Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN INGENIERÍA DE CONTROL, AUTOMATIZACIÓN Y ROBÓTICA

TRABAJO FIN DE MÁSTER

DESARROLLO DE ALGORITMOS DE LOCALIZACIÓN Y NAVEGACIÓN DE VEHÍCULOS INDUSTRIALES EN INTERIORES

Estudiante *Cabezas Olivenza, Mireya*
Director/Directora *Zulueta Guerrero, Ekaitz*
Departamento *Ingeniería de Sistemas y Automática*
Curso académico *2020-2021*

Bilbao, 24 de junio del 2021

Resumen

CASTELLANO

La robótica móvil se presenta como una tecnología revolucionaria, debida a su adaptabilidad a diferentes sectores, pudiendo automatizar su transporte interno. Con la seguridad y flexibilidad que muestran los AGV (Automated Guided Vehicles), se ofrecen como recursos que pueden hacer que las industrias incrementen su productividad.

En el presente Proyecto Fin de Máster, se plantea y resuelve un algoritmo de localización y navegación dirigido al control de los vehículos industriales autónomos. Para ello, se procede al uso de diferentes algoritmos y una red neuronal, que habilitan el movimiento libre de un AGV. El programa implementado a través de Matlab, ofrece una visualización del correcto funcionamiento del diseño. De igual modo, a lo largo del trabajo, se puede percibir el enfoque de innovación que se ha tratado de dar, demostrando su validez en el campo de la robótica móvil.

Palabras clave: Navegación, Redes Neuronales Convolucionales, Vehículos Autónomos, Aprendizaje Profundo.

EUSKERA

Robotika mugikorra teknologia iraultzaile gisa aurkezten da, sektore desberdinetara egokitu daitekeelako, hauen barne garraioa automatizatuz. AGV-ek (Automated Guided Vehicles) erakusten duten segurtasun eta malgutasunaz, industrien produktibitatea handitzeko gai diren baliabide gisa eskaintzen dira.

Master Amaierako Proiektu honetan, industria-ibilgailu autonomoak kontrolatzeko kokapen eta nabigazio algoritmo bat planteatu eta ebazten da. Horretarako, hainbat algoritmo eta sare neuronal bat erabiltzen dira, AGV-aren mugimendu askea ahalbidetzen dutenak. Matlaben bidez inplementatutako programak diseinuaren funtzionamendu egokia bistaratzen du. Era berean, lanean zehar, eman nahi izan den berrikuntza ikuspegia antzeman daiteke, robotika mugikorraren arloan baliozkoa dela erakutsiz.

Hitz gakoak: Nabigazioa, Sare Neuronal Konboluzionalak, Ibilgailu Autonomoak, Ikaskuntza Sakona.

INGLÉS

Mobile robotics is presented as a revolutionary technology, due to its adaptability to different sectors, being able to automate their internal transport. With the safety and flexibility shown by AGVs (Automated Guided Vehicles), they are offered as resources that can help industries to increase their productivity.

In this Master's Thesis, a localisation and navigation algorithm aimed at the control of autonomous industrial vehicles is proposed and solved. For this purpose, different algorithms and a neural network are used to enable the free movement of an AGV. The programme, implemented through Matlab, offers a visualisation of the correct functioning of the design. Likewise, throughout the work, it is possible to perceive the innovative approach that has been tried to give, demonstrating its validity in the field of mobile robotics.

Keywords: Navigation, Convolutional Neural Networks, Autonomous Vehicles, Deep Learning.

Índice de contenido

Índice de figuras

Índice de tablas

Memoria Técnica

1 Introducción

Los vehículos guiados automáticamente, también conocidos como AGV (Automated Guided Vehicles), llevan años en uso en el sector industrial. Esta tecnología hace que dichos vehículos circulen de forma autónoma sin la necesidad de que los humanos intervengan. Para ello se requiere de un conocimiento del entorno, conocido como localización y mapeo simultáneo (SLAM). Esto se puede realizar a través de sensores o visión por computador. Una vez conocida la zona en la que se va a desplazar, es posible estimar la mejor trayectoria hasta un punto objetivo y que el robot móvil la siga. Además, entra en juego la importancia de detectar obstáculos, tanto móviles como fijos, para evitar cualquier colisión.

Se pretende desarrollar un elemento nuevo de automatización, como pueden ser los PLC o los robots industriales, para su aplicación en fábricas. Por lo tanto, puede deducirse que la problemática principal del presente trabajo es la de desarrollar algoritmos de localización y navegación para vehículos industriales en interiores. El trabajo se ha llevado a cabo en la escuela de ingeniería de Vitoria-Gasteiz, junto con el Director del proyecto, Ekaitz Zulueta Guerreo.

Figura 1 Prototipo de AGV industrial de la escuela ingeniería de Vitoria-Gasteiz

Es bien sabido que los sistemas de navegación para este tipo de vehículos deben basarse en la lectura de datos de diferentes fuentes de información, con el objetivo de aumentar la exactitud de la localización estimada del robot móvil. De esta manera es posible recorrer la trayectoria deseada de manera eficiente.

Con tal fin, la técnica que usan los vehículos autónomos industriales aporta múltiples ventajas al campo de la robótica móvil. Por una parte, es necesaria la gestión de la información que se recoge del entorno para poder estimar la propia localización del vehículo autónomo, que no solo incluye su posición en ejes espaciales, sino también su dirección. Esto contribuye a la metodología para el tratamiento de datos, ya que los AGV suelen hacer uso de filtros para quedarse únicamente con los valores que marcan diferencias entre características del entorno. Por otra, la capacidad de la que se le dota al vehículo para calcular la trayectoria que ha de seguir y ser capaz de continuarla, pudiendo evitar obstáculos, hace que los vehículos autónomos sean adaptables a diferentes entornos.

En lo que a las personas se refiere, se eleva el nivel de automatización de los vehículos industriales, pudiendo hacer más llevaderos o incluso eliminar los puestos de trabajo más duros. Además, la producción será más fiable y eficiente.

Para el desarrollo del proyecto, se cuenta con el prototipo de vehículo industrial de la Figura 1, que se encuentra en la escuela de ingeniería de Vitoria-Gasteiz, el cual se puede controlar a través de Matlab, ya que incorpora un PLC de Beckhoff.

1.1 Contexto actual

En la actualidad existen diferentes tipos de vehículos autónomos industriales para interiores. Sin embargo, pocos son realmente de navegación libre. Habitualmente, los robots móviles que se pueden encontrar son filo guiados, que, si bien son un sistema robusto de navegación, el enfoque resulta poco flexible para la realización de infinidad de tareas en líneas de montaje. Por lo tanto, a día de hoy, la mayoría de AGV, se basan en seguir trayectorias preestablecidas por el humano, teniendo poca capacidad de respuesta ante cualquier situación imprevista como puede ser la obstaculización de la ruta fijada.

Por otra parte, los vehículos industriales de navegación libre suelen hacer uso de datos provenientes de diferentes fuentes, como los sensores LiDAR (Laser Imaging Detection and Ranging), la odometría basada en las ruedas, el filtro de partículas, el algoritmo de navegación DWA (Dynamic window approach), etc. siendo estos los enfoques más utilizados. Estas técnicas son muy convencionales, existiendo una necesidad de innovación e introducción de diferentes metodologías para conseguir la localización y navegación de AGV, visto que, en diferentes aplicaciones, no cumplen con necesidades como puede ser la seguridad.

1.2 Estructura de la memoria

Una vez habiendo presentado una visión general de la temática de este proyecto, los próximos apartados se distribuirán de la siguiente manera.

Primero se presenta un apartado de *Objetivos y Alcance* donde se desglosan los objetivos a cumplir y se explica que se va a desarrollar. Tras esto, un capítulo de *Análisis del estado del arte*, en el cual se presenta un estudio del avance de la tecnología hasta la actualidad. Además, se especifican las ideas que se han tomado como base. Se continuará con el propio *Desarrollo del algoritmo de localización y navegación*, dividido en el "Diseño del algoritmo de navegación" y el "Diseño del algoritmo de planificación de trayectoria y localización", presentando todo el trabajo realizado. Después, se expone un *Análisis de los resultados* donde se muestra el nivel de acatamiento de los objetivos. Por último, se tienen los apartados de *Conclusiones* y *Bibliografía*, que darán paso a los *Anexos*.

2 Objetivos y alcance

El propósito de este proyecto es el desarrollo de un algoritmo de navegación para vehículos autónomos industriales. Para llevar a cabo esta finalidad, es necesario desglosar el objetivo principal en diferentes tareas.

2.1 Objetivos

En lo que al control de vehículos autónomos se refiere, se pueden reconocer diferentes problemáticas. Una de ellas es la localización propia que el vehículo hace sobre sí mismo y el reconocimiento del entorno en el que está situado (SLAM). Para ello, se tiene en cuenta que dicha ubicación, además de depender de su posicionamiento en los ejes $\boldsymbol{x}$ e $\boldsymbol{y}$ o incluso $\boldsymbol{z}$, necesita un valor $\boldsymbol{\theta}$ que especifique la dirección. El conjunto de estos datos se denomina $\overrightarrow{pose}$. Una vez conocida esta primera información del AGV, se va a requerir que el vehículo siga una trayectoria, a saber, "path following" o "navigation". Para poder cumplir con este objetivo, el tercero de los problemas a resolver es el "path planning", referido al cálculo del propio recorrido.

Se tiene en cuenta que los dos primeros objetivos se deben de realizar on-line, mientras que el tercero es posible calcularlo off-line. Así pues, se debe de aclarar que no es lo mismo el concepto de planificación de trayectoria y los de navegación y localización.

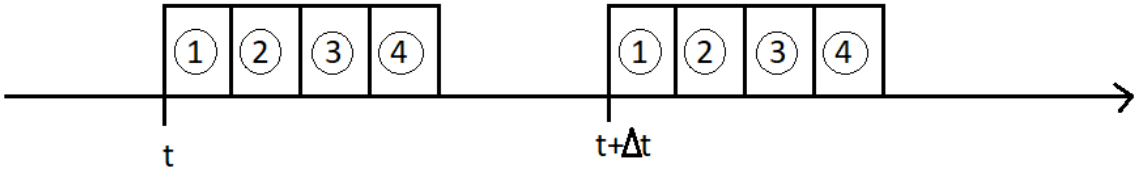


Figura 2 Diagrama de control temporal de un AGV

Desde el punto de vista temporal del control (véase Figura 2), una de las partes se centra en la realización de todas las medidas necesarias a través de diferentes métodos, como bien son los sensores LiDAR, la odometría de las ruedas, la visión por computador o cualquier otro tipo de sensórica (1). Tras esto y contando con toda la información necesaria, ya es posible calcular la $\overrightarrow{pose}$ del AGV (2). Se sigue con el propio algoritmo de navegación (3) que repercute sobre el control de las ruedas del vehículo (4), habiendo definido la velocidad previamente. Este ciclo se ha de repetir cada "t" ms, viendo la necesidad de que sea un cálculo on-line, dado que se busca la mayor eficiencia en el seguimiento de la trayectoria. De este modo, se manifiesta la relación de los objetivos de localización y navegación.

En cuanto al "path planning", no es necesario ese cálculo on-line, ya que cabe la posibilidad de estimar la trayectoria por completo una única vez y mantener dicha información disponible para que los demás algoritmos la usen. También es viable realizar la planificación por tramos, pero la repetición de ese cálculo dependerá de lo que tarde el algoritmo de navegación en recorrer la ruta, y no de un periodo de tiempo determinado.

Por lo tanto, dentro de la globalidad del desarrollo de un algoritmo de navegación y localización para vehículos industriales autónomos, hay que plantear diferentes sistemas. En consecuencia, los principales objetivos a conseguir y en los que se va a basar este documento son:

- Conseguir la $\overrightarrow{pose}$ del robot.
- Lograr realizar un seguimiento de la ruta, englobando un control lateral y un control longitudinal.
- Efectuar el cálculo de la trayectoria para que los objetivos previos puedan cumplirse adecuadamente.

2.2 **Alcance**

Tal y como se ha mencionado, se pretende desarrollar un algoritmo de localización y navegación a través de Matlab. Se dispone de un prototipo de vehículo industrial en la escuela de ingeniería de Vitoria-Gasteiz, que se tomará como modelo para realizar el estudio, considerando que se pretende implementar el diseño sobre él.

Se resuelve la problemática de una aplicación concreta, pero tal y como se verá, el algoritmo desarrollado es flexible para su implementación en diferentes entornos y AGV.

3 Análisis del estado del arte

A pesar de que los tres sistemas para resolver los objetivos del control de los AGV tienen que coexistir, frecuentemente el tratamiento de estos se ha llevado a cabo de manera independiente. Los diferentes estudios previos se han centrado en un único propósito, por lo que el análisis bibliográfico también se ha realizado de manera separada. Además, con el avance de la tecnología, se ha pasado del estudio con métodos más convencionales, a abordar diferentes técnicas más novedosas que han dado resultados prometedores.

Recordemos que la demanda actual busca un sistema de navegación inteligente, con la fusión de los tres sistemas. Se pretende entonces, que además de estimar la trayectoria una vez conocido el entorno y su propia situación, hacer que el robot se mueva libremente y tome decisiones en relación con la zona, en otras palabras, dotarle de autonomía.

3.1 Problemática de la localización del vehículo

Como se ha visto en el apartado *2 Objetivos y alcance*, uno de los sistemas requeridos para poder realizar el control de los vehículos industriales autónomos es el que presenta la problemática de la localización o posicionamiento propio del vehículo. Estimar la $\overrightarrow{pose}$ y, basándose en ella, poder generar un mapa del entorno en el que se encuentra, es una de las tareas fundamentales de la robótica móvil. La decisión entre una localización en posicionamiento absoluto o relativo y el tipo de sensores con el que se va a trabajar ha sido tema de estudio años atrás.

Existen numerosas técnicas que han sido aplicadas en vehículos autónomos para la localización y mapeo simultáneo o SLAM, teniendo como objetivo construir un mapa de un entorno desconocido y referenciar el AGV en él. No obstante, estos procedimientos corresponden a otros campos, derivando en la necesidad de crear técnicas propias de la robótica móvil que estén asociadas al posicionamiento.

Conceptualmente, el propósito recae en estudiar el entorno, que es complejo debido a que varía constantemente. Para ello, hay que centrarse en el análisis de datos y su correspondiente representación. Computacionalmente, la cantidad de información que se puede obtener provoca un alto consumo de recursos, por lo que hay que considerar como procesarla. Esto hace que se cuestione si los algoritmos propuestos son adecuados y, por ende, la posibilidad de obtener soluciones cuya implementación se pueda hacer en tiempo real.

El método más tradicional que se puede encontrar es el basado en la odometría de las ruedas, que se apoya en el movimiento de estas. Autores plantean un algoritmo de actualización de velocidad cero (ZUPT), que proporciona información al sistema de navegación inercial INS. Existe la complicación de determinar la frecuencia con la que debe emplearse dicho algoritmo para conseguir equilibrar tanto la precisión de la localización, como la velocidad a la que se va a poder mover el vehículo. Se ha visto que se requiere de paradas cada cierto tiempo pronosticado para mejorar la información de la localización, ya que, mientras el vehículo autónomo está en movimiento, se genera un deslizamiento en las ruedas que es necesario predecir. Debido a que la medida se realiza por medio de codificadores en las ruedas, es posible conseguir la información del error a través de un modelo de proceso Gaussiano de series de tiempo. Este método da una precisión de un 97%, pero tiene limitaciones relacionadas con el tipo de terreno o el tipo de neumático [1]. Además, de esta manera solo se resuelve el problema de determinar el $\overrightarrow{pose}$, necesitando aun un mapeo del entorno.

El filtro de Kalman puede resultar de utilidad a la hora de estimar la localización del robot móvil si se une a la información aportada por la odometría de las ruedas, además de la que proporcionan sensores adicionales. Se ha llevado a estudio, con muy buenos resultados, ya que se producen valores que tienden a estar más cerca de las verdaderas mediciones espaciales. En este análisis se muestran posibles aplicaciones en las que resulta de utilidad, pero necesita más precisión a la hora de su empleo sobre vehículos autónomos [2].

El uso del filtro de partículas ha sido también muy recurrente a la hora de tratar de localizar AGV. Gracias a él, es posible hacer una estimación del estado de un entorno a lo largo del tiempo. Se compone de un conjunto de partículas o 'n' puntos al que se denomina "nube", con valores de $\overrightarrow{pose}$ diferentes asociados a cada partícula. Estos puntos se generan de manera aleatoria y se evalúan con las mediciones de los sensores. A través de comparaciones entre ellos y con la generación de celdas en el espacio, se consigue realizar el mapeo de un ambiente desconocido.

Un algoritmo basado en un filtrado Bayesiano ha resultado adecuado a la hora de localizar robots móviles. De este modo, no se requiere información previa del entorno acotado, pero debido al uso de dispositivos emisores y receptores, sí que es necesario conocer la ubicación de estos. Con dicha información se consigue el posicionamiento en el SLAM, además de que se reciben componentes de múltiples rutas que se tratan como señales de transmisión [3].

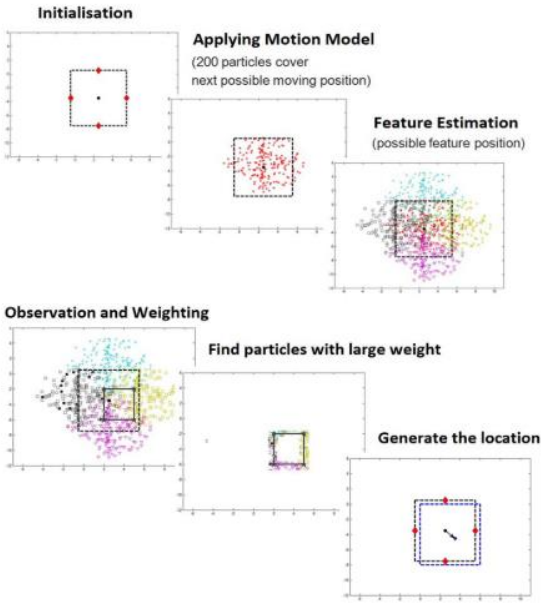


Figura 3 Ejecución de algoritmo de filtro de partículas propuesto en estudio [4]

El algoritmo del filtro de partículas también se ha apoyado en la estimación de direcciones (RFID), donde se utiliza un modelo de movimiento basado en Gauss (véase Figura 3), pudiendo así obtener el mapeo y localización del AGV en una zona determinada. En este caso, a medida que se aumenta el número de partículas, se obtiene mayor fiabilidad, pero añade coste computacional. Además, la antena del RFID está limitada debido al ancho de banda [4].

Igualmente, fundamentarlo en un enfoque de aprendizaje por refuerzo con el modelo de Markov resulta útil pese a recaer en la necesidad de incluir un nuevo filtro que ayude a reducir fallos, debido a que es un algoritmo de alto nivel. En consecuencia del uso del modelo J48 que plantean los autores, existen sensibilidades a variaciones de instancia, que hacen tener incertidumbres en zonas cercanas a bordes, creando nuevamente una zona acotada [5]. Combinar el filtro de partículas con un método para evitar obstáculos como un histograma de campo vectorial (VFH) es una idea razonable, aunque únicamente se ha llevado a cabo en simulación [6], por lo que la fiabilidad en un entorno real es discutible.

Siguiendo con la idea del uso del filtro de partículas, diferentes autores han planteado dicho concepto, pero proponiendo un filtro de optimización para la forma de las celdas del SLAM. Adicionalmente, toman como referencia la localización global. Se genera una forma de enjambre con subceldas que hacen que se puedan estimar localizaciones de larga distancia. Nuevamente se genera la problemática de la necesidad de obtener más datos para conseguir más fiabilidad, implicando coste. También se observa que a más distancia la probabilidad de éxito disminuye. Además, se añade dificultad de distinción en entornos con apariencia similar [7]. En la siguiente imagen se muestra el mapa de ocupación que se obtiene con este planteamiento, pudiendo hacer una identificación de la zona.

Figura 4 Mapa de ocupación del entorno [7]

El algoritmo de punto iterativo más cercano o ICP, minimiza la diferencia entre dos "nubes" de puntos, fijando una nube como referencia y buscando la trasformación de los demás conjuntos en la mejor similitud a esta. Para ello, se vale de matrices de rotación y traslación, buscando la "nube" que minimice una función respecto a dicha referencia. En cuanto a la aplicación de este, se vuelve a abordar la problemática de que la memoria no es ilimitada. Por lo tanto, la reducción del esfuerzo del cálculo puede resultar crucial.

Mediante una arquitectura wP-ICP que reduce iteraciones, unida a sensores LiDAR, se ha analizado el procesamiento de la nube para extraer esquinas y clasificar los puntos de esta. Se ha visto que se consigue la reducción del esfuerzo de cálculo, al lograr que se realicen menos iteraciones y, por lo tanto, se gestiona en menor tiempo real. La parte negativa es que los errores de estimación aumentan (véase Figura 5), teniendo la necesidad de utilizar interpoladores para tratar de solventarlo [8].

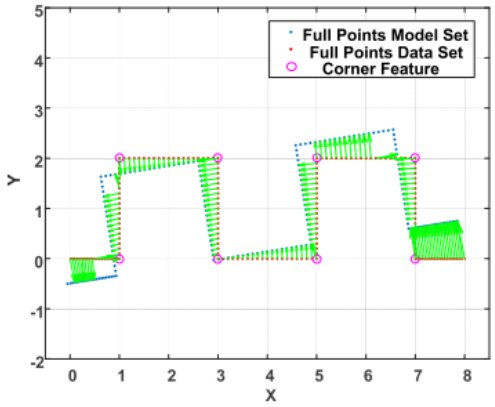


Figura 5 Registros de puntos incorrectos causador por algoritmo del estudio [8]

Ya que hacer una interpolación no es la mejor solución en cuanto a precisión, cabe la posibilidad de generar más conjuntos de puntos 3D, fijándose en la geometría del entorno. Las características de las diferentes formas de la zona ayudan, ya que se establecen como puntos de referencia, mejorando el resultado. Sin embargo, se perciben desvíos de ubicación en zonas con menos densidad de "nube". Además, la generación de más puntos, afecta al ruido de los objetos que pueda haber, sobre todo en sus bordes y esquinas, creando distorsiones [9].

En estos estudios se puede percibir la importancia de la elección de la sensórica que se implementa sobre los vehículos autónomos, que dependerá en gran parte de la superficie en la que se va a hacer uso de este. El empleo de sensores RGB resulta una opción económica. Se procesan imágenes, comparando fotogramas mediante bucles y se extraen características con valores de profundidad. Se generan 100 fotogramas en los que se busca la similitud, pudiendo así estimar el error. En la Figura 6 se puede apreciar el cambio de que el SLAM-ICP se calcula con puntos en un plano, en lugar de con posiciones espaciales donde interfiere el ruido. En este caso, se ha visto la importancia de que las imágenes sean a color, ya al hacerlo en escala de grises se generan indistinciones debido a la luz y paredes con tonalidades similares [10].

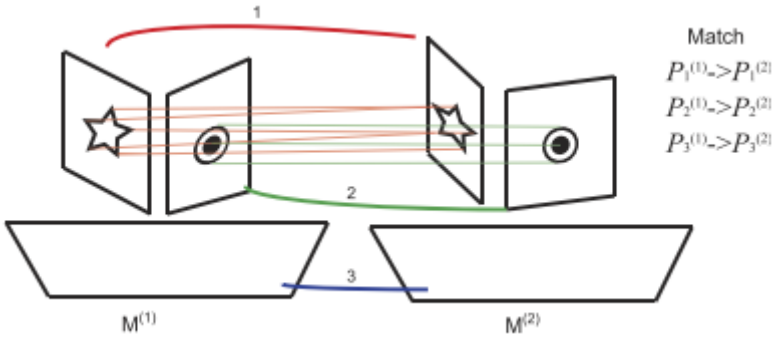


Figura 6 Planos extraídos de la "nube" de puntos [10]

Continuando con el procesamiento en 2D, autores plantean ponderar el algoritmo ICP con ayuda de LiDAR. Manteniéndose en un rango y conociendo la velocidad del AGV además del rumbo, se consigue el mapeo del entorno. Se conoce que los sensores LiDAR no necesitan infraestructura externa espacial, así que el SLAM se construye basándose en planos entre pares de puntos, dependiendo del peso de estos [11].

Los vehículos autónomos ofrecen múltiples usos en diferentes entornos, de modo que, es un punto considerable en el estudio de la localización. Se ha analizado una variación entre zonas exteriores o interiores, donde se combina INS y GPS para las zonas descubiertas y LiDAR en el resto de ellas. Se transita entre dos modos de operación con los sensores, además de unir técnicas de posicionamiento absoluto y relativo, dependiendo de donde se encuentre el robot móvil. Pese a esto, no es capaz de mapear zonas con desnivel [12]. Dicha problemática se ha visto resuelta usando datos abiertos de conjuntos KITTI, donde se propone un algoritmo ICP de búsqueda plano-tierra [13].

Derivar el algoritmo ICP con todos los sensores necesarios y construyendo mapas de puntos aleatorios distribuidos en β, hace que el algoritmo sea aplicable para el SLAM independientemente del sensor instalado en el AGV. El ruido generado por la sensórica se determina empíricamente, de manera que se tienen medidas para cuantificar la incertidumbre, siendo esta una representación probabilística de ocupación de celdas. El no depender del tipo de sensor hace que aumenten los requisitos computacionales y la memoria, pero la fiabilidad continúa recayendo en la elección de este [14].

3.2 **Problemática de la navegación**

La habilidad para que un vehículo autónomo pueda navegar en entornos no estructurados es una característica crucial de estos. Esta aptitud, suele verse limitada debido a la propia capacidad del software de navegación en uso, de ahí que sea un tema ampliamente investigado. La navegación local o también denominada "path following", se basa en las lecturas inmediatas de los sensores del AGV. De esta manera, se consigue gestionar problemas de cambios imprevistos en el mapa, dotándole de la aptitud necesaria para tomar decisiones a corto plazo y así evitar obstáculos, mientras se sigue la trayectoria global lo más precisamente posible. Además, se suelen consideran las propiedades dinámicas del robot móvil.

La prevención de colisiones, por lo tanto, es una de las cuestiones clave para el funcionamiento adecuado de los sistemas de los AGV. Todos estos robots móviles cuentan con algún medio para lograr este objetivo, que van desde algoritmos más simples que detectan el obstáculo y detienen al robot móvil, hasta algoritmos que hacen que el vehículo autónomo se desvíe para evitar chocarse. Por consiguiente, hay que decidir la manera en la que se va a hacer el "path following" y la velocidad a la que se va a recorrer la trayectoria, entre otros factores.

Uno de los métodos más recurrentes a la hora de hacer que un vehículo autónomo siga una ruta es el enfoque de ventana dinámica o DWA. Este algoritmo depende de la función de coste implementada y de los parámetros de configuración de esta, donde, una elección óptima dependiendo del entorno, puede resultar crucial. Por lo general, estos parámetros se eligen dinámicamente basándose en datos.

Realizar una hibridación del DWA con una técnica de optimización basada en enseñanza-aprendizaje (TLBO) hace posible detectar la ubicación tanto de obstáculos, como del punto objetivo que el robot debe alcanzar, como se aprecia en la Figura 7. Para ello, se proporcionan parámetros al DWA, que decide la velocidad óptima, y en relación a ella, el TLBO estima el ángulo de giro óptimo. De este modo, el AGV se desplaza hacia el objetivo final sin parar y con la capacidad de eludir objetos que se encuentren en la trayectoria. Además, el algoritmo es capaz de llevar a cabo el recorrido en una duración óptima. El análisis se ha efectuado con obstáculos estáticos y aleatorios, donde se vio que no es necesario ajustar los parámetros del TLBO, ya que esto mejora la eficacia del enfoque de ventana dinámica. Pese a esto, se ha comprobado que la optimización es mejor si el vehículo autónomo no tiene una gran desviación respecto al recorrido, no mejorando mucho respecto a otras técnicas [15].

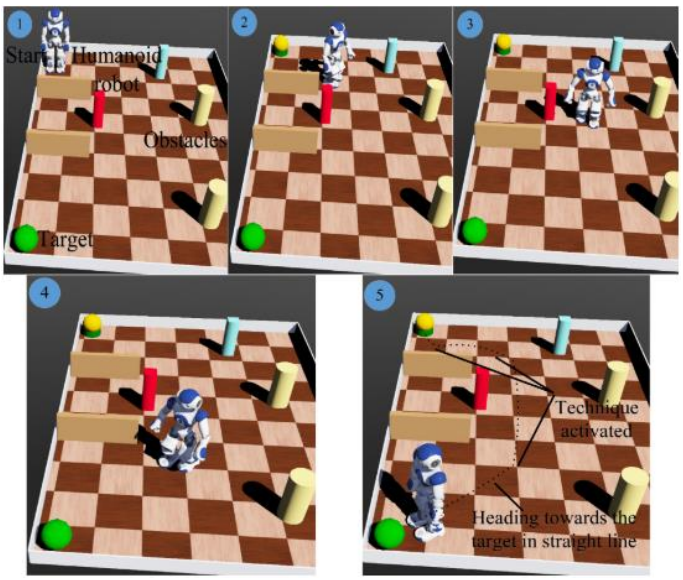


Figura 7 Simulación de algoritmo DWA+TLBO en terreno estático [15]

Continuando con la hibridación de técnicas, es posible combinar métodos de planificación de movimiento y evitación de obstáculos en tiempo real. El primero tiene base en el SLAM, mientras que el segundo gestiona la información de los sensores del robot móvil. Esto permite tener una alta velocidad, además de fiabilidad en entonos estrechos, pero el algoritmo resulta problemático al toparse con obstrucciones de la ruta [16]. Unir el algoritmo Dijkstra para planificar la ruta global y el DWA para la ruta local, hace que sea posible alcanzar la posición designada, valiéndose de la información proporcionada por un sistema SLAM. De esta manera, el vehículo autónomo es capaz de sortear obstáculos, pero se ve limitado, ya que la colocación de estos requiere de un valor mínimo de distancia. Esto se puede apreciar en la Figura 8, donde se ve que el AGV hace giros amplios para sortear los obstáculos. Además, se han detectado problemas con objetos de forma y tamaño no convencional [17].

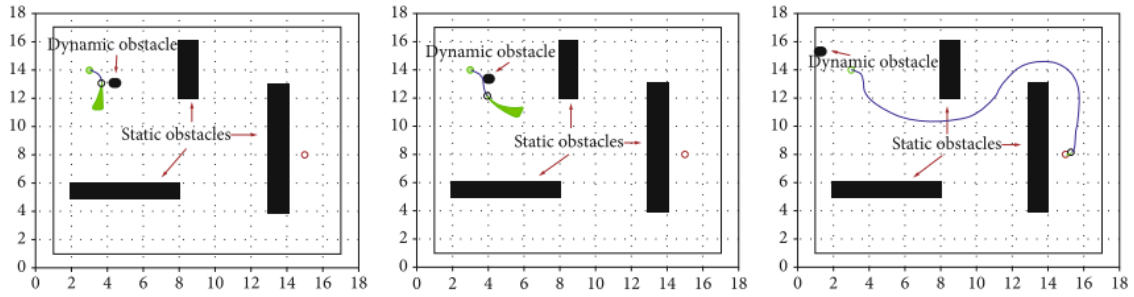


Figura 8 Resultados de simulación de navegación [17]

Otra opción de fusión de técnicas estudiada es la del uso de una red neuronal convolucional profunda, para hacer frente a la elección óptima de los parámetros de la función de coste del DWA. Se combina, por lo tanto, el aprendizaje basado en datos con el modelo dinámico del robot móvil. De esta manera, se predicen dinámicamente los parámetros necesarios, considerando las lecturas del sensor del AGV. La red se entrena con un algoritmo de aprendizaje por refuerzo, con el método de optimización de política proximal (PPO), y tras esto, se hace que la función de coste del DWA varíe en cada iteración. El sensor láser escanea el entorno y la posición del próximo objetivo, generando los pesos de la función de coste. Por lo tanto, se consigue una adaptación al entorno, además de garantizar el objetivo del "path following" (véase Figura 9), pero no supera al algoritmo DWA convencional con una buena combinación de parámetros [18].

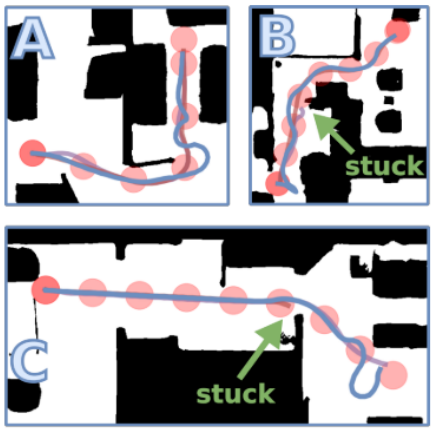


Figura 9 Navegación y detección de atascos con red neuronal y DWA [18]

Existen también vehículos autónomos con accionamiento sincronizado, que tienen que hacer frente a restricciones impuestas por las velocidades y aceleraciones. Estudios proponen que, mediante el uso de sensores infrarrojos y cámaras, se calcule la ubicación de la meta, la velocidad de avance y la distancia hasta el siguiente obstáculo. De este modo y considerando la inercia, además de las velocidades admisibles dentro de la ventana dinámica, se hace que el robot móvil

siga la trayectoria con precisión. Para ello, periódicamente, se calcula el siguiente comando de dirección, que se indica mediante un número finito de segmentos circulares [19]. El uso de este tipo de AGV es cuestionable, considerando que en espacios sin obstáculos la velocidad siempre estará limitada por el hardware.

El método de campo potencial, también conocido como método de fuerza de campo virtual (VFF), es otro procedimiento clásico que se ha implementado sobre los vehículos autónomos. Se usan las fuerzas de campo virtual gravitacional y repulsivas para conseguir la relación entre el robot móvil, los obstáculos del entorno y el objetivo a alcanzar. Las fuerzas, por lo general, disminuyen con la distancia, de forma que es posible detectar las posiciones de los objetos de la zona. Además, cuenta con una expresión matemática muy sencilla que es la atracción principal del VFF y el motivo por el cual se lleva a estudio en este campo.

Se plantea un sistema diseñado en dos capas, dividiendo por una parte la problemática de la toma de decisiones para el cálculo de la trayectoria y por otra, el control para el seguimiento de la ruta, donde el VFF actúa y es capaz de detectar objetos, como se ve en la Figura 10. Este estudio tiene en cuenta la velocidad y dirección de los objetos, ya que tantea que no sean estáticos. En consecuencia, también se combina con un método de seguimiento de ruta con un control predictivo MPC [20]. Del mismo modo, es posible la combinación con un método de campo de flujo potencial (PFF) para construir un medio viable para la navegación. El VFF, al igual que en otros análisis, gestiona el peligro de colisión, pero en este caso, es algoritmo PFF el que se encarga de proporcional los comandos de ángulo de rumbo en tiempo real, derivando el campo del vector de velocidad. Sin embargo, existe la necesidad de girar 90º al toparse con un obstáculo para conseguir una distancia mínima que haga evitar colisionar, además de que esto resulta problemático cerca de paredes [21].

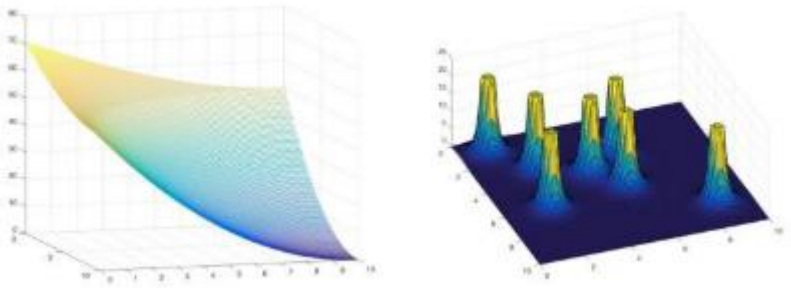


Figura 10 Fuerzas repulsivas y gravitacionales de 7 obstáculos [20]

Otro de los algoritmos utilizados para resolver la problemática de la navegación es el histograma de campo vectorial (VFH), donde, basándose en valores de certeza, se genera un histograma. Dichos valores están relacionados con la velocidad del AGV y, tomándolos como referencia, se crean los picos del histograma a lo largo de la dirección del obstáculo. Autores utilizan la cuadrícula del histograma cartesiano bidimensional como modelo para realizar un mapa del entorno. Este se actualiza continuamente con información de la zona, valiéndose de sensores. Con el fin de evitar un alto consumo de recursos, se emplea un proceso de reducción de datos en dos etapas. La primera es reducir la cuadrícula del histograma a un histograma polar unidimensional, construido alrededor de la ubicación momentánea (véase Figura 11). De este modo, cada sector contiene un valor que representa la densidad de obstáculos polares en esa dirección. La segunda etapa es la que selecciona el sector más adecuado entre todos, el de menor densidad, y hace que la dirección del robot se alinee con la de dicho sector. De esta manera se consiguen calcular los comandos de control. Sin embargo, el vehículo autónomo tiene la necesidad de pararse delante de los obstáculos en caso de toparse con ellos, y es un proceso lento en la ejecución [22].

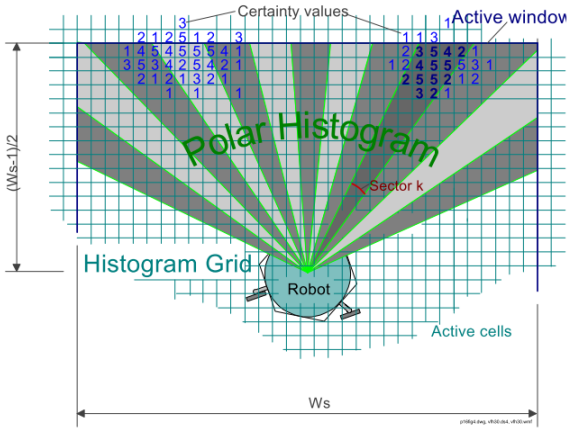


Figura 11 Mapeo de ocupación de histograma polar unidimensional [22]

Existe una modificación del algoritmo VFH llamada histograma de campo de transitabilidad (TFH), donde la diferencia respecto al primero recae en su independencia de la velocidad momentánea del vehículo autónomo. De esta manera se consigue evitar, por ejemplo, que obstáculos distantes creen valores de certeza muy grandes debido a la pequeña velocidad del robot móvil. Se hace uso de un telemetro láser bidimensional (LRF) para cartografiar el terreno, y así poder transformar la información recogida en un mapa de transitabilidad, como se aprecia en la Figura 12. Es posible reconocer pendientes y rugosidades del entorno en dicho mapa, valiéndose del ajuste del plano a mínimos cuadrados. Tras esto, se calcula el denominado índice de transitabilidad polar (PTI), que pondera la dificultad de viajar a lo largo de cualquier dirección. Se genera un nuevo histograma con los datos PTI y en base a este, se deciden la velocidad y el rumbo. Recordemos que dicho índice está relacionado con la elevación de los objetos, pero no con la velocidad. Con este algoritmo se corrigen limitaciones del VFH, pero no se logra alcanzar la ubicación objetivo [23].

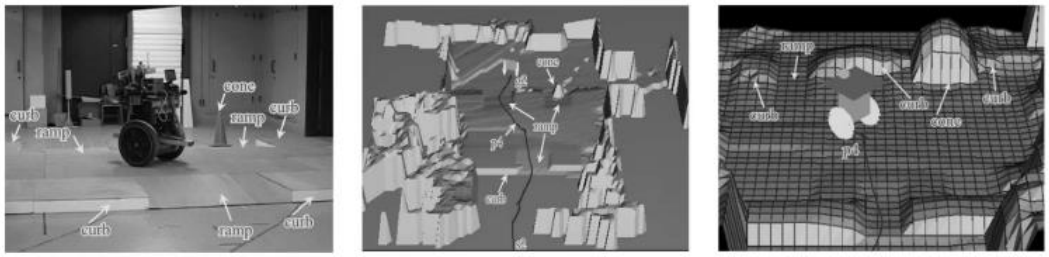


Figura 12 Construcción de mapa de transitabilidad [23]

Desde otro punto de vista, se idea el uso del control lateral para la resolución del "path following", método que se ha introducido con fuerza en la robótica móvil. En la mayoría de los casos, el diseño de este procedimiento se basa en la identificación de un punto de referencia en la trayectoria, el más cercano al AGV. Con ese dato, es posible obtener la información del estado en el que se encuentra el vehículo autónomo.

Uno de los algoritmos que cumple esta función es el de Stanley, donde se plantea un modelo de predicción discreta que anticipa estados futuros del AGV. Autores han logrado dicho control basado en predicciones de estados, con un controlador Stanley predictivo. Haciendo uso de sensores GPS, IMU, LiDAR, cámaras estéreo y codificadores de ruedas, se consigue la posición del vehículo autónomo. Con esa información, se puede gestionar un planificador local de banda elástica en el tiempo (TEB) y hacer una navegación libre de colisiones. Hay que tener en cuenta que el enfoque de un control predictivo genera un tiempo de cómputo extenso, además de que esta modificación no disminuye suficientemente el error en comparación con el algoritmo Stanley convencional [24].

3.3 **Problemática de la planificación de la trayectoria**

La planificación de rutas se encuentra entre las áreas más complejas de la investigación de la robótica móvil en los últimos años, con creciente popularidad, tal y como se ha podido observar en numerosos artículos de investigación accesibles. El "path planning" es una de las tecnologías clave que facilita la maniobrabilidad autónoma de los vehículos autónomos, buscando obtener la ruta óptima sin colisiones desde un punto de partida hasta un punto objetivo.

A diferencia de en el "path following", aquí se habla de navegación global, donde disponiendo de un mapa del entorno, se produce un plan para alcanzar la posición objetivo teniendo en cuenta la geometría del robot. Las técnicas desarrolladas hasta ahora suelen presentar limitaciones relacionadas con la planificación a largo plazo, de manera que el objetivo actual se centra en optimizar parámetros, tales como la distancia recorrida, el tiempo de viaje y el coste computacional. A continuación, se discuten algunas técnicas.

El método más convencional para estimar una trayectoria es el uso de la optimización por enjambre de partículas (Particle Swarm Optimization). Este procedimiento optimiza el problema al pretender mejorar una solución candidata respecto a una medida de calidad dada. La mejora se realiza iterativamente moviendo partículas en el espacio de búsqueda de soluciones. Dicho movimiento está influenciado por la propia posición local, pero también por otras posiciones del espacio que se actualizan a medida que otras partículas encuentran mejores posiciones. En esas posiciones es donde se evalúa la función de coste.

Mejorando los pesos de inercia con una variación lineal, además de los factores de aceleración y localización, se consigue evitar que el algoritmo caiga en un mínimo local y, en consecuencia, el aumento la velocidad de convergencia. Así mismo, variando la aptitud para medir la diversidad de las partículas, se ha logrado superar deficiencias prematuras. Se puede ver como una versión parcial del algoritmo PSO, ya que se usa la varianza de aptitud como medida de variación de partículas y una distribución Gaussiana extendida para aumentar la diversidad de estas. Tras su aplicación, se debe de realizar un suavizado a la propia planificación de la ruta. Denominado ILPSO, resuelve el problema del tiempo computacional y consigue lograr el valor óptimo de trayectoria en cualquier entorno(véase Figura 13) [25]. Comparando este método con el A* la única mejora notable es la de la longitud de la ruta, pero solamente en entornos simples y 2D.

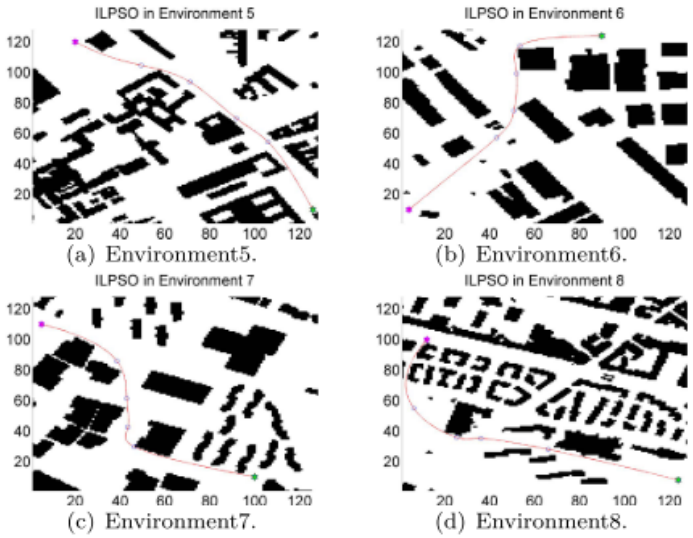


Figura 13 Aplicación de ILPSO en mapas reales [25]

Autores han analizado el uso del algoritmo A* como método de modelado de mapas para conseguir la planificación de rutas. Es posible su uso unido al algoritmo de triangulación Delaunay, que procesa obstáculos complejos y es capaz de generar puntos del diagrama de Voronoi como nodos prioritarios para la búsqueda de la trayectoria, tal y como se ve en la siguiente imagen. Se genera una cuadrícula de la que se extraen los bordes de los obstáculos, validando crear estrategias de evitación de obstáculos. En este caso, el algoritmo A* se usa para realizar un filtrado de nodos y seleccionar los que pertenecen a la ruta más regular, siendo esta la prioritaria. De este modo se ha diseñado una fusión dinámica de algoritmos (DFPA), que es capaz de reducir la longitud de la ruta planificada. Al filtrar nodos, se consigue evitar cálculos repetidos, generando un mejor proceso de búsqueda y una simplificación de la complejidad. Sin embargo, dicha complejidad es alta para estimaciones de trayectorias entornos simples [26].

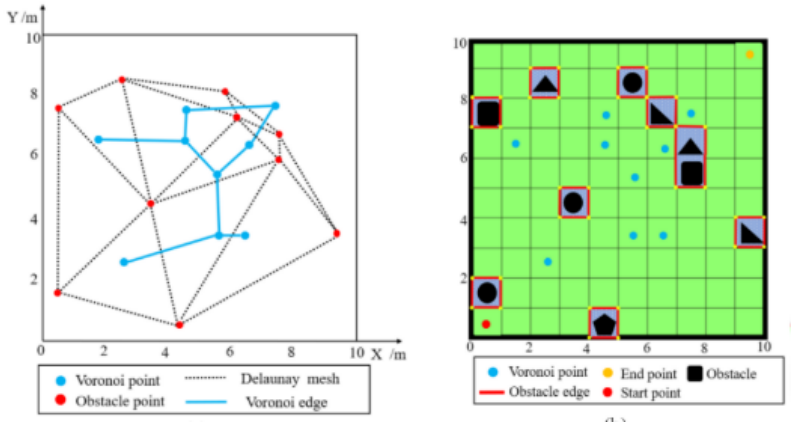


Figura 14 Triangulación, putos Voronoi y detección de obstáculos en cuadricula [26]

Considerar solo la distancia óptima hacia el punto objetivo no suele ser suficiente en muchos casos. En cambio, contemplar el tiempo más corto hasta ese punto suele resultar ser el objetivo ideal, pero no es fácil de lograr, ya que requiere de la introducción de la evaluación del tiempo óptimo factible en la planificación de la ruta. Con este planteamiento se ha logrado reducir el costo de software y la simplificación del propio diseño hardware, a través de la implementación de un algoritmo A* jerárquico mejorado. De esa manera, es posible planificar la ruta óptima y construir una función de evaluación variable en el tiempo, teniendo en cuenta la posición real del objetivo. El algoritmo A* puede encontrar rápidamente las rutas más óptimas con diferentes tiempos, dependiendo de los atributos que se le establezcan [27]. Así se puede funcionar en ambos modos de optimización de trayectoria, ya que se puede calcular la ruta más corta y la más rápida.

Otro enfoque es el uso del algoritmo A* junto con el algoritmo RRT (Rapidly-Exploring Random Tree), presentando muy buen rendimiento. Sin embargo, no se ha logrado la seguridad necesaria en la ruta, ya que tiene una alta tasa de colisiones, sobre todo en entornos dinámicos [28]. En lugar de unir el RRT a otros algoritmos, se han diseñado modificaciones como el KBB-RRT*, estructurado sobre la base del RRT bidireccional y logrando así un nuevo esquema de planificación de rutas. El estudio se ha realizado sobre un automóvil de accionamiento diferencial que incorpora restricciones cinemáticas para evitar el crecimiento innecesario del “tree”, encontrando la ruta factible rápidamente. Esto se demuestra en diferentes ensayos como se puede apreciar en la Figura 15. La estrategia de “poda de ramas” del RRT ayuda a disminuir el coste computacional, además de estimar trayectorias de longitud reducida, aunque no asegura que sean las óptimas. Así mismo, el rendimiento también es bajo dependiendo de la región en la que se sitúe el vehículo autónomo [29].

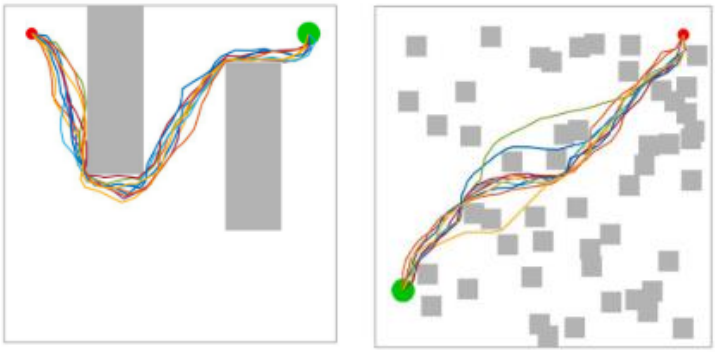


Figura 15 Planificación de ruta con KBB-RTT [29]*

Otro diseño de mejora es el algoritmo llamado Smoothly RRT (S-RRT). La estrategia de optimización de la ruta se plantea basándose en la restricción de curvatura máxima, de forma que se genera una ruta ejecutable suave, como se ve en la siguiente imagen. Además, con el fin de evitar colisiones, se extrae la información de obstáculos estáticos y dinámicos a través de un sensor Kinect RGB-D. Se consigue diferenciar el tipo de obstáculo a través de un preprocesamiento de la nube de puntos, al que se le proporcionan valores de referencia teóricos. Se ha visto una mejora en la velocidad de exploración y en la eficiencia si se apunta a un nodo direccional, a través de simulaciones en MATLAB, pero no gestiona bien obstáculos dinámicos que en un determinado momento se paran para permanecer estáticos [30].

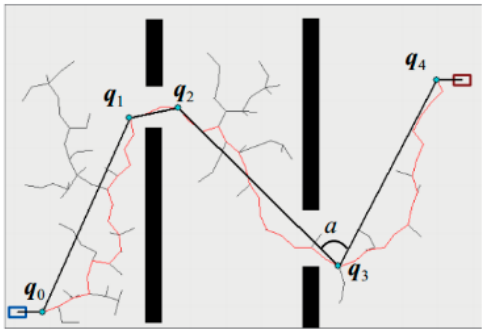


Figura 16 Trayectorias generadas por S-RRT [30]

El Wavefront es otro de los algoritmos recurrentes a la hora de planificar la ruta, usándolo para encontrar el punto de frente más cercano en un tiempo muy reducido. Gracias a esto se puede seleccionar de entre todos los puntos fronterizos detectados, el más óptimo, sujeto a unos requisitos. Tras esto, el vehículo autónomo comienza a moverse hacia dicho punto, con la ruta planificada por el algoritmo de frente de onda (véase Figura 17). El algoritmo consta de dos procesos: primero se debe alcanzar el punto de partida para poder, después, planificar la ruta. Cuando el algoritmo no sea capaz de encontrar más puntos se considera que ha completado la tarea. Se ha visto que el Wavefront tiene dificultad para fijar puntos fronterizos en entornos complejos, pero es capaz de encontrar rutas adicionales, implicando más tiempo de exploración y cálculo [31].

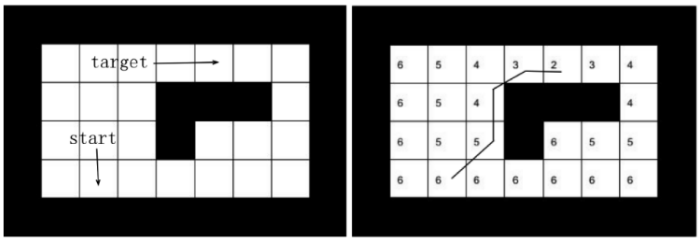


Figura 17 Cálculo de frente de onda y trayectoria [31]

Se ha analizado de igual modo, el uso del algoritmo de frente de onda generalizado para calcular trayectorias. Se combinan múltiples conjuntos de puntos de destino, costos de cuadrícula multinivel y expansiones geométricas alrededor de los obstáculos. Con esta información se consigue poder optimizar la ruta posteriormente, siendo suave y segura. Además, se cumple con las condiciones cinemáticas requeridas, asociadas a la maniobrabilidad real de los robots móviles. En una primera etapa, el algoritmo usa la cuadrícula de costos multinivel para inicializar el entramado de la zona y calcular el costo total. De esta manera, es posible generar una función de expansión de obstáculos logarítmica para evitar colisiones y asegurar que la trayectoria se ciñe a la cinemática. En caso de que el punto objetivo este ocupado, el frente de onda generalizado es capaz de establecer puntos de destino alternativos. La segunda etapa evita generar rutas redundantes y optimiza los puntos generados, suavizando la trayectoria en base a aproximaciones de curvas de Bézier. Se ha visto que se reduce la longitud de la trayectoria en comparación a la obtenida con la fusión del RTT+A*. Pese a esto, tiene limitaciones, ya que es posible que se generen una gran cantidad de puntos en secuencia, aumentando el tiempo de procesamiento [32].

Una forma menos convencional pero muy llamativa de abordar el "path planning" es el uso de redes neuronales. Se emplea un algoritmo basado en el Q-learning con aprendizaje por refuerzo, capaz de coger información del entorno por medio de sensores Sonar. Junto con la $\overrightarrow{pose}$ del robot móvil y la ubicación de los puntos objetivo, se construye un modelo del entorno y con él, un espacio de estados con funciones de recompensa discretas. Se aplica el aprendizaje por refuerzo a dicho espacio de estados, apreciando su eficiencia en cuanto a la convergencia [33]. En la Figura 18 se puede ver como el avance del entrenamiento de la red neuronal hace que la trayectoria calculada se vaya optimizando. Se ha estudiado también el uso de un modo de entrenamiento incremental para el "deep reinforcement learning", donde, primero se evalúan los algoritmos de búsqueda gráficos y de aprendizaje por refuerzo en 2D, para después diseñar el propio algoritmo de aprendizaje profundo. Este incluye estados de observación y funciones de recompensa, además de la propia estructura de la red y la optimización de parámetros. Ya que el análisis en 3D consume más tiempo, no se entrena con este, consiguiendo aliviar la convergencia [34].

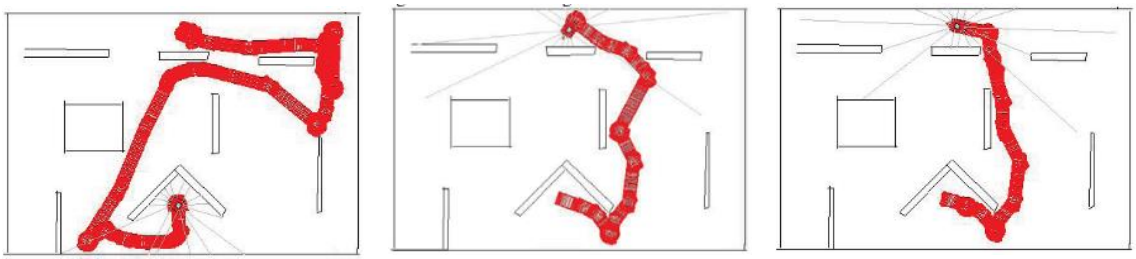


Figura 18 Cálculo de trayectoria con avance de entrenamiento de red [33]

Valiéndose de la red Resnet-50, se ha creado un algoritmo de planificación de rutas basado en el aprendizaje por refuerzo profundo. Autores proponen el uso de la teoría del espectro cíclico para realizar la identificación de la señal de comunicación del radar, que está basada en el aprendizaje de migración de Resnet-50. Con el empleo de dicha señal, el modelo cinemático del vehículo y la distancia hasta el punto objetivo, se plantean dos criterios de recompensa. Por un lado, la tasa de reconocimiento de señales y por otro, la combinación de la red neuronal de aprendizaje profundo con criterios de aprendizaje por refuerzo. Tras esto, la red pasa por el aprendizaje de migración de Resnet-50, que logra mejorar la tasa de reconocimiento de señales del radar. De esta manera, se entrenan parámetros del Deep Q-Network para la planificación autónoma de la trayectoria, haciendo una generación adaptativa de esta y resolviendo la problemática planteada. Este método ha dado resultados con un 90% de precisión, pero no es adecuado bajo condiciones de 0dB [35].

Otra opción es plantear la navegación del AGV mediante la implementación de una CNN que segmenta una imagen para determinar la zona navegable y calcular los comandos de dirección y velocidad, además de la trayectoria. El estudio utiliza una red neuronal residual que participa en el aprendizaje de transferencia de la red Resnet-18. De esta manera, Resnet-18 consigue realizar una segmentación semántica para la navegación del vehículo autónomo. Se distingue entre piso, espacio libre y paredes, consideradas obstáculos, de forma que el resto de ecuaciones puedan usar esta información. Después de hacer el aprendizaje, se usa la máscara de "piso" obtenida de la segmentación para implementar la navegación y los cálculos de movimiento, basados en cuanto difiere la trayectoria estimada de la línea central vertical. Como dirección, se toma el punto más alto de la línea vertical y se hace que el robot móvil se dirija hacia él. Es un planteamiento llamativo pese a que el tiempo de ejecución no sea suficiente como para proporcionar seguridad en situaciones repentinas [36]. En la siguiente imagen se ve la segmentación y la extracción de la zona navegable o máscara "piso", con su trayectoria calculada.

Figura 19 Cálculo de trayectoria en imagen binaria [36]

Centrándose en la visión por computador, se pueden encontrar redes creadas específicamente para el propósito de la obtención de la trayectoria, como PilotNet. Esta red es capaz de realizar una detección de carriles a través de cámaras, además de aplicar algoritmos de seguimiento de vehículos para conocer la dirección de la ruta (véase Figura 20). Se efectua una búsqueda de carriles basada en imágenes mediante el uso de un algoritmo de aprendizaje, mientras que la detección de objetos y vehículos se hace por medio de una red neuronal convolucional rápida basada en regiones (Faster RCNN). La arquitectura de esta última red se establece en las bases de la red neuronal convolucional PilotNet de NVIDIA. Se ha visto la necesidad de aumentar los datos para crear diversidad en el conjunto de estos, pero implica más tiempo de cómputo. Además, solo se han dado resultados en vías de un solo carril y no es independiente del entono ni de las condiciones ambientales [37].

Figura 20 Detecciones hechas por PilotNet [38]

Otra red propia del propósito que usa la visión por computador es LaneNet, que es capaz de detectar la ubicación y apariencia de las marcas que hay en los carriles (véase Figura 21), siendo este un requisito previo para crear mapas y rutas. Se usa una red neuronal simétrica completamente convolucional mejorada por la transformada ondícula. De este modo y usando imágenes aéreas que abarcan áreas amplias, se hace una segmentación semántica precisa, localizando los píxeles de interés y sin utilizar información de terceros. Se aprecia que el uso de capas para el submuestreo produce un alto rendimiento, pero provoca la pérdida de datos en la segmentación semántica que se traduce en perdida de dicho rendimiento. Nuevamente, aparece la limitacion de áreas con sombras y marcas de carriles descoloridas [39].

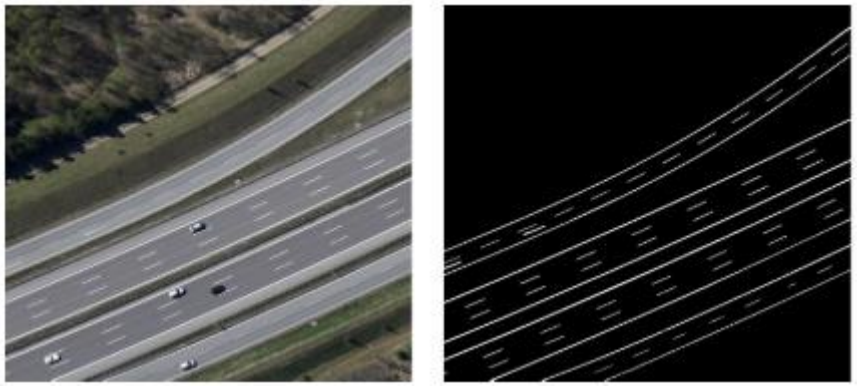


Figura 21 Resultados obtenidos con LaneNet [39]

3.4 **Decisión en base al estado del arte**

Tras realizar el estudio de los antecedentes de las diferentes técnicas aplicables a los AGV, se obtiene una idea del rumbo a seguir en el desarrollo del algoritmo de localización y navegación.

Respecto al "Path following"; se considera que el algoritmo Stanley da muy buenos resultados. Siendo así, se partirá de esa base para desarrollar un algoritmo de navegación que realice el control lateral del vehículo autónomo. A dicho algoritmo, también se le añadirá una política de control longitudinal con la intención de regular la velocidad.

En cuanto al cálculo de la trayectoria, se decide hacer uso de la visión artificial para realizar un reconocimiento del entorno del robot móvil, y en base a él, estimar una trayectoria. Se tomará como idea la segmentación semántica en el diseño del algoritmo, para así extraer la zona navegable y partir de esa información.

Haciendo uso de la trayectoria generada y de la $\overrightarrow{pose}$ del AGV, será posible valorar la localización del vehículo, tomando como referencia la ruta. De este modo se plantea obtener la distancia entre el AGV y la trayectoria, al igual que la dirección del vehículo autónomo.

4 Desarrollo de algoritmo de localización y navegación

En el apartado *3.4 Decisión en base al estado del arte*, se ha descrito la solución de diseño que se propone en este trabajo. Se pueden distinguir dos bloques principales para realizar el desarrollo. Por un lado, se plantea el diseño del algoritmo de "path following", que incluirá el control lateral y el control longitudinal. Por otro, usando la visión por computador, se resolverá la problemática de la planificación de la trayectoria y la localización. Una vez estén completos ambos bloques, se fusionarán para lograr el objetivo del control autónomo del AGV.

4.1 Diseño del algoritmo de navegación

Se presenta el problema del seguimiento de una trayectoria como la unión de los conceptos de control lateral y control longitudinal. El primero se resuelve basándose en el algoritmo Stanley como se verá a continuación. Para el segundo, se diseña una política con la información que se obtiene del primer control.

4.1.1 Control lateral del AGV

La resolución del algoritmo de navegación parte del diseño de un algoritmo que realice un control lateral, de forma que el vehículo autónomo sea capaz situarse encima de una ruta y desplazarse sobre ella. Este control se centra en conseguir que el AGV y la trayectoria coincidan en el ángulo y la posición. Como se ha mencionado, se partirá del algoritmo Stanley.

4.1.1.1 Fundamentos del algoritmo Stanley

Se toma un AGV de cuatro ruedas, pero se modela para tener un vehículo autónomo de dos, como se aprecia en el primer esquema de la siguiente imagen. De esta manera, la rueda trasera se encarga de controlar el parámetro de velocidad **V** y la delantera, el ángulo que marca la dirección, **δ**.

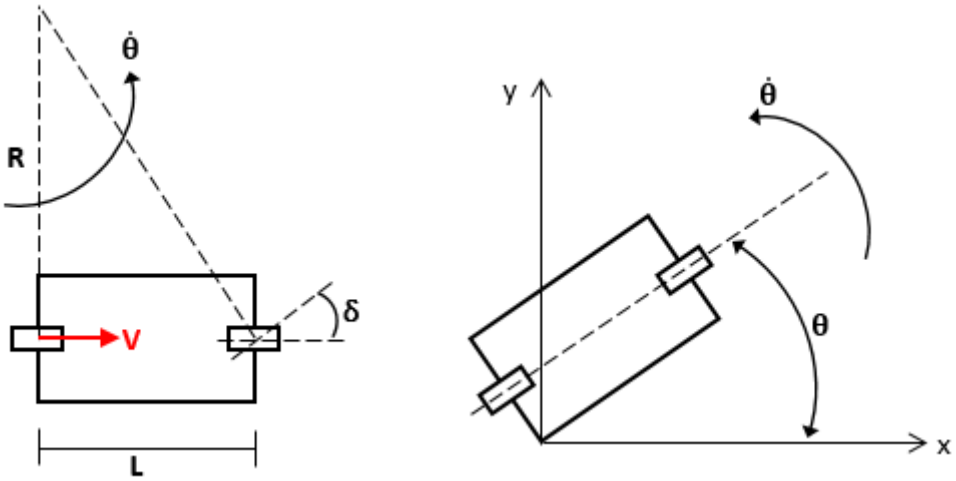


Figura 22 Esquema de velocidades y ángulos de AGV

Se conoce que los cuerpos rígidos tienen un giro instantáneo. Reparando en el primer esquema de la Figura 22, se puede asumir que se tendrá, por lo tanto, una velocidad de giro $\dot{\theta}$, que viene dada por ángulo del vehículo autónomo respecto a los ejes ***x*** e ***y*** (véase segundo esquema de figura superior). Conociendo que el parámetro **L** se refiere a la longitud del AGV y **R** al radio de giro de este, es posible plantear cuáles son los factores que actúan sobre ambas ruedas.

Viendo la figura superior, y como ya se ha mencionado, se tiene una velocidad en el punto correspondiente a la rueda trasera. Además, se puede intuir otra velocidad a la que se llamará **V'**, en la posición de la rueda delantera. Conociendo el ángulo de dirección **δ** de dicha rueda, y sabiendo cuál es el centro de giro instantáneo, es posible saber cuáles son los radios de giro de ambas ruedas, siendo estos **R** y **R'** (véase Figura 23). Se conoce que dichos radios y sus velocidades correspondientes siempre son perpendiculares, de manera que es posible asumir la siguiente relación:

$$\frac{V}{R} = \frac{V'}{R'} = \dot{\theta} \tag{1}$$

Analizando los ángulos que aparecen en el modelo simplificado del AGV, si **R'** y **V'** forman una perpendicular, además de estar girados en un valor de **δ**, se conoce que el ángulo opuesto de la rueda delantera es $\frac{\pi}{2} - \delta$, situado en la parte interior del triángulo que se forma tal y como se aprecia en la siguiente imagen. Siendo esto así y por reglas geométricas, el valor del ángulo generado entre ambos radios de giro instantáneo es también **δ**.

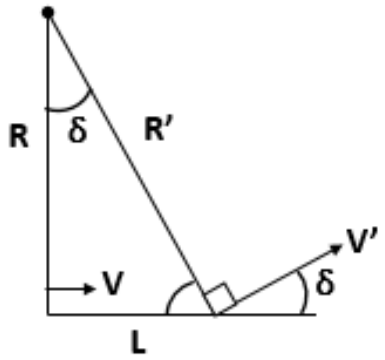


Figura 23 Esquema de parámetros que actúan sobre AGV

De este modo se consiguen conocer todos los parámetros que actúan sobre el vehículo autónomo, de forma que se puede escribir la siguiente ecuación:

$$\frac{L}{R} = tg\ \delta \rightarrow R = \frac{L}{tg\ \delta} \tag{2}$$

El valor del radio de giro instantáneo queda definido, pudiendo retomar la expresión (1). En ella, es posible sustituir el parámetro **R** para obtener la velocidad angular $\dot{\theta}$, siendo:

$$\dot{\theta} = \frac{V}{R} = \frac{V}{L} tg\delta \tag{3}$$

Esta última relación se desarrolla de manera más detallada en el modelo de vehículo de Braitenberg [40]. En este punto es posible considerar que los valores de velocidad y longitud del AGV son constantes, pudiendo establecerlos como un parámetro invariable que los referencie.

$$\dot{\theta} = K\ tg\delta \tag{4}$$

Partiendo de esta base se desarrolla el algoritmo Stanley [41]. Se considera una trayectoria aleatoria que se desea continuar y una posición del vehículo autónomo, también al azar, en referencia a los ejes ***x*** e ***y***, tal y como se ve en la Figura 24. Suponiendo que el vehículo tendrá una velocidad **V** y que estará orientado con un ángulo de dirección **θ**, es posible calcular el punto más cercano entre el AGV y la trayectoria, denominado como punto **P**.

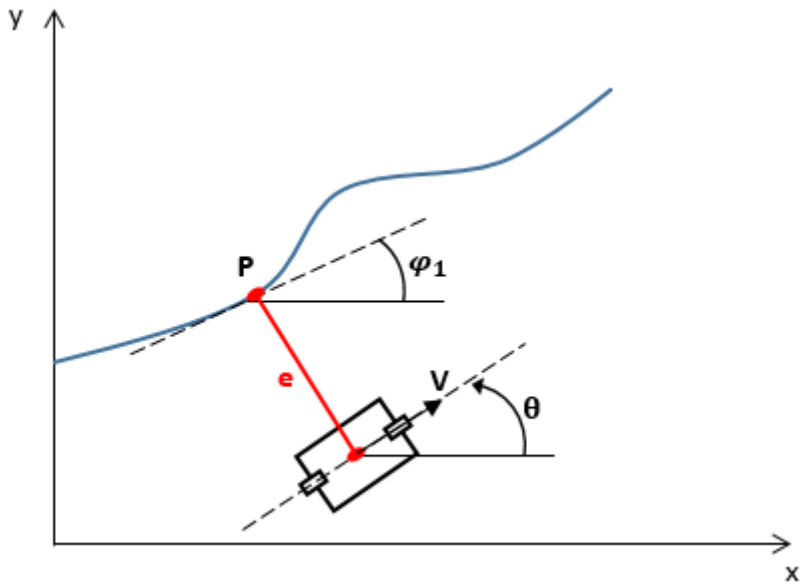


Figura 24 Planteamiento de control lateral de Stanley

Con la información de la situación del punto **P** y la posición del vehículo autónomo se obtiene el valor del error:

$$e = min\|(x,y)_{vehiculo} - (x(u), y(u))\| \qquad donde\, u \in \mathbb{R} \tag{5}$$

Es apreciable que el segundo término de la resta corresponde a un punto de la trayectoria, con notación en forma paramétrica. Así mismo, al trazar la tangente de dicho punto **P**, se logra conocer el ángulo en esa posición, $\boldsymbol{\varphi_1}$. Gracias a este valor, se obtiene el error de alineamiento, al que se referencia como **φ**. Este parámetro define la diferencia entre el ángulo de dirección del AGV y el de la trayectoria:

$$\varphi = \theta - \varphi_1 \tag{6}$$

Retomando Figura 24, se comprende que, además de tener en cuenta el error de alineamiento, el control lateral se diseña considerando que se tiene un error de posición del AGV respecto a la ruta, **e**, ya que el vehículo autónomo no está posicionado sobre la trayectoria a seguir. Con este planteamiento, el algoritmo Stanley propone que valor de ángulo de direccionamiento de rueda **δ** debe de cumplir la siguiente ecuación:

$$\delta = \varphi + arctg\left(\frac{K_c e}{V}\right) \tag{7}$$

Analizando esta relación, se ve que aparece una constante K_c, que simplemente es un parámetro del controlador. De igual modo, se puede conocer que si el error de posicionamiento es muy grande y **φ** es igual a 0, la división tenderá a infinito, teniendo un valor de arcotangente de 90º. Siendo así, el vehículo se situará perpendicular a la trayectoria hasta alcanzarla. En cambio, si el vehículo está situado encima de la trayectoria, esto es, con un valor de error de posicionamiento **e** nulo, actuará la corrección del error de alineamiento **φ**.

4.1.1.1.1 Modificación propuesta

Tomando la idea del algoritmo Stanley, en este proyecto se formula una modificación. El robot móvil del que se dispone en la escuela de ingeniería de Vitoria-Gasteiz, permite el ajuste de los parámetros de velocidad angular y velocidad lineal (véase Figura 25). Desde ese punto de vista, el control es más adecuado que en el AGV visto en la Figura 22. Además, con el fin de igualar notación, se conoce que **ω** es igual al parámetro $\dot{\boldsymbol{\theta}}$.

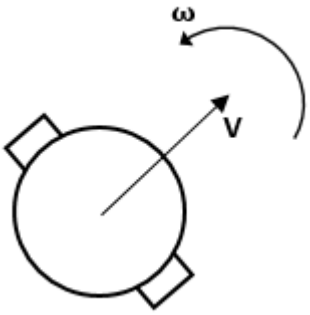


Figura 25 Modelo cinemático de AGV de la escuela de ingeniería de Vitoria-Gasteiz

Para realizar el control lateral, se parte de la suposición de que la velocidad lineal **V** es constante. Por lo tanto, en esta primera parte del algoritmo se centra en controlar la velocidad de giro $\dot{\boldsymbol{\theta}}$. De tal forma, se propone aplicar el control de Stanley de la siguiente manera:

$$\dot{\theta} = K_1 \varphi + K_2 \frac{e}{V} \tag{8}$$

Siendo así, y situando el vehículo con un valor de $\overrightarrow{pose}$ aleatorio, se le aplica al control lateral las constantes **K1** y **K2**, pudiendo analizar cómo se comporta el vehículo autónomo en función de dichos coeficientes.

Ya que la ecuación (8) resulta muy brusca, se replantea en el siguiente apartado, de manera que resulte más acorde al AGV con el que se trabaja.

4.1.1.2 Planteamiento para realizar el control lateral

Se retoma el análisis del apartado *4.1.1.1 Fundamentos del algoritmo Stanley*, basándose en la Figura 22 y la propuesta de modificación de la ecuación (8), continuando con el concepto de que la velocidad lineal sea constante.

El AGV que se utiliza para la resolución de este proyecto cuenta con dos ruedas que giran en el mismo eje, teniendo un valor de $\boldsymbol{\omega_1}$ y otro de $\boldsymbol{\omega_2}$, como se puede ver en la

Figura 26. De esta manera, la velocidad angular total es conocida, resolviéndose como:

$$\omega = K_c(\omega_2 - \omega_1) = \dot{\theta} \tag{9}$$

Se obvia la constante del controlador, centrándose únicamente en la velocidad angular. Por lo tanto, se debe plantear que dicha **ω** tiene que tener en cuenta tanto el error de posicionamiento, como el error de alineamiento **φ**:

$$\omega = \dot{\theta} = f(e, \vartheta - \varphi_{trayect.}) \tag{10}$$

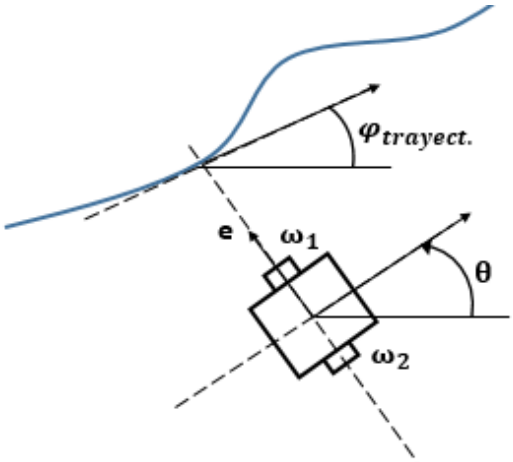


Figura 26 Esquema de control lateral propuesto

El error de alineamiento es un dato conocido, ya que es posible medirlo con el ángulo de la trayectoria y el del AGV, de manera que el diseño se centra en el error de posicionamiento **e**.

- Si **e** es muy grande, interesa poner el vehículo con una dirección perpendicular a la trayectoria para aproximarse a esta, por lo tanto:

$$\theta - \varphi_{trayect.} = \frac{\pi}{2} \tag{11}$$

- Si **e** es pequeño, no habrá desalineamiento, de forma se pretende que ambos se mantengan iguales:

$$\theta - \varphi_{trayect.} = 0 \tag{12}$$

Recordando la ecuación (6), se aprecia que estas relaciones son las que dan el valor del error **φ**, que es medible en las imágenes, tal y como se describe en el apartado *4.2 Diseño del algoritmo de planificación de trayectoria y localización*. Al graficar este análisis, se obtiene una función con la siguiente apariencia.

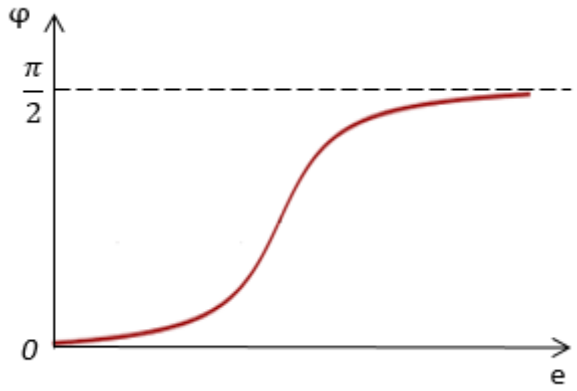


Figura 27 Relación entre error de alineamiento y error de posición

Esta forma gráfica es muy similar a la del Arcotangente, por lo que el planteamiento se valdrá de él. Además, dicha función también aparece en el algoritmo Stanley. Teniendo esto en cuenta, se propone el valor que debe de tener **ω** en función del error de posicionamiento y el de alineamiento, sabiendo que es posible derivar **θ** respecto al tiempo:

$$\frac{d\theta}{dt} = \omega = K_1\left(\varphi_{consigna} - \varphi\right) \tag{13}$$

Se ve necesaria la introducción de un parámetro $\boldsymbol{\varphi_{consigna}}$, que hace posible adaptarse al rango de la gráfica vista en la Figura 27. Su valor se define como:

$$\varphi_{consigna} = arctg(K_2\, e) \tag{14}$$

De esta manera se tienen dos funciones dependientes de las constantes $\mathbf{K_1}$ y $\mathbf{K_2}$ que habrá que determinar. Se aprecia también como la constante $\mathbf{K_1}$ es la relacionada con **φ**, mientras que $\mathbf{K_2}$ controla el valor de **e**.

4.1.1.3 Distinción del signo del error

Al implementar el diseño del control lateral, surge la dificultad de la distinción del signo del error de posicionamiento **e**, ya que no queda contemplado ni en la ecuación (13) ni en la (14). Esta problemática deriva en la necesidad de diferenciar en cuál de los lados de la trayectoria se sitúa el vehículo autónomo en todo momento. Dependiendo de ello, se deberá de direccionar con signo positivo o negativo, con el fin de realizar un seguimiento adecuado. En simulación se puede observar como la respuesta del AGV es adecuada a un lado de la trayectoria, pero no es capaz de ejecutar un buen control si traspasa la ruta (véase Figura 28).

Con objetivo de simplificar la resolución del problema, se plantea la trayectoria como una recta de la forma ***Ax+By+C=0***.

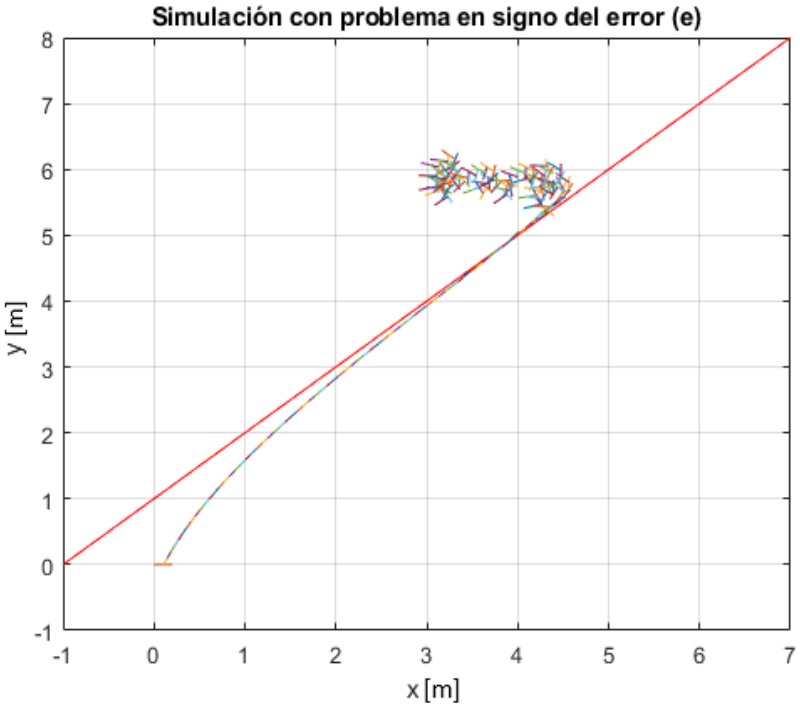


Figura 28 Simulación del problema del signo del error de posicionamiento

Como se ha mencionado, al ejecutar el código planteado en el apartado *4.1.1.2 Planteamiento para realizar el control lateral*, aparece la situación que se ve en la Figura 28. En ella se proyecta una trayectoria completamente lineal y se posiciona el AGV en el punto con valor de $\overrightarrow{pose}$ igual a [0,0,0]. Se puede observar cómo, primeramente, el vehículo es capaz de direccionarse e ir disminuyendo el error de posicionamiento. Además, llega a igualar el valor de $\boldsymbol{\varphi_{trayect.}}$ y $\boldsymbol{\theta}$. Tras esto, y como el control no deja de ejecutarse, en un momento dado el vehículo autónomo traspasa la línea de la trayectoria fijada. Es ahí donde surge la problemática, ya que trata de volver a posicionarse sobre la ruta, pero no es capaz de ello. Como bien se ha dicho, esto sucede por no tener en cuenta el signo del error de posicionamiento.

Para ver con más claridad lo que sucede en la zona problemática, se presenta un aumento de la Figura 28:

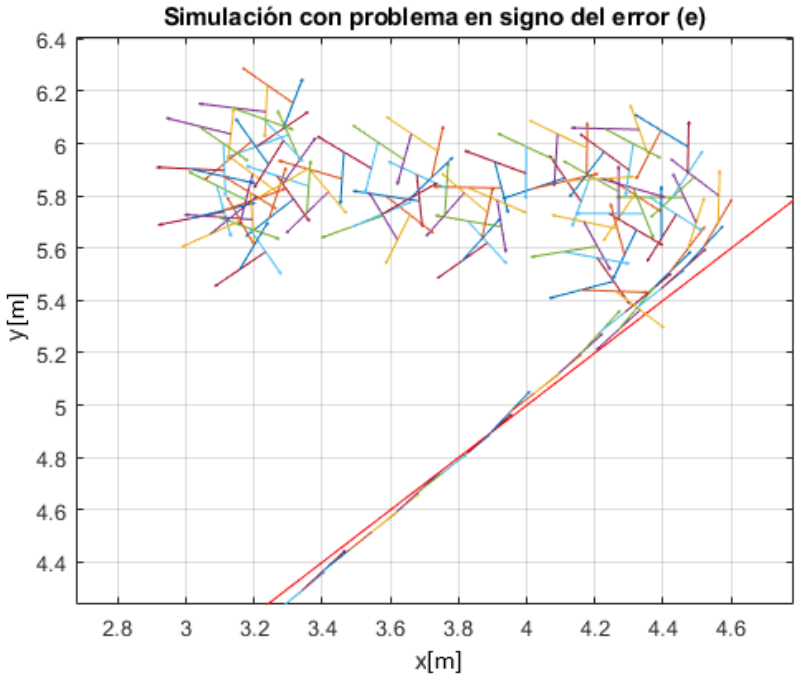


Figura 29 Aumento de simulación del problema del signo del error de posicionamiento

En la imagen superior se puede observar la trayectoria en rojo y, a través de diferentes flechas de colores, se representa la posición del AGV en cada iteración, además del direccionamiento de este. Como se ha explicado, se puede ver que la corrección de ambos errores, **e** y **φ**, se hace de manera adecuada y continua, hasta posicionarse encima de la trayectoria. Por mínimo que sea, siempre va a existir un error de posicionamiento, provocando que **e** no sea exactamente 0 y desviando ligeramente el vehículo autónomo de la ruta. Eso provoca que, en este caso, el AGV se sitúe en el lado superior de la trayectoria. El valor del error de posicionamiento mantiene el mismo signo, no reflejando esta nueva situación en el parámetro $\boldsymbol{\varphi_{consigna}}$ y arrastrado el problema hasta el cálculo de **ω**.

Para solucionar este funcionamiento erróneo, se realiza un nuevo planteamiento que hay que tener en cuenta en el código. En la siguiente imagen, se representa el vector de error de posicionamiento $\vec{e}$ y el vector que indica en que dirección se quiere hacer el seguimiento de la trayectoria, $\vec{N}$. El parámetro **α** hace referencia al ángulo que se forma entre ambos vectores.

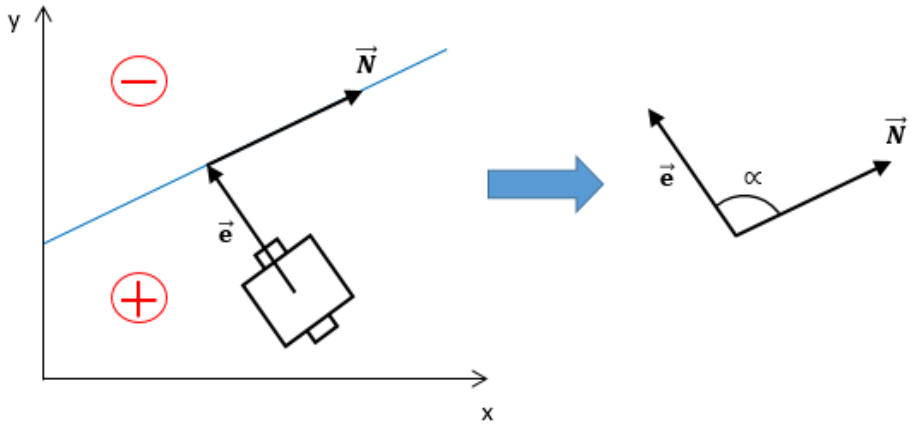


Figura 30 Vectores de error de posición y de dirección de trayectoria

La Figura 30 también muestra las regiones donde se considera el error positivo y el error negativo, con referencia a la trayectoria. De este modo, la dirección del vector $\vec{e}$ estará

condicionada por este signo, siendo la dirección que se necesite tomar para corregir el error de posicionamiento. Es necesario establecer esta política, de forma que $\vec{e}$ siempre tome esos signos, ya que si el AGV está en el lado positivo necesita aumentar el ángulo $\boldsymbol{\varphi}$ y si está en el negativo, disminuirlo. Se realiza el producto vectorial entre $\vec{N}$ y $\vec{e}$ para diseñar la solución:

$$\vec{N} \wedge \vec{e} = \begin{bmatrix} n_x \\ n_y \\ 0 \end{bmatrix} \wedge \begin{bmatrix} e_x \\ e_x \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ n_x e_y - n_y e_x \end{bmatrix} \quad (15)$$

Por definición, este producto al que se denomina como $\vec{T}$, se puede representar de la siguiente manera:

$$\left|\vec{T}\right| = |\vec{n}||\vec{e}| \sin \alpha \quad (16)$$

Al hacer que $|\vec{n}|$ y $|\vec{e}|$ sean 1, es posible realizar un análisis respecto al valor de $\boldsymbol{\alpha}$, donde:

- Si $\boldsymbol{\alpha}$>0, el error de posicionamiento deberá ser positivo.
- Si $\boldsymbol{\alpha}$<0, el error de posicionamiento tendrá que tener signo negativo.

Sabiendo que la trayectoria es conocida es posible calcular el valor de los siguientes puntos. En la imagen contigua el punto **P1** hace referencia al punto más cercano de la trayectoria desde el vehículo autónomo. El punto **P2** es el siguiente punto más cercano al AGV después de **P1**.

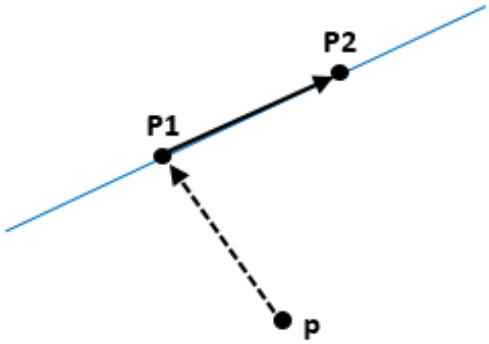


Figura 31 Punto más cercano de la trayectoria, siguiente y actual, respecto a AGV

De esta manera, se pueden calcular los valores correspondientes a $\left|\vec{N}\right|$ y $|\vec{e}|$, con el siguiente planteamiento, siendo $\vec{p}$ el posicionamiento actual del vehículo autónomo:

$$\vec{N} = \frac{\overrightarrow{P2} - \overrightarrow{P1}}{\left\|\overrightarrow{P2} - \overrightarrow{P1}\right\|} \quad (17)$$

$$\vec{e} = \overrightarrow{P1} - \vec{p} \quad (18)$$

Conociendo todos los datos, se deberá incluir en el código la siguiente relación al calcular el error de posicionamiento, teniendo en cuenta que las ecuaciones anteriores devuelven vectores como resultado:

$$e \sin \propto = T = n(1)e(2) - n(2)e(1) \quad (19)$$

Se consigue solucionar así la problemática mostrada en la Figura 28, pudiendo realizar un ajuste adecuado a la trayectoria establecida.

4.1.1.4 Estudio de la estabilidad del sistema de navegación

Es necesario realizar un análisis para comprobar si el diseño propuesto es estable para la navegación frente a cualquier tipo de trayectoria. Con tal finalidad, se va a usar la función de Lyapunov, que demuestran la estabilidad de un sistema dinámico en un punto fijo. Conociendo que es una función de energía, si se asegura que dicha energía es positiva, pero su derivada en el tiempo es siempre negativa, se consigue demostrar la estabilidad asintótica del sistema. Además, a través de este estudio, es posible determinar los valores de las constantes $\mathbf{K_1}$ y $\mathbf{K_2}$, involucradas en las relaciones (13) y (14).

Con el objetivo de simplificar el análisis, se continúa contemplando una trayectoria con la forma **Ax+By+C=0**. Se parte de la ecuación (10), pudiendo plantear el sistema de energías en el que se centrará el estudio, denominado **L**:

$$\begin{cases} L = e^2 + (\theta - \varphi_{trayect.})^2 > 0 \\ \dfrac{dL}{dt} < 0 \end{cases} \tag{20}$$

Del mismo modo, para llevar a cabo la resolución, se tienen diferentes ecuaciones que se necesitan retomar. Algunas de estas son las mismas vistas anteriormente, con diferente planteamiento, mientras que otras son nuevas, pero no necesitan aclaración:

$$\dot{x} = \frac{dx}{dt} = V\cos\theta \tag{21}$$

$$\dot{y} = \frac{dy}{dt} = V\sin\theta \tag{22}$$

$$\omega = \dot{\theta} = K_1\left(f(e) - \left(\theta - \varphi_{trayect.}\right)\right) \qquad donde\ f(e) = \varphi_{consigna} \tag{23}$$

$$f(e) = arctg(K_2\ e) \qquad donde\ f(e) = \varphi_{consigna} \tag{24}$$

$$e = \frac{Ax + By + C}{\sqrt{A^2 + B^2}} = distmin.(\vec{p}, trayectoria) \tag{25}$$

Recordemos que $\vec{\boldsymbol{p}}$ es el parámetro referido a la posición del vehículo autónomo. Además, a partir de ahora, se tratará el dato de $\boldsymbol{\varphi_{consigna}}$ como una función dependiente del error, para comprender mejor el proceso. Como se ha venido diciendo a lo largo del trabajo, los objetivo para efectuar el control lateral son el conseguir que error de posicionamiento **e** tienda a 0 y que el ángulo del vehículo **θ** iguale al de la trayectoria $\boldsymbol{\varphi_{trayect.}}$. Para realizar el análisis, se plantean dos situaciones diferentes dependiendo de este último valor.

4.1.1.4.1 Situación en la que $\varphi_{trayect.}$ es nulo

Se presenta la situación en la que el valor del ángulo de la trayectoria es nulo, como puede verse en la siguiente imagen. Considerando tal idea, los términos **A** y **C** desaparecen de la ecuación **Ax+By+C=0**, quedando **By=0**, y simplificando la resolución.

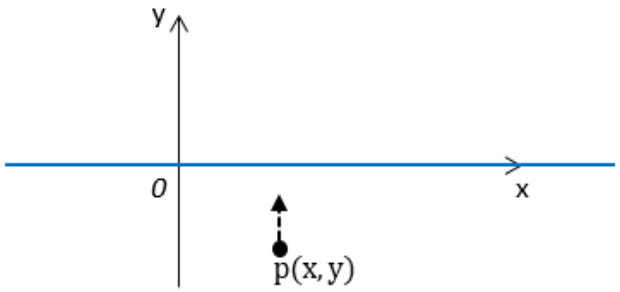


Figura 32 Planteamiento de $\varphi_{trayect.}$ =0 para análisis de estabilidad

4.1.1.4.1.1 Error de posicionamiento elevado

En primer lugar, se hace la suposición de que el vehículo autónomo este situado a una distancia considerable de la trayectoria, pudiendo obtener el valor del error de posicionamiento de manera directa:

$$e = -y \tag{26}$$

Se recupera la ecuación (23) con el planteamiento de la situación de estudio, quedando de la siguiente forma:

$$\dot{\theta} = K_1(f(e) - \theta) \tag{27}$$

Suponiendo que la dinámica de esta relación es muy rápida, es posible conocer el ángulo **θ**, ya que el AGV se va a orientar rápidamente, posicionándose con la dirección que se aprecia en la Figura 32:

$$\theta = f(e) = arctg(K_2\, e) \tag{28}$$

Al retomar las ecuaciones (21) y (22) referidas a la velocidad, se ve que es posible reescribirlas teniendo en cuenta las relaciones (26) y (28):

$$\frac{dx}{dt} = V\cos(-arctg(K_2\, y)) \tag{29}$$

$$\frac{dy}{dt} = V\sin(-arctg(K_2\, y)) = -V\sin(arctg(K_2\, y)) \tag{30}$$

Recordemos que el objetivo es asegurar que **L** es estable, de manera que siempre sea de signo positivo. Se puede formular la energía en esta situación de la siguiente manera, teniendo en cuenta las consideraciones previas:

$$L = e^2 = y^2 \tag{31}$$

Al tener un elevado al cuadrado en la ecuación, queda confirmado que siempre va a ser positiva. La problemática, por lo tanto, recae en la segunda expresión del sistema (20), donde sí que resulta necesario demostrar que la derivada de **L** es siempre negativa. Se plantea así mismo dicha derivada:

$$\dot{L} = \frac{dL}{dt} = 2\, e\dot{e} = 2y\dot{y} \tag{32}$$

Además, es posible complementar la expresión (31) con lo visto en la ecuación (30):

$$\dot{L} = 2y(-V\sin(arctg(K_2\, y))) \tag{33}$$

Por la forma en la que se ha diseñado la relación (24), se conoce que la constante $\mathbf{K_2}$ siempre será positiva, pudiendo hacer las siguientes consideraciones en referencia a la anterior ecuación:

$$signo(arctg(K_2\, y)) = signo(y) \tag{34}$$

$$signo(\sin(arctg(K_2\, y))) = signo(arctg(K_2\, y)) \tag{35}$$

De esta manera se llega a la conclusión de que:

$$signo(y) = signo(\sin(arctg(K_2\, y))) \tag{36}$$

Se analiza ahora la relación (33). Con la anterior deducción y teniendo en cuenta que el error es negativo, tal y como se ha planteado para la ecuación (26), es posible asegurar que la función de Lyapunov se cumple a través del sistema (20).

4.1.1.4.1.2 Error de posicionamiento reducido

Para que el sistema sea estable cuando el vehículo autónomo esté alejado de la trayectoria, se ha supuesto que se tiene una dinámica muy rápida. Con el fin de conseguirla, el AGV se deberá posicionar rápidamente a 90º, esto es, perpendicular a la trayectoria. Además, se aproximará muy rápido a esta, teniendo un error de posicionamiento muy reducido. Siendo así, el planteamiento cambia, considerando ahora únicamente el ángulo **θ** del robot movil, ya que el error de posicionamiento tenderá a 0 rápidamente. Se reformula así la expresión (23):

$$\dot{\theta} = K_1(0 - \theta) = -K_1\theta \tag{37}$$

En esta situación es necesario volver a estudiar la estabilidad del sistema. Se recupera el sistema (20), que, en esta ocasión dependerá únicamente del ángulo del AGV como se aprecia a continuación. Además, se ha visto que la derivada de **θ** es de signo negativo. De igual modo, como **L** está formado por **θ** elevado al cuadrado, el análisis se centra en confirmar el signo negativo de la derivada de la energía.

$$\dot{L} = \frac{dL}{dt} = 2\,\theta\dot{\theta} \tag{38}$$

El estudio plantea que el vehículo autónomo este posicionado debajo de la trayectoria. Tal y como se ha visto en el apartado *4.1.1.3 Distinción del signo del error*, se conoce que **θ** va a tomar valores positivos en esa zona. Así mismo, como el error de posicionamiento es prácticamente nulo, el ángulo del vehículo **θ** tenderá a igualar al de la trayectoria $\boldsymbol{\varphi_{trayect.}}$. Por lo tanto, se puede formular la integral de la ecuación (37), que al ser un sistema lineal queda de la siguiente manera:

$$\theta = \frac{\pi}{2} exp^{-K_1\tau} \tag{39}$$

Se deriva esta expresión, quedando descrita de una forma más completa que en la ecuación (27):

$$\dot{\theta} = -\frac{\pi}{2} K_1 exp^{-K_1\tau} \tag{40}$$

El hecho de que la dinámica del sistema sea rápida, afecta principalmente al error de posicionamiento. Se retoma por lo tanto la expresión (32), de la que se conoce el valor de $\dot{\boldsymbol{y}}$, e integrando esta, el de $\boldsymbol{y}$. Para obtener ambos, se toma la fórmula (39), que es posible sustituir en la ecuación (22):

$$\dot{y} = V \sin\left(\frac{\pi}{2} exp^{-K_1\tau}\right) \tag{41}$$

$$y = V \int_0^t \sin\left(\frac{\pi}{2} exp^{-K_1\tau}\right) d\tau + y(0) \tag{42}$$

Referente a la primera expresión, se puede conocer que $\dot{\boldsymbol{y}}$ siempre será negativa debido a que la función de Seno se mantendrá entre 0 y π. En cuanto a la ecuación (42), la condición inicial siempre será también negativa. Queda por lo tanto asegurar que la integral de dicha ecuación no adopte un valor superior al de **y(0)**, de manera que se cumpla la estabilidad. Para ello, es necesario asegurar que el error de posicionamiento no cambie de signo, criterio que sirve para poder estimar $\mathbf{K_1}$. Se plantea así la siguiente fórmula, identificando con **I** la función de la integral:

$$y = V\, I(K_1, t) + y(0) < 0 \tag{43}$$

$$V\, I(K_1, t) < -y(0) \tag{44}$$

De esta manera se consigue asociar la velocidad y la constante **K_1**. Analizando la relación anterior junto con la expresión (42), se pueden obtener las siguientes conclusiones. Si se fija un valor de **K_1** muy agresivo, la exponencial tenderá a 0 rápidamente, consiguiendo un Seno de 0 que hará que la expresión se desvanezca con brevedad, siendo independiente del valor de **V** que se establezca. En cambio, si el parámetro **K_1** es pequeño, la velocidad quedaría limitada, ya que, si no, se tendría un sistema oscilatorio que no se desea. Se implementa este sistema en Matlab, obteniendo los siguientes resultados en simulación.

En este primer gráfico, se ha establecido una **V** de 100 y una **K_1** de 1. Se visualiza como con una constante **K_1** muy laxa, el valor de la integral a lo largo del tiempo (representado con una línea roja), tarda mucho en desvanecerse, de manera que el valor de la función de **y** (dibujado con una línea azul), toma un valor positivo muy elevado sin poder asegurar la estabilidad.

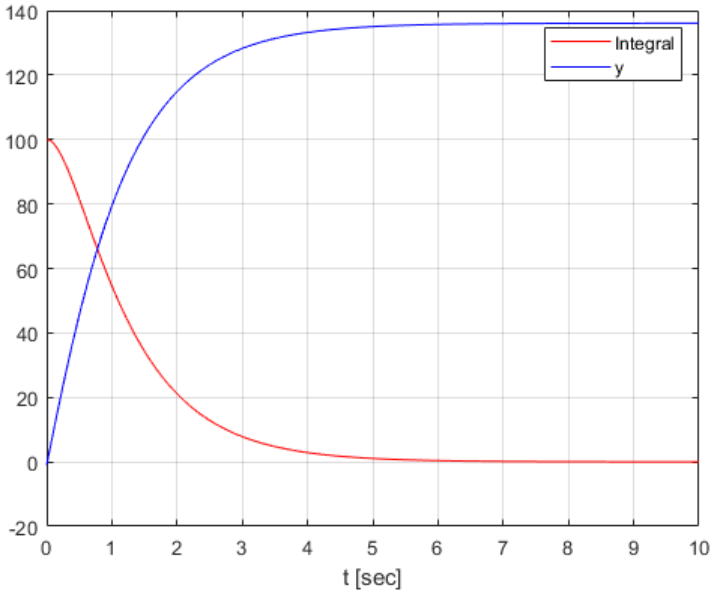


Figura 33 Gráfica con V grande y K_1 pequeña

Sin embargo, en esta segunda gráfica se aprecia que con un valor de velocidad de 10 y una **K_1** de 1000, la integral se desvanece en un tiempo muy reducido. De este modo, el valor de la función de **y**, representada en la ecuación (42), siempre es negativa, pudiendo garantizar la estabilidad del sistema.

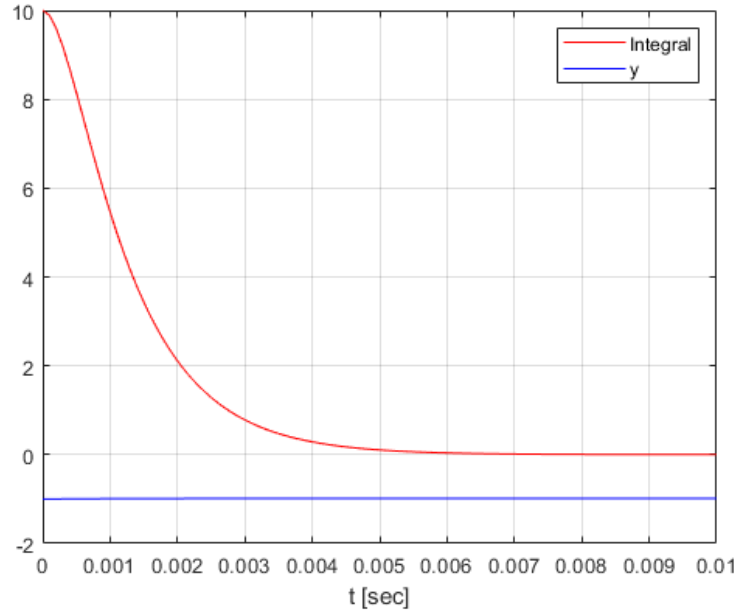


Figura 34 Gráfica con V pequeña y K_1 grande

Se comprende con este análisis la necesidad de un control longitudinal para que el sistema siempre sea estable. Como 10 es un valor razonable de velocidad, en un rango habitual del AGV, se establece que el valor de $\mathbf{K_1}$ sea de 1000, igual que en la Figura 34, siendo la ganancia del lazo de control del error de alineamiento.

$$K_1 = 1000\ (s^{-1})$$

4.1.1.4.2 Situación en la que $\varphi_{trayect.}$ no es nulo

Una vez asegurada la estabilidad cuando la trayectoria tiene un ángulo nulo, se debe de plantear el caso en el que tenga inclinación, tal y como se aprecia en la siguiente figura.

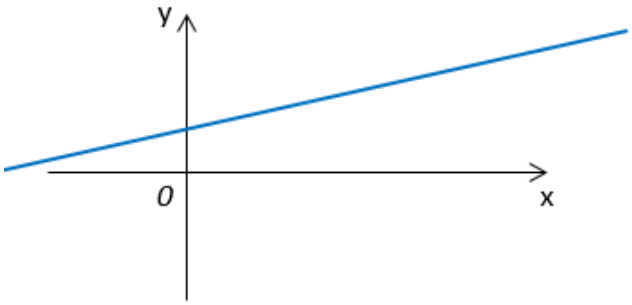


Figura 35 Planteamiento de $\varphi_{trayect.} \neq 0$ para análisis de estabilidad

Como ya se ha analizado, se conoce que el sistema tendrá una dinámica muy rápida consiguiendo en muy poco tiempo que el AGV adopte la dirección deseada. Así pues, si se recupera la ecuación (23), que es posible reescribir de una forma simplificada:

$$\theta = f(e) + \varphi_{trayect.}(e) \tag{45}$$

En esta situación, el ángulo de la trayectoria $\boldsymbol{\varphi_{trayect.}}$ también debe de ser una función que dependa del error de posicionamiento. Esto se debe a la posición en ***x*** e ***y*** del AGV, ya que, en función de ello, variará el punto más cercano a la trayectoria. Como dicho punto más cercano se estima a través de la distancia más corta, el error será:

$$(x_{cerca}, y_{cerca}) = Argmin\left(\left\|(x,y) - \left(x_{trayect.}, y_{trayect.}\right)\right\|\right) \tag{46}$$

$$\varphi_{trayect.}(e) = \ \varphi_{trayect.}(x_{cerca}, y_{cerca}) \tag{47}$$

Resolver esto puede resultar muy complejo si se hace de manera analítica, puesto que nuevamente, hay que plantear el sistema (20), que se reformula de la siguiente manera:

$$\begin{cases} L = e^2 > 0 \\ \dot{L} = 2e\dot{e} < 0 \end{cases} \tag{48}$$

Es posible definir el error de posicionamiento de la manera representada en la relación (25), donde resultaría necesario calcular los parámetros **A**, **B** y **C**, teniendo en cuenta la posición del AGV, ya que de ello dependerá el signo de **e**:

$$(A, B, C) = f\left((x_{cerca}, y_{cerca}), \vec{p}\right) \tag{49}$$

Dada su complicación, se plantea la resolución definiendo el sistema de manera discreta. Se propone un algoritmo de optimización, donde se calcula el error de posicionamiento para todos los puntos de una zona acotada. Una vez obtenidos dichos valores, y habiendo fijado el ángulo **θ** que tendrá el vehículo autónomo, se variará la constante $\mathbf{K_2}$, pudiendo determinar así su valor. Desde este punto de vista, el posible plantear el siguiente sistema en el instante ***t***:

$$\theta(x, y, K_2) = f(e, K_2) + \varphi_{trayect.}(e) \tag{50}$$

$$e(x(t), y(t)) = e(t) \tag{51}$$

En vista de que el ángulo y el error de posicionamiento del AGV dependerán de la posición actual y la de la constante $\mathbf{K_2}$, es posible asumir el valor de **e** en un instante ***t+dt***, teniendo en cuenta las expresiones (21) y (22):

$$x(t+dt) = x + V\cos\theta \tag{52}$$

$$y(t+dt) = y + V\sin\theta \tag{53}$$

$$e\big(x(t+dt), y(t+dt)\big) = e(t+dt) \tag{54}$$

$$\Delta e = e(t+dt) - e(t) \tag{55}$$

Es esta última relación la que necesariamente debe de ser negativa para poder asegurar la estabilidad del sistema. El problema se encuentra en las discontinuidades que pueden generarse. Se crean cuando el punto más cercano de la trayectoria en ***t*** y en ***t+dt*** no es el mismo, no pudiendo asegurar que el error en un instante próximo sea inferior al del instante actual.

Sin embargo, si en lugar de tomar el error por completo en todo su recorrido (hasta alcanzar la trayectoria) se analiza en cada iteración, esto es, a tramos, se confirma que siempre se va a cumplir la condición de que la expresión (55) sea negativa. Lógicamente, a medida que el AGV avanza, se tienen puntos próximos más cercanos a la ruta, de manera que el error de posicionamiento va a ir siempre disminuyendo. Para que esto se acate, resulta necesario tomar el valor de la variación de **e** en valor absoluto, demostrando que el sistema es estable. Una forma de visualizar que el sistema (48) se sigue, es plantear la simulación de la estabilidad para todos los puntos $\vec{\boldsymbol{p}}$ del espacio.

Se establece una trayectoria sinusoidal, que cumple con el planteamiento de que $\boldsymbol{\varphi_{trayect.}}$ no sea nulo, y de la cual se obtienen las siguientes gráficas.

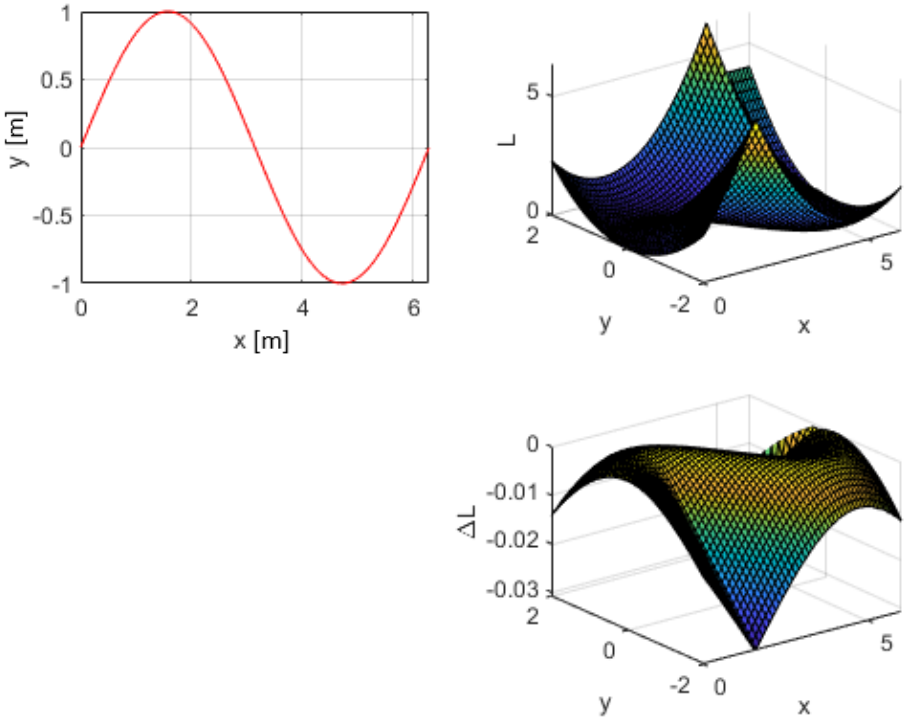


Figura 36 Estabilidad de sistema con $\varphi_{trayect.}$ no nulo

La grafica superior derecha muestra como la ecuación de la energía de Lyapunov siempre es positiva, en todo el rango de la función sinusoidal. En cambio, en la gráfica inferior, es apreciable como la derivada de la energía siempre toma valores negativos, correspondiéndose con la variación del error de posicionamiento. De este modo queda demostrada, de una manera visual, la estabilidad en esta situación.

Conociendo que el sistema es estable, se plantea un código en el que se experimenta con todos los valores de $\mathbf{K_2}$ en un rango, para todo punto $\vec{\boldsymbol{p}}$. Teniendo el valor de la constante $\mathbf{K_2}$ establecido, es posible visualizar la evolución de $\Delta\boldsymbol{e}$ y conseguir el óptimo de dicha invariable, ya que el error de posicionamiento tenderá en todos los casos a 0. Se propone la optimización generando una función de coste que dependa del error cuadrático medio. Además, es sabido que el módulo del error de posicionamiento se obtiene con la resta de la posición en la que se encuentra el AGV y el punto más cercano de la trayectoria en referencia al vehículo:

$$\|e\| = \sqrt{(x - x_{cerca})^2 + (y - y_{cerca})^2} \tag{56}$$

$$J = \frac{1}{t}\left(\int_0^t \|e(K_2, \vec{p}, trayec.)\|\, dt\right) = \frac{1}{N}\sum_{k=0}^{k=N} e^2(k) \tag{57}$$

De este modo, se obtiene un promediado de la función de coste que está sujeto también al punto de inicio del vehículo autónomo:

$$E_{\vec{p}}(K_2, \vec{p}, trayec.) = \bar{J}(K_2, \vec{p}, trayec.) \tag{58}$$

Es obvio que la trayectoria no es un parámetro variable. Igualmente, se ha mencionado que se va a experimentar con todo punto $\vec{\boldsymbol{p}}$, de forma que **J** únicamente dependerá del valor de la constante $\mathbf{K_2}$, siendo:

$$K_2 = Argmin(\bar{J}) \tag{59}$$

Se simula este planteamiento, obteniendo los valores con relación al promediado de la función de coste y al parámetro $\mathbf{K_2}$ que pueden verse en la gráfica contigua.

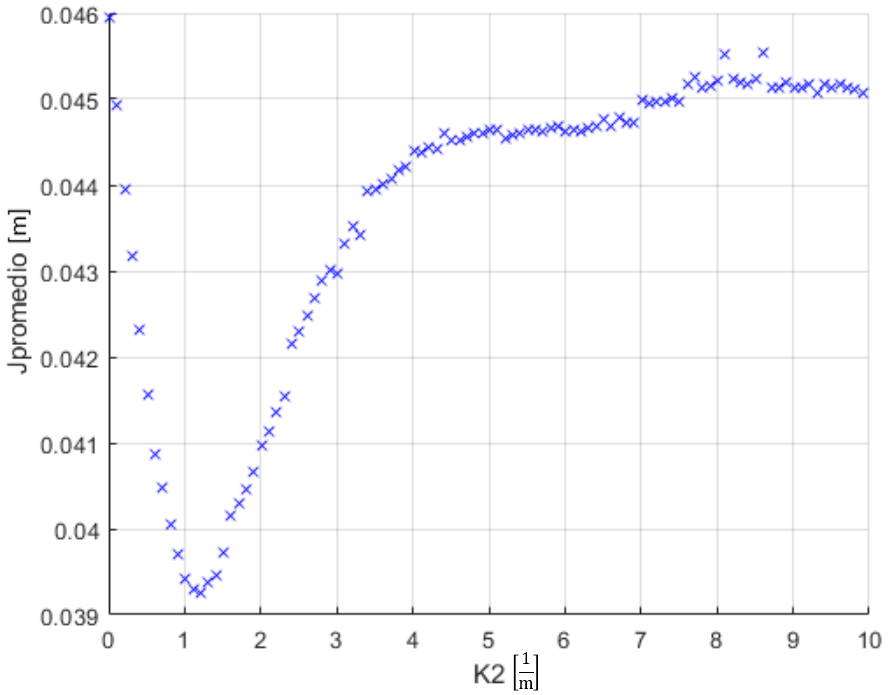


Figura 37 Valor óptimo de K2

En la figura superior se aprecia que hay un valor óptimo que minimiza **J**. Este valor se corresponde a 1,21. Recordar que $\mathbf{K_2}$ es el parámetro que actúa sobre la velocidad de evolución del ángulo de consigna $\boldsymbol{\varphi}_{\mathbf{consigna}}$.

$$K_2 = 1{,}21\ \left(\frac{1}{m}\right)$$

4.1.2 Control longitudinal del AGV

Continuando con el propósito de que un vehículo autónomo industrial sea capaz de situarse encima de una trayectoria y continuarla, se resuelve la problemática del control longitudinal, que tiene como objetivo la supervisión de la velocidad. Adicionalmente, al analizar la estabilidad del algoritmo propuesto, se ha visto la necesidad de un control que regule la velocidad. De igual modo, hay que tener en cuenta que no se podrá tener la misma velocidad en trayectorias lentas, donde el AGV podrá ir más rápido, o en curvas, donde tendrá la necesidad de frenar.

Con el fin de conseguir tal propósito, se diseña una política de velocidad que dependerá de tres situaciones diferentes:

- El primer estado en el que se puede encontrar el vehículo industrial autónomo, es cuando se encuentra lejos de la trayectoria, esto es, con un error de posicionamiento elevado. Siendo así, el ángulo del vehículo resulta indiferente, ya que se requiere que adopte una velocidad alta para alcanzar rápido la ruta a seguir.

- La segunda situación representa las curvas de la trayectoria o el instante en el que AGV esté muy cerca de esta. Cuando existe una curvatura, el ángulo del vehículo y el de la trayectoria no coinciden, de manera que habrá que reducir la velocidad para que no sobrepase la ruta y sea capaz de realizar un buen seguimiento

- El último caso que se contempla, es cuando el vehículo autónomo está situado encima de la trayectoria, y esta es recta. Se conoce que, en esta situación, el ángulo de la trayectoria y el del AGV se igualaran, de manera que se podrá realizar un aumento de la velocidad.

Llegados a este punto se comprende que, si la trayectoria a seguir es conocida, es posible hallar los valores de $\boldsymbol{\varphi_{trayect.}(t)}$ y $\boldsymbol{\varphi_{trayect.}(t+1)}$, tal y como se refleja en la siguiente imagen.

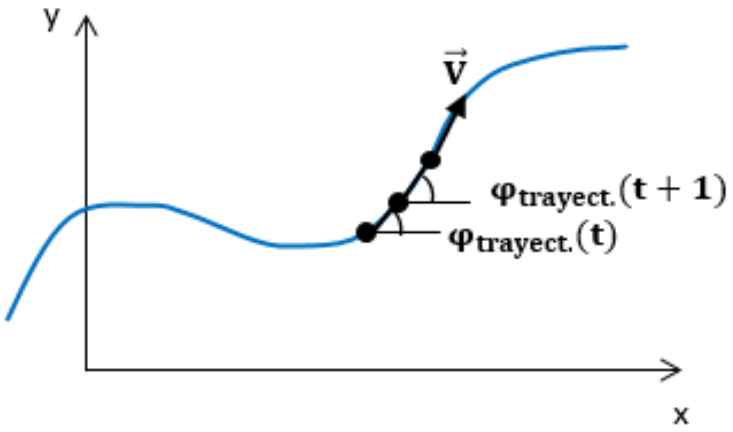


Figura 38 Planteamiento de $\varphi_{trayect.}$ en dos instantes

Conociendo que en algún momento dado el AGV se encontrará con curvas, cabe la posibilidad realizar el planteamiento de la Figura 38, siendo simplemente una aproximación circular.

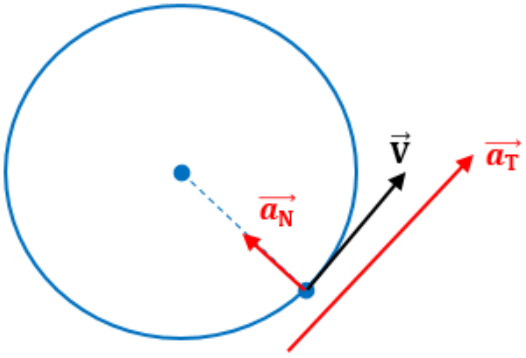


Figura 39 Aproximación circular de trayectoria, con vectores

De esta manera y tal como se aprecia en la figura superior, en cada punto se tendrá una aceleración tangencial y una aceleración normal. Además, es conocido que la aceleración normal puede definirse de las siguientes formas:

$$a_N = \frac{V^2}{\rho} = \omega^2 \rho = V\omega \tag{60}$$

Así mismo, se decide que el módulo de dicha aceleración siempre se mantenga por debajo de una aceleración máxima:

$$\|a_N\| \leq a_{Nmax} \tag{61}$$

Siendo esto así, es posible despejar la velocidad lineal de la expresión (60):

$$V = \frac{a_N}{\omega} \tag{62}$$

Recordemos que la velocidad angular viene dada por las invariables $\mathbf{K_1}$ y $\mathbf{K_2}$, de modo que resulta necesario fijar un valor de velocidad lineal máximo:

$$V_{max}{}' = \frac{a_{Nmax}}{\omega} \tag{63}$$

Tomando la anterior ecuación, se plantea el valor de velocidad lineal del AGV, teniendo en cuenta que deberá depender del error de posicionamiento:

$$V = \min\left(\frac{a_{Nmax}}{\omega},\ V_{max} = f(e)\right) \tag{64}$$

Se aprecia como esta última relación contempla la política de velocidad diseñada. Dependiendo del error de posicionamiento, el AGV tendrá más o menos velocidad, pero también dependerá de la curvatura de la trayectoria, que, si es elevada será quien marque el límite.

En la Figura 38, puede verse como es posible conocer el ángulo que tendrá la trayectoria en el siguiente instante. Este factor es muy importante a la hora de determinar la velocidad lineal, ya que existe una necesidad de predecir el valor que adoptará **V** a medida que el vehículo autónomo avance. Con esta idea, se consigue conocer cuándo se aproxima a una curva, por ejemplo, y con ello reducir la velocidad.

Con objetivo de realizar dicha predicción, se plantean los vectores que componen la aceleración:

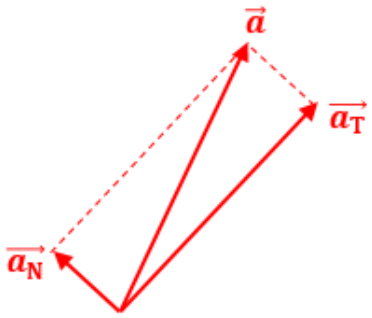


Figura 40 Vectores de aceleración

Se retoma la ecuación (60), donde, como se ha mencionado, el valor de **ω** está dado por las constantes del control lateral. El dato de **ρ** en cambio, se puede obtener de forma directa desde la trayectoria:

$$\rho = \frac{dl}{d\theta} \tag{65}$$

$$\rho = \frac{\left\|\vec{x}_{trayect.}(t+1) - \vec{x}_{trayect.}(t)\right\|}{\varphi_{trayect.}(t+1) - \varphi_{trayect.}(t)} \tag{66}$$

Con el parámetro **l** se referencia a la longitud del arco entre un instante y el siguiente, como puede verse en la expresión (66). En consecuencia, se consigue tener la aceleración normal completamente definida. Además, la aceleración es un dato propio de las características del AGV, pudiendo formular el valor de la aceleración tangencial basándose en la Figura 40:

$$a^2 = {a_N}^2 + {a_T}^2 \rightarrow a_T = \sqrt{a^2 - {a_N}^2} \tag{67}$$

Es posible, a su vez, sustituir la expresión (60) de la aceleración normal, de la que se conocen todos los parámetros, en esta última ecuación, quedando de la siguiente manera:

$$a_T = \sqrt{a^2 - (\omega^2\rho)^2} \tag{68}$$

Por consiguiente, se puede definir la velocidad del AGV en el próximo instante:

$$V(t+1) = V(t) + \Delta t a_T \tag{69}$$

$$V(t+1) = V(t) + \Delta t\sqrt{a^2 - (\omega^2\rho)^2} \tag{70}$$

Para facilitar el desarrollo de esta última formula, evitando así líneas de código, se reescribe utilizando la velocidad lineal en lugar de **ρ**, acorde con la ecuación (60):

$$V(t+1) = V(t) + \Delta t\sqrt{a^2 - V(t)^2\omega(t)^2} \tag{69}$$

Se dispone del valor de todas las variables que componen la relación anterior, de manera que tiene lugar la obtención la velocidad máxima del instante ***t+1***, denominada como **V_{max}''**. De este modo y rescatando el planteamiento de la ecuación (64), se realiza esta formulación:

$$V(t+1) = \min(V_{max}'', V_{max}' = f(e)) \tag{70}$$

Analizando la expresión obtenida, en caso de que el AGV tenga una velocidad angular muy grande, la velocidad lineal se verá reducida, estableciendo un límite. Se consigue así adaptar la **V**. Además, el parámetro **V_{max}'** vendrá dado por el vehículo autónomo, correspondiéndose con la velocidad nominal máxima. De igual modo, se contempla el error de posicionamiento, dando importancia a la rapidez en la ejecución de la trayectoria.

$$V(t+1) = f(e, \omega(t), V(t), a) \tag{71}$$

4.2 Diseño del algoritmo de planificación de trayectoria y localización

Existe la necesidad de calcular una trayectoria y, con relación a ella, poder estimar la localización del vehículo autónomo industrial. Como se ha mencionado en el apartado *3.4 Decisión en base al estado del arte*, se va a realizar una segmentación semántica del entorno, distinguiendo así la zona navegable. A partir de esta información, se planificará la ruta y se sabrá donde está situado el AGV.

4.2.1 Segmentación semántica

Recordemos que el AGV de la escuela de ingeniería de Vitoria-Gasteiz cuenta con visión artificial. Conociendo esto, se va a implementar una red neuronal capaz de realizar una segmentación semántica de una imagen, pudiendo diferenciar la zona hábil para que el vehículo autónomo se desplace.

4.2.1.1 Adaptación de imágenes

Al tomar una imagen de un entorno, se obtiene una representación de este en los ejes ***x*** e ***y***, no reproduciendo las medidas espaciales reales. Sin embargo, a la hora de estimar la trayectoria necesaria, los ejes de interés son el ***y*** y el ***z***, ya que se requiere conocer el ancho y largo real de la zona, basándose en la imagen capturada. Con este objetivo, se propone que la imagen tomada por la visión del AGV se transforme a una imagen en vista de pájaro. Esta interpretación hace posible el poder obtener las medidas reales, siendo las que interesan de cara a transmitir información al algoritmo de navegación.

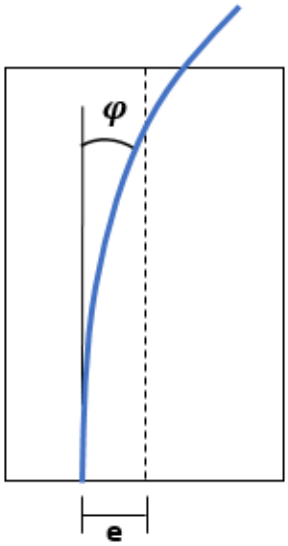


Figura 41 Planteamiento de vista de pájaro

La Figura 41 muestra el planteamiento de la transformación de la imagen, que además da pie a resolver la problemática de la localización. La cámara está posicionada en la parte delantera central del vehículo autónomo, de manera que las fotografías capturadas estarán centradas respecto a este. Al convertir la imagen a vista de pájaro, se obtiene el ancho real de la captura en toda su longitud. Valiéndose de ello y conociendo donde está la trayectoria, calculada tal y como se explica en el apartado *4.2.2 Cálculo de la trayectoria*, es posible conocer el valor del el error de posicionamiento del vehículo respecto a la trayectoria. Se define a través de la diferencia entre la mitad de la imagen y el primer píxel detectado como ruta. De este modo, se obtiene la localización del AGV, resolviendo la problemática correspondiente a ello. Además, se extrae un dato fundamental para poder aplicar el algoritmo de navegación.

Por otra parte, al conseguir la longitud real del eje z, es posible medir las desviaciones de la trayectoria, pudiendo así estimar el ángulo de la trayectoria $\boldsymbol{\varphi_{trayect.}}$. Este valor se calcula también con respecto al eje central de la imagen. De este modo, se obtienen la información restante requerida en la navegación, además de la propia trayectoria.

El procedimiento de la transformada a vista de pájaro se realiza con operaciones que ofrece Matlab, obteniendo los resultados que se muestran en la figura contigua.

Figura 42 Transformada de vista de pájaro

Matlab precisa de cierta información para realizar la conversión a vista de pájaro de una imagen, como son la distancia focal, el punto principal de la fotografía, el tamaño de esta, la altura a la que está posicionada la cámara y el ángulo que abarca en caso de usar ojo de pez, entre otros. Por lo general suelen ser parámetros intrínsecos de la cámara, sin necesidad de cálculos adicionales.

4.2.1.2 Entrenamiento de la red neuronal

Con el propósito de realizar la segmentación semántica, se implementa una red CNN (Convolutional Neural Network). El objetivo principal de esta red es distinguir los píxeles que corresponden a la zona navegable. Matlab ofrece herramientas para ello [42], en las que se puede asociar cada pixel de una imagen a una etiqueta de clase. Con tal fin, es necesario entrenar la red, de forma que, cuantos más datos de entrenamiento se utilicen, mejores resultados se obtendrán.

Matlab entrena una red neuronal convolucional, conocida como Deeplab V3, específicamente diseñada para poder hacer segmentación semántica en imágenes. A tal efecto, se crea dicha red partiendo de pesos inicializados de una red Resnet-18 ya entrenada. Se conoce que el CNN es el tipo de arquitectura de red neuronal profunda más utilizada. Esta tiene matrices asociadas a los colores RGB, que proporcionan un valor entre 0 y 255 a los píxeles. ResNet-18 en cambio, es una red eficiente, con una gran adaptabilidad a aplicaciones con recursos de procesamiento limitados. Se han obtenido diferentes imágenes, pero el conjunto de datos puede resultar pequeño para la aplicación, considerando adecuado el uso de esta última.

El uso de redes neuronales, por lo tanto, incorpora un proceso de aprendizaje en el que se deberán de obtener datos, que luego se prepararán para poder efectuar el entrenamiento. En esta ocasión, el conjunto de datos lo forman 65 imágenes aleatorias extraídas de un video del entorno. Ya que no es un número muy elevado de fotografías, el proceso de aprendizaje se realiza con todas ellas. Se toma la decisión crear dos etiquetas de clases, correspondientes a *Camino* y *No_camino*. En esta segunda, se incorporan paredes, obstáculos y zonas que se consideren parte del espacio no navegable.

Para realizar la presente tarea, se usa la aplicación "Image Labeler" que incorpora Matlab, en la que se hace la distinción de las dos clases, en todas las imágenes de entrenamiento, de forma manual. Esto ayuda a clasificar de una manera adecuada los píxeles por etiquetas, ya que la segmentación semántica es con respecto a ellos. Tras llevar a cabo el proceso, se generan dos bancos de imágenes. Uno pertenece a las imágenes originales con las que se ha realizado el etiquetado, mientras que el otro banco es el propio etiquetado. A continuación, se puede visualizar la clasificación del número de píxeles por cada clase.

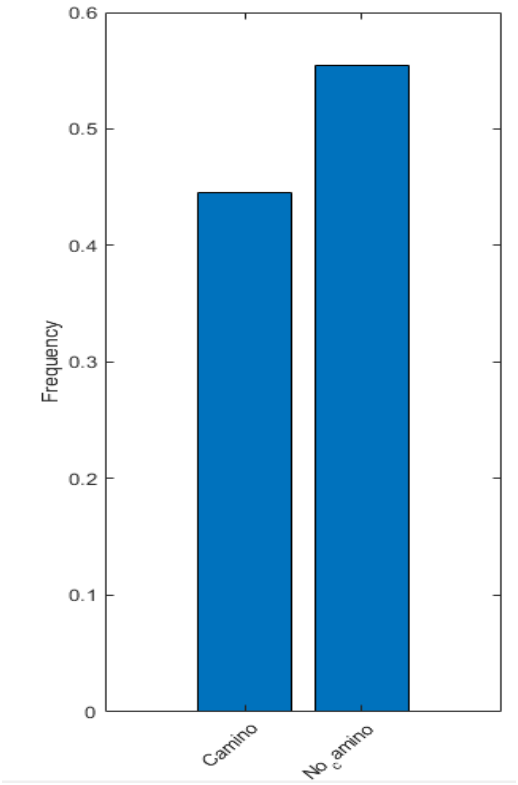


Figura 43 Cuenta de píxeles por clase

Se observa una frecuencia bastante proporcionada para tener únicamente dos clases, lo que verifica que el proceso de entrenamiento será adecuado. Hay que tener en cuenta que por pequeño que sea, ese desequilibro a favor de la etiqueta *No_camino* hará que el aprendizaje esté sesgado a favor de dicha clase. Se considera algo aceptable de modo que no se plantea ninguna ponderación.

Con los dos bancos de imágenes, se realiza el entrenamiento de la red. Se conoce que Deeplab V3 se entrena únicamente con el 60% de las imágenes del conjunto de datos. La mitad de los datos restantes se reserva para el conjunto de validación y la otra mitad para el conjunto de prueba. Esta división es aleatoria. Una vez se disponga de los tres grupos de datos, por medio de la función deeplabv3, se crea la red basada en Resnet-18, a la que hay que introducir el valor del tamaño de la imagen, en este caso, el de la vista de pájaro.

Se entrena la red creada, obteniendo los siguientes resultados.

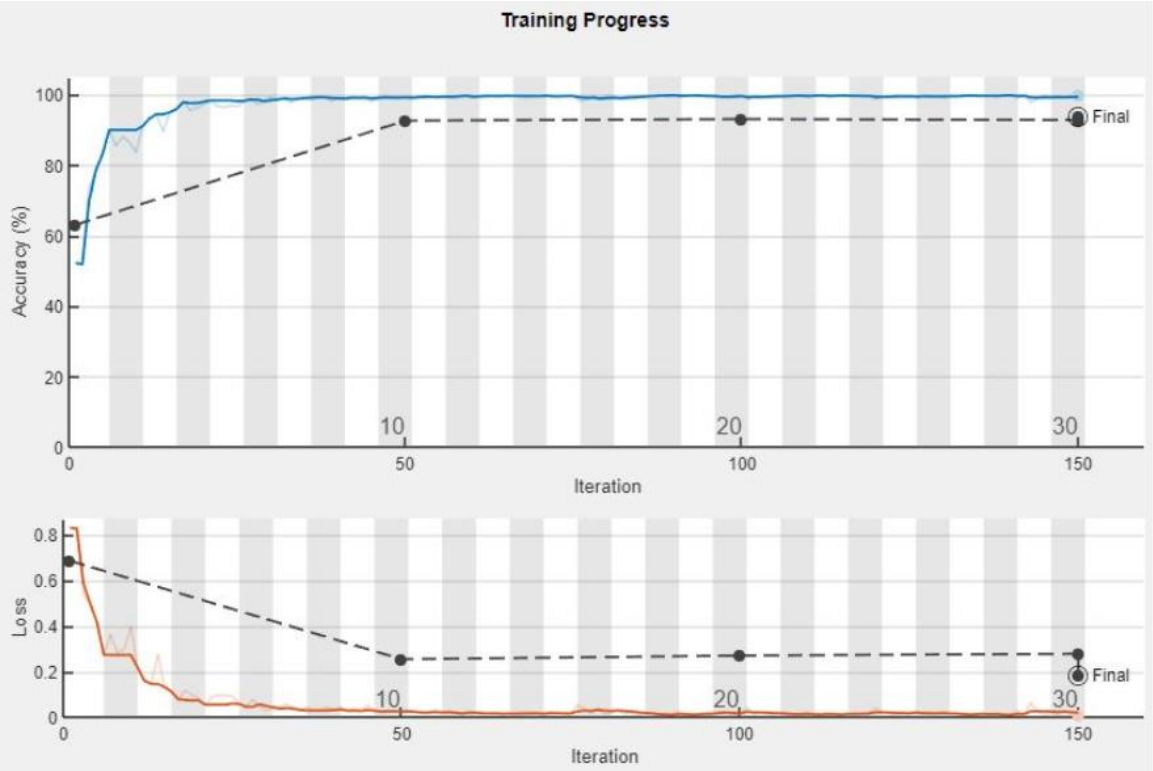


Figura 44 Gráfica de resultados de entrenamiento de red

En la Figura 44 se muestran dos gráficas, una referida al "Accuracy" y otra al "Loss". La primera de ellas representa la precisión, que se determina después de que los parámetros del modelo se ajusten a través de los datos de entrenamiento. Tras esto, el modelo de la red, clasifica el conjunto de prueba, obteniendo el porcentaje de clasificación correcto. Los trazos continuos representan los resultados del entrenamiento, siendo el color fuerte una media móvil del suave. El dato que realmente interesa es la tendencia, representada con una la línea discontinua, y obtenida por medio de los datos de validación. Se aprecia como al comienzo del entrenamiento aumenta y, tras la iteración 50, se establece.

En el caso de la gráfica de "Loss", se conoce que cuanto menor sea la pérdida, mejor modelo se obtendrá. Se calcula teniendo en cuenta los datos de entrenamiento y validación, haciendo referencia a como de bien funciona el modelo de red neuronal para estos. Siendo así, se evalúa la suma de errores cometidos para cada ejemplo de dichos conjuntos. Nuevamente, las líneas naranjas son las referidas al entrenamiento. Se presta atención a la línea discontinua que, una vez más, es estable a partir de la iteración 50. Pese a ello, se observa que tiene una tendencia a ser reducida, como es de interés.

La siguiente imagen muestra los datos numéricos del entrenamiento de la Figura 44. En ella se observa como la precisión de la red generada es de un 93,86%, tal y como se ha podido identificar en las gráficas superiores. Se puede ver también que ha completado el entrenamiento en 30 "epoch" y en 150 iteraciones. La frecuencia de validación es de 50, representada a su vez en la figura superior por puntos de color negro. Estos números los adopta Matlab en su configuración de entrenamiento, con el fin de acelerar el proceso.

Results	
Validation accuracy:	93.86%
Training finished:	Reached final iteration
Training Time	
Start time:	10-May-2021 22:46:16
Elapsed time:	47 min 55 sec
Training Cycle	
Epoch:	30 of 30
Iteration:	150 of 150
Iterations per epoch:	5
Maximum iterations:	150
Validation	
Frequency:	50 iterations
Other Information	
Hardware resource:	Single CPU
Learning rate schedule:	Piecewise
Learning rate:	9e-05

Figura 45 Datos de resultados de entrenamiento de red

Finalmente, se toma la red diseñada y se aplica sobre el algoritmo de "path plannig" y localización, obteniendo el siguiente resultado sobre el mismo instante de video que se ha visto en la Figura 42.

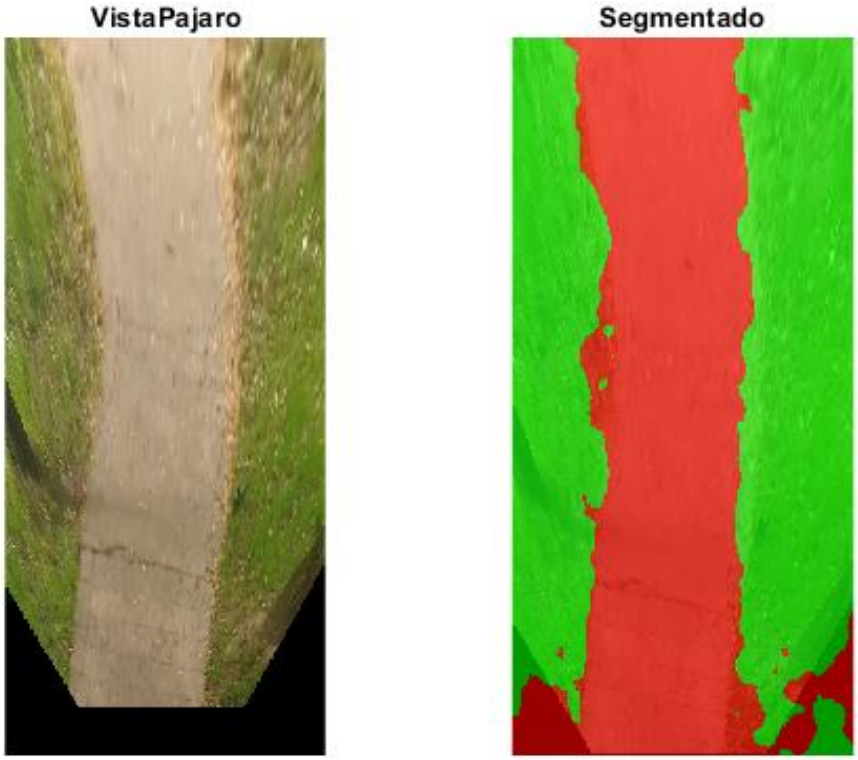


Figura 46 Segmentación semántica de imagen a vista de pájaro

Se aprecia como la segmentación se hace de una manera adecuada. Detecta el *Camino* o zona navegable, identificada por el color rojo, y cumpliendo así con el objetivo. Se debe de tener en cuenta que la transformación de vista de pájaro añade zonas negras, como se contempla en la parte inferior de las imágenes, que habría que clasificar. Para esta aplicación en concreto, no ha resultado necesario, asumiéndola como zona de *No_Camino*. Pese a detectarlas como zona navegable, no resulta problemático. Por lo tanto, se consigue la información necesaria para poder realizar el cálculo de la trayectoria a seguir por el AGV.

4.2.2 Cálculo de la trayectoria

Como se ha especificado en el apartado anterior, se dispone de una red neuronal convolucional, que es capaz de hacer una segmentación semántica de las imágenes obtenidas por la cámara del vehículo autónomo industrial. Se distingue entre las etiquetas *Camino* y *No_camino*, siendo la información importante la correspondiente a la primera clase. De esta manera y, conociendo cuáles son los píxeles que pertenecen a la zona navegable, se plantea el cálculo de la trayectoria.

Se diseña como una operación sencilla, en la que se toman los píxeles que corresponden a la clase *Camino*. Con ellos, simplemente se selecciona el píxel central como punto perteneciente a la trayectoria. Esto se hace para todo el largo de píxeles de la imagen, 561, de manera que se obtienen la misma cantidad de puntos marcados como ruta. En la figura contigua se muestran los puntos centrales de la clase de interés, como se ha mencionado. Para continuar con la misma referencia, se toma la imagen del mismo instante que en la Figura 46

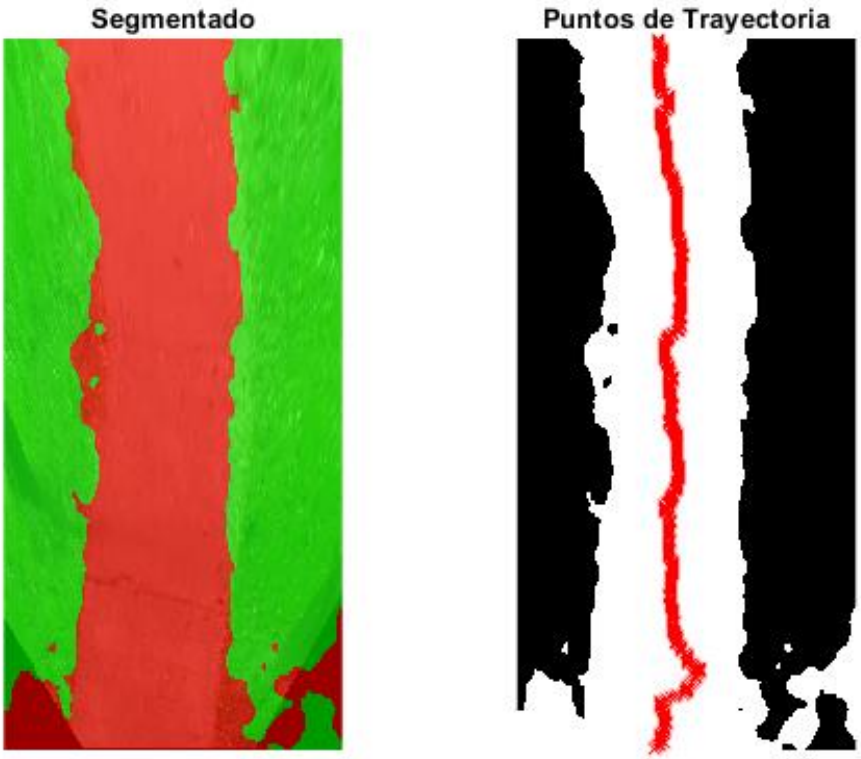


Figura 47 Cálculo de puntos centrales de la clase Camino

Se extrae la distinción de clases realizada a través de la segmentación semántica, representándolas en blanco y negro. Los píxeles seleccionados como parte de la trayectoria de se dibujan en rojo. Se observa que la unión de los puntos centrales de la etiqueta *Camino*, forma el trazado que se toma como trayectoria. Además, se ha recortado la parte inferior negra que se obtenía con la vista de pájaro, buscando mayor precisión. Se conoce que estas zonas se han clasificado como *No_Camino*, y como se mencionó, no afectan en el cálculo de la ruta.

Se unen dichos puntos con una línea para visualizar mejor la trayectoria. Así mismo, se dibuja sobre la propia imagen del entorno, como se ve en la siguiente figura, donde se muestran dos líneas azules. La de color claro representa el trazo que se ha obtenido directamente de los píxeles centrales. Se aprecia cómo se generan curvas y picos que harán que el vehículo autónomo tarde más en recorrer la ruta, por lo que se propone una interpolación. Con la línea azul oscura se dibuja la trayectoria habiendo aplicado dicha interpolación, de manera que se consigue un recorrido más óptimo y suave.

Figura 48 Trayectoria a seguir por AGV

Se considera que la planificación de trayectoria es correcta, valorando que el AGV podrá ser capaz de navegar sobre la zona adecuada. Respecto a la propia ruta, pese a ser de simple obtención, se da como apta, ya que cumple con el objetivo del "path planning".

5 Análisis de los resultados

Se comprueban los resultados de los algoritmos diseñados, analizando el grado cumplimiento de cada objetivo. De igual modo al que se ha hecho en el apartado *4.1 Diseño del algoritmo de navegación,* para la evaluación, se distingue entre el algoritmo de navegación y el algoritmo de planificación de trayectoria y localización. Por último y para completar la navegación y localización autónoma del AGV, se han unido ambas partes, resultado que también se analizará.

5.1 Resultados del algoritmo de navegación

El algoritmo de navegación diseñado se compone de un control lateral y un control longitudinal, como ya se ha visto. Ambos funcionan en conjunto para poder realizar un seguimiento adecuado de la trayectoria. Para comprobar su desempeño, se presenta el resultado obtenido en el "path following" de una ruta sinusoidal, como se puede observar en la imagen contigua.

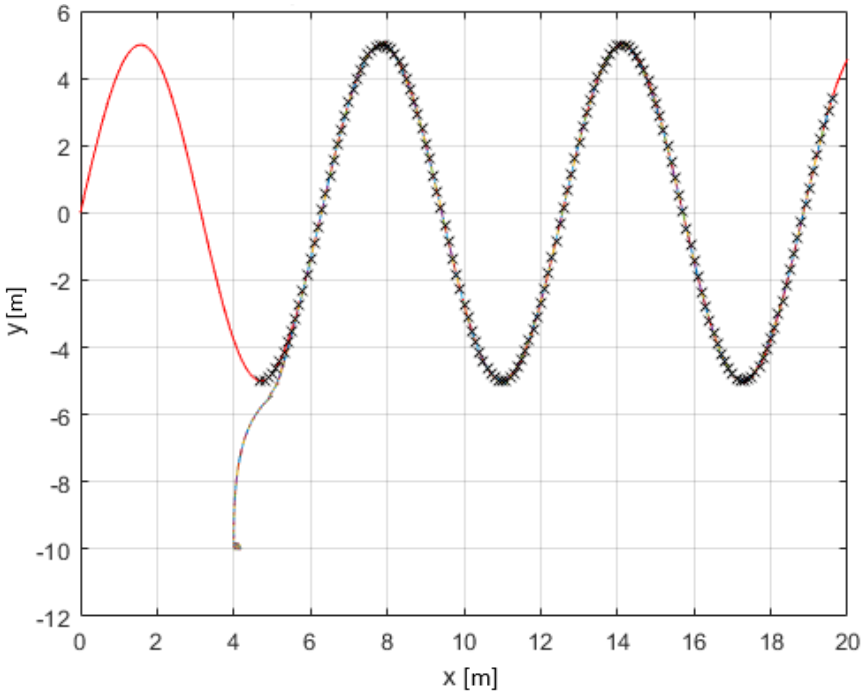
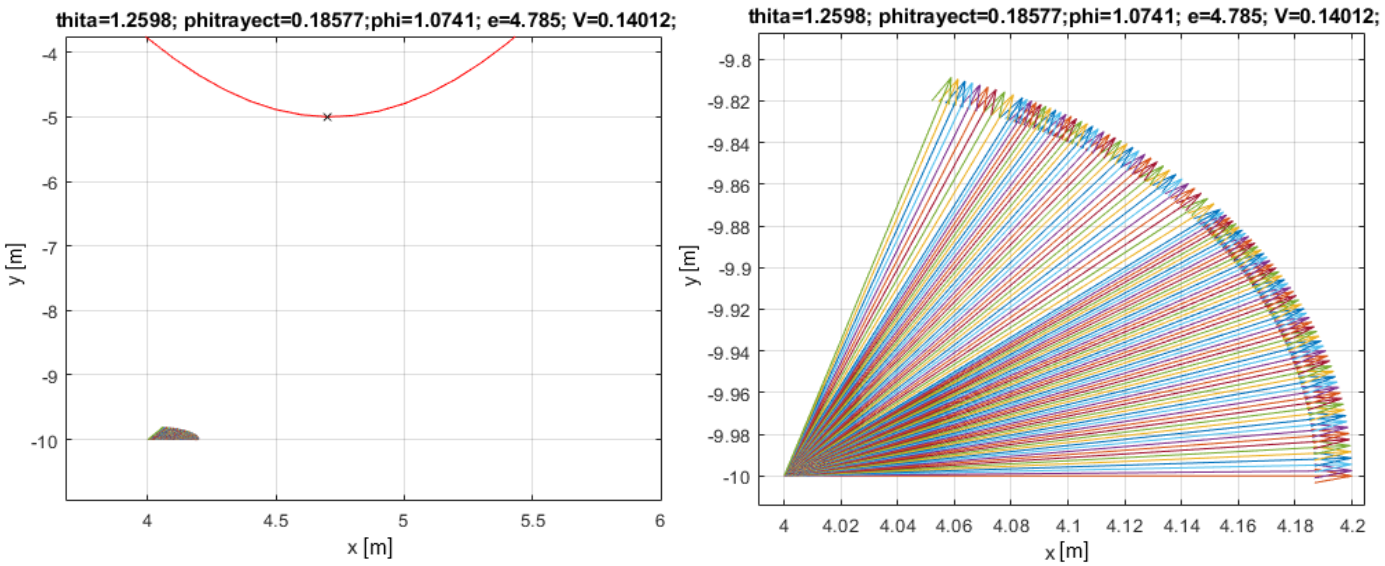


Figura 49 Resultados de algoritmo de navegación

En la figura superior, la línea roja representa la trayectoria, la de colores la dirección y posición del robot móvil y las cruces negras la siguiente posición a alcanzar por este. Como se observa, el AGV está posicionado con un valor de $\overrightarrow{pose}$ inicial igual a [4, -10, 0]. Desde ese punto, el vehículo autónomo industrial es capaz de redireccionarse y dirigirse hacia la trayectoria. A medida que se va acercando, se ajusta para no sobrepasar la ruta, obteniendo como resultado un seguimiento de la trayectoria muy adecuado. A primera vista, se cumple con el objetivo del algoritmo de navegación, pero con intención de realizar un estudio más completo, se descompone la simulación en instantes diferentes, reparando en valores importantes. Estos valores se corresponden a parámetros ya conocidos, siendo estos, el ángulo del AGV **θ** (rad), el ángulo de la trayectoria $\boldsymbol{\varphi_{trayect.}}$ (rad), la diferencia entre ambos ángulos **φ** (rad), el error de posicionamiento **e** (m) y la velocidad lineal **V** (m/s).

5.1.1 Comienzo de ejecución, redireccionamiento

Con el inicio de la simulación de la navegación, esta es la primera situación que se genera:



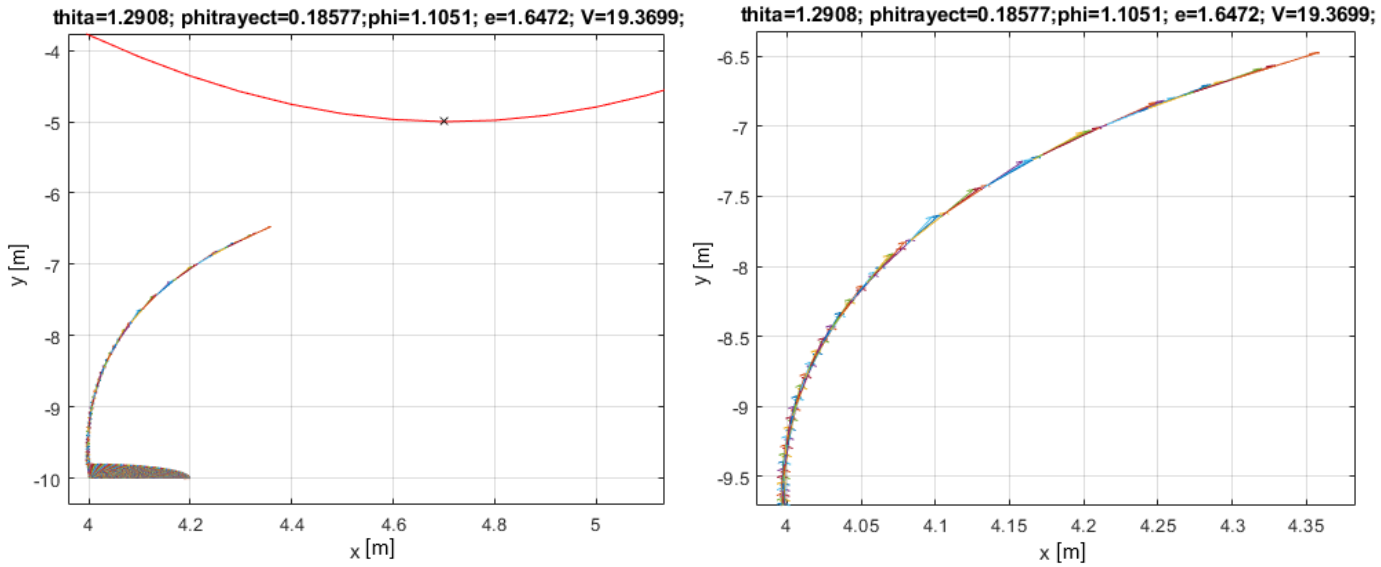
Unidades: thita (rad), phitrayect (rad), phi (rad), e (m), V (m/s)

Figura 50 Situación inicial de navegación de AGV

En la gráfica izquierda de la figura superior, se observa como el algoritmo es capaz de calcular la posición más cercana de la trayectoria en referencia al AGV, señalizada con una cruz negra. De este modo, el primer objetivo es alcanzar dicho punto. Una de las cifras superiores indica el error de posicionamiento, con un valor de **e** = 4,785 m. Independientemente del direccionamiento inicial del vehículo autónomo, en este caso 0 rad, se requiere que se oriente a $\frac{\pi}{2}$ rad, con el fin de corregir el valor considerable del error de posicionamiento. De esta manera, será posible alcanzar la trayectoria prontamente.

Desde la $\overrightarrow{pose}$ inicial se aprecia como el AGV empieza a girar. Se visualiza con más detalle en la segunda gráfica de la *Unidades: thita (rad), phitrayect (rad), phi (rad), e (m), V (m/s)*

Figura 50. El vehículo autónomo únicamente gira, teniendo una velocidad lineal prácticamente nula. Una vez se posicione a $\frac{\pi}{2}$ rad, comienza a avanzar, tal y como se ve en las



gráficas inferiores.

Unidades: thita (rad), phitrayect (rad), phi (rad), e (m), V (m/s)

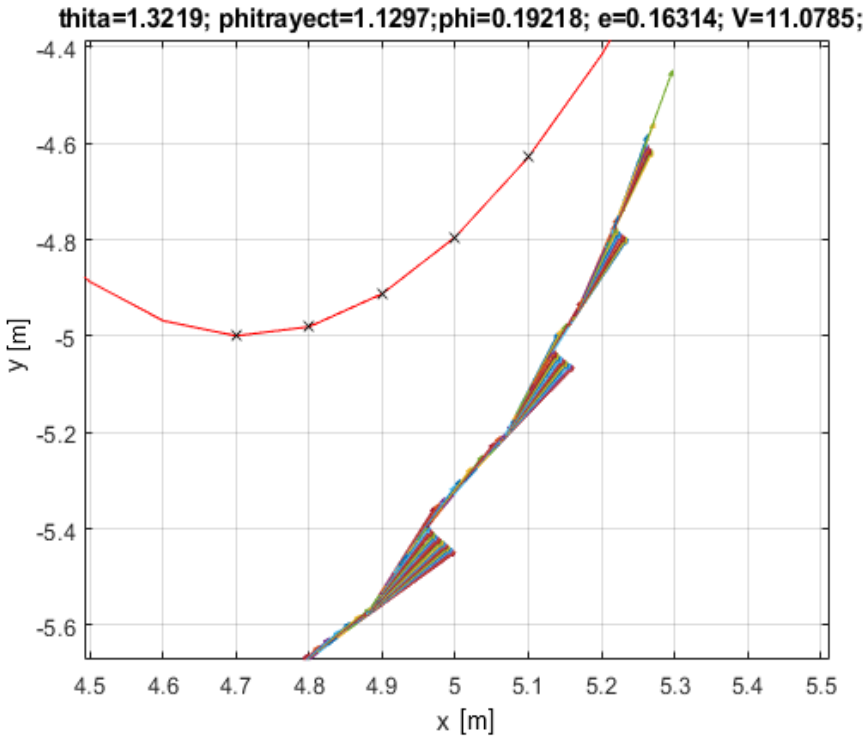
Figura 51 Simulación de comienzo de avance de AGV

La primera gráfica muestra cómo tras alcanzar los 90º, el vehículo autónomo comienza a aproximarse a la trayectoria. Con la disminución del error de posicionamiento, el valor de **θ** también comienza a variar, ajustándose más al ángulo de la propia trayectoria y generando la curva que se observa. También puede apreciar como la velocidad lineal ha aumentado con respecto al valor de la Figura 50. Al prestar atención a la segunda gráfica, se puede percibir este aumento de velocidad, ya que las flechas cada vez están más separadas. Conociendo además que siempre son del mismo tamaño y salen desde el AGV, es obvio dicho aumento de rapidez. De igual modo, pese a ir rápido, se puede seguir apreciando el ajuste de direccionamiento, con la dirección a la que señalan las flechas.

Por lo tanto, el funcionamiento de esta primera situación es el deseado, cumpliendo con lo planteado en su diseño.

5.1.2 Aproximación a trayectoria

Cuando el vehículo autónomo industrial se aproxima a la trayectoria, no necesariamente el punto más cercano a esta se debe de mantener, tal y como se aprecia en la siguiente gráfica. Al igual que en la situación anterior, a medida que el error de posicionamiento **e** disminuye, también se requiere que lo haga el error de alineamiento **φ**. Se observa, a través de las flechas, ese direccionamiento de corrección, que vuelve a necesitar un ajuste cuando el punto más cercano a la trayectoria cambia. De esta manera, se va consiguiendo que ambos errores sean lo más pequeños posibles.



Unidades: thita (rad), phitrayect (rad), phi (rad), e (m), V (m/s)

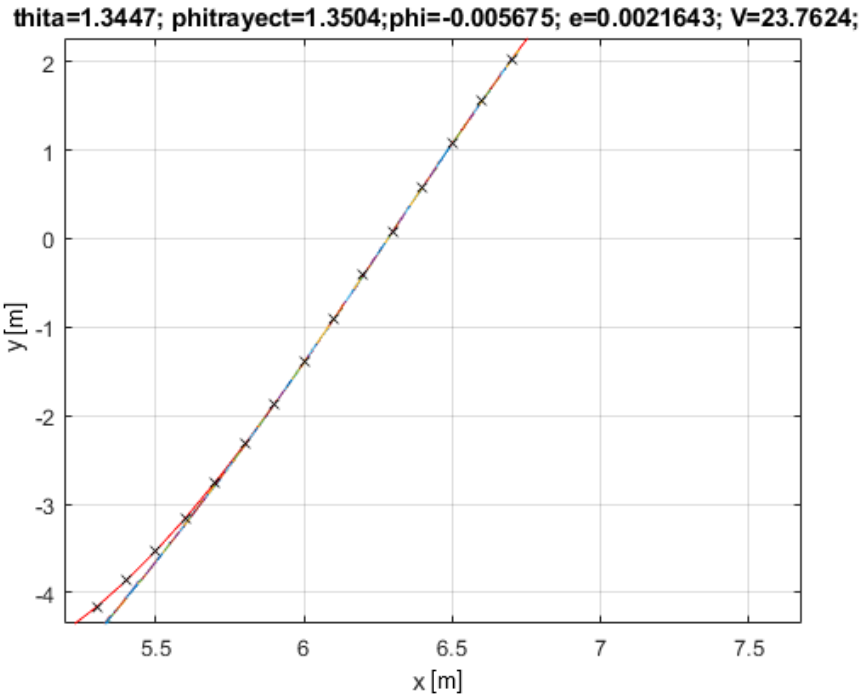
Figura 52 Simulación de aproximación a trayectoria

El valor de ángulo del vehículo **θ** y el de la trayectoria $\boldsymbol{\varphi}_{\mathbf{trayect}}$. cada vez son más similares, disminuyendo el error de alineamiento que tiene un valor de **φ** = 0,19218 rad. El error de posición **e**, también se va aproximando a 0, ya que, tras esta situación, el AGV estará posicionado encima

de la trayectoria. En cuanto a la velocidad lineal, disminuye ligeramente respecto al anterior escenario, debido a que tiene que direccionarse ligeramente para reducir el valor de **φ**. Nuevamente cumple con lo previsto.

5.1.3 Zona recta

Por la forma de la trayectoria planteada, la primera situación con la que se topa el AGV una vez esté posicionado encima de la ruta, es una recta. El punto más cercano de la trayectoria con respecto al vehículo industrial autónomo se va actualizando, tomado como referencia la dirección con la que se quiere recorrer la ruta (véase Figura 53).



Unidades: thita (rad), phitrayect (rad), phi (rad), e (m), V (m/s)

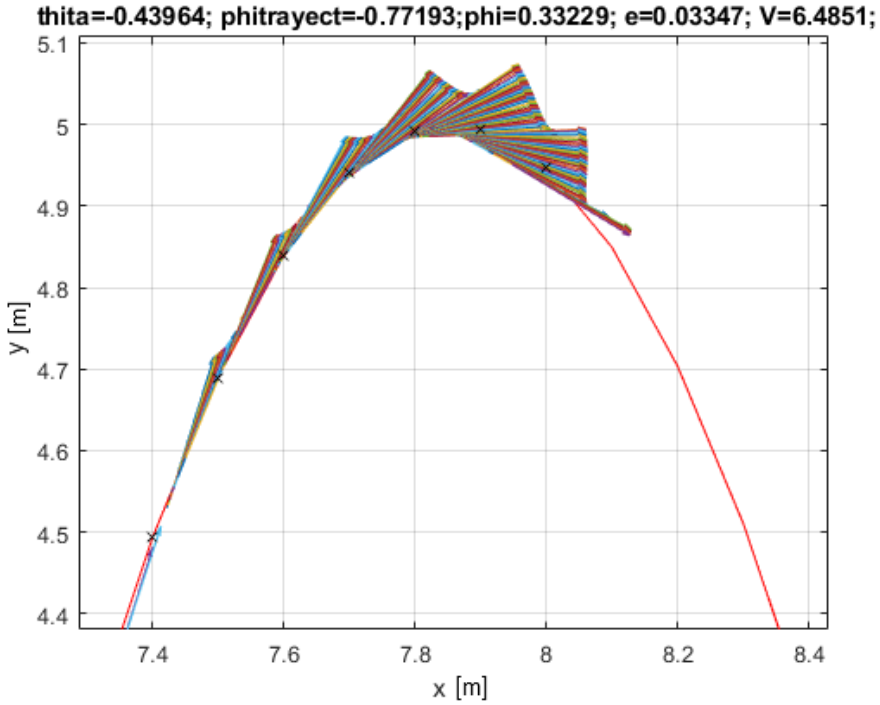
Figura 53 Simulación de seguimiento de trayectoria recta

Los datos reflejan como ambos errores son prácticamente nulos, efectuando un seguimiento muy preciso de la trayectoria. Esto se debe a que el robot móvil está situado perfectamente sobre la trayectoria, y tal y como se aprecia, los valores de los ángulos del vehículo **θ** y de la trayectoria $\boldsymbol{\varphi_{trayect.}}$ son prácticamente iguales. En esta situación, la velocidad está limitada por la $\mathbf{V_{nom}}$ fijada en el diseño, por lo que el AGV va aumentando su velocidad lineal con la intención de realizar el recorrido en un tiempo reducido.

Este funcionamiento también resulta correcto, cumpliendo con lo pronosticado para esta circunstancia.

5.1.4 Zona curva

Por último, al llegar al máximo de la sinusoide, el AGV se encuentra con una curva. En esta situación, el vehículo autónomo industrial debe reducir su velocidad para evitar salirse de la ruta fijada. De igual modo, tiene que redireccionarse en función del siguiente punto a alcanzar.



Unidades: thita (rad), phitrayect (rad), phi (rad), e (m), V (m/s)

Figura 54 Simulación de seguimiento de trayectoria en curva

En la gráfica superior se aprecia como el error de alineamiento **φ** no es nulo, además de que ha aumentado su valor con respecto a la Figura 53. Se debe a la forma de la trayectoria, que hace variar el valor de $\boldsymbol{\varphi_{trayect.}}$ en cada iteración. Tal y como muestran las flechas, el AGV tiene la necesidad de corregir su ángulo **θ**, para reducir dicho error. Con el objetivo de realizar esto de la manera más ajustada posible, la velocidad lineal disminuye, alcanzando el valor mínimo de todo el trayecto en la zona en la que la curva es más pronunciada. Al avanzar, el vehículo autónomo va recuperando velocidad y se observa un valor de **V** = 6,4851 m/s. Sin embargo, este escenario no afecta al error de posicionamiento **e**, que sigue teniendo un valor prácticamente nulo, cumpliendo con las expectativas.

De esta manera, se comprueba que el resultado obtenido es satisfactorio, ya que, ante cualquier situación, el AGV es capaz de responder como se había planteado. Se prueba así que el vehículo autónomo será capaz de realizar un buen seguimiento independientemente de la forma de la trayectoria. Además, se puede apreciar que la problemática descrita en el apartado *4.1.1.3 Distinción del signo del error* se ha corregido debidamente.

Para demostrar su correcto funcionamiento frente a cualquier tipo de recorrido y su independencia de la posición inicial del vehículo autónomo, se muestran diferentes trayectorias y posiciones del AGV respecto a esta. En las simulaciones se visualiza su acertada navegación en todos los casos.

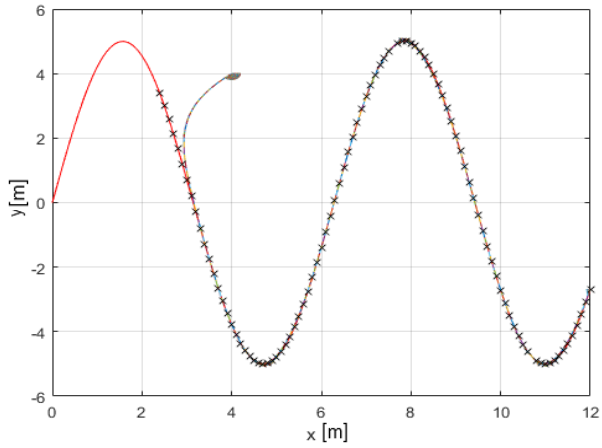
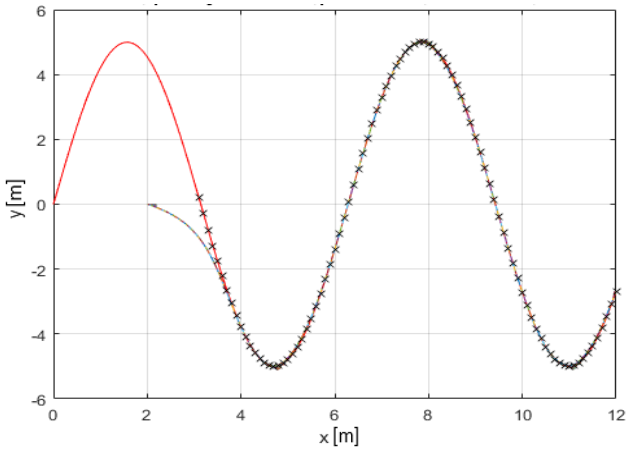
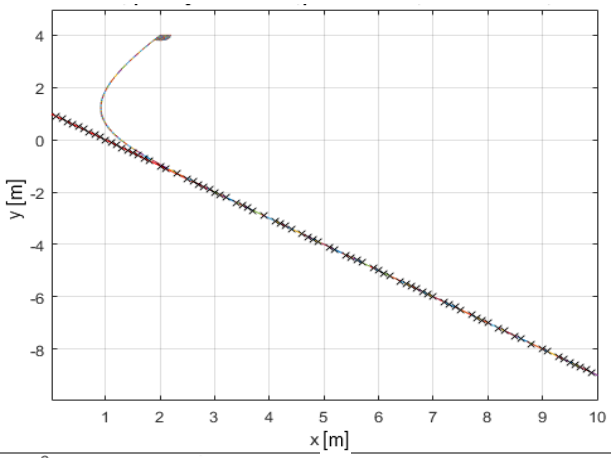
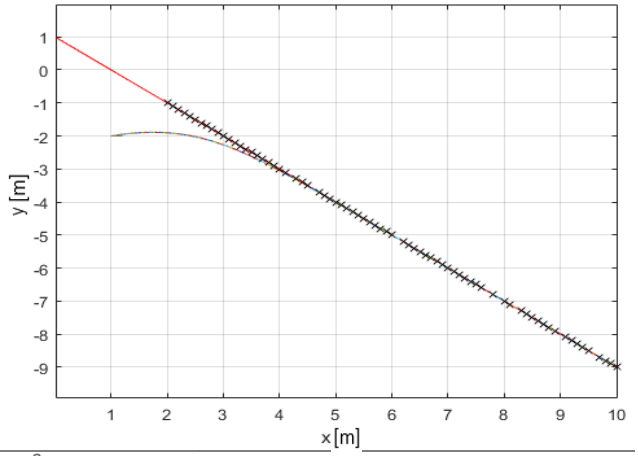
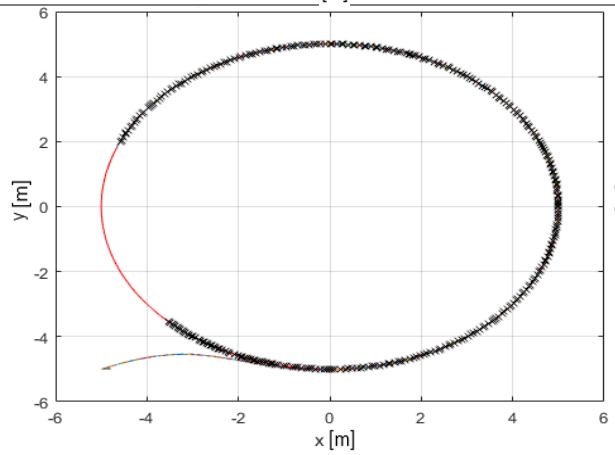
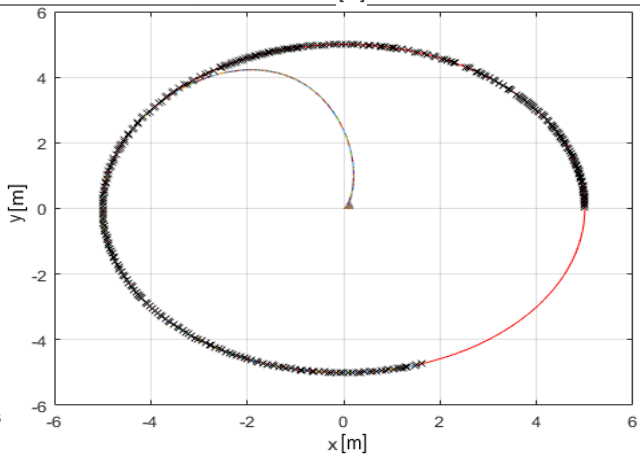
Trayectoria	$\overrightarrow{pose}$ encima	$\overrightarrow{pose}$ debajo
Sinusoidal		
Inclinación negativa		
Circular		

Tabla 1 Simulación de navegación sobre diferentes trayectorias

Se percibe que, con independencia al signo del error de posicionamiento, el algoritmo de navegación es capaz de recorrer cualquier tipo de trayectoria, adaptándose perfectamente a estas. El resultado es adecuado en vista de que se demuestra la robustez del diseño ante cualquier situación.

5.2 Resultados del algoritmo de planificación de trayectoria y localización

En lo referente a la localización del vehículo industrial autónomo, se basa en las medidas de **e** y **φ** que realiza el algoritmo de navegación con respecto a la ruta obtenida del "path planning". El resultado cumple con lo previsto, pudiendo situar al AGV respecto a la trayectoria.

En el apartado *4.2 Diseño del algoritmo de planificación de trayectoria y localización*, se ha visto que la obtención de la ruta se hace a través de una red neuronal. Se realiza una segmentación semántica de las imágenes que obtiene el AGV, de manera que se es capaz de reconocer la zona navegable. La primera necesidad, como ya se mencionó en dicho apartado, es la de transformar la imagen que obtiene el vehículo autónomo a una vista de pájaro. Por lo general, esto se ejecuta de manera correcta, como se ha ido observando en diferentes figuras. Las imágenes se toman con una cámara sin ningún tipo de característica especial, por lo que, en algunas zonas del recorrido dicha modificación no se hace de manera correcta (véase Figura 55).

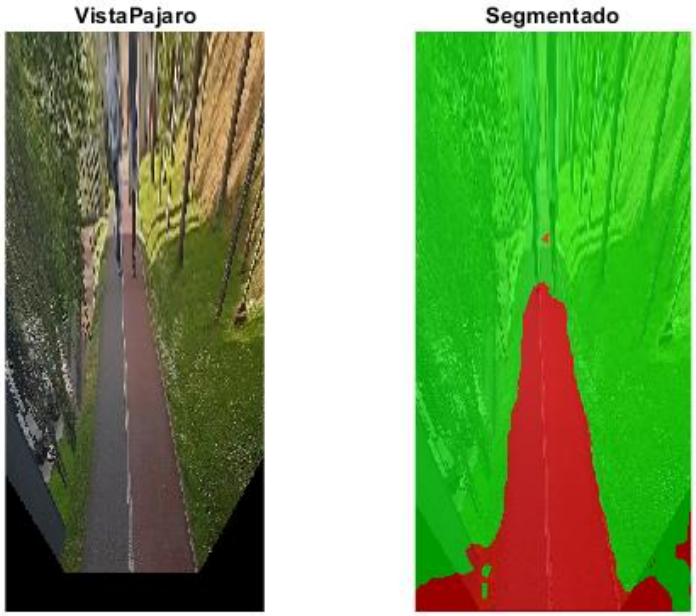


Figura 55 Problemática con vista de pájaro

Se aprecia que la transformación no llega a ser una vista de pájaro por completo, además de que se genera una distorsión importante en la zona superior de la imagen. Esto implica que no se puedan tomar las medidas espaciales reales pese a que el segmentado sea aceptable. El no contar con un valor adecuado de dichas medidas implica un cálculo de errores inexacto, perjudicando a la navegación en una situación real. En conclusión, se considera una técnica muy válida para tomar fotografías del entorno con unos resultados prometedores, pero se ve la necesidad de usar una cámara más adecuada a la aplicación, como podría ser una con una lente de ojo de pez.

En cuanto a la segmentación semántica, se obtienen buenos resultados cumpliendo con el objetivo que se establece. Sin embargo, se ve la necesidad de entrenar más la red neuronal convolucional o incluso crear más etiquetas de segmentos, por ejemplo.

Una de las problemáticas apreciable en las imágenes, es la de la climatología, que implica tener zonas con sombras y zonas más iluminadas por el sol. Como se ve en la siguiente figura, se segmentan erróneamente zonas pertenecientes a la trayectoria. En caso de que la mala clasificación no sea muy grande y se mantenga un trazo continuo de zona navegable no debería de resultar problemático. En cambio, si dicha zona transitable se divide, la trayectoria se seguiría calculando, pero incorrectamente, ya que se atravesarían zonas que no se debieran.

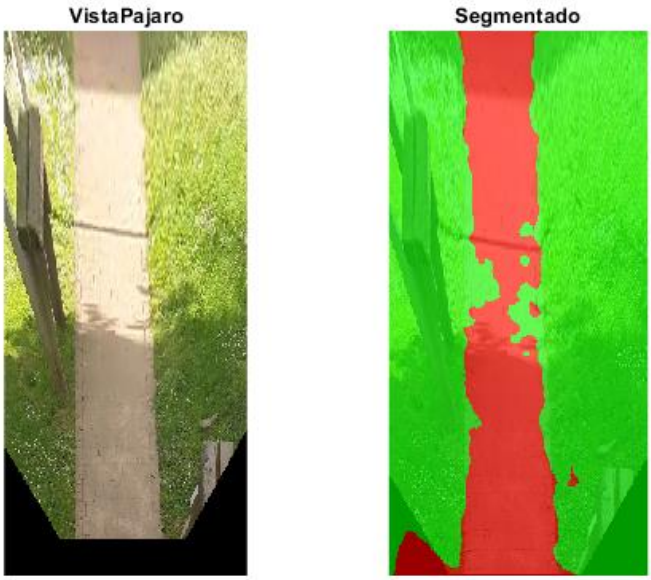


Figura 56 Problemática con luces

Por otra parte, se pueden encontrar también problemas de clasificación debidos a las tonalidades de los obstáculos. Como en el caso que se presenta en la siguiente figura, si un obstáculo resulta coincidir en color con la trayectoria, habría un problema de clasificación. Al fin y al cabo, la red neuronal convolucional planteada, se basa en matrices RGB, gracias a las que distingue los colores de cada píxel. En consecuencia, una clasificación incorrecta implica, a su vez, un cálculo de trayectoria erróneo.

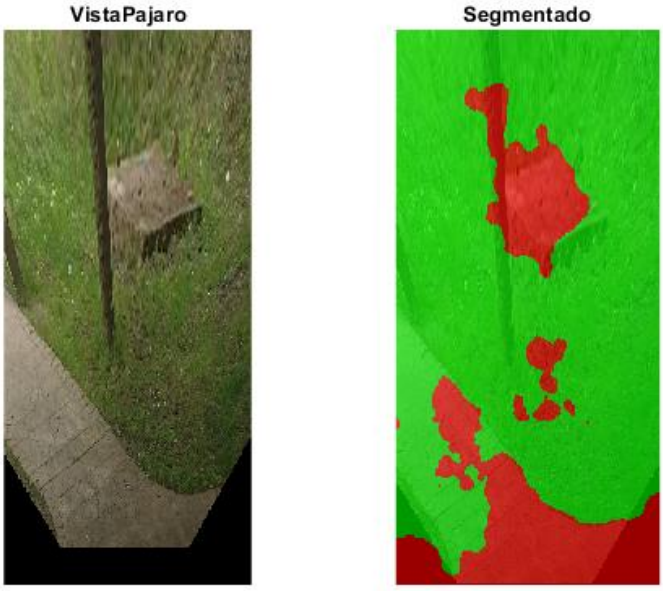


Figura 57 Problemática de tonalidades similares

En general, la segmentación semántica proporciona resultados acertados y es muy útil para hacer el "path planning" en cualquier tipo de entorno. Las problemáticas mostradas podrían resolverse, como bien se ha dicho, con más datos de entrenamiento o con una segmentación más específica y completa. Además, podría probarse con otro tipo de redes y arquitecturas más complejas, pero el resultado para esta aplicación es el esperado.

En cuanto al cálculo de la trayectoria, ya se aclaró que simplemente se toma el píxel central de la zona navegable, de forma que el resultado dependerá en su totalidad de la obtención y segmentación semántica de las imágenes.

5.3 **Resultados de la unión de algoritmos**

El objetivo de este trabajo era el diseño de algoritmos de localización y navegación para que un vehículo industrial pudiese moverse con total autonomía. Uniendo, tanto el algoritmo de navegación, como el de localización y "path planning", se consigue un resultado muy favorable.

Con el objetivo de verificarlo a través de simulaciones, se ha planteado que el vehículo comience directamente sobre la trayectoria, pero con un ángulo **θ** de 0 radianes. De esta manera, se obtienen los resultados que se pueden visualizar en las siguientes imágenes.

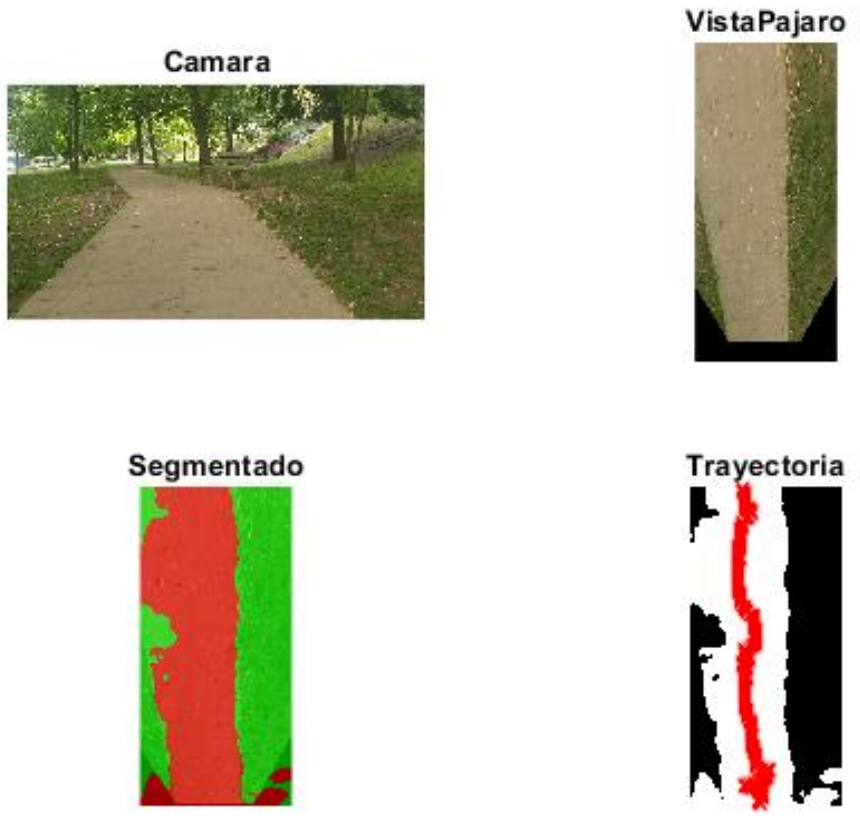


Figura 58 Preparación de trayectoria

La Figura 58 muestra el proceso desde que el AGV toma la imagen, hasta que se extraen los puntos centrales de la zona navegable. Una vez obtenida la trayectoria y habiendo realizado la interpolación, se aplica el algoritmo de navegación, con el fin de que efectúe el seguimiento de la ruta calculada.

Figura 59 Algoritmos de navegación y localización unidos

La línea roja que se presenta sobre la interpolación de la trayectoria, en azul oscuro, es el recorrido que el AGV realiza valiéndose del algoritmo de navegación. Se aprecia que el vehículo autónomo es capaz de continuar perfectamente la ruta planificada, sin desviarse en ningún momento de esta y adaptándose a todas las curvas que se generan.

Se cumple así con la idea inicial de crear algoritmos que den autonomía a un vehículo industrial, con un resultado satisfactorio que cumple todos los objetivos planteados.

6 Conclusiones

El presente trabajo parte de la idea de buscar soluciones apropiadas para la navegación y localización de un AGV. Como se ha podido ver en el apartado *3 Análisis del estado del arte*, es una tecnología que aún requiere de estudio, con el fin de conseguir la máxima precisión y fiabilidad.

En primer lugar, el enfoque de vehículos autónomos resultaba desconocido, por lo que ha resultado necesario familiarizarse con él. El estudio y enseñanza de técnicas asociadas a ello junto con el estado del arte, ha resultado imprescindible para enfocar el trabajo desde el punto de vista que requería. Como se ha comprobado, el estado del arte engloba tres sistemas completamente diferentes pero necesarios para llegar al objetivo final.

Respecto a la navegación, partir de una base como es el algoritmo Stanley, y darle otra perspectiva tiene su complejidad. Además, el enfoque de estabilidad para asegurar que la solución dada es adecuada no es fácil de plantear. Para ello, se deben de probar diferentes alternativas hasta llegar a la solución que demuestre su robustez, pudiendo así también fijar los valores de los parámetros necesarios y darles sentido. Ha resultado ser de las tareas más complicadas de formular y laboriosas para resolver, debido a todas las vertientes desde las que se puede enfocar.

El objetivo fijado al comienzo del trabajo era el de diseñar un algoritmo de navegación que pudiese realizar, tanto un control lateral, como uno longitudinal. Se ha cumplido con creces, obteniendo muy buenos resultados como se ha podido apreciar en anteriores apartados. Tal y como se ha mencionado, la dificultad principal ha recaído sobre el cálculo de la estabilidad, pero, en vista de lo obtenido, se puede asegurar que el análisis se ha hecho de forma correcta.

Continuando con la localización, no diseñar un algoritmo específico para ello y valerse de los datos obtenidos a partir del resto de algoritmos, es otro criterio menos habitual. En ocasiones, se tiende a la solución de generar un nuevo diseño, sin darse cuenta de que todos los modelos tienen mucha información, pudiendo obtener todos los datos que se necesiten con cálculos simples. En este caso, la combinación de ambos algoritmos proporciona las referencias necesarias.

Con este planteamiento, el objetivo de obtener el valor de la $\overrightarrow{pose}$ del robot móvil se cumple, con medidas obtenidas desde la ruta y los errores, en vez de con el uso de sensores.

A la hora del cálculo de trayectoria, los sensores también suelen ser el método más recurrente. Por una parte, debido a la precisión que tienen y por otra, por su fácil implementación en cualquier campo. Sin embargo, este enfoque de reconocimiento del entorno a través de la visión artificial es muy prometedor, además de que las redes neuronales son muy flexibles. Esto genera que un simple aprendizaje haga al AGV adaptarse a cualquier zona, sin necesidad de cambios de sensórica o reajuste de parámetros.

Este último objetivo se planteó con diferentes alternativas, como es, por ejemplo, el uso de transformadas de Hough para detectar las líneas de la zona navegable y en base a ellas generar la trayectoria. Tras diferentes pruebas se llegó a la conclusión de la necesidad de entrenar una red que realizase la segmentación semántica y así obtener la información necesaria del entorno. Se ha cumplido con lo esperado, pero sí se ha percibido la necesidad de indagar en el tema con el fin de asegurar más precisión.

En general, el trabajo ha cumplido con su objetivo, que era crear un algoritmo para la libre navegación de un vehículo industrial autónomo.

6.1 **Líneas futuras**

El primer trabajo a corto plazo que se puede llevar a cabo, es el de la implementación del algoritmo diseñado sobre el AGV de la escuela de ingeniería de Vitoria-Gasteiz, para el cual ha sido propuesto. Esto demostrará si la navegación y localización planteada es buena y se podrán ajustar valores que se requieran, como la velocidad, por ejemplo.

Su implementación en un prototipo industrial real puede mostrar carencias a la hora de la ejecución del algoritmo en tiempo real, tema que queda pendiente de optimizar. Una implementación en tiempo real implica un análisis del consumo de recursos, que no se ha abarcado este proyecto.

Por otra parte, es obvia la necesidad de ensayos con nuevas redes o configuraciones de estas que puedan llevar a una mejor solución a la hora de detectar el entorno navegable. Quizás, la red neuronal Resnet-18 que se ha usado queda limitada para esta aplicación aun con un entrenamiento más intensivo, por lo que ensayos con otro tipo de redes pueden aportar grandes avances y una idea clara de las características de red necesarias.

Realizar un análisis de explicabilidad [43] por medio de la técnica LIME(local interpretable model-agnostic) [44], puede aclarar si los errores de la segmentación semántica se deben a que las imágenes siempre son muy similares, esto es, con una trayectoria prácticamente lineal. Esto permitirá conocer en que se basa esta red neuronal en concreto para decidir si es de una clase u otra. Por ejemplo, es posible que solo considere píxeles de la clase *Camino* si están centrados en la imagen. Por lo tanto, resultaría interesante indagar en el tema de la interpretación del aprendizaje automático [45].

En cuanto a los propios algoritmos diseñados, se puede llevar a cabo el estudio de la incorporación de un método de parada. Actualmente, si el AGV llega al final de una trayectoria, gira 180º y realiza la trayectoria inversa, por lo que habría que plantear una política de parada que contemple esta situación.

Además, si la segmentación semántica se hace de manera errónea, es necesario formular alternativas en el algoritmo y que este pueda finalizar el trazo de la trayectoria al detectar que la zona de *Camino* es incompleta.

7 Bibliografía

[1] C. Kilic, N. Ohi, Y. Gu, and J. Gross, "Slip-Based Autonomous ZUPT through Gaussian Process to Improve Planetary Rover Localization," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4782–4789, 2021.

[2] S. Y. Chen, "Kalman filter for robot vision: A survey," *IEEE Trans. Ind. Electron.*, vol. 59, no. 11, pp. 4409–4420, 2012.

[3] C. Gentner, T. Jost, W. Wang, S. Zhang, A. Dammann, and U. C. Fiebig, "Multipath Assisted Positioning with Simultaneous Localization and Mapping," *IEEE Trans. Wirel. Commun.*, vol. 15, no. 9, pp. 6104–6117, 2016.

[4] P. Yang and W. Wu, "Efficient particle filter localization algorithm in dense passive RFID tag environment," *IEEE Trans. Ind. Electron.*, vol. 61, no. 10, pp. 5641–5651, 2014.

[5] J. L. Carrera Villacres, Z. Zhao, T. Braun, and Z. Li, "A Particle Filter-Based Reinforcement Learning Approach for Reliable Wireless Indoor Positioning," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 11, pp. 2457–2473, 2019.

[6] L. Wang, "Automatic control of mobile robot based on autonomous navigation algorithm," *Artif. Life Robot.*, vol. 24, no. 4, pp. 494–498, 2019.

[7] Q. bin Zhang, P. Wang, and Z. hai Chen, "An improved particle filter for mobile robot localization based on particle swarm optimization," *Expert Syst. Appl.*, vol. 135, pp. 181–193, 2019.

[8] Y. T. Wang, C. C. Peng, A. A. Ravankar, and A. Ravankar, "A single LiDAR-based feature fusion indoor localization algorithm," *Sensors (Switzerland)*, vol. 18, no. 4, 2018.

[9] N. Senin, B. M. Colosimo, and M. Pacella, "Point set augmentation through fitting for enhanced ICP registration of point clouds in multisensor coordinate metrology," *Robot. Comput. Integr. Manuf.*, vol. 29, no. 1, pp. 39–52, 2013.

[10] X. Gao and T. Zhang, "Robust RGB-D simultaneous localization and mapping using planar point features," *Rob. Auton. Syst.*, vol. 72, pp. 1–14, 2015.

[11] K. Naus and Ł. Marchel, "Use of a weighted ICP algorithm to precisely determine USV movement parameters," *Appl. Sci.*, vol. 9, no. 17, 2019.

[12] Y. Gao, S. Liu, M. M. Atia, and A. Noureldin, "INS/GPS/LiDAR integrated navigation system for urban and indoor environments using hybrid scan matching algorithm," *Sensors (Switzerland)*, vol. 15, no. 9, pp. 23286–23302, 2015.

[13] H. Kim, S. Song, and H. Myung, "GP-ICP: Ground Plane ICP for Mobile Robots," *IEEE Access*, vol. 7, pp. 76599–76610, 2019.

[14] J. Clemens, T. Kluth, and T. Reineking, "β-SLAM: Simultaneous localization and grid mapping with beta distributions," *Inf. Fusion*, vol. 52, no. March 2018, pp. 62–75, 2019.

[15] A. K. Kashyap, D. R. Parhi, M. K. Muni, and K. K. Pandey, "A hybrid technique for path planning of humanoid robot NAO in static and dynamic terrains," *Appl. Soft Comput. J.*, vol. 96, p. 106581, 2020.

[16] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 1, no. May, pp. 341–346, 1999.

[17] L. S. Liu *et al.*, "Path Planning for Smart Car Based on Dijkstra Algorithm and Dynamic Window Approach," *Wirel. Commun. Mob. Comput.*, vol. 2021, 2021.

[18] M. Dobrevski and D. Skocaj, "Adaptive dynamic window approach for local navigation," *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 6930–6936, 2020.

[19] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, 1997.

[20] T. Wang, X. Yan, Y. Wang, and Q. Wu, "A distributed model predictive control using virtual field force for multi-ship collision avoidance under COLREGs," *2017 4th Int. Conf. Transp. Inf. Safety, ICTIS 2017 - Proc.*, pp. 296–305, 2017.

[21] E. Burgos and S. Bhandari, "Potential flow field navigation with virtual force field for UAS collision avoidance," *2016 Int. Conf. Unmanned Aircr. Syst. ICUAS 2016*, pp. 505–513, 2016.

[22] J. Borenstein and Y. Koren, "the Vector Field Histogram - Fast obstacle avoidance for mobile robots," *IEEE J. Robot. Autom.*, vol. 7, no. 3, pp. 278–288, 1991.

[23] C. Ye, "Navigating a mobile robot by a traversability field histogram," *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 37, no. 2, pp. 361–372, 2007.

[24] A. AbdElmoniem, A. Osama, M. Abdelaziz, and S. A. Maged, "A path-tracking algorithm using predictive Stanley lateral controller," *Int. J. Adv. Robot. Syst.*, vol. 17, no. 6, pp. 1–11, 2020.

[25] W. Fei, W. Ziwei, and L. Meijin, "Robot Path Planning Based on Improved Particle Swarm Optimization," *2021 IEEE 2nd Int. Conf. Big Data, Artif. Intell. Internet Things Eng. ICBAIE 2021*, vol. 21, no. 5, pp. 887–891, 2021.

[26] Z. Liu, H. Liu, Z. Lu, and Q. Zeng, "A Dynamic Fusion Pathfinding Algorithm Using Delaunay Triangulation and Improved A-Star for Mobile Robots," *IEEE Access*, vol. 9. pp. 20602–20621, 2021.

[27] L. Cheng, C. Liu, and B. Yan, "Improved hierarchical A-star algorithm for optimal parking path planning of the large parking lot," *2014 IEEE Int. Conf. Inf. Autom. ICIA 2014*, no. July, pp. 695–698, 2014.

[28] L. da Silva Costa and F. Tonidandel, "DVG+A_ and RRT Path-Planners_ A Comparison in a Highly Dynamic Environment _ Enhanced Reader.pdf." .

[29] J. Wang, B. Li, and M. Q. H. Meng, "Kinematic Constrained Bi-directional RRT with Efficient Branch Pruning for robot path planning," *Expert Syst. Appl.*, vol. 170, no. December 2020, p. 114541, 2021.

[30] K. Wei and B. Ren, "A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved RRT algorithm," *Sensors (Switzerland)*, vol. 18, no. 2, 2018.

[31] C. Tang, R. Sun, S. Yu, L. Chen, and J. Zheng, "Autonomous Indoor Mobile Robot Exploration Based on Wavefront Algorithm," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11744 LNAI. pp. 338–348, 2019.

[32] R. F. de Oliveira *et al.*, "Mobile Robot Path Planning Based on a Generalized Wavefront Algorithm.pdf," *Nature*, vol. 388. pp. 539–547, 2018.

[33] U. Sdwk *et al.*, "Mobile robot path planning based on Q-learnig algorithm*," pp. 160–165, 2019.

[34] J. Gao, W. Ye, J. Guo, and Z. Li, "Deep reinforcement learning for indoor mobile robot path planning," *Sensors (Switzerland)*, vol. 20, no. 19, pp. 1–15, 2020.

[35] K. Zheng, J. Gao, and L. Shen, "UCAV Path Planning Algorithm Based on Deep Reinforcement Learning," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11902 LNCS. pp. 702–714, 2019.

[36] D. Teso-Fz-Betoño, E. Zulueta, A. Sánchez-Chica, U. Fernandez-Gamiz, and A. Saenz-Aguirre, "Semantic segmentation to develop an indoor navigation system for an

autonomous mobile robot," *Mathematics*, vol. 8, no. 5, 2020.

[37] M. C. Olgun, Z. Baytar, K. M. Akpolat, and O. Koray Sahingoz, "Autonomous vehicle control for lane and vehicle tracking by using deep learning via vision," *2018 6th Int. Conf. Control Eng. Inf. Technol. CEIT 2018*, no. October, pp. 25–27, 2018.

[38] M. Bojarski, L. Jackel, B. Firner, and U. Muller, "Explaining How End-to-End Deep Learning Steers a Self-Driving Car." [Online]. Available: https://developer.nvidia.com/blog/explaining-deep-learning-self-driving-car/. [Accessed: 14-Jun-2021].

[39] S. M. Azimi, P. Fischer, M. Korner, and P. Reinartz, "Aerial LaneNet: Lane-Marking Semantic Segmentation in Aerial Imagery Using Wavelet-Enhanced Cost-Sensitive Symmetric Fully Convolutional Neural Networks," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 5, pp. 2920–2938, 2019.

[40] V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology*, 1984th ed. Cambridge: Bradford Book, 1984.

[41] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," *Proc. Am. Control Conf.*, pp. 2296–2301, 2007.

[42] "Semantic Segmentation Using Deep Learning - MATLAB & Simulink - MathWorks España." [Online]. Available: https://es.mathworks.com/help/vision/ug/semantic-segmentation-using-deep-learning.html. [Accessed: 08-Jun-2021].

[43] "Explicabilidad, transparencia, trazabilidad y equidad: no todo es precisión en el uso responsable de la inteligencia artificial - Inspiring Blog." [Online]. Available: http://blogs.tecnalia.com/inspiring-blog/2019/11/14/explicabilidad-transparencia-trazabilidad-equidad-no-precision-uso-responsable-la-inteligencia-artificial/. [Accessed: 17-Jun-2021].

[44] "Understanding model predictions with LIME | by Lars Hulstaert | Towards Data Science." [Online]. Available: https://towardsdatascience.com/understanding-model-predictions-with-lime-a582fdff3a3b. [Accessed: 17-Jun-2021].

[45] "Interpretable Machine Learning." [Online]. Available: https://christophm.github.io/interpretable-ml-book/. [Accessed: 17-Jun-2021].

Anexos

Anexo I: Pliego de condiciones

Se autoriza a la Universidad del País Vasco/Euskal Herriko Unibertsitatea, al departamento de Sistemas y Automática de ella y al profesor Ekaitz Zulueta Guerrero, para hacer uso y modificaciones, tanto del presente documento, como del código diseñado para llevar a cabo el proyecto.

Anexo II: Código

A lo largo del desarrollo del proyecto se ha visto la necesidad de hacer uso de diferentes códigos, con el fin de avanzar en el desarrollo del algoritmo de localización y navegación para vehículos autónomos industriales. A continuación, se adjuntan dichos códigos, clasificándolos por su utilidad. Recordemos que el diseño se ha realizado en Matlab.

Código para visualizar problemática del signo del error

Por medio del archivo "EHUControlLateralProblemaSignoError.m", se puede visualizar la problemática con el signo del error vista en el apartado *4.1.1.3 Distinción del signo del error*.

EHUControlLateralProblemaSignoError.m

```
clc
clear all
close all

%% Ibilbide bat definituko dugu
%Zuzen bat:
% ax+by=c
a=-5;
b=5;
c=5;
xtrajectory=[-1:0.1:7];
ytrajectory=(c-a*xtrajectory)/b;
plot(xtrajectory,ytrajectory,'r')
grid on
hold on
xlabel('x [m]')
ylabel('y [m]')

%% Kontrol lateraleko parametroak
K1=100;
K2=0.5;
V=10.2; %Translazio abiadura
thitahas=0; %Hasierako norabide angelua
xhas=0;
yhas=0;

%% Simulazioaren parametroak
Niter=200; %Zenbat alditan simulatuko dugun
DeltaT=0.01; %Integrazio urratsa
thita=zeros(1,Niter);
phi=zeros(1,Niter);
w=zeros(1,Niter);
error=zeros(1,Niter);
x=zeros(1,Niter);
y=zeros(1,Niter);
thitaIbilbidea=zeros(1,Niter); %Phitrayect
x(1)=xhas;
y(1)=yhas;
thita(1)=thitahas;

%% Loop
for k=1:Niter-1
    Kokapena=[x(k);y(k)];
```

```
    Ibilbidea=[xtrajectory;ytrajectory];
    [error(k),thitaIbilbidea(k),errorangle]=Ferror(Kokapena,Ibilbidea);
    thitaabs=thita(k)-2*pi*floor(thita(k)/(2*pi));%thita absolututan
    phi(k)=(thitaabs-thitaIbilbidea(k))*sign(errorangle);%lerroketa errorea
    phic(k)=min([K2*error(k),pi/2]);%phiconsigna
    w(k)=K1*(phic(k)-phi(k));
    %Ibilgailuaren ekuazioa dinamikoak integratu
    thita(k+1)=thita(k)+DeltaT*(w(k));
    x(k+1)=x(k)+V*cos(thita(k))*DeltaT;
    y(k+1)=y(k)+V*sin(thita(k))*DeltaT;
    quiver(x(k),y(k),0.2*cos(thita(k)),0.2*sin(thita(k)),0) %geziak
    izenburua=strcat('phi=',num2str(phi(k)));
    izenburua=strcat(izenburua,' phic=',num2str(phic(k)));
    izenburua=strcat(izenburua,' thitaibilbidea=',num2str(thitaIbilbidea(k)));
    title(izenburua)
    pause(0.1)
end

%% f(e)
function [error,thitaIbilbidea,errorangle]=Ferror(Kokapena,Ibilbidea)
NpuntuIbilbidearenak=size(Ibilbidea,2);
Luzeerak=zeros(1,NpuntuIbilbidearenak);
for i=1:NpuntuIbilbidearenak
    Luzeerak(i)=norm(Ibilbidea(:,i)-Kokapena);%puntu guztietara distantzia
end
[error,RefTxikiena]=min(Luzeerak);%distantzia minimoa
errorvector=(Ibilbidea(:,RefTxikiena)-Kokapena);
errorangle=angle(errorvector(1)+sqrt(-1)*errorvector(2));
IbilbidekoGertuena=Ibilbidea(:,RefTxikiena);
IbilbidekoGertuenarenUrrena=Ibilbidea(:,RefTxikiena+1);
IbilbideGehikuntza=IbilbidekoGertuenarenUrrena-IbilbidekoGertuena;
thitaIbilbidea=angle(IbilbideGehikuntza(1)+sqrt(-1)*IbilbideGehikuntza(2));
end
```

Código para estimar valor de K_1

Las líneas del archivo "RangoDeK1.m", demuestran el desvanecimiento de la integral de la ecuación (42), pudiendo establecer un valor de la constante $\mathbf{K_1}$.

RangoDeK1.m

```
clc
clear all
close all

%% Desvanecimiento de integral
y0=-1;%Condición inicial con valor supuesto
DeltaT=0.0001;
t=[0:DeltaT:0.01];
K1=1000;%Se varia esta línea
V=10;%Se varia esta línea
f=V*sin(pi/2*exp(-K1*t));%integral
plot(t,f,'r')
grid on
xlabel('t [sec]')
VI_K1_t=cumsum(f*DeltaT)+y0;%sumatorio, valor de y
hold on
plot(t,VI_K1_t,'b')
```

Código de estabilidad del sistema cuando $\varphi_{trayect.}$ no es nulo

El código diseñado en el fichero "EHUControlLateralEstabilidadIbilbideJakinBaterako.m" demuestra el cumplimiento de la estabilidad del sistema. Además, es posible cambiar la función sinusoidal por cualquier otro tipo de trayectoria.

EHUControlLateralEstabilidadIbilbideJakinbaterako.m

```
clc
clear all
close all

%% Ibilbide bat definituko dugu
%Sinusoidal bat:
u=[0:0.01:2*pi];
r=0.15;
xtrajectory=u;
ytrajectory=sin(u);
Ibilbidea=[xtrajectory;ytrajectory];
figure(1)
subplot(2,2,1)
plot(xtrajectory,ytrajectory,'r')
grid on
hold on
xlabel('x [m]')
ylabel('y [m]')

%% Kontrol lateraleko parametroak
Simulazioa.K1=1000; %suposatuko dugu berehala norabidean jarriko garela
Simulazioa.K2=0.5;%suposatuz balio hau
Simulazioa.V=0.2; %Translazio abiadura
Simulazioa.thitahas=0; %Hasierako norabide angelua
Simulazioa.xhas=0;
Simulazioa.yhas=0;

%% Simulazioaren parametroak
Simulazioa.Niter=2000; %Zenbat alditan simulatuko dugun
Simulazioa.DeltaT=10; %Integrazio urratsa
Simulazioa.xmin=0;%hay que ajustarlos dependiendo de la función
Simulazioa.xmax=6;
Simulazioa.deltax=0.1;%x urratsa
Simulazioa.ymin=-2;
Simulazioa.ymax=2;
Simulazioa.deltay=0.1;%y urratsa
Simulazioa.xvec=[Simulazioa.xmin:Simulazioa.deltax:Simulazioa.xmax];
Simulazioa.yvec=[Simulazioa.ymin:Simulazioa.deltay:Simulazioa.ymax];
%Zein angelu daukan kokapen guztietarako
phiIbilgailua=zeros(length(Simulazioa.xvec),length(Simulazioa.yvec));

%% Lyapunov hasieraketa
L=zeros(length(Simulazioa.xvec),length(Simulazioa.yvec));
deltaL=zeros(length(Simulazioa.xvec),length(Simulazioa.yvec));

%% Loop
for kx=[1:length(Simulazioa.xvec)]
    for ky=[1:length(Simulazioa.yvec)]
        Kokapena=[Simulazioa.xvec(kx);Simulazioa.yvec(ky)];
        %thitaIbilbidea=Phitrayect
        [error,thitaIbilbidea,RefTxikiena]=Ferror(Kokapena,Ibilbidea);
        phic=atan(Simulazioa.K2*error);
```

```
        phiIbilgailua(kx,ky)=phic+thitaIbilbidea;%phiIbilgailua=Thita
        L(kx,ky)=error^2;
        while abs(Simulazioa.V*cos((phiIbilgailua(kx,ky)))*Simulazioa.DeltaT)>0.01
        ||...
              abs(Simulazioa.V*sin((phiIbilgailua(kx,ky)))*Simulazioa.DeltaT)>0.01
              Simulazioa.DeltaT=Simulazioa.DeltaT/2;
        end
        %Hurrengo puntuaren energia
        xhurrengokoa=Kokapena(1)+Simulazioa.V*cos(phiIbilgailua(kx,ky))...
            *Simulazioa.DeltaT;
        yhurrengokoa=Kokapena(2)+Simulazioa.V*sin(phiIbilgailua(kx,ky))...
            *Simulazioa.DeltaT;
        KokapenaHurrengoa=[xhurrengokoa;yhurrengokoa];
        [errorhurrengokoa,thitaIbilbideah,RefTxikienah]=...
            Ferror(KokapenaHurrengoa,Ibilbidea);
        Lhurrengokoa=errorhurrengokoa^2;
        deltaL(kx,ky)=(Lhurrengokoa-L(kx,ky));
        if deltaL(kx,ky)>0
            subplot(2,2,1)
            plot(Ibilbidea(1),Ibilbidea(2),'r')
        end
         simulazioa.DeltaT=10;
    end
    100*kx/length(Simulazioa.xvec)%ejekuzio portzentaia
end
[xmat,ymat]=meshgrid(Simulazioa.xvec,Simulazioa.yvec);
subplot(2,2,2)
surf(xmat,ymat,L')
xlabel('x')
ylabel('y')
zlabel('L')

subplot(2,2,4)
surf(xmat,ymat,deltaL')
xlabel('x')
ylabel('y')
zlabel('\DeltaL')

%% f(e)
function [error,thitaIbilbidea,RefTxikiena]=Ferror(Kokapena,Ibilbidea)
NpuntuIbilbidearenak=size(Ibilbidea,2);
Luzeerak=zeros(1,NpuntuIbilbidearenak);
for k=1:NpuntuIbilbidearenak
    %puntu guztietara distantzia
    Luzeerak(k)=sqrt((Ibilbidea(1,k)-Kokapena(1))^2+(Ibilbidea(2,k)-Kokapena(2))^2);
end
[errormin,RefTxikiena]=min(Luzeerak);;%distantzia minimoa
errorvector=(Ibilbidea(:,RefTxikiena)-Kokapena);
errorangle=angle(errorvector(1)+sqrt(-1)*errorvector(2));
IbilbidekoGertuena=Ibilbidea(:,RefTxikiena);

if RefTxikiena==length(Luzeerak)
   IbilbidekoGertuenarenUrrena=Ibilbidea(:,1);
   IbilbideGehikuntza=IbilbidekoGertuenarenUrrena-IbilbidekoGertuena;
   thitaIbilbidea=angle(IbilbideGehikuntza(1)+sqrt(-1)*IbilbideGehikuntza(2));
else
   IbilbidekoGertuenarenUrrena=Ibilbidea(:,RefTxikiena+1);
   IbilbideGehikuntza=IbilbidekoGertuenarenUrrena-IbilbidekoGertuena;
   thitaIbilbidea=angle(IbilbideGehikuntza(1)+sqrt(-1)*IbilbideGehikuntza(2));
end
```

```
%errorearen zeinuaren zuzenketa
n=IbilbideGehikuntza/(sqrt(IbilbideGehikuntza(1)^2+IbilbideGehikuntza(2)^2));
errorVectorTxikiena=Ibilbidea(:,RefTxikiena)-Kokapena;
T=(n(1)*errorVectorTxikiena(2)-n(2)*errorVectorTxikiena(1));
error=T;
end
```

Código para calcular valor óptimo de K_2

Para estimar el valor de $\mathbf{K_2}$, se comprueban todos los valores de la constante en una zona acotada. Debido a la extensión del código, se ha dividido en dos archivos, siendo el primero el programa principal "RangoDeK2.m" y segundo, la función a la que realiza la llamada para calcular la función de coste, "EHUControlLateralEvaluacionJ.m".

RangoDeK2.m

```
clc
clear all
close all

%% Ibilbide bat definituko dugu
%Sinusoidal bat:
u=[0:0.01:2*pi];
xtrajectory=u;
ytrajectory=sin(u);
Ibilbidea=[xtrajectory;ytrajectory];
grid on
hold on
xlabel('x [m]')
ylabel('y [m]')

%% Kontrol lateraleko parametroak
Simulazioa.K1=100; %suposatuko dugu berehala norabidean jarriko garela
Simulazioa.K2min=0.01;%K2>0
Simulazioa.K2max=10;
Simulazioa.deltaK2=0.1;
Simulazioa.K2vec=[Simulazioa.K2min:Simulazioa.deltaK2:Simulazioa.K2max];
Simulazioa.V=0.2; %Translazio abiadura
Simulazioa.thitahas=0; %Hasierako norabide angelua
Simulazioa.xhas=0;
Simulazioa.yhas=0;
Simulazioa.errorlim=0.1;

%% Simulazioaren parametroak
Simulazioa.Niter=200; %Zenbat alditan simulatuko dugun
Simulazioa.DeltaT=1; %Integrazio urratsa
Simulazioa.xmin=min(xtrajectory);%hay que ajustarlos dependediendo de la función
Simulazioa.xmax=max(xtrajectory);
Simulazioa.deltax=0.5;
Simulazioa.ymin=min(ytrajectory);
Simulazioa.ymax=max(ytrajectory);
Simulazioa.deltay=0.5;
Simulazioa.xvec=[Simulazioa.xmin:Simulazioa.deltax:Simulazioa.xmax];
Simulazioa.yvec=[Simulazioa.ymin:Simulazioa.deltay:Simulazioa.ymax];
%Zein angelu daukan kokapen guztietarako
phiIbilgailua=zeros(length(Simulazioa.xvec),length(Simulazioa.yvec));
errorValor=zeros(length(Simulazioa.xvec),length(Simulazioa.yvec));
%Puntu guztietatik ibilbidera dagoen errorea
errorSuma=0;
```

```
error=100;%hasieratzeagatik
i=0;

%% Función de cote
J=zeros(length(Simulazioa.xvec),length(Simulazioa.yvec));
Jpromedio=0;
JpromedioAntMax=0;
JpromedioAntMin=100;
MaximoK2=0;
MinimoK2=0;

%% Loop
for k=[1:length(Simulazioa.K2vec)]
    for kx=[1:length(Simulazioa.xvec)]
        for ky=[1:length(Simulazioa.yvec)]
            Kokapena=[Simulazioa.xvec(kx);Simulazioa.yvec(ky)];
            K1=Simulazioa.K1;
            K2=Simulazioa.K2vec(k);
            V=Simulazioa.V;
            xhas=Simulazioa.xvec(kx);
            yhas=Simulazioa.yvec(ky);
            thitahas=Simulazioa.thitahas;
            errorlim=Simulazioa.errorlim;
            Niter=Simulazioa.Niter; %Zenbat alditan simulatuko dugun
            DeltaT=Simulazioa.DeltaT; %Integrazio urratsa
            J(kx,ky)=EHUControlLateralEvaluacionJ...
                (xtrajectory,ytrajectory,K1,K2,V,thitahas,...
                xhas,yhas,Niter,DeltaT,errorlim);
        end
    end
Jpromedio=(1/(kx*ky))*sum(sum(J));

 if Jpromedio>JpromedioAntMax
     MaximoK2=Simulazioa.K2vec(k);
     JpromedioAntMax=Jpromedio;
 elseif Jpromedio<JpromedioAntMin
     MinimoK2=Simulazioa.K2vec(k);
     JpromedioAntMin=Jpromedio;
 end

 plot(Simulazioa.K2vec(k),Jpromedio,'xb')
 hold on
 xlabel('K2')
 ylabel('Jpromedio')
 100*k/length(Simulazioa.K2vec)
end
```

EHUControlLateralEvaluacionJ.m

```
function [J]=EHUControlLateralEvaluacionJ...
    (xtrajectory,ytrajectory,K1,K2,V,thitahas,xhas,yhas,Niter,DeltaT,errorlim)

%% Simulazioaren parametroak
thita=zeros(1,Niter);
phi=zeros(1,Niter);
w=zeros(1,Niter);
error=zeros(1,Niter);
x=zeros(1,Niter);
y=zeros(1,Niter);
thitaIbilbidea=zeros(1,Niter);%phitrayect
x(1)=xhas;
```

```
y(1)=yhas;
thita(1)=thitahas;
errorvector=[];

%% Loop
for k=1:Niter-1
    Kokapena=[x(k);y(k)];
    Ibilbidea=[xtrajectory;ytrajectory];
    [error(k),thitaIbilbidea(k),errorangle,RefTxikiena]=Ferror(Kokapena,Ibilbidea);
    phi(k)=(thita(k)-thitaIbilbidea(k));
    phic(k)=atan(K2*error(k));
    zuzenketa=angdiff(phi(k),phic(k));
    w(k)=K1*zuzenketa;
    %Ibilgailuaren ekuazioa dinamikoak integratu
    DeltaThita=DeltaT*w(k);
    while abs(DeltaThita)>pi/180
        DeltaT=DeltaT/2;
        DeltaThita=DeltaT*w(k);
    end
    thita(k+1)=thita(k)+DeltaThita;
    x(k+1)=x(k)+V*cos(thita(k))*DeltaT;
    y(k+1)=y(k)+V*sin(thita(k))*DeltaT;
    DeltaT=0.1;
    errorvector=[errorvector,error(k)];
    if abs(error(k))<errorlim
       break
    end
end
J=sum(errorvector.^2);
J=sqrt(J)/length(errorvector);%batez-bestekoa
end

%% f(e)
function [error,thitaIbilbidea,errorangle,RefTxikiena]=Ferror(Kokapena,Ibilbidea)
NpuntuIbilbidearenak=size(Ibilbidea,2);
Luzeerak=zeros(1,NpuntuIbilbidearenak);
for i=1:NpuntuIbilbidearenak
    Luzeerak(i)=norm(Ibilbidea(:,i)-Kokapena);%puntu guztietara distantzia
end
[error,RefTxikiena]=min(Luzeerak);%distantzia minimoa
errorvector=(Ibilbidea(:,RefTxikiena)-Kokapena);
errorangle=angle(errorvector(1)+sqrt(-1)*errorvector(2));
IbilbidekoGertuena=Ibilbidea(:,RefTxikiena);

if RefTxikiena==length(Luzeerak)
   IbilbidekoGertuenarenUrrena=Ibilbidea(:,1);
   IbilbideGehikuntza=IbilbidekoGertuenarenUrrena-IbilbidekoGertuena;
   thitaIbilbidea=angle(IbilbideGehikuntza(1)+sqrt(-1)*IbilbideGehikuntza(2));
 else
   IbilbidekoGertuenarenUrrena=Ibilbidea(:,RefTxikiena+1);
   IbilbideGehikuntza=IbilbidekoGertuenarenUrrena-IbilbidekoGertuena;
   thitaIbilbidea=angle(IbilbideGehikuntza(1)+sqrt(-1)*IbilbideGehikuntza(2));
end

%errorearen zeinuaren zuzenketa
n=IbilbideGehikuntza/norm(IbilbideGehikuntza);
errorVectorTxikiena=Ibilbidea(:,RefTxikiena)-Kokapena;
T=n(1)*errorVectorTxikiena(2)-n(2)*errorVectorTxikiena(1);
error=T;
end
```

Código para obtener múltiples imágenes del video

Se ve la necesidad de crear un conjunto de datos para que la red neuronal pueda realizar el entrenamiento. Dicho grupo se obtiene de un video, del cual, cada 4 segundos, se toma una captura. Se lleva a cabo a través del archivo "GuardarImagenes.m", y guarda imágenes en orden numérico.

GuardarImagenes.m

```
clc
close all
clear all

%% Parámetros de cámara
focalLength = [1600 1500];
principalPoint = [1000 800];
height = 2.1798;%altura de cámara
pitch = 14;%ángulo que abarca
distAhead = 30;
spaceToOneSide = 6;
bottomOffset = 3;
outImageSize = [NaN,250];

%% Loop
run=true;
count=0;
mostrar=true;
net_input_size=net.Layers(1).InputSize(1:2);
cmap = ade20kColorMap2;
%Directorio donde se quieren guardar
folder='C:\Users\Desktop\TFMImagenes\imagenes';

vidObj = VideoReader('video1.mp4');%video del que se extraen imágenes
vidObj.CurrentTime = 10;%se coge la primera imagen
Im_original_size=size(vidObj);%coge [1,1]
Im_original_size=[360,640,3];
Im_original_size=Im_original_size(1:2);

% vista pájaro
camIntrinsics = cameraIntrinsics(focalLength,principalPoint,Im_original_size);
sensor = monoCamera(camIntrinsics,height,'Pitch',pitch);
outView = [bottomOffset,distAhead,-spaceToOneSide,spaceToOneSide];
birdsEye = birdsEyeView(sensor,outView,outImageSize);

while(run)
    vidObj.CurrentTime = vidObj.CurrentTime+4;%se coge una imagen cada 4 segundos
    vidFrame = readFrame(vidObj);
    if mostrar
        % vista pájaro transformada
        BEV = transformImage(birdsEye,vidFrame);
        BEV_name=sprintf('BEV_Image_%i.jpg',count);%nombre de imagenes
        BEV_path=fullfile(folder,BEV_name);
        imwrite(BEV,BEV_path);
    end
end
```

Código para realizar entrenamiento de la red neuronal

Matlab ofrece un "LiveScript" de ejemplo [42] que se ha utilizado como base para efectuar el entrenamiento de la red. Este modelo no se correspondía con la segmentación semántica deseada, por lo que se han realizado modificaciones.

SemanticSegmentationUsingDeepLearningExampleAdapted.mlx

Segmentación Semántica utilizando aprendizaje profundo

Configuración

Se debe ejecutar esta línea con el fin de verificar que se tiene instalado el Deep Learning Toolbox ™ para la red Resnet-18.

```
resnet18();
```

Cargar Imágenes

Se cargan las imágenes desde el directorio donde se habían guardado las capturas del video.

```
imgDir = fullfile('C:\Users\Desktop\TFMImagenes\imagenes');
imds = imageDatastore(imgDir);
```

Visualizar una imagen

```
I = readimage(imds,1);
I = histeq(I);
imshow(I)
```

Cargar las imágenes segmentadas por píxeles

Se deben especificar las clases en las que se ha etiquetado el conjunto de datos.

```
classes = [
    "Camino"
    "No_camino"
    ];
```

Se asignan IDs a las clases

```
labelIDs = {
[1]
[2]
};
```

Se utilizan las clases y los ID correspondientes para crear el `pixelLabelDatastore.`

```
labelDir = 'C:\Users\Desktop\TFMImagenes\imagenes\.imageLabelingSession_SessionData';
pxds = pixelLabelDatastore(labelDir,classes,labelIDs);
```

Se lee y muestra una imagen, superponiendo las clases definidas en ella

```
C = readimage(pxds,1);
cmap = camvidColorMap;
B = labeloverlay(I,C,'ColorMap',cmap);
imshow(B)
pixelLabelColorbar(cmap,classes);
```

Las áreas sin etiquetar no tienen pixeles asociados a clases y no se usan en el entrenamiento.

Análisis de estadísticas de conjuntos de datos

Para ver la distribución de etiquetas por pixeles se usa `countEachLabel`.

```
tbl = countEachLabel(pxds);
```

Visualizar la cuenta de pixeles por clase

```
frequency = tbl.PixelCount/sum(tbl.PixelCount);

bar(1:numel(classes),frequency)
xticks(1:numel(classes))
xticklabels(tbl.Name)
xtickangle(45)
ylabel('Frequency')
```

Preparar conjunto de entrenamiento, validación y prueba

```
[imdsTrain, imdsVal, imdsTest, pxdsTrain, pxdsVal, pxdsTest] =
partitionCamVidData(imds,pxds);
```

La división 60/20/20 da estos resultados de cantidad de imágenes por conjunto

```
numTrainingImages = numel(imdsTrain.Files)
numValImages = numel(imdsVal.Files)
numTestingImages = numel(imdsTest.Files)
```

Crear la red

Se utiliza la función deeplabv3plusLayers para crear una red DeepLab v3 + basada en ResNet-18.

```
% Specify the network image size. This is typically the same as the traing image sizes.
imageSize = [561 250 3]; %tamaño de imagenes de vista de pajaro

% Specify the number of classes.
numClasses = numel(classes);

% Create DeepLab v3+.
lgraph = deeplabv3plusLayers(imageSize, numClasses, "resnet18");
```

Equilibrio con ponderación de clases

Se utilizan los recuentos de etiquetas de pixeles con countEachLabel. y se calculan las ponderaciones de cada clase

```
imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
classWeights = median(imageFreq) ./ imageFreq
```

Especificar el peso de las clases usando pixelClassificationLayer.

```
pxLayer =
pixelClassificationLayer('Name','labels','Classes',tbl.Name,'ClassWeights',classWeights);
lgraph = replaceLayer(lgraph,"classification",pxLayer);
```

Seleccionar opciones de entrenamiento

```
% Define validation data.
pximdsVal = pixelLabelImageDatastore(imdsVal,pxdsVal);

% Define training options.
options = trainingOptions('sgdm', ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropPeriod',10,...
    'LearnRateDropFactor',0.3,...
    'Momentum',0.9, ...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',0.005, ...
    'ValidationData',pximdsVal,...
    'MaxEpochs',30, ...
    'MiniBatchSize',8, ...
    'Shuffle','every-epoch', ...
    'CheckpointPath', tempdir, ...
    'VerboseFrequency',2,...
    'Plots','training-progress',...
    'ValidationPatience', 4);
```

Comienzo de entrenamiento

Se combinan los datos de entrenamiento y las selecciones de aumento de datos con pixelLabelImageDatastore. La función lee grupos de datos de entrenamiento, aplica el aumento de datos y envía los datos aumentados al algoritmo de entrenamiento.

```
pximds = pixelLabelImageDatastore(imdsTrain,pxdsTrain);
```

Se entrena con trainNetwork si doTraining es True.

```
doTraining = false;
if doTraining
    [net, info] = trainNetwork(pximds,lgraph,options);
else
    data = load(pretrainedNetwork);
    net = data.net;
```

```
end
```

Probar la red en una Imagen

Comprobación rápida de la red

```
I = readimage(imdsTest,1);
C = semanticseg(I, net);
```

Visualizar resultados.

```
B = labeloverlay(I,C,'Colormap',cmap,'Transparency',0.4);
imshow(B)
pixelLabelColorbar(cmap, classes);
```

Se comprueban los resultados con la clasificación realizada manualmente.

```
expectedResult = readimage(pxdsTest,2);
actual = uint8(C);
expected = uint8(expectedResult);
imshowpair(actual, expected)
```

Visualizar la cantidad de superposición por clase con la métrica de intersección sobre unión (IoU). Se usa la función `jaccard`.

```
iou = jaccard(C,expectedResult);
table(classes,iou)
```

Evaluar la red entrenada

Para comprobar la precisión en imágenes de prueba se ejecuta `semanticseg` en todo el conjunto de prueba.

```
pxdsResults = semanticseg(imdsTest,net, ...
    'MiniBatchSize',4, ...
    'WriteLocation',tempdir, ...
    'Verbose',false);
```

Se devuelven los resultados como un objeto pixelLabelDatastore.

```
metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTest,'Verbose',false);
```

evaluateSemanticSegmentation devuelve métricas de todo el conjunto de datos. Para verlas se usa la función metrics.DataSetMetrics .

```
metrics.DataSetMetrics
```

Se proporciona una descripción general del alto nivel de rendimiento de la red con `metrics.ClassMetrics`.

```
metrics.ClassMetrics
```

Funciones auxiliares

```
function labelIDs = camvidPixelLabelIDs()
% Return the label IDs corresponding to each class.
%
% The CamVid dataset has 32 classes. Group them into 11 classes following
% the original SegNet training methodology [1].
%
% The 11 classes are:
%   "Sky" "Building", "Pole", "Road", "Pavement", "Tree", "SignSymbol",
%   "Fence", "Car", "Pedestrian",  and "Bicyclist".
%
% CamVid pixel label IDs are provided as RGB color values. Group them into
% 11 classes and return them as a cell array of M-by-3 matrices. The
% original CamVid class names are listed alongside each RGB value. Note
% that the Other/Void class are excluded below.

labelIDs = {
[1]
[2]
};

end

function pixelLabelColorbar(cmap, classNames)
% Add a colorbar to the current axis. The colorbar is formatted
% to display the class names with the color.

colormap(gca,cmap)

% Add colorbar to current figure.
c = colorbar('peer', gca);

% Use class names for tick marks.
c.TickLabels = classNames;
numClasses = size(cmap,1);

% Center tick labels.
c.Ticks = 1/(numClasses*2):1/numClasses:1;

% Remove tick mark.
c.TickLength = 0;
end

function cmap = camvidColorMap()
% Define the colormap used by CamVid dataset.

cmap = [
    0 0 255   % Camino
    0 255 0   % No Camino
```

```
    ];

% Normalize between [0 1].
cmap = cmap ./ 255;
end

function [imdsTrain, imdsVal, imdsTest, pxdsTrain, pxdsVal, pxdsTest] =
partitionCamVidData(imds,pxds)
% Partition CamVid data by randomly selecting 60% of the data for training. The
% rest is used for testing.

% Set initial random state for example reproducibility.
rng(0);
numFiles = numel(imds.Files);
shuffledIndices = randperm(numFiles);

% Use 60% of the images for training.
numTrain = round(0.60 * numFiles);
trainingIdx = shuffledIndices(1:numTrain);

% Use 20% of the images for validation
numVal = round(0.20 * numFiles);
valIdx = shuffledIndices(numTrain+1:numTrain+numVal);

% Use the rest for testing.
testIdx = shuffledIndices(numTrain+numVal+1:end);

% Create image datastores for training and test.
trainingImages = imds.Files(trainingIdx);
valImages = imds.Files(valIdx);
testImages = imds.Files(testIdx);

imdsTrain = imageDatastore(trainingImages);
imdsVal = imageDatastore(valImages);
imdsTest = imageDatastore(testImages);

% Extract class and label IDs info.
classes = pxds.ClassNames;
labelIDs = camvidPixelLabelIDs();

% Create pixel label datastores for training and test.
trainingLabels = pxds.Files(trainingIdx);
valLabels = pxds.Files(valIdx);
testLabels = pxds.Files(testIdx);

pxdsTrain = pixelLabelDatastore(trainingLabels, classes, labelIDs);
pxdsVal = pixelLabelDatastore(valLabels, classes, labelIDs);
pxdsTest = pixelLabelDatastore(testLabels, classes, labelIDs);
end
```

Código del algoritmo de navegación

Se presenta el código que realiza la navegación de una trayectoria sinusoidal. Para ello, se unen el control lateral y el control longitudinal en un archivo con nombre "EHUNavegacion.m".

EHUNavegacion.m

```
clc
clear all
close all

%% Ibilbide bat definituko dugu
%Sinusoide bat:
xtrajectory=[0:0.1:12];
ytrajectory=5*sin(xtrajectory);
plot(xtrajectory,ytrajectory,'r')
grid on
hold on
xlabel('x [m]')
ylabel('y [m]')

%% Kontrol lateraleko parametroak
K1=1000;
K2=1.21;
thitahas=0; %Hasierako norabide angelua
xhas=4;
yhas=-10;
a=100;%azelerazioa

%% Simulazioaren parametroak
Niter=20000; %Zenbat alditan simulatuko dugun
DeltaT=0.001; %Integrazio urratsa
thita=zeros(1,Niter);
phi=zeros(1,Niter);
w=zeros(1,Niter);
V=zeros(1,Niter);
error=zeros(1,Niter);
x=zeros(1,Niter);
y=zeros(1,Niter);
thitaIbilbidea=zeros(1,Niter);%phitrayect
x(1)=xhas;
y(1)=yhas;
thita(1)=thitahas;

for k=1:Niter-1
    Kokapena=[x(k);y(k)];
    Ibilbidea=[xtrajectory;ytrajectory];
    [error(k),thitaIbilbidea(k),errorangle,RefTxikiena]=Ferror(Kokapena,Ibilbidea);
    phi(k)=(thita(k)-thitaIbilbidea(k));
    phic(k)=atan(K2*error(k));
    zuzenketa=angdiff(phi(k),phic(k));
    w(k)=K1*zuzenketa; %abiadura angeluarra
    %Ibilgailuaren ekuazioa dinamikoak integratu
    DeltaThita=DeltaT*w(k);
    while abs(DeltaThita)>pi/180
        DeltaT=DeltaT/2;
        DeltaThita=DeltaT*w(k);
    end
    thita(k+1)=thita(k)+DeltaThita;
    x(k+1)=x(k)+V(k)*cos(thita(k))*DeltaT;
```

```
    y(k+1)=y(k)+V(k)*sin(thita(k))*DeltaT;
    [V(k+1)]=Ajustevelocidad(abs(error(k)),w(k),V(k),a,DeltaT);
    DeltaT=0.01;
    quiver(x(k),y(k),0.2*cos(thita(k)),0.2*sin(thita(k)),0)%geziak
    plot(xtrajectory(RefTxikiena),ytrajectory(RefTxikiena),'xk');
    izenburua=strcat(' thita=',num2str(thita(k)),'; ');
    izenburua=strcat(izenburua,' phitrayect=',num2str(thitaIbilbidea(k)),'; ');
    izenburua=strcat(izenburua,'phi=',num2str(phi(k)),'; ');
    izenburua=strcat(izenburua,' e=',num2str(error(k)),'; ');
    izenburua=strcat(izenburua,' V=',num2str(V(k)),';');
    title(izenburua)
    pause(0.01)
end

%% f(e)
function [error,thitaIbilbidea,errorangle,RefTxikiena]=Ferror(Kokapena,Ibilbidea)
NpuntuIbilbidearenak=size(Ibilbidea,2);
Luzeerak=zeros(1,NpuntuIbilbidearenak);
for i=1:NpuntuIbilbidearenak
    Luzeerak(i)=norm(Ibilbidea(:,i)-Kokapena);%puntu guztietara distantzia
end
[error,RefTxikiena]=min(Luzeerak);%distantzia minimoa
errorvector=(Ibilbidea(:,RefTxikiena)-Kokapena);
errorangle=angle(errorvector(1)+sqrt(-1)*errorvector(2));
IbilbidekoGertuena=Ibilbidea(:,RefTxikiena);

if RefTxikiena==length(Luzeerak)
   IbilbidekoGertuenarenUrrena=Ibilbidea(:,1);
   IbilbideGehikuntza=IbilbidekoGertuenarenUrrena-IbilbidekoGertuena;
   thitaIbilbidea=angle(IbilbideGehikuntza(1)+sqrt(-1)*IbilbideGehikuntza(2));
else
   IbilbidekoGertuenarenUrrena=Ibilbidea(:,RefTxikiena+1);
   IbilbideGehikuntza=IbilbidekoGertuenarenUrrena-IbilbidekoGertuena;
   thitaIbilbidea=angle(IbilbideGehikuntza(1)+sqrt(-1)*IbilbideGehikuntza(2));
end

%errorearen zeinuaren zuzenketa
n=IbilbideGehikuntza/norm(IbilbideGehikuntza);
errorVectorTxikiena=Ibilbidea(:,RefTxikiena)-Kokapena;
T=n(1)*errorVectorTxikiena(2)-n(2)*errorVectorTxikiena(1);
error=T;
end

%% Abiadura funtzioa
function [V]=Ajustevelocidad(error,w,V,a,DeltaT)
at=(a^2)-(V^2*w^2);
zeinuat=sign(at);%zeinua
at=zeinuat*sqrt(abs(at));
Vmaxprimaprima=V+DeltaT*at;
Vnom=120;
V=min(Vmaxprimaprima,Vnom)
end
```

Código de algoritmo de localización y navegación

Por último, se realiza la unión de ambos algoritmos, el de navegación y el de planificación de trayectoria y localización. Se crea un programa principal, llamado "AlgoritmoLocalizacionYNavegacion.m", Este se encarga de recoger las imágenes, transformarlas a vista de pájaro y ejecutar la segmentación semántica para calcular la trayectoria. Tras esto, se llama a la función "Control_trayectoria.m", que a su vez accede a las funciones "Ferror.m" y "Ajustevelocidad.m".

AlgoritmoLocalizacionYNavegacion.m

```
%%  Segmentation in the loop
clearvars -except net netinfo
close all
clc

%% Cargar red Neuronal
if ~exist('net','var')
    load ('Red_camino')
end

%% Parámetros vista de pájaro
%para video 1
focalLength = [1600 1500];
principalPoint = [1000 800];
height = 2.1798;%altura de cámara
pitch = 14;%ángulo que abarca
distAhead = 30;
spaceToOneSide = 6;
bottomOffset = 3;
outImageSize = [NaN,250];

%% Tamaño imágenes
run=true;
count=0;
mostrar=true;
net_input_size=net.Layers(1).InputSize(1:2);
cmap = ade20kColorMap2;
tic
vidObj = VideoReader('video1.mp4');
vidObj.CurrentTime = 10;%se coge la primera imagen en el instante 10s
Im_original_size=size(vidObj);
Im_original_size=[360,640,3];
Im_original_size=Im_original_size(1:2);

%% Vista pájaro transformaciones
camIntrinsics = cameraIntrinsics(focalLength,principalPoint,Im_original_size);
sensor = monoCamera(camIntrinsics,height,'Pitch',pitch);
outView = [bottomOffset,distAhead,-spaceToOneSide,spaceToOneSide];
birdsEye = birdsEyeView(sensor,outView,outImageSize);

%% Loop
while(run)
    vidObj.CurrentTime = vidObj.CurrentTime+5;%se coge una imagen cada 5 segundos
    vidFrame = readFrame(vidObj);

    if mostrar
        figure(1)
        subplot(2,2,1)
```

```
        imshow(vidFrame)
        hold on
        title ('Camara')
        % vista pájaro
        BEV = transformImage(birdsEye,vidFrame);
        subplot(2,2,2)
        imshow(BEV)
        hold on
        title ('VistaPajaro')
    end

    Segmentado = semanticseg(BEV, net);
    overlay = labeloverlay(BEV,Segmentado,...
        'Colormap',cmap,'Transparency',0.4);%overlay de clases en imágenes
    %recorte para eliminar desviación de vista de pájaro
    overlay_crop=overlay(1:end-35,:,:);
    Segmentado_crop=Segmentado(1:end-35,:,:);

    %Calcular los pixeles que son 'Camino'
    [filas,columnas]=size(Segmentado_crop);
    D=zeros(filas,columnas);
    PuntosSuelo=[];
    for i=1:filas
        C_fila=Segmentado_crop(i,:);
        CrefPos=[1:columnas];
        %Calcular donde se encuentran los pixeles del 'Camino'
        ColumnasSueloFila=CrefPos(C_fila=='Camino');
        D(i,C_fila=='Camino')=1;
        if length(ColumnasSueloFila)>0
            ColumnaPixelesmedio=round(mean(ColumnasSueloFila));
            PuntosSuelo=[PuntosSuelo,[i;ColumnaPixelesmedio]];
        end
    end

    if mostrar
        figure(1)
        subplot(2,2,3)
        imshow(overlay_crop)
        hold on
        title ('Segmentado')
        subplot(2,2,4)
        imshow(D)
        %Dibujar los pixeles de la trayectoria
        hold on
        title ('Trayectoria')
        plot(PuntosSuelo(2,:),PuntosSuelo(1,:),'xr')
    end

    figure(2);
    BEV_crop=BEV(1:end-35,:,:);
    imshow(BEV_crop)
    hold on
    plot(PuntosSuelo(2,:),PuntosSuelo(1,:),'-c','LineWidth',8)
    hold on

    %interpolacion
    u=[1:length(PuntosSuelo(2,:))];
    uinter=u(1:40:end);
    xinter=spline(uinter,PuntosSuelo(2,uinter),u);
    yinter=spline(uinter,PuntosSuelo(1,uinter),u);
```

```
    plot(xinter,yinter,'b','LineWidth',5)

    %Control
    inter=[yinter;xinter];
    Control_trayectoria(inter);

    % Aumentar contador para medir tiempo de ejecucion
    count=count+1
    % Comprobar condicion de parada
    if count==500
        run=false;
        break
    end

    pause(0.1)
    close all
end
total_t=toc;
exe_time=total_t/count
```

Control_trayectoria.m

```
function [k]=Control_trayectoria(trajectory)

xtrajectory=flip(trajectory(2,:));
ytrajectory=flip(trajectory(1,:));
xhas=trajectory(2,end);
yhas=trajectory(1,end);
thitahas=0; %Hasierako norabide angelua

%Kontrol lateraleko parametroak
K1=1000;
K2=1.21;
a=100;
%Simulazioaren parametroak
Niter=5000; %Zenbat alditan simulatuko dugun
DeltaT=0.001; %Integrazio urratsa
thita=zeros(1,Niter);
phi=zeros(1,Niter);
w=zeros(1,Niter);
V=zeros(1,Niter);
error=zeros(1,Niter);
x=zeros(1,Niter);
y=zeros(1,Niter);
thitaIbilbidea=zeros(1,Niter);
x(1)=xhas;
y(1)=yhas;
thita(1)=thitahas;
    for k=1:Niter-1
        Kokapena=[x(k);y(k)];
        Ibilbidea=[xtrajectory;ytrajectory];

[error(k),thitaIbilbidea(k),errorangle,RefTxikiena]=Ferror(Kokapena,Ibilbidea);
        phi(k)=(thita(k)-thitaIbilbidea(k));
        phic(k)=atan(K2*error(k));
        zuzenketa=angdiff(phi(k),phic(k));
        w(k)=K1*zuzenketa; %abiadura angeluarra
        %Ibilgailuaren ekuazioa dinamikoak integratu
        DeltaThita=DeltaT*w(k);
        while abs(DeltaThita)>pi/180
            DeltaT=DeltaT/2;
```

```
            DeltaThita=DeltaT*w(k);
        end
        thita(k+1)=thita(k)+DeltaThita;
        x(k+1)=x(k)+V(k)*cos(thita(k))*DeltaT;
        y(k+1)=y(k)+V(k)*sin(thita(k))*DeltaT;
        [V(k+1)]=Ajustevelocidad(abs(error(k)),w(k),V(k),a,DeltaT);
        DeltaT=0.01;
        plot(xtrajectory(RefTxikiena),ytrajectory(RefTxikiena),'.r','MarkerSize',6);
        izenburua=strcat('phi=',num2str(phi(k)));
        izenburua=strcat(izenburua,' phic=',num2str(phic(k)));
        izenburua=strcat(izenburua,' thitaibilbidea=',num2str(thitaIbilbidea(k)));
        izenburua=strcat(izenburua,' error=',num2str(error(k)));
        izenburua=strcat(izenburua,' v=',num2str(V(k)));
        title(izenburua)
        pause(0.01)
    end
end
```

Ferror.m

```
function [error,thitaIbilbidea,errorangle,RefTxikiena]=Ferror(Kokapena,Ibilbidea)
NpuntuIbilbidearenak=size(Ibilbidea,2);
Luzeerak=zeros(1,NpuntuIbilbidearenak);
for i=1:NpuntuIbilbidearenak
    Luzeerak(i)=norm(Ibilbidea(:,i)-Kokapena);%puntu guztietara distantzia
end
[error,RefTxikiena]=min(Luzeerak);%distantzia minimoa
errorvector=(Ibilbidea(:,RefTxikiena)-Kokapena);
errorangle=angle(errorvector(1)+sqrt(-1)*errorvector(2));
IbilbidekoGertuena=Ibilbidea(:,RefTxikiena);

if RefTxikiena==length(Luzeerak)
   IbilbidekoGertuenarenUrrena=Ibilbidea(:,1);
   IbilbideGehikuntza=IbilbidekoGertuenarenUrrena-IbilbidekoGertuena;
   thitaIbilbidea=angle(IbilbideGehikuntza(1)+sqrt(-1)*IbilbideGehikuntza(2));
else
   IbilbidekoGertuenarenUrrena=Ibilbidea(:,RefTxikiena+1);
   IbilbideGehikuntza=IbilbidekoGertuenarenUrrena-IbilbidekoGertuena;
   thitaIbilbidea=angle(IbilbideGehikuntza(1)+sqrt(-1)*IbilbideGehikuntza(2));
end

%errorearen zeinuaren zuzenketa
n=IbilbideGehikuntza/norm(IbilbideGehikuntza);
errorVectorTxikiena=Ibilbidea(:,RefTxikiena)-Kokapena;
T=n(1)*errorVectorTxikiena(2)-n(2)*errorVectorTxikiena(1);
error=T;
end
```

Ajustevelocidad.m

```
function [V]=Ajustevelocidad(error,w,V,a,DeltaT)
at=(a^2)-(V^2*w^2);
zeinuat=sign(at);
at=zeinuat*sqrt(abs(at));
Vmaxprimaprima=V+DeltaT*at;

Vnom=120;
V=min(Vmaxprimaprima,Vnom);
end
```

Anexo III: Manual de usuario

Se recomienda tener todos los archivos en el mismo directorio, pero en caso de no poder ser así se deberán cumplir ciertas consideraciones respecto al código mostrado en el Anexo anterior. Para poder ejecutar todos los códigos, es obvia la necesidad de disponer del programa Matlab. En este caso se ha utilizado la versión R2020b.

Funcionamiento del código para visualizar problemática del signo del error

Se debe ejecutar el archivo de nombre "EHUControlLateralProblemaSignoError.m". No es necesario que en el mismo directorio exista ningún otro archivo.

Las primeras líneas del código forman la definición de la trayectoria. Después, se establecen los parámetros del control lateral y de la simulación, inicializando todos los valores necesarios. Se accede al bucle que realiza el propio control lateral. En dicho bucle se hace la llamada a la función error, escrita en las últimas líneas.

Se ejecuta, y como resultado, se visualiza una gráfica con la trayectoria y el movimiento del AGV a través de flechas. Además, como título, se observan diferentes parámetros de interés.

Funcionamiento de código para estimar el valor de K_1

Al igual que en el caso anterior, solo se requiere del archivo "RangoDeK1.m" para realizar la ejecución.

Simplemente está definida la función que demuestra el desvanecimiento de la integral, vista en el apartado *4.1.1.4.1.2 Error de posicionamiento reducido*.

Ejecutando este archivo se obtiene una gráfica donde se ve la evolución de **y** y de la integral.

Funcionamiento de código para ver estabilidad cuando $\varphi_{trayect.}$ no es nulo

Se cuenta con el archivo "EHUControlLateralEstabilidadIbilbideJakinBaterako.m", sin necesidad de ficheros adicionales.

La estructura es muy similar a la del código para estimar el valor de $\mathbf{K_1}$, pero el bucle principal cuantificar el sistema visto en la ecuación (20). De esta manera, se calculan datos del punto más cercano del AGV y del siguiente, correspondiente al instante $t+1$. Además, la función del error ya tiene la corrección del signo incorporada.

Se consigue una gráfica con tres "subplot". El primero de ellos hace referencia a la trayectoria sobre la que se ha estimado la estabilidad. Después, en 3D, se visualiza la energía y la derivada de esta, ambas respecto a la trayectoria fijada.

Funcionamiento de código para calcular K_2

Para obtener el valor de $\mathbf{K_2}$ es necesario que en el mismo directorio estén el código correspondiente a "RangoDeK2.m" y el "EHUControlLateralEvaluacionJ.m". En el caso de no ser así, Matlab informará sobre el error. Respecto a la ejecución, se debe dar "RUN" al primero de estos.

El archivo "RangoDeK2.m" está organizado de manera similar a los anteriores códigos, pero en este caso, el bucle hace la llamada a la función de coste. Con ella, se calcula un promediado para obtener el valor óptimo de la constante $\mathbf{K_2}$. El segundo fichero es una función a la que se le indican los parámetros de simulación y cuantifica la **J**. Incorpora también la función de la estimación del error.

Ya que se requiere el cálculo de la función de coste en todos los puntos de una zona acotada, es un archivo de ejecución lenta. Al finalizar se obtiene una gráfica compuesta por cruces donde se visualiza el valor de $\mathbf{K_2}$ que minimiza **J**.

Funcionamiento de código para obtener imágenes de video

El directorio deberá incorporar el archivo de video del que se quieran obtener las capturas, en este caso "video1.mp4", y el código "GaurdarImagenes.m".

El código primero describe los parámetros de la cámara. Tras esto, se debe de indicar el directorio en el que se quieren guardar las imágenes. Las siguientes líneas realizan la lectura del video y fijan las operaciones para la transformación de la vista de pájaro. El bucle recoge "frames" del video cada 4 segundos, tiempo que se puede variar. Se guardan las imágenes con el nombre "BEV_Image_X.jpg", siendo X la numeración de la imagen comenzando desde 1.

De este modo se genera una carpeta con diferentes imágenes del entorno, que servirán como conjunto de datos para realizar el entrenamiento de la red.

Funcionamiento de código para entrenar red neuronal

El código "SemanticSegmentationUsingDeepLearningExampleAdapted,m" es un "LiveScrpit". Para poder ejecutarlo será necesario tener la "Deep Learning Toolbox" instalada.

Se le debe indicar el directorio donde se han almacenado las imágenes obtenidas del video. En las líneas 8 y 9, también se han de especificar las clases que se crearon en la aplicación "Image Labeler" para la sesión de este proyecto. En la línea 14 a su vez, se requerirá fijar un directorio para almacenar las imágenes con sus respectivas clases. Por lo demás, el "script" se debe ejecutar paso a paso.

La obtención de la red tarda un tiempo considerable, que dependerá del conjunto de datos del que se disponga. El "LiveScript" es muy intuitivo, pudiendo visualizar a cada paso los resultados que se van obteniendo.

Funcionamiento de código de algoritmo de navegación

El fichero "EHUNavegacion.m" también puede ejecutarse de manera individual, ya que él mismo incorpora todas las funciones que necesita.

Las primeras líneas del código son para ejecutar inicializaciones de parámetros relacionadas con la trayectoria, el control lateral y longitudinal y la simulación. El bucle es muy similar al de códigos anteriores, pero en este caso realiza la llamada a las funciones de "Ferror" y "AjusteVelocidad" definidas en las últimas líneas.

Como resultado, se obtiene una gráfica con la trayectoria y la navegación del AGV. De igual modo, el título contiene diferentes parámetros de interés.

Funcionamiento de código de algoritmo de localización y navegación

El directorio para poder ejecutar el algoritmo de localización y navegación al completo debe de contener varios archivos. Por un lado, el archivo principal, "AlgoritmoLocalizacionYNavegacion.m" desde el que se ejecutará todo lo demás. Por otro, las funciones de "Control_trayectoria.m", "Ferror.m" y "Ajustevelocidad.m". Será necesario también que en el mismo directorio este la red creada, en este caso denominada "Red_camino.mat". De igual modo, se necesita efectuar la lectura del "video1.mp4", con el que se ha entrenado la red. Por último, el archivo "ade20kColorMap2.m", que posibilita el dibujar la segmentación semántica sobre las imágenes del entorno.

Los ficheros "Control_trayectoria.m", "Ferror.m" y "Ajustevelocidad.m", son los mismos que el que realizan la navegación, pero de manera separada. Además, el primero leerá la trayectoria del programa principal en lugar de indicársela como un parámetro fijo.

El código principal carga la red. Tras esto, se establecen diferentes parámetros relacionados con la cámara y el tamaño de las imágenes. Se lleva a cabo la lectura del video antes de realizar las operaciones de transformación a vista de pájaro. El bucle va recogiendo imágenes cada 5 segundos, que se modifican directamente a la vista de pájaro. Después, se efectúa la segmentación semántica y se calculan los píxeles que se van a establecer como trayectoria. Con esta información se realiza la interpolación y se llama a la función de la navegación.

Se generan dos figuras. La primera de ellas muestra los pasos que se han dado desde la obtención de la imagen original hasta el cálculo de la trayectoria, estando formada por cuatro subimagenes. La segunda, expone la imagen a vista de pájaro con la trayectoria y el algoritmo de navegación, con el AGV en movimiento, sobre esta.