

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño
Máster de Ingeniería Mecatrónica

CONTROL DE TRAYECTORIA
MEDIANTE LIDAR DE UN
VEHÍCULO OMNIDIRECCIONAL

Director:
Prof. CASANOVA CALVO VICENTE FERMÍN

Trabajo de fin de máster de:
STEFANO ANDREONI

CURSO ACADÉMICO 2020-2021

*Caminante no hay camino,
se hace camino al andar,
Caminante no hay camino,
son tus pasos y nada más.*

*Y al volver la vista atrás,
uno recuerda los amigos,
que nunca va a olvidar.*

*Caminante no hay camino,
caminante no hay camino...*

*Más andar no tendría sentido,
si no fuera al lado de buenos amigos.*

-Joan Xavier Gracia Borreguero-

Índice general

Resumen	1
Introducción	2
1. Cinemática	6
1.1. Cinemática de la rueda <i>Mecanum</i>	6
1.2. Cinemática del vehículo	9
1.3. Odometría	11
2. Implementación real	13
2.1. Sistema de procesamiento	14
2.1.1. ROS	16
2.1.2. Raspberry Pi 4	17
2.1.3. YDLIDAR X4	20
2.1.4. Arduino Due	21
2.1.5. Matlab GUI	22
2.2. Sistema de control	23
2.2.1. Motores Polulu	23
2.2.2. Adafruit Motorshield V2	24
2.2.3. Adafruit IMU BNO055	25
3. Implementación software	26
3.1. Hector SLAM	26
3.1.1. Mapa de ocupación de celdillas	27

3.1.2. Comparación de escaneo	28
3.2. Algoritmo de control	29
3.3. Teoría del filtro de Kalman	31
3.3.1. Implementación	33
4. Sistema de seguimiento de trayectoria	36
4.1. Odometría	38
4.2. Filtro de Kalman	42
4.2.1. Filtro de Kalman y corrección del Lidar	47
Presupuesto	51
Conclusiones	53
Bibliografia	55

Índice de figuras

1.	Diferentes tipos de ruedas omnidireccionales	2
2.	<i>4WD Mecanum Wheel Mobile Arduino Robotics Car</i>	3
3.	Rueda <i>Mecanum</i>	4
4.	Funcionamiento de un sensor <i>LIDAR</i> 2D	5
1.1.	Configuración de las ruedas	7
1.2.	Movimientos principales del vehículo	7
1.3.	Cinemática de la rueda <i>Mecanum</i>	8
1.4.	Configuración del vehículo	9
2.1.	Implementación real del vehículo	13
2.2.	Diagrama del sistema de control	14
2.3.	Diagrama del sistema Broadcast	16
2.4.	Raspberry Pi 4	17
2.5.	Nodos de <i>ROS</i>	18
2.6.	<i>YDLIDAR X4</i>	20
2.7.	Arduino Due	21
2.8.	<i>Matlab GUI</i>	22
2.9.	Sistema de control	23
2.10.	Polulu 37Dx70L	24
2.11.	Adafruit Motorshield V2	25
2.12.	Adafruit BNO055 Absolute Orientation Sensor	25
3.1.	Mapa de ocupación de celdillas del laboratorio	27
3.2.	Diagrama de flujo del algoritmo <i>SLAM</i>	28

3.3.	Curvas de velocidad: cuadrada, trapezoidal y curva a S	30
3.4.	Concepto de funcionamiento del filtro de Kalman	31
3.5.	Campana Gaussiana y desviación estándar	35
4.1.	Trayectoria de referencia y leyenda	37
4.2.	Seguimiento de trayectoria con odometría	38
4.3.	Seguimiento de trayectoria con odometría y perturbación . . .	39
4.4.	Posición y orientación en el tiempo con perturbación	40
4.5.	Comparación entre la posición del <i>LIDAR</i> y el <i>GPS</i>	41
4.6.	Seguimiento de trayectoria con filtro de Kalman	43
4.7.	Posición y orientación en el tiempo	44
4.8.	Retraso en la posición del <i>LIDAR</i>	45
4.9.	Corrección de la medida de posición del <i>LIDAR</i>	46
4.10.	Seguimiento de trayectoria con filtro de Kalman y corrección .	48
4.11.	Posición y orientación en el tiempo	49
4.12.	Trayectoria con curva a ocho y trayectoria circular	50

Resumen

El objetivo de este estudio es desarrollar un sistema de seguimiento de trayectoria para un vehículo omnidireccional de cuatro ruedas *Mecanum*.

El uso de este tipo de rueda permite a los vehículos que lo emplean adquirir una extrema maniobrabilidad y por tanto ser utilizados en espacios reducidos como almacenes automáticos, plantas de fabricación, talleres industriales etc. Para poder cumplir estas tareas de manera autónoma, es necesario conocer la posición absoluta del vehículo, es decir, la posición y orientación respecto a su entorno.

En la memoria se reportan las técnicas de control empleadas para lograr un sistema de posicionamiento y seguimiento de trayectoria.

La memoria consta de cuatro capítulos:

- En el primer capítulo se analizarán las características mecánicas del vehículo utilizado explicando exhaustivamente su cinemática.
- En el segundo capítulo se ilustrarán los pasos de diseño y implementación del sistema de control y procesamiento del vehículo. Se prestará especial atención a la comunicación entre los diferentes sistemas empleados.
- El tercer capítulo trata la implementación software del sistema.
- En el cuarto capítulo se describirán las diferentes técnicas aplicadas para mejorar el seguimiento de trayectoria, analizando ampliamente el sistema de posicionamiento elegido.

Introducción

En los últimos años, el uso de robots en la industria ha incrementado notablemente. La búsqueda de la eficiencia en cada aspecto de la producción ha pasado a ser una necesidad para las empresas y el empleo de robots permite compensarla.

Análogamente, también en el campo de la robótica, la eficiencia es un tema fundamental. Por lo tanto, se está buscando incrementar la maniobrabilidad de los robots, reduciendo el espacio y el tiempo necesario para llevar a cabo sus tareas.

Para ello se han estudiado distintos tipos y configuraciones de ruedas.

Es aquí donde entran en juego las ruedas omnidireccionales, mostradas en la Figura 1, que permiten cualquier tipo de desplazamiento a lo largo del plano de trabajo, evitando realizar giros y desplazamientos innecesarios.



Figura 1: Diferentes tipos de ruedas omnidireccionales

Se definen vehículos omnidireccionales los vehículos que pueden desplazarse en cualquier dirección con cualquier, o ninguna rotación del cuerpo. El término omnidireccional se utiliza para describir la habilidad de un sistema de moverse instantáneamente en cualquier dirección a partir de cualquier configuración inicial [1].

Los vehículos de cuatro ruedas omnidireccionales, son vehículos terrestres diseñados con tres grados de libertad para trabajar en un espacio de dos dimensiones. La combinación de ambos componentes, traslación y rotación, hace que el vehículo pueda seguir cualquier movimiento y trayectoria en un plano. Esta categoría de vehículos destaca sobre las demás por su gran capacidad de moverse en espacios reducidos, ofreciendo una buena base para el movimiento de vehículos guiados automáticamente o *AGV (Automated Guided Vehicle)*.

Las aplicaciones más comunes para los *AGV* son los almacenes automáticos, las plantas de fabricación, los talleres industriales etc.

El vehículo utilizado en este proyecto ha sido proporcionado por el fabricante *Nexus Robot* [2] y consta de cuatro ruedas omnidireccionales de tipo *Mecanum*, Figura 2.



Figura 2: *4WD Mecanum Wheel Mobile Arduino Robotics Car*

La rueda *Mecanum*, llamada también rueda Ilon, es un diseño de rueda omnidireccional inventada por Bengt Erland Ilon en 1972 [3].

Como se aprecia en la Figura 3, el diseño consiste en una rueda convencional que dispone de una serie de rodillos a lo largo de su circunferencia colocados a 45° respecto al eje de la rueda. Asimismo, estos rodillos tienen una forma curvada de forma que vista desde el lateral, la rueda mantiene un perfil circular.



Figura 3: Rueda *Mecanum*

Controlando la velocidad y dirección de giro de las ruedas (cada una está acoplada a un motor) se consiguen los movimientos de traslación y rotación. Es cierto que este tipo de rueda permite una gran maniobrabilidad, sin embargo conlleva una serie de problemáticas como: el contacto discontinuo con el suelo, una alta sensibilidad a las variaciones del terreno y un diseño y fabricación compleja.

Como se explicará más adelante, estos problemas mecánicos afectarán notablemente el cálculo de la posición por odometría, la cual se basa en la estimación de la posición utilizando las informaciones de las ruedas.

Por este motivo es necesario utilizar también un sensor que proporciona la posición absoluta del vehículo respecto a su entorno.

En este proyecto, se ha elegido un sensor *LIDAR* como sistema de posicionamiento absoluto. Un *LIDAR* (del inglés, *Laser Imaging Detection And Ranging*) es un sensor que permite medir distancias desde un emisor láser a un objeto o una superficie.

Hoy en día, este tipo de sensor es ampliamente utilizado en los vehículos guiados automáticamente, porque permite crear un mapa del entorno además detectar y así evitar los obstáculos.

Diferentes tipos de *LIDAR* se utilizan también en otros ámbitos como en: astronomía, agricultura, fotogrametría etc.

Las diferencias principales entre los distintos modelos son la capacidad de escanear el entorno en dos o tres dimensiones, la precisión y el alcance. Además, la mayoría de los sensores puede girar a 360° creando una nube de puntos del entorno, obteniendo así un mapa tras una elaboración software.

En la Figura 4 se puede apreciar el funcionamiento de un *LIDAR* de dos dimensiones, como el utilizado en este proyecto.

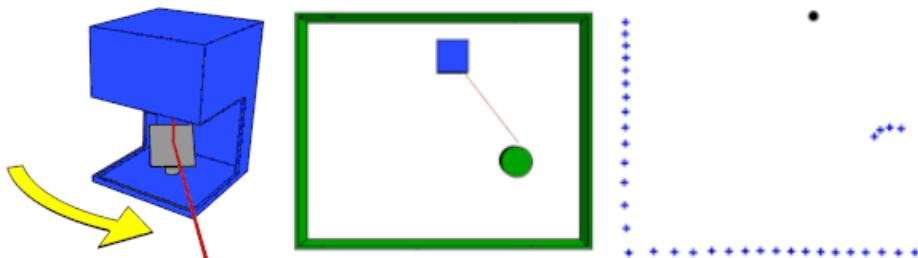


Figura 4: Funcionamiento de un sensor *LIDAR* 2D

Capítulo 1

Cinemática

La cinemática es la rama de la mecánica que describe el movimiento de un objeto sólido sin considerar las causas (fuerzas externas) que lo originan. En este caso concreto, la cinemática permite determinar la trayectoria del vehículo en función del tiempo, utilizando velocidades y aceleraciones de las ruedas.

En este proyecto, se ha utilizado un vehículo en configuración a ruedas cruzadas, visible en la Figura 1.1. Se trata de una configuración donde la rueda delantera izquierda y la trasera derecha tienen los rodillos en sentido derecho, mientras que la rueda delantera derecha y trasera izquierda tienen los rodillos en sentido izquierdo.

Controlando individualmente las ruedas del vehículo, mediante la combinación lineal de las fuerzas resultantes y de las velocidades de cada una, se obtiene cualquier movimiento de desplazamiento y rotación.

1.1. Cinemática de la rueda *Mecanum*

Para poder entender mejor la cinemática de todo el vehículo, es necesario explicar en primer lugar, la cinemática de la rueda *Mecanum*.

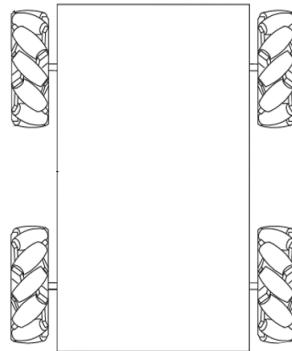


Figura 1.1: Configuración de las ruedas

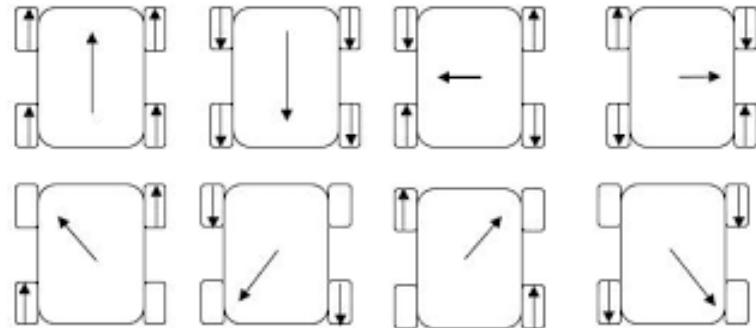


Figura 1.2: Movimientos principales del vehículo

El estudio cinemático se extrae de la literatura: “*Kinematic Model of a Four Mecanum Wheeled Mobile Robot*” [4].

Según la Figura 1.3 se define:

- v_r velocidad tangencial del rodillo.
- v_x velocidad lineal de la rueda a lo largo del eje x .
- v_y velocidad lineal de la rueda a lo largo del eje y .
- ω velocidad angular de la rueda.
- r radio de la rueda

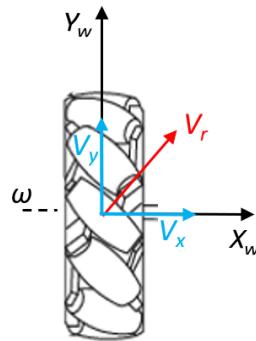


Figura 1.3: Cinemática de la rueda Mecanum

A partir de la velocidad tangencial, que es perpendicular a los rodillos, se obtienen las velocidades compuestas respecto al sistema de referencia de la rueda como sigue (se recuerda que los rodillos están dispuestos a 45° respecto al eje de la rueda):

$$v_x = v_r \cdot \cos(45^{\circ}) = v_r \cdot \frac{\sqrt{2}}{2} \quad (1.1)$$

$$v_y = v_r \cdot \sin(45^{\circ}) = v_r \cdot \frac{\sqrt{2}}{2} \quad (1.2)$$

La velocidad lineal total de cada rueda, se calcula como la suma de la velocidad lineal debida al giro de la rueda más la aportada por los rodillos.

$$v_{x,tot} = v_r \cdot \frac{\sqrt{2}}{2} \quad (1.3)$$

$$v_{y,tot} = \omega \cdot r + v_r \cdot \frac{\sqrt{2}}{2} \quad (1.4)$$

De la comparación de la 1.3 con la 1.4, se nota que en la dirección del eje x la velocidad total corresponde solamente a la proporcionada por el rodillo, mientras que en la dirección del eje y dicha velocidad se suma a la velocidad lineal de la rueda.

1.2. Cinemática del vehículo

En la Figura 1.4 se muestra la configuración del vehículo utilizado.

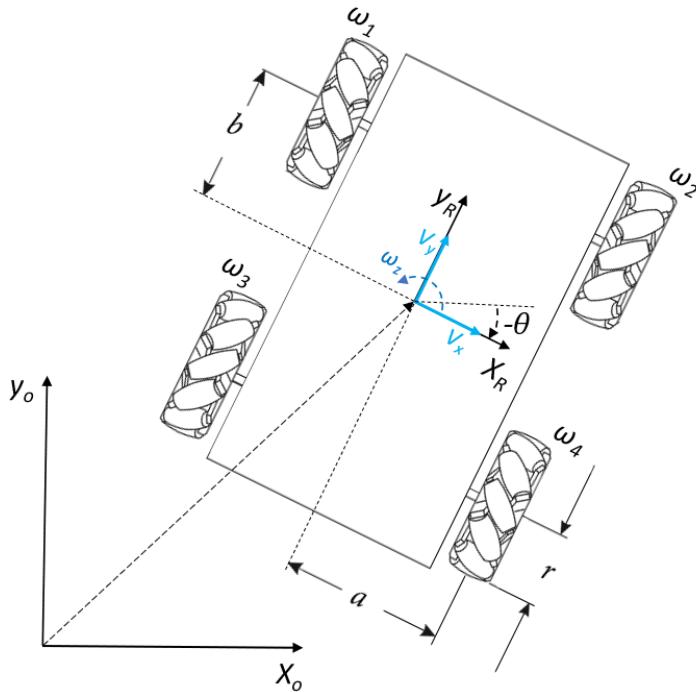


Figura 1.4: Configuración del vehículo

Los parámetros del sistema se definen de la siguiente forma:

- x, y, θ con x, y posición del vehículo y θ su orientación (angulo entre X_0 y X_R).
- X_0, O_0, Y_0 sistema de referencia global.
- X_R, O_R, Y_R sistema de referencia del vehículo.
- a mitad de la distancia entre los centros de las ruedas a lo largo del eje X_R .
- b mitad de la distancia entre los ejes de las ruedas a lo largo del eje Y_R .

- r radio de las ruedas.
- $\omega_1, \omega_2, \omega_3, \omega_4$ [rad/s] velocidades angulares de las ruedas.
- v_x, v_y [m/s] velocidades lineales de traslación y ω_z [rad/s] velocidad angular de rotación del vehículo.

Para desarrollar un sistema de seguimiento de trayectoria es fundamental comprender como evoluciona la aceleración, la velocidad y la posición del vehículo en el tiempo.

Este problema se resuelve a través de la cinemática directa, la cual permite calcular la velocidad de traslación y rotación del vehículo a partir de las velocidades angulares de las ruedas.

Como se demuestra en el artículo [4], las ecuaciones de la cinemática directa son las siguientes:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \frac{r}{4} \begin{bmatrix} -1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ \frac{1}{(a+b)} & -\frac{1}{(a+b)} & \frac{1}{(a+b)} & -\frac{1}{(a+b)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \quad (1.5)$$

De la matriz anterior se extraen las siguientes variables de sistema.

Las velocidades lineales de traslación del vehículo v_x, v_y :

$$v_x(t) = \frac{r}{4} \cdot (-\omega_1 + \omega_2 + \omega_3 - \omega_4) \quad (1.6)$$

$$v_y(t) = \frac{r}{4} \cdot (\omega_1 + \omega_2 + \omega_3 + \omega_4) \quad (1.7)$$

La velocidad angular de rotación del vehículo ω_z :

$$\omega_z(t) = \frac{r}{4(a+b)} \cdot (-\omega_1 + \omega_2 + \omega_3 - \omega_4) \quad (1.8)$$

Análogamente a quanto se ha dicho anteriormente sobre la cinemática directa, es válido también el razonamiento contrario. Es decir que, en un sistema de seguimiento de trayectoria, además conocer la velocidad del vehículo a partir de las velocidades de las ruedas, es necesario también calcular las velocidades de las ruedas para que el vehículo se mueva a la velocidad deseada. Las ecuaciones que describen la cinemática inversa se reportan a continuación.

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} -1 & 1 & a+b \\ 1 & 1 & -(a+b) \\ 1 & 1 & a+b \\ -1 & 1 & -(a+b) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \quad (1.9)$$

1.3. Odometría

La odometría es el estudio de la estimación de la posición de vehículos con ruedas durante la navegación. Para realizar esta estimación se usan las informaciones de las rotaciones de las ruedas para estimar cambios en la posición del vehículo a lo largo del tiempo. La odometría proporciona una buena precisión a corto plazo, sin embargo se basa en la integración de información incremental del movimiento a lo largo del tiempo, lo cual conlleva una inevitable acumulación de errores [5].

A partir de las ecuaciones de la cinemática directa 1.6, 1.7 se estima la posición en x, y del vehículo a cada periodo de tiempo discreto ΔT como sigue:

$$\begin{aligned} x_{t+\Delta T} &= x_t + v_{x,t} \cdot \Delta T \\ &= x_t + \frac{r}{4} \cdot (-\omega_1 + \omega_2 + \omega_3 - \omega_4) \cdot \Delta T \end{aligned} \quad (1.10)$$

$$\begin{aligned} y_{t+\Delta T} &= y_t + v_{y,t} \cdot \Delta T \\ &= y_t + \frac{r}{4} \cdot (\omega_1 + \omega_2 + \omega_3 + \omega_4) \cdot \Delta T \end{aligned} \quad (1.11)$$

Análogamente, la orientación θ del vehículo se calcula a partir de la ecuación 1.8, como:

$$\begin{aligned}\theta_{t+\Delta T} &= \theta_t + \omega_{z,t} \cdot \Delta T \\ &= \theta_t + \frac{r}{4(a+b)} \cdot (-\omega_1 + \omega_2 + \omega_3 - \omega_4) \cdot \Delta T\end{aligned}\tag{1.12}$$

Como se ha explicado anteriormente, la odometría es solamente una estimación de la posición, ya que con este método no se pueden detectar los errores causados por los siguientes factores:

- Diámetros de las ruedas diferentes.
- Mal alineamiento de las ruedas.
- Patinaje de las ruedas debido a:
 - Suelos resbaladizos.
 - Derrapes (debidos a una rotación excesivamente rápida).
 - Fuerzas externas (interacción con cuerpos externos).
 - Diferentes, o ningún, puntos de contacto con el suelo.

Por este motivo, no se puede implementar un sistema de posicionamiento basado en la sola odometría. Para conseguir un correcto sistema de posicionamiento, es necesario corregir las informaciones de la odometría con las medidas de un sensor de posición absoluta.

Capítulo 2

Implementación real

En este capítulo se describirán los componentes electrónicos y mecánicos empleados para la realización del vehículo.

Como se había anticipado en la introducción, para poder enfocar el desarrollo del proyecto en la parte de control y navegación, se ha utilizado un vehículo ya ensamblado. Aunque el vehículo fuese funcional, se ha optado para sustituir la electrónica de control propietario con los componentes que se describen a continuación, dejando intacto solo el chasis y las ruedas.

Para una mejor comprensión del sistema completo, se ha decidido explicarlo por separado, dividiéndolo en dos subsistemas: sistema de procesamiento y sistema de control.



Figura 2.1: Implementación real del vehículo

2.1. Sistema de procesamiento

Con sistema de procesamiento se entiende el conjunto de dispositivos que se encargan de la elaboración de datos a alto nivel y del mando, a través de comandos codificados, del sistema de control.

En la figura 2.2 se pueden apreciar los diferentes dispositivos y las conexiones que componen el sistema de procesamiento del robot.

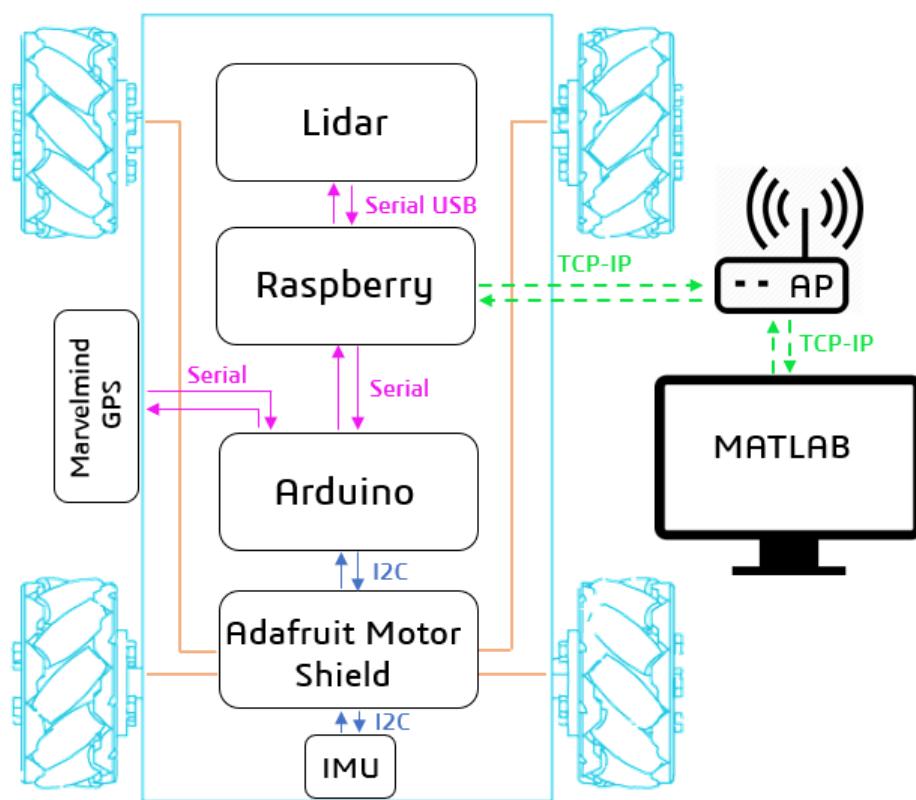


Figura 2.2: Diagrama del sistema de control

A continuación se explicará brevemente la función de cada modulo, aunque los más importantes serán analizados detalladamente en los siguientes apartados.

- **Raspberry:** Se utiliza este ordenador, de tamaño reducido, como unidad de elaboración principal del sistema de procesamiento. Este dispositivo procesa directamente los datos proporcionados por el *LIDAR* y gestiona la comunicación con el ordenador en remoto. En la *Raspberry* se ejecuta localmente *ROS (Robotic Operative System)*, un software que facilita las comunicaciones entre diferentes sistemas y el procesamiento de datos del *LIDAR*. Además, manda el sistema de control a través de comandos codificados enviados por comunicación serial.
- **Arduino:** Este microcontrolador se encarga de controlar directamente las velocidades de las ruedas, utilizando encoders acoplados a los ejes de los motores y modulando la tensión de funcionamiento de estos a través del driver de potencia *Adafruit Motorshield*. Además, ejecuta los algoritmos de seguimiento de trayectoria y de mejora del posicionamiento. Es la unidad de procesamiento del sistema de control.
- **Marvelmind Indoor GPS:** Este sistema replica un sistema de posicionamiento *GPS* para interiores y se utiliza en este proyecto como referencia de posición real del robot (*Ground Truth*). El sistema consta de cuatro balizas fijas y una móvil que comunican entre si con tecnología de ultrasonidos. Disponiendo las balizas en posiciones conocidas permite crear un mapa y así localizar la baliza móvil conectada al vehículo, con una precisión de $\pm 2\text{ cm}$.
Se destaca que este dispositivo se ha utilizado solo como referencia y no como parte integrante del sistema de posicionamiento del vehículo, por lo tanto no será explicado más detalladamente.
- **Matlab:** Para poder controlar y visualizar los datos del vehículo en remoto, se ha desarrollado una interfaz gráfica con el software *Matlab*. Esta aplicación gráfica se ejecuta en un ordenador conectado a la misma red en la cual está conectada la *Raspberry* mediante conexión *WiFi*.

2.1.1. ROS

Para poder entender mejor los siguientes apartados, se describirá brevemente las bases del funcionamiento de *ROS*.

ROS (*Robot Operating System*) es un framework para escribir software para robots, que consta de un conjunto de librerías y algoritmos que permiten desarrollar sistemas complejos y robustos en diferentes plataformas [6].

ROS emplea una arquitectura formada por diferentes nodos (*nodes*) interconectado entre sí, a formar una única red. Un nodo es un algoritmo que procesa determinados datos en entrada y devuelve el resultado de la elaboración como salida. Cada nodo puede intercambiar informaciones con cualquier otro nodo de la red utilizando mensajes predefinidos.

El sistema de comunicación utilizado es de tipo *Broadcast*, donde un nodo emisor envía información a una multitud de nodos receptores de manera simultánea, sin necesidad de reproducir la misma transmisión nodo por nodo.

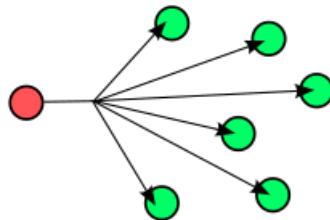


Figura 2.3: Diagrama del sistema Broadcast

Cada nodo proporciona sus datos de salida agrupados en diferentes temas (*topics*). Además, cualquier nodo tiene la posibilidad de leer los datos de un tema si se suscribe al *topic* como *Subscriber*, o también de escribir nuevos datos suscribiéndose como *Publisher*.

2.1.2. Raspberry Pi 4

Raspberry Pi 4 es un ordenador de dimensiones reducidas basado en el chip *Broadcom BCM2711*, Figura 2.4. A diferencia de otros ordenadores, esta tarjeta electrónica incluye 40 pines digitales de entrada-salida con los cuales se permite interactuar con otros dispositivos. El sistema operativo instalado es *Raspbian*, una versión basada en *Linux* y optimizada para *Raspberry*. La elección de este componente se debe principalmente a sus dimensiones reducidas, a su potencia de cálculo y al coste razonable.



Figura 2.4: Raspberry Pi 4

La introducción de este componente en el sistema ha sido necesaria debido a que *Arduino Due* no tiene suficiente potencia de cálculo para procesar los datos provenientes del *LIDAR*. La motivación por la cual no se ha podido utilizar solo *Raspberry Pi 4* como único sistema de control y procesamiento es porque el sistema operativo utilizado no es de tiempo real. Tareas como el control de la velocidad de los motores necesitan un procesamiento de los datos en tiempo real.

En resumen, se utiliza *Raspberry Pi 4* como sistema de procesamiento para el posicionamiento y navegación, mientras se utiliza *Arduino Due* como sistema de control directo del movimiento del vehículo.

Tabla de especificaciones:

Chip	Broadcom BCM2711
CPU	ARM Cortex-A72 de cuatro núcleos a 1.5 GHz
GPU	VideoCore VI
Memoria RAM	2/4/8GB LPDDR4
Conectividad	Wi-Fi, Bluetooth, Ethernet
Puertos	2 x USB 3.0, 2 x USB 2.0
Pines E/S	40 pines GPIO
Alimentación	5V/3A vía USB-C

En la *Raspberry* se ejecuta *ROS* en modalidad maestro para la elaboración directa de los datos, mientras en el ordenador remoto se ejecuta *ROS* en modalidad esclavo para la visualización.

La estructura de nodos implementada en *ROS* se puede apreciar en la Figura 2.5.

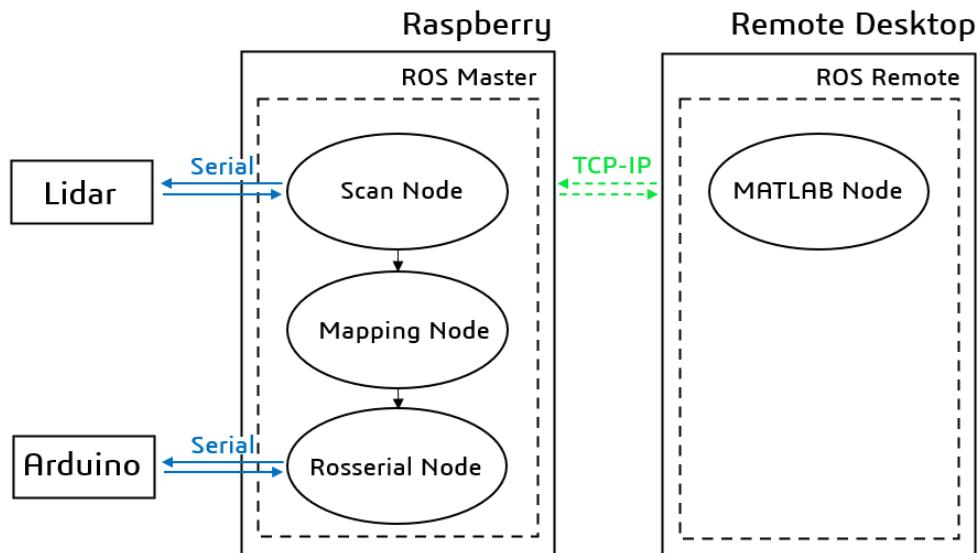


Figura 2.5: Nodos de *ROS*

A continuación se explica brevemente la función de cada nodo:

- **Scan Node:** Este nodo controla la velocidad de rotación del *LIDAR* y lee los datos proporcionados por el sensor a través del puerto serie.
- **Mapping Node:** Este nodo, a partir de los datos de salida del *Scan Node*, o sea, de los datos del *LIDAR*, crea un mapa en 2D y devuelve la posición y orientación del vehículo.
- **Rosserial Node:** Este nodo se encarga de intercambiar informaciones entre *Raspberry Pi 4* y *Arduino Due* a través del puerto serie.
- **Matlab Node:** Este nodo gestiona el intercambio de informaciones, a través de una conexión *Wi-Fi*, entre el software *Matlab* (ejecutado en un ordenador remoto) y la *Raspberry Pi 4*. En este modo, se pueden visualizar y/o modificar los datos de los nodos en ejecución, directamente de un ordenador en remoto.

2.1.3. YDLIDAR X4

Un *LIDAR* (del inglés *Laser Imaging Detection and Ranging*) es un dispositivo que permite determinar la distancia desde un emisor láser a un objeto o superficie utilizando un haz láser pulsado. La distancia al objeto se determina midiendo el tiempo de retraso entre la emisión del pulso y su detección a través de la señal reflejada. En particular el *LIDAR* utilizado, está conectado a un motor de tal modo que, girando, genere una nube de puntos a 360 grados en dos dimensiones.



Figura 2.6: *YDLIDAR X4*

El dispositivo utilizado en este proyecto es el *YDLIDAR X4*, Figura 2.6, cuyas especificaciones se reportan en la tabla a continuación:

Frecuencia de escaneo	5-10 Hz
Alcance	0.12-10 m
Angulo de escaneo	0-360°
Resolución	< 1 % de la distancia (max 0.5 mm)
Tensión de alimentación	4.8-5.2 V
Corriente nominal	330-380 mA
Longitud de ola laser	775-795 nm

En el apartado 3.1, se explicará como se procesan los datos proporcionados por este sensor para obtener la posición y la orientación del vehículo respecto a su entorno.

2.1.4. Arduino Due

Arduino Due, Figura 2.7, es una tarjeta basada en el microcontrolador *Atmel SAM3X8E ARM Cortex-M3* que permite interactuar con otros dispositivos electrónicos a través de sus pines de entrada y salida.

La elección de este microcontrolador se debe a su bajo coste, al lenguaje de programación sencillo basado en *C* y *C++*, y sobretodo a su grande disponibilidad de recursos en internet.

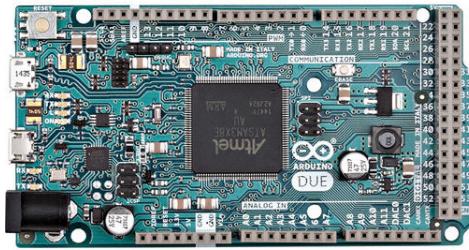


Figura 2.7: Arduino Due

A continuación, se proporciona una tabla de las características más importantes de *Arduino Due*:

Microcontrolador	AT91SAM3X8E a 84 MHz
Tensión de funcionamiento	3.3 V
Voltaje de entrada (recomendado)	7-12 V
E/S digitales	54 (de los cuales 12 PWM)
Memoria flash	512 KB
Memoria SRAM	96 KB
Dimensiones	101.52 x 53.3 mm

Como se menciona anteriormente, este dispositivo se encarga del control directo del movimiento del vehículo. En aras de claridad, se explican los algoritmos de control implementados en la sección 3.2.

2.1.5. Matlab GUI

Para poder visualizar mejor los datos de la telemetría del vehículo, se ha desarrollado una *GUI (Graphic User Interface)* utilizando el software *Matlab*.

La aplicación está dividida en diferentes paneles que corresponden a diferentes funcionalidades, Figura 2.8:

- **Conexión:** Este conjunto de botones permite conectarse a la Raspberry a la dirección *IP* especificada.
- **Modo manual:** A través de los botones presentes en este panel se envían diferentes comandos manuales. Se puede modificar la velocidad, controlar el movimiento del vehículo etc..
- **Modo automático:** Una vez seleccionada esta modalidad, para mover el vehículo es necesario enviar una posición (x,y), bien introduciéndola manualmente o bien enviando un punto de una trayectoria.
De la misma aplicación, es posible parametrizar una trayectoria multipunto en dos dimensiones. Una vez generada una nueva trayectoria, tras pulsar el comando de inicio trayectoria, el programa se encargará de enviar la posición del punto siguiente una vez el vehículo haya alcanzado la posición anterior.
- **Telémetria:** En esta serie de gráficos se reportan, casi en tiempo real, los datos del vehículo entre los cuales velocidad, posición y orientación.

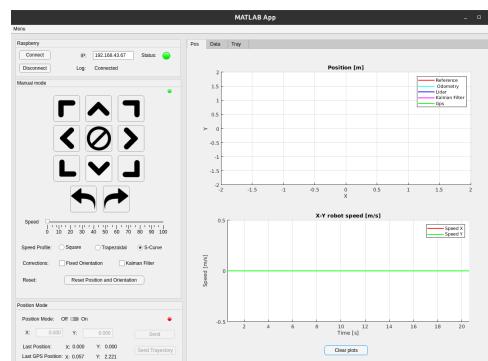


Figura 2.8: *Matlab GUI*

2.2. Sistema de control

Con sistema de control se define el sistema electro-mecánico que controla directamente el movimiento del vehículo.

El sistema se basa en el uso del microcontrolador *Arduino Due*, que se encarga principalmente de controlar las velocidades de las ruedas.

En la Figura 2.9, se puede apreciar una visión general del sistema de control.

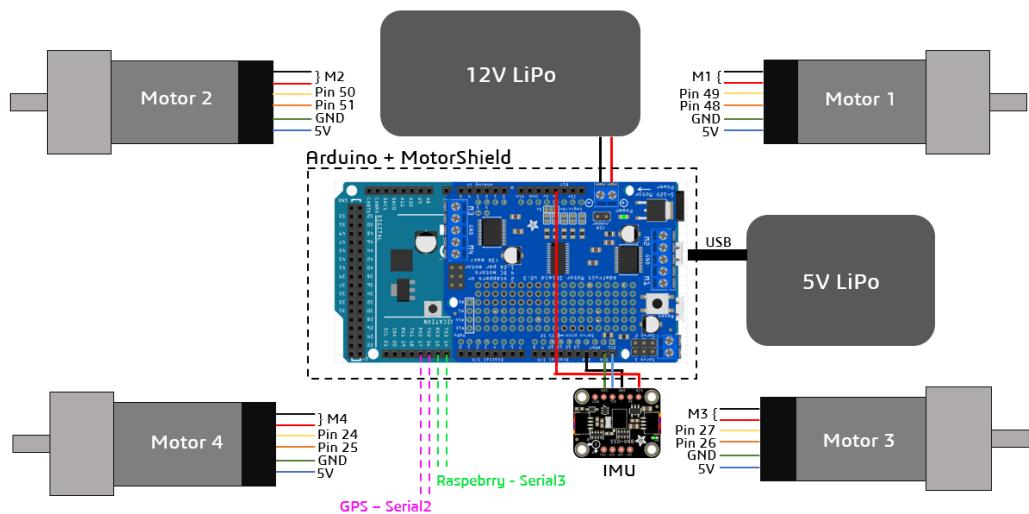


Figura 2.9: Sistema de control

2.2.1. Motores Polulu

Los motores utilizados, *Polulu 37Dx70L*, se pueden apreciar en la Figura 2.10. Este modelo consta de un motor de 12V de corriente continua acoplado con un reductor 70:1. Además, se integra un encoder en cuadratura con una resolución de 64 *CPR* (*Counts Per Revolution*) que corresponden a 4480 *CPR* al eje de salida del reductor.



Figura 2.10: Pololu 37Dx70L

En la tabla siguiente se juntan las características principales.

Tipo de motor	Motor de escobillas
Tensión nominal de funcionamiento	12 V
Tipo de corriente	Corriente Continua (DC)
Potencia máxima	10 W
Velocidad (sin carga) @ 12 V	150 rpm
Corriente consumida (sin carga) @ 12 V	0.2 A
Par @ 12 V	27 kg·cm ³
Ratio del reductor	70:1
Resolución encoder	64 CPR
Peso	205 g

2.2.2. Adafruit Motorshield V2

La *Adafruit Motorshield V2* es una tarjeta electrónica que permite controlar hasta 4 motores en corriente continua y 2 motores paso-paso. Esta tipo de tarjeta, llamada generalmente driver de potencia, es necesaria ya que la corriente proporcionada de *Arduino Due* no es suficiente para alimentar los motores.

La *Adafruit Motorshield V2* comunica con *Arduino Due* a través del bus *I²C* y controla la velocidad de los motores con una señal *PWM* (*Pulse With Modulation*).

Por motivos de ruido eléctrico causado por los motores, se han utilizado dos

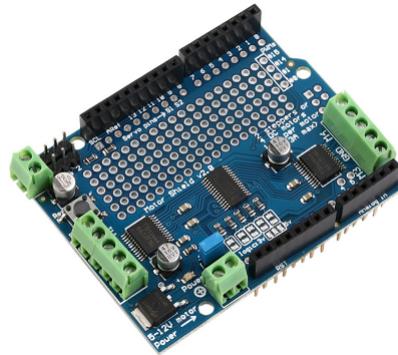


Figura 2.11: Adafruit Motorshield V2

fuentes de alimentación, una para la electrónica de control y la otra para la electrónica de potencia.

2.2.3. Adafruit IMU BNO055

Una unidad de medición inercial o *IMU* (del inglés *Inertial Measurement Unit*), es un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales de un aparato, usando una combinación de acelerómetros, giróscopos y magnetómetros.

Se ha utilizado el sensor *Adafruit IMU BNO055*, Figura 2.12, con el objetivo de medir la orientación del vehículo.



Figura 2.12: Adafruit BNO055 Absolute Orientation Sensor

Esta *IMU* incluye un microprocesador que fusiona los datos provenientes de acelerómetros, giróscopos y magnetómetros devolviendo la orientación absoluta del sensor en el espacio. Los datos proporcionados de este dispositivo se comunican utilizando el bus I^2C .

Capítulo 3

Implementación software

En este capítulo se describirán brevemente los algoritmos implementados para el sistema de control y para el sistema de posicionamiento y navegación del vehículo.

En particular se describirá como se procesan los datos proporcionados por el *LIDAR* para obtener la posición y la orientación del vehículo.

3.1. Hector SLAM

La localización y mapeo simultáneos, o *SLAM* (*Simultaneous Localization And Mapping*), es una técnica usada por robots y vehículos autónomos para construir un mapa de un entorno desconocido en el que se encuentra, a la vez que estima su trayectoria al desplazarse dentro de este entorno [7].

Dicho de otra manera, el *SLAM* busca resolver los problemas que plantea el colocar un robot móvil en un entorno y posición desconocidos, y que él mismo sea capaz de construir incrementalmente un mapa consistente del entorno al tiempo que utiliza dicho mapa para determinar su propia localización.

Existen diferentes algoritmos de *SLAM* dependiendo del sensor utilizado, en este caso concreto se ha utilizado un algoritmo llamado *Hector SLAM*, el cual está optimizado para el uso con *LIDAR* 2D.

Este algoritmo implementado en *ROS*, consta de dos partes, en la primera parte se crea un mapa y en la segunda se compara el escaneo del *LIDAR* con el mapa. La comparación permite determinar como el sensor y por tanto el vehículo, se ha movido respecto al mapa.

3.1.1. Mapa de ocupación de celdillas

Hector SLAM implementa el método del mapa de ocupación de celdillas (*Occupancy Grid*), para construir el mapa del entorno . Este método, se basa en discretizar el espacio dividiéndolo en unidades de tamaño predefinido (celdillas). Estas celdillas se clasifican como ocupadas o vacías con un determinado nivel de confianza o probabilidad.

Entre sus ventajas cabe destacar las siguientes:

- El algoritmo es robusto y su implementación sencilla.
- No hace suposiciones acerca de la naturaleza geométrica de los elementos presentes en el entorno a favor de la velocidad computacional.
- Distingue entre zonas ocupadas y vacías, consiguiendo una partición y descripción completa del espacio explorado.
- Permite descripciones arbitrariamente densas o precisas del entorno, simplemente aumentando la resolución de la rejilla al coste del rendimiento computacional del algoritmo.



Figura 3.1: Mapa de ocupación de celdillas del laboratorio

3.1.2. Comparación de escaneo

En la fase de comparación de escaneo (*Scan Matching*), el algoritmo busca la trasformación, expresada en forma de matrices de rotación y traslación, entre un nuevo escaneo del *LIDAR* y el mapa.

La trasformación se obtiene utilizando el algoritmo *ICP* (*Iterative closest point*), que minimiza la diferencia entre dos nubes de puntos. Como este problema matemático no es objeto del estudio, se deja una mejor profundización en la literatura [8].

El funcionamiento del algoritmo de *SLAM* en su totalidad se puede apreciar en la Figura 3.2.

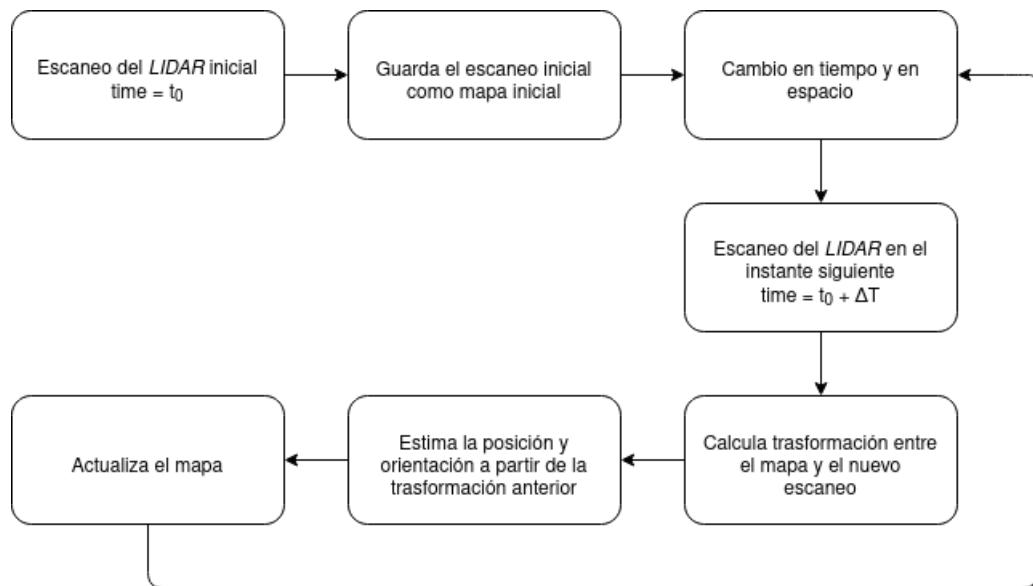


Figura 3.2: Diagrama de flujo del algoritmo *SLAM*

3.2. Algoritmo de control

En este apartado se explica conceptualmente como se ha implementado el algoritmo de control que gestiona el movimiento del vehículo.

El programa se ha desarrollado en *C++* utilizando los paradigmas de la programación orientada a objetos o *OOP (Object-Oriented Programming)*.

El algoritmo de control se presenta en forma de pseudocódigo a continuación.

Algorithm 1: Arduino loop

```

Inicialización;
 $T_{ROS} \leftarrow 0,06;$  // Intercambiar datos vía ROS cada 60 ms
 $T_{control} \leftarrow 0,02;$  // Ejecutar el bucle de control en 20 ms
while true do
     $t \leftarrow tiempo;$ 
    ejecutar comando - ROS;
    actualizar velocidad del vehículo - control PID;
    actualizar posición y orientación del vehículo;
    actualizar filtro de Kalman;
    if  $t \geq T_{ROS}$  then
        leer comando por puerto serie - ROS;
        leer datos GPS;
        leer datos de telémetria del vehículo;
        enviar datos por puerto serie - ROS;
    end
    while  $t \leq T_{control}$  do
        | espera;
    end
end

```

Se observa que el bucle de control se ejecuta cada 20 ms permitiendo el control del vehículo en tiempo real. En cambio, el intercambio de datos con *ROS* se efectúa a una frecuencia menor, o sea cada 60 ms, estando esto limitado también por la velocidad de transmisión del puerto serie.

El vehículo se ha programado para funcionar en dos modalidades:

- **Modo manual:** A través de la interfaz gráfica desarrollada en *Matlab*, párrafo 2.1.5, se pueden enviar diferentes comandos manuales entre los cuales: modificar la velocidad, controlar el desplazamiento y la rotación del vehículo y seleccionar el modo de funcionamiento.

Los comandos se envían utilizando *ROS*, desde la aplicación a la *Raspberry* vía *WiFi* y de aquí a *Arduino* a través de una conexión serial.

- **Modo automático:** En esta modalidad el vehículo se mueve automáticamente nada más recibir una posición (x,y) . La posición se envía de la aplicación mencionada anteriormente, bien introduciéndola manualmente o bien enviando un punto de una trayectoria.

Una vez recibida una nueva posición el programa calcula una trayectoria punto-punto para alcanzarla. Se nota que para aprovecharse de la omnidireccionalidad del vehículo los movimientos se efectúan con orientación fija. Además, para suavizar el movimiento y así mejorar el posicionamiento del vehículo, se han implementado y experimentado diferentes tipos de curvas de velocidad. En la Figura 3.3 se muestran las diferentes curvas de velocidad en un movimiento de traslación de 2 m ad una velocidad máxima de 0,5 m/s.

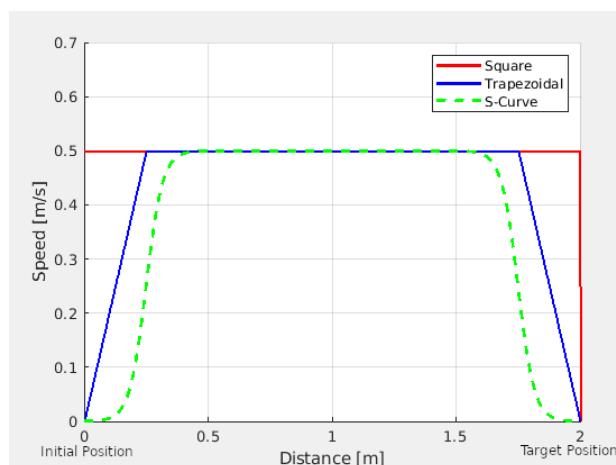


Figura 3.3: Curvas de velocidad: cuadrada, trapezoidal y curva a S

3.3. Teoría del filtro de Kalman

En teoría de control, el filtro de Kalman, se define como un algoritmo que estima el valor más probable de las variables de un sistema. Este valor se estima a partir del valor predicho por el modelo dinámico del sistema y del valor medido por los sensores.

El filtro asume que el modelo del sistema y las medidas de los sensores contienen incertidumbres de distribución Gaussiana y que el valor estimado obtenido del conjunto de informaciones es mejor que el medido directamente. El filtro de Kalman es ampliamente utilizado por su implementación sencilla y su coste computacional reducido. La teoría que se explica a continuación es una versión simplificada de la más completa descrita en la literatura [9].

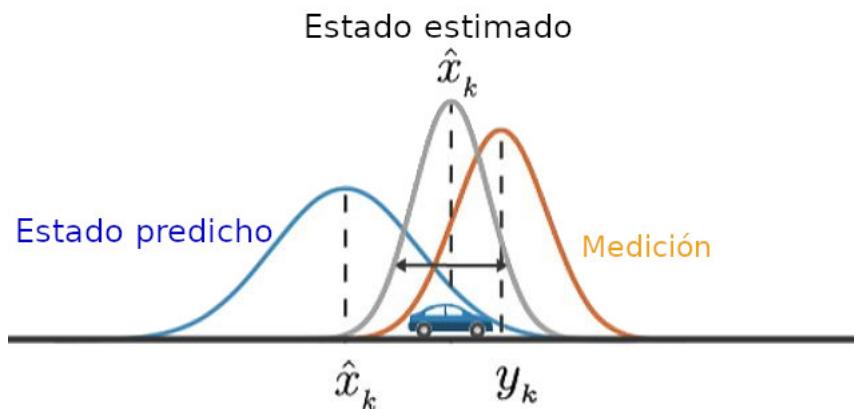


Figura 3.4: Concepto de funcionamiento del filtro de Kalman

El modelo dinámico de proceso define la evolución del estado del sistema del instante de tiempo k al instante $k + 1$, siendo su ecuación:

$$x_{k+1} = Fx_k + Bu_k + w_k \quad (3.1)$$

Las informaciones del modelo de proceso se juntan con las del modelo de medición, que describe la relación entre el estado y su medida al instante k :

$$z_k = Hx_k + v_k \quad (3.2)$$

En las ecuaciones 3.1 y 3.2 se define:

- x vector de las variables de estado o simplemente estado.
- u vector de las variables de control, o entrada del sistema.
- F matriz de evolución de estado.
- B matriz de control.
- w vector del ruido de proceso, se asume sea Gaussiano de media nula y covarianza Q .
- z vector de medidas.
- H matriz de medidas.
- v vector del ruido de medición, se asume sea Gaussiano de media nula y covarianza R .

El objetivo del filtro de Kalman es proporcionar el estado del sistema estimado x al tiempo $k + 1$, a partir del estado inicial x_0 y utilizando las informaciones del modelo de proceso y de las mediciones.

El algoritmo se divide en dos etapas:

Predicción:

$$\text{Estado estimado predicho} \quad \hat{x}_{k+1} = F\hat{x}_k + Bu_k$$

$$\text{Error de covarianza predicho} \quad P_{k+1} = FP_kF^T + Q$$

Corrección:

$$\text{Ganancia de Kalman} \quad K_k = P_{k-1}H^T(R + HP_{k-1}H^T)^{-1}$$

$$\text{Estado estimado corregido} \quad \hat{x}_k = \hat{x}_{k-1} + K_k(z_k - H\hat{x}_{k-1})$$

$$\text{Error de covarianza corregido} \quad P_k = (I - K_kH)P_{k-1}$$

En las ecuaciones anteriores, con el operador $\hat{\cdot}$ se indica el valor estimado de una variable. Por ejemplo \hat{x} es el estado estimado de x .

3.3.1. Implementación

En este proyecto, se ha implementado el filtro de Kalman con el fin de mejorar el posicionamiento del vehículo, juntando así las informaciones de la odometría (modelo dinámico) con las mediciones del *LIDAR* y de la *IMU*.

Se definen las variables de estado x y las entradas u del sistema como sigue:

$$x = \begin{bmatrix} x \\ y \\ \theta \\ v_x \\ v_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \text{posición eje x} \\ \text{posición eje y} \\ \text{orientación eje z} \\ \text{velocidad lineal eje x} \\ \text{velocidad lineal eje y} \\ \text{velocidad angular eje z} \end{bmatrix} \quad u = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \quad (3.3)$$

En la ecuación 3.3, se consideran como entradas del sistema las velocidades angulares de las ruedas ω_i medidas de los encoders.

El modelo de predicción del estado se obtiene del conjunto entre la cinemática del vehículo (ecuación 1.5) y la odometría (ecuaciones 1.10, 1.11, 1.12), quedando:

$$\hat{x}_{k+1} = F\hat{x}_k + Bu_k \quad (3.4)$$

$$\text{con } F = \begin{bmatrix} 1 & 0 & 0 & \Delta T & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta T & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta T \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \frac{r}{4} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ \frac{1}{(a+b)} & -\frac{1}{(a+b)} & \frac{1}{(a+b)} & -\frac{1}{(a+b)} \end{bmatrix}$$

La matriz de covarianza de ruido del proceso, relaciona cuantos una variable cambia en función de otra variable que presenta ruido.

Por ejemplo nos informa de cuantos cambiaria el valor de la posición x si la velocidad v_x presenta ruido. Si las variables son independientes entre sí, la matriz Q será diagonal. En el caso concreto la matriz no sería diagonal en cuanto las variables x, y, θ son dependientes de v_x, v_y, ω_z .

Sin embargo, se han obtenido resultados mejores ajustando de forma experimental la matriz Q como sigue:

$$Q = \begin{bmatrix} q_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & q_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & q_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & q_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & q_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & q_{66} \end{bmatrix} \quad (3.5)$$

con $q_{11} = q_{22} = q_{44} = q_{55} = 0,001^2$ $q_{33} = q_{66} = 0,8^2$

El modelo de medición se obtiene a partir de los sensores que pueden medir las variables de sistemas, en este caso se utiliza el *LIDAR* para medir la posición y la orientación mientras la *IMU* solo para la orientación del vehículo.

La ecuación es:

$$\text{con } z = \begin{bmatrix} \theta_{imu} \\ x_{lidar} \\ y_{lidar} \\ \theta_{lidar} \end{bmatrix} \quad z_k = Hx_k \quad (3.6)$$

$$H = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

En lo referente a la matriz de covarianza de ruido de medición R , se calcula a partir de la varianza de los sensores empleados.

La varianza, indicada con σ^2 , es una medida que se utiliza para cuantificar la variación o la dispersión de un conjunto de datos numéricos. Una varianza

baja indica que la mayor parte de los datos de una muestra tienden a estar agrupados cerca de su media (también denominada el valor esperado), mientras que una varianza alta indica que los datos se extienden sobre un rango de valores más amplio, Figura 3.5.

Un sensor con baja varianza indica una elevada precisión y un sensor con alta varianza indica poca precisión.

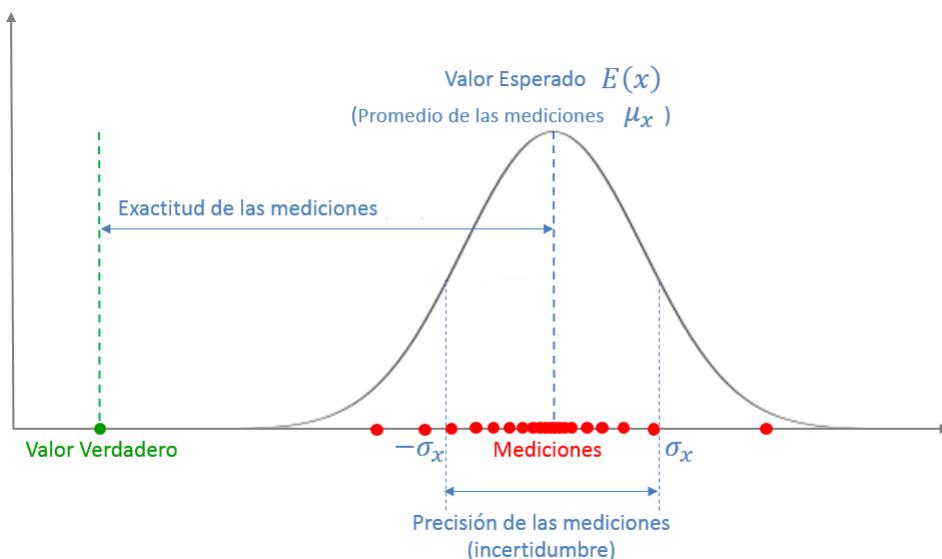


Figura 3.5: Campana Gaussiana y desviación estándar

Se ha definido la matriz R de forma experimental como sigue:

$$R = \begin{bmatrix} r_{11} & 0 & 0 & 0 \\ 0 & r_{22} & 0 & 0 \\ 0 & 0 & r_{33} & 0 \\ 0 & 0 & 0 & r_{44} \end{bmatrix} \quad (3.7)$$

con $r_{11} = r_{44} = 0,05^2$ $r_{22} = r_{33} = 0,01^2$

En el capítulo cuatro se explicará mas detalladamente la motivación de la elección de estos valores.

Capítulo 4

Sistema de seguimiento de trayectoria

En este capítulo se describen las diferentes técnicas empleadas para obtener un sistema de seguimiento de trayectoria más preciso posible.

La trayectoria es el lugar geométrico de las posiciones sucesivas por las que pasa un cuerpo en su movimiento.

Con sistema de seguimiento de trayectoria se entiende un sistema capaz de alcanzar las posiciones de una trayectoria con el menor error posible.

Para que el vehículo siga una trayectoria de referencia, es necesario conocer la posición del vehículo en cada instante.

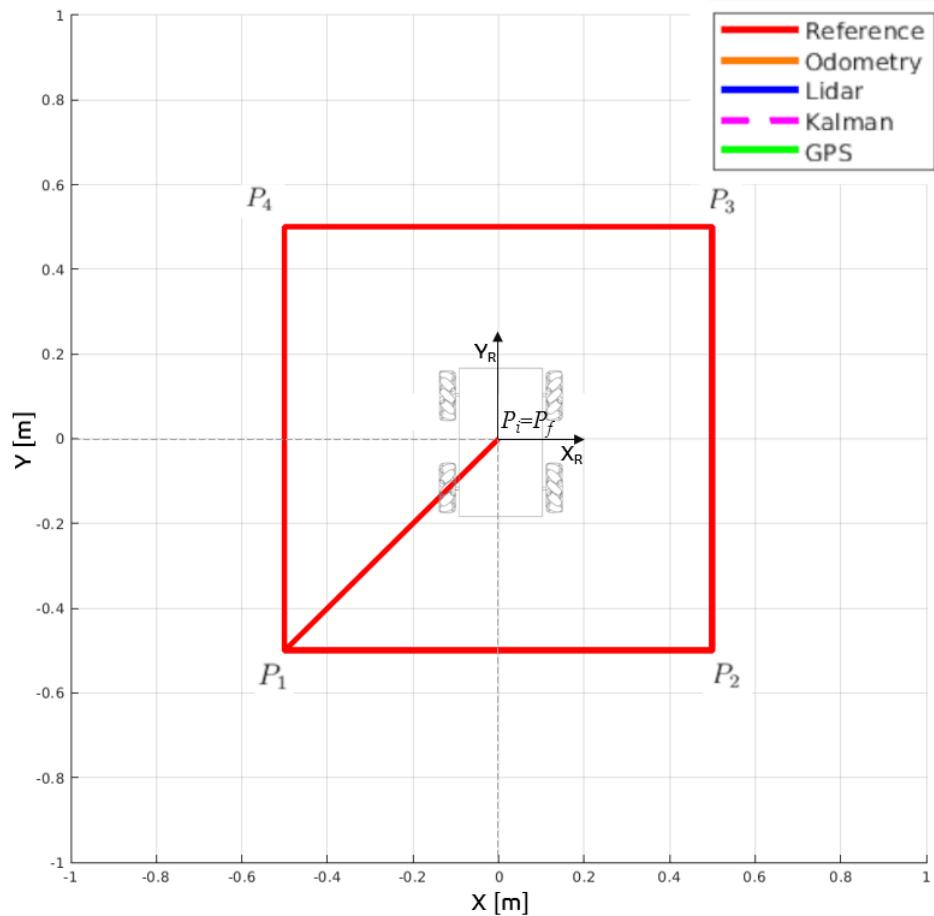


Figura 4.1: Trayectoria de referencia y leyenda

Para poder comparar mejor los resultados de las diferentes técnicas que se van a presentar, se utiliza la misma trayectoria de referencia y el mismo código color en las graficas, visibles en Figura 4.1.

En particular la trayectoria corresponde a un cuadrado de lado 1 m, centrado en la posición inicial $P_i(x, y) = (0, 0)$.

El cuadrado se repite dos veces y al terminar, el vehículo vuelve a la posición inicial. Además, en las graficas de la posición a continuación, el cambio de color de más claro a más oscuro identifica el paso del tiempo.

4.1. Odometría

En esta primera prueba experimental, se ha utilizado solo la odometría como sistema de posicionamiento.

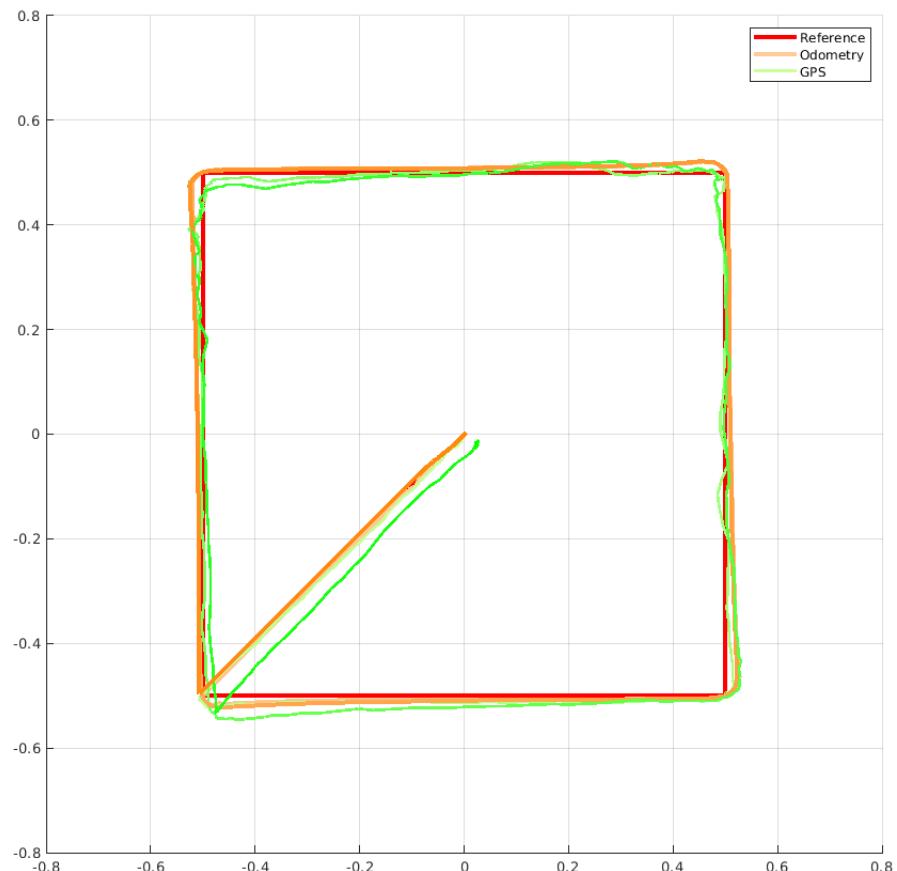


Figura 4.2: Seguimiento de trayectoria con odometría

Como se aprecia en la Figura 4.2, el vehículo sigue bien la referencia, pero la acumulación de errores debido a la integración, causa un error notable en la posición final.

Al volver a la posición inicial la odometría calcula que el vehículo está en $(0,0)$, pero si se compara con la posición real del *GPS* se nota un error de $+2\text{ cm}$ en x y -2 cm en y .

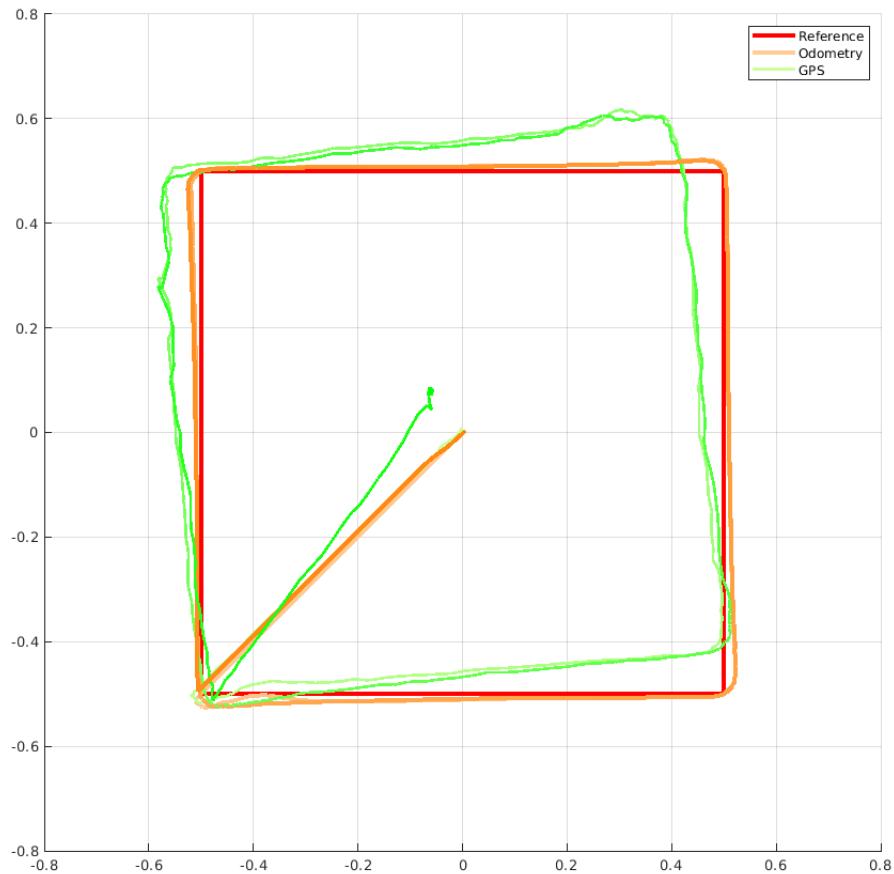


Figura 4.3: Seguimiento de trayectoria con odometría y perturbación

En esta segunda prueba, se ha acentuado el efecto de los errores introduciendo una perturbación. De echo, como se aprecia en las Figuras 4.4, se ha aplicado una fuerza externa que ha causado un cambio en la orientación del vehículo de aproximadamente $0,10 \text{ rad} \approx 6^\circ$.

La Figura 4.3, confirma lo comentado a cerca de los errores a largo plazo de la odometría explicados en el apartado 1.3. En resumen, los errores se acumulan debido a la integración en el cálculo de la odometría y que con el paso del tiempo, la posición real se alejará siempre más de la calculada.

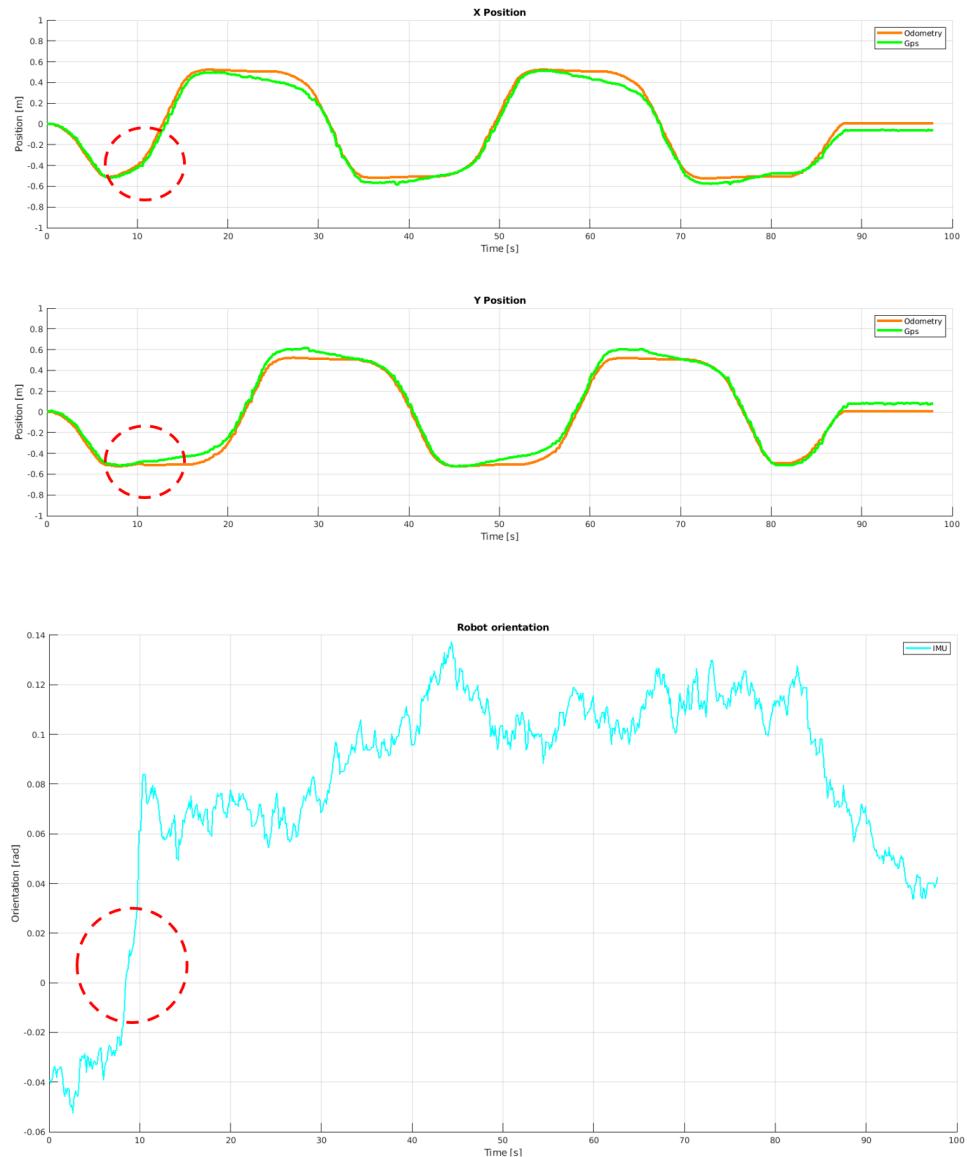


Figura 4.4: Posición y orientación en el tiempo con perturbación

En la Figura 4.4 se ha evidenciado con un circulo rojo el momento en el cual se ha introducido la perturbación.

A partir de la introducción de la perturbación, el vehículo no puede seguir más la referencia porque no conoce su posición respecto a su entorno, siendo la odometría un sistema de posicionamiento relativo que depende de la posición y orientación inicial.

Para poder seguir la trayectoria de referencia correctamente también ante perturbación, el vehículo necesita tener un sistema de posicionamiento absoluto.

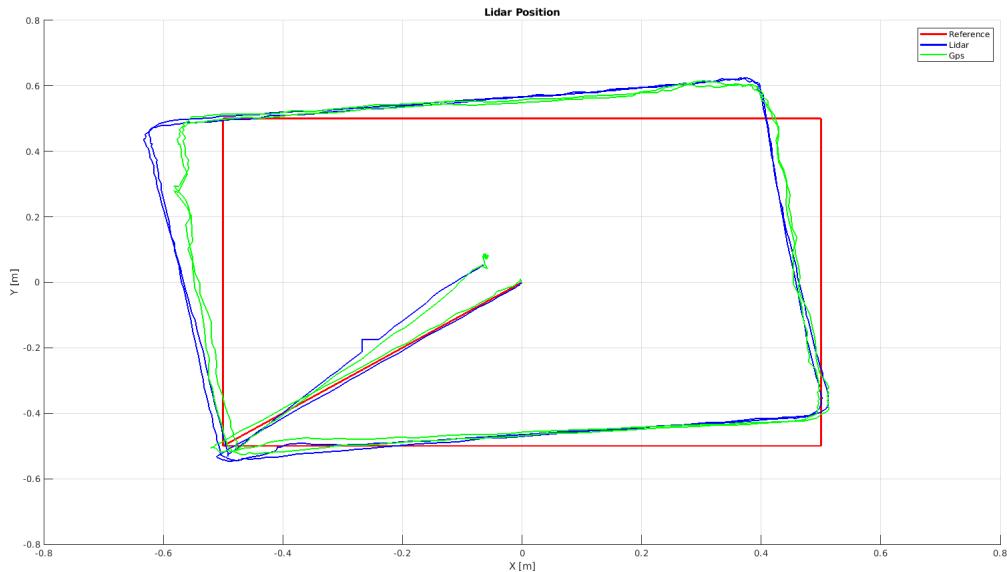


Figura 4.5: Comparación entre la posición del *LIDAR* y el *GPS*

Se puede apreciar de la Figura 4.5, que durante la prueba anterior, la posición proporcionada por el *LIDAR* es muy similar a la real proporcionada por el *GPS*.

La idea es utilizar la posición del *LIDAR* como sistema de posicionamiento absoluto del vehículo. La solución implementada más simple y inmediata es la de corregir el valor de posición calculado por odometría sustituyéndolo por el medido del *LIDAR*.

Esta técnica ha sido ensayada y sucesivamente descartada porque el cambio repentino de la posición causaba inestabilidad en el sistema. A continuación, veremos que el uso del filtro de Kalman permite suavizar esta transición obteniendo un sistema estable y de mejor precisión.

4.2. Filtro de Kalman

El filtro de Kalman, como se ha explicado en el apartado 3.3, permite fusionar las informaciones provenientes de diferentes sensores con las del modelo dinámico del sistema, con el fin de estimar el valor más probable de las variables de sistema.

Como se ha demostrado en el ensayo anterior, el *LIDAR* utilizado, tiene una buena precisión y por lo tanto una baja varianza. Por este motivo, se ha asignado un valor de 0,01 a la posición medida del *LIDAR* en la matriz R , como se puede apreciar en la ecuación 3.7.

De tal manera, el filtro de Kalman “confiará” más en los datos de las medidas del *LIDAR* respecto a la calculada por el modelo del sistema.

En lo referente a la orientación, se fusionan las medidas de la *IMU* y del *LIDAR*, y una vez calculado el valor estimado con Kalman, se aplica un control *PID* para mantener la orientación fija. Este control favorece el seguimiento de trayectoria del vehículo, pero la acción de las ruedas que permiten al vehículo mantener la orientación, se repercute negativamente en la odometría.

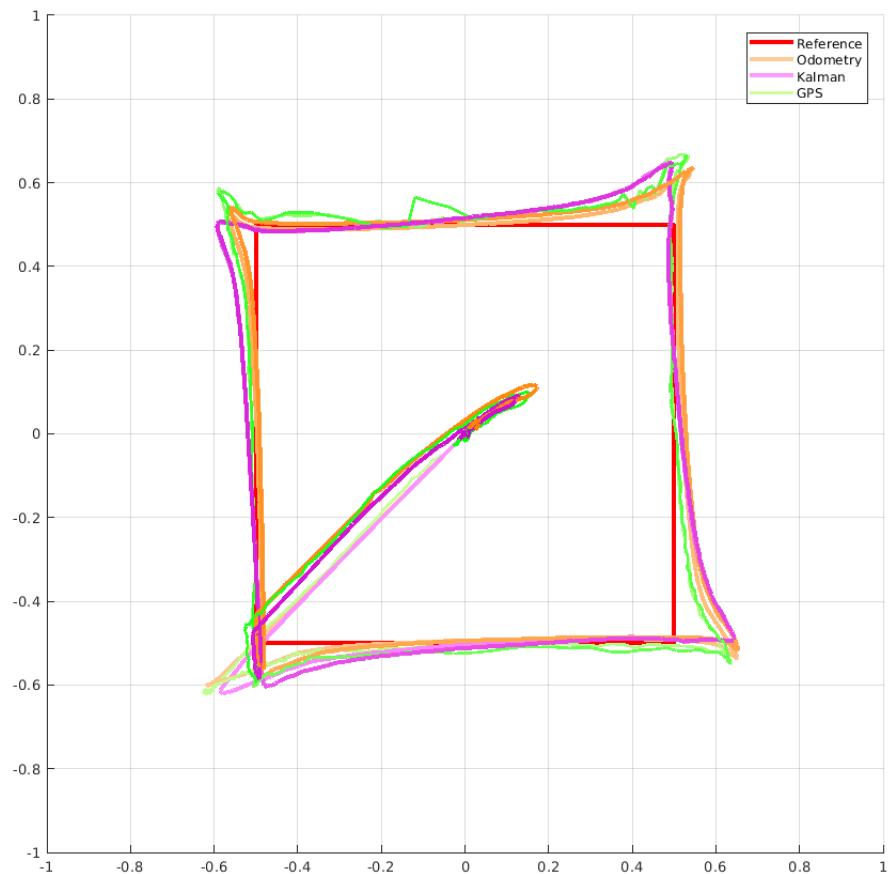


Figura 4.6: Seguimiento de trayectoria con filtro de Kalman

En la Figura 4.6, se nota como el vehículo recupera la trayectoria de referencia también después de la introducción de la perturbación. Sin embargo, cabe destacar que el vehículo no se para en los puntos de la trayectoria si no que sobrepasa estas posiciones.

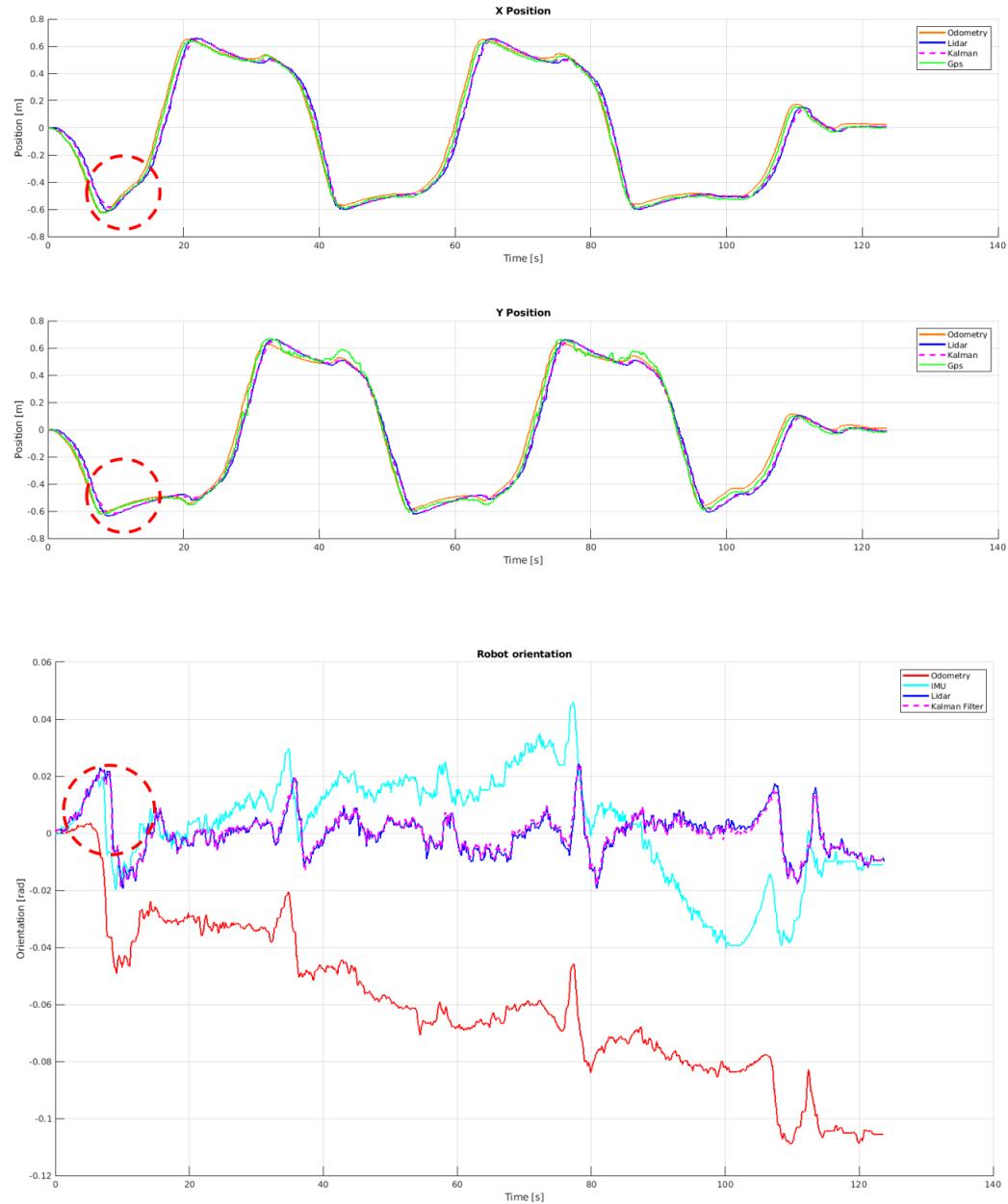


Figura 4.7: Posición y orientación en el tiempo

Esto se debe a que las medidas del *LIDAR* vienen proporcionadas y procesadas en un tiempo mucho mayor respecto al tiempo con el cual viene actualizado el filtro de Kalman.

En particular, se nota en la Figura 4.8, el retraso de aproximadamente un segundo entre la posición real del vehículo y la posición proporcionada por el *LIDAR*.

Este retraso está causado por dos factores: la frecuencia de escaneo del mismo sensor y el tiempo de elaboración del algoritmo de *SLAM*.

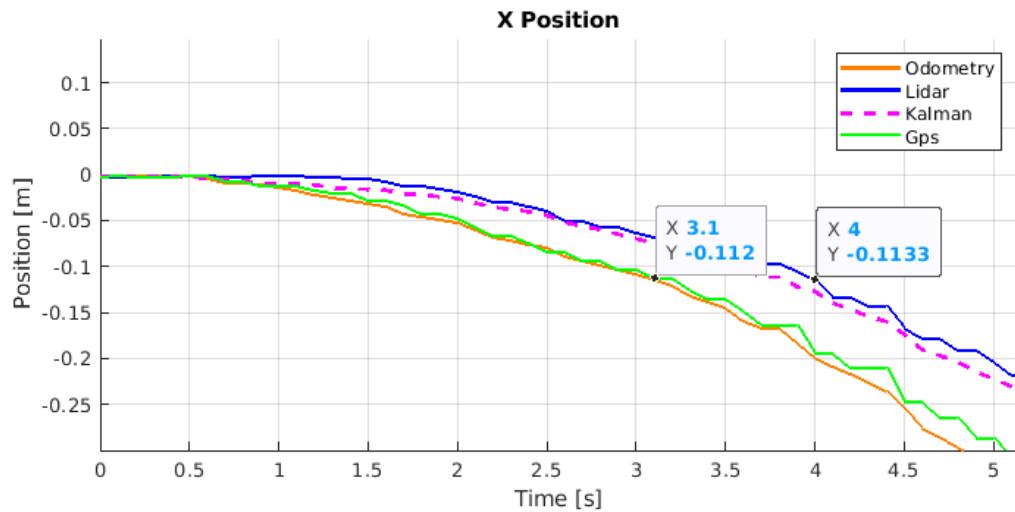


Figura 4.8: Retraso en la posición del *LIDAR*

Para solucionar este problema se han implementado dos modificaciones al filtro de Kalman:

1. Se han separado las etapas de predicción y corrección del filtro, en modo tal que la corrección se aplique solo cuando está disponible una nueva medida del *LIDAR*. Sabiendo que la frecuencia de escaneo del *LIDAR* es de 10 *Hz* y que el tiempo de procesamiento del algoritmo *SLAM* no es despreciable pero desconocido, se ejecuta el paso de corrección cada 200 *ms*. Durante este periodo de tiempo, se ejecuta solo el paso de predicción, utilizando el modelo del sistema para estimar la posición y orientación del vehículo.

2. Se ha corregido la medida del *LIDAR* estimando la posición real tras la compensación del retraso del dato. En el tiempo que tarda el sensor y el algoritmo a proporcionar la medida, el vehículo se habrá desplazado y por lo tanto, conociendo la velocidad durante este tiempo se puede estimar cual sería la medida real.

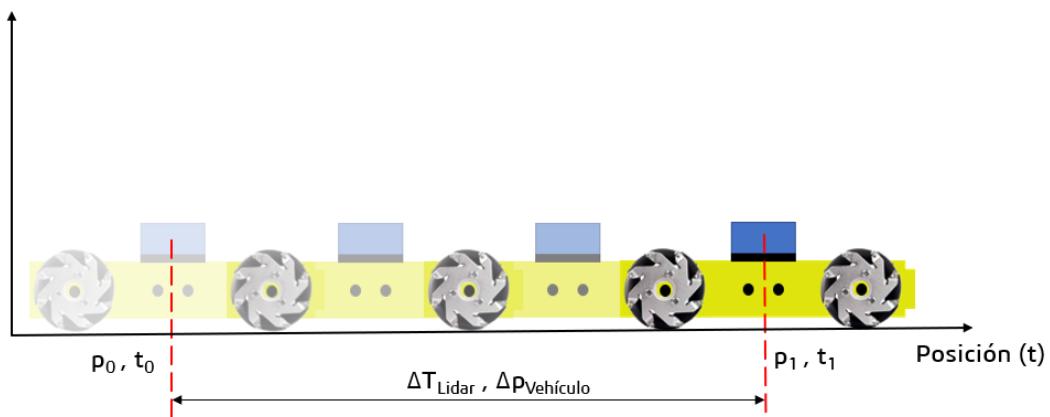


Figura 4.9: Corrección de la medida de posición del *LIDAR*

En la Figura 4.9, se pude apreciar conceptualmente lo comentado en el punto 2. Se nota que, debido al retraso del *LIDAR*, la posición recibida al instante t_1 , sea en realidad la posición medida en el instante t_0 . Si durante el periodo de tiempo entre estos dos instantes, definido con ΔT_{Lidar} , el vehículo se ha desplazado de una distancia $\Delta p_{Vehículo}$ a una velocidad media v_{media} , se puede estimar la posición p al tiempo t_1 como:

$$p_{estimada,t1} = p_{Lidar,t1} + v_{media} \cdot \Delta T_{Lidar} \quad (4.1)$$

4.2.1. Filtro de Kalman y corrección del Lidar

A continuación se muestra el algoritmo implementado del filtro de Kalman con corrección del *LIDAR*, teniendo en consideración que las operaciones de corrección y predicción del filtro corresponden a las ecuaciones explicadas en el apartado 3.3.

Algorithm 2: Filtro de Kalman con corrección del *LIDAR*

```

 $u \leftarrow$  velocidades de las ruedas ;
 $T_{Lidar} \leftarrow 0,2;$  // periodo de corrección
 $\Delta T_{Lidar} \leftarrow 0,96;$  // retraso del Lidar
if  $t \geq T_{Lidar}$  then
     $z_0 \leftarrow orientación_{z,IMU};$ 
     $z_1 \leftarrow posición_{x,Lidar} + v_{x,media} * \Delta T_{Lidar};$ 
     $z_2 \leftarrow posición_{y,Lidar} + v_{y,media} * \Delta T_{Lidar};$ 
     $z_3 \leftarrow orientación_{z,Lidar} + \omega_{z,media} * \Delta T_{Lidar};$ 
    Predicción filtro de Kalman;
    Corrección filtro de Kalman;
else
    | Predicción filtro de Kalman;
end

```

Esta rutina corresponde a la operación de actualización del filtro de Kalman visible en el algoritmo de la sección 3.2.

El filtro ejecuta el paso de predicción cada 20 ms y el paso de corrección cada 200 ms .

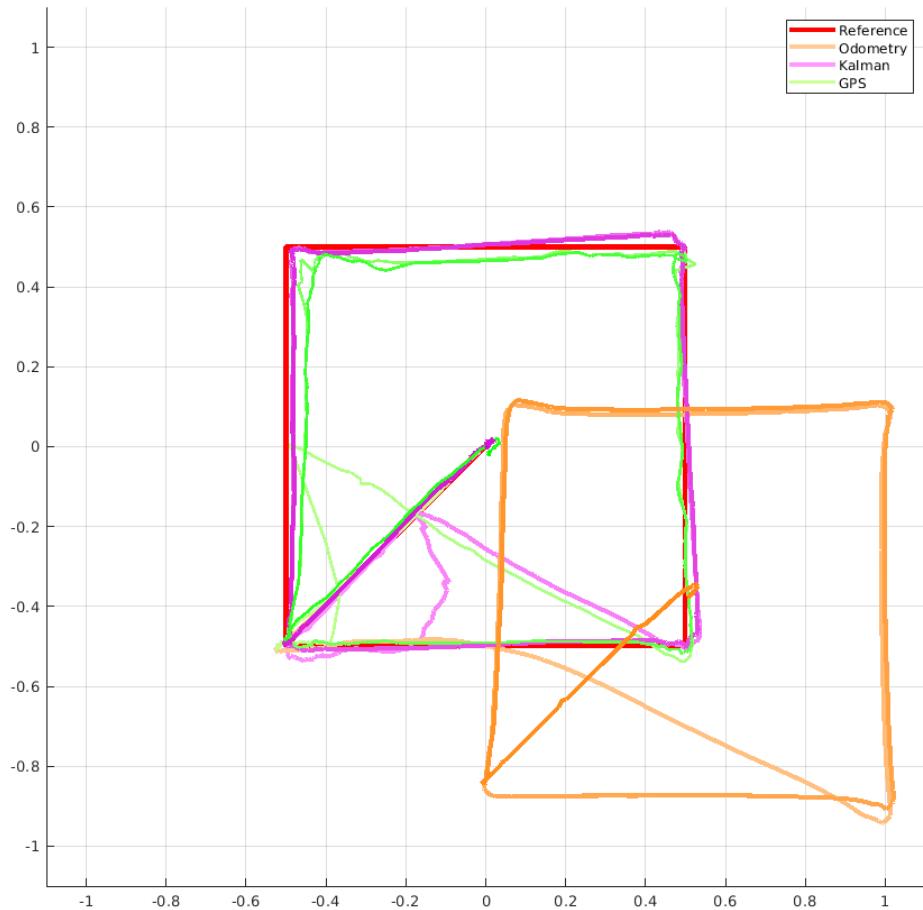


Figura 4.10: Seguimiento de trayectoria con filtro de Kalman y corrección

Como se aprecia en la Figura 4.10, tras la introducción de la perturbación, el vehículo es capaz de recuperar la trayectoria de referencia.

Como se nota, las señales del filtro de Kalman (en violeta) y la del *GPS* (en verde) son muy parecidas y siguen la trayectoria de referencia (en rojo). Sin embargo después de la perturbación, la posición calculada por odometría (en naranja) es muy diferente de la real teniendo un error al final de la trayectoria de $0,5\text{ m}$ en x y $-0,38\text{ m}$ en y .

Como se observa en la Figura 4.11, la perturbación introducida ha causado un desplazamiento de aproximadamente $-0,10\text{ m}$ en x y $0,5\text{ m}$ en y , además de un cambio en la orientación de $-0,28\text{ rad} \approx -16^\circ$.

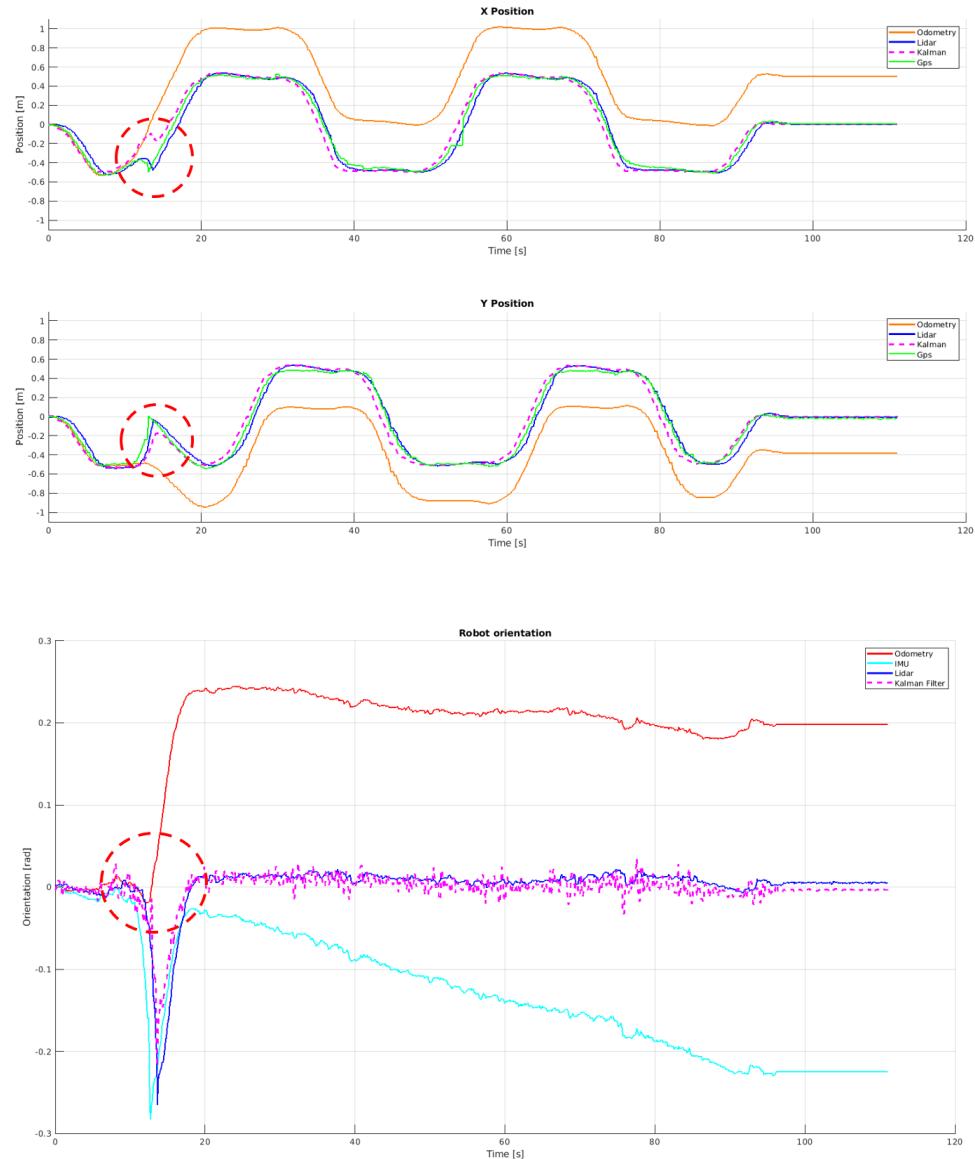


Figura 4.11: Posición y orientación en el tiempo

Cabe destacar que la posición final real es igual a la inicial de $(x, y) = (0, 0)$ así como la orientación es de 0° .

Resumiendo, gracias al filtro de Kalman y a la corrección de la medida de posición del *LIDAR*, se ha obtenido un sistema de seguimiento de trayectoria con un error final nulo en posición y orientación.

A demostración de la efectividad de este sistema de seguimiento de trayectoria, se reportan diferentes ensayos con trayectorias de referencias más complejas.

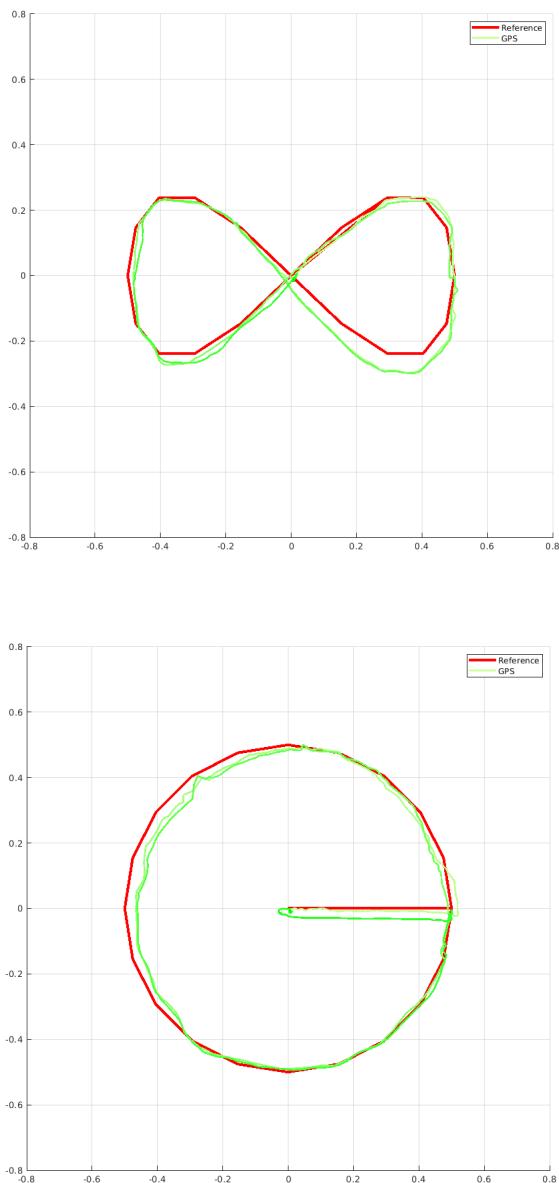


Figura 4.12: Trayectoria con curva a ocho y trayectoria circular

Presupuesto

En este apartado se estima el coste económico del proyecto. Los presupuestos están divididos en varios apartados en función de su naturaleza dentro del propio proyecto.

El primer presupuesto contiene el precio de cada componente utilizado para el montaje del vehículo.

Componente	Coste por unidad [€]	Unidades	Total [€]
4WD Mecanum Wheel Arduino Robot	647,47	1	647,47
Arduino Due	29,65	1	29,65
Adafruit IMU BNO055	27,99	1	27,99
Adafruit Motor Shield v2	30,00	1	30,00
Polulu 37Dx70L con encoder 64 CPR	33,58	4	134,32
Raspberry Pi 4	56,68	1	56,68
YDLIDAR X4	100,73	1	100,73
Marvelmind Indoor GPS (*)	418,57	1	418,57
Ordenador (**)	550,00	1	550,00
			1995,41

(*) Opcional

(**) El coste depende de las especificaciones

En segundo lugar, se analiza el coste derivado del sueldo percibido por las personas que han participado en este proyecto, en este caso se considera que el único personal retribuido es el autor. Para la estimación de este sueldo, en base al *XIX Convenio colectivo del sector de empresas de ingeniería y oficinas de estudio técnicos*, se establece que sea de *20€/hora*.

Actividad	Coste por hora [€]	Horas	Total [€]
Análisis del problema	20,00	5	100,00
Diseño de la solución	20,00	4	80,00
Montaje	20,00	16	320,00
Implementación software	20,00	50	1000,00
Pruebas experimentales	20,00	40	800,00
Depuración de errores	20,00	10	200,00
			2500,00

Una vez estipulados los diferentes costes del proyecto, se muestran de manera conjunta en la tabla a continuación, en la que se puede observar el coste final del proyecto.

Concepto	Total [€]
Material	1995,41
Personal	2500,00
Licencia Matlab	2000,00
	6495,41

Se puede apreciar que estos presupuestos están sujetos a modificaciones como el uso de una licencia ya en propiedad o la utilización de material disponible.

Conclusiones

El objetivo de desarrollar un sistema de seguimiento de trayectoria para un vehículo omnidireccional de cuatro ruedas *Mecanum* se considera cumplido.

Se han comprobado las características de omnidireccionalidad de las ruedas *Mecanum* así como las problemáticas que conllevan.

Se ha implementado correctamente el sistema de control basado en el microcontrolador *Arduino Due*, que se encarga del control del movimiento del vehículo. El software del sistema de control incluye el control de velocidad de las ruedas además de las ecuaciones de la cinemática directa y inversa, cuya implementación se ha demostrado correcta tras diferentes ensayos de movimiento del vehículo.

El sistema de procesamiento implementado, basado en la tarjeta *Raspberry Pi 4* y en el software *ROS*, ha permitido gestionar el *LIDAR* además de mandar el sistema de control y comunicar con el ordenador en remoto de manera inalámbrica.

El desarrollo de la aplicación grafica con el software *Matlab* ha facilitado la visualización de la telemetría del vehículo y el análisis a posteriori de los datos. En lo referente al sistema de seguimiento de trayectoria, se puede con-

cluir que como se ha demostrado en los ensayos, no se puede utilizar solo la odometría como sistema de posicionamiento. Por este motivo se ha empleado el *LIDAR*, que junto al algoritmo de *SLAM* permite conocer la posición absoluta del vehículo.

Además, se observa que gracias a la implementación del filtro de Kalman, los resultados del seguimiento de trayectoria han mejorado notablemente y por lo tanto se queda como mejor solución encontrada.

Sin embargo, se podrían mejorar las prestaciones del sistema optimizado el filtro de Kalman tras solucionar el problema de sincronización de los datos proporcionados por los diferentes sensores empleados.

Además, es probable que el uso de sensores con características superiores podrían mejorar aún más los resultados.

Todo el software y la documentación adicional de la implementación se pueden encontrar al siguiente enlace [10].

Bibliografía

- [1] Veaceslav Spinu Ioan Doroftei, Victor Grosu. *Omnidirectional Mobile Robot – Design and Implementation.* PhD thesis, “Gh. Asachi” Technical University of Iasi, Romania, 2007.
- [2] Nexus Robot. <https://www.nexusrobot.com/>.
- [3] G. Bright O. Diegel, A. Badve and J. Potgieter. *Improved Mecanum Wheel Design for Omni-directional Robots.* pp. 27-29,, November, 2002.
- [4] Nurallah Ghaeminezhad. Hamid Taheri, Bing Qiao. *Kinematic Model of a Four Mecanum Wheeled Mobile Robot.* International journal of computer applications, volume 113 - no.3,, March 2015.
- [5] Wikipedia. <https://es.wikipedia.org/wiki/odometria>.
- [6] ROS. <https://www.ros.org/about-ros/>.
- [7] Wikipedia. <https://es.wikipedia.org/wiki/localizacionymodelo de los simultaneos>.
- [8] Arun; Somani; Thomas S. Huang; Steven D. Blostein. *Least-square fitting of two 3-D point sets.* PhD thesis, IEEE Pattern Analysis and Machine Intelligence, 1987.
- [9] Hyochoong Bang Youngjoo Kim. *Introduction to Kalman Filter and Its Applications.* PhD thesis, Korea Advanced Institute of Science and Technology, Daejeon, South Korea, 2018.
- [10] Stefano Andreoni. <https://github.com/andresteve/mecanumwheelrobot>.