ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

SEGUIMIENTO DE TRAYECTORIAS MEDIANTE CUATRO TÉCNICAS DE CONTROL UTILIZANDO UNA PLATAFORMA ROBÓTICA PIONEER 3DX Y EL SISTEMA OPERATIVO ROBÓTICO ROS

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y CONTROL

LINDA JENNY CAPITO RUIZ

linda.capito@epn.edu.ec

PABLO ANDRÉS PROAÑO CHAMORRO

pablo.proano@epn.edu.ec

DIRECTOR: DR. JORGE ANDRÉS ROSALES ACOSTA

andres.rosales@epn.edu.ec

CODIRECTOR: DR. ÓSCAR EDUARDO CAMACHO QUINTERO

oscar.camacho@epn.edu.ec

Quito, septiembre 2015

DECLARACIÓN

Nosotros, Linda Jenny Capito Ruiz y Pablo Andrés Proaño Chamorro, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____ _____

Linda Jenny Capito Ruiz Pablo Andrés Proaño Chamorro

CERTIFICACIÓN

Certificamos que el presente trabajo fue desarrollado por Linda Jenny Capito Ruiz y Pablo Andrés Proaño Chamorro, bajo nuestra supervisión.

_____	_____
Dr. Jorge Andrés Rosales Acosta	Dr. Óscar Eduardo Camacho Quintero
DIRECTOR DEL PROYECTO	CO-DIRECTOR DEL PROYECTO

AGRADECIMIENTO

A mis padres y a mi hermano por siempre haberme brindado su apoyo incondicional y siempre estar a mi lado frente a las adversidades que me ha puesto la vida.

A mi abuelita Carmen por ser un ángel en mi vida y nunca dejar de brindarme su apoyo y cariño.

A mi tía Lourdes y a mi primo Santiago por haberme acogido en su casa como un hijo y hermano brindándome el calor de un hogar.

A mis maestros por haberme guiado y aconsejado en cada paso del aprendizaje, agradezco con especial afecto a los doctores Andrés Rosales, Óscar Camacho y Gustavo Scaglia, a quienes guardo un profundo cariño y respeto.

Al MSc. Alejandro Chacón y por su intermedio a la ESPE por habernos prestado el robot utilizado para realizar la parte experimental de este proyecto.

A mis grandes amigos Linda y Edwin por brindarme su apoyo en el camino del conocimiento recorrido hasta alcanzar esta meta.

A mi muñequita por siempre estar a mi lado y por los buenos momentos que pasamos juntos, Karlita.

Pablo

AGRADECIMIENTO

A mis padres Marina y Hernán por siempre haberme apoyado y alentado frente a cada reto que se ha presentado en mi vida, incluso en aquellos en que me he involucrado por decisión propia.

A mi hermano Vicente, por siempre inspirarme con su espíritu emprendedor a pensar y actuar diferente.

A mis abuelitos, tíos y primos, por siempre estar pendientes de mí y apoyarme en cada paso.

A todos mis grandes profesores, de quienes he podido aprender lecciones valiosas útiles para mi vida profesional y personal. Además quisiera agradecer profundamente a los doctores Andrés Rosales, Óscar Camacho y Gustavo Scaglia por su invaluable colaboración y acertada guía para la consecución de este proyecto.

Al MSc. Alejandro Chacón de la Universidad de las Fuerzas Armadas – ESPE por habernos prestado el robot Pioneer 3DX con el que se realizaron las pruebas experimentales del proyecto.

A todos mis amigos de la universidad, gracias por hacer de este camino una experiencia inolvidable y llena de buenos momentos. Especialmente quiero agradecer a Pablo, Edwin y Karlita por haber compartido conmigo durante la realización de este proyecto, chicos los quiero (normal).

A Jordan por todos los buenos momentos durante nuestra vida universitaria.

Linda

DEDICATORIA

Dedicamos este trabajo a las personas ávidas de conocimiento que encuentren en este proyecto una aplicación útil para su formación y esperamos así contribuir al desarrollo de la sociedad científica ecuatoriana.

Pablo y Linda

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

Dentro del ámbito de la robótica móvil el seguimiento de trayectorias es un área activa de investigación para aplicaciones académicas e industriales; donde se ha encontrado que los robots móviles son especialmente apropiados en aplicaciones donde se requiera una planeación de movimiento flexible.

La planeación de trayectorias permite generar un camino sobre el cual el robot debe navegar como una función del tiempo sobre un área determinada. Además, el reto actual es diseñar controladores que puedan ser implementados fácilmente y puedan reaccionar apropiadamente a diferentes tipos de perturbaciones externas.

Respecto a lo anterior, la finalidad de este proyecto es la de desarrollar, implementar y evaluar el desempeño de cuatro controladores, dos basados en álgebra lineal (aproximación de Euler y aproximación trapezoidal) y dos basados en PID (clásico y robustecido gracias al controlador por modo deslizante), haciendo uso del robot Pioneer 3DX.

Para todos los controladores se establece una comparación de desempeño y se analizan tanto sus ventajas como desventajas al someter al robot a diferentes velocidades, trayectorias, obstáculos y perturbaciones. Estos algoritmos se han desarrollado en función de los datos provistos por los sensores con los que cuenta el robot, sin el uso de dispositivos externos.

Adicionalmente se ha realizado la implementación de una interfaz gráfica que permite monitorear varios aspectos del proyecto mediante el uso de un computador comunicado con el robot.

Para la implementación de los algoritmos se ha hecho uso del sistema operativo robótico ROS y del programa QtCreator para la implementación de la interfaz gráfica, pudiendo probarse el proyecto desde la computadora que se conecta al robot o a distancia desde un punto de trabajo.

PRESENTACIÓN

En el presente trabajo se desarrollan e implementan cuatro controladores basados en dos métodos de control diferentes (métodos numéricos y PID), utilizados para evaluar el seguimiento de trayectorias, evasión de obstáculos y reacción ante perturbaciones.

En el Capítulo 1 se presenta un resumen de la teoría a utilizarse durante el desarrollo de este proyecto, en donde se empieza con una introducción a la robótica móvil y después se analiza a la plataforma robótica utilizada, se describen brevemente los métodos de control a ser implementados así como el software usado.

En el Capítulo 2 se tiene el desarrollo matemático de los algoritmos de generación de trayectorias, los algoritmos de control (dos basados en álgebra lineal y dos basados en el clásico PID) y la calibración de estos últimos.

En el Capítulo 3 se explica la implementación del proyecto en sí, describiendo de manera desglosada el uso de las herramientas de software y el flujo de información en cada etapa del control.

En el Capítulo 4 se realiza una descripción de los experimentos realizados, y luego se detallan los resultados obtenidos en gráficas que relacionan los cuatro controladores a la vez. Además se realiza el cálculo de un índice de desempeño que sirve como medida de comparación de los controles.

Finalmente, en el Capítulo 5 se presentan las conclusiones y recomendaciones obtenidas a lo largo del desarrollo de este trabajo.

CAPÍTULO 1

MARCO TEÓRICO

La finalidad de este capítulo es proveer la información teórica suficiente para entender el resto del contenido de este documento. Para ello se realiza una breve introducción al alcance del proyecto antes de iniciar con definiciones básicas de robótica, robots móviles y el modelo específico de la plataforma usada. Después se explican los fundamentos básicos de los métodos de control implementados así como los conceptos de funcionamiento del software empleado.

1.1 INTRODUCCIÓN

En el proyecto que se describe en este documento se realizó la comparación de desempeño de cuatro estrategias de control diferentes implementadas en el robot móvil Pioneer 3DX para realizar seguimiento de trayectorias.

Para ello en primer lugar se desarrollaron dos controladores por métodos numéricos utilizando conceptos de álgebra lineal y el modelo cinemático con restricción no holonómica mejorada del robot. El primer control utiliza la aproximación de Euler y el segundo la aproximación de Runge-Kutta o trapezoidal. Después se obtuvo el controlador PID y del controlador por modo deslizante, que utilizan un modelo aproximado del robot.

Utilizando estos cuatro controladores que usan modelos diferentes del mismo robot se realizaron pruebas a diferentes velocidades y ante diferentes trayectorias previamente generadas. Además se implementó un método de evasión de obstáculos basado en el método de impedancia y se observó la reacción ante el ingreso de una perturbación. Cada prueba presentada en este documento consta de su descripción, resultados y comentarios respectivos.

Adicionalmente se explica el sistema operativo robótico ROS usado para implementar los controladores y el programa QtCreator para la interfaz gráfica.

Conociendo lo que contendrá este texto, se verán ahora conceptos generales de robótica.

1.2 ROBÓTICA

La palabra “robot” tiene muchas acepciones, desde los ficticios R2D2 y C3PO de Star Wars hasta el Rover Sojourner que exploró la superficie marciana como parte de la misión Mars Pathfinder; pero para este proyecto se conserva la siguiente definición:

> “Máquina automática programable capaz de realizar determinadas operaciones de manera autónoma y sustituir a los seres humanos en algunas tareas, en especial las pesadas, repetitivas o peligrosas; puede estar dotada de sensores, que le permiten adaptarse a nuevas situaciones”
>
> - Diccionario Oxford (español), [13]

Este concepto refleja parte de lo que en este proyecto se logra con la plataforma robótica, pues se la programa para que realice el seguimiento de trayectorias en forma autónoma después de otorgarle una condición inicial.

Al ser la definición de robot un concepto tan amplio, da paso a una gran categorización de plataformas robóticas de acuerdo a sus capacidades y aplicaciones, encontrándose hoy en día robots móviles, estáticos, manipuladores, teleoperados, voladores, entre otros. [40]

Una categoría especialmente importante para el desarrollo de este proyecto de titulación son los robots móviles de tracción diferencial, pues se utiliza uno de ellos para realizar las pruebas experimentales de los controladores implementados.

1.3 ROBOT MÓVIL DE TRACCIÓN DIFERENCIAL NO HOLONÓMICO

Los robots móviles son aquellos que están provistos de un sistema de locomoción determinado, como patas, orugas o ruedas que permiten el desplazamiento en el plano horizontal [37]. Dentro de la categoría de robots móviles se encuentran los

robots de tracción diferencial. Estos consisten de dos ruedas motrices y una o más ruedas locas como indica la Figura 1.1.

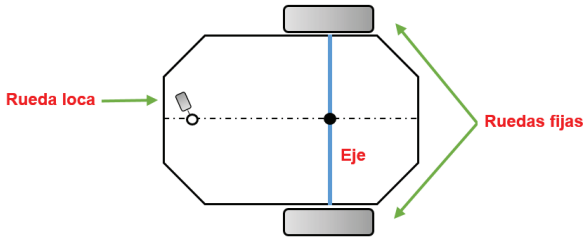


Figura 1.1. Representación de un robot móvil de tracción diferencial

Las restricciones no holonómicas se dan cuando la plataforma no puede moverse en direcciones arbitrarias debido a su configuración espacial.

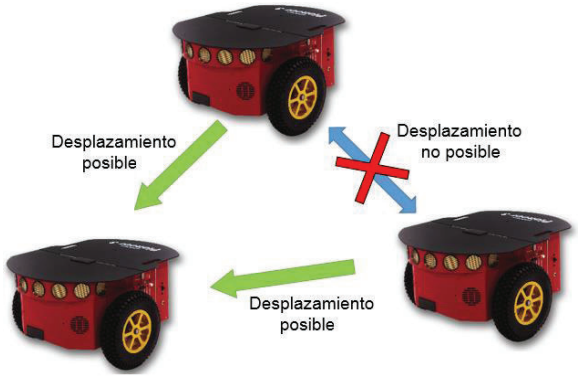


Figura 1.2. Representación del movimiento de un robot móvil de tracción diferencial no holonómicas, [41]

El concepto de restricción no holonómica se entiende mejor analizando el caso de la Figura 1.2 donde el robot móvil, al querer desplazarse, solo puede hacerlo hacia adelante (o atrás) y no directamente hacia los lados, como un automóvil no puede deslizarse hacia un lado para estacionarse en paralelo. [16],[18]

1.4 PLATAFORMA MÓVIL PIONEER 3DX

El robot Pioneer 3DX, de la empresa MobileRobots, es un robot de tracción diferencial no holonómico de dos ruedas con dos motores que fue utilizado para implementar los controladores y efectuar las pruebas respectivas.

A continuación se presenta el modelo cinemático y aproximado utilizado para el desarrollo del proyecto.

1.4.1 MODELO CINEMÁTICO DEL ROBOT PIONEER 3DX CON RESTRICCIÓN NO HOLONÓMICA MEJORADA

Antes de revisar el modelo cinemático del robot, vale acotar la diferencia entre restricción no holonómica no mejorada y mejorada.

Como se vio en la sección 1.3, una restricción no holonómica se produce cuando el robot tiene restricción en su movimiento lateral, y esto se da porque el punto de interés del robot (punto que va a describir el movimiento) se encuentra en el centro del eje de las ruedas, lo que limita su movimiento. Para mejorar esta condición lo que se hace es desplazar el punto de interés del robot una distancia a del centro del eje de las ruedas. A esto se denomina restricción no holonómica mejorada.

Un modelo del robot móvil es el que se presenta en la Figura 1.3. La posición actual del robot está definida por el punto (x, y) que se ubica a una distancia a del centro del eje que une las ruedas B, y que en este proyecto se lo hace coincidir con el centro de gravedad G del robot móvil.

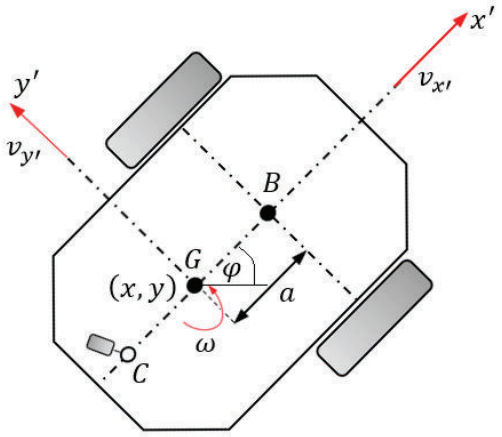


Figura 1.3. Modelo del robot móvil, [3]

En la Figura 1.3 si se considera que el punto de interés (x, y) se encuentra originalmente en B se tiene un robot con restricción no holonómica no mejorada, pues este punto está en el centro del eje de las ruedas.

Ahora para el análisis y desarrollo de los controladores se adelanta el punto de interés, haciendo que coincida con el centro de gravedad G del robot móvil, que se encuentra a una distancia a = 0,2 (m) [2]. Así se trabaja con un robot con restricción no holonómica mejorada. El modelo cinemático de este último caso está representado por (1.1) (validado en [17]).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} cos(\varphi) & -a\,sin(\varphi) \\ sin(\varphi) & a\,cos(\varphi) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{1.1}$$

1.4.2 MODELO APROXIMADO DEL ROBOT PIONEER 3DX

Si bien los controladores Euler y Trapezoidal utilizan el modelo cinemático con restricción no holonómica mejorada del robot móvil, los controladores PID y modo deslizante usan un modelo aproximado de la plataforma para el cálculo de las acciones de control. Se propone encontrar este modelo aproximado del robot para que los controladores que van a ser desarrollados e implementados puedan ser generalizados a varias plataformas.

Para esto se considera al robot como una caja negra (Figura 1.4) a la que se ingresan las señales de control; y a la salida se obtienen las variables a controlar.

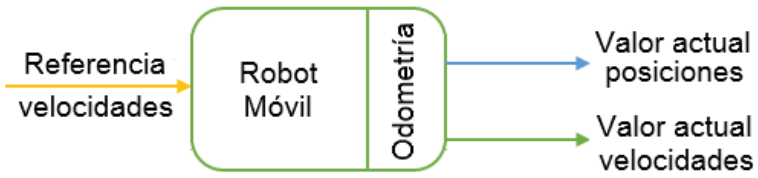


Figura 1.4. Representación del robot como caja negra

Al aplicar una señal paso a las velocidades lineal y angular del robot Pioneer 3DX se obtienen las respuestas de la Figura 1.5, donde se observa claramente que se posee un sistema de primer orden con retardo para ambas velocidades.

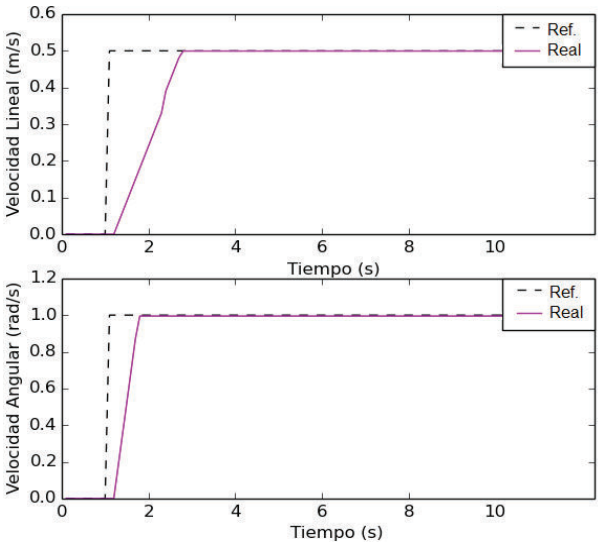


Figura 1.5. Respuestas de velocidad lineal y angular del robot Pioneer 3DX ante una entrada paso en lazo abierto y sin controlador

Destaca el hecho de que no existe error en estado estacionario ante una señal paso, no obstante se debe tener en cuenta que para esta plataforma existen PIDs internos que regulan la velocidad angular de cada motor y debido a eso el error en estado estable de la velocidad angular y lineal es cero [31].

Entonces el modelo reducido tanto para la velocidad lineal como para la velocidad angular del robot tiene la siguiente forma [27]:

$$\frac{F(s)}{U(s)} = \frac{Ke^{-t_0 s}}{\tau s + 1} \tag{1.2}$$

Donde:

$F(s)$: salida del sistema (velocidades lineal y angular)

$U(s)$: entrada al sistema (velocidades lineal y angular de referencia)

K : ganancia en estado estable del sistema

t_0 : tiempo muerto o retardo del sistema

τ : constante de tiempo del sistema

1.4.2.1 Aproximación del sistema

Puesto que se utiliza el modelo aproximado para el desarrollo de los controladores por modo deslizante, y éste como tal no posee un procedimiento general para el manejo del tiempo muerto (sección 1.5.5.3), se hace necesario realizar una nueva aproximación que permita trabajar de una manera más sencilla en el desarrollo del controlador. Camacho [29] propone el uso de una aproximación de series de Taylor para el efecto (1.3).

$$e^{-t_0 s} \cong \frac{1}{t_0 s + 1} \tag{1.3}$$

La ecuación (1.3) permite aproximar el retardo sin crear inestabilidad y sin provocar una pérdida excesiva de información. Se pueden proponer otras aproximaciones a este término, sin embargo en [28] se determinó que la aproximación por series de Taylor no provoca oscilaciones o controladores inestables a diferencia de otras aproximaciones como Padé.

Una vez hecha esta consideración, se obtiene la siguiente función de transferencia resultante (1.4) al reemplazar (1.3) en (1.2).

$$\frac{F(s)}{U(s)} = \frac{K}{(\tau s + 1)(t_0 s + 1)} \tag{1.4}$$

Se hallan los parámetros τ, t_0 y K correspondientes a la plataforma Pioneer 3DX utilizando la salida real presentada en la Figura 1.5, y luego se ingresan en MatLab los valores correspondientes a la salida real del robot ante la entrada paso y la salida del modelo aproximado descrito por (1.4). El resultado obtenido se halla en la Figura 1.6 donde se observa que el modelo aproximado presenta un comportamiento similar a la salida real del robot, por lo que se concluye que el modelo es válido y puede ser usado para la implementación de los controladores.

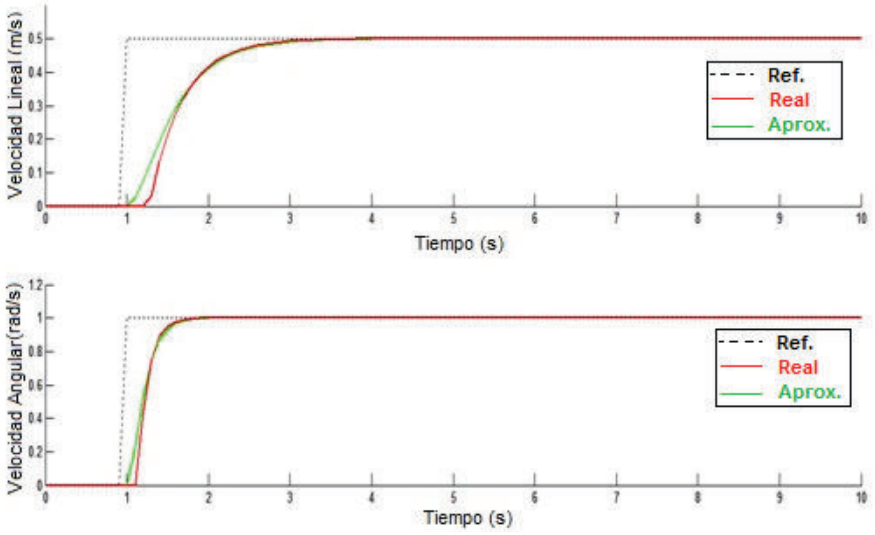


Figura 1.6. Respuestas a una entrada paso, modelo real con retardo (rojo) y modelo aproximado por series de Taylor (verde)

1.5 MÉTODOS DE CONTROL

1.5.1.1 Control de robots

El propósito de los controladores que se presentan a continuación es el cálculo de las velocidades lineal y angular con el objetivo de seguir una trayectoria de referencia. Como se observa en la Figura 1.7 el robot móvil, cuya posición actual en t_m es (x_m, y_m) con un ángulo de orientación φ_m, debe pasar del estado actual al estado deseado de referencia en t_m que es (x_{ref_m}, y_{ref_m}), y luego a la referencia en t_{m+1} que es $(x_{ref_{m+1}}, y_{ref_{m+1}})$.

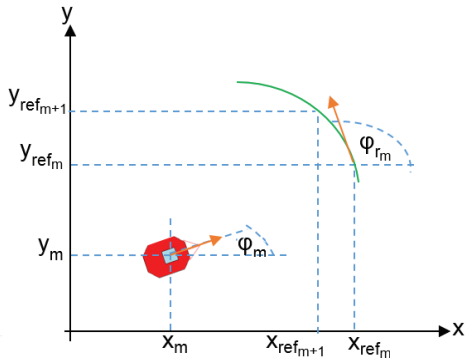


Figura 1.7. Representación del estado actual y estado deseado del robot móvil

Para esto se plantean algunas técnicas y dependiendo de la metodología escogida se usa el modelo exacto de la plataforma robótica o una aproximación del modelo como se vio en la sección 1.4.

1.5.1.2 Diferencia entre camino y trayectoria

Las palabras camino y trayectoria son confundidas a menudo, y se utilizan informalmente como sinónimos. No obstante, para la aplicación de un método de control es necesario establecer la diferencia entre ellas [41]:

- Camino: es un conjunto de puntos ordenados localizados en el espacio que el robot debe seguir y describe una descripción puramente geométrica del movimiento.
- Trayectoria: es un camino parametrizado en el tiempo.

Para el desarrollo de este proyecto se generan trayectorias y del seguimiento de las mismas se encarga el controlador seleccionado, que puede utilizar uno de los métodos propuestos a continuación.

1.5.2 MÉTODOS NUMÉRICOS CON APROXIMACIÓN DE EULER

La aproximación de Euler consiste en calcular el siguiente estado de la una variable mediante una aproximación lineal [15].

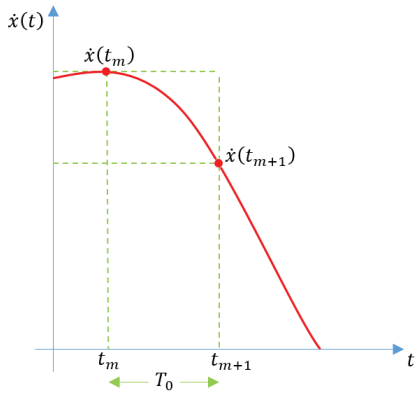


Figura 1.8. Aproximación de Euler

En la Figura 1.8 se observa que para calcular el estado siguiente de la variable $x(t)$ se debe integrar su derivada $\dot{x}(t)$ desde t_m hasta t_{m+1}. La aproximación de Euler permite reemplazar la integral por una diferencia finita hacia adelante, como se presenta en (1.5).

$$\Delta x(t) = \int_{t_m}^{t_{m+1}} \dot{x}(t)dx \approx \dot{x}(t_m)\,(t_{m+1} - t_m) \tag{1.5}$$

Gracias a estas consideraciones, se consigue la siguiente relación entre el estado actual y el siguiente.

$$X_{m+1} = X_m + T_0 \dot{X_m} \tag{1.6}$$

Esta relación sirve como base para el desarrollo del algoritmo explicado en el capítulo dos (sección 2.2).

1.5.3 MÉTODOS NUMÉRICOS CON APROXIMACIÓN TRAPEZOIDAL

La aproximación trapezoidal es una variación de la aproximación clásica, que toma el área trapezoidal bajo una curva (Figura 1.9), en lugar del área rectangular considerada para el caso anterior.

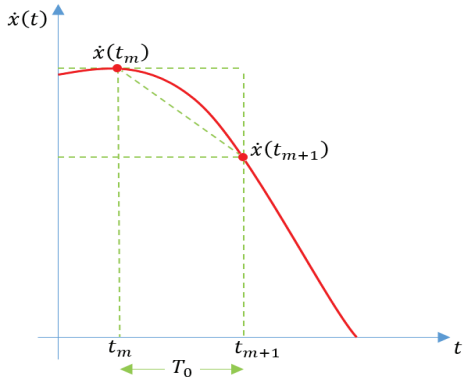


Figura 1.9. Aproximación trapezoidal

En la ecuación (1.7) se observa la deducción de la ecuación que describe a la nueva función $\dot{x}(t)$ y el resultado de la integral nuevamente entre t_{m+1} y t_m.

$$\Delta x(t) = \int_{t_m}^{t_{m+1}} \dot{x}(t)dx \approx \frac{(t_{m+1} - t_m)}{2}(\dot{x}(t_m) + \dot{x}(t_{m+1})) \tag{1.7}$$

Entonces la relación entre el estado actual y el siguiente es:

$$X_{m+1} = X_m + \frac{T_0}{2}(\dot{X}_m + \dot{X}_{m+1}) \tag{1.8}$$

1.5.4 CONTROLADOR PID

El controlador PID es aquel que combina las tres acciones clásicas del control: acción proporcional, integral y derivativa para actuar sobre plantas de diversos tipos, cuyo modelo puede o no ser conocido [24]. El controlador toma el error (diferencia entre la referencia y la salida de la planta) y calcula la acción P, PI, PD o PID dependiendo de la calibración de sus parámetros.

1.5.4.1 Acción proporcional

La acción proporcional se aplica cuando el sistema realimentado presenta error en estado estable, el principio de esta acción es tomar el valor de error y multiplicarlo por una constante obteniendo así la salida del control.

Como se observa en la Figura 1.10, donde t_a es el instante en que se realiza la acción de control, un control solo proporcional no elimina el error en estado estable, porque el mismo necesita tener un error para poder multiplicarlo con la ganancia y así obtener la salida.

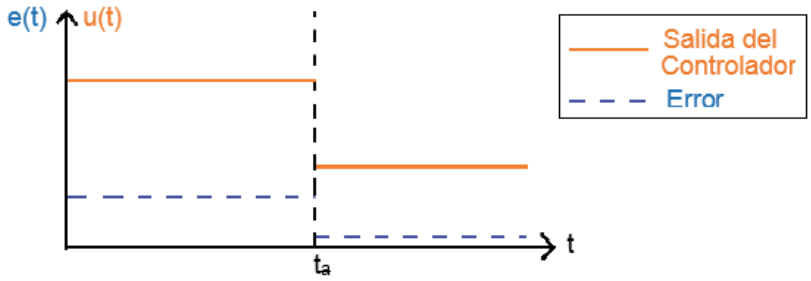


Figura 1.10. Acción proporcional

1.5.4.2 Acción diferencial

Un control con acción diferencial se aplica cuando el sistema es lento frente a los cambios en la referencia o en el error. Así, si existe una variación del error, la derivada del mismo actúa como un impulso obligando al sistema a alcanzar la referencia como se observa en la Figura 1.11, donde t_a es el instante en que se realiza la acción de control diferencial.

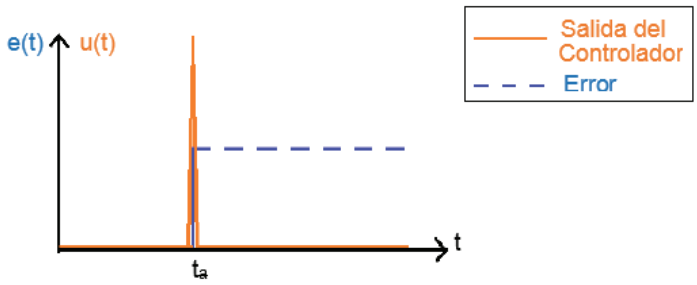


Figura 1.11. Acción diferencial

En la práctica no puede implementarse solo un control diferencial debido a que los actuadores no pueden generar señales impulso (de valor infinito), por ello siempre se aplica esta acción junto a una acción proporcional, que suaviza el efecto del impulso haciendo al controlador más estable.

1.5.4.3 Acción integral

Un control con acción integral se aplica cuando existe error en estado estable, pero a diferencia de la acción proporcional, logra eliminar este error.

Si el error en estado estable es nulo la salida del controlador se mantiene en el último valor que provocó que el error se haga cero, caso contrario la acción de control integra el error en el tiempo hasta alcanzar la referencia. Por ejemplo, si el error es constante, la salida del controlador es una rampa que provoca que el error disminuya hasta llegar a cero (Figura 1.12, donde t_a es el instante en que se alcanza la referencia al aplicar la acción de control integral).

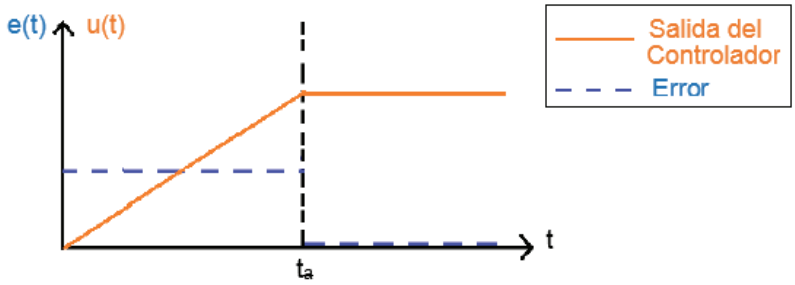


Figura 1.12. Acción integral

En la práctica se combina siempre la acción integral con la proporcional, pues un control únicamente integral provoca inestabilidad por su comportamiento en el plano de la frecuencia (agrega un polo en el origen).

1.5.5 MODO DESLIZANTE

El control por modo deslizante (SMC por sus siglas en inglés) se concibe debido a la necesidad de un método de control robusto, que pueda compensar errores, imprecisiones de modelado, y el cambio de los parámetros del sistema.

El control por modo deslizante es un método de Control por Estructura Variable (VSC por sus siglas en inglés) [42] que se puede definir como un método de control no lineal que altera el funcionamiento y la dinámica de un sistema mediante la aplicación de una señal de control que se diseña para que las trayectorias siempre se desplacen hacia una región adjunta diferentemente estructurada, por lo que la trayectoria final no existirá completamente dentro de una estructura de control, sino que se "deslizará" a lo largo de las fronteras de la misma, [23].

El movimiento del sistema cuando se desliza a lo largo de estas fronteras es lo que se denomina modo deslizante y la localización geométrica de las fronteras se llama superficie de deslizamiento [4].

Para este proyecto se implementa un controlador por modo deslizante de primer orden, mismo que se compone de dos leyes de control diferentes: función equivalente o continua y función switching o discontinua.

$$U(t) = U_C(t) + U_D(t) \tag{1.9}$$

Donde:

$U(t)$: salida del controlador por modo deslizante
$U_C(t)$: función equivalente o continua
$U_D(t)$: función switching o discontinua

1.5.5.1 Función equivalente (continua)

El problema de diseño consiste en seleccionar parámetros de cada estructura y definir una lógica de desplazamiento. Así que el primer paso es definir una superficie de deslizamiento $S(t)$, a lo largo de la cual el proceso pueda deslizarse hasta el valor final deseado, como se indica en la Figura 1.13.

Una vez que el sistema ha sido llevado a la superficie deslizante, la manera en que se comporta la dinámica del sistema en lazo cerrado se define únicamente por la superficie de deslizamiento.

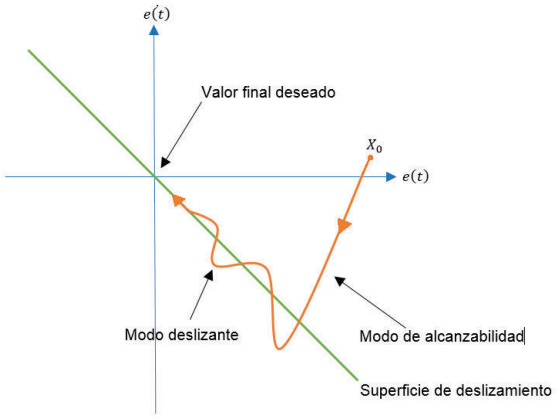


Figura 1.13. Representación gráfica de un SMC, [29]

La parte equivalente del controlador actua para seguir la dinámica impuesta por la ecuación que describe a la superficie deslizante. Para obtener esta sección del control se utiliza el modelo del proceso y la condición de deslizamiento.

Esta función debe cumplir además la condición de deslizabilidad, que asegura que la trayectoria sea tangente a la superficie de deslizamiento, es decir, se debe cumplir que $\dot{S}(t) = 0$.

1.5.5.2 Función discontinua (switching)

La parte discontinua del controlador fuerza al vector de error hacia una ley de decisión llamada superficie de deslizamiento durante el modo de alcanzabilidad (Figura 1.13). Esta parte del control está switcheando (cambiando) entre diferentes partes de la ecuación de la superficie de deslizamiento y permite alcanzar la superficie de deslizamiento con gran velocidad, por lo que su acción es transitoria. Inicialmente se puede representar como:

$$U_D(t) = K_D sign(S(t)) \tag{1.10}$$

Donde K_D es el parámetro de ajuste responsable por la velocidad con la que se alcanza la velocidad de deslizamiento, y $sign(S(t))$ es una función no lineal de la superficie de deslizamiento $S(t)$ (Figura 1.14).

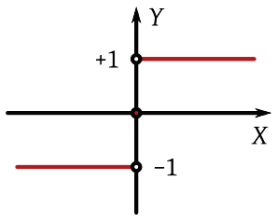


Figura 1.14. Función signo, [23]

Es importante que la función elegida cumpla la condición de alcanzabilidad, es decir, que sea lo suficientemente rápida para lograr llegar a la superficie de deslizamiento. Para ello debe satisfacer $S(t)\dot{S}(t) = 0$ [42].

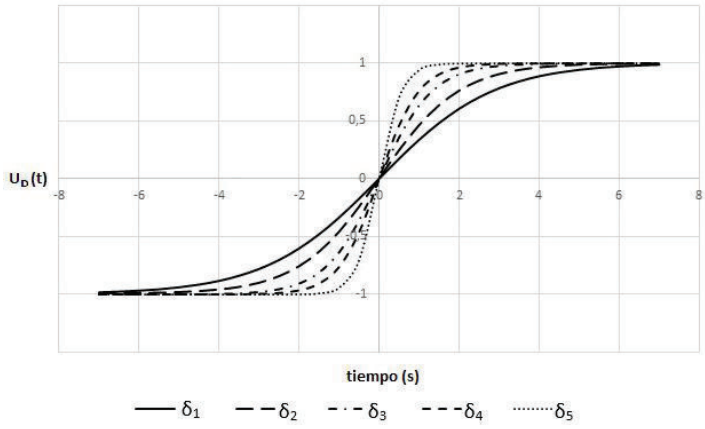
1.5.5.3 Problemas

- Debido a que la función es atraída por la superficie normalmente con una ley de control diseñada con la función signo, para llegar al modo deslizante se debería tener un cambio de estructura que ocurra a frecuencia infinita. No obstante, al no ser esto posible, se crea el fenómeno de "chattering" que provoca oscilaciones de alta frecuencia que los actuadores difícilmente toleran. Para evitar esto se utilizan funciones de suavizado (como la función (1.11) representada por la Figura 1.15) que permiten la aplicación de la ley de control, pero disminuyen la respuesta en cuanto a desempeño. Entonces se considera que se ha alcanzado un modo cuasideslizante o pseudodeslizante.

$$U_D(t) = K_D \frac{S(t)}{|S(t)| + \delta} \tag{1.11}$$

Donde:

$U_D(t)$: función switching o discontinua
$S(t)$: superficie deslizante
δ : parámetro de ajuste de la sigmoide
K_D : parámetro de ajuste responsable por la velocidad con la que se alcanza la velocidad de deslizamiento



$$\delta_1 < \delta_2 < \delta_3 < \delta_4 < \delta_5$$

Figura 1.15. Función sigmoide (aproximación de la función signo)

- No hay un procedimiento general para el manejo del tiempo muerto o retardo de tiempo. Cuando éste es muy grande, la respuesta del sistema que se está controlando se aleja de la referencia, pues normalmente el SMC no considera el retardo como tal (tan solo se realiza una aproximación de Taylor o Padé para el término del retardo).
- La sintonización de sus parámetros se realiza en base a prueba y error.
- Se requiere el modelo del proceso, lo que puede ser problemático debido a la presencia inevitable de errores de modelación. Además si se selecciona una superficie deslizante en función del modelo del sistema, el desempeño en lazo cerrado puede disminuir también debido a estos errores.

1.6 EVASIÓN DE OBSTÁCULOS

La evasión de obstáculos provoca un desvío del robot de su trayectoria a seguir, haciendo que la plataforma regrese a su trayectoria natural después de la evasión. El algoritmo presentado para este propósito se basa en el principio de evasión por método de impedancia, mismo al que se ha modificado para obtener un algoritmo más sencillo de implementar y adaptar al robot y controladores utilizados.

1.6.1 PRINCIPIO BÁSICO DE EVASIÓN POR MÉTODO DE IMPEDANCIA

Este método se basa en la generación de fuerzas ficticias que aparecen al presentarse un obstáculo en la trayectoria del robot y provocan que la plataforma

se desvíe de su trayectoria natural. Estas fuerzas son la fuerza de repulsión, la fuerza normal y una fuerza tangencial ([17], [33]), mismas que aparecen esquematizadas en la Figura 1.16.

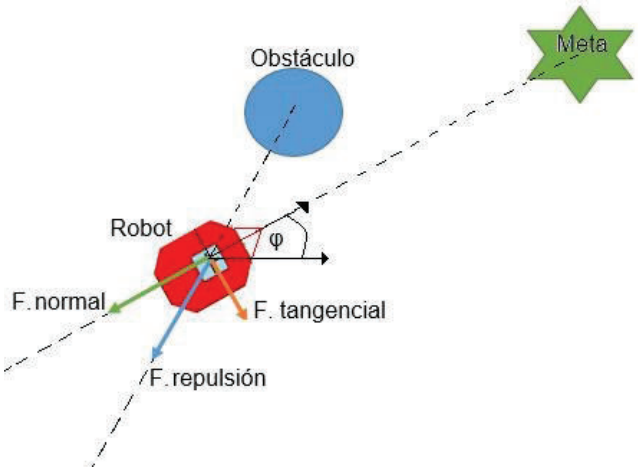


Figura 1.16. Representación de las fuerzas a considerar durante la detección de obstáculos, [33]

La acción de cada fuerza es la siguiente:

- La fuerza de repulsión es generada en función de la distancia actual entre el robot y el obstáculo, incrementando a medida que el robot se va acercando al obstáculo.
- La fuerza normal es generada en función de la posición actual del robot y la referencia que el mismo tiene que seguir, si la referencia se encuentra muy lejos o muy cerca del robot, el módulo de esta fuerza aumenta o disminuye.
- Finalmente la fuerza tangencial es la encargada de generar una nueva referencia que permita realizar la evasión de obstáculo, pues es una función de la distancia del robot al obstáculo y la distancia del robot a la referencia.

Como se puede observar en este método de evasión se considera que se posee una meta fija, pero en el presente trabajo se tiene una trayectoria que por definición se encuentra parametrizada en el tiempo, por lo que se hace necesario realizar una modificación a este método.

1.6.2 EVASIÓN POR MÉTODO DE IMPEDANCIA MODIFICADO

La variación del método consiste en tomar el concepto de fuerza tangencial y transformarlo en un ángulo de rotación, y tomar el principio de variación de magnitud de la fuerza de repulsión para calcular el valor del ángulo de desviación.

Así se tiene un ángulo de desviación que varía desde 0° hasta ±90°, y que cambia su módulo dependiendo de la distancia a la cual se encuentra el robot del obstáculo y la posición del obstáculo con respecto al robot.

Para realizar esto se introducen dos conceptos adicionales para realizar la evasión [32]:

- La distancia de inicio de evasión d_{iv} se define como la distancia a la cual el robot ya empieza a evadir el obstáculo.
- La distancia mínima de evasión $d_{mín}$ se define como la mínima distancia que puede existir entre el robot y el obstáculo para que al realizar la evasión la plataforma no colisione con el obstáculo. Esta distancia se utiliza en caso de que el obstáculo aparezca repentinamente o no haya sido detectado por los sensores anteriormente, y se establece considerando las dimensiones propias de la plataforma.

El funcionamiento es el siguiente:

- Si el robot se encuentra a una distancia mayor a la distancia de inicio de evasión d_{iv} el ángulo de rotación del robot es cero, por lo que el robot sigue de manera normal a la trayectoria.
- Si el robot se encuentra a una distancia menor a la distancia de inicio de evasión d_{iv} del obstáculo, el ángulo comienza a aumentar provocando que el robot inicie su rotación, con un ángulo positivo o negativo dependiendo de la posición del obstáculo.
- Si el robot traspasa la distancia mínima de evasión $d_{mín}$, se define que el ángulo de rotación debe ser 90° o -90°.

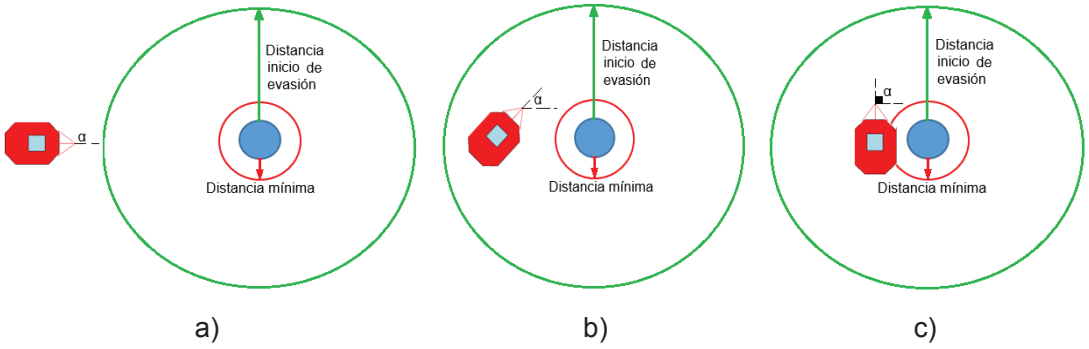


Figura 1.17. Ángulo de rotación del robot:
a) Antes de ingresar al rango delimitado por la distancia de inicio de evasión; b) Dentro de rango entre la distancia de inicio de evasión y la distancia mínima de evasión; c) Cuando se ha traspasado el rango delimitado por la distancia mínima de evasión

En los casos descritos por la Figura 1.17 se puede ver como varía el ángulo respecto a la distancia, pero esta variación no es lineal, sino que corresponde a la ecuación (1.12) :

$$\alpha = \begin{cases} 0° & ,si \quad d_{(R-O)} > d_{iv} \\ signo(\sigma) tan^{-1}\left(\dfrac{k}{\left|d_{(R-O)_1} - d_{mín}\right|}\right) & ,si \quad d_{(R-O)} \leq d_{iv} \end{cases} \tag{1.12}$$

Donde:

$d_{(R-O)}$: distancia entre el robot y el obstáculo

d_{iv} : distancia de inicio de evasión

$d_{mín}$: distancia mínima de evasión

α : ángulo de orientación del robot

σ : variable auxiliar que otorga el sentido de giro

k : constante de calibración de la rapidez con que el ángulo de giro tiende a ±90°

$d_{(R-O)_1}$: permite que el ángulo crítico sea siempre 90º, de acuerdo a (1.13)

$$d_{(R-O)_1} = \begin{cases} d_{(R-O)} & ,si \quad d_{(R-O)} \geq d_{mín} \\ d_{mín} & ,si \quad d_{(R-O)} < d_{mín} \end{cases} \tag{1.13}$$

Para la evasión de obstáculos se toma este método modificado y se adapta a la información que proveen los 8 sensores ultrasónicos con los que cuenta la plataforma robótica.

Debido a que únicamente se puede detectar obstáculos en el rango que permiten los sensores ultrasónicos, se han asignado distintos valores de $d_{(R-O)}$ y σ, de acuerdo a cada sensor ha detectado el obstáculo. Se considera lo siguiente:

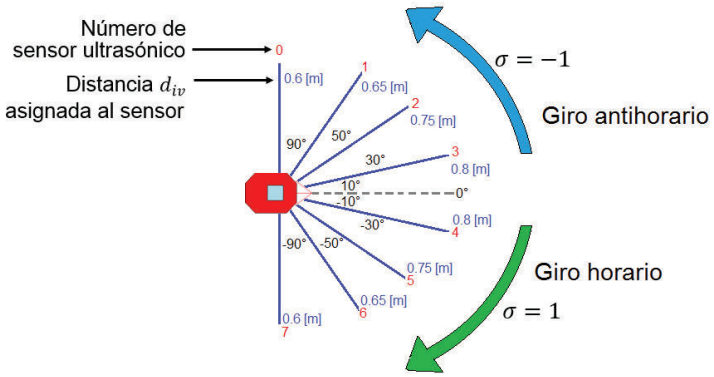


Figura 1.18. Valores de $d_{(R-O)}$ y σ respecto a cada sensor ultrasónico

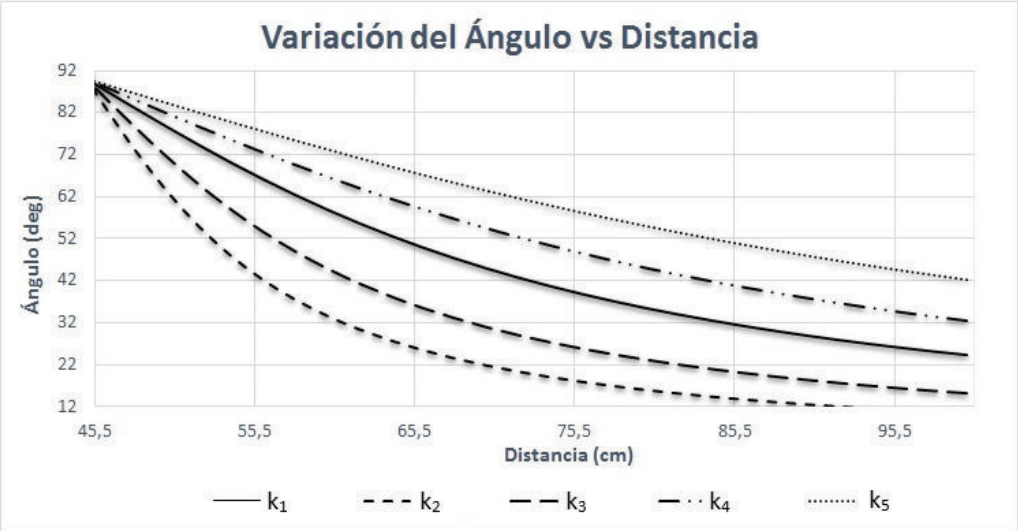
Como se indica en la Figura 1.18 cuando el obstáculo es detectado por los sensores 0, 1, 2 ó 3 el giro será en sentido horario, y si los sensores 4, 5, 6 ó 7 son los que detectan el obstáculo, el giro será en sentido antihorario, teniendo siempre prioridad respecto al sensor que detecte la menor distancia dentro del rango d_{iv}.

Al realizarlo de esta forma se considera además el problema de la aparición de varios obstáculos, pues se asegura que el robot evada al obstáculo que se encuentre más cercano al él. Si durante la evasión el robot encontrara un nuevo obstáculo, éste será evadido si su distancia al robot es menor que la distancia del robot al obstáculo a evadir actualmente, y así sucesivamente.

También para establecer la $d_{mín}$ se debe procurar establecer una distancia mayor a la distancia mínima de detección de los sensores ultrasónicos, pues caso contrario los obstáculos no serán detectables. En el caso del robot Pioneer 3DX la distancia mínima de detección que poseen los sensores es de 0.10 (m) [31] .

Después resulta importante la selección de k (constante de calibración de la rapidez con que el ángulo de giro tiende a 90°) debido a que de acuerdo a este parámetro el ángulo α de rotación llega más o menos rápido a 90º.

En la Figura 1.19 se evidencia la relación que existe entre el ángulo α y la distancia $d_{(R-O)}$ con diferentes valores de k. Aquí se aprecia que la relación no es lineal pues a medida que la distancia $d_{(R-O)}$ disminuye el ángulo α tiende a 90°. Después de realizar la prueba con algunos valores de k y evaluar el desempeño de la evasión finalmente se eligió $k = 0.25$.



k_1	k_2	k_3	k_4	k_5
0.25	0.1	0.15	0,35	0.5

$d_{mín} = 0{,}45$

Figura 1.19. Variación del ángulo α vs. la distancia $d_{(R-O)}$ con diferentes valores de k

El ángulo α luego es utilizado en una matriz de rotación ((1.14), [33]) que se aplica sobre los errores de trayectoria en cada instante de tiempo cuando el robot se encuentra con un obstáculo, y que permite a la plataforma alterar su trayectoria para realizar la evasión.

$$Matriz\ de\ rotación = \begin{bmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{bmatrix} \quad (1.14)$$

Adicionalmente se considera un factor β que encarga de frenar al robot para que la evasión no se realice a la velocidad actual, pues de ser así existe una gran probabilidad de colisión. Gracias a este valor se logra que el robot evada a una velocidad suficientemente moderada para que el robot pueda rotar.

$$\beta = \begin{cases} 1 & ,si \quad d_{(R-O)} > d_{iv} \\ ]0;1[& ,si \quad d_{(R-O)} \leq d_{iv} \end{cases} \tag{1.15}$$

Finalmente (1.16) indica cómo queda la ecuación de rotación aplicada a los errores de la trayectoria.

$$\begin{bmatrix} \Delta x_m \\ \Delta y_m \end{bmatrix} = \begin{cases} \begin{bmatrix} \Delta x_m \\ \Delta y_m \end{bmatrix} & ,si \quad d_{(R-O)} > d_{iv} \\ \beta \begin{bmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{bmatrix} \begin{bmatrix} \Delta x_m \\ \Delta y_m \end{bmatrix} & ,si \quad d_{(R-O)} \leq d_{iv} \end{cases} \tag{1.16}$$

Se observa que si la distancia del robot al obstáculo $d_{(R-O)}$ es mayor que la distancia de inicio de evasión d_{iv} el robot sigue su trayectoria normalmente, pero caso contrario se aplica la matriz de rotación multiplicada por el factor β para evadirlo.

1.7 SISTEMA OPERATIVO ROBÓTICO (ROS)

El Sistema Operativo Robótico (ROS por sus siglas en inglés) es un framework flexible para la creación de software orientado a la robótica, este es una colección de herramientas, librerías, drivers para dispositivos, visualizadores gráficos y convenciones con el objetivo de ayudar a los desarrolladores de software a crear aplicaciones robóticas. [34]

Existen tres partes principales en el ambiente de ROS:

- Herramientas de lenguaje (C++. Python y LISP) y plataformas independientes, para construir y distribuir software basado en ROS.
- Implementaciones de librerías cliente de ROS como roscpp, rospy y roslisp.
- Paquetes que contienen código relativo a la aplicación que usa una o más librerías cliente de ROS.

En general ROS está enfocado a sistemas UNIX (Ubuntu-Linux), aunque existen versiones experimentales para Fedora, Mac OS X, Arch, Debian o Microsoft Windows.

ROS posee una licencia de código abierto BSD[1] que permite libertad para su uso comercial e investigativo, gracias a esto existe un vasto apoyo de la comunidad para la creación u obtención de paquetes relativos a diferentes aplicaciones.

1.7.1 PAQUETES DE ROS

El software creado en ROS se organiza en forma de paquetes, que son un contenedor donde se encuentran las librerías internas y externas a ROS, bases de datos, archivos de configuración, código propio y de terceros, archivos de definición de mensajes y servicios, y cualquier otro elemento necesario para el proyecto.

La organización de Ros en forma de paquetes presenta las siguientes ventajas:

- El código creado en un lenguaje de programación puede coexistir con código creado en diferentes lenguajes como Python, C++, etc.
- El código desarrollado es estandarizado haciendo así que un mismo paquete sirva para varias plataformas robóticas.
- Los paquetes pueden ejecutarse en paralelo en una o varias máquinas.

1.7.2 ESPACIO DE TRABAJO

El espacio de trabajo es la carpeta donde se instalan, editan y compilan los diferentes paquetes que se crean para la realización del proyecto. Además aquí se encuentra los archivos *CMakeList.txt* y *Package.xml*, los cuales gestionan las dependencias y la estructura de los diferentes paquetes que contendrá el espacio de trabajo.

1.7.3 NODOS

Los nodos son procesos que se ejecutan en ROS, y que cuentan con las siguientes características:

[1] La licencia BSD (Berkeley Software Distribution) es una licencia de software libre permisiva que permite el uso del código fuente en software no libre a diferencia de otras como la GPL, lo que la convierte en muy cercana al dominio público.

- Son procesos de tipo bucle, y su programación se encuentra dentro del código que compone un paquete, mismo que a su vez puede contener más de un nodo.
- La comunicación entre nodos se realiza a través de tópicos, los cuales llevan información de un nodo a otro de manera unidireccional.
- Los nodos pueden ejecutarse en paralelo o individualmente. Una vez definidos los nodos estos se comunican automáticamente entre sí, ya sea en una o varias máquinas.
- Un nodo programado en un lenguaje de programación puede comunicarse con otros programados en distintos lenguajes.

1.7.4 TÓPICO

Los tópicos son los medios por los cuales los nodos intercambian información y cuentan con las siguientes características:

- Llevan información de manera unidireccional, por lo que suelen llamarse tópico de entrada o salida.
- La información que lleva un tópico está estandarizada en forma de estructuras llamadas mensajes.
- Los tópicos de entrada tienen nodos publicadores y los de salida tienen nodos suscriptores, además un mismo tópico puede tener varios publicadores o suscriptores.

1.7.4.1 Publicación/Suscripción

La Publicación es el proceso mediante el cual un nodo coloca información en forma de mensajes en un tópico de salida con el propósito de enviarla a otros nodos.

A su vez, la Suscripción es el proceso mediante el cual un nodo obtiene información de otro nodo a través de su tópico de salida.

Un nodo puede estar suscrito a varios tópicos a la vez, de igual manera puede estar publicando en uno o varios tópicos de salida.

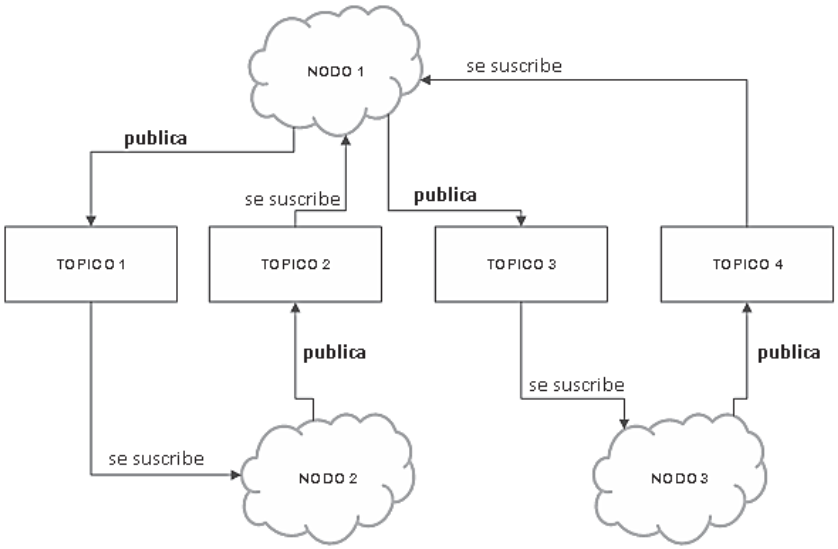


Figura 1.20. Representación del flujo de información entre tópicos y nodos

Por ejemplo en la Figura 1.20 se observa cómo se realiza el flujo de información entre tópicos y nodos, donde las flechas indican si el nodo está suscribiéndose a un tópico determinado o publicando en él.

1.7.4.2 Servicio

Los servicios son otra manera con la cual los nodos se comunican entre sí, pero a diferencia de las publicaciones y las suscripciones éstos se basan en el envío de una propuesta y la recepción de una respuesta.

Cuando un nodo denominado cliente desea que otro nodo llamado servidor realice una acción, envía una propuesta, si el servidor tiene los datos suficientes y está listo retorna una respuesta acorde a dicha propuesta.

1.7.5 ROSARIA

RosAria es un paquete de ROS provisto por MobileRobots [5] basado en la librería Aria que se encuentra programado en lenguaje C++.

La función principal de este paquete es la obtención y envío de información al robot, por lo que en el código del paquete RosAria se encuentra programado el nodo *rosaria*, que consta con tópicos de salida y entrada.

Los tópicos de salida usados en este trabajo ofrecen la siguiente información:

- Odometría del robot.
- Obstáculos detectados por el sonar del robot.
- Nivel de carga de la batería.
- Estado de carga de la batería.
- Información de los bumpers del robot.

El tópico de entrada llamado *cmd_vel* se encarga de tomar las velocidades lineal y angular que lleguen a él, para que el nodo *rosaria* las envíe a la plataforma robótica en forma de velocidad angular de los motores.

1.7.6 ROS EN RED

El sistema operativo ROS facilita el trabajo maestro esclavo, ofreciendo comunicación mediante protocolo IP en una red wireless. Para hacer uso de esta característica basta con referenciar la IP del maestro y el o los esclavos, esto es lo que se utiliza en las pruebas experimentales de este trabajo.

Gracias a esta cualidad, los nodos que se ejecutan en el maestro y esclavo intercambian información a través de sus respectivos tópicos de entrada y salida como si se ejecutaran en una sola máquina, haciendo que en el código para la generación de los nodos no necesite tomar en cuenta el trabajo en red.

El trabajo de sistemas multimaestro se encuentra en desarrollo en el momento en el que este trabajo fue escrito, y tiene como finalidad el avance de la plataforma ROS en la industria.

1.8 INTERFAZ GRÁFICA – QT CREATOR

Una interfaz gráfica es un dispositivo o software que le permite al usuario comunicarse con una máquina o sistema automático. Su principio fundamental se refiere a la mediación entre el hombre y la máquina, que implica facilitar la interacción entre estos dos sistemas de diferente naturaleza. [7]

Resulta importante crear un diseño que se enfoque en el usuario y no en el sistema. Para el presente proyecto se ha considerado utilizar un programa que permita cumplir estos principios, y cuya programación sea práctica e intuitiva: Qt Creator.

Qt Creator es un entorno de desarrollo integrado (IDE – integrated development environment) que provee las herramientas que permiten completar las tareas a través de todo el ciclo de vida del diseño y desarrollo de una aplicación, desde crear un proyecto hasta desplegar la aplicación en las plataformas objetivo. Incluye un depurador visual y un layout de GUI integrado. [21], [32]

1.8.1.1 Qt Designer

Es la herramienta de Qt que permite diseñar y construir interfaces gráficas de usuario desde componentes de Qt. Esto permite componer, personalizar y probar widgets o ventanas de diálogo de manera what-you-see-is-what-you-get WYSIWYG ("lo que ves es lo que obtienes"). [32]

Los widgets y las formas creadas con el Qt Designer se integran sin problemas con el código programado usando el mecanismo de Qt de señales y ranuras (signals and slots) que permite asignarle determinados comportamientos a los elementos gráficos. También todas las propiedades definidas en el Qt Designer pueden ser modificadas dinámicamente dentro del código.

Adicionalmente las características de promoción de widgets y plugins permiten usar y crear componentes personalizados con funciones específicas.

1.9 FUNDAMENTOS DE ÁLGEBRA LINEAL

En esta sección se presentan conceptos básicos de álgebra lineal utilizados para implementar los algoritmos de Euler y Trapezoide del capítulo dos (secciones 2.2 y 2.3).

Considerando el sistema matricial de ecuaciones:

$$A_{mxn}\vec{u}_{nx1} = \vec{b}_{mx1} \tag{1.17}$$

Donde:

$$\vec{u}_{nx1} = \begin{bmatrix} u_{11} \\ u_{21} \\ \vdots \\ u_{n1} \end{bmatrix} \tag{1.18}$$

Se tiene que el espacio columna de A_{mxn} (ECA) es un subespacio vectorial de $\mathbb{R}^m$ que contiene todas las combinaciones lineales de las columnas de A_{mxn}; y es generado por la base que tiene como componentes las columnas linealmente independientes de la matriz A_{mxn}[20].

$$Base\ de\ ECA = \left\{\overrightarrow{a_1}_{mx1}, \overrightarrow{a_2}_{mx1}, \dots, \overrightarrow{a_n}_{mx1}\right\} \tag{1.19}$$

Considerando esta descripción se reformula (1.17) por columnas:

$$u_{11}\overrightarrow{a_1}_{mx1} + u_{21}\overrightarrow{a_2}_{mx1} + \cdots + u_{n1}\overrightarrow{a_n}_{mx1} = \vec{b}_{mx1} \tag{1.20}$$

Ahora el problema es hallar los valores de los elementos de $\vec{u}_{nx1}$ que multiplican a las columnas de A_{mxn} para producir $\vec{b}_{mx1}$. En general el sistema $A_{mxn}\vec{u}_{nx1} = \vec{b}_{mx1}$ es resoluble sí y solo sí el vector $\vec{b}_{mx1}$ puede ser expresado como la combinación lineal de las columnas de A_{mxn}; entonces $\vec{b}_{mx1}$ está en el espacio columna de A_{mxn} [11], [20].

Sin embargo para el caso particular de este proyecto (como se ve en 2.2 y 2.3) se tiene:

- Número de ecuaciones mayor a número de incógnitas $(m > n)$.
- Columnas de la matriz A_{mxn} linealmente independientes entre sí.

Como $m > n,$ el sistema (1.17) es inconsistente y probablemente no exista un $\vec{u}_{nx1}$ que haga que $\vec{b}_{mx1}$ sea una combinación de las columnas de A_{mxn}. Entonces el objetivo es hallar los elementos de $\vec{u}_{nx1}$ de manera que si no existe solución por lo menos se tome los mejores valores de estos, los cuales hacen que el error (1.21) sea mínimo [20].

$$E = \left\|A_{mxn}\vec{u}_{nx1} - \vec{b}_{mx1}\right\| \tag{1.21}$$

En [3] muestra que el error es minimo cuando se proyecta el vector $\vec{b}_{mx1}$ en el espacio columna de A_{mxn} debido a que proyección ortogonal es sinónimo de minimización, ademas se muestra que $\vec{b}_{mx1}$ se puede descomponer en dos componentes, una componente en el espacio columna de A_{mxn} $(\overrightarrow{b(ECA)}_{mx1})$ y otra

en el espacio nulo de A_{mxn} traspuesta $(\overrightarrow{b(N(A^T))}_{mx1})$ como se ve en la Figura 1.21, [3]:

$$\vec{b}_{mx1} = \overrightarrow{b(ECA)}_{mx1} + \overrightarrow{b(N(A^T))}_{mx1} \tag{1.22}$$

Donde el espacio nulo de A_{mxn} $(N(A))$ esta formado por el conjunto de vectores que resuelven la ecuación $A_{mxn}\vec{u}_{nx1} = \vec{0}$ y el espacio nulo de A_{mxn} transpuesta $N(A^T)$ es perpendicular al espacio columna de A_{mxn} (ECA), [3].

$$ECA \perp N(A^T) \tag{1.23}$$

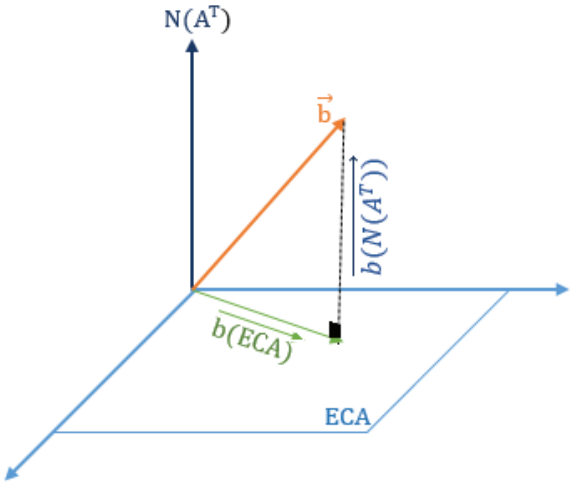


Figura 1.21. Proyección ortogonal del vector $\vec{b}$ en el espacio columna de A, [3]

Entonces como se expone en [20], una forma de realizar la proyección ortogonal es igualar $\overrightarrow{b(N(A^T))}_{mx1} = \vec{0}$, y hallar los elementos de $\vec{u}_{nx1}$ para que $\vec{b}_{mx1} = \overrightarrow{b(ECA)}_{mx1}$; estos elementos representan la solución para la cual el error (1.21) es mínimo, [19], [20]. Finalmente con estos valores de $\vec{u}_{nx1}$, se debe hallar la restricción que hace que el vector $\overrightarrow{b(N(A^T))}_{mx1}$ sea nulo.

Para lograr lo anterior se expresa el vector $\overrightarrow{b(ECA)}_{mx1}$ como una combinación lineal de la base del espacio columna de la matriz A_{mxn}:

$$\vec{b}_{mx1} = \overrightarrow{b(ECA)}_{mx1} = k_1\left(\overrightarrow{a_1}_{mx1}\right) + k_2\left(\overrightarrow{a_2}_{mx1}\right) + \cdots + k_n\left(\overrightarrow{a_n}_{mx1}\right) \tag{1.24}$$

En este caso particular (como se ve en 2.2 y 2.3) los vectores columna ($\overrightarrow{a_{\iota_{mx1}}}$) son linealmente independientes y ortogonales entre sí, por lo que el producto punto entre ellos es cero [3].

De esta forma al realizar el producto punto por un $\overrightarrow{a_{\iota_{mx1}}}$ a ambos lados de la ecuación (1.24) y al despejar se pueden calcular los k_i con la ecuación (1.25) como se ve en [11]:

$$k_i = \frac{< \overrightarrow{a_{\iota_{mx1}}}.\ \vec{b}_{mx1} >}{||\overrightarrow{a_{\iota_{mx1}}}||^2} \qquad i = 1,2,3,\dots,n \tag{1.25}$$

Al ser análogos (1.20) y (1.24) se obtiene que:

$$k_i = u_{i1} = \frac{< \overrightarrow{a_{\iota_{mx1}}}.\ \vec{b}_{mx1} >}{||\overrightarrow{a_{\iota_{mx1}}}||^2} \qquad i = 1,2,3,\dots,n \tag{1.26}$$

Siendo estos valores de u_{i1} los que producen el error (1.21) sea mínimo para el sistema de ecuaciones matriciales originalmente propuesto (1.17).

CAPÍTULO 2

DESARROLLO DE LOS ALGORITMOS DE CONTROL

Uno de los problemas que se presentan para la implementación de controladores en robot móviles es tener una señal de control que le permita al vehículo seguir una trayectoria, posición o camino deseado.

Para este proyecto se aborda el problema particular de seguimiento de trayectorias, para ello en este capítulo se presenta en primer lugar el algoritmo utilizado para la generación de las trayectorias de prueba y después se diseñan cuatro controladores: controlador de Euler, Trapezoide, PI y por modo deslizante.

2.1 ALGORITMO DE GENERACIÓN DE TRAYECTORIAS

La generación de trayectorias es de mucha relevancia, debido a que en su cálculo se incluyen parámetros del seguimiento como el tiempo de muestreo, velocidad lineal y tiempo de duración del experimento.

La trayectoria cuenta con dos vectores que corresponden a sus componentes X y Y, además se genera un vector que corresponde al vector de tiempo. Se toman uno a uno los elementos de estos vectores y junto con las expresiones de los controladores se calculan las acciones de control.

El procedimiento de cálculo de los vectores es el siguiente:

- Usando el tiempo de duración y el tiempo de muestreo se calcula el número de elementos que tiene el vector tiempo.

$$n = \frac{t}{T_0}$$

  n : número de elementos del vector tiempo
  t : tiempo de duración del experimento
  T_0 : tiempo de muestreo

- Para obtener el vector tiempo se genera un vector que empiece en cero, termine en el tiempo de duración y contenga tantos elementos como n

- Ahora se calculan los vectores X y Y correspondientes a las componentes de la trayectoria usando sus ecuaciones paramétricas, tal como se ve en la Tabla 2.1. Los elementos de los vectores son las componentes x_{ref_m} y y_{ref_m} donde $m = 0,1,2,3,4,\dots,n-1$, v es la velocidad lineal, r es el radio del círculo o de la trayectoria de doble frecuencia y *p* es la pendiente de la recta para trayectorias lineales.

Tabla 2.1. Ecuaciones consideradas para la generación de trayectorias

Trayectoria	Componente X	Componente Y
Lineal *p*=0	$\vec{x} = v\vec{t}$	$\vec{y} = \vec{0}$
Circular	$\vec{x} = r\ cos\left(\frac{v}{r}\vec{t}\right)$	$\vec{y} = r\ sin\left(\frac{v}{r}\vec{t}\right)$
Doble Frecuencia	$\vec{x} = r\ cos\left(\frac{v}{r\sqrt{5}}\vec{t}\right)$	$\vec{y} = r\ sin\left(\frac{2v}{r\sqrt{5}}\vec{t}\right)$
Lineal *p*=1	$\vec{x} = \frac{\sqrt{2}}{2} v\ \vec{t}$	$\vec{y} = \frac{\sqrt{2}}{2} v\ \vec{t}$
Cuadrangular	$\vec{x} = v\vec{t}$ $\vec{x} = \vec{0}$	$\vec{y} = \vec{0}$ $\vec{y} = v\vec{t}$

Nota: La trayectoria de doble frecuencia no tiene velocidad constante, y el cálculo de sus ecuaciones paramétricas se ha hecho para que la velocidad oscile alrededor de la velocidad lineal seleccionada.

2.2 MÉTODOS NUMÉRICOS CON APROXIMACIÓN DE EULER

Para solucionar el problema de direccionamiento del robot, se toma en cuenta que se aplica el controlador a la velocidad lineal y a la velocidad angular, y que estas son medibles y controlables cada periodo de muestreo.

Para realizar el desarrollo de este controlador se usa el modelo cinemático del robot descrito en la sección 1.4.1 y los criterios de álgebra lineal explicados en la sección 1.9 del marco teórico.

2.2.1 ANÁLISIS CON EL PUNTO DE INTERÉS DESPLAZADO Y EN EL CENTRO DE GRAVEDAD DEL ROBOT

Al reemplazar el modelo cinemático del robot (1.1) en la aproximación de Euler descrita en el marco teórico (1.6) y despejar, se obtiene el siguiente conjunto de ecuaciones:

$$\frac{1}{T_0}\begin{bmatrix} x_{m+1}-x_m \\ y_{m+1}-y_m \\ \varphi_{m+1}-\varphi_m \end{bmatrix} = \frac{1}{T_0}\begin{bmatrix} \Delta x_m \\ \Delta y_m \\ \Delta\varphi_m \end{bmatrix} = \begin{bmatrix} cos(\varphi) & -a\,sin(\varphi) \\ sin(\varphi) & a\,cos(\varphi) \\ 0 & 1 \end{bmatrix}\begin{bmatrix} v_m \\ \omega_m \end{bmatrix} \tag{2.1}$$

Se observa que el sistema descrito en (2.1) toma la forma:

$$A\vec{u} = \vec{b} \tag{2.2}$$

Donde:

$$A = \begin{bmatrix} cos(\varphi) & -a\,sin(\varphi) \\ sin(\varphi) & a\,cos(\varphi) \\ 0 & 1 \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} v_m \\ \omega_m \end{bmatrix} \qquad \vec{b} = \frac{1}{T_0}\begin{bmatrix} \Delta x_m \\ \Delta y_m \\ \Delta\varphi_m \end{bmatrix} \tag{2.3}$$

Puesto que las velocidades son en todo momento las deseadas, se denotan de aquí en adelante como v_{d_m} y ω_{d_m}.

$$\vec{u} = \begin{bmatrix} v_{d_m} \\ \omega_{d_m} \end{bmatrix} \tag{2.4}$$

Al igual que en (1.22) se descompone el vector $\vec{b}$ en una componente en el espacio columna de A y una componente en el espacio nulo de A traspuesta:

$$\vec{b} = \overrightarrow{b(ECA)} + \overrightarrow{b(N(A^T))} \tag{2.5}$$

De la misma forma que en (1.19) la base del espacio columna de A es:

$$Base\ ECA = \left\{ \begin{bmatrix} cos(\varphi_m) \\ sin(\varphi_m) \\ 0 \end{bmatrix}, \begin{bmatrix} -a\,sin(\varphi_m) \\ a\,cos(\varphi_m) \\ 1 \end{bmatrix} \right\} \tag{2.6}$$

Entonces se puede expresar el vector $\overrightarrow{b(ECA)}$ como una combinación lineal de la base del espacio columna de A.

$$\vec{b} = \overrightarrow{b(ECA)} = k_1 \begin{bmatrix} cos(\varphi_m) \\ sin(\varphi_m) \\ 0 \end{bmatrix} + k_2 \begin{bmatrix} -a\, sin(\varphi_m) \\ a\, cos(\varphi_m) \\ 1 \end{bmatrix} \tag{2.7}$$

Como se ve en (2.8) los vectores columna de A son ortogonales:

$$< \begin{bmatrix} cos(\varphi_m) \\ sin(\varphi_m) \\ 0 \end{bmatrix} \cdot \begin{bmatrix} -a\, sin(\varphi_m) \\ a\, cos(\varphi_m) \\ 1 \end{bmatrix} >= a\, sin(\varphi_m)\, cos(\varphi_m) - a\, cos(\varphi_m)\, sin(\varphi_m) = 0 \tag{2.8}$$

Entonces como en (1.26) se puede obtener k_1 y k_2 usando las siguientes expresiones:

$$k_1 = \frac{< \begin{bmatrix} cos(\varphi_m) \\ sin(\varphi_m) \\ 0 \end{bmatrix} \cdot \frac{1}{T_0} \begin{bmatrix} \Delta x_m \\ \Delta y_m \\ \Delta \varphi_m \end{bmatrix} >}{\left\| \begin{bmatrix} cos(\varphi_m) \\ sin(\varphi_m) \\ 0 \end{bmatrix} \right\|^2} \tag{2.9}$$

$$k_1 = \frac{\frac{1}{T_0}(\Delta x_m\, cos(\varphi_m) + \Delta y_m\, sin(\varphi_m) + (\Delta \varphi_m)(0))}{cos^2(\varphi_m) + sin^2(\varphi_m) + 0^2} \tag{2.10}$$

$$k_2 = \frac{< \begin{bmatrix} -a\, sin(\varphi_m) \\ a\, cos(\varphi_m) \\ 1 \end{bmatrix} \cdot \frac{1}{T_0} \begin{bmatrix} \Delta x_m \\ \Delta y_m \\ \Delta \varphi_m \end{bmatrix} >}{\left\| \begin{bmatrix} -a\, sin(\varphi_m) \\ a\, cos(\varphi_m) \\ 1 \end{bmatrix} \right\|^2} \tag{2.11}$$

$$k_2 = \frac{\frac{1}{T_0}(-a\Delta x_m\, sin(\varphi_m) + a\Delta y_m\, cos(\varphi_m) + \Delta \varphi_m)}{a^2 cos^2(\varphi_m) + a^2 sin^2(\varphi_m) + 1^2} \tag{2.12}$$

De igual manera que en (1.26) se obtienen los componentes del vector $\vec{u}$:

$$v_{d_m} = k_1 = \frac{1}{T_0}(\Delta x_m\, cos(\varphi_m) + \Delta y_m\, sin(\varphi_m)) \tag{2.13}$$

$$\omega_{d_m} = k_2 = \frac{1}{T_0(a^2 + 1)}(-a\Delta x_m\, sin(\varphi_m) + a\Delta y_m\, cos(\varphi_m) + \Delta \varphi_m) \tag{2.14}$$

Obtenidas las expresiones que permiten calcular la velocidad angular (2.14) y la velocidad lineal (2.13) se debe hallar la condición para que $\overrightarrow{b(N(A^T))} = \vec{0}$.

$$\overrightarrow{b(N(A^T))} = \vec{b} - \overrightarrow{b(ECA)} = \vec{0} \tag{2.15}$$

Reemplazando (2.13) y (2.14) en $\overrightarrow{b(ECA)}$ de (2.7) para luego sustituirlo en (2.15) junto con $\vec{b}$ de (2.3) se obtiene:

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = \frac{1}{T_0}\begin{bmatrix}\Delta x_m\\ \Delta y_m\\ \Delta\varphi_m\end{bmatrix} - \frac{1}{T_0}\left(\Delta x_m\, cos(\varphi_m) + \Delta y_m\, sin(\varphi_m)\right)\begin{bmatrix}cos(\varphi_m)\\ sin(\varphi_m)\\ 0\end{bmatrix} + \frac{1}{T_0(a^2+1)}\left(-a\Delta x_m\, sin(\varphi_m)\right. + a\Delta y_m\, cos(\varphi_m) + \Delta\varphi_m\Big)\begin{bmatrix}-a\, sin(\varphi_m)\\ a\, cos(\varphi_m)\\ 1\end{bmatrix} \tag{2.16}$$

Realizando las operaciones de (2.16) y factorando se obtiene:

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix}\Delta x_m(1-cos^2(\varphi_m))\left(\frac{1}{a^2+1}\right) - \Delta y_m\, cos(\varphi_m)sin(\varphi_m)\left(\frac{1}{a^2+1}\right) + \Delta\varphi_m sin(\varphi_m)\left(\frac{a^2}{a^2+1}\right)\\ \Delta x_m\, sin(\varphi_m)\, cos(\varphi_m)\left(\frac{1}{a^2+1}\right) - \Delta y_m(1-sin^2(\varphi_m))\left(\frac{1}{a^2+1}\right) + \Delta\varphi_m\, cos(\varphi_m)\left(\frac{a^2}{a^2+1}\right)\\ \Delta x_m\, sin(\varphi_m)\left(\frac{a}{a^2+1}\right) - \Delta y_m\, cos(\varphi_m)\left(\frac{a}{a^2+1}\right) + \Delta\varphi_m\left(\frac{a^2}{a^2+1}\right)\end{bmatrix} \tag{2.17}$$

Aplicando equivalencias trigonométricas y simplificando se consigue:

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix}\Delta x_m\, sin(\varphi_m)\left(\frac{1}{a^2+1}\right) - \Delta y_m\, cos(\varphi_m)\left(\frac{1}{a^2+1}\right) + \Delta\varphi_m\left(\frac{a^2}{a^2+1}\right)\\ \Delta x_m\, sin(\varphi_m)\left(\frac{1}{a^2+1}\right) - \Delta y_m\, cos(\varphi_m)\left(\frac{1}{a^2+1}\right) + \Delta\varphi_m\left(\frac{a^2}{a^2+1}\right)\\ \Delta x_m\, sin(\varphi_m)\left(\frac{a}{a^2+1}\right) - \Delta y_m\, cos(\varphi_m)\left(\frac{a}{a^2+1}\right) + \Delta\varphi_m\left(\frac{a^2}{a^2+1}\right)\end{bmatrix} \tag{2.18}$$

Las dos primeras filas de (2.18) son linealmente dependientes, así que el sistema de ecuaciones queda reducido a la segunda y tercera fila:

$$\begin{cases}\bullet\ \Delta x_m\, sin(\varphi_m)\left(\frac{1}{a^2+1}\right) - \Delta y_m\, cos(\varphi_m)\left(\frac{1}{a^2+1}\right) + \Delta\varphi_m\left(\frac{a^2}{a^2+1}\right) = 0 & (2.19)\\ \bullet\ \Delta x_m\, sin(\varphi_m)\left(\frac{a}{a^2+1}\right) - \Delta y_m\, cos(\varphi_m)\left(\frac{a}{a^2+1}\right) + \Delta\varphi_m\left(\frac{a^2}{a^2+1}\right) = 0 & (2.20)\end{cases}$$

Sustrayendo (2.19) de (2.20) y factorando, se consigue la expresión:

$$(\Delta x_m \, sin(\varphi_m) - \Delta y_m \, cos(\varphi_m)) \left(\frac{a-1}{a^2+1}\right) = 0 \tag{2.21}$$

La ecuación (2.21) presenta dos posibles soluciones

$$\Delta x_m \, sin(\varphi_m) - \Delta y_m \, cos(\varphi_m) = 0 \qquad \text{ó} \qquad a = 1 \tag{2.22}$$

Siendo la única solución posible (2.23) debido a que a es un parámetro (Figura 1.3) y no una variable.

$$\Delta x_m \, sin(\varphi_m) - \Delta y_m \, cos(\varphi_m) = 0 \tag{2.23}$$

$$\Delta x_m \, sin(\varphi_m) = \Delta y_m \, cos(\varphi_m) \tag{2.24}$$

$$\frac{sin(\varphi_m)}{cos(\varphi_m)} = \frac{\Delta y_m}{\Delta x_m} \tag{2.25}$$

$$\varphi_{ez_m} = tan^{-1}\left(\frac{\Delta y_m}{\Delta x_m}\right) \tag{2.26}$$

Donde φ_{ez_m} (Figura 2.1) es el ángulo que hace que la condición $\overrightarrow{b(N(A^T))} = \vec{0}$ se cumpla. Si se calculan las velocidades lineal v_{d_m} y angular ω_{d_m} con la restricción (2.26), se logra que el sistema $A\vec{u} = \vec{b}$ tenga solución, debido a que se asegura que $\vec{b}$ ahora sea igual a su proyección ortogonal en el espacio columna de A (es decir $\vec{b} = \overrightarrow{b(ECA)}$), que es el valor para el cual el error (1.21) es mínimo en la Figura 1.21.

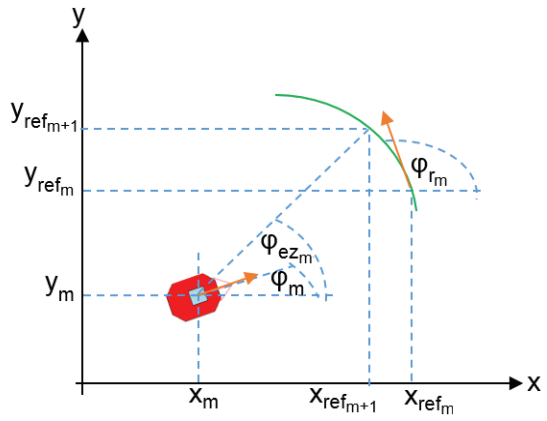


Figura 2.1. Direccionamiento del robot mediante el ángulo φ_{ez_m}

En resumen para que el sistema descrito por (1.1) siga una trayectoria deseada utilizando la aproximación de Euler, las velocidades lineal y angular deben ser iguales a:

$$\begin{aligned} v_{d_m} &= \frac{1}{T_0}\left(\Delta x_m \cos\left(\varphi_{ez_m}\right) + \Delta y_m \sin\left(\varphi_{ez_m}\right)\right) \\ \omega_{d_m} &= \frac{1}{T_0(a^2+1)}\left(-a\Delta x_m \sin\left(\varphi_{ez_m}\right) + a\Delta y_m \cos\left(\varphi_{ez_m}\right) + \Delta\varphi_m\right) \end{aligned} \tag{2.27}$$

Donde el ángulo de referencia φ_{ez_m} es igual a:

$$\varphi_{ez_m} = tan^{-1}\left(\frac{\Delta y_m}{\Delta x_m}\right) \tag{2.28}$$

2.2.2 CONSIDERACIONES PARA QUE EL ERROR TIENDA A CERO DE MANERA ESTABLE

El siguiente paso para el diseño del controlador es asegurar que el error disminuya suavemente de manera estable a medida que el tiempo transcurre, para esto se escoge la expresión de (2.29), [11],[19].

$$x_{m+1} = x_{ref_{m+1}} - k_x\left(x_{ref_m} - x_m\right) \tag{2.29}$$

$$0 = \left(x_{ref_{m+1}} - x_{m+1}\right) - k_x\left(x_{ref_m} - x_m\right) \tag{2.30}$$

$$e_{x_{m+1}} = k_x e_{x_m} \tag{2.31}$$

$$\frac{e_{x_{m+1}}}{e_{x_m}} = k_x \tag{2.32}$$

Analizando (2.31) se tiene que:

$$\begin{aligned} e_{x_1} &= k_x e_{x_0} \\ e_{x_2} &= k_x e_{x_1} = {k_x}^2 e_{x_0} \\ e_{x_3} &= k_x e_{x_2} = {k_x}^3 e_{x_0} \\ &\vdots \\ e_{x_i} &= {k_x}^i e_{x_0} \quad i = 1,2,3,4,\ldots \end{aligned} \tag{2.33}$$

Como se desea que el error tienda a cero a medida que el tiempo avanza, se elige un valor de k_x entre cero y uno como se ve en (2.34), puesto que valores superiores

a uno harían que el error aumente en el tiempo y valores menores a cero provocan inestabilidad en el sistema al haber un continuo cambio del signo del error.

$$0 < k_x < 1 \tag{2.34}$$

De manera análoga a (2.29) se realiza la sustitución planteada en [11] para y_{m+1} y para φ_{m+1}, y se reemplazan en (2.35) obteniéndose las expresiones a utilizar para $\Delta x_m, \Delta y_m$ y $\Delta\varphi_m$ (2.36).

$$\begin{aligned} \Delta x_m &= x_{m+1} - x_m \\ \Delta y_m &= y_{m+1} - y_m \\ \Delta\varphi_m &= \varphi_{m+1} - \varphi_m \end{aligned} \tag{2.35}$$

$$\begin{aligned} \Delta x_m &= x_{ref_{m+1}} - x_m - k_x(x_{ref_m} - x_m) \\ \Delta y_m &= y_{ref_{m+1}} - y_m - k_y(y_{ref_m} - y_m) \\ \Delta\varphi_m &= \varphi_{ez_m} - \varphi_m - k_\varphi(\varphi_{ez_m} - \varphi_m) \end{aligned} \tag{2.36}$$

Nota:

$$\varphi_{ref_{m+1}} = \varphi_{ref_m} = \varphi_{ez_m}$$

2.2.3 ANÁLISIS CON EL PUNTO DE INTERÉS EN EL CENTRO DEL EJE DE LAS RUEDAS

Ahora se halla la restricción para la que se cumple que $\overrightarrow{b(N(A^T))} = \vec{0}$ utilizando el modelo no holonómico no mejorado del robot móvil, que corresponde al representado por la Figura 1.3 cuando $a = 0$. Esto se hace con el objetivo de comparar las restricciones obtenidas al usar el modelo no holonómico no mejorado y mejorado del robot móvil.

Sustituyendo $a = 0$ en (2.16) se obtiene:

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = \frac{1}{T_0}\begin{bmatrix}\Delta x_m\\ \Delta y_m\\ \Delta\varphi_m\end{bmatrix} - \frac{1}{T_0}(\Delta x_m\, cos(\varphi_m) + \Delta y_m\, sin(\varphi_m))\begin{bmatrix}cos(\varphi_m)\\ sin(\varphi_m)\\ 0\end{bmatrix} + \frac{1}{T_0}(\Delta\varphi_m)\begin{bmatrix}0\\0\\1\end{bmatrix} \tag{2.37}$$

Realizando las operaciones de (2.37) y simplificando se tiene:

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix}\frac{1}{T_0}\Delta x_m - v_{d_m}cos(\varphi_m)\\ \frac{1}{T_0}\Delta y_m - v_{d_m}sin(\varphi_m)\\ \frac{1}{T_0}\Delta\varphi_m - \omega_{d_m}\end{bmatrix} \tag{2.38}$$

La ecuación (2.39) del sistema matricial de ecuaciones (2.38) no se toma en cuenta para hallar la restricción, puesto que esta es la ecuación para el cálculo de la velocidad angular (2.14) cuando $a = 0$

$$\omega_{d_m} = \frac{1}{T_0}\Delta\varphi_m \tag{2.39}$$

Tomando las filas restantes de (2.37) se tiene el siguiente siguiente sistema de ecuaciones:

$$\begin{cases} \bullet\ \frac{1}{T_0}\Delta x_m - v_{d_m}cos(\varphi_m) = 0 & (2.40)\\ \bullet\ \frac{1}{T_0}\Delta y_m - v_{d_m}sin(\varphi_m) = 0 & (2.41)\end{cases}$$

Dividiendo (2.41) para (2.40) y despejando se obtiene:

$$\frac{\frac{1}{T_0}\Delta y_m}{\frac{1}{T_0}\Delta x_m} = \frac{v_{d_m}\, sin(\varphi_{m+1})}{v_{d_m}\, cos(\varphi_{m+1})} \tag{2.42}$$

Simplificando (2.42) se tiene la restricción para $\overrightarrow{b(N(A^T))} = \vec{0}$ del sistema no holonómico no mejorado.

$$\varphi_{ez_m} = tan^{-1}\left(\frac{\Delta y_m}{\Delta x_m}\right) \tag{2.43}$$

Donde φ_{ez_m} (Figura 2.1) es el ángulo que hace que la condición $\overrightarrow{b(N(A^T))} = \vec{0}$ se cumpla.

Como se observa, las ecuaciones (2.28) y (2.43) son las mismas, es decir, las restricciones para que $\overrightarrow{b(N(A^T))} = \vec{0}$ son las mismas tanto para el modelo no holonómico mejorado como para el mejorado.

Saber que la restricción es la misma para ambos casos resulta especialmente útil para la simplificación del cálculo de la restricción usando la aproximación trapezoidal.

2.3 MÉTODOS NUMÉRICOS CON APROXIMACIÓN TRAPEZOIDAL

Al reemplazar el modelo cinemático del robot (1.1) en la aproximación trapezoidal descrita en el marco teórico (1.8) y despejar, se obtiene el siguiente sistema matricial de ecuaciones:

$$\begin{bmatrix} cos(\varphi_{m+1}) & -a\,sin(\varphi_{m+1}) \\ sin(\varphi_{m+1}) & a\,cos(\varphi_{m+1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{m+1} \\ \omega_{m+1} \end{bmatrix} = \begin{bmatrix} \frac{2}{T_0}\Delta x_m - v_m\,cos(\varphi_m) + a\omega_m\,sin(\varphi_m) \\ \frac{2}{T_0}\Delta y_m - v_m\,sin(\varphi_m) - a\omega_m\,cos(\varphi_m) \\ \frac{2}{T_0}\Delta\varphi_m - \omega_m \end{bmatrix} \tag{2.44}$$

Se observa que el sistema descrito en (2.44) toma la forma:

$$A\vec{u} = \vec{b} \tag{2.45}$$

Donde:

$$A = \begin{bmatrix} cos(\varphi_{m+1}) & -a\,sin(\varphi_{m+1}) \\ sin(\varphi_{m+1}) & a\,cos(\varphi_{m+1}) \\ 0 & 1 \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} v_{m+1} \\ \omega_{m+1} \end{bmatrix} \tag{2.46}$$

$$\vec{b} = \begin{bmatrix} \frac{2}{T_0}\Delta x_m - v_m\, cos(\varphi_m) + a\omega_m\, sin(\varphi_m) \\ \frac{2}{T_0}\Delta y_m - v_m\, sin(\varphi_m) - a\omega_m\, cos(\varphi_m) \\ \frac{2}{T_0}\Delta\varphi_m - \omega_m \end{bmatrix}$$

Puesto que las velocidades son en todo momento las deseadas, se denotan de aquí en adelante como $v_{d_{m+1}}$ y ω_{d_m+1}.

$$\vec{u} = \begin{bmatrix} v_{d_{m+1}} \\ \omega_{d_m+1} \end{bmatrix} \tag{2.47}$$

Al igual que en (2.5) se descompone el vector $\vec{b}$ en una componente en el espacio columna de A y una componente en el espacio nulo de A traspuesta:

$$\vec{b} = \overrightarrow{b(ECA)} + \overrightarrow{b(N(A^T))} \tag{2.48}$$

De la misma forma que se hizo en (2.6) la base del espacio columna de A, donde al igual que (2.8) se ve que los vectores columna son ortogonales entre sí, es:

$$Base\ ECA = \left\{ \begin{bmatrix} cos(\varphi_{m+1}) \\ sin(\varphi_{m+1}) \\ 0 \end{bmatrix}, \begin{bmatrix} -a\, sin(\varphi_{m+1}) \\ a\, cos(\varphi_{m+1}) \\ 1 \end{bmatrix} \right\} \tag{2.49}$$

Entonces se expresa al vector $\overrightarrow{b(ECA)}$ como una combinación lineal de la base del espacio columna de A.

$$\vec{b} = \overrightarrow{b(ECA)} = k_3 \begin{bmatrix} cos(\varphi_{m+1}) \\ sin(\varphi_{m+1}) \\ 0 \end{bmatrix} + k_4 \begin{bmatrix} -a\, sin(\varphi_{m+1}) \\ a\, cos(\varphi_{m+1}) \\ 1 \end{bmatrix} \tag{2.50}$$

Como se ve en (1.26) se puede obtener k_3 y k_4 usando las siguientes expresiones:

$$k_3 = \frac{< \begin{bmatrix} cos(\varphi_{m+1}) \\ sin(\varphi_{m+1}) \\ 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{2}{T_0}\Delta x_m - v_m\, cos(\varphi_m) + a\omega_m\, sin(\varphi_m) \\ \frac{2}{T_0}\Delta y_m - v_m\, sin(\varphi_m) - a\omega_m\, cos(\varphi_m) \\ \frac{2}{T_0}\Delta\varphi_m - \omega_m \end{bmatrix} >}{\left\| \begin{bmatrix} cos(\varphi_{m+1}) \\ sin(\varphi_{m+1}) \\ 0 \end{bmatrix} \right\|^2} \tag{2.51}$$

$$k_3 = \frac{\frac{2}{T_0}(\Delta x_m\, cos(\varphi_{m+1}) + \Delta y_m\, sin(\varphi_{m+1})) - v_m\, cos(\varphi_{m+1} - \varphi_m) + a\omega_m\, sin(\varphi_m - \varphi_{m+1})}{cos^2(\varphi_{m+1}) + sin^2(\varphi_{m+1}) + 0^2} \quad (2.52)$$

$$k_4 = \frac{< \begin{bmatrix} -a\, sin(\varphi_{m+1}) \\ a\, cos(\varphi_{m+1}) \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{2}{T_0}\Delta x_m - v_m\, cos(\varphi_m) + a\omega_m\, sin(\varphi_m) \\ \frac{2}{T_0}\Delta y_m - v_m\, sin(\varphi_m) - a\omega_m\, cos(\varphi_m) \\ \frac{2}{T_0}\Delta\varphi_m - \omega_m \end{bmatrix} >}{\left\| \begin{bmatrix} -a\, sin(\varphi_m) \\ a\, cos(\varphi_m) \\ 1 \end{bmatrix} \right\|^2} \quad (2.53)$$

$$k_4 = \frac{\left(\frac{2}{T_0}\Delta\varphi_m - \omega_m + \frac{2a}{T_0}(\Delta y_m\, cos(\varphi_{m+1}) - \Delta x_m\, sin(\varphi_{m+1})) + av_m\, sin(\varphi_{m+1} - \varphi_m) - a^2\omega_m\, cos(\varphi_{m+1} - \varphi_m)\right)}{a^2 cos^2(\varphi_{m+1}) + a^2 sin^2(\varphi_{m+1}) + 1^2} \quad (2.54)$$

De igual manera que en (2.13) y (2.14) se obtienen los componentes del vector $\vec{u}$:

$$v_{d_{m+1}} = \frac{2}{T_0}(\Delta x_m\, cos(\varphi_{m+1}) + \Delta y_m\, sin(\varphi_{m+1})) - v_m\, cos(\varphi_{m+1} - \varphi_m) + a\omega_m\, sin(\varphi_m - \varphi_{m+1}) \quad (2.55)$$

$$\omega_{d_{m+1}} = \frac{1}{a^2+1}\left(\frac{2}{T_0}\Delta\varphi_m - \omega_m + \frac{2a}{T_0}(\Delta y_m\, cos(\varphi_{m+1}) - \Delta x_m\, sin(\varphi_{m+1})) + av_m\, sin(\varphi_{m+1} - \varphi_m) - a^2\omega_m\, cos(\varphi_{m+1} - \varphi_m)\right) \quad (2.56)$$

Obtenidas las expresiones que permiten calcular la velocidad lineal (2.55) y la velocidad angular (2.56) se debe hallar la condición para que $\overrightarrow{b(N(A^T))} = \vec{0}$.

$$\overrightarrow{b(N(A^T))} = \vec{b} - \overrightarrow{b(ECA)} = \vec{0} \quad (2.57)$$

Como se vio en la sección 2.2.3 la condición para que se cumpla (2.57) es la misma para el modelo no holonómico mejorado como para el no mejorado del robot (Figura 1.3 cuando $a \neq 0$ y cuando $a = 0$). Por esta razón se halla la condición con $a = 0$ ya que resulta más directa obteniéndose el mismo resultado.

Reemplazando (2.55) y (2.56) en $\overrightarrow{b(ECA)}$ de (2.50) para luego sustituirlo en (2.48) junto con $\vec{b}$ de (2.46) con $a = 0$ se obtiene:

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix}\frac{2}{T_0}\Delta x_m - v_m\, cos(\varphi_m)\\ \frac{2}{T_0}\Delta y_m - v_m\, sin(\varphi_m)\\ \frac{2}{T_0}\Delta\varphi_m - \omega_m\end{bmatrix} - \left(v_{d_{m+1}}\right)\begin{bmatrix}cos(\varphi_{m+1})\\ sin(\varphi_{m+1})\\ 0\end{bmatrix} + \left(\omega_{d_{m+1}}\right)\begin{bmatrix}0\\0\\1\end{bmatrix} \tag{2.58}$$

Realizando las operaciones de (2.58) y simplificando se obtiene:

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix}\frac{2}{T_0}\Delta x_m - v_m\, cos(\varphi_m) - v_{d_{m+1}}\, cos(\varphi_{m+1})\\ \frac{2}{T_0}\Delta y_m - v_m\, sin(\varphi_m) - v_{d_{m+1}}\, sin(\varphi_{m+1})\\ \frac{2}{T_0}\Delta\varphi_m - \omega_m - \omega_{d_{m+1}}\end{bmatrix} \tag{2.59}$$

La ecuación (2.60) del sistema matricial de ecuaciones (2.59) no se toma para hallar la restricción, puesto que esta es la ecuación para el cálculo de la velocidad angular (2.56) cuando $a = 0$

$$\omega_{d_{m+1}} = \frac{2}{T_0}\Delta\varphi_m - \omega_m \tag{2.60}$$

Tomando las filas restantes de (2.58) se tiene el siguiente siguiente sistema de ecuaciones:

$$\begin{cases} \bullet\ \frac{2}{T_0}\Delta x_m - v_m\, cos(\varphi_m) - v_{d_{m+1}}\, cos(\varphi_{m+1}) = 0 & (2.61)\\ \bullet\ \frac{2}{T_0}\Delta y_m - v_m\, sin(\varphi_m) - v_{d_{m+1}}\, sin(\varphi_{m+1}) = 0 & (2.62)\end{cases}$$

Dividiendo (2.62) de (2.61) se obtiene:

$$\frac{\frac{2}{T_0}\Delta y_m - v_m\, sin(\varphi_m)}{\frac{2}{T_0}\Delta x_m - v_m\, cos(\varphi_m)} = \frac{v_{d_{m+1}}\, sin(\varphi_{m+1})}{v_{d_{m+1}}\, cos(\varphi_{m+1})} \tag{2.63}$$

Finalmente simplificando se obtiene la restricción para que (2.56) se satisfaga:

$$\varphi_{ez_m} = tan^{-1}\left(\frac{\Delta y_m \frac{2}{T_0} - v_m\, sin(\varphi_m)}{\Delta x_m \frac{2}{T_0} - v_m cos(\varphi_m)}\right) \tag{2.64}$$

Donde φ_{ez_m} (Figura 2.1) es el ángulo que hace que la condición $\overrightarrow{b(N(A^T))} = \vec{0}$ se cumpla.

De igual manera que al usar la aproximación de Euler, si se calculan las velocidades lineal $v_{d_{m+1}}$ y angular $\omega_{d_{m+1}}$ con la restricción de (2.64), se logra que el sistema $\vec{b} = A\vec{u}$ tenga solución, debido a que se asegura que $\vec{b}$ ahora sea igual a su proyección ortogonal en el espacio columna de A (es decir $\vec{b} = \overrightarrow{b(ECA)}$), que es el valor para el cual el error (1.21) es mínimo en la Figura 1.21.

En resumen para que el sistema descrito por (1.1) siga una trayectoria deseada utilizando la aproximación trapezoidal, las velocidades lineal y angular deben ser iguales a:

$$\begin{aligned} v_{d_{m+1}} &= \frac{2}{T_0}\left(\Delta x_m\, cos\left(\varphi_{ez_m}\right) + \Delta y_m\, sin\left(\varphi_{ez_m}\right)\right) - v_m\, cos\left(\varphi_{ez_m} - \varphi_m\right) \\ &\quad + a\omega_m\, sin\left(\varphi_m - \varphi_{ez_m}\right) \\ \omega_{d_{m+1}} &= \frac{1}{a^2+1}\left(\frac{2}{T_0}\Delta\varphi_m - \omega_m + \frac{2a}{T_0}\left(\Delta y_m\, cos\left(\varphi_{ez_m}\right) - \Delta x_m\, sin\left(\varphi_{ez_m}\right)\right)\right. \\ &\quad \left. + av_m\, sin\left(\varphi_{ez_m} - \varphi_m\right) - a^2\omega_m\, cos\left(\varphi_{ez_m} - \varphi_m\right)\right) \end{aligned} \tag{2.65}$$

Donde el ángulo de referencia φ_{ez_m} es igual a:

$$\varphi_{ez_m} = tan^{-1}\left(\frac{\Delta y_m \frac{2}{T_0} - v_m\, sin(\varphi_m)}{\Delta x_m \frac{2}{T_0} - v_m cos(\varphi_m)}\right) \tag{2.66}$$

2.4 CONTROLADOR PI

2.4.1 IMPLEMENTACIÓN DEL ALGORITMO DE CONTROL

Se implementa un PI en la práctica, pero se inicia a partir de la definición de PID, por lo que se hace referencia a la estructura más conocida de este control en Laplace (2.67), y se realiza la respectiva acción de discretización para poder ingresarlo al robot móvil.

$$\frac{U(s)}{E(s)} = k_p + k_i\frac{1}{s} + k_d s \tag{2.67}$$

Cada acción que presenta el PID se discretiza de manera diferente, de acuerdo a su naturaleza. Para la acción Integral se utiliza el método de integración trapezoidal o Tustin, que siempre toma un promedio entre el valor anterior y el actual [6].

$$s = \frac{2}{T_0}\left(\frac{z-1}{z+1}\right) \tag{2.68}$$

Para la acción derivativa se utiliza el método de integración hacia atrás, que toma directamente el valor anterior y el valor actual de la variable a ser controlada, [6].

$$s = \frac{1}{T_0}\left(\frac{z-1}{z}\right) \tag{2.69}$$

Reemplazando (2.68) y (2.69) en (2.67), y resolviendo se obtiene el PID en diferencias:

$$\frac{U(z)}{E(z)} = k_p + k_i\left(\frac{T_0}{2}\frac{z+1}{z-1}\right) + k_d\left(\frac{1}{T_0}\frac{z-1}{z}\right) \tag{2.70}$$

$$(z^2 - z)U(z) = E(z)\left(z^2\left(k_p + \frac{k_i T_0}{2} + \frac{k_d}{T_0}\right) + z\left(-k_p + \frac{k_i T_0}{2} - \frac{2k_d}{T_0}\right) + \left(\frac{k_d}{T_0}\right)\right) \tag{2.71}$$

$$(1 - z^{-1})U(z) = E(z)\left(\left(k_p + \frac{k_i T_0}{2} + \frac{k_d}{T_0}\right) + z^{-1}\left(-k_p + \frac{k_i T_0}{2} - \frac{2k_d}{T_0}\right) + z^{-2}\left(\frac{k_d}{T_0}\right)\right) \tag{2.72}$$

$$U(z) - z^{-1}U(z) = E(z)\left(k_p + \frac{k_i T_0}{2} + \frac{k_d}{T_0}\right) + E(z)z^{-1}\left(-k_p + \frac{k_i T_0}{2} - \frac{2k_d}{T_0}\right) + E(z)z^{-2}\left(\frac{k_d}{T_0}\right) \quad (2.73)$$

Aplicando la transformada Z inversa:

$$z^{-n}F(z) = F(k-n) \quad (2.74)$$

Se obtiene finalmente la ecuación en diferencias para la implementación del PID (2.76):

$$U(k) - U(k-1) = E(k)\left(k_p + \frac{k_i T_0}{2} + \frac{k_d}{T_0}\right) + E(k-1)\left(-k_p + \frac{k_i T_0}{2} - \frac{2k_d}{T_0}\right) + E(k-2)\left(\frac{k_d}{T_0}\right) \quad (2.75)$$

$$U(k) = U(k-1) + E(k)\left(k_p + \frac{k_i T_0}{2} + \frac{k_d}{T_0}\right) + E(k-1)\left(-k_p + \frac{k_i T_0}{2} - \frac{2k_d}{T_0}\right) + E(k-2)\left(\frac{k_d}{T_0}\right) \quad (2.76)$$

Con la expresión (2.76) se realiza la calibración de constantes, luego de tomar en cuenta las consideraciones del modelo detalladas en la siguiente sección.

2.4.2 CONSIDERACIONES DEL MODELO

Puesto que para el desarrollo del controlador PID se utiliza un modelo aproximado (sección 0), es necesario conocer algunas características del robot para luego realizar una calibración adecuada de constantes.

Para identificar estas características se aplican señales paso a las velocidades lineal y angular de la plataforma y se observa su respuesta en lazo abierto y en lazo cerrado.

2.4.2.1 Prueba a lazo abierto con velocidades mayores a las máximas permitidas por el robot [27] y sin controlador

El objetivo de esta prueba es conocer cuáles son las velocidades lineal y angular máximas permitidas por el robot y cuáles son los mejores tiempos de establecimiento que presentan la velocidad lineal y angular de la plataforma; de esta manera al momento de calibrar el PID se considera que no se pueden disminuir estos tiempos porque son intrínsecos del robot.

A lazo abierto se ingresan señales de velocidad superiores a las máximas sugeridas por el fabricante [30], y como se puede observar en la Figura 2.2, las magnitudes de las velocidades llegan a 1 (m/s) para la velocidad lineal y 1,745 (rad/s) para la velocidad angular. El tiempo de establecimiento se encuentra alrededor de los cuatro segundos para la velocidad lineal y alrededor de los dos segundos para la velocidad angular.

Al haber ingresado la máxima señal de velocidad se asegura que el tiempo de establecimiento es el mínimo que puede presentar la plataforma, por lo que sin importar la calibración del controlador que se implemente, los tiempos obtenidos se mantienen en el orden de los dos a cuatro segundos.

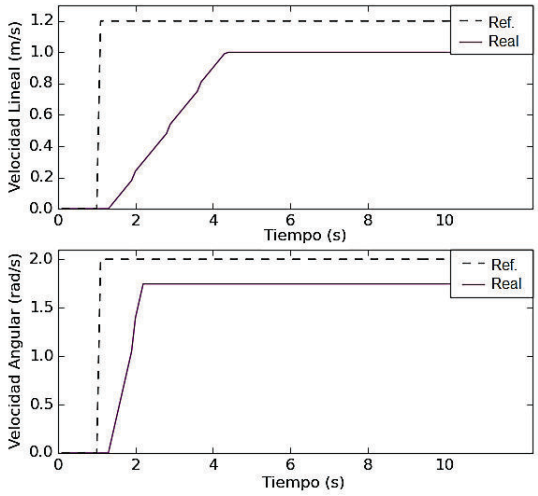


Figura 2.2. Velocidades lineal y angular del robot en lazo abierto ante una entrada paso de velocidades mayores a las máximas permitidas por la plataforma

2.4.2.2 Prueba a lazo abierto con velocidades inferiores a las máximas permitidas por el robot y sin controlador

En la prueba anterior se ingresaron señales paso de velocidad mayores a las velocidades permitidas por el robot, por lo que se obtuvo una respuesta restringida a las velocidades máximas de la plataforma. Ahora se repite el experimento ingresando señales paso de velocidad lineal y angular inferiores a las máximas soportadas por el robot (velocidad lineal de 0.5 (m/s) y velocidad angular de 1 (rad/s)) con el objetivo de comprobar si la respuesta de primer orden presenta algún sobretiro.

En la Figura 2.3 se nota que no existe sobretiro en la respuesta y que las velocidades llegan a la referencia en un tiempo similar al observado en la Figura 2.2 (para la velocidad lineal alrededor de los 3 segundos y para la angular alrededor de los 2 segundos).

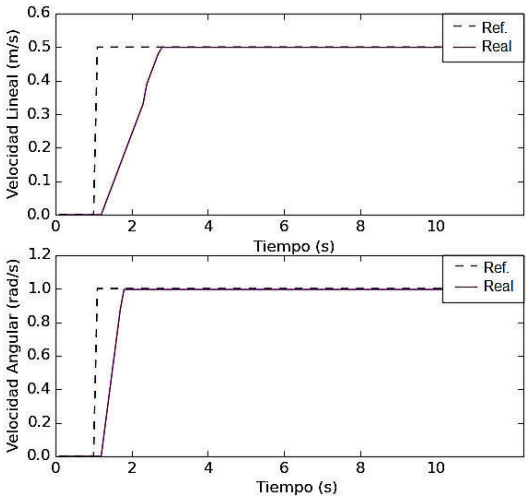


Figura 2.3. Velocidades lineal y angular del robot en lazo abierto ante una entrada paso de velocidades inferiores a las máximas permitidas por la plataforma

2.4.2.3 Prueba a lazo cerrado con velocidades inferiores a las velocidades máximas permitidas por el robot y sin controlador

Como se vio en la prueba anterior, al ingresar señales paso de velocidad lineal y angular en lazo abierto el robot alcanza fácilmente la referencia, pero como el controlador en lazo cerrado se aplica a las velocidades, es necesario realimentar estas señales (Figura 2.4).

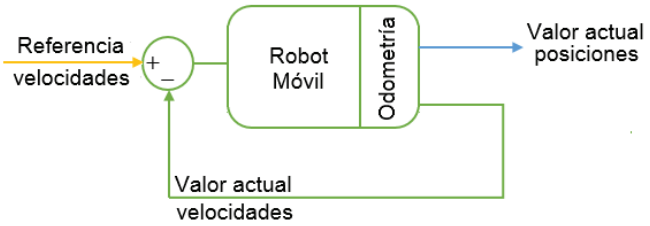


Figura 2.4. Representación del sistema en lazo cerrado sin controlador

Así, en lazo cerrado se ingresan señales paso de velocidad lineal y angular al robot y se obtiene la respuesta de la Figura 2.5, en donde se observa un error en estado estable de aproximadamente 50% y oscilaciones sostenidas.

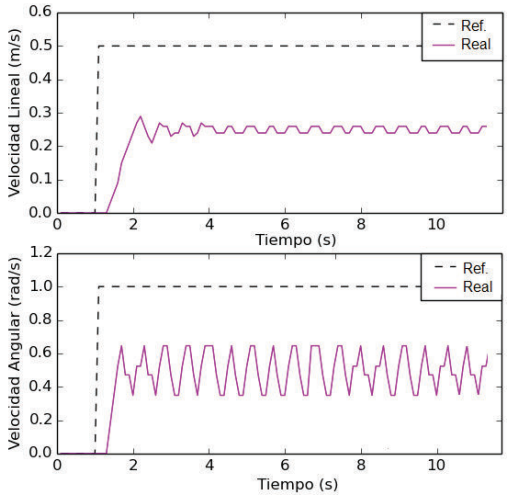


Figura 2.5. Velocidades lineal y angular del robot en lazo cerrado sin controlador ante una entrada paso

Con toda la información obtenida en estas pruebas, se implementa el controlador y se procede a realizar la calibración de sus constantes.

2.4.3 CALIBRACIÓN DEL CONTROLADOR

En la Figura 2.5 es evidente que el nuevo sistema realimentado no cumple con las condiciones para poder seguir las referencias de velocidad por lo que se hace necesario un controlador, cuyo objetivo es lograr que las velocidades lineal y angular alcancen la referencia. Así al lazo cerrado se agrega un controlador PID (2.76) tanto para la velocidad lineal v como para la velocidad angular ω (Figura 2.6).

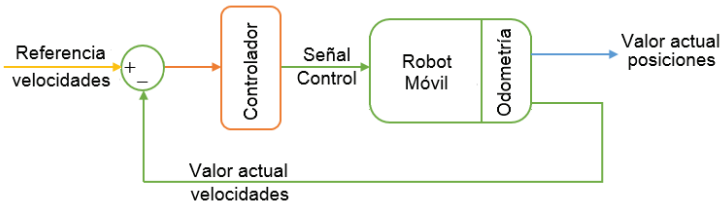


Figura 2.6. Representación del sistema en lazo cerrado con controlador

El propósito de este controlador es lograr reducir el error que existe entre la referencia y entre la salida en el menor tiempo posible, aun después de haberse producido una perturbación en el sistema.

Para realizar la calibración se propone un método a través del cual se pueda caracterizar el tiempo de respuesta del sistema.

$$\Phi = \int_0^\infty F[e(t), t]dt \tag{2.77}$$

Donde F resulta ser una función tanto del error como del tiempo y Φ es el criterio de comportamiento, que será cero solamente si el error es cero todo el tiempo; lo que es imposible debido a que el sistema suele empezar en condiciones diferentes a las referencias y además pueden haber perturbaciones que sacan al sistema de su punto de trabajo.

Debido a la diversidad de criterios que se pueden adoptar, hay diferentes funciones F que permiten definir la calibración, como el ISE (integral del error cuadrático), IAE (integral del error absoluto), ITAE (integral del absoluto del error por el tiempo), entre otros. [39]

En este caso se usa la integral del valor absoluto del error IAE como criterio de comportamiento, y está descrita por la ecuación (2.78).

$$IAE = \int_0^\infty |e(t)|dt \tag{2.78}$$

Este criterio es sensible a pequeños errores, pero no muy sensible a grandes errores. Además presenta una sencilla forma de aplicación y provee amortiguamiento y respuesta aceptables a la salida del lazo de control.

Se ha desarrollado una serie de pruebas experimentales para determinar las mejores constantes para el controlador, es decir, las constantes que provoquen el menor índice de error absoluto. El procedimiento utilizado el siguiente:

- Para el controlador se ha considerado únicamente parte proporcional e integral, pues la expresión derivativa hacía inestable al sistema.
- Se establece una relación entre las constantes del controlador (k_p y k_i) tomando como base la calibración de las constantes del PID interno que presenta el robot. Utilizando los datos provistos en el manual de operación ([31], pág. 40-41) se obtiene la siguiente proporción.

$$\frac{k_p}{k_i}:\frac{50}{10} \tag{2.79}$$

- Se establece un factor modulante (valor inicial igual a 1) por el cual se multiplican ambas constantes para obtener el valor final de k_p y k_i. Este factor se varía y con cada uno se obtiene el valor del IAE correspondiente.
- Se realizan las pruebas variando el factor en múltiplos de 10 y comparando los valores de IAE.
- Se restringen los valores de prueba hacia el segmento que presenta menor valor de IAE.
- Se elige el factor para el mínimo IAE obtenido y se hace un ajuste fino para obtener los valores finales de las constantes calibradas.

Los resultados de la calibración se muestran en forma gráfica para evidenciar que se obtiene el valor de las constantes que producen el mínimo IAE.

2.4.3.1 Resultados de la calibración

Tabla 2.2. Ensayos para calibración de velocidad lineal

Valores iniciales												
	k_p = 10		k_i = 50		**Factor =** 1		**IAE =** 0,311					

Ensayos												
Factor	10	1	0.5	0.1	0,05	0,035	0,03	0,025	0,022	0,02	**0,018**	0,01
IAE	0,470	0,311	0,302	0,270	0.124	0,0887	0,0826	0,0750	0,0691	0,0643	**0,0608**	0.1084

Valores finales									
	k_p = 0,2		k_i = 0,905		**Factor =** 0,018		**IAE =** 0,0608		

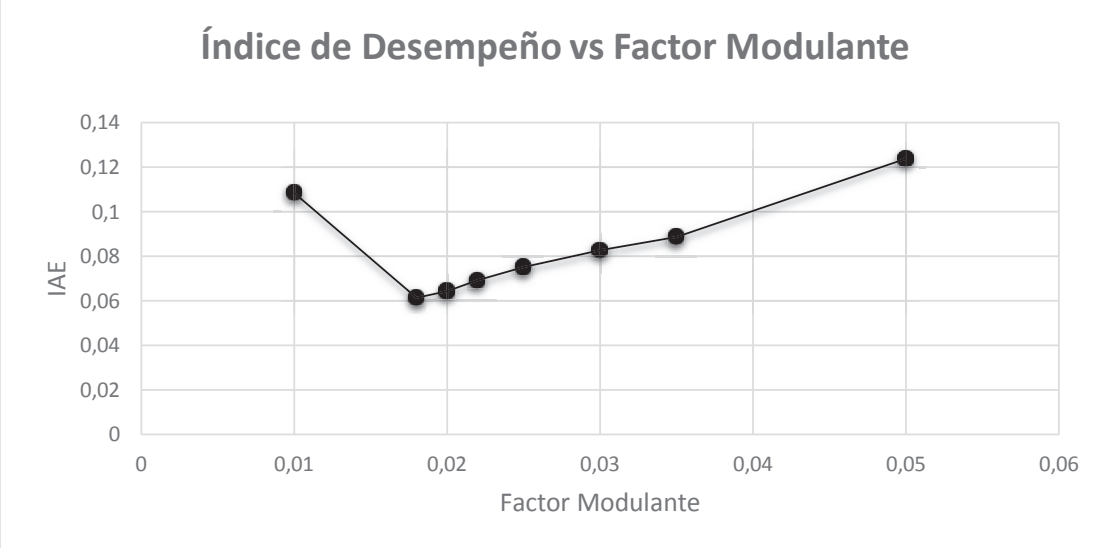


Figura 2.7. Índice de desempeño vs Factor modulante para velocidad lineal

Tabla 2.3. Ensayos para calibración de velocidad angular

Valores iniciales												
	k_p = 10		k_i = 50		**Factor =** 1		**IAE =** 0,8399					

Ensayos												
Factor	10	1	0.5	0.1	0,05	0,037	**0,0355**	0,035	0,033	0,03	0,01	10
IAE	0,968	0,840	0,815	0,223	0,0799	0,0670	**0,0654**	0,0670	0,0702	0,0770	0,2170	0,9675

Valores finales									
	k_p = 0,355		k_i = 1,775		**Factor =** 0,0355		**IAE =** 0,0654		

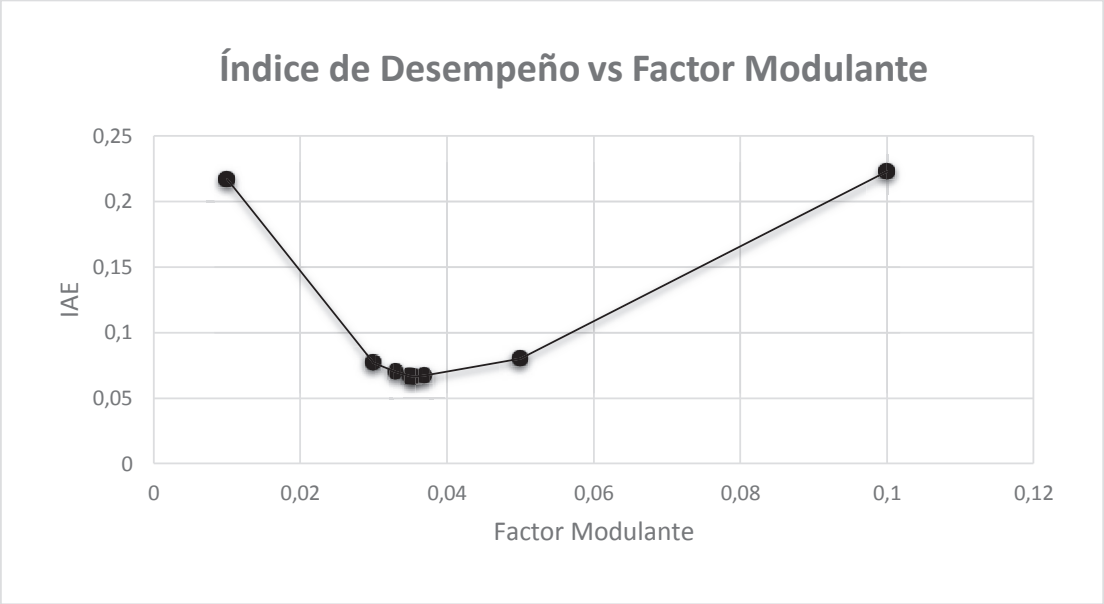


Figura 2.8. Índice de desempeño vs Factor modulante para velocidad angular

2.4.4 CRITERIOS PARA GENERACIÓN DE REFERENCIAS DE VELOCIDAD

El controlador que se plantea para las velocidades lineal y angular necesita tener referencias, pero esto genera problemas, pues se cuenta únicamente con referencias de la trayectoria.

Esto se resuelve utilizando los siguientes principios básicos:

- Se utilizan ecuaciones de movimiento rectilíneo uniforme para obtener las ecuaciones de referencia para las velocidades lineal (2.80) y angular (2.81), ya que las trayectorias de referencia se generan en lo posible a velocidad constante.

$$v_{ref_m} = \sqrt{\left(\frac{\Delta x_m}{T_0}\right)^2 + \left(\frac{\Delta y_m}{T_0}\right)^2} \tag{2.80}$$

$$\omega_{ref_m} = \frac{1}{T_0}(\Delta\varphi_m) \tag{2.81}$$

- Antes de que el robot alcance la trayectoria, los errores de trayectoria tienden a ser muy grandes, provocando así velocidades de referencia de iguales magnitudes. Esto es un problema debido al gran tiempo de muestreo con el

que se trabaja; y puesto que las rampas de velocidad son lentas se pueden producir oscilaciones una vez alcanzada la trayectoria. Por esta razón cuando se tienen errores de gran magnitud se introducen saturadores al sistema que se calibran a velocidades prudenciales dependiendo de la velocidad de la trayectoria a seguirse.

- Cuando se calcula la variación angular se debe tomar en cuenta el sentido de giro que debe realizar el robot, para esto se usa una solución por software (sección 3.1.6).

2.4.5 REPRESENTACIÓN COMPLETA DEL CONTROL – MODELO

Como se explica en la sección 2.4.3 para realizar la calibración de constantes se utiliza solamente la parte proporcional e integral del PID (2.82), pues la parte derivativa provocaba inestabilidad en el sistema:

$$U(k) = U(k-1) + E(k)\left(k_p + \frac{k_i T_0}{2}\right) + E(k-1)\left(-k_p + \frac{k_i T_0}{2}\right) \tag{2.82}$$

Se debe tener en cuenta que el controlador se aplica a las velocidades por lo que no hay interacción directa de las componentes de la trayectoria, sin embargo las mismas se encuentran implícitas en la generación de las referencias de velocidad tal como se vio en la sección 2.4.4.

Una vez creadas estas referencias, las mismas se realimentan con la velocidad actual del robot y se calculan los controladores que cumplan con el propósito de que las velocidades lineal y angular del robot alcancen las referencias de velocidad creadas a partir de los errores de trayectoria del sistema.

En el diagrama de bloques de la Figura 2.9 se observa la interacción completa esquematizada, considerando todas las variables involucradas.

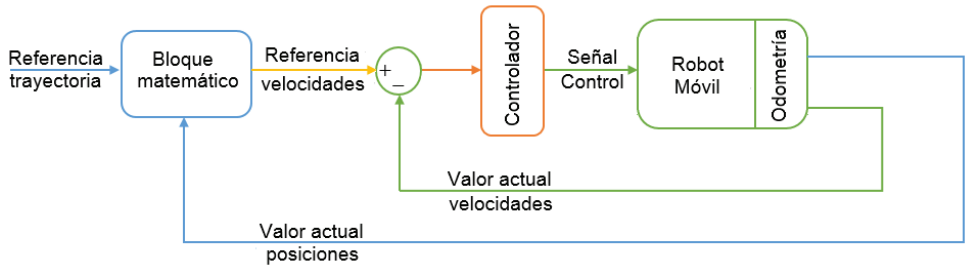


Figura 2.9. Representación completa de la obtención de referencias y aplicación de control PI

El bloque matemático de la Figura 2.9 se encarga de calcular las referencias de velocidades a partir de los errores de trayectoria. En la Figura 2.10 se puede apreciar una diagrama de bloques de la forma matemática de como se ha implementado.

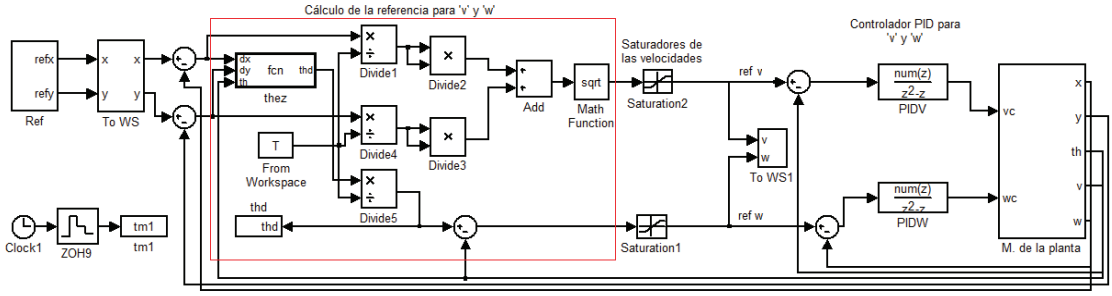


Figura 2.10. Representación matemática completa de la obtención de referencias y aplicación de control

El controlador descrito por (2.82) se aplica a las velocidades lineal y angular de la plataforma, y una vez sintonizados estos controladores, se obtiene una respuesta como la que se aprecia en la Figura 2.11 frente a una entrada paso.

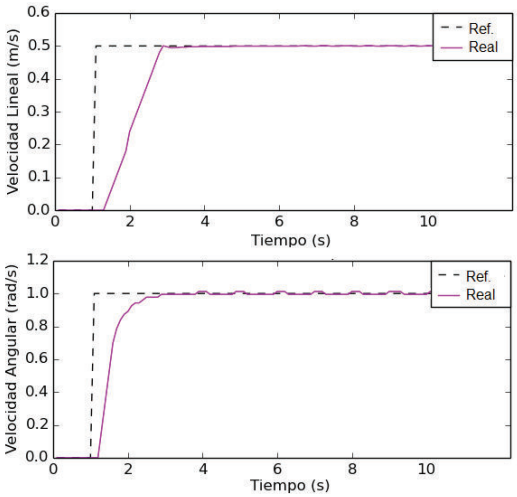


Figura 2.11. Velocidades lineal y angular del robot en lazo cerrado con controlador ante una entrada paso

En la Figura 2.11 se nota que al introducir el controlador la respuesta del sistema mejora, pues ya alcanza la referencia y las oscilaciones disminuyen su magnitud considerablemente, aunque el tiempo de establecimiento es similar al presentado en pruebas anteriores (para la velocidad lineal alrededor de los 3 segundos y para la angular alrededor de los 2 segundos).

2.5 CONTROLADOR POR MODO DESLIZANTE

2.5.1 IMPLEMENTACIÓN DEL ALGORITMO DE CONTROL

El primer paso en control por modo deslizante o SMC es definir una superficie $\dot{S}(t) = 0$ sobre la cual el proceso pueda deslizarse hasta el valor final deseado. La superficie de deslizamiento divide al plano en regiones donde la función switching tiene diferentes signos.

Hay varias opciones para escoger una superficie de deslizamiento, pero la seleccionada en este caso es una superficie tipo PID, por ser muy conocida, por cumplir todos los criterios de convergencia de una superficie de deslizamiento [28] y ser relativamente sencilla de implementar (2.83).

$$S(t) = \frac{de(t)}{dt} + \lambda_1 e(t) + \lambda_0 \int_o^t e(t) \tag{2.83}$$

Una vez elegida la superficie de deslizamiento, el siguiente paso es reemplazar al error como la diferencia entre la referencia y el valor actual de la variable (2.84).

$$S(t) = \frac{d(R(t) - X(t))}{dt} + \lambda_1 (R(t) - X(t)) + \lambda_0 \int_o^t (R(t) - X(t)) \tag{2.84}$$

El objetivo es que el proceso siga en todo momento a la superficie de deslizamiento, es decir que se mantenga en todo momento con una pendiente igual cero sobre ella. Para que el sistema logre esto la derivada de la superficie debe ser igual a cero (2.85).

$$0 = \frac{dS(t)}{dt} = \frac{d^2(R(t) - X(t))}{dt^2} + \lambda_1 \frac{d(R(t) - X(t))}{dt} + \lambda_0 (R(t) - X(t)) \tag{2.85}$$

Se desprecian las derivadas de la referencia, pues Camacho [27] ha demostrado que pueden ser descartadas sin ningún efecto en el rendimiento del controlador.

$$\frac{dS(t)}{dt} = -\frac{d^2\big(X(t)\big)}{dt^2} - \lambda_1 \frac{d\big(X(t)\big)}{dt} + \lambda_0\big(R(t) - X(t)\big) = 0 \tag{2.86}$$

El SMC tal como se vio en la sección 1.5.5 está constituido por dos partes, una continua o equivalente y una discontinua o switching que se desarrollan a continuación.

2.5.1.1 Función continua o equivalente

Para obtener la expresión que corresponde a la parte continua se resuelve la ecuación (2.86) para su derivada de mayor grado y se reemplaza en la ecuación (2.87) tal como propone Camacho [27]:

$$\frac{1}{K}\left(t_0\tau \frac{d^2\big(X(t)\big)}{dt^2} + (\tau + t_0)\frac{d\big(X(t)\big)}{dt} + X(t)\right) = U_C(t) \tag{2.87}$$

$$U_C(t) = \frac{t_0\tau}{K}\left(\left(-\lambda_1 + \frac{\tau + t_0}{t_0\tau}\right)\frac{d\big(X(t)\big)}{dt} + \lambda_0 E(t) + \frac{X(t)}{t_0\tau}\right) \tag{2.88}$$

Los parámetros λ_1 y λ_2 definen el comportamiento de la superficie y por ende de como el sistema tiende hacia la referencia. Para simplificar (2.88) se realiza la elección de λ_1 (2.89) con el cual se ha demostrado en Camacho [27] que es la mejor elección para que la parte continua del controlador tenga un comportamiento estable. Adicionalmente para asegurar que la superficie de deslizamiento se comporte como un sistema críticamente amortiguado o sobreamortiguado se selecciona (2.90).

$$\lambda_1 = \frac{\tau + t_0}{t_0\tau} \tag{2.89}$$

$$\lambda_0 \leq \frac{{\lambda_1}^2}{4} \tag{2.90}$$

Se reemplazan (2.89) y (2.90) en (2.88) para obtener la expresión final de la parte continua del controlador:

$$U_C(t) = \frac{t_0\tau}{K}\left(\lambda_0\big(R(t) - X(t)\big) + \frac{X(t)}{t_0\tau}\right) \tag{2.91}$$

2.5.1.2 Función discontinua o switching

Para la sección discontinua del controlador se debería tomar en cuenta la salida de un controlador tipo PID (superficie de deslizamiento elegida), sin embargo en el presente trabajo la salida a considerarse es una tipo PI, lo que se realiza por dos razones muy importantes:

- La parte derivativa torna inestable al sistema con el que se está trabajando provocando oscilaciones en el mismo, como se explicó en 2.4.3.
- La función discontinua del controlador por modo deslizante ya cumple las funciones de la acción derivativa del PID (que el sistema alcance la referencia con rapidez y que reaccione más rápido antes los cambios). Así, la función switching del controlador prácticamente reemplaza el término derivativo del controlador y se implementa un PI en la práctica.

Considerando a la superficie deslizante ahora como un PI (2.92) y considerando la suavización de la función switching presentada en 1.5.5.3, se obtiene (2.94):

$$S(t) = U_{PI} \tag{2.92}$$

$$U_D(t) = K_D \frac{S(t)}{|S(t)| + \delta} \tag{2.93}$$

$$U_D(t) = K_D \frac{U_{PI}(t)}{|U_{PI}(t)| + \delta} \tag{2.94}$$

2.5.1.3 Expresión completa para el controlador por modo deslizante

Finalmente, para obtener el controlador completo se suman las secciones continua y discontinua del mismo (2.96).

$$U(t) = U_C(t) + U_D(t) \tag{2.95}$$

$$U(t) = \frac{X(t)}{K} + \frac{t_0 \tau}{K} \lambda_0 (R(t) - X(t)) + K_D \frac{U_{PI}(t)}{|U_{PI}(t)| + \delta} \quad (2.96)$$

La expresión (2.96) sirve tanto para la velocidad lineal como para la angular. Este controlador se basa en la salida del PI previamente calibrado en la sección 2.4.3, el valor de referencia y el valor actual de la variable para el cálculo de la salida del controlador.

2.5.2 CONSIDERACIONES DEL MODELO

Modo deslizante es un control basado en modelo, por lo que se debe conocer el modelo matemático de la planta para el cálculo del controlador, lo que hace más complicado su desarrollo pues no siempre se tiene acceso al modelo completo del sistema que se debe controlar.

Por esta razón para el desarrollo de este proyecto se hace uso de la solución presentada en [27] a este problema, que es aproximar la planta a un modelo más simple y calcular el controlador en torno a este modelo aproximado. El modelo usado para el desarrollo del controlador por modo deslizante es el mismo usado en el control PI, que corresponde al de la sección 0.

Se puede realizar esta aproximación del modelo debido a que el control por modo deslizante es un control robusto [27], que logra compensar imprecisiones de modelado, siempre y cuando el modelo aproximado usado no diste mucho del exacto. Esta propiedad resulta útil cuando el modelo matemático de la planta es muy complicado o cuando se desconoce el modelo del sistema a controlar.

Como se ve en 2.5.1 la superficie de deslizamiento elegida para el controlador es una superficie tipo PID (con derivativa igual a cero), y por ello se denomina al control por modo deslizante como un control PID robusto [27].

Los principales objetivos de hacer robusto al PID usando el controlador por modo deslizante son los siguientes [12]:

- Compensar los errores de modelado.
- Asegurar mayor estabilidad frente a perturbaciones externas.

- Usar la parte discontinua o switching para alcanzar la referencia a mayor velocidad de manera más estable.

Al usar el mismo modelo que el controlador PI, los problemas a resolver son los mismos para ambos controladores, sin embargo la representación en diagrama de bloques del modelo varía incluyendo ahora un bloque que toma los resultados del control PI para el cálculo del control por modo deslizante (Figura 2.12).

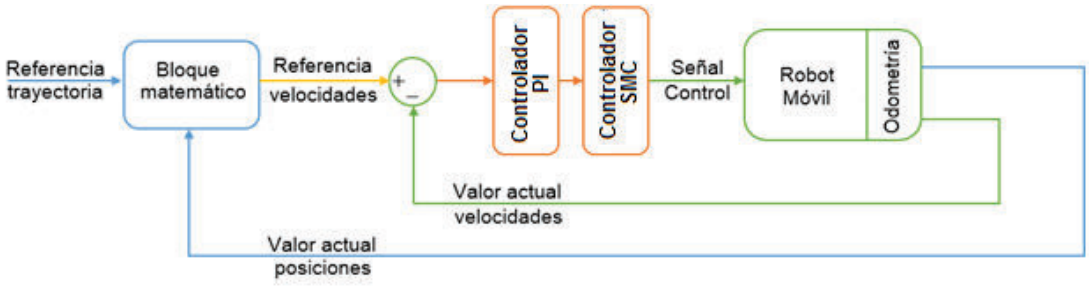


Figura 2.12. Representación completa de la obtención de referencias y aplicación de control SMC.

2.5.3 CALIBRACIÓN DEL CONTROLADOR

Las ecuaciones de calibración para los parámetros de la parte continua del controlador y la superficie de deslizamiento, como se evidencia en la sección 2.5.1.1, son las siguientes:

$$\lambda_1 = \frac{\tau + t_0}{t_0 \tau} \tag{2.97}$$

$$\lambda_0 \leq \frac{{\lambda_1}^2}{4} \tag{2.98}$$

Donde:

$$\tau \approx 66\%\, T$$

τ : constante de tiempo del sistema

T : tiempo de subida del sistema

t_0 : tiempo muerto o retardo

Para el modelo del robot móvil de tracción diferencial el procedimiento para hallar la constante de tiempo parte de su respuesta normal con control PI, donde se mide

el tiempo de subida T, la ganancia en estado estable K y el tiempo muerto t_0, a partir de allí se obtiene las constantes necesaria para hallar los parámetros del controlador.

Además de estas consideraciones iniciales, para la calibración de la parte discontinua del controlador se utiliza un procedimiento para minimizar el índice de desempeño ISE basado en el algoritmo de búsqueda de Nelder-Mead [10] y que resulta en las siguientes ecuaciones:

$$K_D = \frac{0.51}{|K|}\left[\frac{\tau}{t_0}\right]^{0.76} \tag{2.99}$$

$$\delta = 0.68 + 0.12(KK_D\lambda_1) \tag{2.100}$$

Donde:

- K_D : parámetro de ajuste responsable por la velocidad con la que se alcanza la velocidad de deslizamiento
- K : ganancia en estado estable del sistema
- τ : constante de tiempo del sistema
- t_0 : tiempo muerto o retardo
- δ : parámetro de ajuste de la sigmoide

Las ecuaciones (2.97), (2.98), (2.99) y (2.100) se toman de Camacho [27], donde se desarrollan para procesos químicos que son aproximados a funciones de primer orden más tiempo muerto (FOPDT) al igual que el robot móvil. Por este motivo es viable trabajar con estas ecuaciones teniendo en cuenta que una vez usadas, los valores calculados necesitaran una calibración manual adicional para que el controlador pueda trabajar correctamente.

CAPÍTULO 3

IMPLEMENTACIÓN DE LOS ALGORITMOS DE CONTROL

El objetivo del capítulo tres es presentar como la información es intercambiada a través del computador maestro, el computador esclavo y el robot móvil, además de mostrar en que sección de la implementación se encuentra el cálculo de los controladores, el envío de los datos, la evasión de obstáculos, las etapas de funcionamiento de los programas, entre otros.

Los primeros diagramas muestran el flujo general de los datos a través de los distintos nodos, y mientras se sigue avanzando en el capítulo los diagramas reflejan en detalle cuáles son los datos que circulan, las etapas en la que se transmiten y las secciones donde se realiza el cálculo de los datos que están viajando del robot a la computadora maestro, a la computadora esclavo y viceversa.

3.1 ROS - PYTHON PARA IMPLEMENTACIÓN DE ALGORITMOS

Una de las principales ventajas del uso de nodos en ROS es que los mismos se ejecutan en paralelo y se comunican entre ellos simultáneamente. Gracias a esto se pueden calcular las acciones de control, enviar y recibir datos de la interfaz, y obtener los datos provenientes de los sensores con los que cuenta el robot a la vez.

En la Figura 3.1 se representan los cuatro nodos con los que se trabajó a manera de nubes, de los cuales tres se ejecutan en la computadora maestro y uno en la computadora esclavo, además se representan los siete tópicos usados, encerrados en rectángulos.

Se crearon los nodos Acumulador, Interfaz y Control que cuentan con un solo tópico de salida, mientras que el nodo RosAria es provisto por el fabricante y cuenta con diez tópicos, de los cuales se usa uno de entrada y tres de salida.

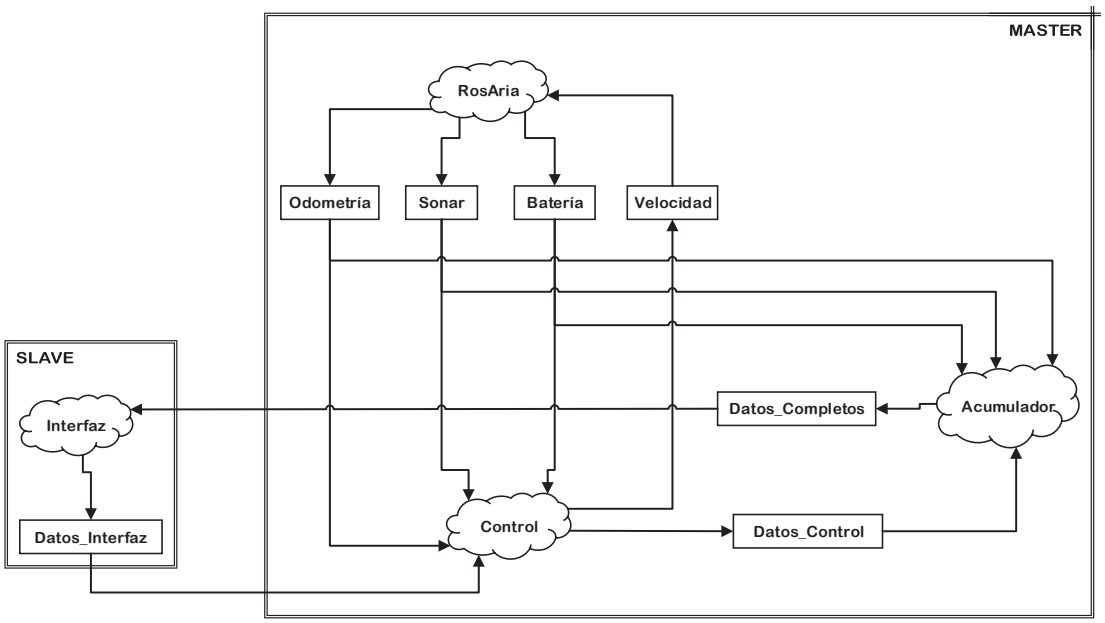


Figura 3.1. Representación de los cuatro nodos y siete tópicos utilizados en ROS

Los nodos usan los tópicos para comunicarse entre sí; estos tópicos envían la información de un nodo hacia otro que se encuentre corriendo usando el mismo núcleo de ROS sin importar que se encuentra en la misma computadora o no.

El nodo RosAria tiene dos funciones:

- Recibir la información de los sensores del robot, realizar los respectivos cálculos para obtener los datos de odometría, presencia de obstáculos y estado de la batería de la plataforma, y publicar estos datos en los respectivos tópicos de salida.
- Leer los datos del tópico Velocidad, transformar los valores de velocidad angular y lineal del robot en velocidad angular de los motores y finalmente enviar estos datos al robot.

El nodo Control realiza dos acciones:

- Calcula las acciones de control usando los datos provistos por los tópicos del nodo RosAria y los valores de referencia que obtiene del tópico Datos_Interfaz que se encuentra en el nodo Interfaz.
- Publica parte de los datos que más adelante se visualizan en la interfaz en el tópico Datos_Control que se encuentra entregando información al nodo Acumulador.

El nodo Acumulador tiene una única función, la cual es recoger la información de los tópicos presentes en el Máster para ordenarlos y publicarlos en el tópico Datos_Completos; estos datos deben presentarse en la interfaz y llegan a la computadora esclavo mediante el nodo Interfaz.

El nodo Interfaz tiene dos funciones:

- Publicar los parámetros ingresados por el usuario en la interfaz en el tópico Datos_Interfaz, esto se hace leyendo los archivos de texto generados por QtCreator para posteriormente transformar y ordenar esta información para ser publicada.
- Recoger la información del tópico Datos_Completos y transformarlos a un formato legible para ser leído por el programa que realiza la interfaz, finalmente esta información es almacenada en forma de vectores para su visualización.

En los siguientes diagramas de flujo se detalla cómo cada nodo creado realiza las acciones antes mencionadas, además de en qué momento la información va de un nodo a otro.

3.1.1 NODO INTERFAZ

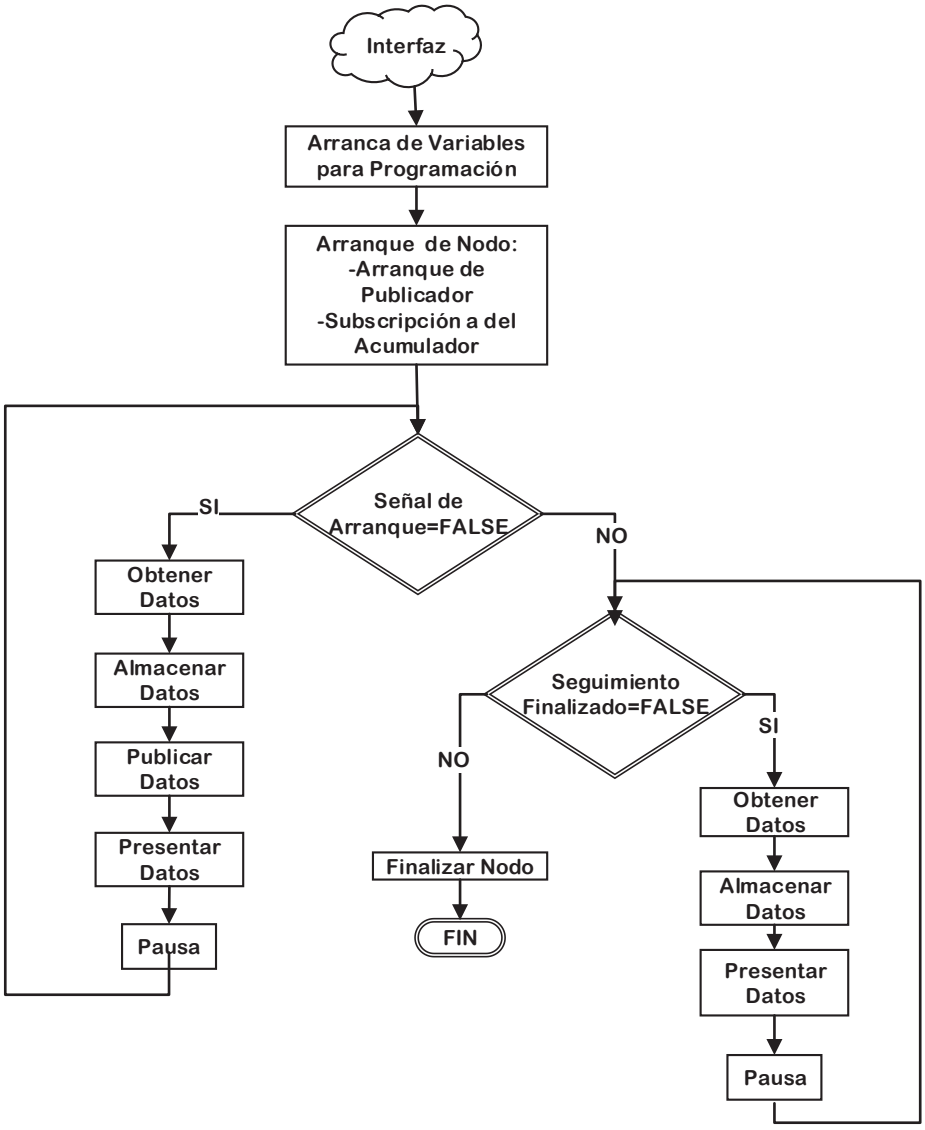


Figura 3.2. Diagrama de flujo del nodo Interfaz

Este nodo consta de tres etapas de funcionamiento, la primera etapa es común para todos los nodos así que solamente se la detalla en esta sección y más adelante solo es mencionada como etapa de arranque.

3.1.1.1 Etapa de arranque (general para todos los nodos)

La primera etapa de cualquier nodo es el arranque de sus funciones, para esto primero se definen los tópicos a publicarse y los tópicos a los cuales el nodo se

suscribe. En esta publicación/suscripción se define el tipo de trama y el tipo de dato con el que van a trabajar los tópicos del nodo.

Luego se arrancan las variables necesarias para el trabajo del programa y sus subfunciones, y allí el nodo empieza su funcionamiento leyendo información de las suscripciones automáticamente en el momento que estas lleguen.

3.1.1.2 Segunda etapa del funcionamiento del nodo interfaz

La segunda etapa empieza cuando desde la interfaz se ordena enviar los parámetros (entre ellos la orden de arranque) del esclavo al maestro. El proceso es tomar los datos de la interfaz en forma de archivos de texto que QtCreator ha generado, transformarlos en la trama y tipo de dato adecuados para el tópico de salida y publicarlos.

3.1.1.3 Última etapa del funcionamiento del nodo interfaz

La última etapa comienza cuando el robot confirma la orden de arranque y empieza su movimiento, enseguida el nodo deja de enviar información al control y comienza a recibir los datos del tópico de entrada, los almacena en archivos de texto y los presenta en la terminal de Ubuntu tal como se ve en la Figura 3.3.

Finalizado el seguimiento de la trayectoria este nodo se cierra esperando ser abierto de nuevo por la interfaz cuando la misma necesite reiniciar sus funciones.

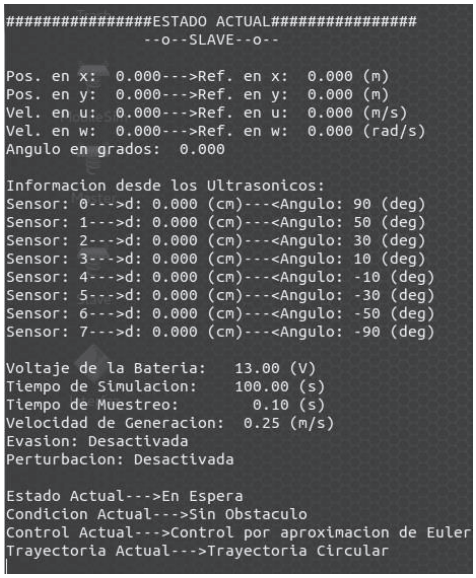


Figura 3.3. Interfaz en el terminal del ordenador esclavo

3.1.2 NODO ACUMULADOR

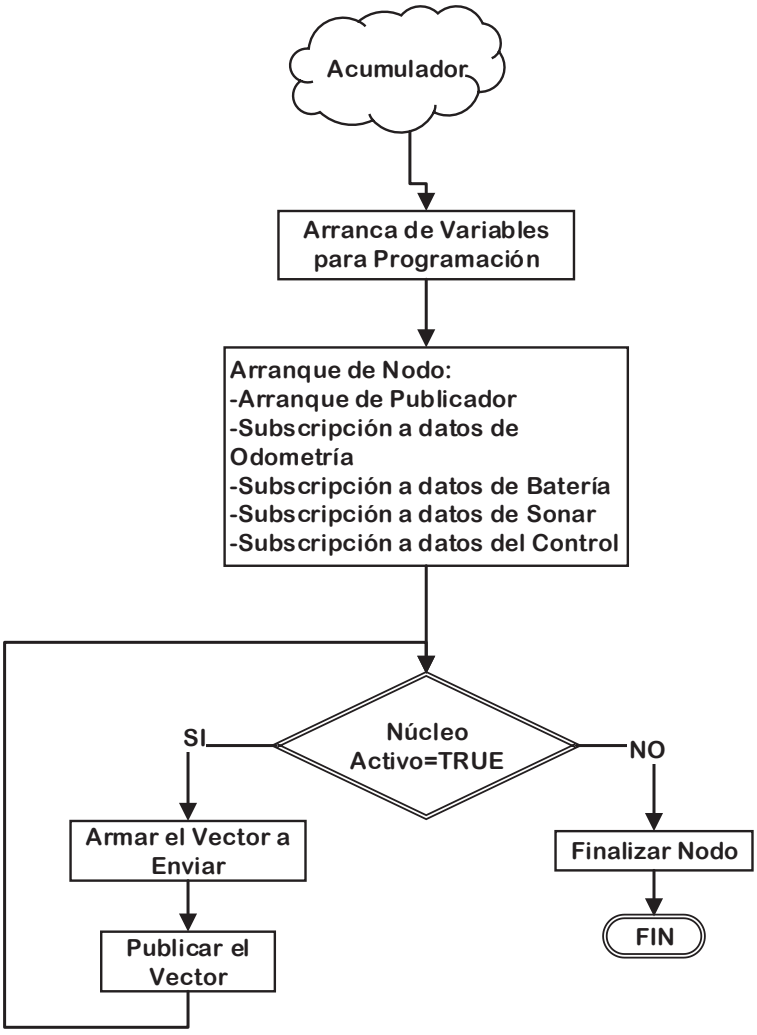


Figura 3.4. Diagrama de flujo del nodo de acumulación

El nodo Acumulador consta de dos etapas, la primera de ellas es la etapa de arranque descrita en la anterior sección, que comienza en conjunto con las etapas de arranque de los demás nodos pues todos los nodos existentes en el maestro se arrancan de manera simultánea.

3.1.2.1 Segunda etapa del funcionamiento del nodo acumulación

En la segunda etapa el nodo cumple con la única función de recolectar los datos de los tópicos del nodo RosAria y el tópico Datos_Control. La información se entrama en un vector y es publicada en su tópico de salida, esto lo hace desde que arranca

sus funciones hasta que sus funciones finalizan junto con los demás nodos existentes en el Máster.

3.1.3 NODO DE CONTROL

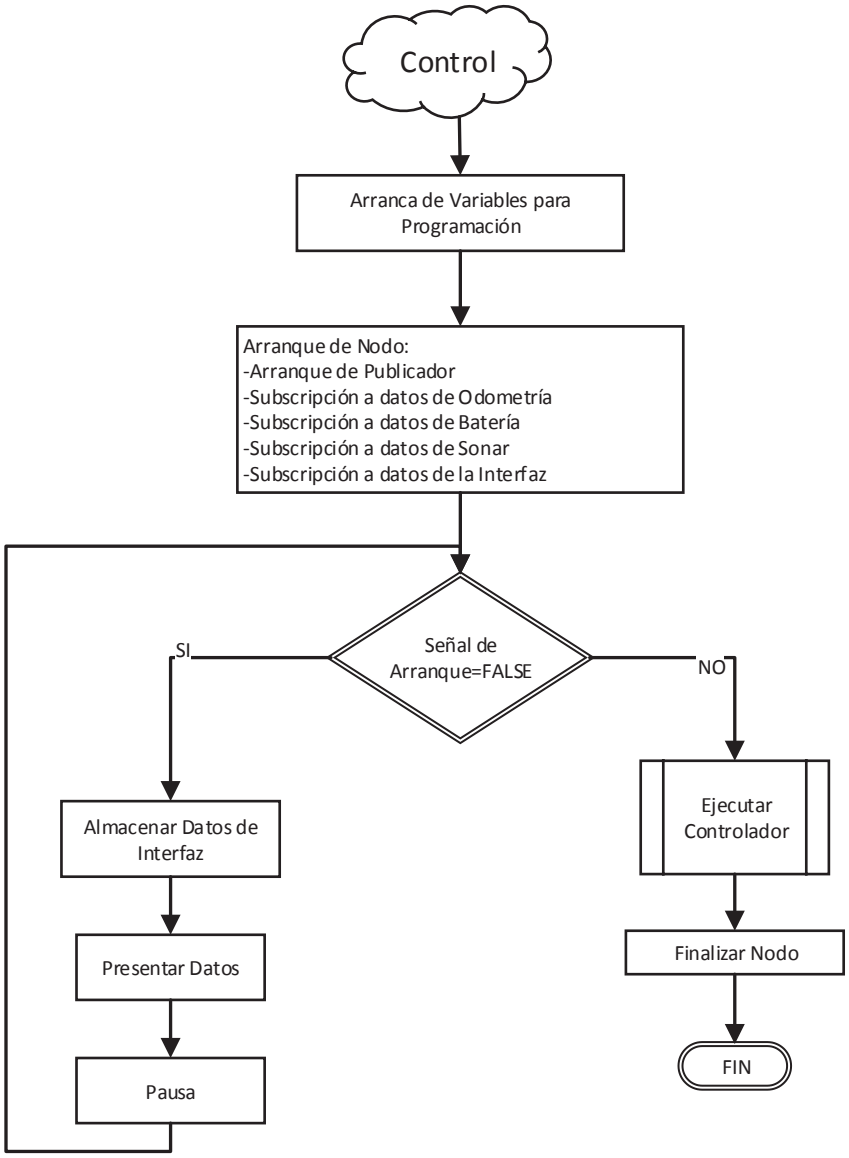


Figura 3.5. Diagrama de flujo del nodo de control

El nodo de control es el más importante para este trabajo, debido a que aquí se encuentran las funciones que se encargan de que el robot siga las trayectorias dependiendo de los parámetros que se han ingresado en la interfaz; y los algoritmos de control se adaptan para ingresar perturbaciones o evadir obstáculos.

Este nodo al igual que el nodo Interfaz cuenta con tres etapas, siendo la primera de ellas la etapa ya mencionada de arranque de nodo.

3.1.3.1 Segunda etapa del funcionamiento del nodo control

Aquí el nodo se encuentra esperando los datos de la interfaz que llegan por el tópico Datos_Interfaz; y mientras se encuentra en esta etapa almacena los datos que han llegado en un archivo de texto.

3.1.3.2 Tercera etapa del funcionamiento del nodo control

La tercera etapa empieza cuando desde la interfaz se envía la señal de arranque, esta acción se ve reflejada en el maestro cuando la presentación de la terminal cambia el tipo de información a mostrarse tal como se ve en la Figura 3.6.

Esta etapa es la más importante debido a que aquí es donde se ejecuta la función Control (junto con todas sus subfunciones), misma que es la encargada de realizar el seguimiento de trayectoria y evasión de obstáculos.

Mientras se ejecuta esta función el nodo Control deja de recibir datos desde la interfaz y trabaja con los últimos datos almacenados. También se empaqueta en un vector la información usada en el control (referencias, estado de evasión y perturbaciones) y lo envía al nodo Acumulador por medio del tópico Datos_Control.

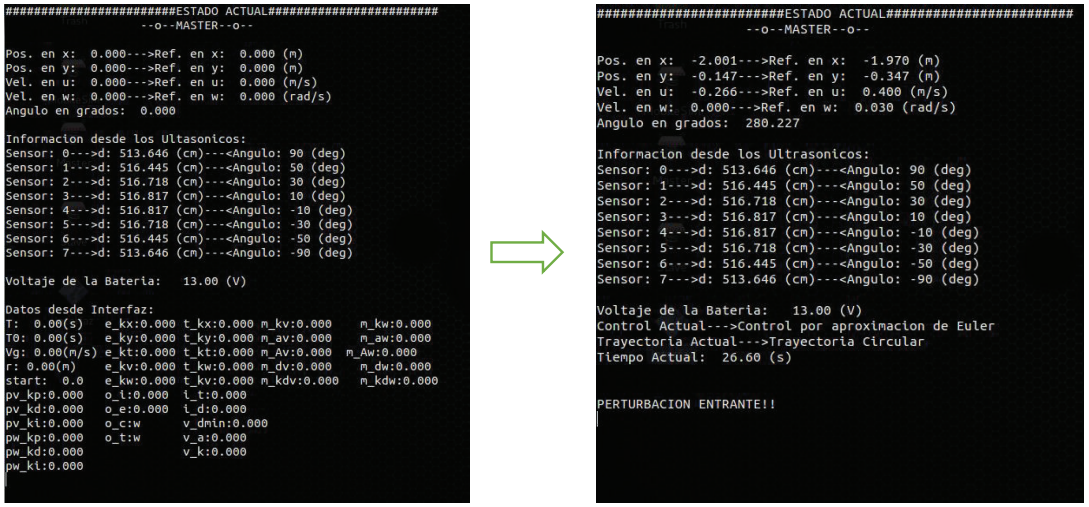


Figura 3.6. Cambio de información presentada en la terminal de Ubuntu después de haber enviado la señal de arranque

3.1.4 FUNCIÓN CONTROL

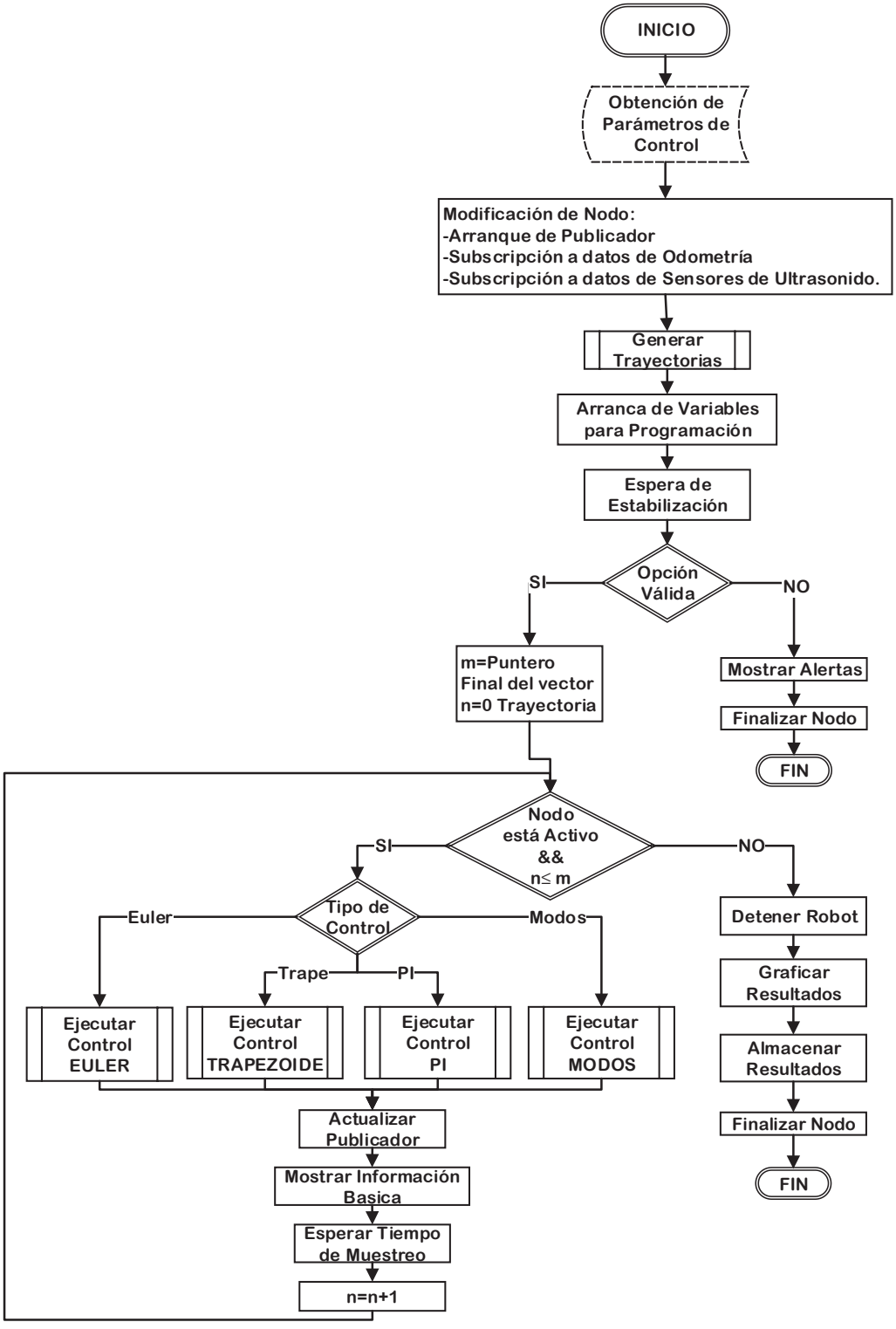


Figura 3.7. Diagrama de flujo de la función control

El objetivo de esta función es realizar el seguimiento de trayectoria, junto con la evasión de obstáculos y la introducción de perturbaciones al sistema. Para esto en su algoritmo se encuentran dos subfunciones muy importantes: una que genera las trayectorias y otra que calcula las acciones de control en cada tiempo de muestreo.

Para detallar esta función se la puede dividir en 5 etapas:

3.1.4.1 Primera etapa de la función control

La primera etapa es una modificación de nodo, donde esta función genera el tópico a usarse en el nodo Control y también se suscribe a los tópicos sonar y odometría del nodo RosAria.

3.1.4.2 Segunda etapa de la función control

En la segunda etapa se llama a la subfunción Generar Trayectorias, la cual usa los parámetros que llegan desde la interfaz para generar dos vectores donde se encuentran los componentes x y y de la trayectoria. Los cálculos usados para la generación de estos vectores se detallaron en la sección 2.1.

3.1.4.3 Tercera etapa de la función control

La tercera etapa realiza la validación de los parámetros desde los archivos de texto que se obtuvieron desde la interfaz, y a continuación se hace una pausa hasta que se terminen de inicializar los demás nodos arrancados en paralelo para luego continuar con el control.

3.1.4.4 Cuarta etapa de la función control

La cuarta etapa es la más importante debido que aquí se ejecuta la subfunción Controlador que es donde se calculan las leyes de control. Esta subfunción es ejecutada tantas veces como elementos tengan los vectores que contienen los componentes de la trayectoria en intervalos de tiempo de un tiempo de muestreo.

Una vez que se ejecuta la subfunción se toman los datos que esta usó para el cálculo de las leyes de control y datos relevantes para la interfaz como las referencias de posición, las referencias de velocidades, el estado de evasión, el estado del seguimiento y estado de la perturbación; y se publican en el tópico de

salida del nodo Control y son presentados en la terminal de Ubuntu tal como se ven en la Figura 3.6.

3.1.4.5 Quinta etapa del funcionamiento de la función control

La quinta y última etapa tiene lugar cuando todos los elementos de los vectores trayectoria han sido usados para la ejecución de la subfunción Controlador. Aquí todos los datos acumulados en el seguimiento son empleados para generar los gráficos de resultados, mismos que son almacenados para su posterior uso.

3.1.5 SUBFUNCIÓN CONTROLADOR

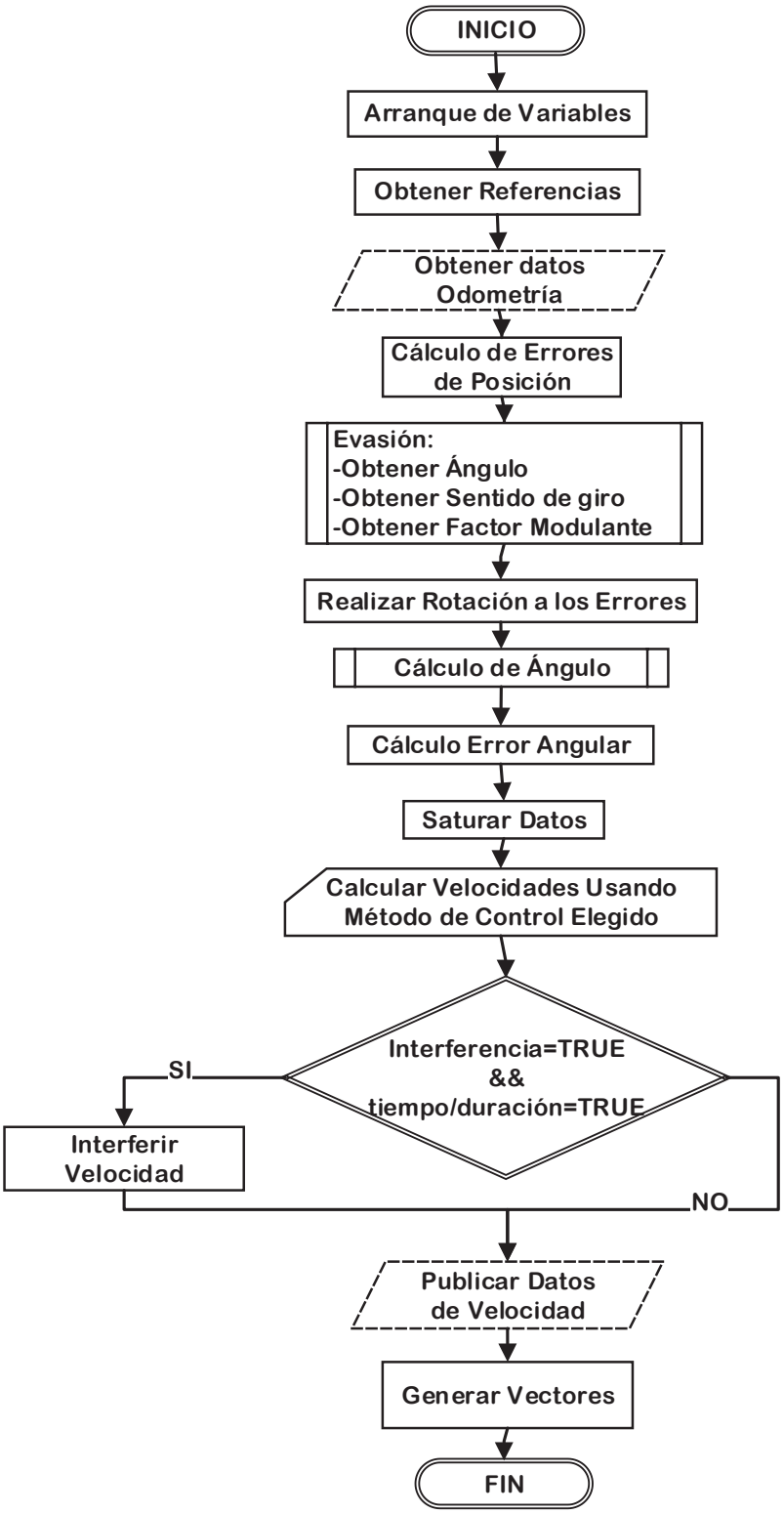


Figura 3.8. Diagrama de flujo de la subfunción controlador

Esta subfunción se encarga del cálculo de las velocidades angular y lineal para el seguimiento de las trayectorias. Este algoritmo es común para todos los controles diferenciándose solamente en las ecuaciones necesarias para obtener los valores a enviarse al robot.

El procedimiento que sigue esta subfunción es el siguiente:

- En primer lugar las variables son inicializadas, se obtienen los valores necesarios de referencia y odometría para el cálculo de los errores.
- Luego se llama a la subfunción Evasión, de donde se obtienen los datos para la rotación de los errores y el factor modulante para el freno de evasión.
- Obtenidos estos parámetros se realiza la rotación de los errores, y con ellos se calcula el ángulo de referencia con la subfunción Cálculo del ángulo para finalmente calcular el error angular.
- Ahora que se tienen todos los errores se calculan las velocidades angular y lineal.
- En el siguiente paso se ingresan perturbaciones al sistema si la opción fue elegida en la interfaz.
- Finalmente los valores de velocidad lineal y angular se envían al nodo RosAria por medio del tópico Velocidad, y se retorna los valores usados en el cálculo para el envío al nodo Acumulador.

3.1.6 SUBFUNCIÓN CÁLCULO DEL ÁNGULO

El cálculo de la referencia angular se realiza usando las ecuaciones (2.43) y (2.66) descritas en el capítulo dos (secciones 2.2 y 2.3), sin embargo es necesario tener en cuenta con qué signo del ángulo se desea trabajar, pues se puede representar un ángulo de dos maneras:

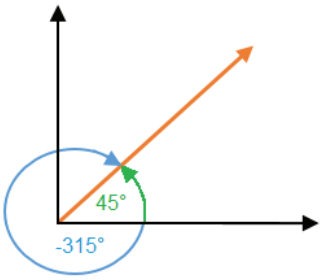


Figura 3.9. Diferentes representaciones de un ángulo

Dependiendo de cuál ángulo se considere el error angular tiene signo positivo o negativo, haciendo que el movimiento sea en sentido antihorario u horario respectivamente. Por este motivo es necesario determinar cuál de los dos giros provee un menor recorrido angular; y de esto se encarga esta subfunción, evaluar los dos casos y retornar el ángulo que otorgue el menor recorrido.

3.1.7 SUBFUNCIÓN EVASIÓN

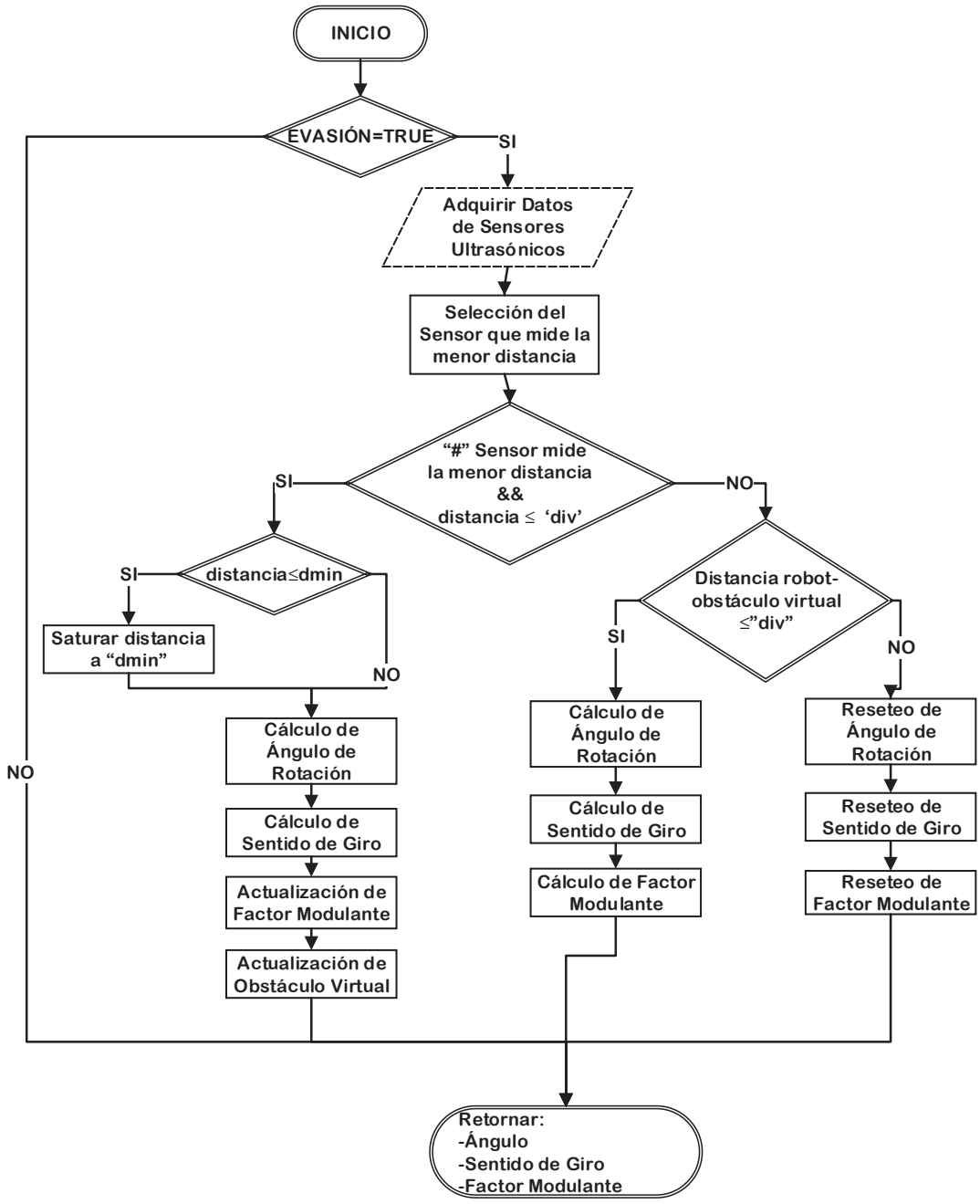


Figura 3.10. Diagrama de flujo de la función evasión

El objetivo de esta subfunción es calcular el módulo del ángulo de rotación, el sentido de giro del robot y el factor modulante, para realizar la rotación y frenado de los componentes de la trayectoria.

Existe un problema físico en la detección del obstáculo y se debe a los puntos ciegos ubicados entre los sensores ultrasónicos. Para solucionar esto se plantean tres posibles casos que corresponden a los ramales del diagrama de flujo (Figura 3.10) en los que se toman diferentes acciones:

3.1.7.1 Caso 1: Los sensores detectan al obstáculo y este se encuentra dentro de la distancia de evasión.

En este caso si la distancia medida es menor que la distancia mínima, se limita el valor medido al de la distancia mínima, se calcula el ángulo, sentido de giro y factor de frenado o factor modulante y se almacena el obstáculo detectado, al que se denomina obstáculo virtual.

3.1.7.2 Caso 2: Los sensores no detectan al obstáculo y la distancia del robot al obstáculo virtual es menor que la distancia de evasión.

En este caso los sensores no han detectado al obstáculo; los valores son calculados de la misma manera que en el primer caso con la diferencia que la distancia usada en las ecuaciones es la distancia calculada (del robot al obstáculo virtual) y no la medida.

3.1.7.3 Caso 3: Los sensores no detectan al obstáculo y la distancia del robot al obstáculo virtual es mayor que la distancia de evasión.

En este caso el robot ya ha evadido al obstáculo por lo que se retorna los valores de evasión nulos, los cuales son factor modulante o de frenado igual a uno y ángulo de rotación igual a cero.

3.2 QTCREATOR PARA VISUALIZACIÓN GRÁFICA

La interfaz está constituida por algunas GUI (interfaces gráficas de usuario), cada una con su archivo source (en donde se realiza la programación de todos los objetos de la GUI) y su respectivo archivo header (en donde se inicializan variables globales y declaran funciones) como se observa en la Tabla 3.1.

Tabla 3.1. Lista de archivos generados para la interfaz del proyecto y sus funciones

GUI	Función	Sources	Headers
mainwindow.ui	Ventana principal de la interfaz (Figura 3.11)	mainwindow.cpp	mainwindow.h
acerca.ui	Ventana que despliega información de los autores del proyecto	acerca.cpp	acerca.h
warning.ui	Ventana que aparece si no se cumple una condición al activar la opción de perturbación	warning.cpp	warning.h
seguimiento.ui	Ventana en donde se observa el seguimiento de la trayectoria, velocidades de referencia y errores	seguimiento.cpp	seguimiento.h
		qcustomplot.cpp	qcustomplot.h
info.ui	Ventana que presenta la información numérica durante el movimiento del robot	info.cpp	info.h

La pantalla principal de la interfaz gráfica posee el aspecto de la Figura 3.11 y presenta directamente todas las opciones de configuración para el funcionamiento del robot. Se encuentran precargados los valores por default correspondientes a las constantes de calibración y parámetros básicos, que pueden ser variados de acuerdo al gusto del usuario. Su funcionamiento es descrito por el diagrama de flujo de la Figura 3.12.

Figura 3.11. Pantalla principal de la interfaz gráfica

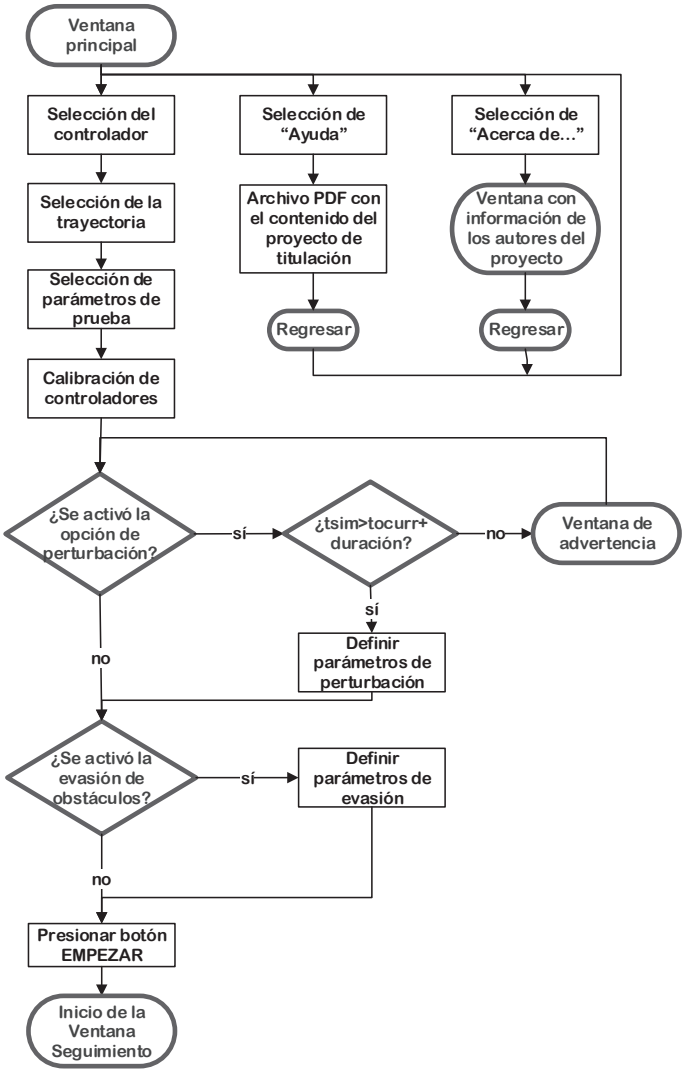


Figura 3.12. Diagrama de flujo de la interfaz gráfica

La segunda parte importante de la interfaz gráfica es la ventana de seguimiento (Figura 3.13). Al iniciar esta pantalla ya no se permite volver a la pantalla principal pues se ha desvinculado al programa del robot (ya no se pueden variar parámetros), y se actúa en un estado pasivo en el que solo recibe información. Su funcionamiento se describe en el diagrama de flujo de la Figura 3.14.

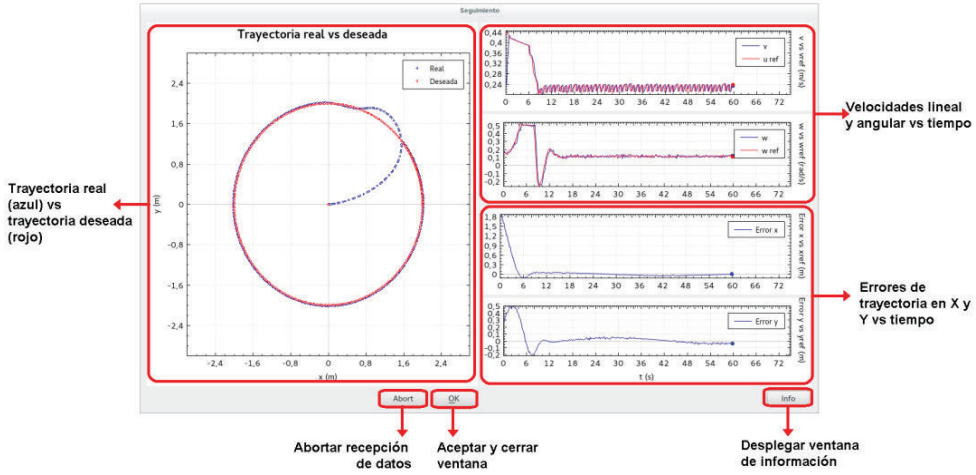


Figura 3.13. Pantalla de visualización del seguimiento de trayectoria

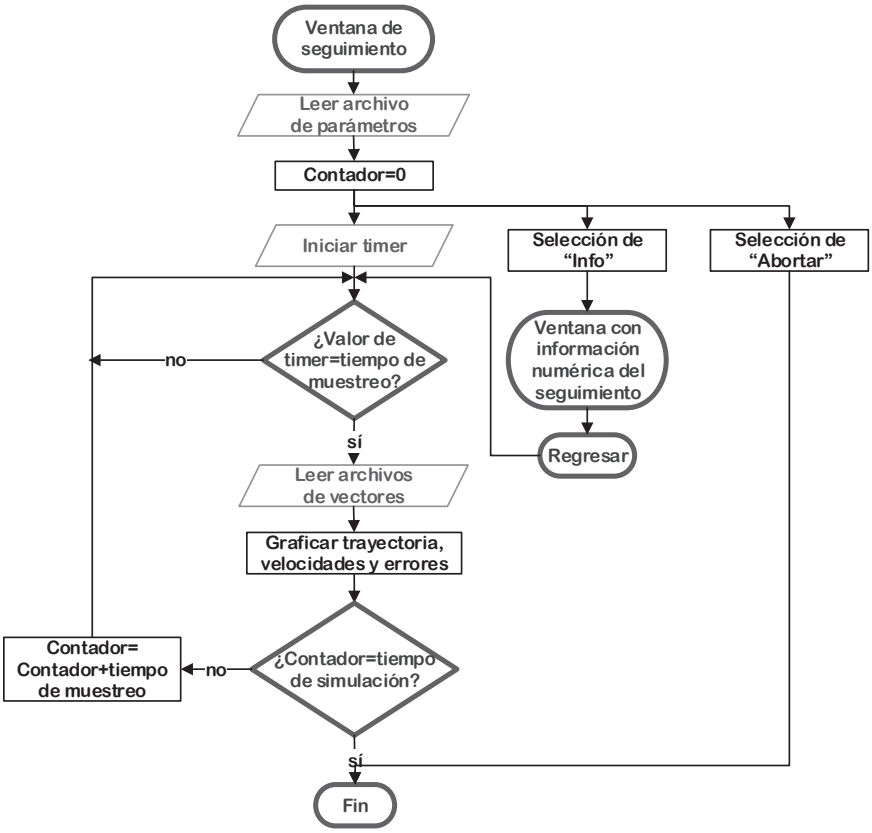


Figura 3.14. Diagrama de flujo de la ventana de seguimiento

CAPÍTULO 4

PRUEBAS Y RESULTADOS

Los cuatro controladores desarrollados en el capítulo dos son implementados en el robot Pioneer 3DX y para evaluar su desempeño se realizan los siguientes experimentos:

- Seguimiento de una trayectoria circular con velocidad lineal de referencia de 0.1, 0.25 y 0.5 (m/s).
- Seguimiento de una trayectoria cuadrada con velocidad lineal de referencia de 0.1 y 0.25 (m/s).
- Evasión de obstáculos cuando el robot se desplaza en línea recta a una velocidad lineal de referencia de 0.1, 0.25 y 0.5 (m/s).
- Introducción de una perturbación cuando el robot se desplaza en una trayectoria circular a una velocidad lineal de referencia de 0.25 (m/s).

De cada experimento se presentan los gráficos de la parte inicial de la respuesta, en la que se observa como el robot alcanza la referencia, debido a que es la parte de interés que evidencia el comportamiento de los controladores.

Para comparar el desempeño los controladores en el seguimiento de trayectorias se ha utilizado el valor promedio del índice de desempeño IAE (Integral del Error Absoluto) [39].

$$\overline{IAE} = \frac{\int_0^T |e(t)| dt}{T} \tag{4.1}$$

Donde $e(t)$ es error en la trayectoria.

De la plataforma robótica Pioneer 3DX se consideran las siguientes características para la implementación de los controladores:

- El punto de interés se ubica a una distancia de 20 (cm) delante del centro del eje de las ruedas del robot. Esta distancia se denomina a (Figura 1.3).
- La distancia entre las ruedas del robot es de 40 (cm), [30].

- Se tiene un arreglo de sensores ultrasónicos propios de la plataforma (Figura 4.1) que fue utilizado para realizar la evasión de obstáculos

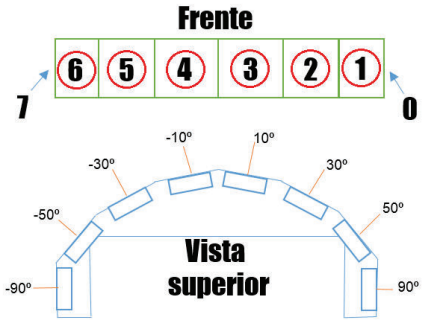


Figura 4.1. Arreglo de sensores ultrasónicos del Pioneer 3DX, [31]

- La computadora a bordo (la laptop externa usada) se comunica con el microcontrolador H8S a través del puerto serial /dev/ttyS0 en Linux.
- La comunicación entre la PC maestro y esclavo se realiza a través de una red wireless.

Además la herramienta de software MobileSim, provisto por la empresa Mobile Robots, permite simular todas las pruebas antes de ser implementadas en el propio robot, lo cual resulta muy útil para realizar calibración de parámetros y correcciones de errores antes de realizar los experimentos prácticos.

El procedimiento de ejecución de las pruebas se explica con mayor detalle en el Anexo A – Manual de usuario.

4.1.1 TRAYECTORIA CIRCULAR

La trayectoria circular está definida por $\vec{x} = r\ cos(\omega\vec{t})$ y $\vec{y} = r\ sin(\omega\vec{t})$ donde $r = 2\ (m)$ es el radio del círculo y $\omega = \frac{v}{r}$ es la velocidad angular de la trayectoria de referencia.

El período de muestreo es de $T_0 = 0.1\ (s)$ con las siguientes condiciones iniciales $[x_0, y_0, \varphi_0]^T = [0,0,0]^T$ y $[v_0, \omega_0]^T = [0,0]^T$.

Se efectuaron tres experimentos a velocidades lineales diferentes de 0.1, 0.25 y 0.5 (m/s). Para cada uno se presentan cinco gráficos: trayectoria, las componentes de la trayectoria x y y, las velocidades lineal y angular, el ángulo del robot y los errores en la trayectoria.

4.1.1.1 Experimento con trayectoria circular y velocidad lineal de 0.1 (m/s)

A 0.1 (m/s) se observa que los controladores de Euler y Trapezoidal tienen un mejor desempeño que el controlador PI y el controlador por modo deslizante. En la Figura 4.2 y Figura 4.3 se aprecia como los controladores basados en métodos numéricos se estabilizan inmediatamente al alcanzar la trayectoria, y aunque los controladores PI y modo deslizante llegan más rápido a la referencia, tardan más tiempo en estabilizarse.

Además a pesar del sobreimpulso que presenta el controlador por modo deslizante, se evidencia que el tiempo de estabilización es mucho menor que el PI.

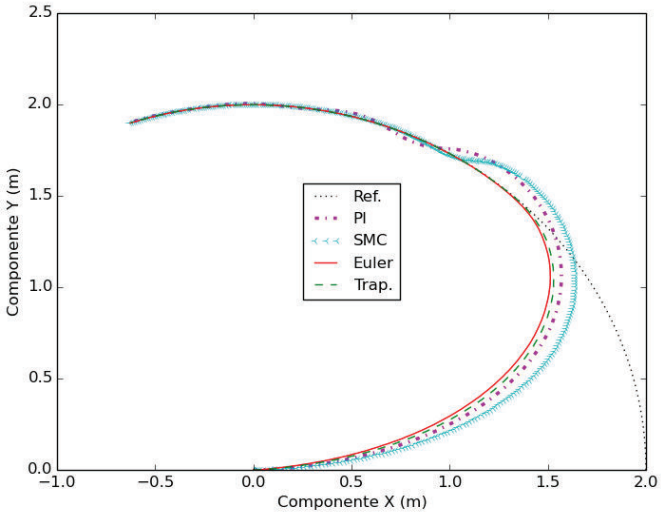


Figura 4.2. Trayectoria deseada y trayectorias del robot con los cuatro controladores (Trayectoria circular)

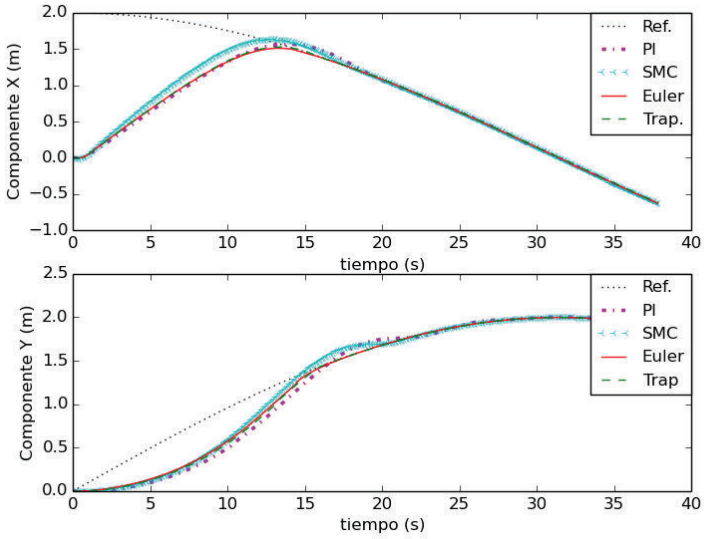


Figura 4.3. Trayectoria en X y Y del robot con los cuatro controladores (Trayectoria circular)

En la Figura 4.4 se observa que los errores de trayectoria en X son similares para todos los controladores, iniciando con un error de aproximadamente 2 (m) que se reduce mientras el robot se moviliza hasta alcanzar la trayectoria. El controlador PI presenta los mayores errores de trayectoria en Y, alcanzando un valor máximo de 0.5 (m); y es el último en estabilizarse (en alrededor de t = 30 (s)).

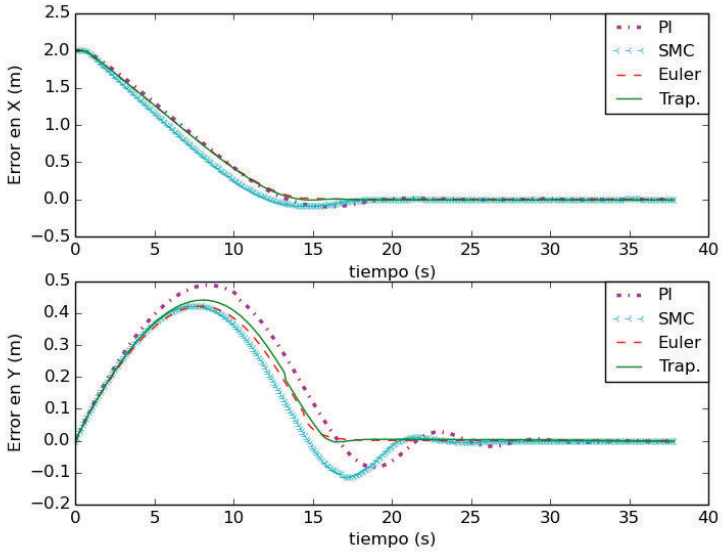


Figura 4.4. Errores de trayectoria en X y Y del robot con los cuatro controladores (Trayectoria circular)

La Figura 4.5 refleja la orientación del robot con cada controlador, pues se ve como el ángulo obtenido con los controladores Euler y Trapezoide evoluciona formando una curva suave, mientras que con los controladores PI y modo deslizante se presenta un ángulo que cambia bruscamente hasta que el robot logra estabilizarse.

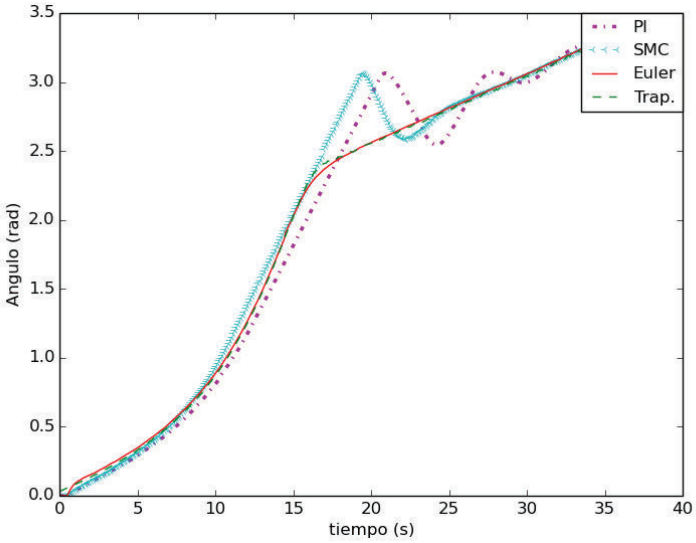


Figura 4.5. Ángulo φ del robot para los cuatro controladores (Trayectoria circular)

Como la plataforma robótica inicia desde el punto (0,0) (m) durante los primeros segundos del experimento los controladores envían señales de velocidad mayores a las velocidades de referencia con las que se creó la trayectoria. Esto ocurre hasta que el robot logra alcanzar la trayectoria, como se ve en la Figura 4.6.

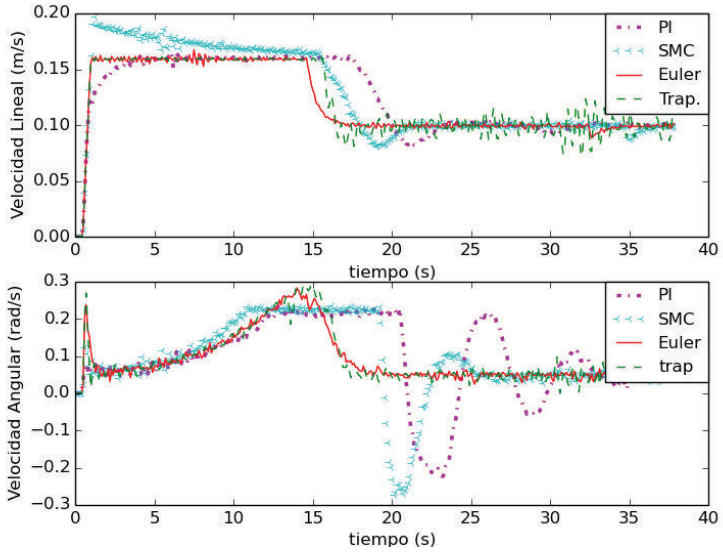


Figura 4.6. Velocidad lineal y angular del robot para los cuatro controladores (Trayectoria circular)

Como los controladores basados en métodos numéricos (Euler y Trapezoide) alcanzan más rápido a la trayectoria sus velocidades lineal y angular, alcanzan más rápido el valor de referencia de la trayectoria.

Los resultados del índice de error absoluto (IAE) que se muestran en la Tabla 4.1 indican que el controlador por modo deslizante presenta el menor error en X y Euler el menor error en Y.

Esto se explica conociendo que el controlador por modo deslizante gracias a su parte switching es más rápido que los demás controladores, lo que provoca que el error en X disminuya rápidamente haciendo que sea el primero en llegar a la referencia (como se ve en la Figura 4.2, Figura 4.3 y Figura 4.4) aunque presenta un sobreimpulso que le dificulta estabilizarse de inmediato debido a que usa un modelo aproximado de la plataforma.

Por otro lado el controlador de Euler presenta un mejor índice de desempeño en Y puesto que no tiene sobretiro, como se ve en la Figura 4.3, esto se debe al reemplazo que se hace en la sección 2.2.2 para que el error tienda a cero de manera estable.

Tabla 4.1. Comparación de IAE de los cuatro controladores (Trayectoria circular)

O	Euler	Trapezoide	PI	Modos
0.1 (m/s)	**IAE*(10^{-3})**	**IAE*(10^{-3})**	**IAE*(10^{-3})**	**IAE*(10^{-3})**
x	13.763	13.938	14.130	**12.487**
y	**4.385**	4.841	5.532	4.489

4.1.1.2 Experimento con trayectoria circular y velocidad lineal de 0.25 (m/s)

A 0.25 (m/s) se observa que los controladores presentan un comportamiento similar al observado con velocidad lineal de 0.1 (m/s). Sin embargo se puede notar en la Figura 4.7 y la Figura 4.8 que a mayor velocidad lineal aparece un error de trayectoria de 4 (cm) para los controladores PI y modo deslizante, despreciable respecto a las dimensiones de la plataforma [30].

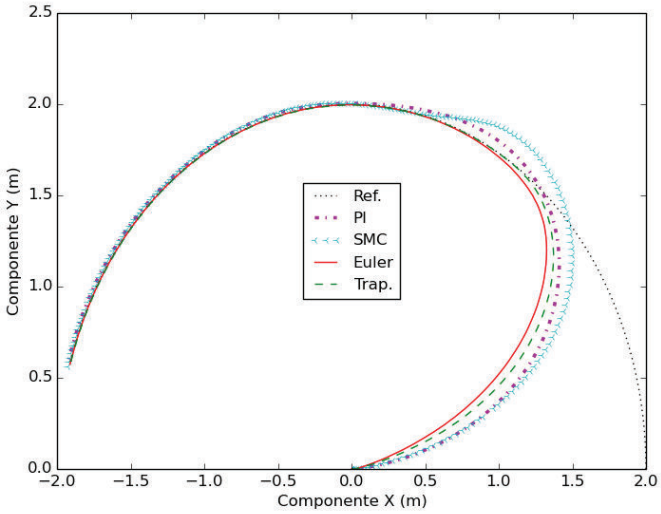


Figura 4.7. Trayectoria deseada y trayectorias del robot con los cuatro controladores (Trayectoria circular)

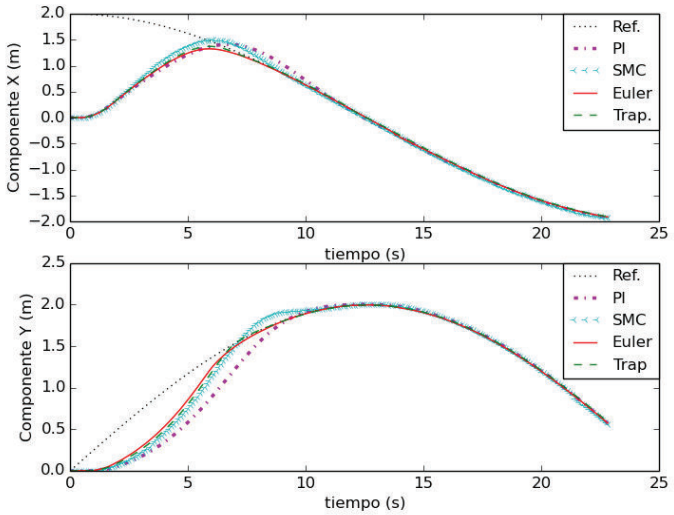


Figura 4.8. Trayectoria en X y Y del robot con los cuatro controladores (Trayectoria circular)

En la Figura 4.9 se observa que el comportamiento de los errores de trayectoria en X son parecidos a los observados a 0.1 (m/s), no obstante en errores de trayectoria en Y el controlador PI destaca con su elevado error máximo (63 (cm)), que provoca que llegue a cero alrededor de cinco segundos después de los demás controladores. También se observa que el controlador por modo deslizante es ahora el único que presenta error de trayectoria negativo considerable, mismo que provoca el sobretiro mostrado en la Figura 4.7.

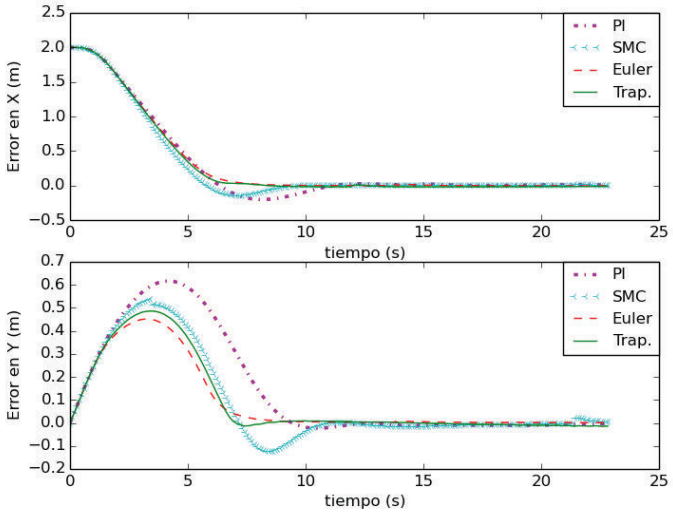


Figura 4.9. Errores de trayectoria en X y Y del robot con los cuatro controladores (Trayectoria circular)

En la Figura 4.10 destaca el ángulo de orientación obtenido con el controlador PI, que ahora forma una curva suave a diferencia del experimento a 0.1 (m/s) aunque demora en estabilizarse, a diferencia del controlador por modo deslizante, que es más brusco pero se estabiliza más rápido que el controlador PI.

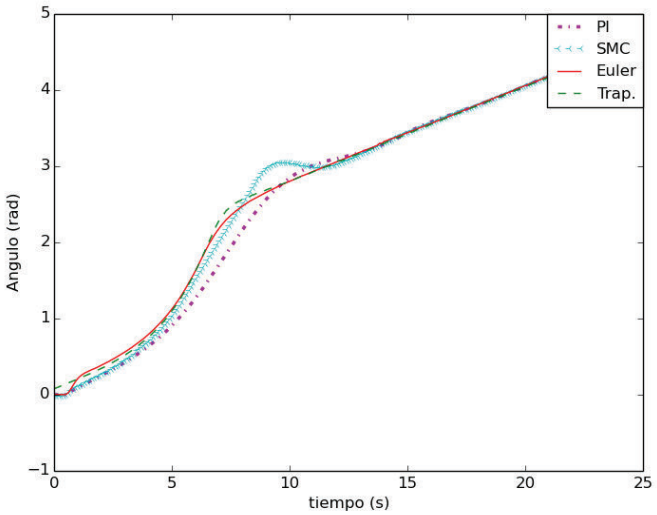


Figura 4.10. Ángulo φ del robot para los cuatro controladores (Trayectoria circular)

Respecto a las velocidades, nuevamente durante los primeros segundos del experimento los controladores envían señales de velocidad mayores a las velocidades con las que fue creada la trayectoria hasta que la plataforma logre alcanzarla, como se indica en la Figura 4.11.

En este caso los controladores Euler y Trapezoide son los primeros en alcanzar la referencia de velocidad lineal de la trayectoria a los 7 segundos, modo deslizante lo hace alrededor de 3 segundos después y PI es el último controlador en llegar alrededor de los 13 segundos. Este orden se repite también en el caso de la velocidad angular.

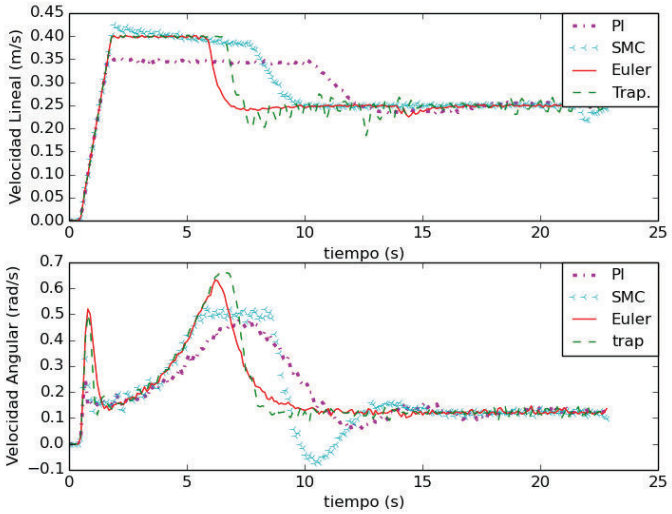


Figura 4.11. Velocidad lineal y angular del robot para los cuatro controladores (Trayectoria circular)

Los resultados del índice de error absoluto (IAE) que se muestran en la Tabla 4.2 presentan el menor error en X y Y para el control Trapezoidal con la nueva velocidad de 0.25 (m/s), esto debido a que la aproximación usada para estimar el siguiente estado es más cercana al valor real que la aproximación de Euler, dando mejores resultados cuando aumenta la velocidad.

Como se ve en la Figura 4.7 y la Figura 4.8, el controlador por modo deslizante sigue siendo el controlador más rápido, no obstante, al incrementarse la velocidad también aumenta el sobretiro que presenta.

También se nota que Trapezoide se comporta de manera similar al experimento anterior, pero esta ocasión logra la estabilización más rápidamente que los demás controladores, lo que le proporciona la ventaja al comparar los resultados de IAE.

Tabla 4.2. Comparación de IAE de los cuatro controladores (Trayectoria circular)

O	Euler	Trapezoide	PI	Modos
0.25 (m/s)	**IAE*(10^{-3})**	**IAE*(10^{-3})**	**IAE*(10^{-3})**	**IAE*(10^{-3})**
x	8.027	**7.438**	8.566	7.792
y	3.163	**2.284**	4.246	3.433

4.1.1.3 Experimento con trayectoria circular y velocidad lineal de 0.5 (m/s)

Al incrementar la velocidad a 0.5 (m/s) se observa en primer lugar que el robot alcanza a la trayectoria en un punto más lejano que en los experimentos anteriores, pero es comprensible pues se ha doblado la velocidad lineal.

En este caso y según la Figura 4.12 y la Figura 4.13, se obtiene un error de trayectoria considerable para el PI de alrededor de 12 (cm). Los demás controladores ahora alcanzan a la trayectoria e inmediatamente se estabilizan.

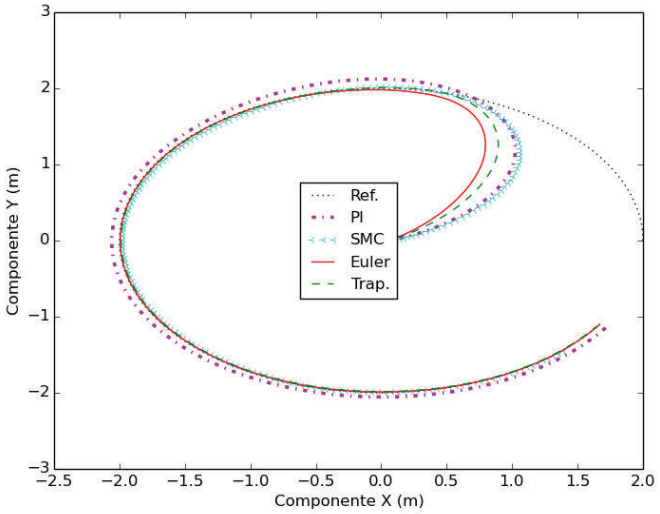


Figura 4.12. Trayectoria deseada y trayectorias del robot con los cuatro controladores (Trayectoria circular)

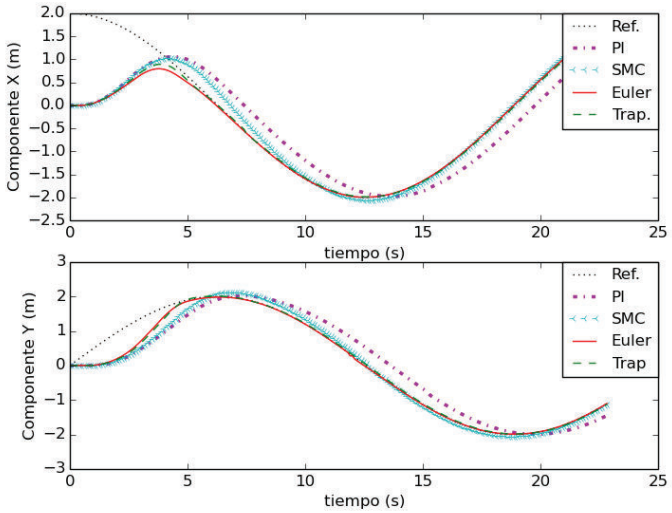


Figura 4.13. Trayectoria en X y Y del robot con los cuatro controladores (Trayectoria circular)

Dentro de los errores de trayectoria presentes en la Figura 4.14, se refuerza lo indicado a inicio de la descripción de este experimento, pues el error del PI es muy grande cuando se lo compara con los de los demás controladores, esto debido a que no es capaz de alcanzar a la trayectoria en ningún momento. El error de modo deslizante también es considerable, pero como se aprecia en la Figura 4.12 y la Figura 4.13 la acción de este controlador mantiene al robot alrededor de la trayectoria. Esto sucede debido a que, a diferencia del control PI, este controlador posee la parte switching que hace que el robot siga la referencia a una mayor velocidad.

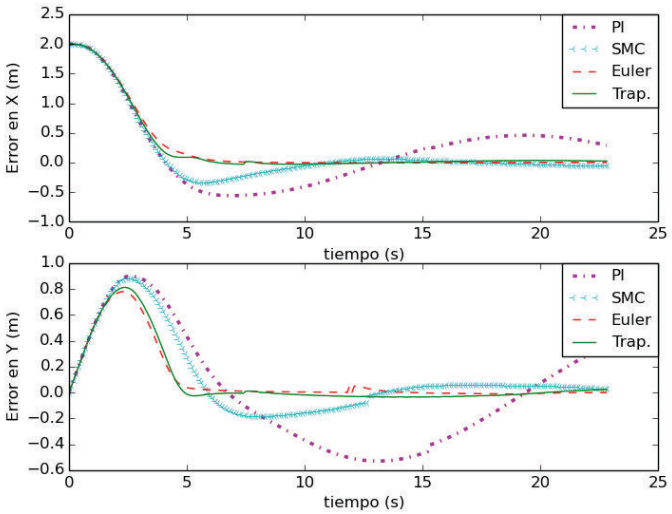


Figura 4.14. Errores de trayectoria en X y Y del robot con los cuatro controladores (Trayectoria circular)

En la Figura 4.15 se observa como ahora con todos los controladores se obtiene una curva suave del ángulo de orientación, con la particularidad de que con el controlador PI está desfasado angularmente de los resultados obtenidos con los demás controladores en todo momento.

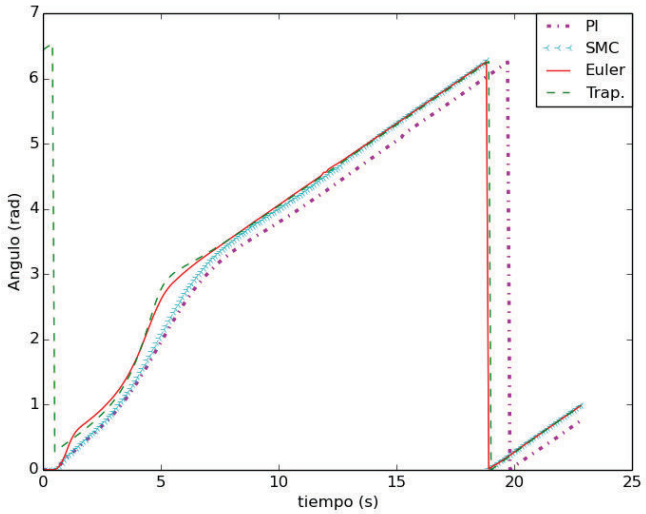


Figura 4.15. Ángulo φ del robot para los cuatro controladores (Trayectoria circular)

Al observar las gráficas de velocidad lineal y angular de la Figura 4.16 es evidente el esfuerzo que realizan los controladores para alcanzar la trayectoria, para la velocidad lineal el controlador PI no alcanza la referencia hasta t = 20 (s) cuando los demás controladores ya han alcanzado la trayectoria alrededor de t = 7 (s).

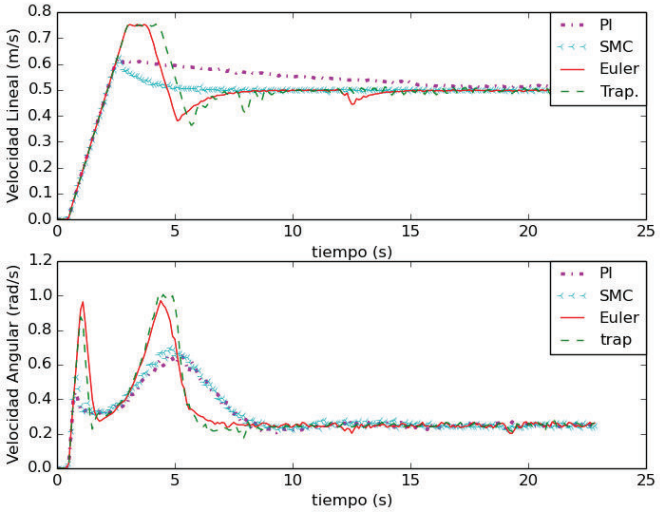


Figura 4.16. Velocidad lineal y angular del robot para los cuatro controladores (Trayectoria circular)

Al calcular el índice de error absoluto (IAE) para una velocidad lineal de referencia de 0.5 (m/s) se nota el mejor desempeño de los controladores basados en métodos numéricos, siendo Euler el controlador que presenta mejores valores de IAE, como se observa en la Tabla 4.3. Además como se observó en la Figura 4.16 y la Figura 4.15 el controlador PI no es lo suficientemente rápido para lograr alcanzar la referencia de velocidad angular, mismo motivo por el que mantiene un error de trayectoria constante y sus valores de IAE son mucho mayores que los obtenidos para los demás controles.

Esto sucede por dos razones: en primer lugar el uso del modelo exacto de la plataforma otorga una gran ventaja a los controladores basados en métodos numéricos y en segundo lugar, los controladores PI y por modo deslizante necesitan mayores acciones de control que los anteriores para lograr alcanzar las referencias.

Tabla 4.3. Comparación de IAE de los cuatro controladores (Trayectoria circular)

O	Euler	Trapezoide	PI	Modos
0.5 (m/s)	**IAE*(10^{-3})**	**IAE*(10^{-3})**	**IAE*(10^{-3})**	**IAE*(10^{-3})**
x	**6.208**	7.424	32.534	18.515
y	**2.789**	4.354	29.267	15.421

4.1.2 TRAYECTORIA CUADRADA

Para esta trayectoria se ha elegido un cuadrado de lado $= 2.5\ (m)$ descrito por las siguientes ecuaciones de movimiento rectilíneo uniforme:

$$\vec{x} = v\vec{t} \quad \text{cuando} \quad \vec{y} = \vec{0}$$
$$\vec{x} = \vec{0} \quad \text{cuando} \quad \vec{y} = v\vec{t}$$

El período de muestreo es de $T_0 = 0.1\ (s)$ con las siguientes condiciones iniciales $[x_0, y_0, \varphi_0]^T = [0,0,0]^T$ y $[v_0, \omega_0]^T = [0,0]^T$.

Se efectuaron dos experimentos a velocidades lineales diferentes de 0.1 y 0.25 (m/s). Para cada uno se presentan nuevamente cinco gráficos: trayectoria, las componentes de la trayectoria x y y, las velocidades lineal y angular, el ángulo del robot y los errores en la trayectoria.

4.1.2.1 Experimento con trayectoria cuadrada y velocidad lineal de 0.1 (m/s)

A una velocidad lineal de 0.1 (m/s) todos los controladores siguen la trayectoria establecida presentando pequeños sobrepicos en los giros de 90º. Como se observa en la Figura 4.17 los controladores basados en métodos numéricos presentan un desempeño similar entre sí, con un sobretiro de alrededor del 16% (40 (cm)) y los resultados obtenidos con los controladores PI y por modo deslizante también son parecidos entre sí, con un sobreimpulso de 2.8% (7 (cm)). La única diferencia entre estos últimos, es que el PI presenta oscilaciones antes de estabilizarse, mientras que el controlador por modo deslizante se estabiliza de inmediato al llegar a la referencia. Este comportamiento es propio del control modo deslizante, cuya parte discontinua le provee alta velocidad para alcanzar la trayectoria, y una vez alcanzada la referencia la parte continua le brinda estabilidad para quedarse sobre ella.

En la Figura 4.18 de hecho resulta difícil diferenciar los resultados del controlador PI y modo deslizante, pues están prácticamente superpuestos.

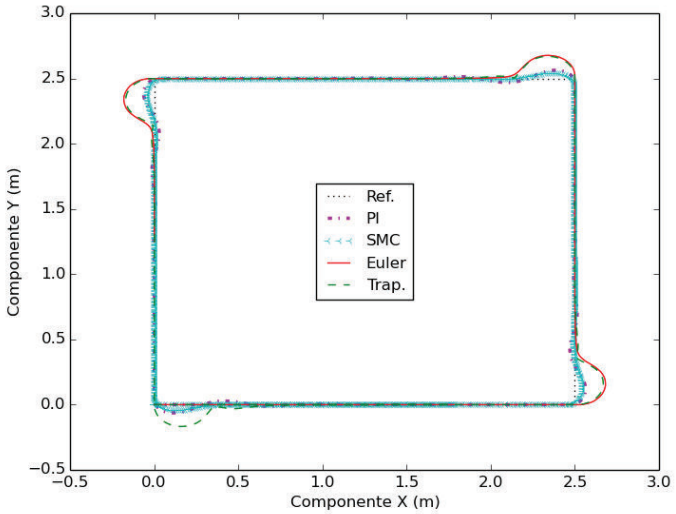


Figura 4.17. Trayectoria deseada y trayectorias del robot con los cuatro controladores (Trayectoria cuadrada)

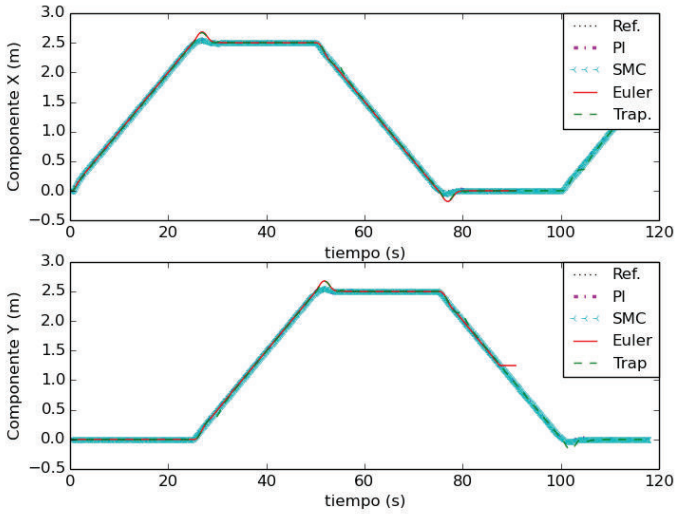


Figura 4.18. Trayectoria en X y Y del robot con los cuatro controladores (Trayectoria cuadrada)

En las gráficas de errores de trayectoria presentadas en la Figura 4.19 se observa que la mayoría del tiempo se tiene un error de cero en X y Y para los controladores PI y por modo deslizante. Similarmente, los controladores de Euler y Trapezoide apenas presentan errores en ciertos instantes de tiempo, que no afectan de mayor forma el seguimiento total de la trayectoria, como se observa en la Figura 4.17.

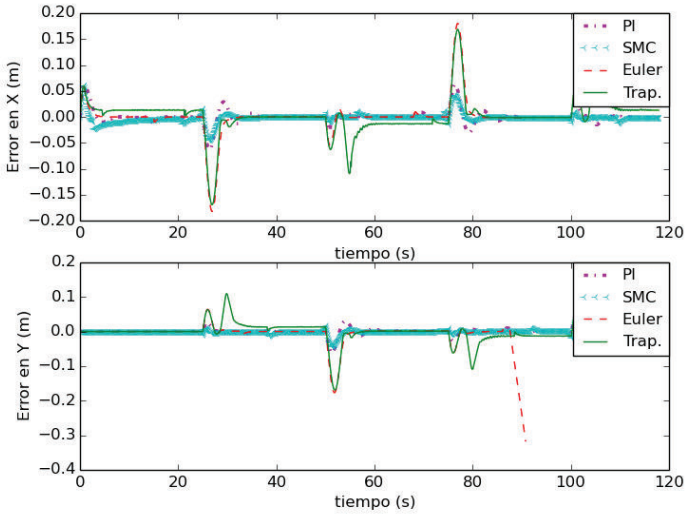


Figura 4.19. Errores de trayectoria en X y Y del robot con los cuatro controladores (Trayectoria cuadrada)

Continuando con la tendencia de las gráficas anteriores, al analizar el ángulo de orientación presentado en la Figura 4.20 se observa que en los giros de 90º los controladores basados en métodos numéricos presentan un mayor sobreimpulso

que el PI y el controlador por modo deslizante. En este resultado se aprecia también como oscila el PI hasta estabilizarse, a diferencia del controlador por modo deslizante que lo hace de inmediato.

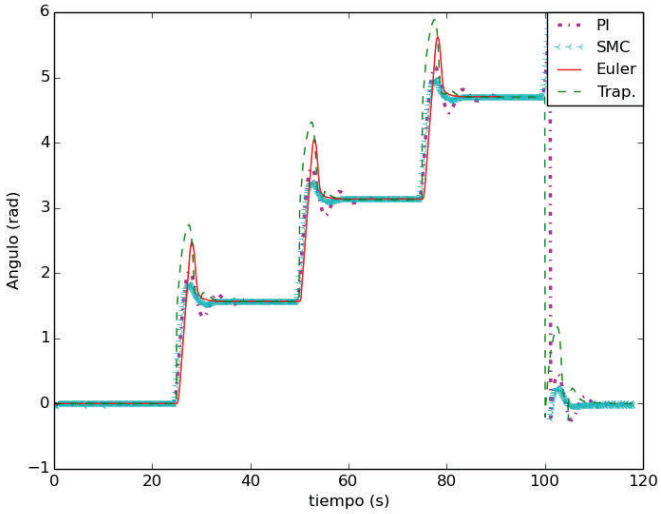


Figura 4.20. Ángulo φ del robot para los cuatro controladores (Trayectoria cuadrada)

En la Figura 4.21 se observa que frente a cambios bruscos en la trayectoria se requieren cambios bruscos de velocidad, como los presentados por el control de Euler y Trapezoide. Sin embargo, se ve que las acciones de control generadas por el controlador PI y por modo deslizante evitan que los valores de velocidad lineal sean altos, mientras que para la velocidad angular presentan valores similares a los controladores que usan métodos numéricos.

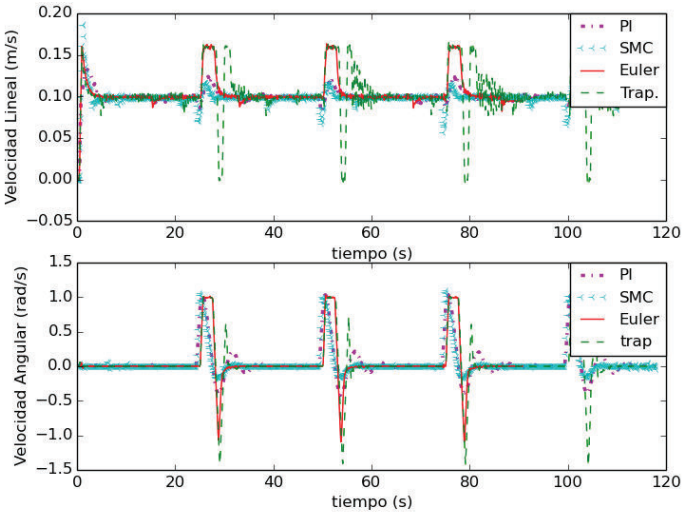


Figura 4.21. Velocidad lineal y angular del robot para los cuatro controladores (Trayectoria cuadrada)

Los IAE presentados en la Tabla 4.4 reflejan la manera en que las características del controlador por modo deslizante en el desarrollo de su controlador le ha beneficiado mucho al momento de tomar cambios de dirección bruscos, como los ángulos de 90º del cuadrado, al contrario de lo que pasa con los controladores basados en métodos numéricos, que no evitan tener un pronunciado sobreimpulso. Sin embargo, los controladores de Euler y Trapezoide – si bien tardan más en estabilizarse - no presentan oscilaciones, como sucede con el PI.

Tabla 4.4. Comparación de IAE de los cuatro controladores (Trayectoria cuadrada)

□	Euler	Trapezoide	PI	Modos
0.1 (m/s)	**IAE*(10^{-3})**	**IAE*(10^{-3})**	**IAE*(10^{-3})**	**IAE*(10^{-3})**
x	1.173	1.783	0.598	**0.427**
y	1.303	1.534	0.447	**0.244**

4.1.2.2 Experimento con trayectoria cuadrada y velocidad lineal de 0.25 (m/s)

Al incrementar la velocidad a 0.25 (m/s) se observa en la Figura 4.22 que todos los controladores presentan mayor dificultad que en el caso anterior para realizar el seguimiento, pues al llegar al giro de 90º todos presentan un sobreimpulso y logran estabilizarse después de alrededor de 5 segundos como se ve en la Figura 4.23.

Sin embargo, todos los controladores presentan una tendencia similar al caso anterior, donde Euler y Trapezoide presentan un gran sobretiro del 20% (50 (cm)) y el PI y el controlador por modo deslizante tienen un sobrepico del 8% (20 (cm)).

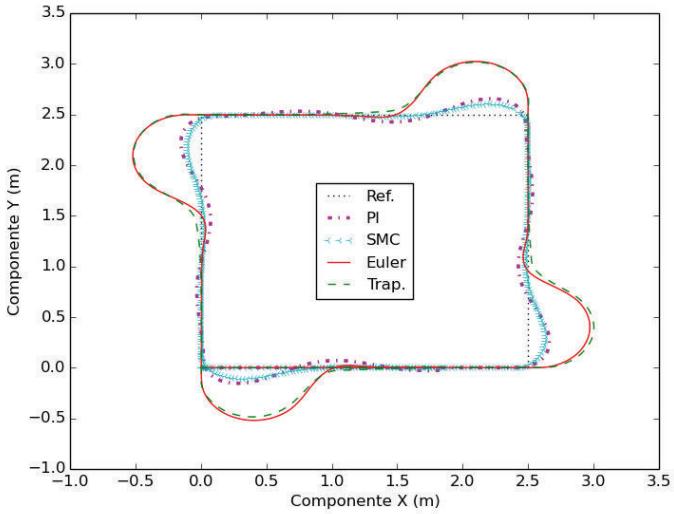


Figura 4.22. Trayectoria deseada y trayectorias del robot con los cuatro controladores (Trayectoria cuadrada)

A diferencia del caso anterior, el controlador PI ahora se demora más en estabilizarse, pues a pesar de que llega a la trayectoria junto a los resultados obtenidos con los demás controladores, se queda oscilando alrededor de la referencia.

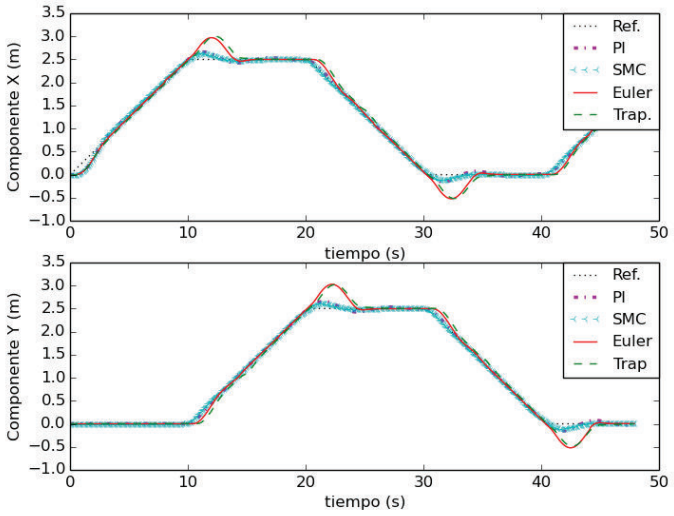


Figura 4.23. Trayectoria en X y Y del robot con los cuatro controladores (Trayectoria cuadrada)

Los errores de trayectoria que se presentan en este caso reflejan el mejor desempeño de los controladores PI y modo deslizante, pues su error en X y Y es siempre cercano a cero, mientras que para los controladores basados en métodos

numéricos el error tiene un mayor rango de variación, como se observa en la Figura 4.24.

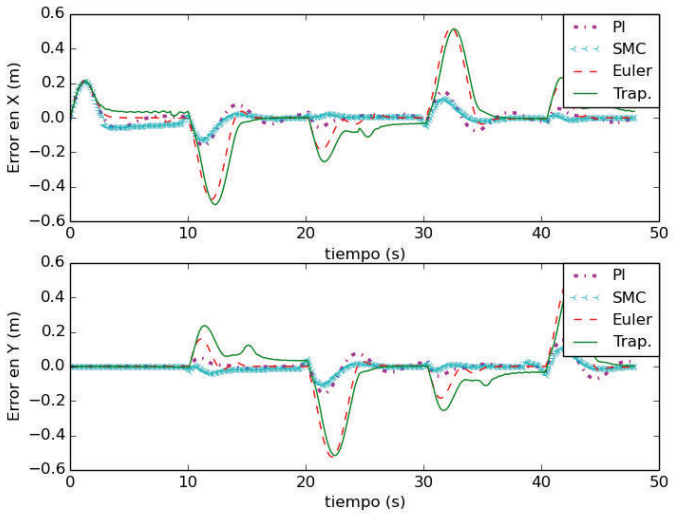


Figura 4.24. Errores de trayectoria en X y Y del robot con los cuatro controladores (Trayectoria cuadrada)

Análogamente al caso anterior, al analizar los resultados de la evolución del ángulo (Figura 4.25) se nota que el mejor comportamiento lo presenta el controlador por modo deslizante, al poseer el menor sobretiro y ser el más rápido en estabilizarse, mientras el controlador PI aún oscila y los controladores Euler y Trapezoide presentan un gran sobreimpulso.

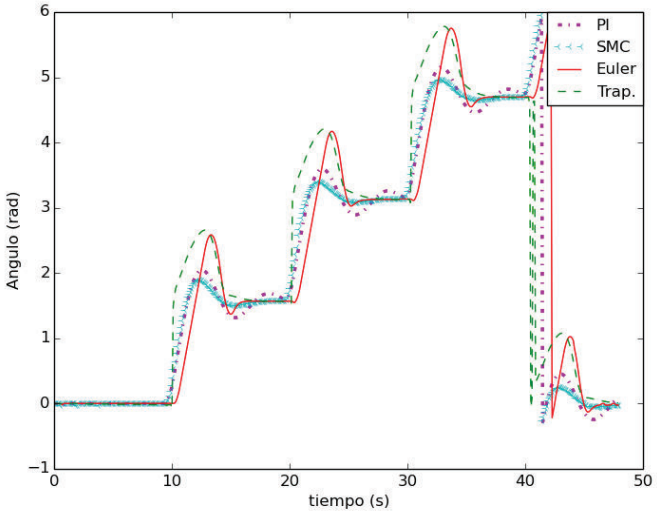


Figura 4.25. Ángulo φ del robot para los cuatro controladores (Trayectoria cuadrada)

En la Figura 4.26 al igual que en el caso anterior las acciones de control del PI y del controlador por modo deslizante evitan que los valores de velocidad lineal se

disparen al presentarse un cambio brusco de trayectoria, al contrario de Euler y Trapezoide que presentan violentas acciones de control para seguir la referencia al llegar al giro de 90º en la trayectoria.

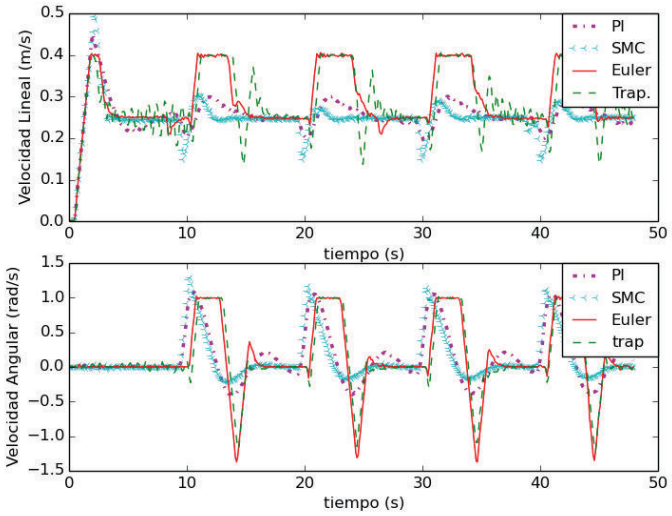


Figura 4.26. Velocidad lineal y angular del robot para los cuatro controladores (Trayectoria cuadrada)

Nuevamente según los resultados gráficos y los IAE obtenidos (Tabla 4.5) se puede apreciar que el controlador por modo deslizante presenta una mejor reacción ante los cambios bruscos debido al comportamiento de su parte continua y discontinua. A una mayor velocidad los controladores basados en álgebra lineal tardan más en regresar a la trayectoria referencia, por lo que se observan mayores sobrepicos y mayor tiempo de establecimiento. Esto pasa al igual que en el caso anterior debido a la propia constitución de los controladores.

Tabla 4.5. Comparación de IAE de los cuatro controladores (Trayectoria cuadrada)

□	Euler	Trapezoide	PI	Modos
0.25 (m/s)	**IAE*(10^{-3})**	**IAE*(10^{-3})**	**IAE*(10^{-3})**	**IAE*(10^{-3})**
x	7.127	9.879	3.042	**2.582**
y	6.509	8.702	2.128	**1.460**

4.1.3 EVASIÓN DE OBSTÁCULOS

Para realizar la evasión de obstáculos se ha asignado al robot una trayectoria lineal descrita por la ecuación de movimiento rectilíneo uniforme $\vec{x} = v\vec{t}$ con $\vec{y} = \vec{0}$. El obstáculo usado para las pruebas experimentales fue una

columna que se localiza entre los puntos (3.55, 0) y (4.23, 0) de la trayectoria del robot.

El período de muestreo es de $T_0 = 0.1\ (s)$ con las siguientes condiciones iniciales $[x_0, y_0, \varphi_0]^T = [0,0,0]^T$ y $[v_0, \omega_0]^T = [0,0]^T$.

Se efectuaron tres experimentos a velocidades lineales diferentes de 0.1, 0.25 y 0.5 (m/s). Se realizaron las pruebas a las tres velocidades lineales para poder establecer con qué controlador el robot se desvía menos de la trayectoria deseada y se demora menos en regresar a la referencia.

Para cada experimento se presentan cuatro gráficos: trayectoria, las componentes de la trayectoria x y y, las velocidades lineal y angular, y el ángulo del robot.

4.1.3.1 Experimento de evasión de obstáculos y velocidad lineal de 0.1 (m/s)

En la Figura 4.27 se ilustra cómo los cuatro controladores realizan la evasión. A primera vista el controlador de Euler es el que menos se desvía de su trayectoria y que regresa más rápido a la misma.

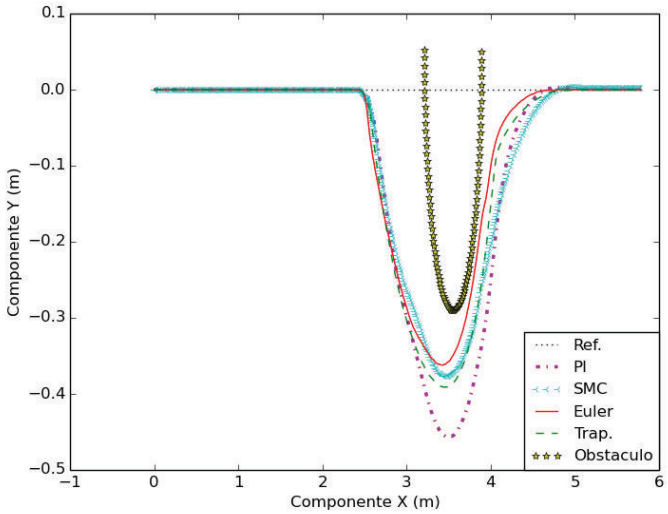


Figura 4.27. Trayectoria deseada y trayectorias del robot con los cuatro controladores durante la evasión de obstáculos

En la Figura 4.28 se corrobora lo observado anteriormente. A partir de t = 25 (s) aproximadamente el robot inicia el proceso de evasión, alterando su trayectoria en x y y. Se aprecia claramente que a esta velocidad los controladores de Euler y

Trapezoide inician el proceso de evasión antes que el PI y el controlador por modo deslizante.

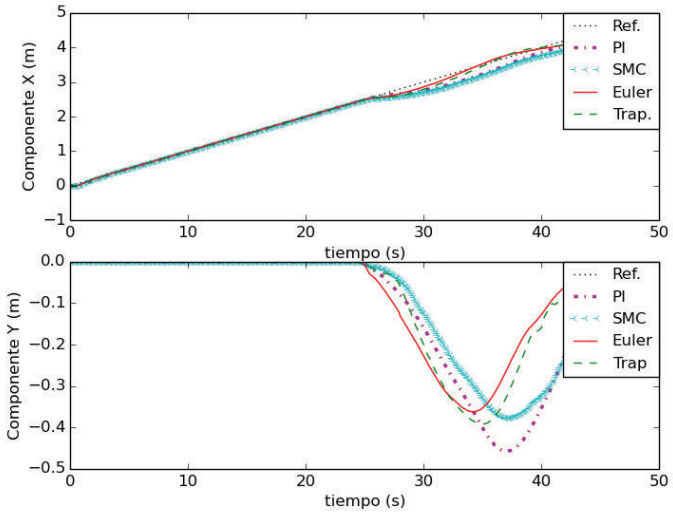


Figura 4.28. Trayectoria en X y Y del robot con los cuatro controladores durante la evasión de obstáculos

En el gráfico de la evolución del ángulo de orientación del robot (Figura 4.29) se observa que cuando el robot inicia la evasión el ángulo cambia inmediatamente para permitirle a la plataforma girar y realizar la evasión. Aquí también se puede notar que los controladores de Euler y Trapezoide son más rápidos, pues apenas termina la evasión el ángulo resultado regresa a ser el original (0º), mientras que para el PI y modo deslizante tardan alrededor de 3 segundos más en regresar al ángulo original, lo cual es comprensible ya que no usan un modelo exacto del robot.

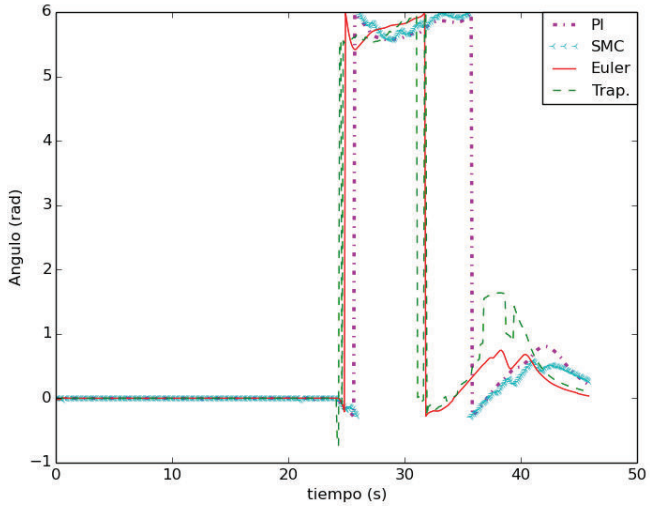


Figura 4.29. Ángulo φ del robot para los cuatro controladores durante la evasión de obstáculos

Al observar la Figura 4.30 se nota claramente el momento en que inicia la evasión (t = 25 (s)), pues de inmediato se disminuye la velocidad lineal ya que el robot debe evadir el obstáculo a menor velocidad que la de navegación. Una vez que el robot ya no detecta el obstáculo, se incrementa la velocidad lineal hasta lograr que la plataforma alcance la referencia de velocidades de la trayectoria.

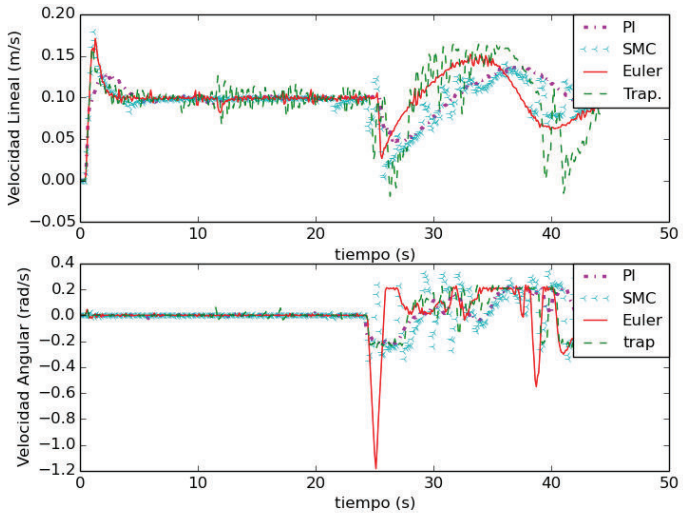


Figura 4.30. Velocidad lineal y angular del robot para los cuatro controladores durante la evasión de obstáculos

4.1.3.2 Experimento de evasión de obstáculos y velocidad lineal de 0.25 (m/s)

Al incrementar la velocidad a 0.25 (m/s) los resultados anteriores cambian, pues ahora el controlador por modo deslizante es el que reacciona mejor según la Figura 4.31. En la Figura 4.32 se ve que el controlador PI es el que inicia primero la evasión, pero regresa a la trayectoria original último, mientras que el controlador Euler y por modo deslizante inician prácticamente en el mismo instante (t = 11 (s)) y aunque Euler regresa más rápido, es el controlador por modo deslizante el que se aleja menos de la trayectoria.

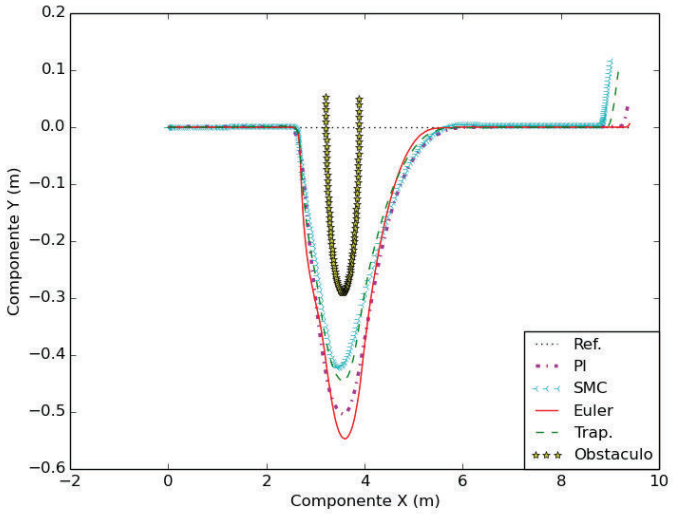


Figura 4.31. Trayectoria deseada y trayectorias del robot con los cuatro controladores durante la evasión de obstáculos

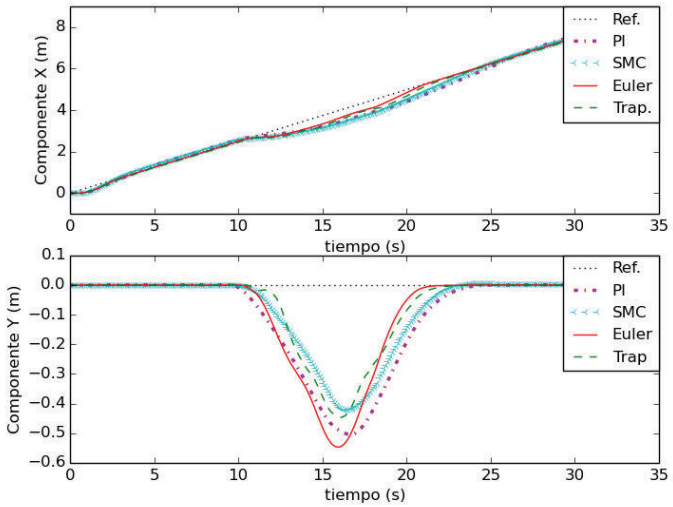


Figura 4.32. Trayectoria en X y Y del robot con los cuatro controladores durante la evasión de obstáculos

En el caso del ángulo de orientación indicado en la Figura 4.33 se presenta un resultado similar al anterior, donde el ángulo cambia desde el momento en que se inicia la evasión hasta que esta termina.

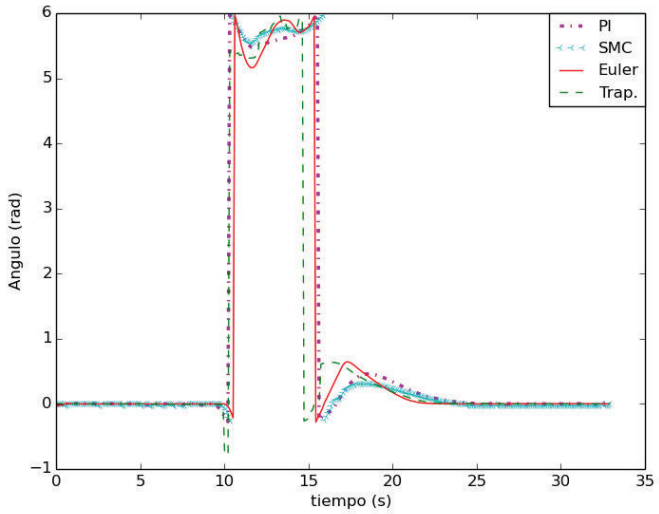


Figura 4.33. Ángulo φ del robot para los cuatro controladores durante la evasión de obstáculos

En la gráfica de velocidades lineal y angular (Figura 4.34) se nota claramente que la evasión inicia en t = 11 (s), pues la velocidad lineal disminuye drásticamente, y luego se recupera lentamente hasta terminar la evasión, para luego incrementar hasta alcanzar el valor de referencia de trayectoria al igual que antes.

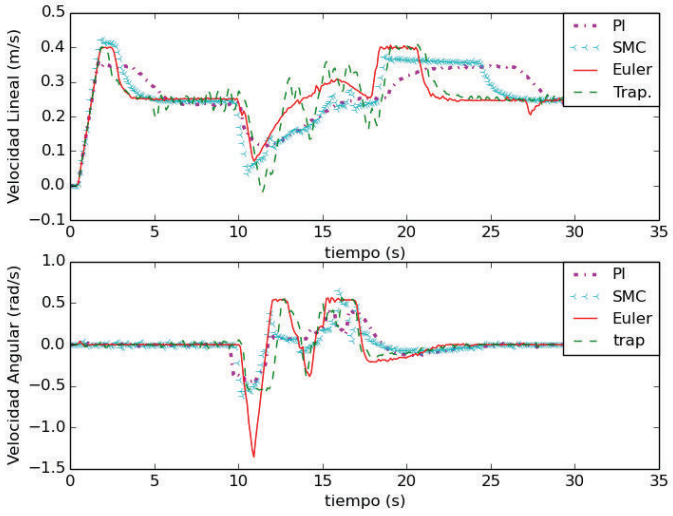


Figura 4.34. Velocidad lineal y angular del robot para los cuatro controladores durante la evasión de obstáculos

4.1.3.3 Experimento de evasión de obstáculos y velocidad lineal de 0.5 (m/s)

A la máxima velocidad lineal todos los controladores inician la evasión prácticamente al mismo tiempo. Sin embargo en la Figura 4.35 y Figura 4.36 se nota que el mejor desempeño lo presentan los controladores Trapezoide y por modos deslizantes pues se alejan menos de la trayectoria original, aunque tardan más en estabilizarse que Euler. A esta velocidad lineal también se nota que los controladores de Euler y PI se alejan más de 0.8 (m) de la referencia, por lo que no es aconsejable elegir estos controladores para evasión de obstáculos a esta velocidad.

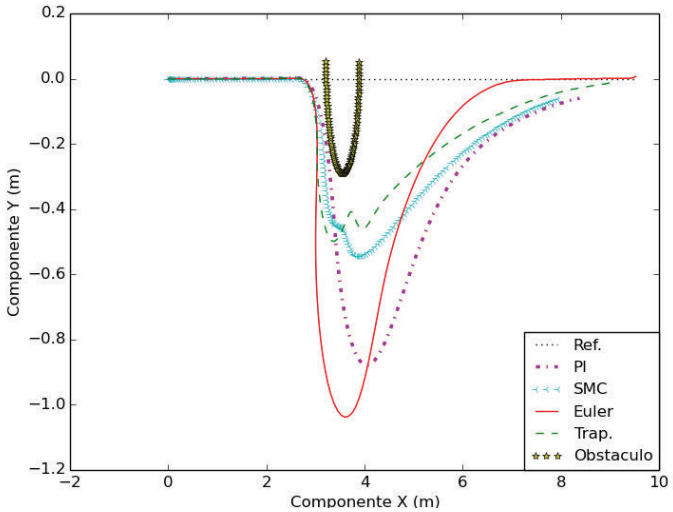


Figura 4.35. Trayectoria deseada y trayectorias del robot con los cuatro controladores durante la evasión de obstáculos

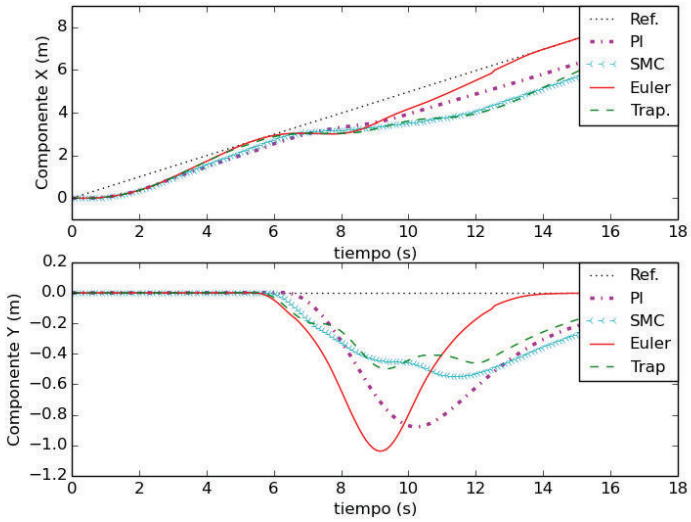


Figura 4.36. Trayectoria en X y Y del robot con los cuatro controladores durante la evasión de obstáculos

Al analizar el ángulo de orientación (Figura 4.37) se observa el mismo comportamiento de los casos anteriores, con la diferencia de que ahora Trapezoide presenta los cambios más bruscos en el ángulo.

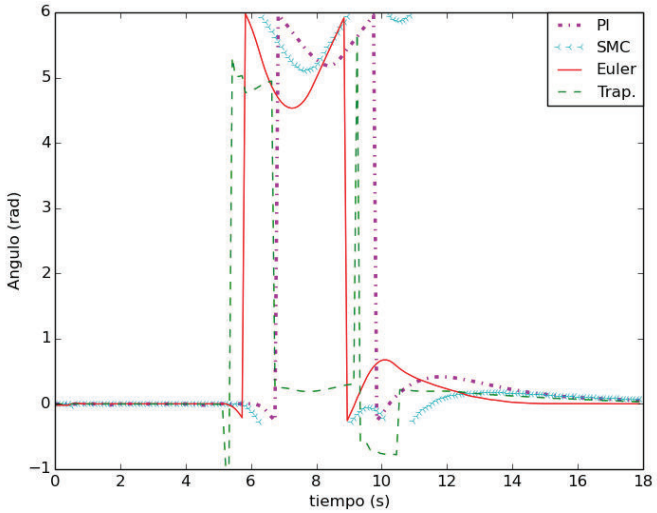


Figura 4.37. Ángulo φ del robot para los cuatro controladores durante la evasión de obstáculos

En la Figura 4.38 se presenta la gráfica de velocidades lineal y angular. Se observa que la evasión inicia en t = 5 (s) aproximadamente y el comportamiento de los controladores es similar a lo observado anteriormente: se disminuye la velocidad lineal para la evasión y luego aumenta para alcanzar la referencia. Cabe notar que los controladores Euler y Trapezoide presentan mayores velocidades para intentar llegar a la trayectoria de referencia.

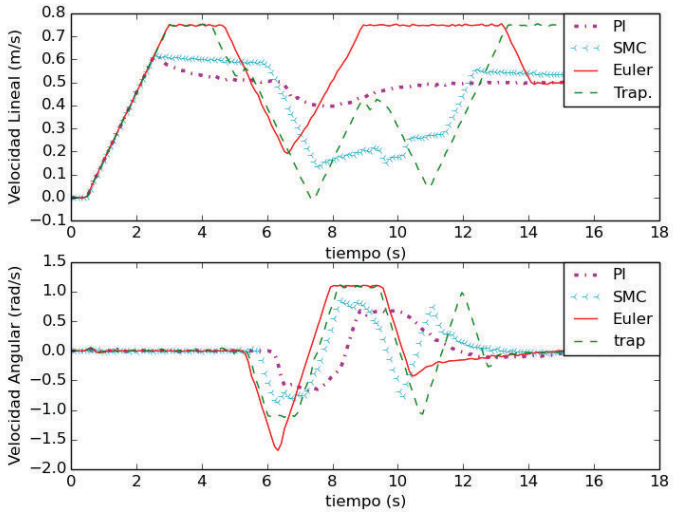


Figura 4.38. Velocidad lineal y angular del robot para los cuatro controladores durante la evasión de obstáculos

Analizando los resultados se puede concluir que para bajas velocidades Euler reacciona, evade y se estabiliza más rápido, mientras que para altas velocidades es el controlador por modo deslizante es el que presenta el mejor desempeño al evadir un obstáculo.

Sin embargo, cabe notar que aunque el controlador Trapezoidal está basado en álgebra lineal al igual que Euler, para moderadas y altas velocidades presenta un desempeño muy bueno, similar al de modo deslizante, evadiendo y estabilizándose rápido ante la presencia de un obstáculo.

4.1.4 PERTURBACIONES

Para el experimento de comportamiento ante perturbaciones se ha elegido una trayectoria circular definida por $\vec{x} = r\ cos(\omega\vec{t})$ y $\vec{y} = r\ sin(\omega\vec{t})$ donde $r = 2\ (m)$ es el radio del círculo y $\omega = \frac{v}{r}$ es la velocidad angular de la trayectoria de referencia.

La perturbación es un valor aleatorio que se añade por software durante cierto tiempo a las velocidades lineal y angular del robot, que son las señales de entrada al sistema, de acuerdo a la siguiente desigualdad:

$$-2v < v_{perturbaciòn} < 2v \tag{4.2}$$

Donde:

v : valor de velocidad lineal o angular del robot

$v_{perturbaciòn}$: valor de perturbación

Esta prueba se ha implementado a una velocidad lineal de 0.25 (m/s) y se ha ingresado la perturbación durante cinco segundos a los 25 segundos de haber empezado el experimento.

El período de muestreo al igual que en pruebas anteriores es de $T_0 = 0.1\ (s)$ con las siguientes condiciones iniciales $[x_0, y_0, \varphi_0]^T = [0,0,0]^T$ y $[v_0, \omega_0]^T = [0,0]^T$.

Para el experimento se presentan cuatro gráficos: trayectoria, las componentes de la trayectoria x y y, las velocidades lineal y angular, y el ángulo del robot.

4.1.4.1 Experimento de reacción ante perturbación con velocidad lineal de 0.25 (m/s)

El valor de la perturbación, si bien posee un valor considerable respecto al valor de las referencias de velocidad lineal y angular (Figura 4.42) no afecta de mayor forma a la trayectoria que describe el robot, como se evidencia en la Figura 4.39. En la Figura 4.40 apenas se nota la acción de la perturbación en la gráfica de la componente Y vs el tiempo.

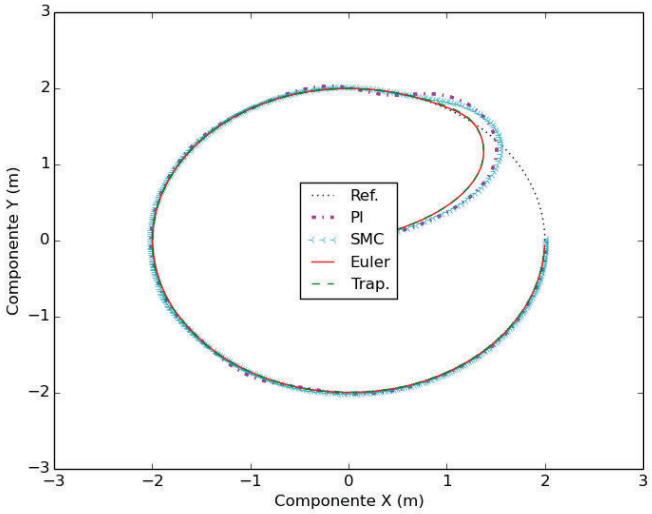


Figura 4.39. Trayectoria deseada y trayectorias del robot con los cuatro controladores (con presencia de perturbación)

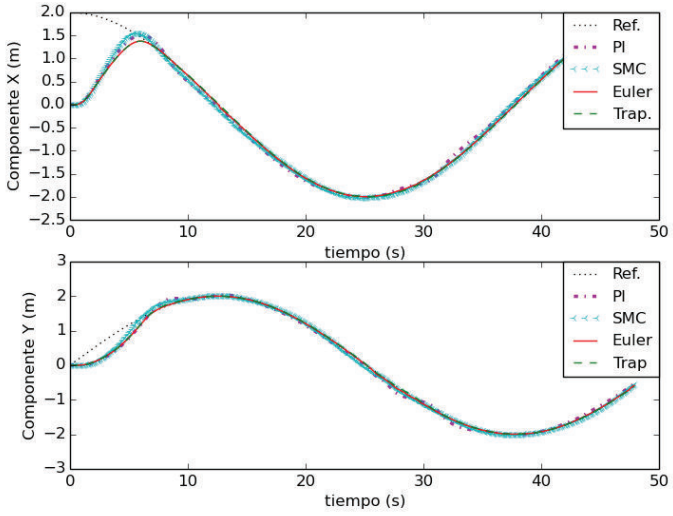


Figura 4.40. Trayectoria en X y Y del robot con los cuatro controladores (con presencia de perturbación)

De igual manera al observar el ángulo de orientación del robot presentado en la Figura 4.41, a partir de t = 25 (s) se nota apenas la perturbación en el ángulo, con valores inferiores a 0.2 (rad). Sin embargo, la perturbación provoca que el controlador PI oscile por algunos periodos de muestreo.

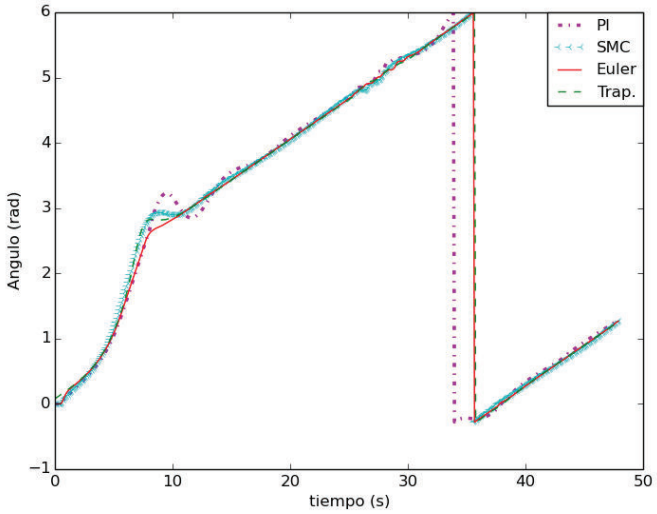


Figura 4.41. Ángulo φ del robot para los cuatro controladores ante una perturbación

Puesto que el valor de perturbación se añadió a los valores de velocidades, resulta lógico que sea en la Figura 4.42 en donde se observe su efecto. Aquí se nota que mientras los controladores Euler, Trapezoide y por modo deslizante sobrellevan bien la acción de la perturbación, volviendo a las velocidades referenciales de la trayectoria inmediatamente al terminar la perturbación; el controlador PI presenta problemas para su estabilización hasta 10 segundos después de que se ha terminado la perturbación.

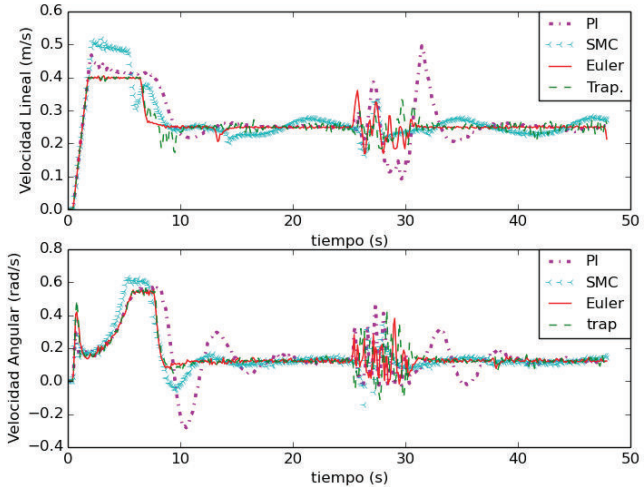


Figura 4.42. Velocidad lineal y angular del robot para los cuatro controladores (con presencia de perturbación)

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

- De las pruebas realizadas se puede concluir que para trayectorias sin cambios bruscos como las circulares, los controladores Euler y Trapezoide son más adecuados pues a bajas velocidades alcanzan la referencia con menor tiempo de establecimiento, en cambio los controladores PI y modo deslizante son más rápidos pero presentan sobretiro. Así mismo para trayectorias circulares a velocidades altas, los controladores Euler, Trapezoide y modo deslizante logran seguir la referencia presentando pequeños errores de trayectoria, mientras que el controlador PI no es capaz de alcanzar el setpoint.

  Analizando los resultados se concluye también que los controladores PI y modo deslizante son más aptos para trayectorias cuadradas, debido a que son más rápidos reaccionando ante cambios bruscos en la referencia, presentando menor sobrepico que los controladores Euler y Trapezoide, aunque el controlador PI presenta oscilaciones hasta estabilizarse.

  En lo que se refiere a evasión de obstáculos se concluye que se puede escoger cualquier método de control para velocidades bajas, debido a que todos logran evadir la colisión sin alejarse mucho de la trayectoria de referencia. No obstante se recomiendan los controles por modo deslizante y Trapezoide debido a que para velocidades altas estos evitan la colisión con el menor alejamiento de la trayectoria de referencia.
- El controlador de Euler funciona muy bien con todas las trayectorias probadas y presenta un error en estado estable muy cercano a cero. Sin embargo, para lograr aplicarlo es necesario tener un conocimiento preciso del modelo, pues de la fidelidad con que este se ha identificado depende el adecuado desempeño del controlador.

- El controlador Trapezoide al ser una variación del método anterior, con la diferencia que modifica la manera en que se calculan los puntos de movimiento futuros, resulta en controladores más extensos; lográndose reacciones más rápidas que el control clásico de Euler, sin embargo, esto también provoca un mayor tiempo de establecimiento y un ligero incremento del error en estado estable.
- El controlador PI al estar aplicado a un modelo aproximado de la plataforma robótica presenta inconvenientes como sobretiro en el seguimiento de trayectorias y un tiempo de establecimiento mayor que los otros controladores con los que se lo compara. Sin embargo, su capacidad de reacción ante cambios bruscos en la trayectoria, lo ponen como una opción válida ante este tipo de situaciones.
- El controlador por modo deslizante al igual que el PI que usa como superficie deslizante, usa un modelo aproximado, teniendo también como inconveniente el sobretiro en la trayectoria. Sin embargo, este controlador destaca en primer lugar por su tiempo de establecimiento que es más corto que el control PI, y segundo destaca por su rápida respuesta ante cambio bruscos en la trayectoria sin casi presentar oscilaciones en la curva.
- La parte discontinua o switching que posee la ecuación de modos deslizantes hace que el controlador presente una mejor reacción a cambios inesperados en comparación con los controladores basados en métodos numéricos, sin embargo si no se eligen las constantes que conforman dicha ecuación de forma adecuada el robot puede presentar oscilaciones continuas a lo largo de la trayectoria.
- Se pudo apreciar un pequeño error en estado estable en los resultados de los controladores para velocidades no muy elevadas (0.1 a 0.3 (m/s)), sin embargo, se debe considerar que su magnitud (alrededor de los 2-4cm) es despreciable respecto a las dimensiones de la plataforma (40 (cm) de lado, [30]), notándose un ligero incremento del error en estado estable para los controladores basados en PI.
- El adelantar el punto de interés (Figura 1.3, [3]) del robot móvil permitió mejorar la holonomía del modelo cinemático, permitiendo que este punto tenga velocidades en X y Y, sin embargo, el cálculo de la restricción para

que los controladores basados en métodos numéricos tengan una componente en el espacio nulo igual a cero es la misma, tanto para el modelo mejorado (presentado en (2.28)) como no mejorado (presentado en (2.43)).

- El cálculo de las acciones de control es de suma relevancia para el seguimiento de trayectorias, sin embargo, para implementar los algoritmos se deben solucionar problemas como el correcto cálculo del ángulo y las saturaciones a los valores calculados de referencia de velocidad, pues si no se toman en cuenta estos aspectos el seguimiento de trayectorias presenta sobretiro u oscilaciones..
- La variación del método de evasión de obstáculos por impedancia usado en este proyecto, presentó un desempeño satisfactorio debido a que no hubo presencia de colisiones en las pruebas realizadas. El método fue rápidamente adaptado a los sensores con los que cuenta la plataforma robótica, y el cálculo de los parámetros de evasión es relativamente sencillo comparado con el método original.
- El uso del Sistema Operativo Robótico ROS ofrece grandes ventajas respecto al software propietario de cada robot, entre las cuales destacan la forma de integración entre diferentes plataformas robóticas, su facilidad de trabajo dentro de una entorno multiusuario, y el soporte de los desarrolladores y la comunidad; todo esto a cambio de traducir el software del usuario a un formato estándar determinado.
- El paquete RosAria de ROS ofrece muchas facilidades, entre ellas la transformación de los datos de los sensores en información relevante para el control, y la transformación de los datos de control en señales de entrada para el robot. Resultaron particularmente útil para este proyecto las funciones de tratamiento previo de los datos, entre los cuales se incluyen la inclusión del mejoramiento no holonómico, filtrado de las señales de comunicación y la corrección de odometría.
- La pruebas realizadas en la plataforma robótica Pioneer 3DX y las realizadas en el software simulación MobileSim arrojan resultados muy similares entre sí, permitiendo al desarrollador realizar pruebas en el simulador para luego aplicarlas a la plataforma sin que los parámetros de calibración varíen demasiado.

- El trabajo con el lenguaje de programación Python presenta varias ventajas respecto a C++, pues es más intuitivo y amigable con el usuario, además de la facilidad que presenta para utilizar funciones, pareciéndose mucho al lenguaje de programación propietario Matlab.

5.2 RECOMENDACIONES

- Antes de utilizar la plataforma robótica Pioneer 3DX se recomienda revisar cuidadosamente su manual de operación, para lograr comprender los conceptos de operación de la odometría, funcionamiento de sensores y comunicación con el computador externo.
- El robot Pioneer 3DX puede funcionar con una, dos o tres baterías, sin embargo siempre es necesario colocarlas de forma balanceada, de manera que no altere la operación del controlador debido al movimiento del centro de gravedad causada por la desproporción de peso en uno de los lados del robot.
- Se recomienda otorgar permisos de super usuario a los puertos USB del ordenador antes de conectar el adaptador USB-Serial, pues caso contrario no se podrá efectuar la comunicación con el robot.
- Se aconseja que la red utilizada para la comunicación no esté saturada o con mucho tráfico de datos, debido a que pueden presentarse intermitencias en la presentación de información en la interfaz. De ser posible, se recomienda utilizar una red exclusiva con un rango de alcance lo suficientemente grande para que no existan fallas en la comunicación.
- Se recomienda que las trayectorias a las que se someta el robot tengan en cuenta las limitaciones físicas del mismo, pues de tener curvas o esquinas muy cerradas inevitablemente se tiene fallas en el seguimiento de la trayectoria o errores de trayectoria muy grandes.
- No se recomienda variar los parámetros propios del robot, tales como los parámetros del PID internos, dimensiones del robot, y demás valores que se encuentran dentro del paquete RosAria, debido a que los mismos ya han sido previamente calibrados y comprobados.

- El usuario mediante la interfaz puede ingresar perturbaciones al sistema, teniendo como parámetros el momento y duración en las que se desea que ocurran. La sumatoria del momento de ocurrencia más el tiempo que dure la perturbación no debe ser mayor al tiempo de duración del experimento, debido a que después de la perturbación el robot necesita tiempo para regresar a la trayectoria y estabilizarse.

REFERENCIAS BIBLIOGRÁFICAS

[1] A. Ollero, “Robótica: manipuladores y robots móviles”, Primera Edición, Editorial Marcombo, 2001

[2] A. Rosales, G. Scaglia, V. Mut, F. Di Sciascio, “Control dinámico mediante métodos numéricos para robots móviles tipo uniciclo”, Universidad Nacional de San Juan, Argentina, 2008

[3] A. Rosales, G. Scaglia, V. Mut, F. Di Sciascio, “Navegación de Robots Móviles en Entornos no Estructurados utilizando Álgebra Lineal”, en Revista Iberoamericana de Automática e Informática Industrial, ISSN: 1697-7912, Vol. 6, Núm. 2, Abril 2009, pp.79-88

[4] A. Zinober, “Deterministic control of uncertain systems”, London: Peter Peregrinus Press, 1990

[5] *ARIA Manual*, Mobile Robots, ActivMedia Robotics. [Online] Disponible en: http://robots.mobilerobots.com/wiki/ARIA#Ubuntu_12_64-bit

[6] B. Kuo, “Sistemas de Control Automático”, Editorial Pearson Educación, 1996, pp. 838-841

[7] Beijer Electronics, “What is HMI?”. [Online] Disponible en: http: http://www.beijerelectronics.com/web/web_en_be_com.nsf/AllDocuments/3A92C047F1F9BC6FC1256EC1004A630C

[8] C. Aimacaña, “Interfaz de Usuario”. [Online] Disponible en: http: http://www.monografias.com/trabajos6/inus/inus.shtml#ixzz3QK9pSCHv

[9] C. De la Cruz, “Control de formación de robots móviles”, PhD. Dissertation, Universidad Nacional de San Juan, San Juan, Argentina, Diciembre 2006

[10] D. Himmelblau, “Applied Nonlinear Programming”, McGraw-Hill, New York

[11] D. Poole, “Álgebra lineal. Una introducción moderna”, Cengage Learning Editores, 2011, pp. 588-589

[12] Departamento de Matemáticas CSI/ITESM, “Espacios de una Matriz”, México, 2008, [Online] Disponible: http://www.mty.itesm.mx/etie/deptos/m/ma00-130/lecturas/m130-04a.pdf

[13] Diccionario Oxford, Definición de robótica, [Online] Disponible: http://www.oxforddictionaries.com/es/definicion/ingles_americano/robot

[14] DISA, “Introducción al control robusto”, Universidad de Valladolid, [Online] Disponible: http://www.isa.cie.uva.es/~fernando/papers/doctorado_parte1.pdf

[15] E. Chavez, “Integración numérica en el tiempo”, Universidad de Castilla-La Mancha, España, 2010

[16] E.O. Torres, “Study of Wheel slip and traction forces in differential drive robots and slip avoidance control strategy”, USA, 2014. [Online] Disponible en: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6859308

[17] G. Andaluz, “Modelación, Identificación y Control de robots móviles”. Bachelor dissertation, Escuela Politécnica Nacional, 2011. [Online] Disponible: http://bibdigital.epn.edu.ec/bitstream/15000/4321/1/CD-3931.pdf

[18] G. Oriolo, “Control of Nonholonomic Systems”, Universidad de Roma “La Sapienza”, Italia, 2005. [Online] Disponible en: http://www.dis.uniroma1.it/~oriolo/cns/cns_slides.pdf

[19] G. Scaglia, “Control de sistemas basados en algebra lineal”, Curso dictado en la Escuela Politécnica Nacional, Mayo-Agosto 2014

[20] G. Strang, “Linear Algebra and its Applications”, Cuarta Edición, Ed. Academic Press, New York, 1980, pp. 79-100

[21] *Introducing Qt Creator*, Qt Documentation. [Online] Disponible en: http://doc.qt.digia.com/qtcreator-2.3/creator-overview.html

[22] J. Acedo, “Instrumentación y control avanzado de procesos”, Ed. Díaz de Santos, México, 2013

[23] J. E. Slotine, “Sliding controller design for non-linear systems”, International Journal of Control, Vol. 40, Issue 2, Noviembre 1984, pp. 421-434

[24] K. Ogata, “Modern Control Engineering”, Cuarta edición, Editorial Digital Design, 2010, cap. 10, pp. 682-700

[25] M. Mataric, “The Basics of Robot Control”. [Online] Disponible en: http://www-robotics.usc.edu/~maja/robot-control.html

[26] *MobileSim*, Mobile Robots, ActivMedia Robotics. [Online] Disponible en: http://robots.mobilerobots.com/wiki/MobileSim

[27] O. Camacho, “A New Approach To Design And Tune Sliding Mode Controllers For Chemical Processes”, Ph.D. dissertation, University of South Florida, Tampa, Florida. 1996,

[28] O. Camacho, R. Rojas, W. García, “Variable Structure Control Applied to Chemical Processes with Inverse Response”, in ISA Transactions, Vol. 38, Issue 1, Enero 1999, pp. 55-72

[29] O. Camacho and R. Rojas, “A General Sliding Mode Controller for Nonlinear Chemical Processes”, en Journal of Dynamic Systems, Measurement and Control, Vol. 122, Diciembre 2000, pp. 650-655

[30] *Pioneer 3DX Brochure*, Mobile Robots, ActivMedia Robotics. [Online] Disponible en: http://www.mobilerobots.com/Libraries/Downloads/Pioneer3DX-P3DX-RevA.sflb.ashx

[31] *Pioneer 3™ & Pioneer 2™ Operations Manual*, Mobile Robots, ActivMedia Robotics, version 3, 2003

[32] *Qt User Manual*, Qt Documentation. [Online] Disponible en: http://qt-project.org/doc/qt-4.8/designer-manual.html

[33] R. Carelli, “Control de robots móviles”, Escuela de Ingenieros Industriales, Madrid, 2009, pp. 48

[34] *Ros.org Documentation,* ROS Documentation. [Online] Disponible en: http://wiki.ros.org/

[35] S. Martin, Effective Visual Communication for Graphical User Interfaces. [Online] Disponible en: http://web.cs.wpi.edu/~matt/courses/cs563/talks/smartin/int_design.html

[36] S. Tran, “Ubuntu 14.04.2 LTS Released, Includes 3.16 Kernel”. [Online] Disponible en: http://www.omgubuntu.co.uk/2015/02/ubuntu-14-04-2-lts-released-includes-3-16-kernel

[37] Simple Organization, “Tipos de Robots”. [Online] Disponible en: http://www.tiposde.org/general/460-tipos-de-robots/

[38] Stanford U. Courses, “Robotics: A Brief History”, [Online] Disponible en: http://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/history.html

[39] T. Kealy, A. O’Dwyer, “Analytical ISE Calculation and Optimum Control System Design”, IISC Conferencia Irlandesa de Señales y Sistemas, Universidad de Limerick, Irlanda, 2003

[40] U. Nehmzow, “Mobile Robotics: Research, Applications and Challenges”. [Online] Disponible en: http://citeseerx.ist.psu.edu/

[41] V. Hlaváč, “Robot trajectory planning”, Universidad Técnica Checa en Praga, [Online] Disponible: http://cmp.felk.cvut.cz/~hlavac/TeachPresEn/55IntelligentRobotics/080ManipulatorTrajectoryPlanning.pdf

[42] V. I. Utkin, “Variable Structure Systems with Sliding Modes”, en IEEE Transactions on Automatic Control, ISSN: 0018-9286, Vol. 22, Issue 2, Abril 1977, pp. 212-222

ANEXOS

A. MANUAL DE USUARIO

A.1. SISTEMA OPERATIVO

La presente guía muestra los pasos necesarios para poder ejecutar los controladores sobre la plataforma Pioneer 3DX.

El sistema operativo sobre el cual se está corriendo los controladores tiene impacto sobre la instalación de los diferentes elementos necesarios para la ejecución de los mismos tal como el núcleo ROS y RosAria, sin embargo no tienen ningún impacto en la ejecución del controlador.

Es recomendable que tanto el computador maestro como el esclavo tengan las mismas versiones de sistema operativo, ya que como la instalación del núcleo depende del sistema operativo, si los ambientes de trabajo son diferentes, las versiones del núcleo ROS serán diferentes y provocarían problemas en la comunicación inalámbrica. Además es necesario aclarar que ambas computadoras necesitan instalar y compilar todo el software que se indica en este manual, desde la instalación del núcleo hasta la preparación de archivos.

La versión escogida para la realización de la tesis fue la 14.04.1 Trusty Tahr (Figura A - 1), debido a que la misma fue la más actual en el momento de realizarse el presente proyecto y además porque la misma posee LTS (Apoyo a Largo Plazo):

Figura A - 1. Logo de Ubuntu 14.04.1 Trusty Tahr, [36]

Recomendación: Se sugiere instalar el sistema operativo en idioma inglés debido a que la mayoría del soporte para ROS y RosAria está orientado a carpetas que se encuentran en dicho idioma, por lo que algunos comandos podrían ser incompatibles con el idioma español.

A.2. INSTALACIÓN DEL NÚCLEO ROS ÍNDIGO

Antes de empezar con la presente guía es necesario recalcar que la misma fue elaborada para ejecutar los controladores en el sistema operativo Ubuntu versión 14.04 (Trusty Tahr-Idioma Ingles)

A.2.1. CONFIGURACIÓN DE REPOSITORIOS

Antes de iniciar con la instalación de Ros, es necesario preparar al sistema operativo, para esto se debe cerciorar que el sistema acepte software de terceros siguiendo los siguientes pasos:

- Ingresar a “Ubuntu Software Center”
- Ingresar al menú” Edit-Software Source”
    - Marcar todas las opciones tal como se muestra en la Figura A - 2.

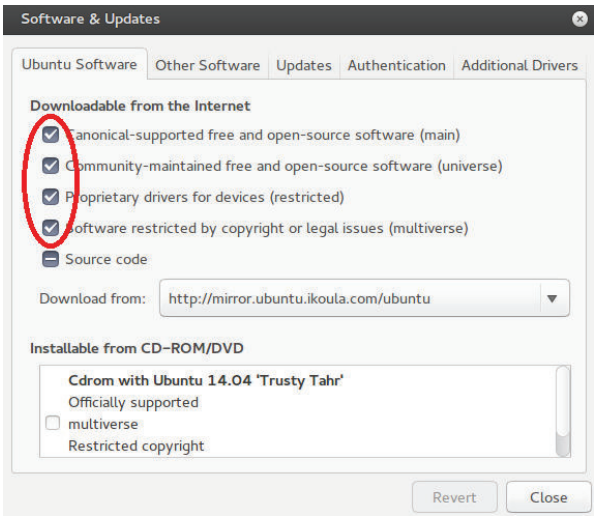


Figura A - 2. Ventana de Diálogo "Software Source"

A.2.2. CONFIGURAR LAS FUENTES DE DESCARGA

En Ubuntu la instalación y descarga de los programas se hace de forma casi automática, sin embargo para esto primero se debe proporcionar cierta información al sistema, la primera de ellas es especificar de qué lugar se van a descargar los

archivos necesarios para la instalación, esto se logra copiando la siguiente información en la terminal de Ubuntu:

- `sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" > /etc/apt/sources.list.d/ros-latest.list'`

A.2.3. DEFINIR LAS LLAVES

Esta llave permite tener acceso a todos los repositorios necesarios para la instalación, para esto se ingresa el siguiente comando en la terminal de Ubuntu:

- `wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - | sudo apt-key add -`

A.2.4. INSTALACIÓN DEL NÚCLEO ROS

Una vez definido las fuentes y los permisos para la descarga basta con copiar los siguientes comandos y la descarga e instalación se realizara de manera automática.

- `sudo apt-get update`
- `sudo apt-get install ros-indigo-desktop-full`

A.2.5. INICIALIZACIÓN DE "ROSDEP"

Rosdep es una herramienta que permite la instalación rápida y sencilla de dependencias adicionales del núcleo ROS, este es necesario para poder realizar la instalación y compilación de RosAria. Para iniciarlo se ingresan los siguientes comandos en la terminal de Ubuntu:

- `sudo rosdep init`
- `rosdep update`

A.2.6. CONFIGURACIÓN DEL MEDIO AMBIENTE

Hay comandos que siempre deben ingresarse en la terminal de Ubuntu antes de poder iniciar el núcleo ROS, para que esto se haga de manera automática se debe ingresar los siguientes comandos:

- `echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc`
- `source ~/.bashrc`

A.2.7. INSTALACIÓN DE "ROSINSTALL"

Rosinstall se usa con mucha frecuencia para la rápida descarga e instalación de paquetes para el uso en el núcleo ROS, para instalarlo se debe ingresar el siguiente comando en la terminal de Ubuntu:

- `sudo apt-get install python-rosinstall`

A.3. CREACIÓN DEL ESPACIO DE TRABAJO

El espacio de trabajo es lugar donde se encuentran los paquetes que el usuario crea, y donde los controladores estos se ejecutan, para eso se deben ingresar los siguientes comandos en la terminal de Ubuntu:

- `. /opt/ros/indigo/setup.bash`
- `mkdir -p ~/catkin_ws/src`
- `cd ~/catkin_ws/src`
- `catkin_init_workspace`
- `cd ~/catkin_ws/`
- `catkin_make`
- `source devel/setup.bash`
- `echo $ROS_PACKAGE_PATH`
  `/home/USER/catkin_ws/src:/opt/ros/indigo/share:/opt/ros/indigo/stacks`

En los comandos se debe tener cuidado cuando aparecen direcciones, debido a que en las mismas el término "USER" debe ser reemplazado con el nombre de Usuario que haya sido elegido en la instalación de Ubuntu, un ejemplo de esto se muestra en la Figura A - 3.

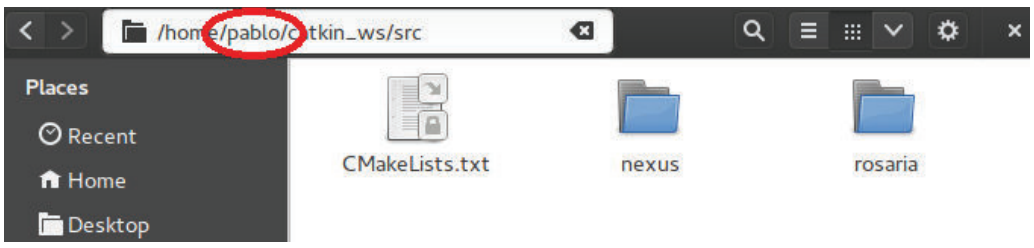


Figura A - 3. Ilustración de reemplazo "USER"

A.4. INSTALACIÓN DE ROSARIA

A.4.1. CONFIGURACIÓN DE REPOSITORIOS

Al igual que cuando se instaló el núcleo ROS, es necesario configurar los repositorios donde se encuentra el Software RosAria, para esto se ingresa los siguientes comandos en la terminal:

- `cd ~/catkin_ws/src`
- `git clone https://github.com/amor-ros-pkg/rosaria.git`

A.4.2. INSTALACIÓN DE ARIA Y CONSTRUCCIÓN DE ROSARIA

Los siguientes comandos a escribirse en la terminal de Ubuntu tienen la función de descargar e Instalar el software “Aria” provisto por MobileRobots, el cual se usa como base para la construcción del paquete RosAria, una vez instalado dicho software es necesario la creación y compilación del paquete RosAria, todo esto se logra con los siguientes comandos:

- `source ~/catkin_ws/devel/setup.bash`
- `echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc`
- `source ~/.bashrc`
- `rosdep update`
- `rosdep install rosaria`
- `cd ~/catkin_ws`
- `catkin_make`

A.5. CREACIÓN DEL PAQUETE

Una vez creado y compilado el paquete RosAria es necesario establecer el paquete en el cual el Usuario va a crear los diferentes programas que le permitan ejecutar los controladores, esto se logra con los siguientes comandos:

- `cd ~/catkin_ws/src`
- `catkin_create_pkg nexus std_msgs rospy roscpp`
- `cd ~/catkin_ws`
- `catkin_make`

A.6. PREPARACIÓN DE ARCHIVOS

Una vez creados los paquetes es necesario copiar las carpetas donde se encuentran los programas los cuales contienen los controladores, para esto se debe situar en la siguiente dirección:

```
/home/USER/catkin_ws/scr/nexus
```

Una vez ahí se debe copiar las carpetas quedando una configuración como la de la Figura A - 4.

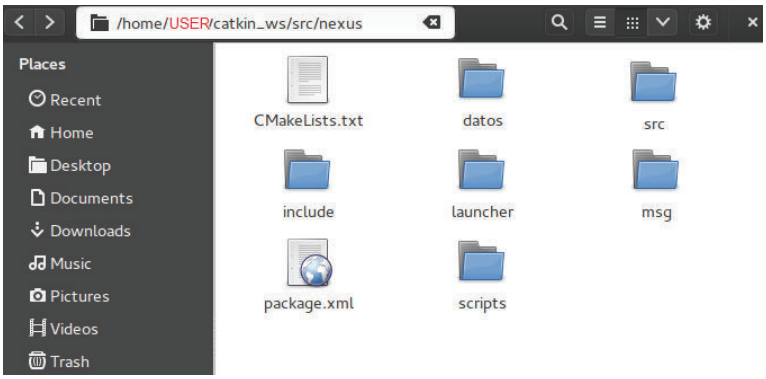


Figura A - 4. Configuración de Carpetas

Este procedimiento debe hacerse para la computadora que sirve como maestro y como esclavo, las carpetas a copiarse son las mismas pero el contenido de ellas es diferente dependiendo si la carpeta es la del maestro o el esclavo.

Una vez hecho esto se debe dar permiso de ejecución a los diferentes programas, esto debe hacerse para todo los programas dentro de la carpeta “scripts”, a manera de ejemplo se lo hace para el archivo de la Figura A - 5.

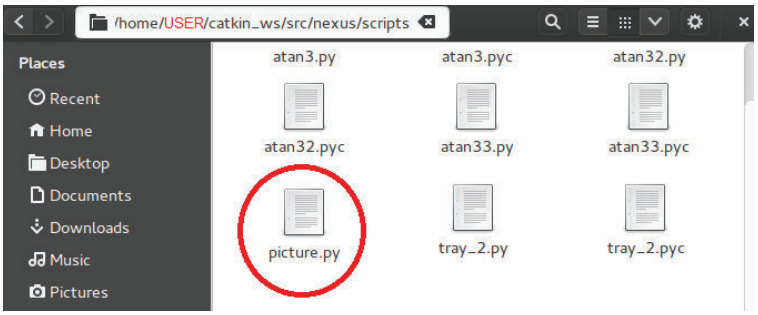


Figura A - 5. Archivo de Ejemplo

Los comandos son los siguientes:

- `cd /home/USER/catkin_ws/scr/nexus/scripts`

- `chmod +x picture.py`

El paso final es reemplazar el Usuario en los diferentes programas por el propio Usuario usado el momento de la instalación de Ubuntu, a manera de ejemplo:

- `savefig('/home/USER/catkin_ws/src/nexus/datos/VS/todos vs todos/Resultados.png')`

A.7. INSTALACIÓN DE QT

La instalación y programación del programa Qt es mucho más fácil que la instalación del núcleo ROS, esto se logra abriendo el Ubuntu Software Center (Centro de Software de Ubuntu), escribiendo “qt creator” en el buscador y luego seleccionando la opción instalar tal como se ve en la Figura A - 6.

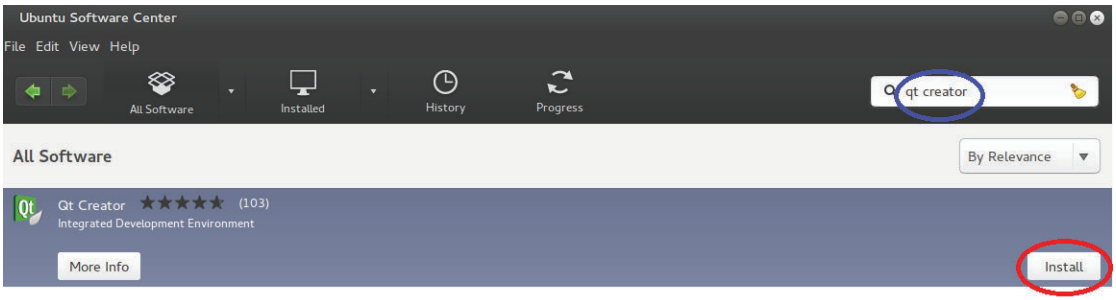


Figura A - 6. Instalación de Qt

Una vez instalado el programa que permite crear la interfaz, es necesario copiar las carpetas necesarias para su funcionamiento. Estas carpetas deben quedar tal como se ve en la Figura A - 7.

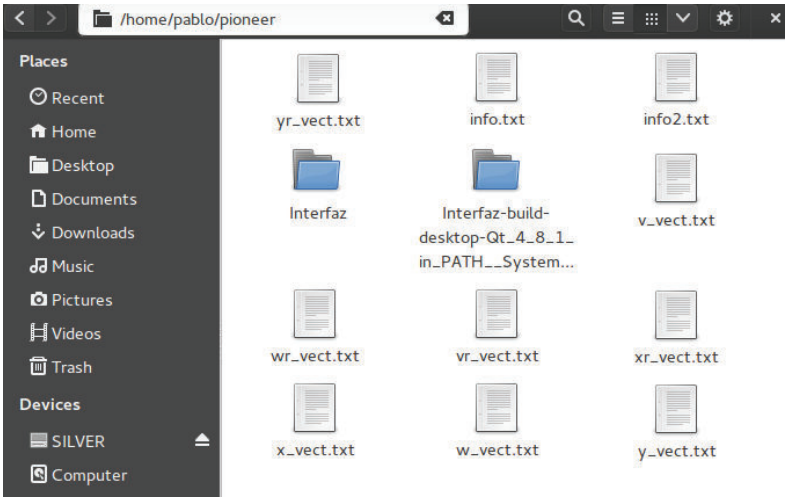


Figura A - 7. Configuración de Carpetas Qt

A.8. PROCEDIMIENTO DE EJECUCIÓN

Para poder ejecutar los controladores es necesario seguir una serie de pasos, los cuales permiten al usuario ejecutar los controladores sobre la plataforma pioneer 3DX, para seguir estos pasos es necesario tener una computadora que haga el papel de maestro y otra que haga el papel de esclavo.

A.8.1. CONEXIÓN CON EL ROBOT PIONEER 3DX

Se debe conectar al robot con la PC Máster a través del puerto serial de la plataforma y el puerto USB0 de la PC, utilizando un adaptador serial – USB. A continuación se ingresa en el terminal de la PC el siguiente comando:

- `sudo chmod 777 /dev/ttyUSB0`

A.8.2. OBTENER LAS IP

Si se va a trabajar en un sistema maestro-esclavo es necesario referenciar las IP del maestro el cual arranca el núcleo ROS y los esclavos los cuales usan el núcleo del Maestro, para esto es necesario saber cuál es la IP entrando a “system settings-network” y seleccionar la red a la que estén conectadas ambas computadoras, la IP de la computadora se muestra tal como se ve en la Figura A - 8.

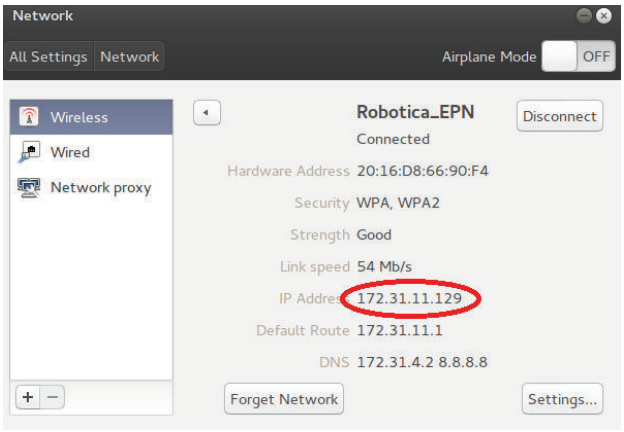


Figura A - 8. IP del Computador

A.8.3. EXPORTAR Y REFERENCIAR LA IP

En el maestro la IP debe ser exportada, para esto en la terminal se ingresa el siguiente comando:

- `export ROS_IP=172.31.11.129`

En el esclavo la IP debe ser exportada y la IP del maestro debe ser referenciada ingresando estos comandos en la terminal

- `export ROS_IP=172.31.11.130`
- `ROS_MASTER_URI=http:// 172.31.11.129:11311/`

A.8.4. ARRANQUE DEL MAESTRO

Primero el maestro debe arrancar el núcleo ROS, para esto en la terminal se debe ingresar el siguiente comando:

- `roslaunch nexus rosaria_3dx.launch`

Teniendo un resultado como el de la Figura A - 9.

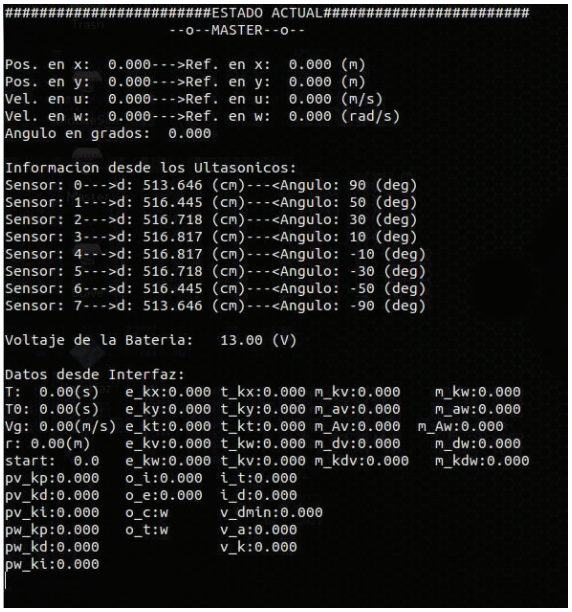


Figura A - 9. Arranque del Maestro

A.8.5. ARRANQUE DEL ESCLAVO

El procedimiento para el arranque del esclavo es similar al del maestro, pero antes del arranque se debe cerciorar que el esclavo esta enlazado con el maestro, esto se logra con el siguiente comando:

- `Rosnode list`

Una vez ingresado este comando se debería tener un resultado como el de la Figura A - 10, caso contrario se debería revisar si la conexión es estable.

```
linda@lijecaru:~$ rosnode list
/R/RosAria
/R/nodo_arranque
/R/send_data_2
/rosout
linda@lijecaru:~$
```

Figura A - 10. Comprobación de Enlace

Una vez comprobada la conexión se arranca al esclavo con el siguiente comando:

- `rosrun nexus trans_recep.py`

Obteniendo un resultado como el de la Figura A - 11.

```
################ESTADO ACTUAL################
              --o--SLAVE--o--

Pos. en x:  0.000--->Ref. en x:  0.000 (m)
Pos. en y:  0.000--->Ref. en y:  0.000 (m)
Vel. en u:  0.000--->Ref. en u:  0.000 (m/s)
Vel. en w:  0.000--->Ref. en w:  0.000 (rad/s)
Angulo en grados:  0.000

Informacion desde los Ultrasonicos:
Sensor: 0--->d: 0.000 (cm)---<Angulo: 90 (deg)
Sensor: 1--->d: 0.000 (cm)---<Angulo: 50 (deg)
Sensor: 2--->d: 0.000 (cm)---<Angulo: 30 (deg)
Sensor: 3--->d: 0.000 (cm)---<Angulo: 10 (deg)
Sensor: 4--->d: 0.000 (cm)---<Angulo: -10 (deg)
Sensor: 5--->d: 0.000 (cm)---<Angulo: -30 (deg)
Sensor: 6--->d: 0.000 (cm)---<Angulo: -50 (deg)
Sensor: 7--->d: 0.000 (cm)---<Angulo: -90 (deg)

Voltaje de la Bateria:   13.00 (V)
Tiempo de Simulacion:    100.00 (s)
Tiempo de Muestreo:        0.10 (s)
Velocidad de Generacion:  0.25 (m/s)
Evasion: Desactivada
Perturbacion: Desactivada

Estado Actual--->En Espera
Condicion Actual--->Sin Obstaculo
Control Actual--->Control por aproximacion de Euler
Trayectoria Actual--->Trayectoria Circular
```

Figura A - 11. Arranque del Esclavo

A.8.6. ARRANQUE DE LA INTERFAZ

Para el arranque de la interfaz basta con entrar dentro de la carpeta llamada “interfaz-build-de…System_ _Release” y dar doble click en el archivo “Interfaz” tal como se ve en la Figura A - 12.

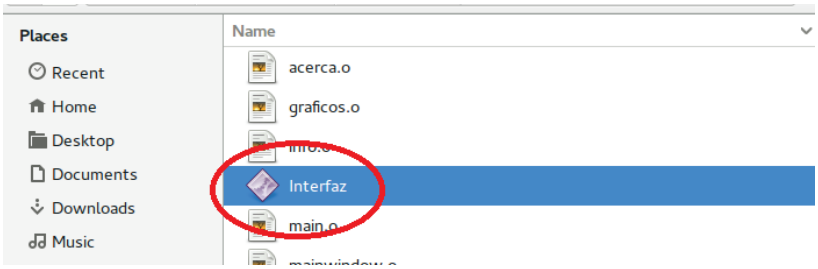


Figura A - 12. Arranque de la Interfaz

Una vez hecho esto se debería abrir una ventana con la interfaz realizada para interactuar con el usuario tal como se ve en la Figura A - 13.

Figura A - 13. Interfaz para el Usuario

A.8.7. CONFIGURACIÓN DE PARÁMETROS

Una vez abierta la Interfaz, el usuario debe ingresar los parámetros para realizar el control y la generación de trayectorias, para esto basta con cambiar los valores predefinidos en la Interfaz, de la misma forma interactuando con la interfaz se puede agregar interferencia o activar la opción de evasión de obstáculos.

Mientras el usuario varia estos parámetros en el esclavo se debería reflejar estos datos tal como se ve en la Figura A - 14.

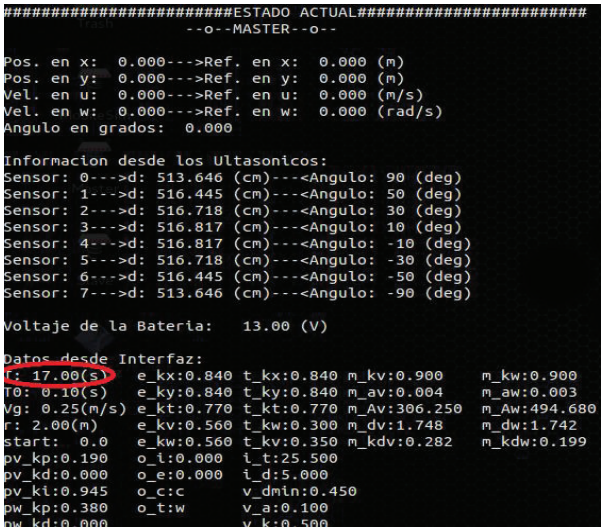


Figura A - 14. Master recibiendo datos

A.8.8. ARRANQUE DEL CONTROL

Una vez elegidos los parámetros para el control y demás, se debe presionar el botón “Empezar” en la interfaz, una vez hecho esto el seguimiento de trayectorias comienza.

El Master cambia la información que muestra a la de la Figura A - 15 y la interfaz cambia para mostrar los datos en función del tiempo tal como se ve en la Figura A - 16, adicionalmente si se presiona el botón “info” en la Interfaz se tiene información numérica del Control tal como se ve en la Figura A - 17.

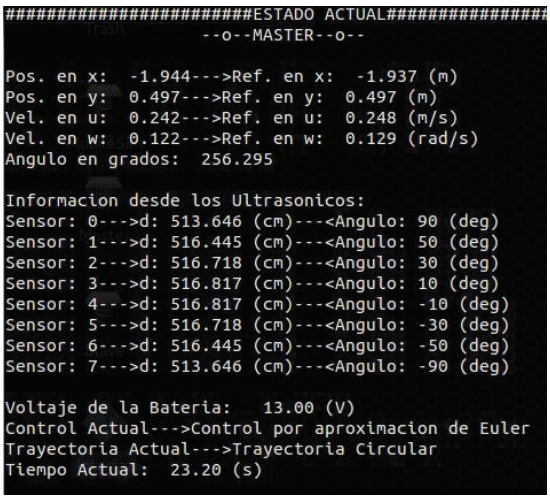


Figura A - 15. Cambio de Información del Maestro

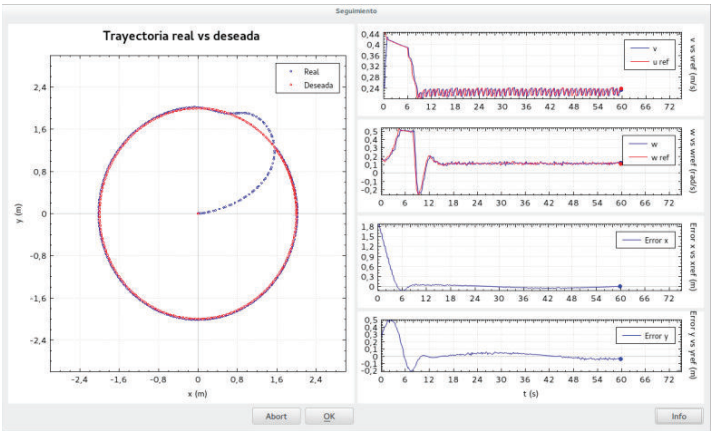


Figura A - 16. Datos en Función del Tiempo

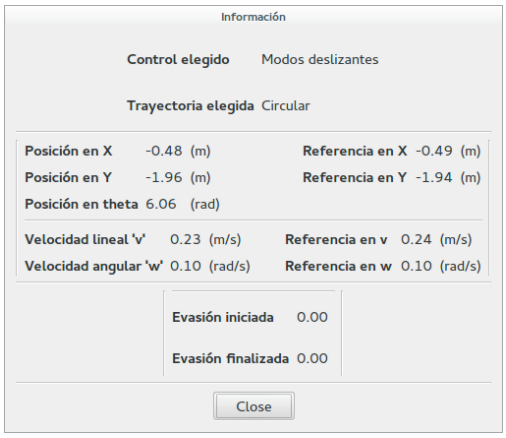


Figura A - 17. Información Numérica

A.8.9. FINALIZACIÓN DEL PROCESO

Una vez terminado el seguimiento de la trayectoria en el Maestro aparecen los resultados del seguimiento en forma gráfica tal como se ve en la Figura A - 18.

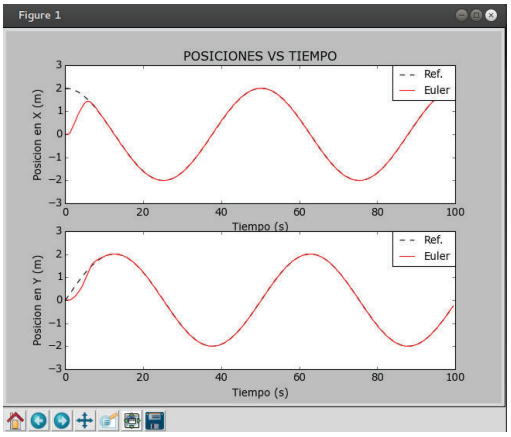


Figura A - 18. Resultados del Seguimiento

Una vez cerrados estos gráficos el lanzador arrancado se cierra automáticamente, en caso contrario se debe ir a la terminal del maestro y presionar la siguiente combinación de teclas:

- `Ctrl + C`

Una vez hecho esto se cierra el núcleo ROS, el programa que corría en la terminal del esclavo terminó automáticamente una vez finalizado el Control.

NOTA: Si se desea correr una vez más el Control, no es necesario volver a referenciar las IP, éstas deben ser referenciadas una sola vez por cada ventana de la terminal abierta.

B. NOTAS SOBRE EL USO DE QTCREATOR

B.1. CREACIÓN DE UN PROYECTO CON QTCREATOR

Al abrir el programa, la pantalla principal presenta proyectos recientes, y las opciones de nuevo proyecto y abrir proyecto. Para crear un nuevo proyecto se selecciona New Project. Después se debe elegir qué tipo de proyecto se desea crear (Figura B - 1). Aquí se selecciona Qt Widgets Application ya que permite utilizar todas las opciones de desarrollo de Qt, programación en C++ y además crear widgets personalizados. Se selecciona por default Desktop Templates o plantilla para escritorio. Al elegir otras opciones, también se habilitan otras plataformas soportadas, como MeeGo/Harmattan Symbian (desarrollo de aplicaciones para dispositivos móviles).

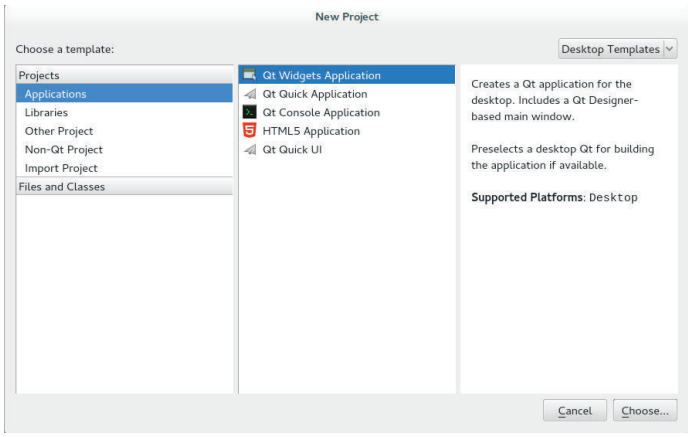


Figura B - 1. Opciones para crear nuevo proyecto

A continuación se le otorga un nombre al proyecto, y se selecciona el directorio de destino. Luego se elige un Kit para la depuración y liberación del proyecto (Debug & Release). La opción por default es Desktop, pero pueden aparecer más si se tiene más de una versión de Qt Creator instalada. Se elige una o varias, pero se recomienda usar la correspondiente por default.

Acto seguido aparece la ventana de Información sobre Clase[2] Class Information (Figura B - 2). Aquí se define un nombre para la clase, que se reproduce en el

[2] Es una plantilla fundamental de la programación orientada a objetos. Define un tipo de objeto al especificar sus atributos y operaciones disponibles.

archivo de cabecera (Header) y archivo fuente (Source File). Aquí también se decide si se desea generar una forma GUI o no.

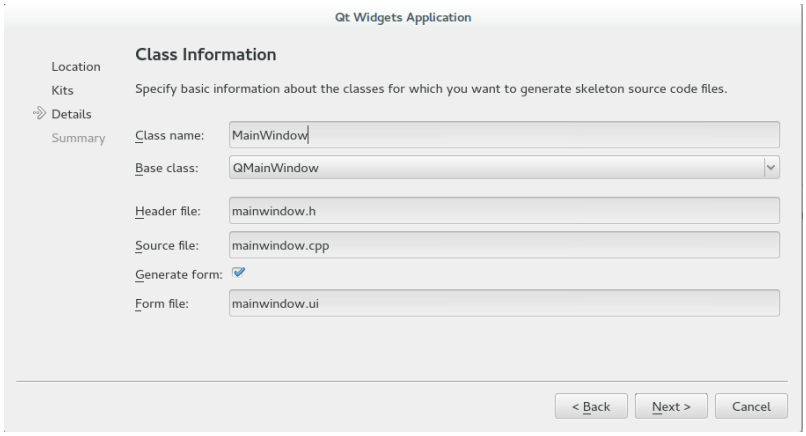


Figura B - 2. Información de las clases del proyecto

Al presionar Next aparece la ventana de diálogo con los ajustes del proyecto, se revisa si son correctos y se presiona Finish.

B.2. DESCRIPCIÓN DEL ENTORNO DE QTCREATOR

Después de crear el proyecto aparece inmediatamente el editor del proyecto (Figura B - 3). En el lado izquierdo se ven todos los archivos generados para el proyecto. A continuación se revisa el uso de cada archivo generado. En el Header se definen variables globales a ser utilizadas, funciones determinadas por los objetos insertados en la GUI (funciones privadas), y funciones que cree el usuario (funciones públicas).

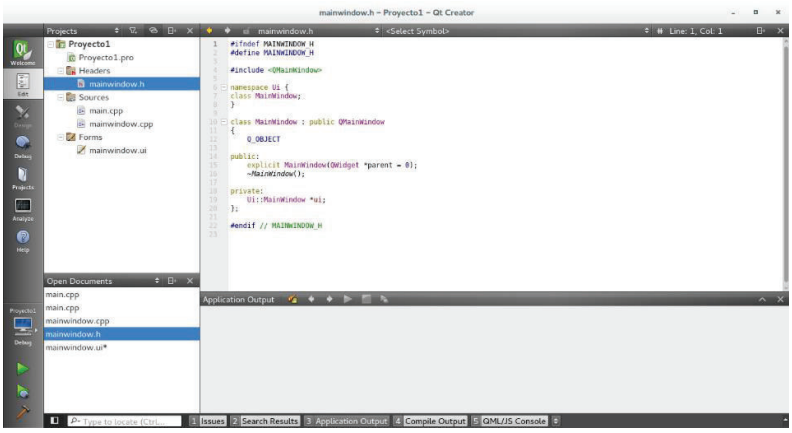


Figura B - 3. Header del proyecto

Todo proyecto que se cree tiene un archivo main.cpp. Este archivo permite que el programa inicie y la ventana principal se muestre. De preferencia no se debe modificar este archivo, pues es considerado un archivo de arranque.

En el archivo mainwindow.cpp (que tendrá el nombre asignado por el usuario) es donde se realiza toda la programación de la interfaz. Aquí se utilizan las funciones públicas, privadas y tanto las variables definidas en el header, como variables locales definidas en cada sección de programa. La programación se realiza en lenguaje C++ con las variaciones propias de Qt.

Si se realiza doble clic en mainwindow.ui, Qt Designer se abre y presenta todas las posibilidades para el diseño de la interfaz. En la mitad aparece la forma vacía. En la columna de la izquierda aparecen los objetos a ser insertados. A la derecha aparece un resumen con los tipos y nombres de los objetos insertados.

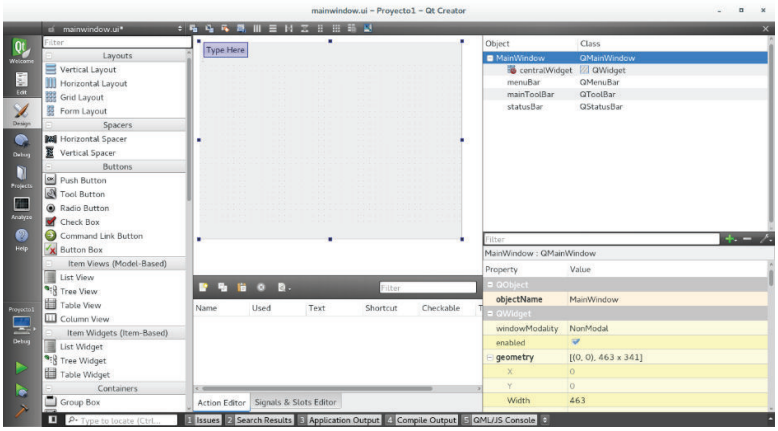


Figura B - 4. Qt Design — GUI

Para el desarrollo de los interfaces en Qt Creator se realiza el diseño de la interfaz en primer lugar y después la programación relativa a la misma. Para asignar eventos a los objetos se da clic en el objeto deseado y se elige Go to slot. A continuación aparecen varias opciones, de la cuales se elige la más apropiada de acuerdo al objeto. Por ejemplo, para un Pushbutton se elige normalmente clicked(), que es un vínculo al contenido de la función que se ejecuta cuando el Pushbutton es presionado (Figura B - 5).

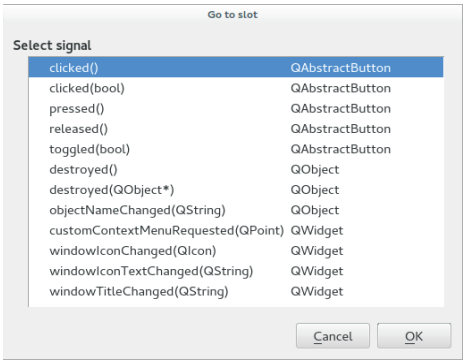


Figura B - 5. Opciones de Go to slot

Para que los elementos de la GUI se vean ordenados y no pierdan su disposición al redimensionar el tamaño de la ventana, se recomienda utilizar las opciones Vertical Layout, Horizontal Layout, Grid Layout y Form Layout.

B.3. USO DE QTDESIGN

A continuación se presentan algunos consejos útiles para el diseño de una interfaz gráfica en QtCreator.

B.3.1. INGRESO DE VALORES NUMÉRICOS

Cuando se requiere que el usuario ingrese valores numéricos en la interfaz se pueden usar objetos tipo QDoubleSpinBox. Una característica útil del objeto es que permite limitar el rango de valores a ser ingresados directamente en la sección de opciones de la pestaña Design de QtCreator, lo que permite evitar el ingreso de valores no soportados por el algoritmo programado (por ejemplo, para este proyecto no se pueden elegir velocidades más allá de la máxima permitida por el robot Pioneer 3DX).

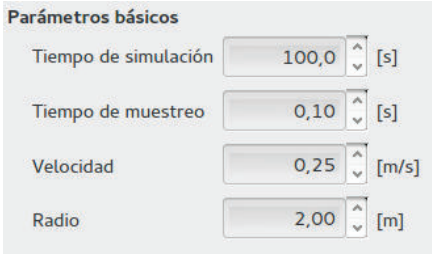


Figura B - 6. Ejemplo del uso del objeto QDoubleSpinBox

B.3.2. USO DE OBJETOS DE AGRUPACIÓN

Los objetos tipo QGroupBox permiten agrupar algunos elementos para crear menúes o listas. Además pueden ser programados para estar habilitados o deshabilitados bajo ciertas condiciones.

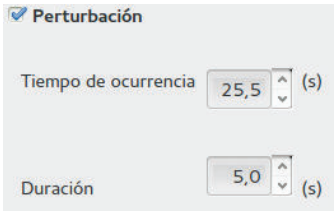


Figura B - 7. Ejemplo de QGroupBox que permite ser habilitado o deshabilitado por el usuario

Por ejemplo, para la sección de perturbación de este proyecto, se necesitó que la opción de perturbación no estuviera siempre activa, sino que el usuario debía definir cuando habilitarla o deshabilitarla. Para hacer esto se selecciona el objeto QGroupBox y en la sección de opciones (Figura B - 8) se marca *checkable* para permitir seleccionar o deseleccionar este QGroupBox.

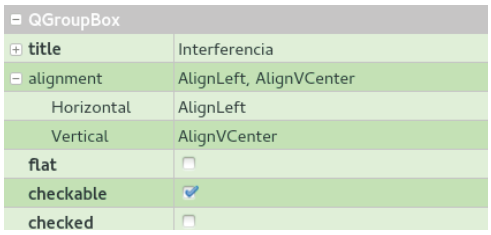


Figura B - 8. Opciones del objeto QGroupBox

B.3.3. CREACIÓN DE MENÚ DE SELECCIÓN CON RADIOBUTTON

Para la creación de este tipo de secciones, en la pestaña de diseño de QtCreator se insertan algunos objetos tipo RadioButton. Luego éstos se ordenan usando la opción Layout Vertically, y para agruparlos se colocan dentro de un objeto tipo QGroupBox. Esta acción permitirá elegir solamente un elemento a la vez de la lista que se ha creado.

Luego en el archivo mainwindow.cpp se programa la acción de cada objeto al ser seleccionado.

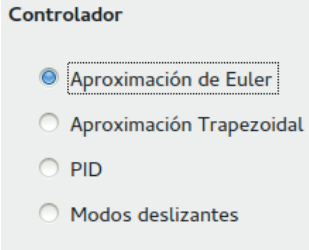


Figura B - 9. Ejemplo de un menú creado con Radiobutton

B.3.4. USO DE TABWIDGET

Este elemento permite insertar un objeto que contiene n pestañas y resulta útil si se quieren mostrar las opciones de varios ítems sin ocupar mucho espacio, pues además resulta muy sencillo habilitar o deshabilitar cada pestaña dependiendo de los requerimiento del proyecto. Para el presente trabajo se ha utilizado este objeto para mostrar los parámetros de calibración de los cuatro controladores utilizados, permitiéndole al usuario variar sus valores.

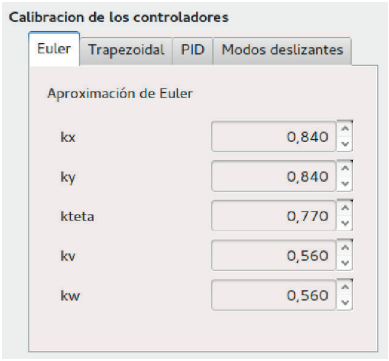


Figura B - 10. Ejemplo de uso del objeto TabWidget

B.3.1. INSERTAR GRÁFICOS 2D DE CALIDAD

A pesar de que QtCreator tiene algunas opciones para presentar gráficos, en caso de necesitar gráficos de calidad (como lo es este proyecto), se puede recurrir a widgets con librerías especializadas, como QCustomPlot[3,4]. Esta librería se enfoca en generar gráficos y tablas 2D de calidad, además de presentar un alto desempeño para aplicaciones de visualización en tiempo real.

[3] Disponible en http://www.qcustomplot.com/index.php/download
[4] Los códigos y software fuente están bajo la licencia GNU GPL a menos que se indique específicamente lo contrario.

Para incluir esta librería en el proyecto actual se hace clic derecho sobre el nombre del proyecto y se elige la opción Add Existing Files, eligiendo a continuación los archivos qcustomplot.h y qcustomplot.cpp.

Para poder utilizar las bondades de esta librería, se inserta un Container tipo Widget en la interfaz gráfica. Después se selecciona el objeto insertado y en la parte derecha de la pantalla en la opción Class se selecciona Promote Widget to… que permite cambiar el objeto tipo QWidget a un objeto tipo QCustomPlot. La pantalla queda similar a lo ilustrado por la Figura B - 11.

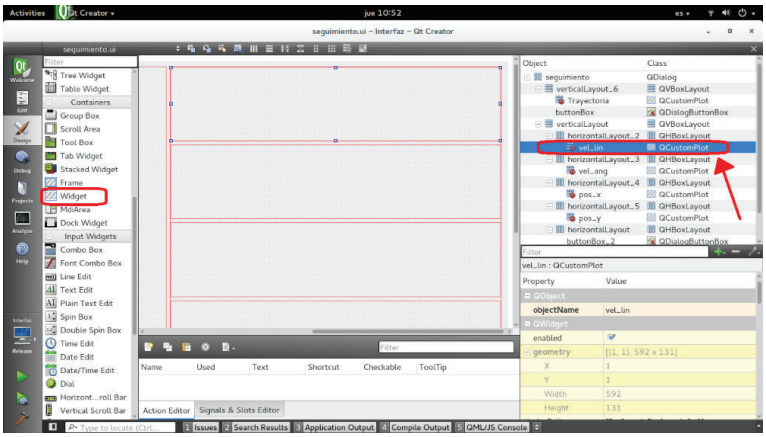


Figura B - 11. Inclusión de un widget tipo QCustomPlot en QtDesign