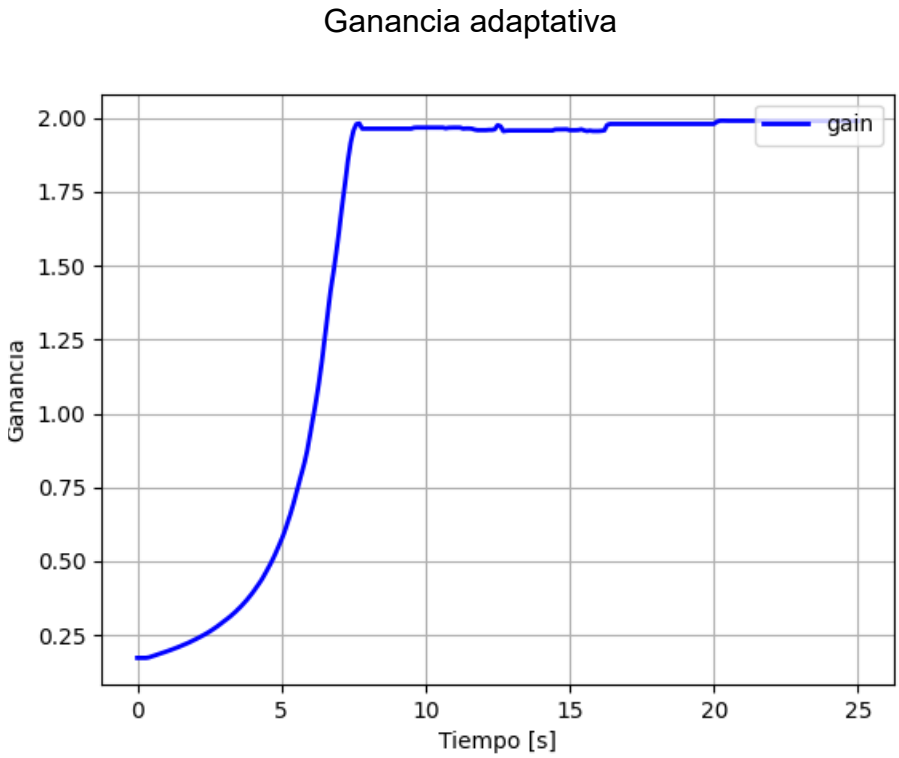


Figura 70. Ganancia adaptativa generada

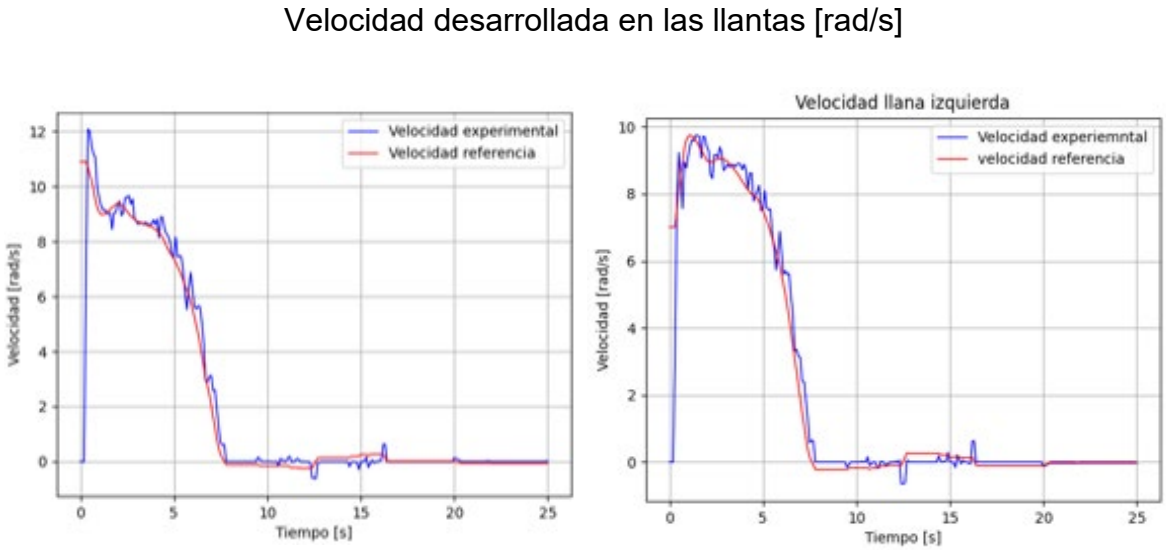


Figura 71. Velocidad medida en el encoder para cada llanta

En la figura 70 y 71 se observan las velocidades necesarias en cada rueda para llegar al punto especificado.

Prueba 2 para el algoritmo de control de posición:

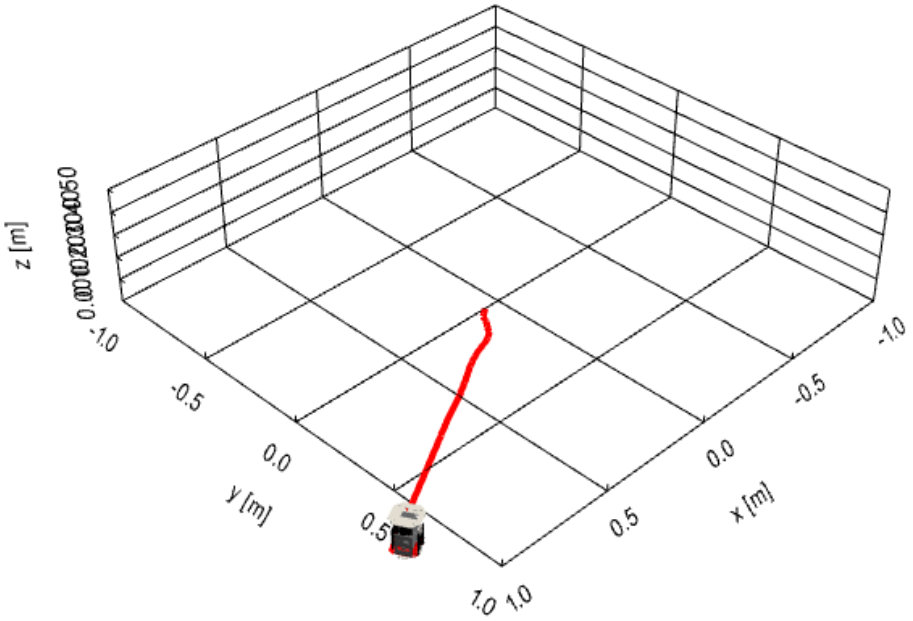


Figura 72. Reconstrucción virtual de la implementación.

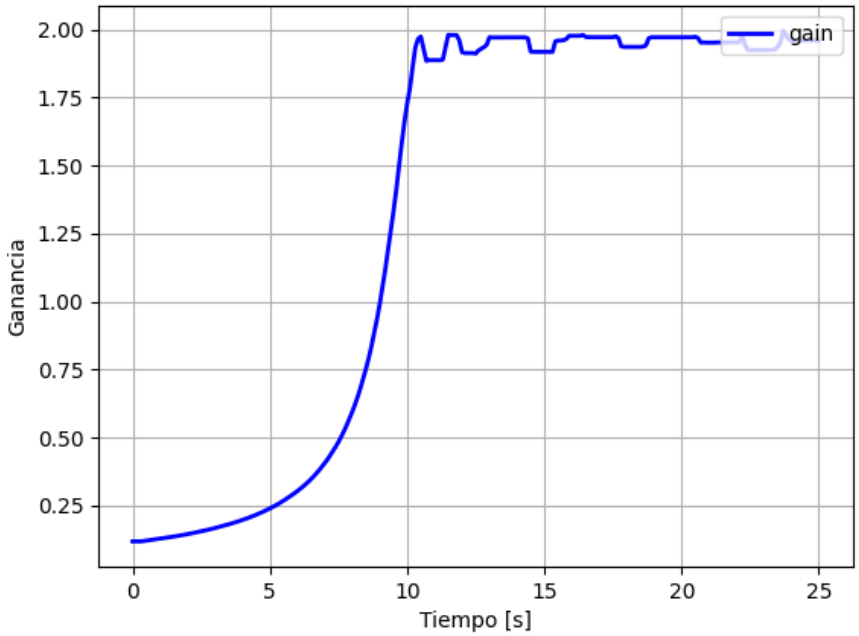


Figura 73. Variación de la ganancia

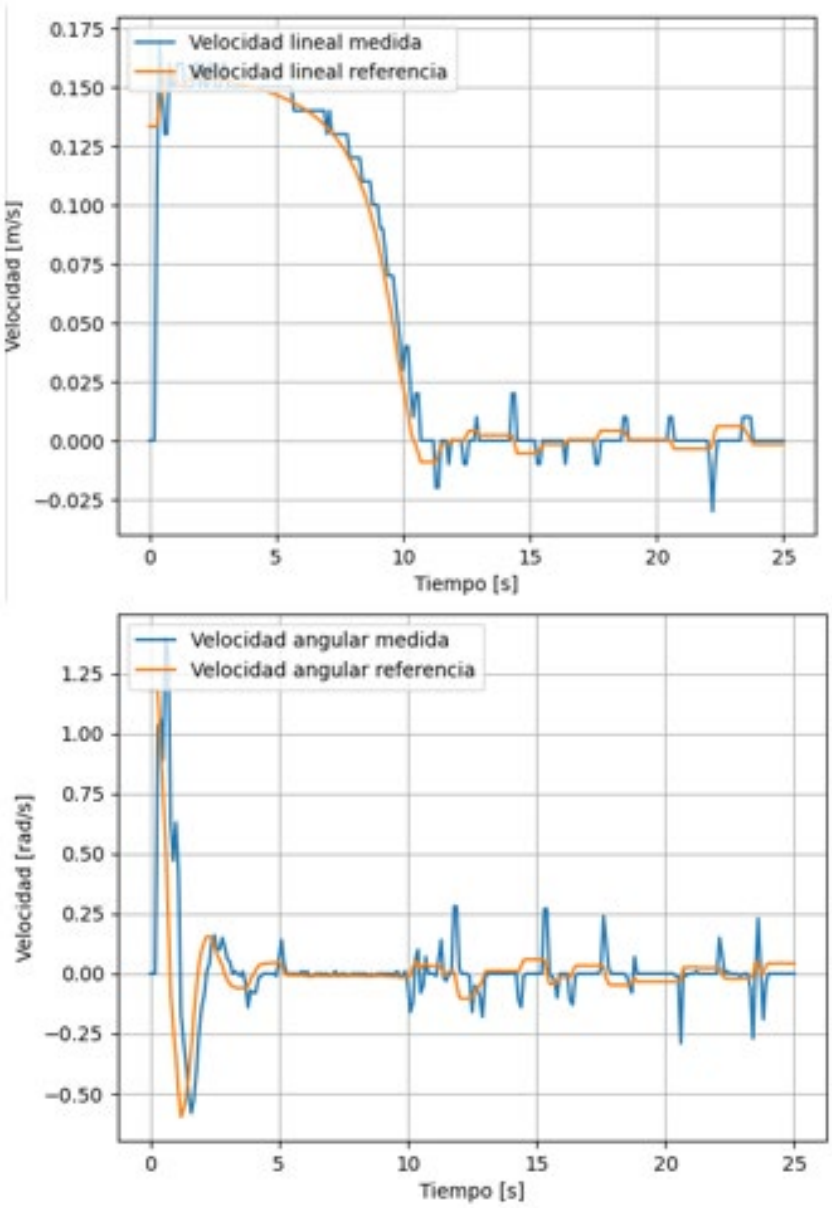


Figura 74. Velocidad lineal y velocidad angular

Se puede observar en la figura 74 la velocidad lineal y angular de referencia y las velocidades con las que se desplaza el robot, estas presentan un seguimiento óptimo de la señal de SP.

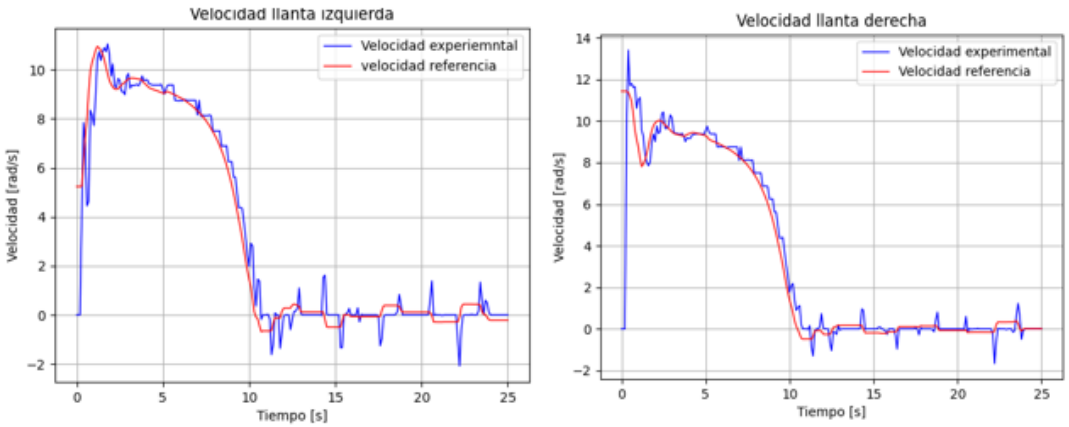


Figura 75. Velocidad angular llanta izquierda **y velocidad angular llanta derecha.**

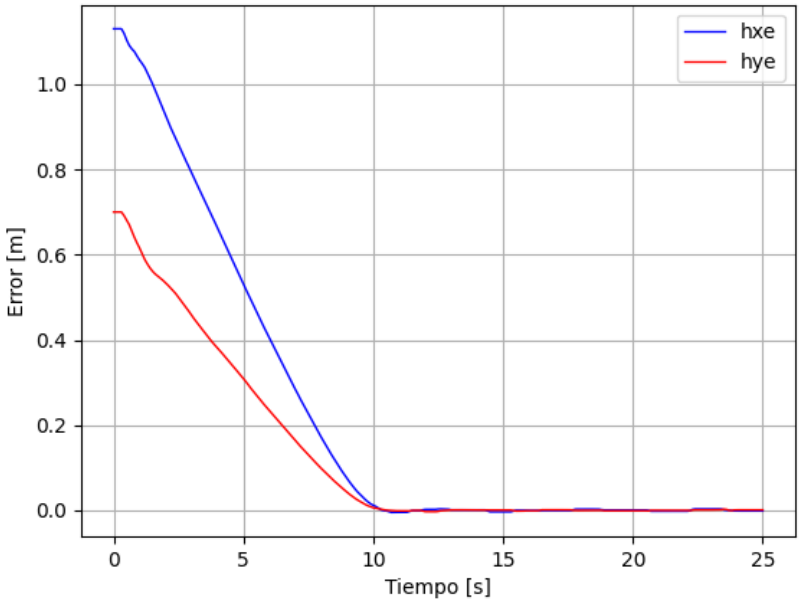


Figura 76. Errores de posición.

Se puede observar en la figura 75 un correcto seguimiento a las velocidades de referencia para cada rueda, y en la figura 76 se comprueba que el error de posición converge a cero (0).

- **Análisis de los resultados del algoritmo de control de posición.**

El robot logra desplazarse hacia la meta objetivo y moverse a las velocidades globales lineal y angular indicadas por el algoritmo de control de posición, como se evidencia en las gráficas de las figuras 74, y 75.

Como se puede observar en la figura 76 el robot logra llegar a la posición objetivo con un error cero, en la trayectoria X y Y.

7.2. Pruebas experimentales para el algoritmo de control de caminos.

Prueba 1.

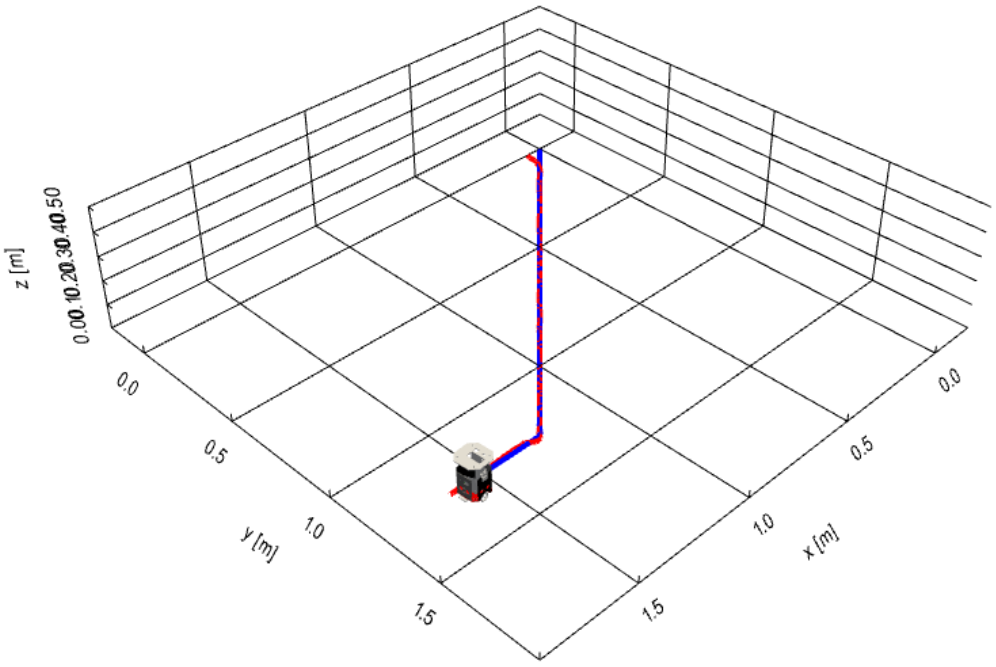


Figura 77. Grafica de la implementación con el camino a seguir.

Se observa en la figura 77 la trayectoria que el robot sigue reconstruida en el entorno virtual.

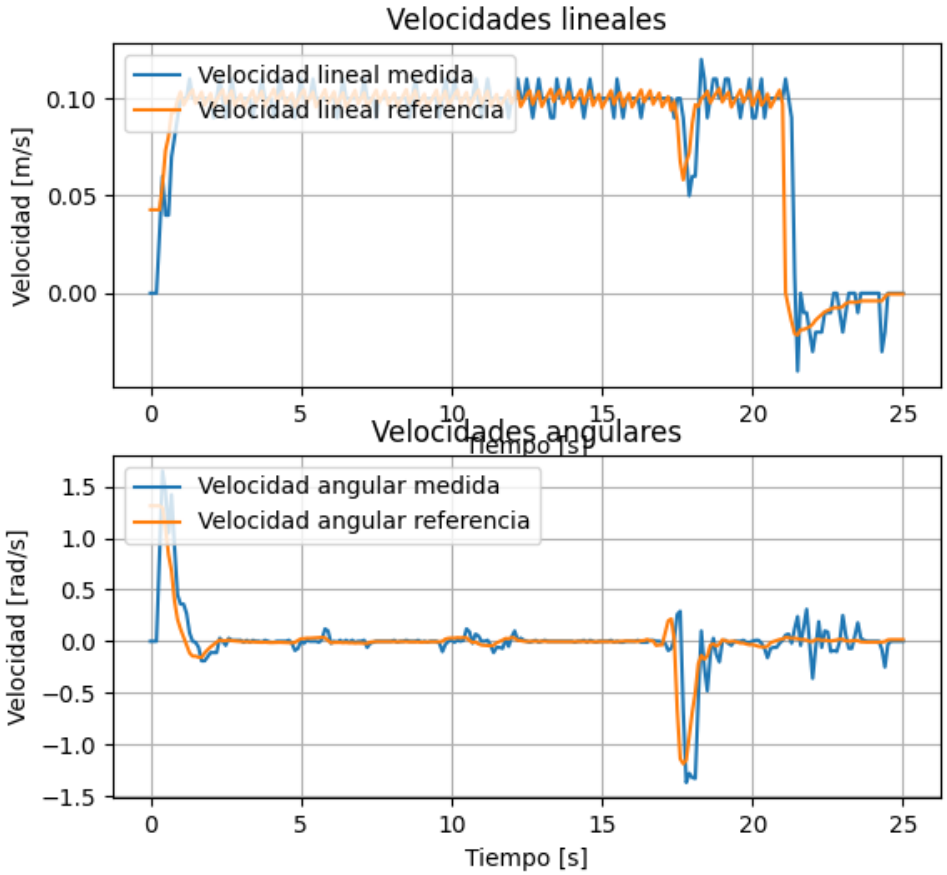


Figura 78. Velocidad lineal y velocidad angular

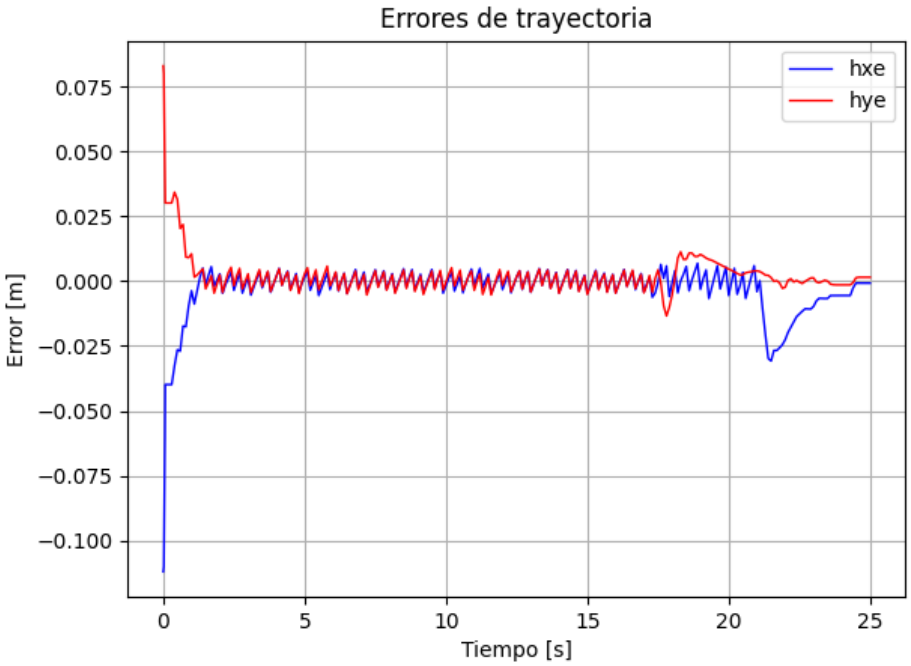


Figura 79. Errores de trayectoria.

Se puede determinar de la figura 78 el seguimiento que le hace el robot a las velocidades angulares y lineales calculadas por el algoritmo, y en la figura 79 se evidencia que el error converge en cero.

Prueba 2:

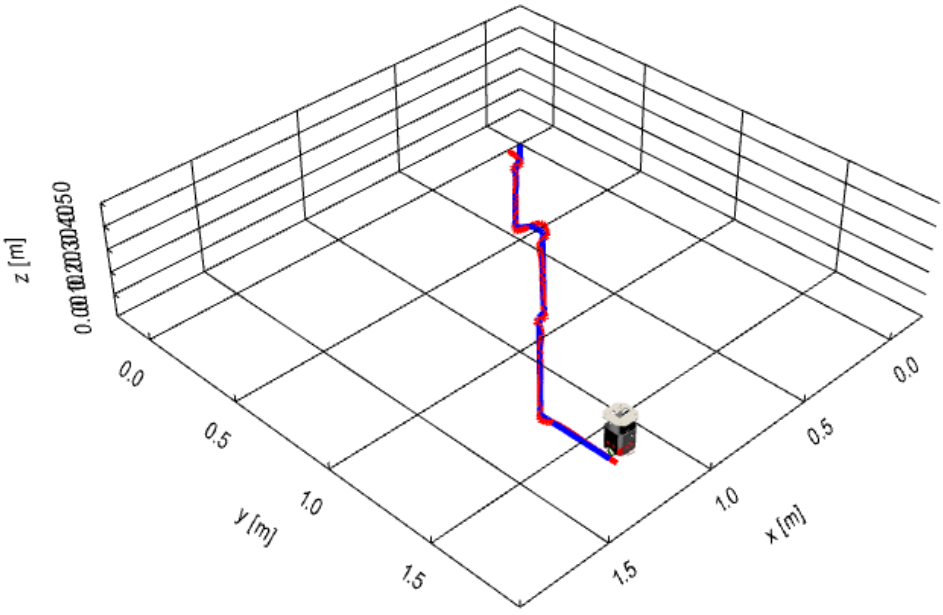


Figura 80. Grafica de la implementación con el camino a seguir prueba 2.

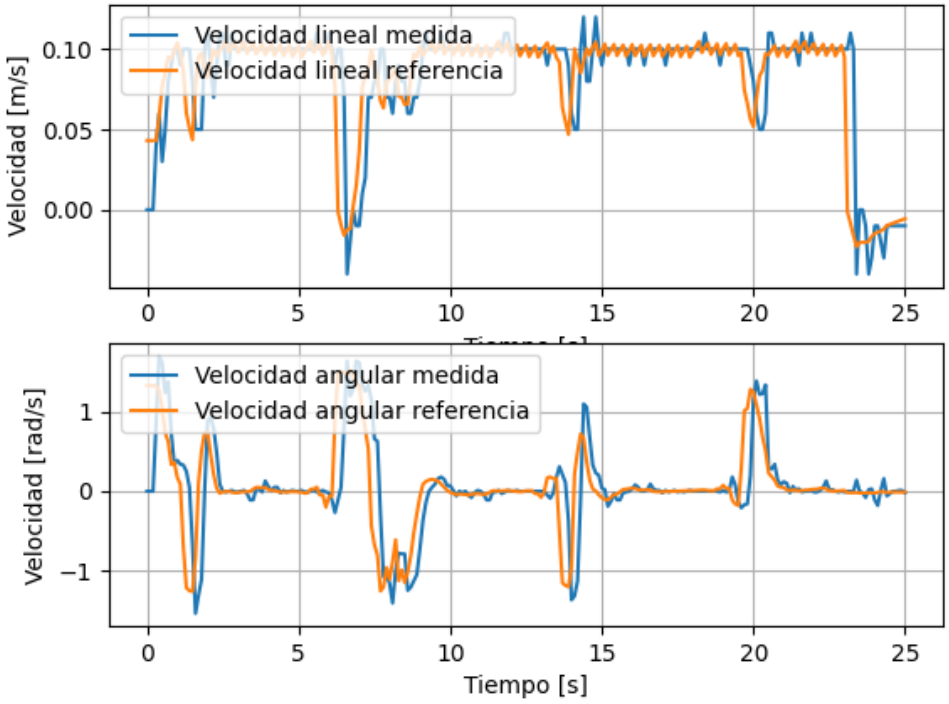


Figura 81. Velocidad lineal y velocidad angular.

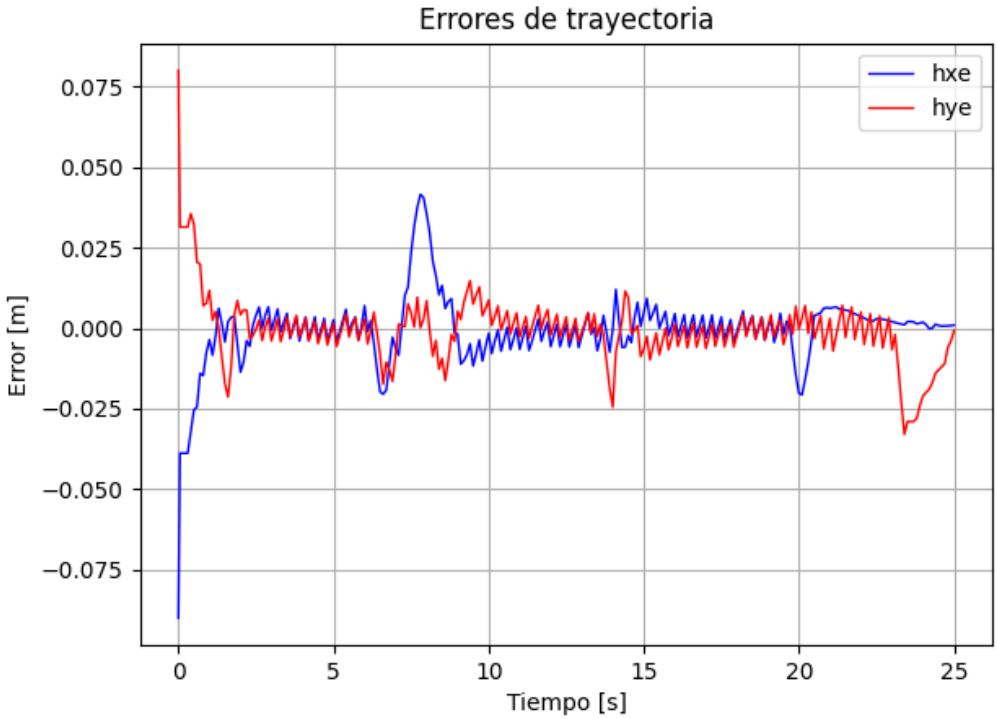


Figura 82. Errores de trayectoria.

Al igual que para la prueba 1 se evidencia que el robot sigue de forma óptima las velocidades lineales y angulares de referencia y los errores de trayectoria convergen a cero, se evidencia en la figura 80 el correcto seguimiento del camino planteado.

7.3. Pruebas experimentales de las tareas de exploración y navegación.

Para las pruebas experimentales se realizaron pruebas en el mismo escenario en el cual se probaron los algoritmos de control.

7.4. Pruebas experimentales Tareas de exploración.

Para las pruebas experimentales de las tareas de exploración se planteó el siguiente escenario:

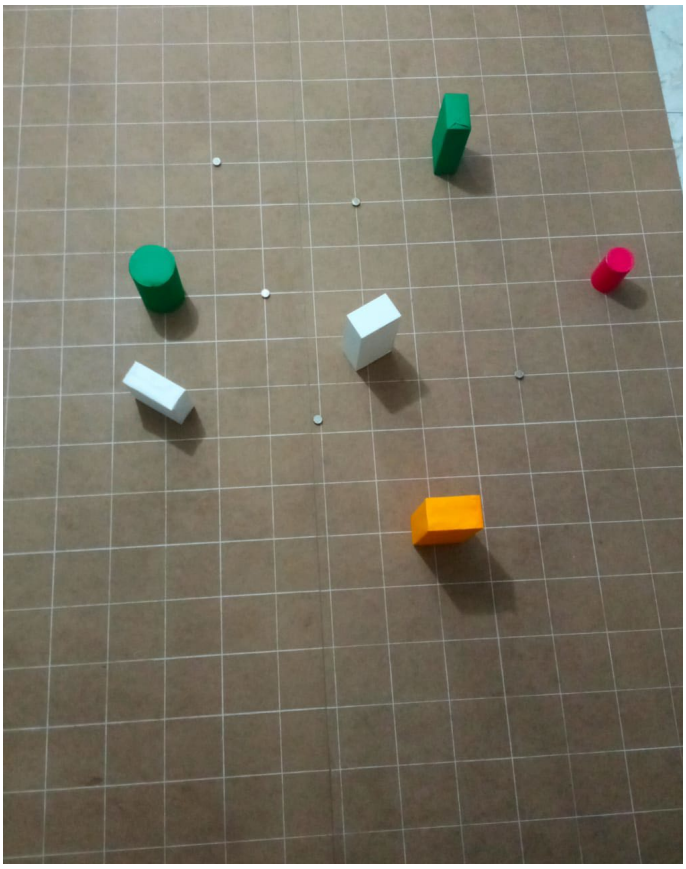


Figura 83. Escenario para pruebas de exploración.

Para la prueba experimental también se usará la interfaz gráfica presentada en el capítulo 3 como se muestra en la siguiente imagen.

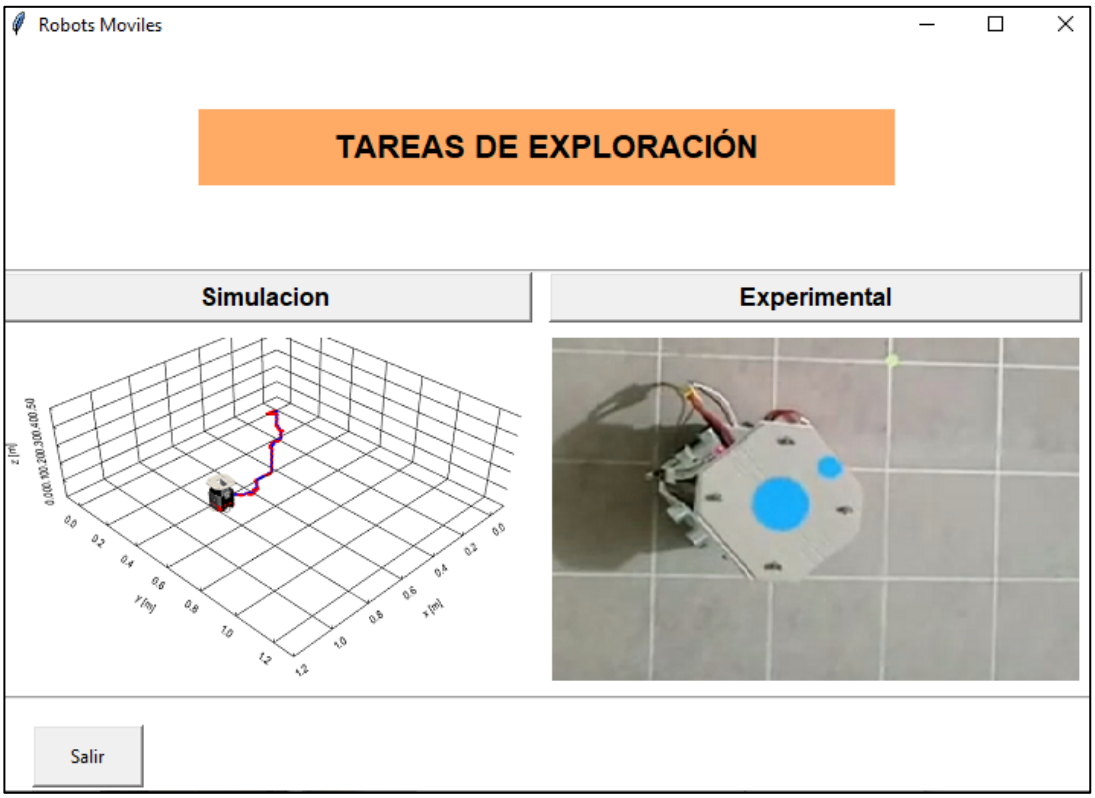


Figura 84. Ventana tareas de exploración – Experimental.

Se indican las condiciones iniciales del robot y el algoritmo procede a ejecutarse.

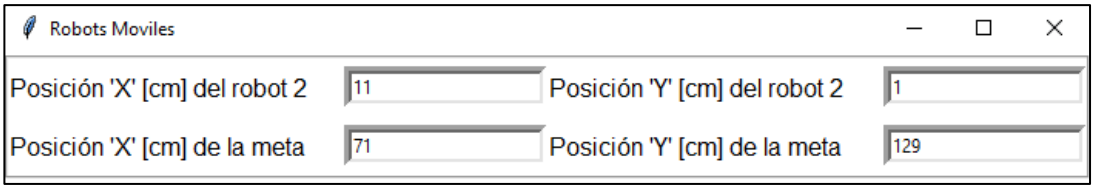


Figura 85. Condiciones iniciales de los robots y el escenario

Se ubicó el robot en un punto inicial y este atraviesa el escenario detectando objetos que encuentra en su trayectoria como se muestra en la siguiente imagen.

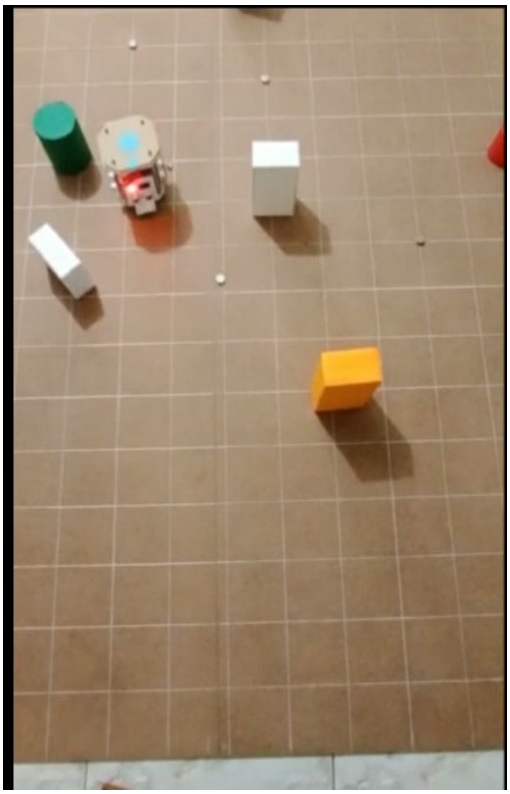


Figura 86. Robot atravesando el escenario (video adjunto: Exploracion_1).

La figura anterior es un recorte del video adjunto con el nombre “Exploracion_1” en el cual se observa a un robot atravesando el escenario mientras realiza exploración.

De la cual se obtiene el siguiente mapa con los obstáculos identificados.

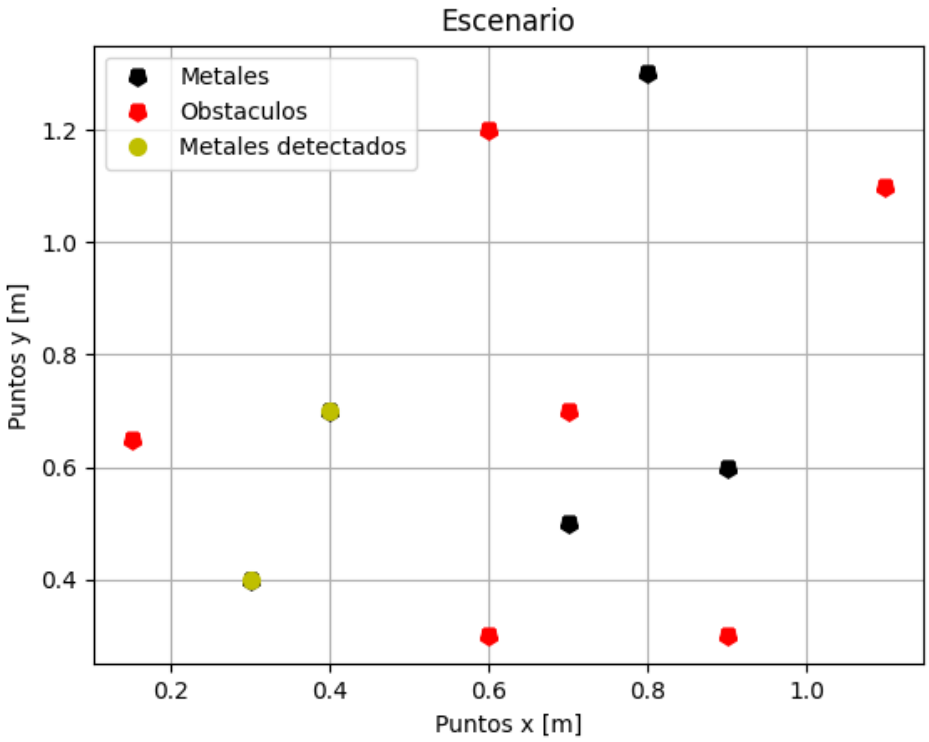


Figura 87. Mapa 2D con la ubicación de los objetos (metales) detectados en el escenario (color amarillo).

La figura anterior representa en color negro los metales distribuidos en el escenario, en color rojo los obstáculos y en amarillo los metales que el robot detecto durante la exploración.

Esta grafica presenta la ubicación de los objetos y artefactos metálicos en el entorno de validación, y en amarillo se representan los artefactos metálicos detectados, según el video adjunto con el nombre “Exploracion_1”

El robot se mueve a través del escenario de validación hacia un punto objetivo con un camino determinado, si mediante sensores de distancia o de metales (los cuales representan los detonadores metálicos de minas antipersonales) encuentra algo, este se detiene registra la ubicación del objeto y recalcula una nueva ruta de exploración esquivando el objeto encontrado.

Como se observa en la figura 87 y en el video adjunto el robot detecto 2 objetos metálicos durante su paso por el escenario

7.5. Prueba numero 2 para un segundo robot.

Se repitió el algoritmo para un segundo robot en el escenario como se ve en el video adjunto “Exploracion_2”

Figura 88. Robot atravesando el escenario (video adjunto: Exploracion_2).

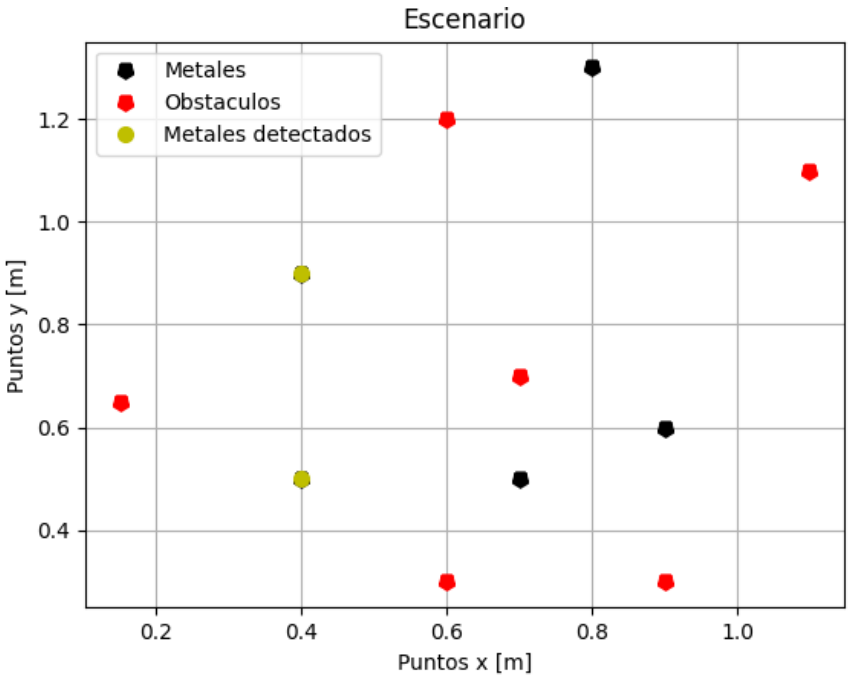


Figura 89. Objetos (metales) detectados en el escenario (color amarillo).

La figura anterior representa en color negro los metales distribuidos en el escenario, en color rojo los obstáculos y en amarillo los metales que el robot detecto durante la exploración según el video adjunto con nombre: Exploración_2.

7.6. Pruebas experimentales Tareas de navegación.

Para la prueba experimental de las tareas de navegación, también se usará la interfaz gráfica presentada en el capítulo 3: Como se muestra en la siguiente imagen.

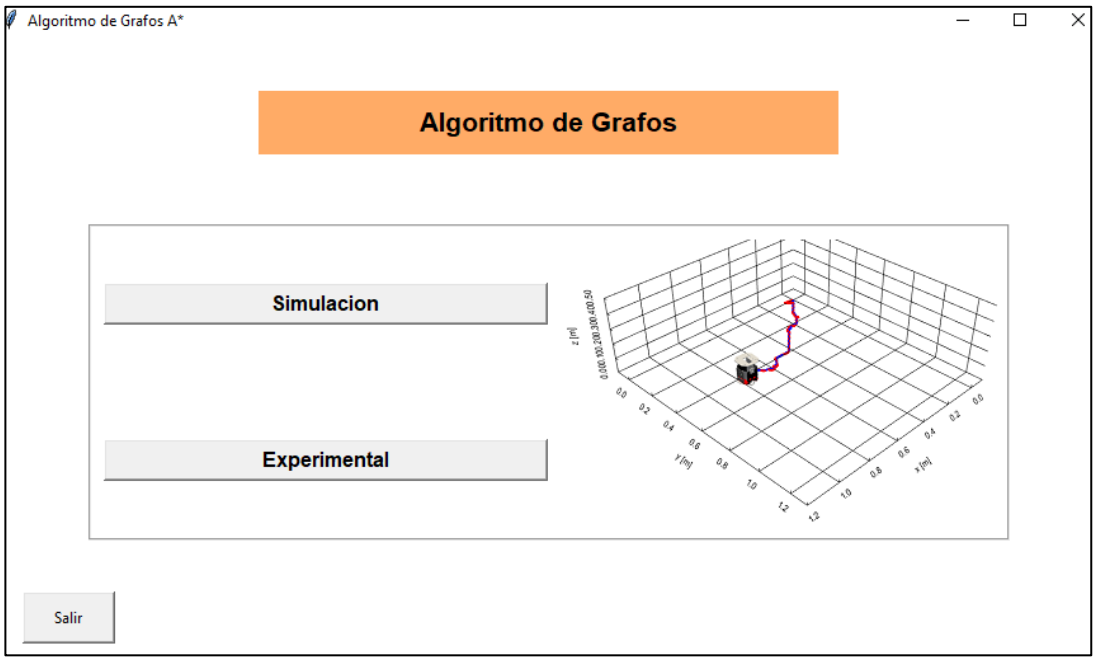


Figura 90. Ventana para la simulación experimental de la tarea de Navegación.

Se realizará la prueba experimental con 2 robots ubicándolos en el escenario y distribuyendo obstáculos a través de este, que luego serán identificados (posicionalmente) y registrados en el programa para los algoritmos de planificación de trayectorias.

Se asignan condiciones iniciales a los robots, de igual forma para el algoritmo de navegación basado en grafos A* se debe ingresar la posición de los obstáculos en el escenario, y se procede a ejecutar la prueba de validación

Se ubican en el escenario de validación obstáculos y los robots en los puntos de partida para la prueba experimental como se muestra en la siguiente imagen.

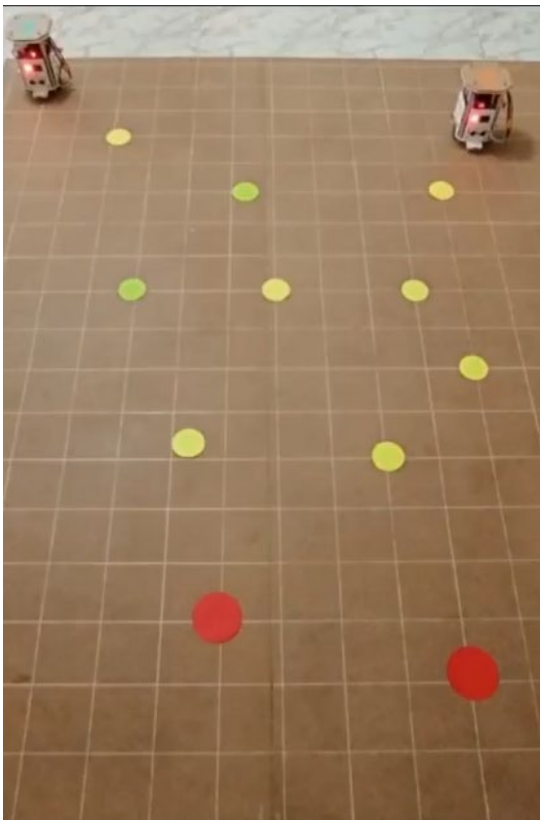


Figura 91. Escenario para validación de navegación. (video adjunto: Navegacion_1).

La figura 99 es un recorte del video adjunto con el nombre “Navegación_1” donde se observa que los robots de forma conjunta siguen una ruta alejada de los obstáculos hasta el objetivo.

7.7. Prueba número 2.

Para una segunda prueba de las tareas de navegación se configuró un nuevo escenario y se ejecutó el algoritmo a través de la interfaz como se muestra en la siguiente imagen y en el video adjunto “Navegación_2”.

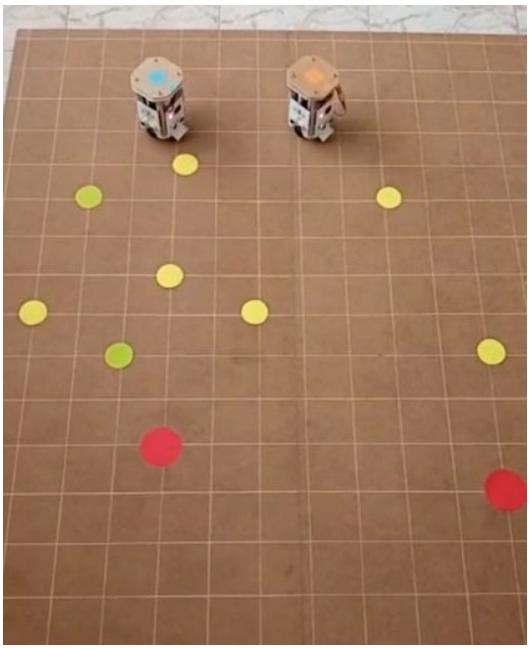


Figura 92. Escenario para validación de navegación. (video adjunto: Navegacion_2).

El anterior es un recorte del video adjunto “Navegación_2” donde se muestra el resultado de la prueba experimental realizada.

- **Análisis de los resultados**

Se establece como la mejor metodología para la generación de caminos en robots móviles al algoritmo de campos potenciales, puesto que mediante este es posible ejecutar las dos tareas planteadas en este informe (Navegación y Exploración) gracias a que este algoritmo puede funcionar sin especificar algún dato de obstáculos lo que lo hace funcional tanto en ambientes estáticos (con conocimiento de la posición de los obstáculos) y en ambientes dinámicos (sin conocimiento de los obstáculos o con obstáculos en movimiento).

En las pruebas experimentales se evidencia como los robots siguen la ruta calculada por los algoritmos con un mínimo error, este se presenta debido a los efectos del deslizamiento en las ruedas, puesto a que la posición del robot se determina mediante los encoders en cada motor se puede dar el caso que la rueda deslice un poco y el algoritmo contar esto como desplazamiento del robot.

8. CONCLUSIONES

Debido al rango limitado del sensor HC-SR04 (sensor de proximidad) el cual solo detecta objetos si estos se ubican justo en frente del buzzer “Trigger” es común que el robot no detecte obstáculos que si bien se encuentran en su lado frontal no están justo frente al Trigger del sensor, por lo cual el sonido no rebota en este.

El control de velocidad de las ruedas debe ser preciso puesto que esto evitará que se produzcan velocidades angulares no deseadas debido a una pequeña desviación en la velocidad de alguna de las ruedas, esto conlleva a tener un sensor de medición de velocidad de alta resolución y/o precisión.

Para la generación de trayectorias en robots móviles de sistemas de tracción diferencial, se deben considerar las características físicas (geometría y construcción), ya que de estas dependen el tipo de trayectorias que puede seguir el robot; se debe tener en cuenta que los puntos del camino o trayectoria no pueden presentar ángulos rectos al cambiar de dirección puesto que las restricciones cinemáticas del robot no permiten al robot hacer cruces cerrados o con ángulos a 90°.

Es importante interpolar la ruta o camino que genera los algoritmos de generación de ruta, puesto que en ocasiones dichos caminos tienen un cruce a 90° y la geometría diferencial de los robots no permite un desplazamiento en esas condiciones para evitar estas situaciones se realiza una interpolación para los puntos de la ruta buscando redondear los tramos rectos y permitir de esa forma el tránsito para la locomoción de tipo diferencial.

Para validar de forma experimental los algoritmos de planificación de trayectorias de campos potenciales se debe seleccionar de forma correcta las dimensiones de la cuadricula emulada y el tamaño del objeto más grande en el escenario esto para disminuir la aparición de mínimos locales.

Se debe tener en cuenta la geografía del espacio donde el robot se desplaza, asegurando de ser una superficie de buen acople con las llantas de los robots para evitar deslizamientos, puesto que esto podría incurrir en estimaciones erradas en la posición mediante odometría.

Como oportunidad de mejora se propone adaptar a los robots con sensores tipo lidar que den un mayor rango espacial de detección de objetos, así como un menor tiempo de respuesta lo cual mejora el control simultaneo de múltiples agentes.

9. BIBLIOGRAFIA

[1] Badesa F., Díez S., “Métodos de control basados en campos potenciales y de fuerza para robótica de rehabilitación”. Universidad Miguel Hernández de Elche. España. 2015

[2] Tibaduiza D, “Planeamiento de trayectorias de un robot movil”. Universidad Industrial de Santander. Colombia 2006

[3] Espitia E., Sofrony J. “Algoritmo para la planeación de trayectorias de robots móviles empleando enjambres de partículas brownianas”. Universidad Distrital Francisco José de Caldas. Colombia. 2011.

[4] Morales E. “Localización y mapeo simultaneo por robots móviles con ruedas en mapas y matrices dispersas”. Universidad de Sao Paulo 2018.

[5] D. Fox, W. Burgard, and S. Thrun. Controlling synchro-drive robots with the dynamic window approach to collision avoidance. In Proc. of thelEEE/RSJ International Conference on IntelligentRobots and Systems, 1996

[6] J. Cornejo, J. Magallanes, E. Denegri and R. Canahuire, "Trajectory Tracking Control of a Differential Wheeled Mobile Robot: a Polar Coordinates Control and LQR Comparison," 2018 IEEE XXV International Conference on Electronics, Electrical Engineering and Computing (INTERCON), Lima, 2018

[7] J. Sparbert and E. Hofer, “Numerical Path Optimization for Path Planning With Cell Decomposition Methods,” 2001.

[8] G. Azkune. “Robótica: Navegación autónoma” obtenido de: https://cuentos-cuanticos.com/2012/05/31/buscando-caminos/. Consultado: 2021

[9] León D., “Diseño y construcción de una plataforma robótica para el control de formación y distribución de tareas” Universidad Autónoma de Bucaramanga. Colombia. 2018.

[10] J. Borenstein, H. R. Everett, and L. Feng, “Where am I? Sensors and

methods for mobile robot positioning,” Univ. Michigan, vol. 119, p. 120, 1996.

[11] N.A, “Capitulo 8: Grafos,” 2003.

[12] Badesa F., Díez S., “Métodos de control basados en campos potenciales y de fuerza para robótica de rehabilitación”. Universidad Miguel Hernández de Elche. España. 2015

[13] Bin Azhar, Bilial M. “Empirical evaluation of formation control scheme based on artificial potential fields” PAF-KIET 2018.

[14] T. Abbas, M. Arif, and W. Ahmed, “Measurement and correction of systematic odometry errors caused by kinematics imperfections in mobile robots,” 2006 SICE-ICASE Int. Jt. Conf., vol. 12, no. 6, pp. 2073–2078, 2006.

[15] S. M. Lavalle, “Planning Algorithms,” Cambridge, 2006. 78

[16] “Planificación de Caminos Mediante Grafos de Visibilidad.,”

[17] L. E. Kavraki et al., “Probabilistic roadmaps for path planning in highdimensionalconfiguration spaces,” Robot. Autom. IEEE Trans., vol. 12, 1996.

[18] F. R. Mendoza, “Geometría Computacional.”

[19] Nguyet Tran, Duy-Tung Nguyen, Duc-Lung Vu, and Nguyen-Vu Truong, “Global path planning for autonomous robots using modified visibility-graph,” 2013 Int. Conf. Control. Autom. Inf. Sci., 2013.

[20] D. E. T. Con et al., “PLANIFICACIÓN DE TRAYECTORIAS CON EL ALGORITMO RRT. APLICACIÓN A ROBOTS NO HOLÓNOMOS,” vol. 3, 2006.

[21] E. E. N. M. Luiz S. Martins-Filho Ronilson Rocha, Romuel F. Machado, Laos A. Hirano, “Kinematic Control of Mobile Robots To Produce Chaotic Trajectories,” ABCM Symp. Ser. Mechatronics, vol. 2, , 2006.

[22] V. F. Muñoz Martinez, “Planificación de Trayectorias para Robots Móviles,” 1995.

[23] P. Quintero, “MODELO CINEMATICO DINAMICO DEL MINI ROBÓT MÓVIL RICIMAF,” 2012.

[24] R. Illah, Autonomous Mobile Robots. .

[25] S. G. Tzafestas, “Mobile Robot Kinematics,” Introd. to Mob. Robot Control, 2014.

[26] DEWESoft. PID Control [En línea] Disponible en: <https://www.dewesoft.com/pro/course/pid-control-53>

[27] MAZZONE, Virginia. Controladores PID [En línea] Disponible en: http://www.eng.newcastle.edu.au/~jhb519/teaching/caut1/Apuntes/PID.pdf

[28] RIVEROS, Adriana; SOLAQUE, Leonardo. Formación de robots móviles

mediante el uso de controladores. En: Ing. USBMed. Julio-diciembre, 2013. Vol. 4, n.º 2, p. 63. ISBN: 2027-5846.

[29] TDROBÓTICA. Micromotor 100:1 con eje extendido / 2.2 kg-cm / 320 rpm [En línea] Disponible en: <http://tdrobotica.co/micromotor-1001-con-ejeextendido22- kg-cm320-rpm/387.html>

[30] TDROBÓTICA. Encoder magnético [En línea] Disponible en: <http://tdrobotica.co/kit-encoder-magnetico-para-micromotor-con-ejeextendido/114.html>

[31] TDROBÓTICA. Sensor ultrasónico (HC-SR04) [En línea] Disponible en: <https://www.ardobot.com/productos/sensores/distancia-presencia-huellas-ycorriente/sensor-ultrasonido-hc-sr04.html>

[32] TDROBÓTICA. Arduino Mega 2560 R3 [En línea] Disponible en: <https://www.ardobot.com/arduino-mega-2560-r3.html>

[33] TDROBÓTICA. Módulo L298N [En línea] Disponible en: <http://tdrobotica.co/modulo-driver-l298n/543.html>

10. ANEXOS

A continuación, se presentan anexos que se usaron para el desarrollo de este proyecto.

- **ANEXO A: Interfaz Gráfica para las pruebas de simulación y validación experimental.**

A continuación, se presenta la ventana principal de la interfaz GUI programada para las pruebas de simulación y experimentación:

Figura 93. Ventana principal de la GUI

En la ventana principal se puede navegar mediante 2 botones:

- Tareas de Navegación
- Tareas de exploración.

Al presionar el botón “Tareas de exploración” se inicia una ventana en la cual se permite elegir entre la simulación y la validación experimental.

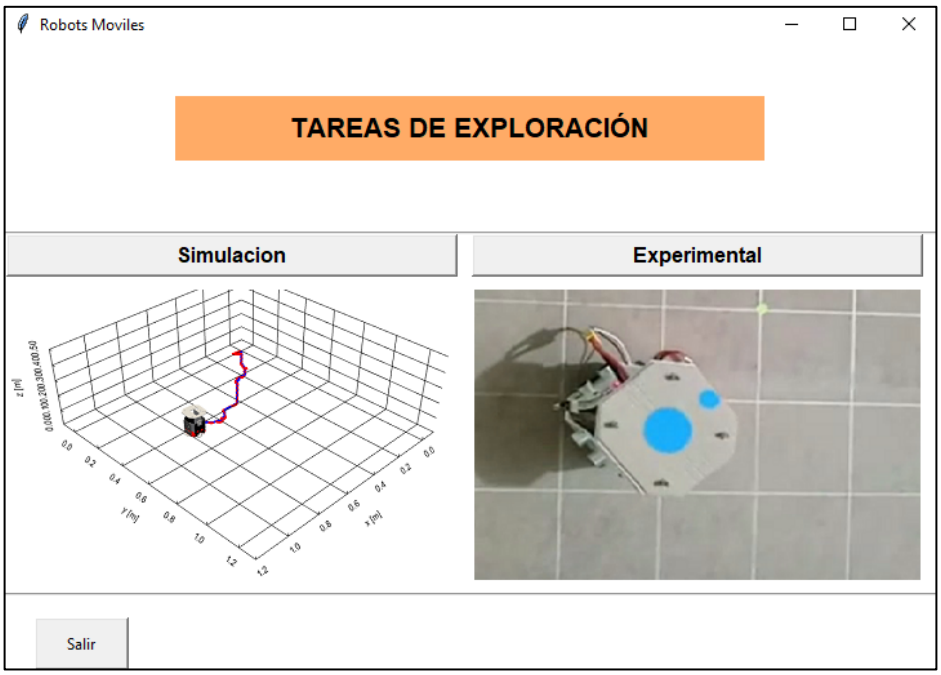


Figura 94. Ventana Tareas de exploración.

Se da clic en simulación para proceder a simular el algoritmo y se inicia la siguiente ventana en la cual se elige el número de robots, para este caso se seleccionan 2 robots.

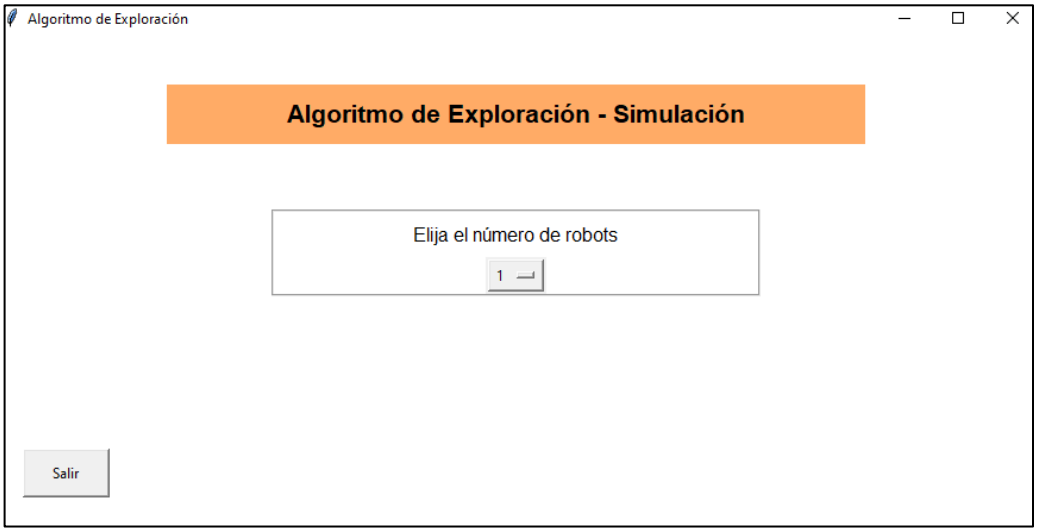


Figura 95. Seleccionar robots para simulación.

Se debe indicar las condiciones iniciales de los robots en la ventana que aparece a continuación.

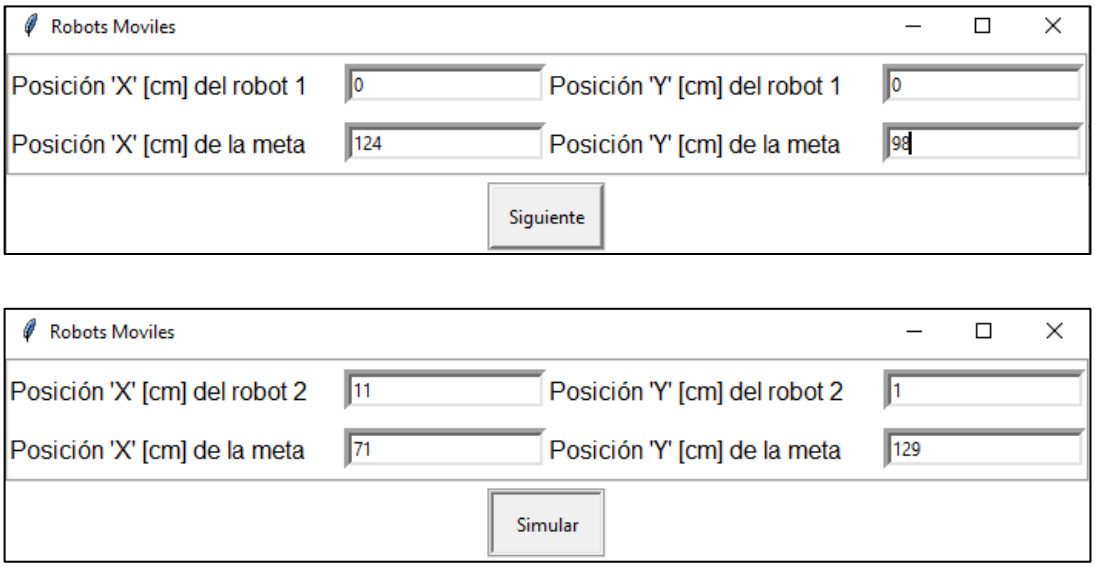


Figura 96. Condiciones iniciales de los robots.

Nótese que no se indica ubicación de los obstáculos o artefactos metálicos en la simulación puesto que serán generados de forma aleatoria y se ubicarán en el escenario para que el robot los detecte.

Simulación para las tareas de navegación.

Al presionar el botón “Tareas de navegación” se inicia una ventana en la cual se permite elegir uno de los dos algoritmos desarrollados (Campos potenciales o grafos de visibilidad), en los cuales se observa la gráfica de simulación característica de cada algoritmo

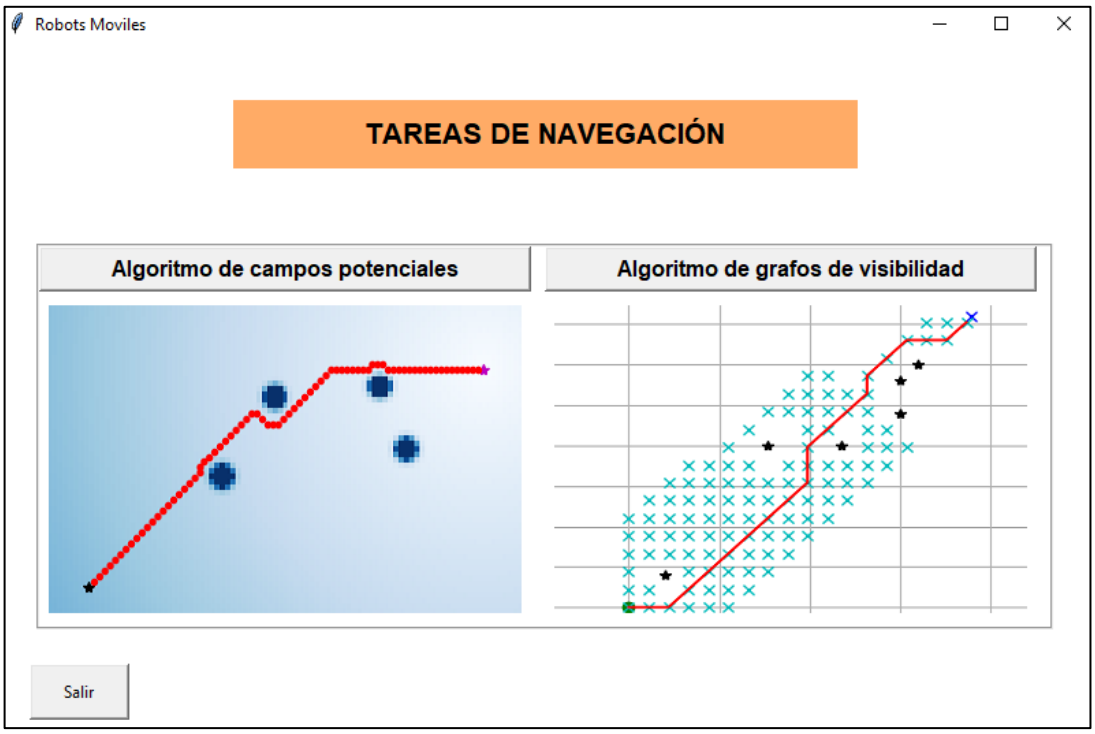


Figura 97. Ventana Tareas de simulación.

Al seleccionar la opción Campos potenciales se inicia otra ventana en la cual se puede seleccionar entre pruebas experimentales o simulaciones, para el caso de este capítulo se selecciona el botón “Simulación”.

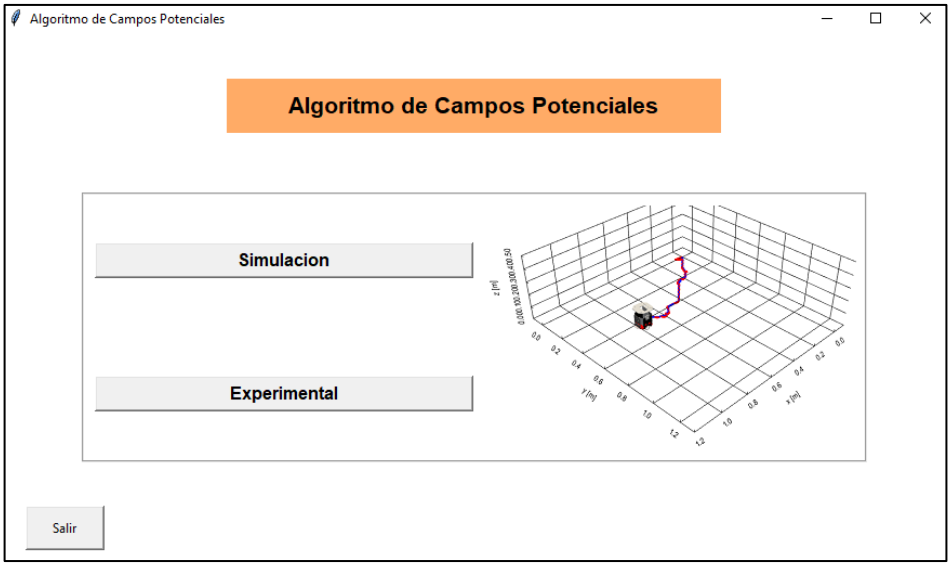


Figura 98. Ventana Tareas de simulación – Algoritmo campos potenciales.

Al presionar el botón simulación la interfaz solicita seleccionar el número de robots para la simulación, en un botón desplegable como se ve en la siguiente imagen.

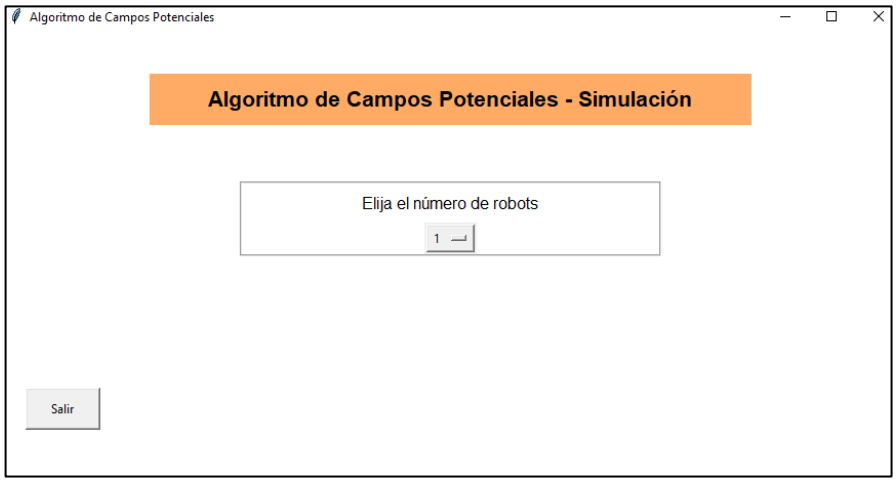
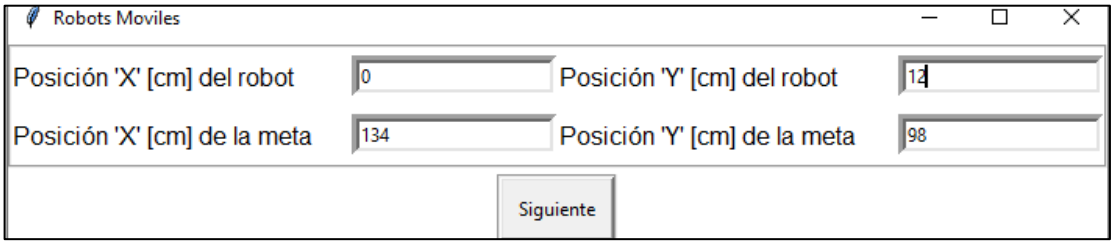
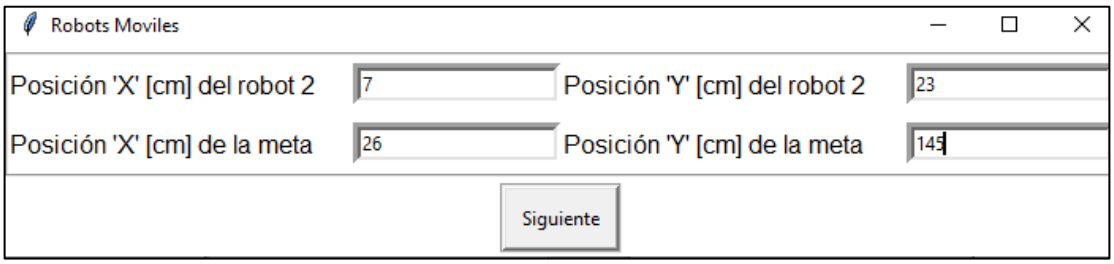


Figura 99. Seleccionar robots para simulación.

- **Pruebas de simulación Para 3 robots.**

Se selecciona 3 robots para el ejemplo de simulación, y la interfaz solicita ingresar las coordenadas de inicio y fin de cada robot, también se debe indicar la ubicación de los obstáculos separados por coma, para la ejecución del algoritmo de navegación.





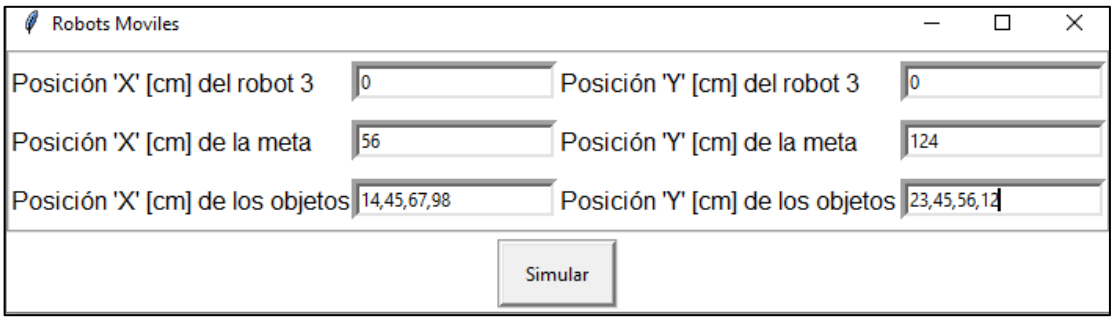


Figura 100. Condiciones iniciales de los robots y el escenario.

Presiona el botón simular e inician los cálculos y simulación de los algoritmos detallados en el capítulo 3.

- **Pruebas de simulación Tareas de Navegación – Algoritmo Grafos.**

Para simular el algoritmo de grafos de visibilidad se da clic en la opción grafos de visibilidad en la ventana “Tareas de navegación”. Y se llega a la siguiente ventana:

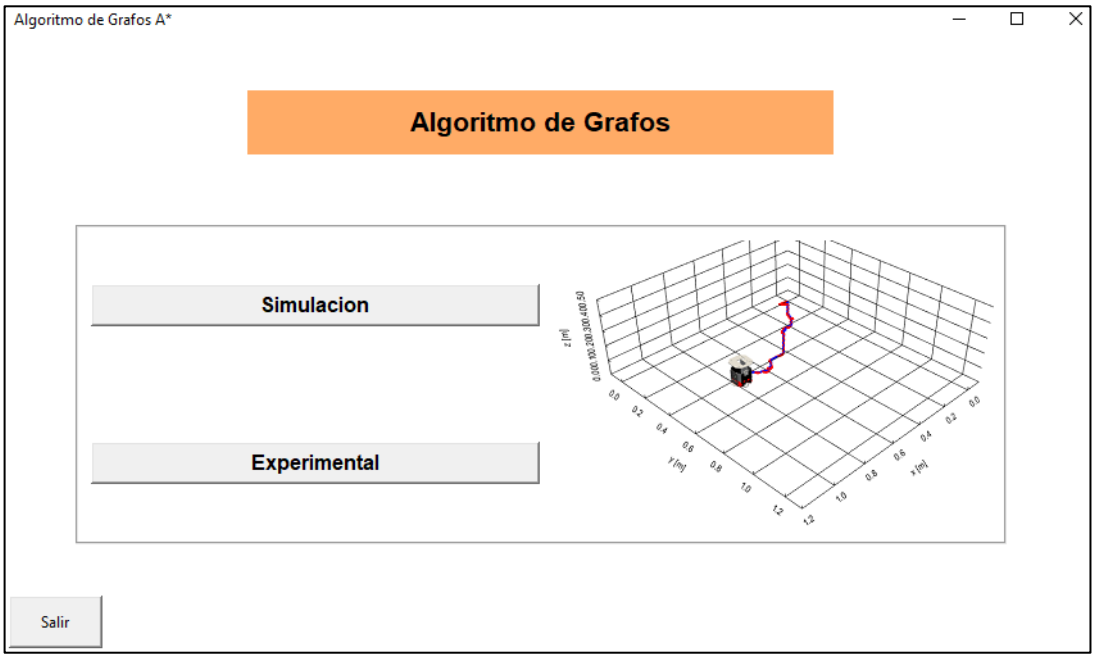


Figura 101. Ventana Tareas de simulación – Algoritmo de grafos

Al presionar en la opción simulación se habilita la siguiente ventana donde solicita el número de robots a simular, para este caso se selecciona 2.

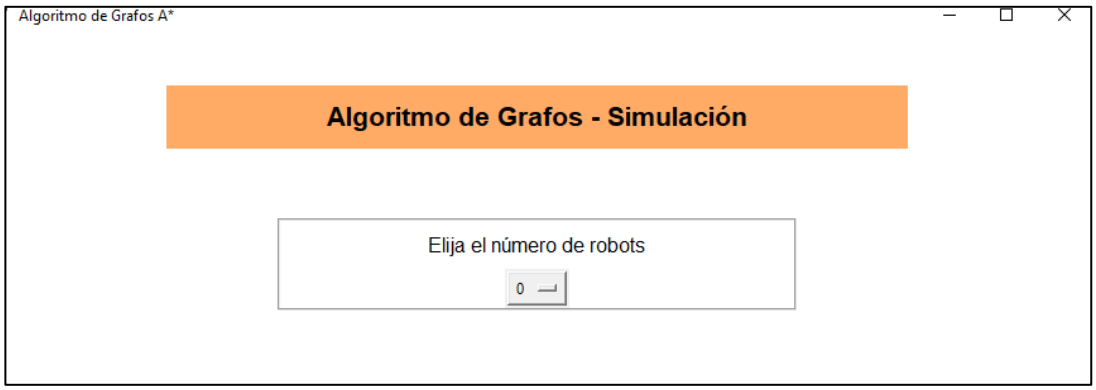


Figura 102. Ventana Tareas de simulación – Algoritmo de grafos

- **Pruebas de simulación para 2 robots – algoritmo grafos**

Para simular 2 robots el número 2 en la ventana de selección de robots, posteriormente la interfaz solicita ingresar las coordenadas de inicio y fin de cada robot, se debe tener en cuenta que las coordenadas de inicio y fin de los robots no deben coincidir y debe estar distanciados por lo menos 10 cm entre sí.

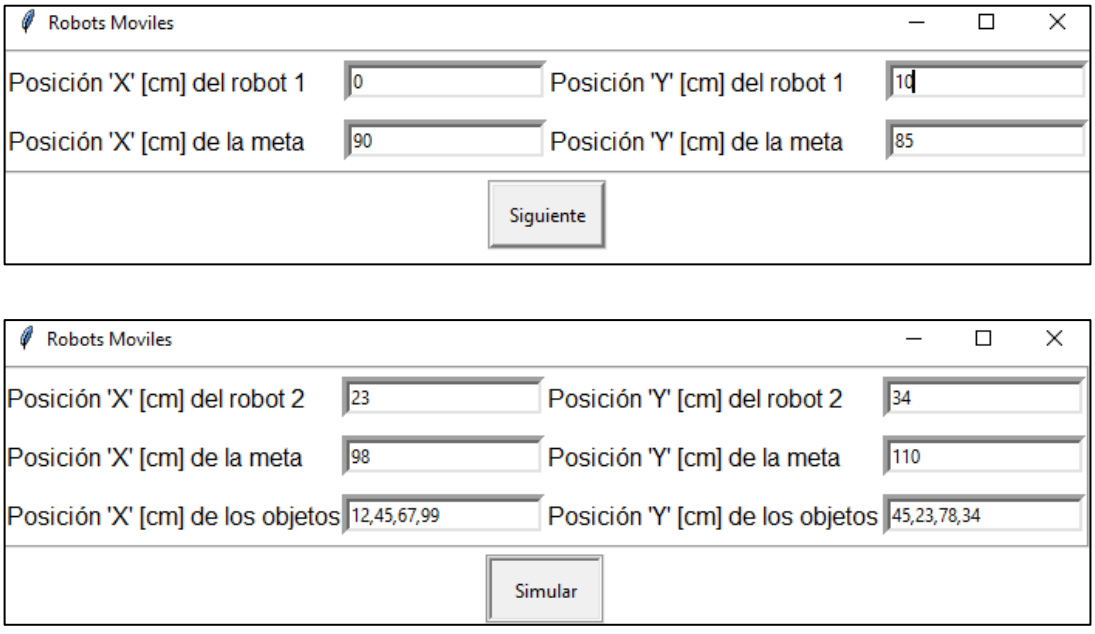


Figura 103. Condiciones iniciales de los robots y el escenario.

- **ANEXO B: Código implementado en el microcontrolador.**

```
#include "motorControl.h"
String inputString ="";
bool stringComplete =false;
const char separator =',';
const int dataLength =  2;
double data[dataLength];
unsigned long lastTime = 0;  // Tiempo anterior
unsigned long sampleTime = 100;  // Tiempo de muestreo
motorControl motorR(sampleTime);
motorControl motorL(sampleTime);
const int    C1R = 18; // Entrada de la señal A del encoder. --2,3
const int    C2R = 3; // Entrada de la señal B del encoder. --19,18
//  señales A y B reales
const int    C1L = 2; // Entrada de la señal A del encoder. --2,3
const int    C2L = 19; // Entrada de la señal B del encoder. --19,18
volatile int  nR    = 0;
volatile int antAR=0;
volatile int antBR=0;
volatile int actAR =0;
volatile int actBR =0;
```

```
volatile int  nL = 0;
volatile int antAL=0;
volatile int antBL=0;
volatile int actAL =0;
volatile int actBL =0;
double wRRef = 0.0;  // Velocidad angular de referencia en rad/s.
double wRRefpm = 0.0;
double wR = 0.0;
double wRrad = 0;
int cvR=0;
double wLRef = 0.0;  // Velocidad angular de referencia en rad/s.
double wLRefpm = 0.0;  // Velocidad angular de referencia en rad/s.
double wL = 0.0;
double wLrad =0.0;
int cvL=0;
double uRobot = 0;
double wRobot = 0;
double phi = 0;
const double R = 0.016; // radio llanta
const double d = 0.084; //distancia entre llantas
const double constValue = 5.21;
const byte in1=8;//10,13  (+) LLANTA DERECHA
const byte in2=9;//11,12
```

```
const byte in3=10;
const byte in4=11;  //  (+) LLANTA izquierda
void setup()
{
   Serial3.begin(115200);
   Serial.begin(115200);
motorR.setGains(0.31,0.09,0.02); //K, Ti, Td
motorL.setGains(0.31,0.09,0.02); //K, Ti, Td
  motorR.setCvLimits(255,0);
  motorR.setPvLimits(43,0);
  motorL.setCvLimits(255,0);
  motorL.setPvLimits(43,0);
  pinMode(C1R, INPUT);
  pinMode(C2R, INPUT);
  pinMode(C1L, INPUT);
  pinMode(C2L, INPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  digitalWrite(in1,false);
  digitalWrite(in2,false);

  attachInterrupt(digitalPinToInterrupt(C1R), encoderR, CHANGE);
```

```
  attachInterrupt(digitalPinToInterrupt(C2R), encoderR, CHANGE);
  attachInterrupt(digitalPinToInterrupt(C1L), encoderL, CHANGE);
  attachInterrupt(digitalPinToInterrupt(C2L), encoderL, CHANGE);
   lastTime = millis();
}
void loop() {

  if (stringComplete){
    for (int i=0; i<dataLength; i++)
    {
      int index =inputString.indexOf(separator);
      data[i]=inputString.substring(0,index).toFloat();
      inputString = inputString.substring(index+1);
      }
      velocidadMotor(data[0],data[1]); //velocidad lineal, velocidad angular
       inputString="";
      stringComplete =false;
       }
  if (millis() - lastTime >= sampleTime)
  {
     wR = (nR*constValue)/(millis()-lastTime);
     wL = (nL*constValue)/(millis()-lastTime);
      lastTime=millis();
       nR=0;
```

```
      nL=0;
      cvR =motorR.compute(wRRef,wR);
      cvL =motorL.compute(wLRef,wL);
        if(cvR>0)clockwise(in2,in1,cvR); else anticlockwise(in2,in1,abs(cvR));
        if(cvL>0)anticlockwise(in3,in4,cvL); else clockwise(in3,in4,abs(cvL));
      velocidadRobot(wR,wL);
     Serial3.println(uRobot);
     Serial3.println(wRobot);
   }
}
void serialEvent3(){
  while (Serial3.available()){
    char inChar = (char)Serial3.read();
    inputString +=inChar;
    if(inChar=='\n'){
      stringComplete = true;
     }
    }
  }
// Encoder precisión cuádruple.
void encoderR(void)
{

    antAR=actAR;
```

```
    antBR=actBR;
    actAR=digitalRead(C1R);
    actBR=digitalRead(C2R);
    if(antAR==0  && actAR==0 && antBR==0 && actBR==1)  nR++;
    if(antAR==0  && actAR==1 && antBR==1 && actBR==1)  nR++;
    if(antAR==1  && actAR==1 && antBR==1 && actBR==0)  nR++;
    if(antAR==1  && actAR==0 && antBR==0 && actBR==0)  nR++;
//
    if(antAR==1  && actAR==1 && antBR==0 && actBR==1)  nR--;
    if(antAR==1  && actAR==0 && antBR==1 && actBR==1)  nR--;
    if(antAR==0  && actAR==0 && antBR==1 && actBR==0)  nR--;
    if(antAR==0  && actAR==1 && antBR==0 && actBR==0)  nR--;
//
}
  void encoderL(void)
{
    antAL=actAL;
    antBL=actBL;
    actAL=digitalRead(C1L);
    actBL=digitalRead(C2L);
    if(antAL==0  && actAL==0 && antBL==0 && actBL==1)  nL++;
    if(antAL==0  && actAL==1 && antBL==1 && actBL==1)  nL++;
    if(antAL==1  && actAL==1 && antBL==1 && actBL==0)  nL++;
    if(antAL==1  && actAL==0 && antBL==0 && actBL==0)  nL++;
```

```
//
    if(antAL==1  && actAL==1 && antBL==0 && actBL==1)  nL--;
    if(antAL==1  && actAL==0 && antBL==1 && actBL==1)  nL--;
    if(antAL==0  && actAL==0 && antBL==1 && actBL==0)  nL--;
    if(antAL==0  && actAL==1 && antBL==0 && actBL==0)  nL--;
}
void anticlockwise(int pin2, int pin1, int pwm){
  analogWrite(pin1,pwm);
  digitalWrite(pin2,LOW);
    }
void clockwise(int pin2, int pin1, int pwm){
  digitalWrite(pin1,LOW);
    analogWrite(pin2,pwm);
    }
void velocidadMotor(double u, double w){
  wRRef = (u+(d*w/2))/R;
  wLRef = (u-(d*w/2))/R;
  }
void velocidadRobot(double w1, double w2){
  uRobot = (R*(w1+w2))/2;
   wRobot = (R*(w1-w2))/d;
 }
```