TRABAJO FINAL DE MASTER

Control de seguimiento y velocidad de un AGV mediante redes neuronales y aprendizaje por refuerzo

Septiembre de 2021

Autor:
Javier Argente Mena

Directores:
Matilde Santos Peñas
J. Enrique Sierra-García

TRABAJO FINAL DE MASTER

Control de seguimiento y velocidad de un AGV mediante redes neuronales y aprendizaje por refuerzo

Septiembre de 2021

Autor:
Javier Argente Mena

Directores:
Matilde Santos Peñas
J. Enrique Sierra-García

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado: Javier Argente Mena

Resumen

Los vehículos autónomos de guiado (Automated Guided Vehicle o AGV) se están convirtiendo actualmente en los sistemas más populares en lo que respecta a la logística interna de las fábricas ya que pueden transportar varias toneladas de peso y se pueden adaptar a cualquier tipo de carga. Son equipos muy flexibles que facilitan el transporte interno de elementos y tienen muchas ventajas como la trazabilidad, seguridad y automatización de flujos.

En este trabajo se presenta la descripción del desarrollo del control de seguimiento y velocidad de un AGV empleando para ello el uso de técnicas tradicionales y de inteligencia artificial, con el objetivo de diseñar el control más apropiado para que el robot sea capaz de realizar un desempeño adecuado en diferentes escenarios. Las técnicas empleadas se resumen en: control PID sintonizado con algoritmo genético y control incremental PID con Q-Learning para el seguimiento; control PI y control neuronal con modelo de referencia para el control de la velocidad de las ruedas.

Para ello se crea un modelo matemático del AGV con sus parámetros nominales. Tras el diseño de los distintos tipos de control, varias combinaciones de estos controles se aplican al modelo del AGV para ponerlo a prueba. Tras el análisis de los resultados, se realiza una valoración final de los más apropiados.

Palabras clave: *AGV, Modelado, Redes Neuronales, Aprendizaje por Refuerzo, Control PID, Algoritmos Genéticos, Q-Learning, Control Neuronal con Modelo de Referencia*

Abstract

Currently, Automated Guided Vehicles (AGV) are becoming the most popular systems at industry with regards to internal factory logistics as they can carry several tons of weight and can be adapted to any type of load. They are very flexible systems which make easy the internal transport of elements and have many advantages such as traceability, security and flow automation.

This work shows a description of the development of a tracking and velocity control of an AGV making use of traditional and artificial intelligence control techniques in order to design the most suitable control for the robot, for this to achive the best performance under different conditions. The techniques used can be summarized in: PID control with genetic algorithm tuning and incremental PID control making use of Q-Learning for tracking control; and PI control and neural network control with reference model for velocity control of the wheels.

For this purpose, a mathematical model of the AGV with nominal parameters is created. Once the controllers are designed, several combinations of controllers are tested on the AGV. After analyzing the results, a final selection of the most appropriate controller is made.

Keywords: *AGV, Modelling, Neural Networks, Reinforcement Learning, PID Control, Genetic Algorithms, Q-Learning, Model Reference Neural Network Controller*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Introducción

Los sistemas móviles se pueden definir como sistemas que no están adheridos al entorno y pueden moverse en un espacio determinado. Dependiendo del entorno en el que se mueven, estos se pueden clasificar en los siguientes grupos:

- **Sistemas móviles terrestres**: Aquí se pueden encontrar varios tipos de plataformas móviles como vehículos móviles con ruedas u orugas, robots con patas (humanoides o imitadores de animales), o robots que imitan algún otro tipo de locomoción animal, por ejemplo, serpientes. Los sistemas móviles terrestres con ruedas u orugas que no llevan al operador se denominan vehículos terrestres no tripulados.
- **Sistemas móviles aéreos**: Este grupo está formado por sistemas móviles que vuelan en un determinado espacio aéreo (aviones, helicópteros, drones, cohetes o satélites). Cuando se utilizan sin piloto, son denominados vehículos aéreos no tripulados.
- **Sistemas móviles acuáticos y subacuáticos**: En este grupo se encuentran diferentes tipos de embarcaciones, submarinos, vehículos submarinos autónomos, etc.

Los vehículos móviles se consideran autónomos cuando son capaces de moverse de forma autónoma dentro de su entorno. La autonomía debe estar garantizada por lo siguiente:

1. **desde el punto de vista energético**: el vehículo debe llevar consigo alguna fuente de energía.
2. **desde el punto de vista de decisión**: el vehículo debe ser capaz de tomar ciertas decisiones y las consecuentes acciones adecuadas.

Esto significa que el vehículo móvil recibe las ordenes de un operador humano en función del nivel de autonomía del sistema. El vehículo se comporta de manera que intenta llevar a cabo una tarea ordenada y las subtareas apropiadas en los niveles inferiores. En caso de que no se produzcan imprevistos, la tarea suele ejecutarse en un intervalo de tiempo determinado. Según el nivel de autonomía del robot, algunas de las consignas que puede recibir del operador son:

- **Velocidad de crucero deseada**: El sistema de control que se ejecuta en el vehículo es consciente del modelo cinemático del robot, de manera que puede calcular las velocidades de rueda adecuadas para lograr dicha velocidad de crucero deseada.
- **Trayectoria deseada a seguir**: El robot es capaz de determinar y controlar su localización dentro de su entorno. Dicha la localización se define generalmente como posición y la orientación en relación con algún sistema de coordenadas fijo o embarcado en el propio vehículo. Para obtener la localización, se utilizan diferentes sensores montados sobre el vehículo o en el entorno con el objetivo de obtener la mejor aproximación posible sobre su localización.

- **Tarea deseada dentro de un entorno conocido con obstáculos**: El vehículo debe de ejecutar una tarea en un entorno conocido pero con la presencia de algunos obstáculos (estáticos o en movimiento). En este caso debe de ser capaz de planificar su camino y modificarlo ante la presencia de obstáculos que puedan entorpecer o impedir la ejecución de la tarea encomendada.
- **Tarea deseada dentro un entorno desconocido**: El vehículo desconoce su entorno. Tiene que realizar simultáneamente las acciones de determinar su localización y crear un mapa de dicho entorno. Este enfoque se conoce como SLAM (localización y mapeo simultáneos).

Dependiendo del tipo de aplicación, existen diversas tecnologías de navegación para los AGV que les dotan de la habilidad de ejecutar una tarea siguiendo para ello una trayectoria y evitando obstáculos al mismo tiempo. Dependiendo del sistema de navegación empleado, estos robots autónomos se pueden clasificar como:

1. Navegación con caminos preestablecidos: Se utilizan en aplicaciones industriales, donde se prima conocer muy bien por donde va el AGV y los tiempos para optimizar los procesos al máximo y garantizar la seguridad
2. Navegación libre: Al AGV se le envía un punto destino y el calcula la trayectoria para llegar, los caminos no están preestablecidos en el plano. Se utilizan sobre todo en robótica de servicio

Hoy en día, los robots móviles se utilizan en numerosas aplicaciones y en diferentes campos, y cada día están adquiriendo más importancia en el entorno industrial por los beneficios que aportan a los sistemas de producción. Algunas de las numerosas aplicaciones de este tipo de robots en la industria son:

- El transporte de materias primas: como papel, acero, caucho, metal y plástico.
- Sistemas de montaje: Transporte del producto fabricado para apoyar los procesos de montaje en serie.
- Transporte Work-in-Process: transporte repetitivo de materiales durante todo el proceso de fabricación o entre celdas de fabricación .
- Manipulación de palets: transporte de palets desde el paletizador hasta el sistema de embalaje, almacén y/o los muelles de salida.
- Carga y descarga en o desde remolques: transporte, carga y descarga de productos terminados directamente en remolques sin ningún equipo especial de muelle.
- Vinculación de células de fabricación: transporte de productos de un proceso a otro.

Este trabajo se centra especialmente en los vehículos guiados automatizados (AGV) para aplicaciones industriales con una configuración de tipo triciclo con caminos preestablecidos.. Sin embargo, para poder validar las propuestas hechas, se han empleado los datos del AGV Easybot Standard 410-480 de la empresa ASTI Mobile Robotics (*https://www.astimobilerobotics.com*), de los cuales, algunos son del propio fabricante y otros sólo estimaciones. La imagen 1.1 muestra este AGV comercial.

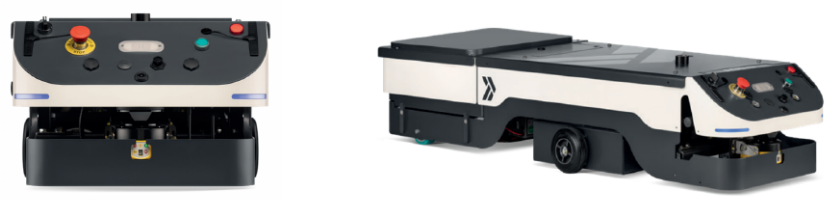

Figura 1.1: Easybot Standard 410-480

1.2. Objetivos

El objetivo de este trabajo es diseñar un control de seguimiento y velocidad de crucero del Easybot empleando para ello técnicas de la inteligencia artificial, las cuales son aptas para adaptarse a las distintas condiciones de trabajo y con las cuales se puedan alcanzar tanto las consignas de velocidad como el trazado exitoso de la trayectoria presentada.

Este objetivo se desglosa en los siguientes objetivos específicos:

- **Modelo matemático**: identificación de causalidad computacional y creación de un modelo computacional sobre AGV. Dicho modelo está basado en un trabajo anterior [1] en el cual se puede encontrar un modelo detallado de este AGV comercial.
- **Controlador adaptativo de velocidad con redes neuronales**: inicialmente, se diseña un controlador neuronal con modelo de referencia mediante el cual se establece un tiempo de establecimiento y una respuesta críticamente amortiguada para las velocidades de las ruedas. Con este control se consigue establecer en todo momento una dinámica deseada para las velocidades del AGV, la cual es alcanzable ante cualquier variación de parámetros mediante la actualización del controlador una vez se obtiene un modelo neuronal preciso sobre el AGV. Esta dinámica deseada se basa en un sistema multivariable lineal desacoplado. Para poder simular el AGV en movimiento, se realiza un control PID que se sintoniza mediante un algoritmo genético.
- **Controlador adaptativo de seguimiento mediante aprendizaje por refuerzo**: finalmente, con el objetivo de poder trazar diferentes trayectorias sin que el AGV se salga de éstas, se realiza un control mediante Q-Learning. A pesar de que los controles PID, como se comprobara posteriormente, realizan un buen desempeño para esta tarea, dicho desempeño empeora conforme alejamos al PID de su punto de trabajo, es decir, conforme se le hace trazar otras trayectorias diferentes a la empleada para la sintonización de sus parámetros. Este control se acompaña de dos controladores PI para controlar las velocidades de las ruedas y así poder simular el movimiento del AGV.
- **Combinación de las técnicas de aprendizaje por refuerzo con redes neuronales**: una vez diseñados y comprobados los controladores anteriores, se realiza una combinación de ambas técnicas de control tratadas en este trabajo con el objetivo de dotar al AGV con un control que sea capaz de adaptarse a nuevas condiciones internas y de su entorno.

Cuando todos los tipos de control están ajustados y entrenados, se prueban distintas combinaciones de ellos sobre el modelo del AGV y se analizan sus respuestas.

1.3. Estructura

La estructura de este documento es de la siguiente manera:

En el *Capitulo 1* se puede encontrar breve introducción a los robot móviles y de los objetivos a conseguir y desarrollo.

En el *Capitulo 2* se hace una breve descripción sobre el tipo de AGV empleado y se explican los fundamentos teóricos necesarios para comprender tanto las técnicas de control empleadas como el método de programación empleado en su desarrollo. Por ultimo

En el *Capitulo 3* se obtiene el modelo dinámico del robot Easybot. Se eligen varios puntos de un modelo completo [1] y se resuelve el sistema de ecuaciones.

En el *Capitulo 4* se realiza una explicación sobre el diseño del controlador neuronal con modelo de referencia, el cual es ajustado y entrenado para cumplir con un tipo de respuesta predefinido para las velocidades de las ruedas. Este controlador es acompañado con un PID cuya sintonización de los parámetros se hace mediante un algoritmo genético.

Tras finalizar el diseño del control de velocidad, se pasa al *Capitulo 5* donde se diseña un control PID incremental con adaptación de sus parámetros Kp, Kd y Ki mediante Q-Learning. Dicho control incremental es aplicado al seguimiento de la trayectoria. Para el control de los motores se usan dos controladores PI.

En el *Capitulo 6* se realizan simulaciones con el AGV controlado por los controladores y se analiza su funcionamiento, comentando los resultados.

El *Capitulo 7* recoge las conclusiones sobre este trabajo, la selección final del control del AGV y futuras líneas de trabajo.

Capítulo 2

Fundamentos teóricos y estado del arte

2.1. Vehículo AGV filoguiado

Los AGVs se adaptan a cualquier tipo de tarea y entorno, conservando la opción de poder ser utilizados de forma manual[1]. Un tipo de AGV muy común en los entornos industriales son los de tipo filoguiado, y es el tipo de AGV sobre el que trata este trabajo. Estos se desplazan guiándose por una cinta magnética instalada bajo el suelo. Este método de guiado es muy sencillo pero es el de menor flexibilidad, ya que las rutas de movimiento del AGV se limitan a las rutas que describen dichas cintas instaladas. La cinta emite un campo magnético y éste es captado por un sensor magnético embarcado en el AGV 2.1, reportando la posición de un campo magnético a lo largo de su eje horizontal. Estos sensores usan un procesado de señal avanzado para medir su distancia lateral desde el centro del sensor con una precisión milimétrica. Dicha posición es enviada al sistema de control bien mediante una señal analógica, bus de comunicaciones o señales PWM, entre otros.

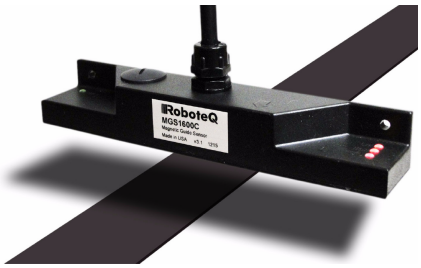

Figura 2.1: Sensor Magnético MGS1600GY

Dado que estas cintas magnéticas son totalmente pasivas, son muy fáciles de instalar y modificar. También son inmunes a la suciedad y pueden quedar totalmente cubiertas debajo de alfombras, baldosas u otra cubierta para suelos.

Aparte de la cinta magnética que describe la trayectoria, también se suelen instalar marcadores. Estos son trozos de cinta magnética que se colocan en el lado izquierdo o derecho de la cinta principal, con el objetivo de ser diferenciados de la cinta principal 2.2. La polaridad magnética de estos marcadores es opuesta a la principal y son empleados para reportar información al AGV sobre áreas especiales a lo largo de la trayectoria, bifurcaciones o fusiones, zonas de alta o baja velocidad, estaciones de carga, etc. Estos marcadores deben colocarse a una distancia relativamente cercana a la cinta principal 2.3.

Figura 2.2: Ejemplo de marcadores

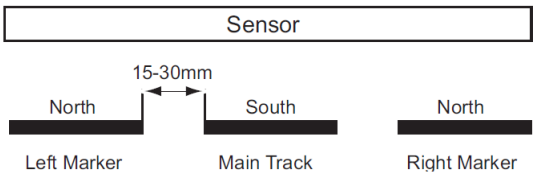

Figura 2.3: Instalación de marcadores

2.2. Controlador Proporcional Integral Derivativo (PID)

En este capítulo se procede a la implementación tanto del control de velocidad como el de seguimiento del AGV con el objetivo de que este pueda trazar una trayectoria de la manera más precisa posible y manteniendo una velocidad de crucero v_{ref} constante, obteniendo así el AGV-nominal que se comentó al inicio de este documento.

El tipo de controlador a usar es el control Proporcional Integral Derivativo. Este controlador es un mecanismo de control que a través de un lazo de retroalimentación permite regular la velocidad, temperatura, presión y flujo entre otras variables de un proceso en general. El controlador PID calcula la diferencia entre una variable real y una variable deseada.

El algoritmo del control PID consta de tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor proporcional depende del error actual, el integral depende de los errores pasados y el derivativo es una predicción de los errores futuros. La suma de estas tres acciones es usada para ajustar el proceso por medio de un elemento de control.

La ecuación en su forma más simple es la siguiente:

$$u\left(t\right) = K_p e\left(t\right) + K_i \int_0^t e\left(t\right) \mathrm{dt} + K_d \frac{d\, e\left(t\right)}{\mathrm{dt}}$$

donde:

- K_p constante proporcional. Esta aumenta la velocidad de respuesta y disminuye el error, pero también aumenta la inestabilidad del sistema
- K_d constante derivativa. Aumenta la estabilidad y disminuye un poco la velocidad del sistema. El error en régimen permanente no se ve afectado.
- K_i constante integral. Disminuye el error del sistema en régimen permanente, aumenta la estabilidad y también en cierto la grado la velocidad del sistema.

A pesar de que en este controlador se indique como 'clásico', esto hace referencia únicamente al tiempo que lleva empleándose y no significa para nada que el control PID sea un mal control y/o obsoleto, sino todo lo contrario, el PID lleva durante muchos años en la industria demostrando ser una muy buena opción de control, aplicándose al 80 % de los problemas de control de la industria. En este trabajo solo se usa su forma más sencilla por temas de simplificación, pero existen variantes mejoradas sobre él como por ejemplo: PID robusto, PID con anti-windup y filtro en la derivada, PID con tracking mode, etc.

2.3. Algoritmos genéticos (AG)

Los algoritmos genéticos forman parte de las técnicas de búsqueda inteligente, siendo empleados para resolver problemas de búsqueda y optimización complejos de resolver mediante otras técnicas. Estos algoritmos son técnicas de búsqueda heurística aportando soluciones buenas a un costo computacional y tiempo razonables. Están basados en conceptos biológicos derivados de la teoría Darwiniana de la evolución natural, empleando para ello una selección probabilística. A pesar de que en muchas ocasiones no se llegue a obtener una solución óptima absoluta, sus resultados son más que válidos y aceptables. Se realiza a continuación un breve resumen sobre sus elementos y metodología de diseño.

2.3.1. Componentes de un AG

Brevemente, sus elementos son:

- **Genes**: son los parámetros que se pretenden optimizar.
- **Cromosomas**: vectores de parámetros.
- **Individuo**: miembros de la población de soluciones potenciales al problema.
- **Población**: Es la colección de individuos.
- **Función de evaluación (fitness)**: función que evalúa como de buena es una solución para el problema de optimización.
- **Fenotipo**: solución construida a partir del genotipo
- **Genotipo**: conjunto de parámetros de un cromosoma en particular y que especifica un diseño determinado

2.3.2. Diseño de un algoritmo genético:

Los algoritmos genéticos trabajan con una población de individuos (inicializada de manera aleatoria), cada uno de ellos aportando una solución posible al problema. Estos individuos son evaluados de acuerdo a su adecuación para el problema dado, siendo los que aportan las mejores soluciones los que tienen una mayor probabilidad de reproducción y por lo tanto de pasar a la siguiente generación. Se distinguen las siguientes etapas:

- **Codificación** Los parámetros de un problema (genes) se agrupan juntos en una cadena de valores (cromosoma). Estos parámetros se codifican mediante valores binarios, decimales,... etc. Existen muchas posibilidades para esta representación.
- **Selección** Una vez evaluados los individuos mediante la función de fitness, se les asigna una probabilidad (p) de tener N hijos ($N = m\ p$), siendo los de mayor adecuación los que serán seleccionados para la reproducción un mayor número de veces que los individuos con poca. Existen diversos métodos de selección, como por ejemplo:
  - *Selección proporcional*: Se obtiene un numero aleatorio (r) entre 0 y 1 que es comparado con la probabilidad acumulada de cada individuo (q) de la generación actual (t). Si $r > q$, entonces se selecciona dicho individuo para la reproducción. Este método presenta el inconveniente de que la presencia de individuos por encima de la probabilidad media puede dar lugar a convergencia prematura.

$$p_i^t = \frac{f_i^t}{\sum_{i=1}^{m} f_i^t} \tag{2.1}$$

$$q_i^t = \sum_{j=1}^{i} p_i^t \tag{2.2}$$

- *Selección por Ranking*: Los individuos son ordenados en una lista en orden de mejor a peor adecuación, asignándoles una probabilidad de reproducción que depende de la posición ocupada en dicha lista (c).

$$p_i^t = q\,(1-q)^{\mathrm{Rank}(C_i^t)-1} \qquad 0 < q < 1 \tag{2.3}$$

- *Método de la ruleta*: Considerado un método de selección proporcional y siendo el más popular y empleado por su simplicidad de implementación. Se basa en crear un vector de valores acumulativos de los individuos e ir comparando el valor fitness de cada individuo con valores aleatorios entre en menor y el mayor valor acumulativo. El cromosoma con el valor inmediatamente superior al aleatorio generado, será seleccionado para la reproducción. En ocasiones el peor individuo puede ser seleccionado varias veces de manera que el algoritmo se vuelve ineficiente a medida que el tamaño de la población es mayor.

- **Operador de cruce**: Una vez finalizada la selección, se toman dos individuos de manera aleatoria o consecutiva y se parten sus cadenas de cromosomas por una o varias posiciones elegidas aleatoriamente. Estas colas se intercambian entre ellas, permaneciendo las cabeceras intactas. Los cromosomas resultantes son los hijos que formaran la nueva generación.

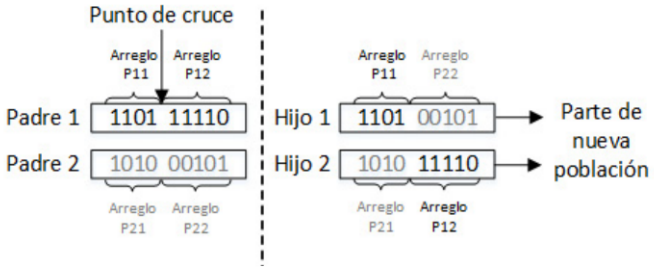

Figura 2.4: Ejemplo de cruce [15]

Existen distintas técnicas de cruce entre las cuales encontramos:

- *Cruce de un punto*: No es muy empleado ya que puede presentar problemas de sesgo posicional.
- *Cruce de dos puntos*: Muy empleado ya que minimiza los efectos disruptivos o destructivos del cruce.
- *Cruce uniforme*: incluye n puntos de cruce sin que se hayan fijado de antemano. Presenta el mayor efecto disruptivo de los tres presentados

- **Mutación**: Este operador es aplicado a cada componente de los genes y es indispensable para la obtención de buenos resultados ya que evita el atasco en mínimos locales. Se basa en comparar un numero aleatorio (r) con una probabilidad de mutación (Pm) para cada componente de los genes en las cadenas cromosómicas y cambiarlo si $Pm > r$. La probabilidad de mutación suele fijarse entre 0.001 y 0.01. Otras posibilidades para elegir la probabilidad son, por ejemplo, la de iniciar Pm con valores e ir decrementándolo de manera exponencial; o escoger Pm en función de la inversa de la longitud de la cadena cromosómica. Entre los distintos métodos de mutación, encontramos:

- *Mutación básica*
- *Mutación por inserción*
- *Mutación por desplazamiento*
- *Mutación por intercambio reciproco*
- *Mutación heurística*

- **Elitismo**: Es aconsejable asegurarse de que el mejor individuo de cada generación pase intacto a la siguiente
- **Sustitución generacional**: Posterior al cruce y mutación, se evalúan los descendientes obtenidos para decidir que individuos formaran la nueva población. Para ello, se evalúan los hijos obtenidos de dos padres y, de entre todos ellos, se escogen los mejores. Hay que destacar que la sustitución generacional completa puede producir un efecto negativo en la explotación del AG. Para resolver este problema, se puede emplear otra técnica de sustitución generacional llamada *Sustitución Steady-State* mediante la cual se elige un número de individuos que serán reemplazados, eliminando los demás. La selección de los individuos a eliminar se realiza mediante un ranking inverso (del peor al mejor). Los seleccionados para reproducción tendrán N=m*p hijos. Sin embargo, este algoritmo no funciona bien si hay elementos duplicados.
- **Convergencia**: Sin un AG es correctamente implementado, cada generación se ira aproximando a un óptimo global. Cuando el 95 % de la población poseen el mismo valor se dice que la población ha convergido.

2.4. Redes Neuronales

Las redes neuronales son de gran utilidad para la resolución de problemas complejos los cuales no pueden ser resueltos de una manera sencilla mediante un enfoque algorítmico tradicional. La principal característica de estas es su capacidad de trabajar eficazmente con las incertidumbres e imprecisiones del mundo real. A continuación, se enumeran varios puntos de importancia sobre los fundamentos de estas:

1. **Similitud entre el modelo biológico de una red neuronal y su modelo artificial**: Las neuronas de nuestro cerebro poseen la capacidad de comunicarse entre ellas. Para ello, se componen de:
   - *Dendritas*: reciben los estímulos. Estas corresponden a las entradas en una neurona artificial, las cuales tienen asociado un peso que se autoajusta en el entrenamiento y son los responsables de la adquisición del conocimiento. Estos pesos representan la sinapsis entre las neuronas biológicas. Si el valor del peso es positivo, indica que la relación entre las neuronas interconectadas es excitadora; si el valor es negativo, la relación es inhibidora y si su valor es cero, no existe conexión entre dichas neuronas.
   - *Cuerpo celular*: Es la parte de la neurona que procesa la información recibida a través de las dendritas y transmite las señales de salida. En la neurona artificial, este procesamiento se realiza con la suma ponderada de todas sus entradas y representa el nivel de excitación de la neurona. Dicho valor es evaluado por una función de activación (o función de transferencia) que transforma el estado de la neurona en una salida, siendo las funciones más empleadas:
     a) Función escalón.
     b) Función lineal mixta
     c) Función sigmoidea
     d) Función gaussiana
     e) Función tangente hiperbólica
   - *Axón*: Parte de la neurona encargada de transmitir (eléctricamente) las señales a los terminales axónicos, los cuales distribuyen la información a las siguientes neuronas interconectadas a esta mediante una señal química. Esta parte, en la neurona artificial, corresponde a la salida, la cual es enviada por igual a las entradas de la siguiente capa de neuronas en la red.
   - *Regla de aprendizaje*: En los seres humanos, el aprendizaje se puede entender como la modificación del comportamiento causado por la interacción con el entorno, es decir, experiencias, las cuales provocan nuevas respuestas a estímulos externos. Este conocimiento se

encuentra en la sinapsis. En las RNA esta sinapsis se encuentra en los pesos, cuya variación corresponde al aprendizaje. Todo proceso de aprendizaje implica un cierto número de cambios en estas conexiones.

2. **Características de las redes neuronales artificiales**: las RNA se pueden caracterizar de la siguiente manera:

   - *Topología*: la cual se basa en el número de capas que existen entre la entrada y la salida, el número de neuronas que componen cada capa, el grado de conectividad y el tipo de conexiones entre las neuronas. Podemos distinguir entre las siguientes topologías:
     a) Redes monocapa: En este tipo de redes solo existe una capa de neuronas entre las cuales se realizan conexiones en laterales, cruzadas o autorecurrentes.
     b) Redes multicapa: Formadas por conjuntos de neuronas agrupadas en capas (o niveles), las cuales se clasifican a su vez como feedforward o feedforward/feeback, dependiendo del tipo de conexión realizada entre ellas.
   - *Mecanismo de aprendizaje*: Este mecanismo es el que determina de qué manera se modifican los pesos ante las distintas informaciones de entrada. Se considera que la red ha aprendido una vez el valor de estos pesos se mantienen estables en el tiempo. Se distinguen dos tipos de aprendizaje:
     a) **Supervisado**: En este tipo de aprendizaje, es un agente externo el que fija que valor de salida corresponde a una determinada información de entrada. Si la salida de la red no corresponde a la proporcionada por dicho agente, se procede a la modificación de los pesos, hasta conseguir que la salida se aproxime a la deseada. Este aprendizaje se puede realizar de tres formas:
        - Por corrección del error: El ajuste de los pesos se hace en función del error, es decir, la diferencia entre la salida proporcionada por la RNA y la deseada.
        - Por refuerzo: durante este tipo de aprendizaje no se le proporciona a la red neuronal que salida se desea ante la información de entrada presentada. En su defecto, tan solo se le indica una señal de refuerzo que indica si la salida aportada por la red es válida o no, ajustándose los pesos en función de ello mediante un mecanismo de probabilidades.
        - Estocástico: únicamente se realizan cambios aleatorios de los valores de los pesos, evaluando los resultados obtenidos en comparación con los deseados y distribuciones de probabilidad.
     b) **No supervisado**: No se requiere de un agente externo ya que no se le proporciona a la RNA que salida se desea para la información de entrada, pudiendo interpretar la salida de distintas maneras, como por ejemplo, abstracción de características de los datos de entrada o grado de similitud entre la información actual e informaciones presentadas en el pasado. Se suelen considerar dos tipos de aprendizaje:
        - Aprendizaje Hebbiano: ajuste de los pesos mediante correlación. Si dos neuronas están activas se produce un reforzamiento de la conexión. Si una está activa y la otra no, se produce un debilitamiento de la conexión.
        - Aprendizaje competitivo y cooperativo: con este aprendizaje se pretende que solo una de las neuronas de salida esté activa ante cierta información de entrada, por lo tanto, las neuronas compiten entre ellas para conseguir su activación.
   - *Tipo de asociación entre entrada y salida*: Se distinguen entre:
     a) Redes heteroasociativas: las redes aprenden parejas de datos de tal manera que cuando se presenta una información de entrada, su salida es un resultado asociado a dicha entrada.
     b) Redes autoasociativas: Estas redes aprenden informaciones de tal manera que cuando se les presenta una nueva información en la entrada, se realiza una autocorrelación dando como salida el dato que más se le parece.

- *Representación de la información de entrada y salida*: Otra manera de clasificar las RNA es dependiendo del tipo de datos presentados a la entrada y los proporcionados a la salida de estas. Los datos pueden ser analógicos o discretos y en función de esto, se suele elegir el tipo de función de activación a emplear.

3. **Ventajas de las RNA**: Entre otras, se destacan:
   - Aprendizaje adaptativo: las RNA son capaces de aprender tras un entrenamiento inicial, pero también existen redes que continúan aprendiendo a lo largo de su vida útil después de dicho entrenamiento.
   - Tolerancia a fallos: una RNA puede continuar funcionando (con cierta degradación) si una o varias neuronas fallan, debido a que la información almacenada se encuentra distribuida en los pesos de todas las neuronas de la red.
   - Fácil inserción en la tecnología existente: se pueden añadir a sistemas ya en funcionamiento debido a su rapidez en el entrenamiento, verificación e implementación hardware de bajo coste.

En lo que respecta al uso de las redes neuronales en control automático, cuando se modela un sistema dinámico se desprecian ciertos parámetros importantes (ej. desgaste de las partes de un motor) con los cuales sería casi imposible obtener la solución del sistema. Aunque el funcionamiento del sistema modelado es efectivo, se pierde precisión al descartar estos parámetros, pudiendo no llegar a funcionar en algunos casos con el paso del tiempo. Con las RNA este problema no existe ya que el sistema, después de haber recibido una información inicial, empieza a identificar, aprender y responder ante distintas informaciones diferentes, sin importar que estas no sean idénticas a los patrones iniciales.

2.5. Aprendizaje por Refuerzo

El Aprendizaje por refuerzo es una de disciplinas que más rápidamente está creciendo y ayudando a crear inteligencia artificial. Toda la inteligencia comienza con algo de conocimiento. Sin embargo, conforme se interactúa con el mundo y se ganan experiencias, se aprende a adaptarse al entorno mejorando así la manera de realizar tareas. En aprendizaje automático, generalmente podemos encontrar dos técnicas fundamentales:

1. **Aprendizaje supervisado**: el entrenamiento del modelo esta basado en usar un histórico de datos (observaciones) para realizar predicciones sobre nuevos datos presentados.

2. **Aprendizaje no supervisado**: el modelo por sí solo intenta encontrar patrones dentro de un conjunto de datos y asignar una agrupación lógica con el objetivo de realizar una clasificación del espacio.

El Aprendizaje por Refuerzo (RL) es el tercer algoritmo de aprendizaje automático [1] que es igualmente de importante que los anteriores para la evolución de la inteligencia artificial. Los algoritmos de (RL) se basan en lo siguiente:

En los algoritmos de RL:

- **Agente**: está conectado a un entorno con su observación y acción. El agente percibe el entorno de una manera única y decide que acción tomar basándose en alguna técnica. En cada paso, el agente recibe señales que representan el estado del entorno en el que se desenvuelve. El agente responde con una acción concreta de un conjunto de acciones posibles.

- **Entorno**: es el mundo en el que el agente se desenvuelve e interactúa. El agente puede interactuar con el entorno realizando alguna acción, pero no puede influir en las reglas o la dinámica del entorno mediante esas acciones.

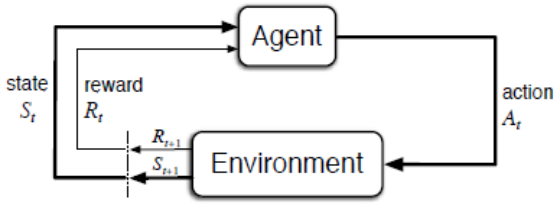

Figura 2.5: Esquema del algoritmo RL [8]

- **Acción**: La acción genera un cambio en el estado del entorno. El agente busca siempre mejorar sus resultados y para ello debe tomar las acciones adecuadas en función del entorno y el objetivo marcado.

- **Observación**: El cambio de estado del entorno, dada una acción concreta, es evaluado y comunicado de vuelta al agente mediante una señal de refuerzo (*reward*).

- **Recompensa**: esta señal de recompensa define el objetivo en un problema de aprendizaje por refuerzo. En cada paso de tiempo, el entorno envía al agente un valor númerico, es decir, una recompensa que depende de la evaluación efectuada sobre el cambio desde un estado a otro nuevo teniendo en cuenta la acción tomada que produjo dicho cambio. El único objetivo del agente es maximizar la recompensa total que recibe a largo plazo.

El aprendizaje por refuerzo se basa en tomar decisiones óptimas utilizando experiencias pasadas. Hay dos tipos principales de algoritmos en RL y estos dependen de si se emplea un modelo conocido o no. Este trabajo está centrado únicamente los algoritmos sin modelo (*model-free*) los cuales buscan aproximar un comportamiento (*policy*) óptimo sin necesidad de conocer la dinámica del entorno.

2.5.1. Q-Learning

Q-Learning es un algoritmo que no requiere el conocimiento de un modelo *model-free* siendo útil para situaciones cuando el agente sabe todos los posibles estados en los que se puede encontrar y las acciones que puede tomar, debiendo de elegir entre valores de pares estado-acción que reporten una recompensa inmediata o a largo plazo, lo cual proporciona optimización para llegar al objetivo de maximizar la recompensa acumulada a lo largo del tiempo.

Está basado en valores (*values*) y su función es la converger a un valor óptimo $Q^*(s,a)$ aplicando para ello una función de actualización de valor (*value-function*) por por Diferencia Temporal $TD(0)$ [8].

$$Q^*(s_t,a_t) = Q\left(s_t,a_t\right)(1-\alpha) + \alpha\left(R_{t+1} + \gamma \max_a Q\left(s_{t+1},a\right)\right) \tag{2.4}$$

donde:

- $Q\left(s_t,a_t\right)(1-\alpha)$ es el valor actual ponderado por la tasa de aprendizaje α. Los valores de esta tasa próximos 1 hacen que los valores de Q cambien más rápidamente.

- αR_{t+1} es el valor ponderado de la recompensa obtenida.

- $\alpha\gamma \ \max_a Q\left(s_{t+1},a\right)$ es la recompensa máxima que se puede obtener del estado $S_t + 1$ dada una acción, ponderada por la tasa de aprendizaje y el factor de descuento γ ($discount factor$). Este factor de descuento puede tener valor $0 \leq \gamma \leq 1$ y su función es la de tener en cuenta recompensas inmediatas ($\gamma \longrightarrow 0$) o durante un largo periodo de tiempo ($\gamma \longrightarrow 1$)

En este algoritmo, se crea una tabla llamada Q-tabla (*Q-table*) o matriz en donde sus filas corresponden a los posibles estados y las columnas a las posibles acciones a tomar para cada estado. Los valores en cada una de las celdas de la matriz se conocen como *q-values* y representan el mayor valor esperado de recompensa que se puede obtener tomando acción concreta en ese estado y siguiendo el comportamiento actual (*policy*). Esta tabla Q es un tabla de referencia que sirve para el agente seleccione la mejor acción basándose en el valor $q(s, a)$.

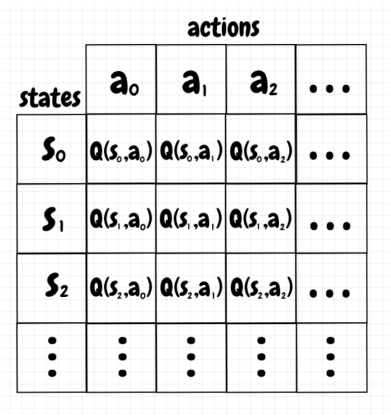

Figura 2.6: Q-table (*nestedsoftware.com*)

En cada paso que el agente toma se evalúa el siguiente estado dada la acción tomada mediante una recompensa y esta es usada para actualizar la tabla Q. Es un proceso iterativo en el que la velocidad de convergencia a unos valores q^* óptimos depende fuertemente de la cantidad de estados y cantidad de acciones posibles a tomar en cada estado.

dado que los valores q contienen la información del valor esperado de recompensa, es fácil deducir que para un estado dado, la mejor acción a tomar es la que tenga un valor $q(s, a)$ máximo. Esta forma de comportamiento es llamada *greedy-policy* y si siempre se optara por este método de decisión, una vez alcanzado el objetivo del algoritmo de aprendizaje, el agente quedaría constantemente repitiendo los mismos pasos dejando así buena parte del espacio de estados sin explorar, siendo muy frecuente que existan otras alternativas que inicialmente reportan una recompensa baja pero que a la larga la recompensa acumulada es mayor. Para evitar esto, hay un método llamado *ε-greedy-policy* el cual se basa en elegir aleatoriamente una acción distinta a la correspondiente con el máximo valor de $q(s, a)$. Esa aleatoriedad viene dada por el valor de una probabilidad marcada por la variable ε. Se fija un valor para esta y en cada paso se obtiene un valor aleatorio que es comparado con el valor de ε. Si el valor aleatorio es menor que ε, entonces se elige una acción siguiendo una *ε-greedy-policy*, de lo contrario se sigue una $greedy - policy$. Con ello se asegura una exploración más amplia del espacio.

Sin embargo, cuando se actúa siguiendo una estrategia *ε-greedy* existen otras maneras diferentes y más óptimas de seleccionar un estado distinto al correspondiente con el valor máximo $q(s, a)$ en lugar de hacerlo de manera aleatoria. Un ejemplo de ellas es mostrada en [8] y consiste en emplear la siguiente ecuación:

$$A_t = \underset{a}{\operatorname{argmax}} \left[Q_t(a) + c \sqrt{\frac{\operatorname{Ln}(t)}{N_t(a)}} \right] \tag{2.5}$$

donde t es el instante de tiempo, $c > 0$ es el grado de exploración $Q_t(a)$ es el valor q para tal acción y $N_t(a)$ es el número total de recompensas obtenidas por dicha acción. Esta ecuación es denominada *upper confidence bound (UCB)* y con ella se consigue seleccionar una acción potencialmente optima de entre todas las que no son *greedy* teniendo en cuenta como de cerca están sus estimaciones de ser optimas y las incertidumbres de dichas estimaciones.

Respecto a la inicialización de los valores de las tablas Q, esta suele ser fijando dichos valores a cero. Por otro lado, en [8] se muestra un método simple pero eficaz de impulsar una exploración del espacio por parte del agente mediante una simple inicialización de estos valores a un valor positivo, el cual debe de ser suficientemente alto para que independientemente de las acciones iniciales seleccionadas la recompensa sea inferior a este valor. Con ello el algoritmo de aprendizaje va cambiando a otras acciones explorando de esta manera el entorno. El resultado es que todas las acciones resultan ser probadas varias veces antes de que los valores converjan.

Una vez implementados los pasos descritos, ya solo queda el entrenamiento del agente en un entorno durante un periodo de tiempo lo suficientemente largo para que los valores $q(s, a)$ estén próximos de sus valores óptimos $q^*(s, a)$. Hay que tener muy presente de que cuanto mayor sea la cantidad de estados y posibles acciones a tomar para cada estado, el agente necesitará adquirir más experiencia, lo que se traduce a un incremento del gasto computacional que en ocasiones no es factible.

2.6. Programación orientada a objetos en control

Históricamente, la creación de una aplicación de software para un sistema de control implicaba el desarrollo de algoritmos y estructuras de datos que resolvían un problema en concreto. El problema podría resolverse en una serie de pasos (o procedimientos). Este enfoque de programación se conoce como programación imperativa o procedimental. Dicho enfoque para el desarrollo de software funciona muy bien si el problema solo necesita resolverse una vez y si su tamaño es relativamente pequeño [13].

A medida que se aumentaba el tamaño de los problemas que se resolvían con aplicaciones informáticas para sistemas de control, también aumentaba la complejidad de estos problemas. Con este aumento de la complejidad, la programación tradicional se volvió más engorrosa y compleja y el programa tratado era a menudo difícil de describir, mantener y ampliar.

Con el objetivo de reducir la cantidad de código que había que desarrollar se trató de mejorar la reutilización. Un avance fundamental en el diseño de software para sistemas y control fue el concepto de abstracción de datos utilizando objetos. En lugar de desarrollar software modelando el flujo del programa, el software se modela utilizando objetos. Los objetos son los sustantivos del sistema. Un objeto es una unidad de modularidad estructural y de comportamiento que tiene propiedades dentro del código. Un objeto no solo encapsula las decisiones de diseño, sino que también encapsula el comportamiento, la identidad, el estado e incluso las reglas del sistema. El proceso de representar un problema como un conjunto de objetos que cooperan y las relaciones entre los objetos se conoce como Programación Orientada a Objetos.

Usando técnicas de diseño programación orientada a objetos, se puede abstraer el dominio del problema con un conjunto de objetos. Cada objeto encapsula completamente una parte del problema a resolver. Las técnicas de programación orientada a objetos representan entidades en el dominio del problema con objetos en código. Esta abstracción puede ayudar en la conceptualización del programa y la comunicación del diseño entre desarrolladores.

Los tres principios de la programación orientada a objetos son: encapsulación, herencia y polimorfismo; los tres son los pilares de la reutilización del código. La encapsulación es una técnica de modelado e implementación que separa los aspectos externos del objeto de los detalles internos de implementación. La herencia permite ampliar los objetos existentes sin cambiar el código de trabajo original o el contexto. El polimorfismo (enlace dinámico) permite que los algoritmos probados (como objetos) sean reutilizados.

2.7. Trabajos relacionados

En [20] podemos encontrar el trabajo de modelado del un AGV híbrido-triciclo, en el cual este trabajo se ha basado, de manera mas resumida que en [1]. En el se obtiene un modelo completo y detallado sobre este tipo de AGV, al que se le somete a distintas condiciones y se analiza su respuesta dinámica.

En [21] podemos ver un trabajo sobre como dotar de inteligencia suficiente a un conjunto de AGVs de manera que sean capaces de realizar una tarea de transporte de manera colaborativa en un entorno industrial, consiguiendo de esta manera trasladar grandes cargas con maquinas pequeñas.

En [22] se desarrolla una herramienta de simulación, basada en IEC-61131, para poder valorar el rendimiento de controladores de seguimiento para AGVs sobre distintas trayectorias ya que el ajuste de un controlador en una trayectoria puede no ser valido para otras. La herramienta diseñada es de gran utilidad para hacer un ajuste fino de los controladores.

En [23] se propone una alternativa en lo que respecta a la seguridad del AGV durante su movimiento en entornos dinámicos, mediante el uso de banda ultraancha (UWB), equipando al AGV de sensores capaces de localizar tags en personas u objetos. Con esta alternativa se consigue reducir el coste que supone la instalación de elementos hardware en las infraestructuras.

En [24] se puede encontrar un trabajo donde se utilizan técnicas de aprendizaje por refuerzo para el control de seguimiento de un AGV, ya que el desgaste de las partes electromecánicas provocan desviaciones de las trayectorias. Los resultados obtenidos sobre trayectorias complejas superan con diferencia a los obtenidos por un controlador PID.

Capítulo 3

Modelado del AGV

3.1. Configuración del AGV

Existen diversos tipos de AGV comerciales, cada uno con diferentes características cinemáticas y dinámicas. La cinemática del AGV viene determinada, entre otras cosas, por el número y posición de las ruedas. De entre todas las configuraciones posibles, las más típicas, que son las que se emplean en este trabajo, son [3]:

- **Configuración Diferencial**: Dispone de dos ruedas de tracción. El eje de rotación está en el medio del eje de tracción fig. 3.1. Habitualmente dispone de una o dos ruedas locas de apoyo.

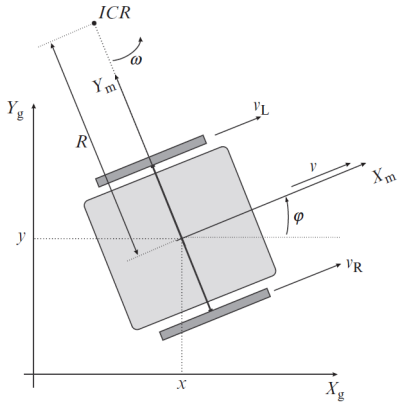

Figura 3.1: Robot Diferencial

La cinemática de esta configuración viene dada por las siguientes ecuaciones:

$$\begin{bmatrix} \dot{x}_{m(t)} \\ \dot{y}_{m(t)} \\ \dot{\varphi}\,(t) \end{bmatrix} = \begin{bmatrix} v_{x_m}\,(t) \\ v_{y_m}\,(t) \\ w\,(t) \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ -\frac{r}{L} & -\frac{r}{L} \end{bmatrix} \begin{bmatrix} w_L\,(t) \\ w_R\,(t) \end{bmatrix} \tag{3.1}$$

donde $w_L(t)$, $w_R(t)$, $v_{x_m}(t)$ y $v_{y_m}(t)$ son las velocidades angulares y tangenciales de las ruedas respectivamente, r es el radio de las ruedas, L es la distancia entre las ruedas y w es la velocidad angular con la que gira el robot alrededor de su centro instantáneo de rotación $ICR(m)$.

- **Configuración Triciclo**: Dispone de dos ruedas traseras y de una rueda delantera (o trasera) orientable fig. 3.2. Sólo se puede actuar sobre dos variables, habitualmente la orientación de la rueda delantera y su velocidad de avance. En este caso las ruedas traseras son libres. Tiene mejores características en cuanto a la adherencia (menor derrapaje). Además, permite sensorizar más ruedas (sensorización redundante), con lo que se pueden realizar estimaciones más precisas.

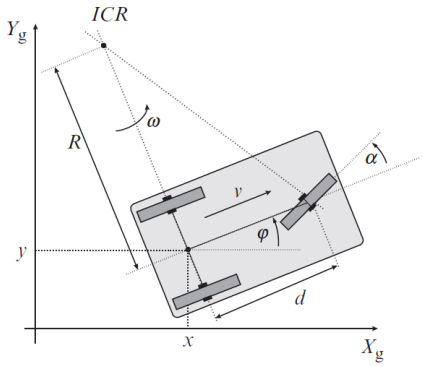

Figura 3.2: Robot Triciclo

para esta configuración, la cinemática es:

$$\begin{aligned} \dot{x} &= v_s(t)\cos(\alpha(t))\cos(\varphi(t)) \\ \dot{y} &= v_s(t)\cos(\alpha(t))\sin(\varphi(t)) \\ \dot{\varphi} &= \frac{v_s(t)}{d}\sin(\alpha(t)) \end{aligned} \tag{3.2}$$

donde $\dot{x}$ y $\dot{y}$ son las velocidades en los ejes x e y del sistema no inercial, v_s es la velocidad tangencial de la rueda motriz, α es el ángulo que forma la rueda motriz respecto al sistema de referencia embarcado en el vehículo, $\dot{\varphi}$ es la velocidad angular con la que gira el robot alrededor de su centro instantáneo de giro ICR y d la distancia entre el eje trasero y el delantero.

3.2. Características del Easybot Standard 410-480

La información sobre las características del AGV comercial Easybot Standard 410-480, con una configuración híbrida triciclo-diferencial, se pueden obtener de la página web del fabricante, ASTI Mobile Robotics, *http://www.astimobilerobotics.com/* en donde, aparte de la hoja de características de este AGV, se puede encontrar diversa información sobre aplicaciones y distintos tipos de AGV en función de su carga máxima a transportar. Una vista general sobre este AGV y sus dimensiones se muestra en la figura 3.3:

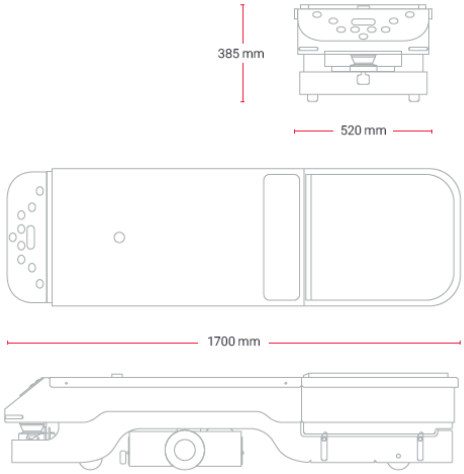

Figura 3.3: Easybot Standard 410-480

Dado que no toda la información relativa a distancias de los elementos del AGV se muestra en la hoja de características, se hace una aproximación de algunas necesarias para la parametrización del modelo dinámico. Los parámetros y sus valores a emplear quedan recogidos en la siguiente tabla:

	Descripción	**Valores**
dD	Distancia del eje del bloque diferencial al sensor	$0.2m$
L	Distancia entre trasero y delantero	$0.9m$
B	Distancia entre centros de ruedas	$0.5m$
Ws	Longitud del sensor	$0.5m$
d	Distancia entre centro del eje diferencial y extremo frontal	$0.58m$
m_{AGV}	Peso del AGV	$200kg$

Tabla 3.1: Parámetros Easybot Standard 410-480

En lo que respecta al sensor magnético embarcado, la figura 3.4 muestra su ubicación y anchura así como las partes en las que se divide el AGV comercial (ASTI):

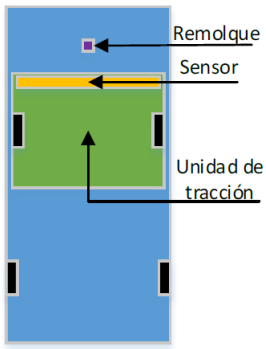

Figura 3.4: Esquema del AGV [1]

Como se puede observar, el sensor magnético está embarcado en la unidad de tracción y se asume que su anchura total es la misma que la de la unidad de tracción. Esto difiere de las dimensiones dadas por fabricantes de este tipo de sensor, pero es necesario para poder realizar simulaciones más largas durante la sintonización de los controladores.

3.3. Modelo dinámico del AGV

En el trabajo [1] sobre el modelado de este AGV se puede ver un modelo bastante completo sobre el que se tratan puntos como: modelo cinemático, modelo de las rueda, dinámicas de la unidad de tracción y cuerpo del AGV, modelado de cargas verticales, laterales, derrape, vuelco, aceleración y frenado, interacción con el suelo y modelo de carga remolcada.

Dado que el objetivo de este trabajo es diseñar un control adecuado y adaptativo, únicamente se seleccionan ciertos puntos sobre el trabajo mencionado y se crea un modelo funcional, para posteriormente aplicarle las técnicas de control descritas. Estos puntos son los siguientes:

- Modelo de las ruedas
- Modelo de los motores DC
- Modelo de la unidad de tracción

- Modelo del cuerpo del AGV
- Modelado de las cargas verticales en las ruedas
- Desplazamiento del centro de gravedad

3.3.1. Modelo de las ruedas

Cuando a una rueda se le aplica un par, se genera una fuerza tracción F en el punto de contacto A entre la rueda y el suelo, a la que se le opone una fuerza de rozamiento f [1] provocando como resultado el giro de la rueda sobre la superficie fig. 3.5:

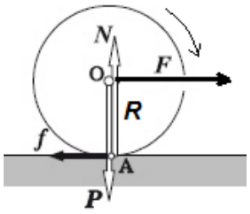

Figura 3.5: Rodadura

Para que la rueda no deslice, se debe cumplir en todo momento que la fuerza de tracción no supere a la de rozamiento.

$$f = \text{masa} \cdot g \cdot \mu \geq F \tag{3.3}$$

donde m es la masa soportada por la rueda, g la aceleración de la gravedad y μ el coeficiente de rozamiento con el suelo.

3.3.2. Modelo de los motores

Actualmente, los tipos de motores más empleados para este tipo de vehículos son los motores de corriente continua sin escobillas (*DC Brushless*) ya que poseen numerosas ventajas frente a los motores de corriente continua con escobillas. Estas ventajas, entre otras, son: mejor relación velocidad-par motor, mayor respuesta dinámica, mayor eficiencia, menor ruido, menor mantenimiento, etc. Sin embargo, y debido únicamente a motivos de simplicidad, se opta en este trabajo por usar motores estándar de tipo DC fig. 3.6

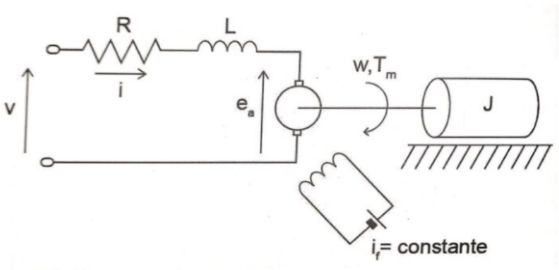

Figura 3.6: Elementos del motor DC

La armadura del motor DC se modela como si tuviera una resistencia constante R en serie con una inductancia constante L que representa la inductancia de la bobina de la armadura, y una fuente de alimentación v es la tensión generada en la armadura, T_m el par generado por el motor, e_a la fuerza contraelectromotriz que aparece cuando los conductores de la armadura se mueven a través del flujo de campo establecido por la corriente del campo i_f, Bm el coeficiente de fricción, J momento de inercia de la carga y w la velocidad angular a la que gira el motor.

Estableciendo la una relación proporcional, K_b (constante contraelectromotriz), entre el voltaje inducido en la armadura y la velocidad angular del eje del motor, se tiene:

$$E_a = K_b\, w \tag{3.4}$$

otra relación electromecánica establece que el torque mecánico es proporcional, K_m (constante de torque), a la corriente eléctrica que circula por el motor DC:

$$T_m = K_m\, w \tag{3.5}$$

por lo que se obtienen (sin entrar en detalles sobre el desarrollo) las ecuaciones de los motores izquierdo y derecho del AGV:

$$\dot{i_L} = \frac{u_L}{L_m} - \frac{R_m\, i_L}{L_m} - \frac{K_b\, w_L}{L_m} \tag{3.6}$$

$$\dot{w_L} = \frac{K_m\, i_L}{J_L} - \frac{B_m\, w_L}{J_L} \tag{3.7}$$

$$T_{\mathrm{mL}} = \frac{K_m\, i_L - B_m\, w_L}{f} \tag{3.8}$$

$$F_L = \frac{T_{\mathrm{mL}}}{r} \tag{3.9}$$

$$\dot{i_R} = \frac{u_R}{L_m} - \frac{R_m\, i_R}{L_m} - \frac{K_b\, w_R}{L_m} \tag{3.10}$$

$$\dot{w_R} = \frac{K_m\, i_R}{J_R} - \frac{B_m\, w_R}{J_R} \tag{3.11}$$

$$T_{\mathrm{mR}} = \frac{K_m\, i_R - B_m\, w_R}{f} \tag{3.12}$$

$$F_R = \frac{T_{\mathrm{mR}}}{r} \tag{3.13}$$

donde los subíndices R y L hacen referencia al motor derecho *Right* e izquierdo *Left* respectivamente, F_R y F_L son las fuerzas que los motores aplican al AGV y Los parámetros r y f son el radio de las ruedas en metros y el coeficiente de reducción de la caja de engranajes respectivamente

Las velocidades angulares de los motores se ven afectadas por el peso vertical que soporta cada una ellas a través de los momentos de inercia de las mismas. Para simplificar el sistema, estos momentos de inercia se calculan como el momento de inercia del conjunto motor-engranajes (valor del fabricante) más el momento de inercia de un cilindro sólido. Esto realmente no es del todo cierto ya que en un cálculo exacto de estos momentos de inercia se observa que éste depende de la aceleración de las propias ruedas. La consecuencia de usar un momento de inercia simplificado, dado que las fuerzas de los motores se usan como fuerza de entrada al conjunto unidad de tracción-cuerpo del AGV, es un descuadre en la velocidad lineal de las ruedas y la del centro de gravedad del AGV. Para paliar este inconveniente, en el apartado del modelo del cuerpo del AGV se muestra una solución válida para este trabajo.

Debido a la dificultad de encontrar todos los parámetros de motores comerciales de corriente continua, algunos de ellos se obtienen de otros trabajos similares a éste [7]. Con un reajuste de valores prueba y error en simulaciones para dar finalmente con unos valores que se consideran válidos para la respuesta deseada de dichos motores. El resultado se recoge en la tabla 3.2:

	Descripción	**Valores**
R_m	Resistencia	1.3Ω
L_m	Inductancia	$0.00005H$
K_b	Constante fuerza contra electromotriz	$0.36V/rad/s$
K_m	Constante de la armadura	$0.5Nm/A$
B_m	Fricción viscosa	$0.58N\ m\ s$
f	Coeficiente de reducción	1
r	Radio de las ruedas	$0.075m$
J_m	Momento del inercia del conjunto motor-reductor (fabricante)	$0.0640kg\ mm^2$

Tabla 3.2: Parámetros de los motores

3.3.3. Modelo de la unidad de tracción

Los motores transmiten la fuerza a la unidad de tracción y ésta al cuerpo del AGV. Para la obtención del modelo dinámico de la unidad de tracción junto con la interacción entre el cuerpo del AGV y dicha unidad, se realiza el análisis sobre el balance de fuerzas y momentos aplicados al centro de gravedad. Dado que la unidad de tracción va unida al cuerpo del AGV, el análisis se realiza respecto a un sistema de referencia embarcado en el cuerpo del AGV y que coincide con la posición del par cinemático entre ambos elementos. Dado el esquema fig. 3.7:

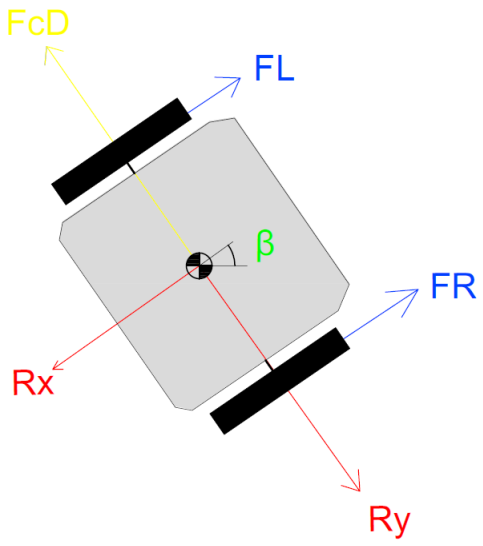

Figura 3.7: Fuerzas sobre unidad diferencial

el análisis de fuerzas y momentos viene dado por las siguientes ecuaciones:

$$-R_x + F_R + F_L = \dot{v}_{Dx}\, m_1 \tag{3.14}$$

$$-R_y + F_{\mathrm{cD}} = \dot{v}_{Dy}\, m_1 \tag{3.15}$$

$$-\frac{B\,(-F_R + F_L)}{2} = I_D\, \dot{w}_D \tag{3.16}$$

$$\dot{\beta} = w_D \tag{3.17}$$

donde:

- R_x y R_y son las fuerzas de reacción en el par cinemático de rotación entre la unidad de tracción y el cuerpo del AGV (N)
- w_D es la velocidad angular con la que gira la unidad de tracción (rad/s)
- F_L y F_R son las fuerzas aplicadas por los motores (N)
- I_D es el momento de inercia de la unidad de tracción (kgm^2)
- F_{cD} es la fuerza centrípeta que actúa sobre el centro de gravedad (N)
- β es el ángulo que forma la unidad de tracción respecto al sistema de referencia embarcado en el cuerpo del AGV (rad)

Por simplicidad, el momento de inercia de la unidad de tracción se considera el de un paralelepípedo rectangular. Respecto a la fuerza centrípeta, no se expone aquí el desarrollo para su cálculo, pero se obtiene mediante relaciones trigonométricas entre el radio de giro de la unidad de tracción y el del cuerpo del AGV.

Al no poder separarse el origen del sistema de referencia de la unidad de tracción del origen de un sistema de referencia embarcado sobre el par cinemático del cuerpo del AGV [1], tenemos que $\dot{v}_{Dx} = \dot{v}_{Dy} = 0$.

3.3.4. Modelo del cuerpo del AGV

Para modelar el cuerpo del AGV, que tiene una configuración triciclo y con un sistema de referencia embarcado con origen en el centro del eje trasero, se sigue el mismo procedimiento que en el caso de la unidad de tracción. En la figura 3.8 se pueden apreciar las fuerzas y momentos que actúan sobre el centro de gravedad y que se asume que puede estar descentrado como consecuencia de cargas añadidas:

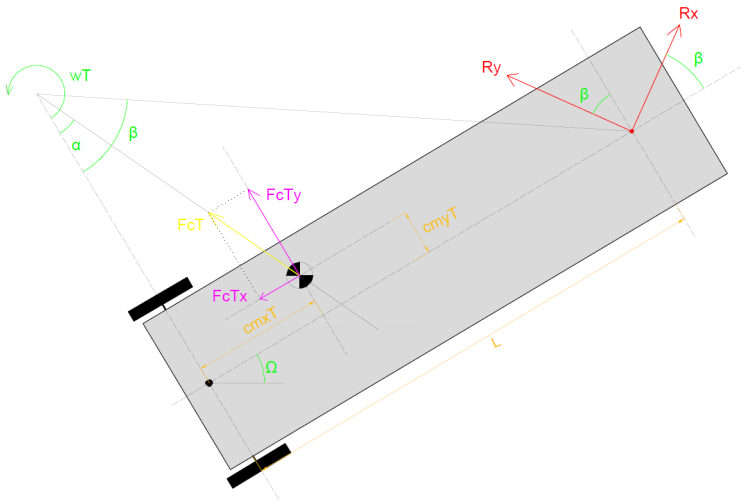

Figura 3.8: Esquema de fuerzas sobre cuerpo del AGV

Tal y como se observa en la figura 3.8, la fuerza centrípeta actúa sobre el centro de gravedad y se aplica en la dirección de la recta que une este con el centro de rotación [1]. A pesar de ello, y por simplificar nuevamente, se asume que dicha dirección será siempre en el eje y del sistema de referencia del cuerpo del AGV (y_T). Con esto el balance fuerzas y momentos queda:

$$F_{\mathrm{Tx}} = R_x \cos(\beta) - R_y \sin(\beta) \tag{3.18}$$

$$F_{\mathrm{Ty}} = R_y \cos(\beta) + R_x \sin(\beta) \tag{3.19}$$

$$F_{\mathrm{Tx}} = \dot{v}_{Tx}\, m_2 \tag{3.20}$$

$$F_{\mathrm{Ty}} + F_{\mathrm{cT}} = \dot{v}_{Ty}\, m_2 \tag{3.21}$$

$$F_{\mathrm{Tx}}\, \mathrm{cmy}_T + F_{\mathrm{Ty}} (L - \mathrm{cmx}_T) = I_T\, \dot{w}_T \tag{3.22}$$

$$\dot{\Omega} = w_T \tag{3.23}$$

donde:

- cmx_T y cmy_T son las coordenadas del centro de gravedad del AGV en el sistema embarcado del mismo (m).
- w_T es la velocidad angular con la que gira el AGV (rad/s)
- $\dot{v}_{Tx}$ y $\dot{v}_{Ty}$ son las aceleraciones del sistema de referencia del AGV (m/s^2)
- I_T es el momento de inercia del cuerpo del AGV (paralelepípedo rectangular) ($kg.m^2$)
- F_{cT} es la fuerza centrípeta que actúa sobre el centro de gravedad (N)
- Ω es el ángulo que forma el cuerpo del AGV respecto al sistema no inercial (rad)

La relación existente entre las velocidades del sistema de referencia embarcado y las del sistema de referencia no inercial, es decir, la cinemática, viene dada por las siguientes ecuaciones:

$$v_{\mathrm{T0x}} = v_{\mathrm{Tx}} \cos(\Omega) - v_{\mathrm{Ty}} \sin(\Omega) \tag{3.24}$$

$$v_{\mathrm{T0y}} = v_{\mathrm{Ty}} \cos(\Omega) + v_{\mathrm{Tx}} \sin(\Omega) \tag{3.25}$$

cuyas derivadas son:

$$\dot{v}_{T0x} = \dot{v}_{Tx} \cos(\Omega) - \dot{v}_{Ty} \sin(\Omega) - v_{\mathrm{Ty}}\, w_T \cos(\Omega) - v_{\mathrm{Tx}}\, w_T \sin(\Omega) \tag{3.26}$$

$$\dot{v}_{T0y} = \dot{v}_{Ty} \cos(\Omega) + \dot{v}_{Tx} \sin(\Omega) + v_{\mathrm{Tx}}\, w_T \cos(\Omega) - v_{\mathrm{Ty}}\, w_T \sin(\Omega) \tag{3.27}$$

Dado que se desea evitar los deslizamientos en el eje y, hay que establecer una restricción noholonómica. La cinemática se puede expresar por las siguientes ecuaciones:

$$v_{\mathrm{T0x}} = v_{\mathrm{Tx}} \cos(\Omega) \tag{3.28}$$

$$v_{\mathrm{T0y}} = v_{\mathrm{Tx}} \sin(\Omega) \tag{3.29}$$

igualando las ecuaciones 3.26 con 3.28 derivada y 3.26 con 3.29, tras realizar un poco de álgebra y despejando $v_{\dot{T}y}$, obtenemos el valor que dicha aceleración debe de tener en cada momento para que se cumpla la restricción de velocidad en el eje y, siendo ésta:

$$\dot{v}_{Ty} = \frac{v_{\mathrm{Ty}}\, w_T \cos(\Omega) + v_{\mathrm{Ty}}\, w_T \sin(\Omega)}{\cos(\Omega) - \sin(\Omega)} \tag{3.30}$$

Por último, para garantizar una rodadura sin deslizamiento de las ruedas y que todas las velocidades lineales del sistema cuadren, debido a la simplificación de los momentos de inercia de las ruedas, forzamos en cada iteración de la simulación a que las velocidades angulares de la estas obtengan un valor que es función de la velocidad del centro de masas del AGV. Es decir, por la cinemática sabemos que:

$$v_{Dx} = 0{,}5(w_R\, r + w_L\, r) = \frac{v_{Tx}}{cos(\beta)} \tag{3.31}$$

$$v_L = v_{Dx} - 0{,}5\; w_D\; B \tag{3.32}$$

$$v_R = w_D B + v_L \tag{3.33}$$

por lo que forzamos w_L y w_R a:

$$w_R = \frac{v_R}{f\; r} \tag{3.34}$$

$$w_L = \frac{v_L}{f\; r} \tag{3.35}$$

donde f es el factor de reducción y r el radio de las ruedas. A pesar de que esto introduce pequeñas discontinuidades en el sistema, los resultados de las simulaciones son completamente válidos para el propósito del trabajo.

La posición del AGV (centro geométrico del eje trasero) viene dada por:

$$x_T = \int v_{\mathrm{T0x}}\ \mathrm{dt} \tag{3.36}$$

$$y_T = \int v_{\mathrm{T0y}}\ \mathrm{dt} \tag{3.37}$$

3.3.5. Desplazamiento del centro de gravedad

La colocación de cargas sobre el AGV provoca un desplazamiento del centro de gravedad de éste fig. 3.9

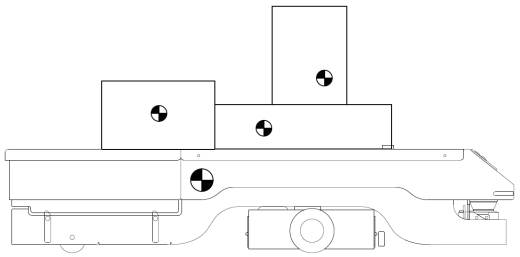

Figura 3.9: Distribución centros de gravedad

El cálculo de la nueva ubicación del centro de gravedad del AGV se obtiene mediante la siguiente ecuación:

$$\overline{r_G} = \frac{\sum_{i=1}^{m} m_i\ g_i\ \overline{r_i}}{\sum_{i=1}^{m} m_i\ g_i}$$

donde:

- $\overline{r_G}$ es el vector posición del centro de gravedad resultante (m)
- m son las masas de los centros de gravedad de cada elemento (kg)
- g_i es la aceleración de la gravedad que actúa sobre cada centro de gravedad de los elementos (m/s^2). Si ésta no varía, puede ser cancelada.

$$v_{Dx} = 0{,}5(w_R\ r + w_L\ r) = \frac{v_{Tx}}{cos(\beta)} \tag{3.38}$$

$$v_L = v_{Dx} - 0{,}5\ w_D\ B \tag{3.39}$$

$$v_R = w_D B + v_L \tag{3.40}$$

3.3.6. Modelado de las cargas verticales en las ruedas

Para el cálculo de la distribución de la carga sobre las ruedas se asume que cuando el AGV está en movimiento, las fuerzas que actúan sobre él producen deslizamientos de la carga transportada. Esta simplificación se realiza ya que únicamente es de interés conocer el desplazamiento del centro de gravedad del AGV y el peso que soportan cada una de las ruedas para el cálculo de su momento de inercia, el cual influirá a su vez en el comportamiento del control.

Con esto, y una vez calculada la nueva ubicación del centro de masas según el apartado anterior, se procede al cálculo de la distribución del peso mediante el balance de fuerzas y momentos respecto a dos ejes de giro distintos del AGV cuando éste se encuentra estático.

Sin entrar en detalles sobre el desarrollo en [1] se puede encontrar una explicación detallada. Las ecuaciones para el cálculo de la distribución de cargas en el AGV son:

$$m = m_{\text{AGV}} + m_{\text{Load}} \tag{3.41}$$

$$m_D = \frac{\text{cmx}_T \; m}{L} \tag{3.42}$$

$$m_{rL} = \frac{\text{cmy}_T \; m}{B} + \frac{m \; (L - \text{cmx}_T)}{2 \; L} \tag{3.43}$$

$$m_{rR} = \frac{m \; (L - \text{cmx}_T)}{2 \; L} - \frac{m \; \text{cmy}_T}{B} \tag{3.44}$$

$$m_R = m_L = \frac{m_D}{2} \tag{3.45}$$

donde:

- m_{AGV} es la masa nominal del AGV (kg)
- m_{Load} es la suma de todas las masas añadidas al AGV (kg)
- m_D es la carga sobre el eje delantero (kg)
- m_L y m_R son la carga sobre las ruedas de tracción izquierda y derecha, respectivamente (kg)
- m_{rL} y m_{rR} son la carga sobre las ruedas traseras izquierda y derecha, respectivamente (kg)

3.4. Resolución del sistema de ecuaciones

Una vez obtenidas todas las ecuaciones sobre los distintos puntos para seleccionar el modelo del AGV usado en este trabajo, se procede a la resolución de éstas para obtener el modelo final. Puesto que tenemos un sistema de ecuaciones diferenciales algebraicas (DAE), resolvemos éste mediante el análisis de su matriz de incidencia. Tomando las ecuaciones 3.6-3.18, 3.26-3.27, 3.36-3.37 y sustituyendo el valor de la ecuación 3.30 en las anteriores, formamos su matriz de incidencia en la que tras realizar las necesarias permutaciones se consigue formar una matriz BLT (*Block Lower Triangular*) lo que indica que el sistema de ecuaciones DAE es no singular [2].

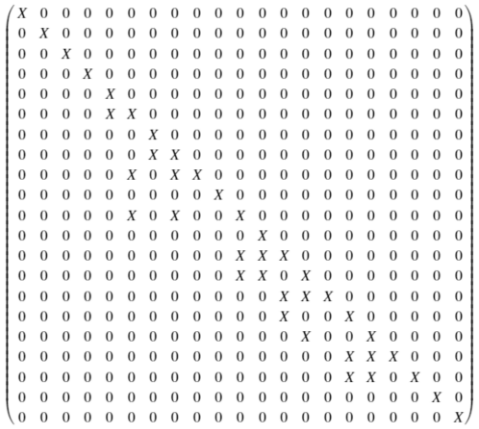

Figura 3.10: Matriz BLT

en donde las columnas de la figura 3.10 están formadas por las variables del siguiente vector fila en el mismo orden:

$$\left(\begin{array}{cccccccccccccccccccc} \dot{i}_L & \dot{w}_L & \dot{i}_R & \dot{w}_R & T_{\mathrm{mR}} & F_R & T_{\mathrm{mL}} & F_L & \dot{w}_D & \dot{\beta} & R_x & R_y & F_{\mathrm{Tx}} & F_{\mathrm{Ty}} & \dot{w}_T & \dot{v}_{Tx} & v_{\mathrm{Ty}} & \dot{v}_{T0x} & \dot{v}_{T0y} & \dot{x}_T & \dot{y}_T \end{array} \right)$$

y sus filas son las ecuaciones dadas por el siguiente vector columna en el mismo orden:

$$
\left(\begin{array}{c}
\dot{i}_L = \frac{u_L}{L_m} - \frac{R_m\, i_L}{L_m} - \frac{K_b\, w_L}{L_m} \\
\dot{w}_L = \frac{K_m\, i_L}{J_L} - \frac{B_m\, w_L}{J_L} \\
\dot{i}_R = \frac{u_R}{L_m} - \frac{R_m\, i_R}{L_m} - \frac{K_b\, w_R}{L_m} \\
\dot{w}_R = \frac{K_m\, i_R}{J_R} - \frac{B_m\, w_R}{J_R} \\
T_{\mathrm{mR}} = \frac{K_m\, i_R - B_m\, w_R}{f} \\
F_R = \frac{T_{\mathrm{mR}}}{r} \\
T_{\mathrm{mL}} = \frac{K_m\, i_L - B_m\, w_L}{f} \\
F_L = \frac{T_{\mathrm{mL}}}{r} \\
-\frac{B\,(-F_R + F_L)}{2} = I_D\, \dot{w}_D \\
\dot{\beta} = w_D \\
-R_x + F_R + F_L = 0 \\
-R_y + F_{\mathrm{cD}} = 0 \\
F_{\mathrm{Tx}} = R_x\, \cos(\beta) - R_y\, \sin(\beta) \\
F_{\mathrm{Ty}} = R_y\, \cos(\beta) + R_x\, \sin(\beta) \\
F_{\mathrm{Tx}}\, \mathrm{cmy}_T + F_{\mathrm{Ty}}\, (L - \mathrm{cmx}_T) = \mathrm{I_T}\, \dot{w}_T \\
F_{\mathrm{Tx}} = \dot{v}_{Tx}\, m_2 \\
F_{\mathrm{Ty}} + F_{\mathrm{cT}} = \frac{m_2\, (\sigma_3 + \sigma_2)}{\sigma_1} \\
\dot{v}_{T0x} = \dot{v}_{Tx}\, \cos(\Omega) - \sigma_3 - v_{\mathrm{Tx}}\, w_T\, \sin(\Omega) - \frac{\sin(\Omega)\,(\sigma_3 + \sigma_2)}{\sigma_1} \\
\dot{v}_{T0y} = \dot{v}_{Tx}\, \sin(\Omega) + v_{\mathrm{Tx}}\, w_T\, \cos(\Omega) - \sigma_2 + \frac{\cos(\Omega)\,(\sigma_3 + \sigma_2)}{\sigma_1} \\
\dot{x}_T = v_{\mathrm{T0x}} \\
\dot{y}_T = v_{\mathrm{T0y}}
\end{array}\right) \tag{3.46}
$$

con

$$\sigma_1 = \cos(\Omega) - \sin(\Omega)$$

$$\sigma_2 = v_{\mathrm{Ty}}\, w_T\, \sin(\Omega)$$

$$\sigma_3 = v_{\mathrm{Ty}}\, w_T\, \cos(\Omega)$$

El sistema de ecuaciones completo con la causalidad computacional asignada se puede encontrar en el apéndice A. Con dicho sistema resuelto, y realizando las sustituciones pertinentes, se llega a un sistema de ecuaciones diferenciales de primer orden no lineal (ODE) de la forma:

$$
\begin{array}{l}
\dot{w}_T = f_1\,(i_L, w_L, i_R, w_R, \beta, \Omega, w_D, w_T, v_{\mathrm{Tx}}) \\
\dot{v}_{T_x} = f_2\,(i_L, w_L, i_R, w_R, \beta, w_D, v_{\mathrm{Tx}}) \\
\dot{v}_{T0x} = f_3\,(i_L, w_L, i_R, w_R, \beta, \Omega, w_D, w_T, v_{\mathrm{Tx}}) \\
\dot{v}_{T0y} = f_4\,(i_L, w_L, i_R, w_R, \beta, \Omega, w_D, w_T, v_{\mathrm{Tx}}\,) \\
\dot{w}_D = f_5\,(i_L, w_L, i_R, w_R) \\
\dot{\beta} = f_6\,(w_D) \\
\dot{i}_L = f_7\,(i_L, w_L, u_L) \\
\dot{w}_L = f_8\,(i_L, w_L) \\
\dot{i}_R = f_9\,(i_R, w_R, u_R) \\
\dot{w}_R = f_{10}\,(i_R, w_R) \\
\dot{x}_T = f_{11}\,(v_{\mathrm{T0x}}) \\
\dot{y}_T = f_{12}\,(v_{\mathrm{T0y}}) \\
\dot{\Omega} = f_{13}\,(w_T)
\end{array} \tag{3.47}
$$

el cual consta de un total de trece estados y no se muestra debido a la longitud de alguna de ellas. Este sistema del AGV resulta ser un sistema no lineal multivariable acoplado, en donde sus variables

manipuladas son las tensiones de los motores de corriente continua u_L y u_R, y las variables controladas son las velocidades longitudinales de las ruedas v_L y v_R.

3.5. Arquitectura de control

Para poder realizar una tarea, el AGV debe de ser capaz de seguir una trayectoria de la manera más precisa posible y manteniendo una velocidad de crucero constante. Para conseguir tal objetivo, la arquitectura de control general que se emplea se muestra en fig. 3.11

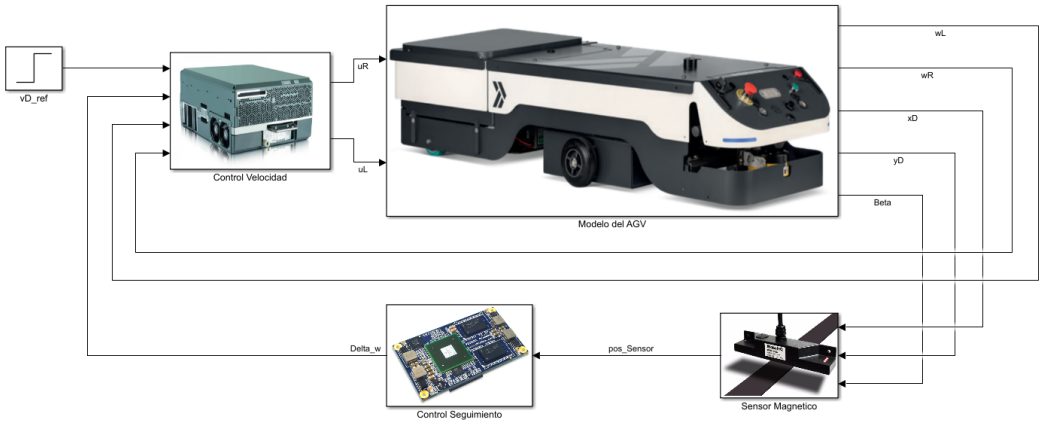

Figura 3.11: Esquema general de control

En ella se ve que dicha arquitectura consta de:

- un control de seguimiento el cual se encarga de que el AGV trace la trayectoria predefinida. La señal de entrada a dicho control es proporcionada por el sensor magnético y la señal de salida son los incrementos de velocidad angular Δw_D que la unidad de tracción debe realizar para corregir su trayectoria en caso de error. El sensor magnético proporciona un valor de distancia positiva desde su centro hacia su extremo izquierdo y negativa hacia su extremo derecho, la cual se considera como un error de seguimiento. Dicha salida del sensor varia desde -0.25 hasta 0.25, ya que se toma la anchura total del sensor con valor de 0.5m. Fuera de este rango, la distancia devuelta por el sensor es de cero exacto y se considera que el robot se ha salido de su trayectoria. El control de seguimiento se aplica con un periodo de muestreo de $100ms$.
- un control de velocidad que, dada una velocidad longitudinal de referencia para el desplazamiento del AGV, debe de mantener esta velocidad lo más constante posible al mismo tiempo que realiza una corrección de las velocidades de las ruedas ante un desvío de la trayectoria. Este control actúa directamente sobre los motores de las ruedas con un periodo de muestreo de $50ms$.

Con la adecuada combinación de los controles anteriores y una elección correcta de las técnicas de control a emplear, se consigue el comportamiento deseado del AGV.

Capítulo 4

Control de velocidad neuronal

4.1. Diseño del controlador PID

Puesto que el robot debe seguir una trayectoria que puede ser de cualquier forma, la sintonía de este control resulta ser bastante más complejo que el anterior. Por ello se opta por usar un algoritmo de búsqueda inteligente, concretamente un algoritmo genético, cuyo procedimiento se describe en el siguiente apartado.

4.1.1. Sintonización del control PID con algoritmo genético

La configuración del algoritmo genético para la sintonización del PID es la siguiente:

- Genes = 3 (K_p, K_d y K_i)
- Cromosomas: 40
- Generaciones: 100
- Población: 40
- Espacio de búsqueda: 0-8
- Método de selección: Torneo
- Probabilidad de cruce: 0.7
- Elitismo: 2

Respecto a la función de evaluación, se calcula con la simulación del AGV siguiendo una trayectoria durante un periodo de tiempo máximo de 12s, en la cual se busca minimizar el valor de la siguiente función de coste:

$$\text{loss} = C_1 \sum_{k=1}^{m} \text{atan}\left(\frac{d_{\text{sensor}_k}}{\text{dD}}\right)^2 + C_2 \sum_{k=1}^{m} {\Delta w_D}^2 \tag{4.1}$$

Con la que se penaliza tanto en ángulo que forma el punto de la trayectoria detectado por el sensor como las acciones de control agresivas. Con los coeficientes C1 y C2 se varia la importancia que tienen cada uno. Tras la ejecución del AG, los valores obtenidos son:

$$K_p = 0.763631 \quad K_d = 5.840928 \quad K_i = 0.162562$$

4.2. Diseño del controlador neuronal con modelo de referencia

Las redes neuronales han demostrado ser muy eficaces en cuanto a la identificación y control de modelos dinámicos ya que son muy buenas aproximando funciones no lineales, siempre y cuando hayan sido bien entrenadas en todo su espacio de trabajo. Dada la habilidad de estas para aprender funciones no lineales, son bastante populares y empleadas en un amplio rango de aplicaciones, destacando entre otras: modelado y control de sistemas dinámicos y procesado de señales [5].

En este trabajo se hará uso del tipo de controlador neuronal con modelo de referencia para el control de la velocidad de las ruedas del AGV.

La selección de este para el control de velocidad en lugar de usarlo para el control de seguimiento es debido que no se ha podido obtener un modelo matemático que relacione el error de seguimiento con una respuesta adecuada de la velocidad angular de la unidad de tracción. Esto mismo ocurre en el caso del control PID, pero dado que la ecuación de este es mucho más simple (solo tres parámetros) se consideró mejor opción para una sintonización con un AG. La sintonización de la red neuronal con un AG, aunque posible, resulta bastante más compleja en lo que refiere a cantidad de parámetros y distintas configuraciones de la red.

Otro motivo por la que el control neuronal se considera apropiado para corregir la velocidad es que al ser adaptativo se puede mantener una dinámica deseada para las velocidades de las ruedas ante variaciones en los parámetros del modelo o dinámica no modelada. Otra ventaja que este controlador ofrece es que para su uso no es necesario conocer el modelo a controlar ya que mediante identificación con la red neuronal se puede obtener una aproximación bastante precisa, a pesar de que en este trabajo si se dispone de tal modelo. De entre los distintos tipos de control adaptativos existentes, este es de los más usado ya que su implementación no es muy compleja. [5]. También hay que decir que obtener un ajuste fino final no es sencillo.

Existen distintas variantes en cuanto a la estructura se refiere, pero de entre ellas, se opta seguir la misma que está implementada en Simulink, ya que es su forma más estándar y dado que se dispone del modelo del sistema (lo cual facilita la etapa de identificación de la red del modelo) crear un modelo de referencia no es tarea difícil. Dicha estructura, obtenida de la pagina web de Matlab, es la siguiente fig. 4.1

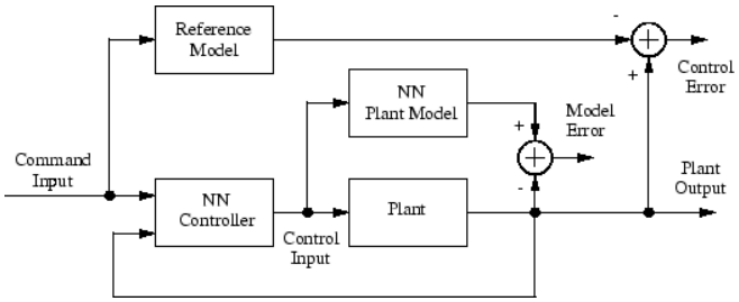

Figura 4.1: Esquema controlador neuronal con modelo de referencia

En el que su estructura interna esta formada por dos redes neuronales multicapa fig. 4.2:

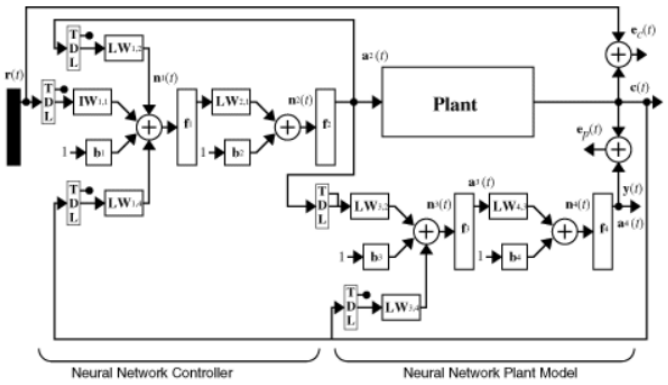

Figura 4.2: Esquema interno del controlador

Los componentes que forman este controlador son:

- **Modelo neuronal**: identificación del sistema a controlar. Este debe de ser lo más preciso posible
- **Modelo de referencia**: modelo neuronal (o no) cuya salida sea la dinámica deseada para la salida del sistema a controlar.
- **Controlador neuronal**: genera las acciones de control necesarias para controlar el sistema con la dinámica del modelo de referencia.

donde las entradas a las redes están formadas por vectores de entradas pasadas y la cantidad estas depende del orden del sistema, siendo mayor estas cuanto mayor es dicho orden.

4.2.1. Diseño del modelo neuronal

Como se comentó previamente, el modelo del AGV es un modelo MIMO acoplado, por lo que el modelo neuronal deberá aproximar tal sistema multivariable, teniendo como entradas principales las tensiones para cada uno de los motores u_R y u_L, y como salidas las velocidades tangenciales de las ruedas v_R y v_L. Se crea para ello una red neuronal con la siguiente configuración:

- **Número de entradas**: 30
- **Número de salidas**: 2
- **Número de capas**: 2
- **Número de neuronas en la capa oculta**: 40
- **Número de neuronas en la capa de salida**: 2
- **Método para actualización de pesos**: descenso por el gradiente con momento
- **Función de activación en capa oculta**: tangente sigmoidea
- **Función de activación en capa de salida**: lineal

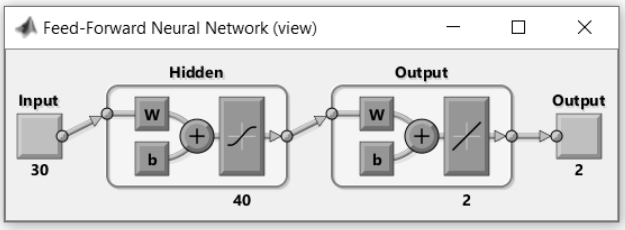

Figura 4.3: Estructura de la red modelo (imagen representativa)

Cada una de las dos redes neuronales dispone de dos capas en las que, dada su implementación, tanto el número de entradas, número de salidas y cantidad de neuronas en cada capa es ajustable. Como se puede ver en la figura 4.2 las entradas del modelo neuronal están compuestas por los siguientes vectores:

$$\begin{aligned} U_R &= [u_R(k),\ u_R(k-1),\ldots,\ u_R(k-6)] \\ U_L &= [u_L(k),\ u_L(k-1),\ldots,u_L(k-6)] \\ V_R &= [v_R(k-1),\ldots,v_R(k-8)] \\ V_L &= [v_L(k-1),\ldots,v_L(k-8)] \end{aligned} \tag{4.2}$$

La cantidad de regresores mostrada arriba es resultado de realizar varias pruebas en las que se comienza por un número bajo y se va incrementando hasta llegar a un resultado que se considera aceptable, intentando, siempre que sea posible, emplear el menor número posible para no incrementar el coste computacional.

Respecto a la implementación de esta red se puede resumir en los siguientes pasos [6]:

1. **Propagación de las entradas hacia adelante por la red neuronal**: se realiza mediante una operación matricial

$$y_m = f_2\left({W_2}^T\ f_1\left({W_1}^T\ x + B_1\right) + B_2\right) \tag{4.3}$$

   donde W_1 y W_2 son las matrices de pesos de la capa oculta y de salida respectivamente y B_1 y B_2 los vectores columna de los pesos de ambas capas. Estas matrices y vectores se estructuran de la siguiente manera:

$$W_n = \begin{bmatrix} w_{11}^n & w_{12}^n & . & . & w_{1j}^n \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ w_{i1}^n & w_{i2}^n & . & . & w_{ij}^n \end{bmatrix} \tag{4.4}$$

$$B_n = \begin{bmatrix} b_1 \\ . \\ . \\ b_i \end{bmatrix} \tag{4.5}$$

   siendo n en número de capa (1-2), i el número de entrada a la capa n y j el número de neurona de la capa n. En lo que respecta a las funciones de activación f_1 y f_2, con a como el nivel de activación de entrada a cada neurona:

$$f_1 = \frac{2}{1 + e^{-2a}} - 1 \qquad f_2 = a \tag{4.6}$$

2. **Actualización de pesos**: una vez obtenida la salida de la red neuronal, se calcula la aportación de cada peso al error total de la red mediante la siguiente función de coste:

$$E_m = \frac{1}{2} \sum_{i=1}^{n} (y_{p_i} - y_{m_i})^2 \tag{4.7}$$

donde E_m es el error en la aproximación del modelo e y_p son las salidas del sistema dinámico del AGV (velocidades de las ruedas). Una vez calculado el error del modelo, este se propaga hacia atrás en la red neuronal y se realiza el ajuste de los pesos usando el descenso por el gradiente mediante el mecanismo de *Backpropagation*. Este método calcula el gradiente del error con respecto a los pesos de la red neuronal. Para ello, primero de todo se definen las sensibilidades de las neuronas de cada capa como [6]:

$$S_i^n = \frac{\partial}{\partial a_i^n} E_m \tag{4.8}$$

donde a_i^n es el nivel de activación de entrada a la neurona i de la capa n. Una vez hecha esta definición, se procede al cálculo de las contribuciones de los pesos y biases de la capa de salida al error:

$$S_i^2 = \frac{\partial}{\partial h_i^2} E_m \; \frac{\partial}{\partial a_i^2} h_i^2 = \frac{\partial}{\partial h_i^2} E_m \;\; \dot{f}_i^2 \tag{4.9}$$

$$\frac{\partial}{\partial w_{\mathrm{ij}}^2} E_m = \sum_{j=1}^{m_2} S_j^2 h_i^1 \tag{4.10}$$

$$\frac{\partial}{\partial b_i^2} E_m = S_i^2 \tag{4.11}$$

donde $\dot{f}_i^2$ es la derivada de la función de activación de la neurona i en la capa de salida, h_i^1 es la salida de la función de activación de la neurona i en la capa anterior, es decir, la entrada a la capa de salida y m_2 el número total de neuronas en la capa de salida.

finalizado el paso anterior, se obtienen finalmente las contribuciones al error de los pesos y biases de la capa oculta de la siguiente manera:

$$S_i^1 = \dot{f}_i^1 \sum_{j=1}^{m_2} S_j^2 \; w_{\mathrm{ij}}^2 \tag{4.12}$$

$$\frac{\partial}{\partial w_{\mathrm{ij}}^1} E_m = \sum_{j=1}^{m_1} S_j^1 \; x_i \tag{4.13}$$

$$\frac{\partial}{\partial b_i^1} E_m = S_i^1 \tag{4.14}$$

donde m_1 es el número total de neuronas en la capa oculta.

Con el gradiente del error calculado, el método de actualización de pesos que se emplea es el descenso por el gradiente con momento. En lo que respecta al descenso por el gradiente, a pesar de que posee muchas virtudes, es un método bastante lento. La ecuación de este es la siguiente:

$$w_{\mathrm{ij}}^{n}\left(k+1\right)=w_{\mathrm{ij}}^{n}\left(k\right)-\alpha\,\frac{\partial}{\partial w_{\mathrm{ij}}^{n}}E_{m} \tag{4.15}$$

con α como tasa de aprendizaje y con un valor pequeño, el descenso por el gradiente realiza una actualización en cada iteración. La convergencia de este está asegurada aunque sea a un mínimo local. Sin embargo, presenta el problema de oscilar alrededor del espacio de búsqueda cuando el problema de optimización presenta grandes cantidades de curvatura o gradientes ruidosos, incluso puede llegar a atascarse en puntos planos del espacio de búsqueda que no tienen gradiente. Por dicha razón y en un intento de mejorar la velocidad de actualización de la red, se añade momento, el cual no es más que una extensión de descenso por el gradiente y está diseñado para acelerar el proceso de optimización consiguiendo así un resultado mejor. Esta modificación implica añadir un hiperparámetro extra (parámetro que no se actualiza automáticamente) el cual controla la cantidad de información pasada sobre los previos gradientes del error. De esta manera, la actualización actual se ve afectada por la influencia de la información pasada y la dirección de esta. La actualización de pesos y biases usando descenso por el gradiente con la adición del momento en forma matricial queda de la siguiente manera:

$$W_{v}^{n}\left(k+1\right)=\beta\,W_{v}^{n}\left(k\right)+\left(1-\beta\right)\frac{\partial}{\partial W^{n}}E_{m} \tag{4.16}$$

$$B_{v}^{n}\left(k+1\right)=\beta\,B_{v}^{n}\left(k\right)+\left(1-\beta\right)\frac{\partial}{\partial B^{n}}E_{m} \tag{4.17}$$

$$W^{n}\left(k+1\right)=W^{n}\left(k\right)-\alpha W_{v}^{n}\left(k+1\right) \tag{4.18}$$

$$B^{n}\left(k+1\right)=B^{n}\left(k\right)-\alpha B_{v}^{n}\left(k+1\right) \tag{4.19}$$

donde β es el nuevo hiperparámetro con valores entre 0 y 1, incluyendo mayor cantidad de información pasada cuanto mayor es este y n la capa de la red neuronal a ser actualizada.

4.2.2. Diseño del modelo de referencia

Con el modelo de referencia se establece el rendimiento requerido del sistema en bucle cerrado ya que con este se obtienen tanto el tiempo de subida, tiempo de establecimiento, sobreelongación y otras características. Este debe de ser elegido cuidadosamente para obtener la respuesta deseada y ha de tenerse en cuenta que la dinámica que este modelo establezca debe de poder ser alcanzable. Por este ultimo motivo, inicialmente se pensó en crear un modelo de referencia basado en el modelo del AGV controlado con un PI para las velocidades de las ruedas de manera que con ello se aseguraba estar próximo de la dinámica del AGV ante distintas condiciones. Posteriormente se optó por poner a prueba este tipo de control neuronal y se implementó como modelo de referencia un sistema lineal multivariable desacoplado el cual únicamente se basa en los modelos lineales de los motores (descritos en el capítulo de modelado) controlados por sendos PI y con ganancias ajustadas para tener una respuesta críticamente amortiguada y un tiempo de establecimiento de 2s ante una entrada escalón. El esquema de este modelo puede ver en fig. 4.4:

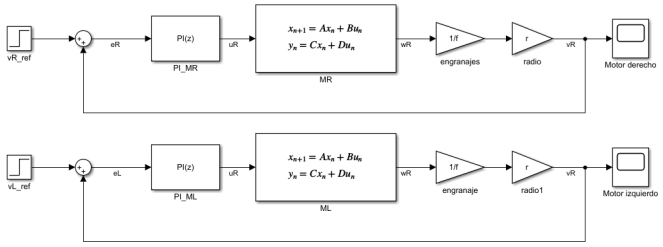

Figura 4.4: Esquema del modelo de referencia

La forma más usual de elegir el modelo de referencia suele ser mediante una función de transferencia con una respuesta deseada o un modelo simplificado sobre el modelo original y debiendo siempre elegir una dinámica que sea alcanzable por el modelo dinámico. Sin embargo, en este trabajo se opta tomar por usar modelos lineales de motores de corriente continua por los siguientes motivos:

1. Proximidad de la dinámica de los motores del modelo de referencia con los del sistema dinámico del AGV. Dado que en el apartado 2.2.4 se explica como las velocidades angulares de las ruedas se fuerzan a adquirir una velocidad angular que es función de la velocidad del cuerpo del AGV (simplificación de los momentos de inercia de las ruedas), la dinámica de las velocidades de rueda del AGV ya no corresponden a la de un sistema lineal. Modelando el sistema de referencia con modelos lineales de los motores de continua, ambas dinámicas estarán relativamente cerca.

2. Facilitar la generación de datos de entrenamiento y pre-entrenamiento de la red del controlador neuronal. Con este modelo de referencia se pueden obtener tantos datos de velocidades de referencia de entrada, velocidades de salida y acciones de control que generaron esas velocidades de salida. Esto es importante para realizar un pre-ajuste de los pesos del controlador con una red de Matlab ya que dispone de distintos métodos de entrenamiento que son más rápidos que el implementado en el controlador. Este ajuste, aunque lejos de ser optimo, resulta muy útil para evitar acciones de control desproporcionadas que saquen al AGV de la trayectoria forzando así a reiniciar la simulación en numerosas ocasiones.

Ajustando las ganancias de los PI a un valor de $K_p = 48{,}956558$ y $K_i = 45{,}230363$ y simulando el sistema:

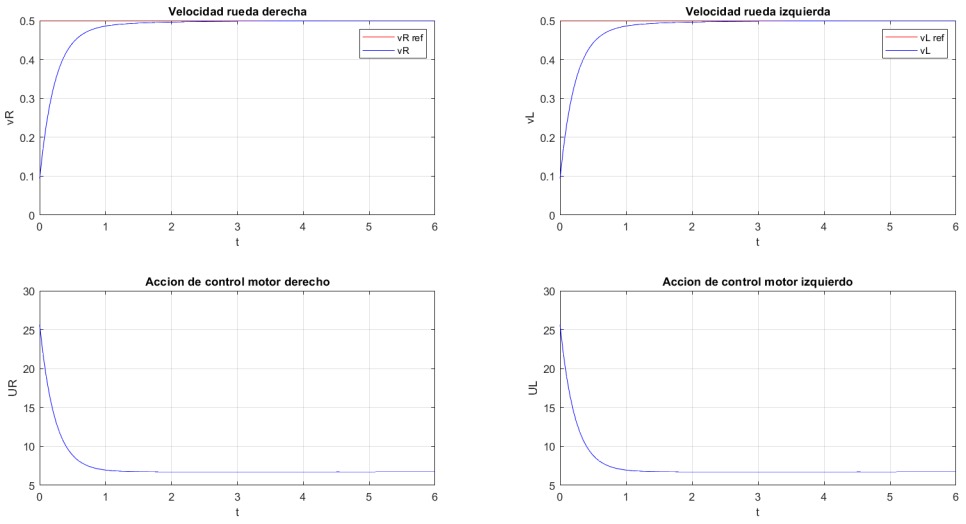

Figura 4.5: Esquema del modelo de referencia

se observa en la figura 4.5 como se consigue el tipo de respuesta críticamente amortiguada con un tiempo de establecimiento aproximado de 2s. Respecto a las acciones de control, únicamente se puede comentar que sus valores de tensión están en un rango adecuado pudiéndose asemejar a un motor comercial. De todas maneras, hay que destacar que este tiempo de establecimiento no podrá ser observado en la respuesta del AGV durante su funcionamiento normal ya que, al producirse continuas variaciones del error durante el seguimiento de una trayectoria, las consignas de velocidad están variando puesto que dado el tiempo de muestreo de 100 milisegundos del controlador de seguimiento. Como se desconocen especificaciones técnicas de AGV en cuanto a sus tiempos de establecimiento para las velocidades, la respuesta mostrada en la figura 4.5 se considera aceptable al no producirse oscilaciones durante el transitorio y tener un tiempo de asentamiento suficientemente largo para evitar arranques bruscos dada la baja velocidad a la que este robot se desplaza.

4.2.3. Diseño del controlador neuronal

El controlador que puede ser lineal o no lineal, está formado por un conjunto de parámetros ajustables y su estructura debería de ser elegida de manera que se pueda lograr una convergencia con el modelo de referencia en condiciones iniciales. Esto significa que si los parámetros del sistema dinámico son conocidos, entonces deberían haber parámetros del controlador que lleven a la salida del sistema a ser igual que la del modelo de referencia. Generalmente, los parámetros del sistema son desconocidos y es el método de adaptación del control neuronal el que reajusta sus parámetros para conseguir la convergencia del sistema dinámico con el modelo de referencia.

Tras varios intentos prueba-error en la configuración de la red del controlador, la siguiente estructura es la que se elige:

- **Número de entradas**: 38
- **Número de salidas**: 2
- **Número de capas**: 2
- **Número de neuronas en la capa oculta**: 48
- **Número de neuronas en la capa de salida**: 2
- **Método para actualización de pesos**: descenso por el Gradiente con Momento
- **Función de activación en capa oculta**: tangente sigmoidea
- **Función de activación en capa de salida**: lineal

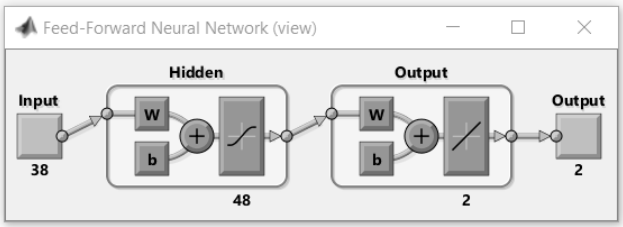

Figura 4.6: Estructura de la red del controlador (imagen representativa)

donde sus entradas están compuestas por los siguientes vectores de entrada:

$$
\begin{aligned}
V_{R\text{ref}} &= [v_{R_{\text{ref}}}(k), v_{R_{\text{ref}}}(k-1), \ldots, v_{R_{\text{ref}}}(k-6)\ ] \\
V_{L\text{ref}} &= [v_{L_{\text{ref}}}(k), v_{L_{\text{ref}}}(k-1), \ldots, v_{L_{\text{ref}}}(k-6)\ ] \\
U_R &= [u_R(k-1), \ldots,\ u_R(k-5)] \\
U_L &= [u_L(k-1), \ldots, u_L(k-5)] \\
V_R &= [v_R(k-1), \ldots, v_R(k-7)] \\
V_L &= [v_L(k-1), \ldots, v_L(k-7)]
\end{aligned}
$$

Para el diseño de la red neuronal, el procedimiento que se sigue es muy similar al usado en la red del modelo:

1. **Propagación de las entradas hacia adelante por la red neuronal**:

$$u = f_2\left(W_2{}^T\ f_1\left(W_1{}^T\ x + B_1\right) + B_2\right) \tag{4.20}$$

donde W_1, W_2,B_1 y B_2 son las matrices de pesos y biases de de ambas capas respectivamente.

$$W_n = \begin{bmatrix} w_{11}^n & w_{12}^n & . & . & w_{1j}^n \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ w_{i1}^n & w_{\text{i2}}^n & . & . & w_{\text{ij}}^n \end{bmatrix} \tag{4.21}$$

$$B_n = \begin{bmatrix} b_1 \\ . \\ . \\ b_i \end{bmatrix} \tag{4.22}$$

siendo n en número de capa (1-2), i el número de entrada a la capa n y j el número de neurona de la capa n. En lo que respecta a las funciones de activación f_1 y f_2, con a como el nivel de activación de entrada a cada neurona:

2. **Actualización de pesos**: asumiendo una buena aproximación del modelo neuronal:

$$y_p \approx y_m \tag{4.23}$$

usando la función de coste cuyas salidas representan las salidas de la iteración anterior:

$$E_c = \frac{1}{2}\sum_{i=1}^{n}\left(y_{ref_i} - y_{p_i}\right)^2 \tag{4.24}$$

calculamos el error de control y propagamos este hacia atrás por la red neuronal del modelo usando las ecuaciones 4.9-4.14. Una vez hecho esto, obtenemos las contribuciones de las acciones de control al error según:

$$\frac{\partial}{\partial u_i}E_c = \sum_{j=1}^{m_1} S_j^1 w_{\text{ij}}^1 \tag{4.25}$$

donde S_j^1 es la sensibilidad de la neurona j en la capa oculta del modelo neuronal y w_{ij}^1 los pesos de esta misma capa. Calculando las sensibilidades de la capa de salida del controlador neuronal como:

$$S_i^2 = \dot{f}_i^2 \frac{\partial}{\partial u_i} E_c \tag{4.26}$$

y usando las ecuaciones 4.10-4.19 pero aplicadas al error de control E_c y para las sensibilidades, pesos y biases del controlador neuronal, se realiza la actualización de los parámetros de la red.

Para el diseño de la red neuronal, la señal de error es la diferencia error entre la salida del modelo de referencia y_{ref} y la salida del sistema dinámico y_p.

4.3. Entrenamiento de las redes del controlador

4.3.1. Modelo neuronal

Un correcto entrenamiento de la red del modelo es crucial para el funcionamiento del controlador al completo, lo que implica poder disponer de un conjunto de datos de entrenamiento lo más amplio posible, pero esto no es tarea fácil en este trabajo ya que existen posiciones comprometedoras del AGV en las que se generan datos de simulación erróneos. Por ello, en lo que a la generación de datos de alta frecuencia respecta, es necesario que el AGV mantenga unos cambios de rueda adecuados y dentro de unos rangos. La solución para esto es limitar los movimientos del AGV a los generados por el seguimiento de una trayectoria.

Para el ajuste de esta red se generan datos de referencia basados en escalones de tensión de misma amplitud a diferentes frecuencias durante un periodo de tiempo y muestras del comportamiento del AGV con el control PID-PI durante el seguimiento de una trayectoria a lo largo de un periodo de tiempo determinado. Estas dos distintas formas de obtener datos de entrenamiento es debido a que, si se aplicamos distintas amplitudes de tensión durante intervalos de tiempo variable, siendo alta la diferencia entre tales amplitudes, el robot trataría de adquirir posiciones problemáticas dando el modelo dinámico datos de entrenamiento no válidos. Con estas dos formas de crear los datos de entrenamiento nos aseguramos de entrenar la red en un rango de tensiones para entradas escalón, tanto de alta como de baja frecuencia, y también de obtener muestras suficientes en alta frecuencia.

Tras el procedimiento anterior se consigue recoger un total de 23697 muestras de entrenamiento. Dichos datos de entrenamiento son permutados antes de introducirlos la red neuronal con el objetivo de eliminar la correlación existente entre muestras sucesivas.

Para un primer pre-ajuste de los pesos, se opta por hacer uso de una red neuronal ya implementada en Matlab dado que esta presenta más opciones de métodos de entrenamiento como condiciones de parada, tratando así de obtener unos valores de parámetros ajustados en el menor tiempo posible. Durante el entrenamiento de la red se busca minimizar el error cuadrático medio (MSE) usando método de actualización de Levenberg Marquardt y se consigue obtener un valor aproximado de $4e^{-5}$

Tras este primer ajuste, esta pasa a un segundo ajuste *on-line* durante otro periodo de tiempo para conseguir un ajuste más fino ya que el funcionamiento del controlador depende fuertemente de la precisión del modelo aproximado.

4.3.2. Controlador neuronal

El procedimiento de entrenamiento, al igual que para el modelo neuronal, se basa en dos pasos: pre-ajuste de los pesos *off-line* y un posterior ajuste *on-line* para completar. Para el pre-ajuste *off-line* únicamente se recogen datos de la respuesta del modelo de referencia ante entradas de velocidades de referencia aleatorias de distinta frecuencia. Esto puede parecer no tener mucho sentido ya que el modelo de referencia poco tiene que ver con el dinámico real, pero el contar con un controlador ajustado en cierto grado ayuda bastante en cuanto a rapidez dado que se emplea una tasa de actualización *on-line* de pesos baja con el objetivo de llegar a una buena aproximación y no crear acciones de control desproporcionadas durante el ajuste fino.

La cantidad total de muestras recogidas para el pre-ajuste es de 24752 y al igual que para la red del modelo, también se busca minimizar el error cuadrático medio (MSE) mediante el método Levenberg Marquardt. Los datos de entrenamiento son permutados previamente a este. El valor de MSE conseguido es de $5e^{-8}$. Un ejemplo sobre un corto periodo de tiempo de estos datos de entrenamiento generados se puede ver en fig. 4.7:

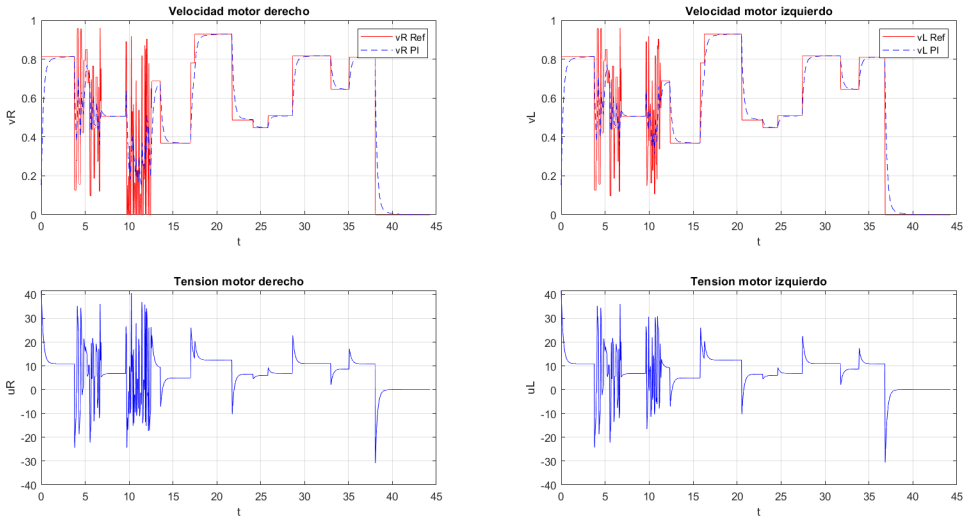

Figura 4.7: Ejemplo datos de pre-entrenamiento

Como se aprecia, en todo momento se mantiene la respuesta deseada para la dinámica de las velocidades del AGV. Las acciones de control en este caso no son relevantes ya que el controlador neuronal calculará los valores necesarios para llevar la salida del modelo a la del modelo de referencia y obviamente serán distintas.

Capítulo 5

Control de seguimiento con aprendizaje por refuerzo

5.1. Diseño de controladores de velocidad PI

En cuanto a lo que respecta al control de motores DC y la naturaleza lineal de sus ecuaciones en espacio de estados (dada la simplificación realizada sobre los momentos de inercia), para su control se opta por emplear un control PI. También se podría haber usado un control PID, pero dado que con solo las acciones proporcional e integral es suficiente para controlar la velocidad de las ruedas, se selecciona el controlador más simple.

La señal del error del controlador PI para cada motor se obtiene de la diferencia entre las velocidades de las ruedas y las velocidades de referencia de estas para cada momento, las cuales vendrán impuestas por el control de seguimiento y serán variadas conforme el robot necesite modificarlas para corregir su trayectoria. La velocidad de crucero v_{Dref} deberá de ser mantenida en cada momento, independientemente de que velocidad adquiera cada una de las ruedas. Las ecuaciones del control que realizan estos cálculos se pueden ver en el apéndice B.

Estos controles de velocidad se aplican al sistema con una frecuencia de 50ms y su sintonización se hace en dos pasos:

1. Se crean los modelos discretizados de las ecuaciones de los motores para el peso nominal del AGV y se sintonizan los parámetros del PI para obtener una respuesta críticamente amortiguada y con tiempo de establecimiento aproximado de 2 segundos fig. 5.1.

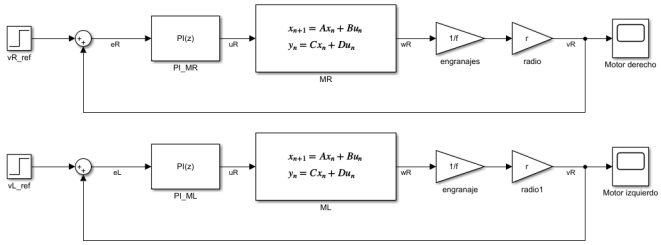

Figura 5.1: Controladores PI

2. Posteriormente se re-ajustan los valores obtenidos en el punto anterior mediante prueba-error usando el modelo dinámico completo, ya que los modelos lineales de los motores usados no describen completamente la dinámica de las ruedas del AGV con las simplificaciones hechas. Ambos motores se consideran iguales, por lo que las ganancias son las mismas para los dos:

$$K_p = 49.956558 \qquad K_i = 28.2303623$$

5.2. Control PID incremental con Q-Learning ΔQPID

Para el diseño del control de seguimiento se opta por usar el algoritmo de aprendizaje por refuerzo Q-Learning dada su popularidad y no compleja implementación. Este controlador se basa en la implementación de un control PID, con ganancias pre-sintonizadas a unos valores y trayectoria nominal, sobre el que el algoritmo de RL actúa modificando las ganancias para trazar distintas trayectorias a la nominal.

Inicialmente se realizaron varias pruebas usando Q-Learning pero estas no fueron satisfactorias debido a que tomando únicamente la señal de distancia del sensor como señal de estado, no se considera que fuera la forma más idónea de aplicar el algoritmo de esta manera. Se requiere de más señales de información para formar el estado, es decir, conocer sus valores pasados e información sobre su tendencia si es posible. Puesto que el control PID utiliza dichas señales y dado que se puede usar una pre-sintonización de estos para partir de una respuesta del sistema valida, se agiliza mucho el proceso de aprendizaje.

El esquema de este controlador se puede ver en la siguiente figura:

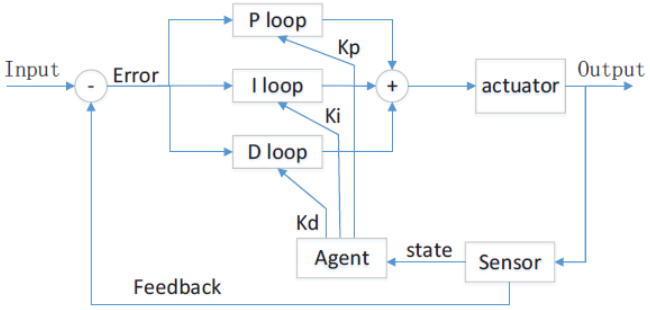

Figura 5.2: Esquema controlador ΔQPID [11]

El diseño de este controlador se resume en los siguientes pasos:

1. **Asignación de estados**: se toma como error de posición la distancia devuelta por el sensor $e(t)$ discretizada. Como hay tres parámetros a ajustar K_p, K_d y K_i, basándonos en la información del apartado 2.2, se realiza la siguiente asignación de estado y sus valores según el parámetro:

   - $S_{K_p} = \frac{\mathrm{d}}{\mathrm{d}t} e\,(t)$ para el parámetro K_p con rango entre -0.3 y 0.3 e incrementos de 0.01.
   - $S_{K_d} = \frac{d^2}{\mathrm{d}t^2} e\,(t)$ para el parámetro K_d con rango entre -0.3 y 0.3 e incrementos de 0.01.
   - $S_{K_i} = e(t)$ para el parámetro K_i con rango entre -0.3 y 0.3 e incrementos de 0.01.

   Con ello obtenemos un total de 61 estados para cada uno de los parámetros.

2. **Asignación de acciones**: Como los parámetros actúan de manera distintas sobre el comportamiento del sistema controlado, se escogen distintos valores de acciones dependiendo del parámetro. Procediendo de la misma manera que el punto anterior:

   - A_{K_p} incrementos sobre el valor actual de parámetro K_p en un rango de -0.02 a 0.02 y con un paso de discretización de 0.001.
   - A_{K_d} incrementos sobre el valor actual de parámetro K_d en un rango de -0.02 a 0.02 y con un paso de discretización de 0.001
   - A_{K_i} incrementos sobre el valor actual de parámetro K_i en un rango de -0.002 a 0.002 y con un paso de discretización de 0.0001

Tras esta asignación, se obtiene una cantidad total de 41 acciones distintas para cada uno de los estados.

3. **Creación de tablas** $Q(s,a)$: con los estados y acciones ya asignadas, se crean las tablas de valores Q. En ellas, las filas representan cada uno de los posibles estados discretos para cada uno de los parámetros del PID y las columnas las posibles acciones que el agente puede tomar en cada uno de los estados. El tamaño de estas matrices es de 61x41 y quedan de la siguiente manera.

$$Q_{K_p}(s,a) = \begin{bmatrix} q^{k_p}_{1.1} & \cdot & \cdot & \cdot & \cdot & q^{k_p}_{1.41} \\ \cdot & \cdot & & & & \cdot \\ \cdot & & & & \cdot & \cdot \\ q^{k_p}_{61.1} & \cdot & \cdot & \cdot & \cdot & q^{k_p}_{61.41} \end{bmatrix} \tag{5.1}$$

$$Q_{K_d}(s,a) = \begin{bmatrix} q^{k_d}_{1.1} & \cdot & \cdot & \cdot & \cdot & q^{k_d}_{1.41} \\ \cdot & \cdot & & & & \cdot \\ \cdot & & & & \cdot & \cdot \\ q^{k_d}_{61.1} & \cdot & \cdot & \cdot & \cdot & q^{k_d}_{61.41} \end{bmatrix} \tag{5.2}$$

$$Q_{K_i}(s,a) = \begin{bmatrix} q^{k_i}_{1.1} & \cdot & \cdot & \cdot & \cdot & q^{k_i}_{1.41} \\ \cdot & \cdot & & & & \cdot \\ \cdot & & & & \cdot & \cdot \\ q^{k_i}_{61.1} & \cdot & \cdot & \cdot & \cdot & q^{k_i}_{61.41} \end{bmatrix} \tag{5.3}$$

4. **Inicialización de valores** q: esta inicialización se hace según el método descrito en [8] para impulsar la exploración inicial del espacio por parte del agente mediante la elección de unos valores suficientemente altos para que, independientemente de las acciones iniciales seleccionadas, la recompensa sea inferior a este valor. Dado que en este caso se utiliza una función de recompensa que en lugar de recompensar, sólo penaliza los errores con valores negativos, para impulsar esta exploración los valores iniciales de las matrices se fijan a cero. En este caso, esto equivale a inicializar las matrices con un valor superior a cero tal y como se describe en [8].

5. **Balance entre exploración/explotación** q: para el caso del $\Delta QPID$, se implementa una estrategia ε-*greedy* en la que se elige un valor de $\varepsilon = 1$ inicial y se va decrementando con el tiempo acorde a la siguiente ecuación:

$$\varepsilon = \varepsilon_{\text{mín}} + (\varepsilon_{\text{máx}} - \varepsilon_{\text{mín}})\, e^{-\varepsilon_{\text{decay}}\, k} \tag{5.4}$$

donde $\varepsilon_{\text{mín}}$ es la cantidad de exploración mínima que queremos usar de manera permanente (probabilidad), $\varepsilon_{\text{máx}}$ es el valor máximo de exploración, $\varepsilon_{\text{decay}}$ es la tasa de decremento de ε y k es el número de paso efectuado por el agente (o instante de tiempo).

Por otro lado, puede parecer poco eficiente inicializar los valores de las matrices Q de la manera explicada en el punto anterior y seleccionar también una estrategia ε-*greedy* dado que con ambas se consigue una exploración inicial por parte del agente. La razón de esto es que el AGV está limitado a ser probado siguiendo una trayectoria concreta, por lo que no es posible explorar todo el espacio. Cuando se inicializa el entrenamiento del algoritmo RL con estrategia ε-*greedy*, el valor de ε decae hasta un valor mínimo con el tiempo. Si se ha obtenido una estrategia óptima y se cambia de trayectoria, la tasa de exploración sobre la nueva trayectoria es muy baja. Inicializando los valores según el punto anterior, los estados que no han sido explorados durante el entrenamiento quedan fijados con un valor alto de manera que de darse estos, se lleva a cabo un cierto grado de exploración incluso si el valor de ε es muy bajo. Esto se puede ver más claramente en fig. 5.3, fig. 5.4 y fig. 5.5 que corresponden con las tablas Q para los parámetros K_p,K_d y K_i:

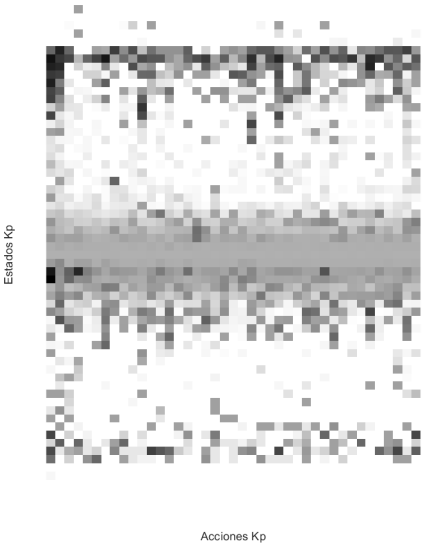

Figura 5.3: Q-tabla para K_p

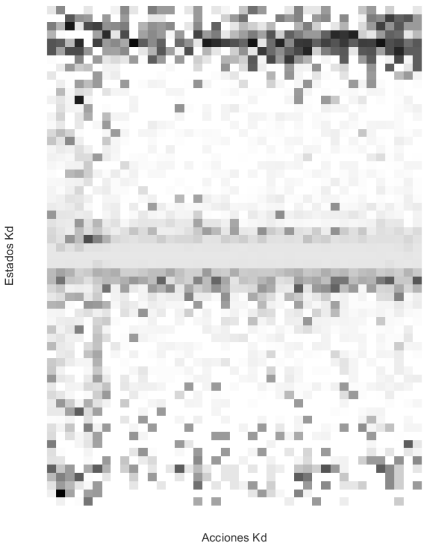

Figura 5.4: Q-tabla para K_d

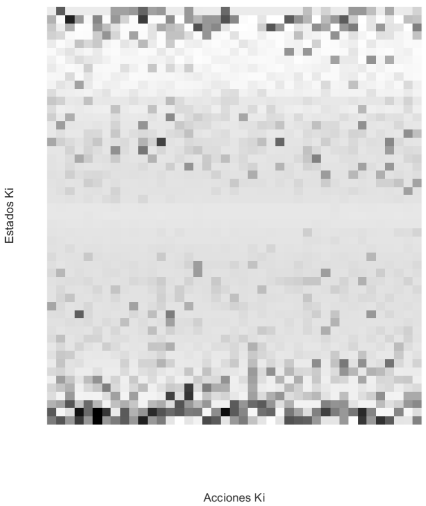

Figura 5.5: Q-tabla para K_i

en estas figuras observa que los valores $q(s, a)$ adquiridos de las matrices tras un largo entrenamiento. En cada una de ellas se observa las regiones de estados en las que se han recibido más muestras y corresponden a las áreas que presentan un contraste más suave. Las áreas blancas corresponden los pares $q(s, a)$ que no han sido explorados durante el entrenamiento, quedando con su valor inicial y por último, las regiones más oscuras son aquellos pares estado-acción que han recibido una recompensa bastante menor que el resto, ya sea por su cantidad de error o porque han generado la salida el robot de su trayectoria.

Debido a la función usada para el cálculo de la recompensa, que devuelve valores negativos en función del error como se verá más adelante, en todas las figuras el color blanco indica el valor q asignado durante la inicialización (es decir el más alto). El algoritmo fue entrenado durante 600 episodios con duración variable y una estrategia inicial ε-*greedy* con decaimiento de ε. A pesar de la larga duración del entrenamiento siguen quedando regiones en blanco que señalan que tales parejas de $q(s, a)$ no han sido exploradas aún. Cuando se observaron estas imágenes por primera vez, las regiones en blanco aún eran más amplias y tras cambiar a otras trayectorias se fueron reduciendo. Por este motivo se opto por realizar la inicialización con valores q altos y una estrategia ε-*greedy* al mismo tiempo.

6. **Elección de la función de recompensa**: esta es la siguiente:

$$\text{Reward} = -\left[C_1\, e\,(k)^2 + C_2 \Delta w_D^2\,(k) + C_3 \dot{e}(k)^2\right] \tag{5.5}$$

Si el AGV se sale de la trayectoria, la recompensa es de -10 y cuando $e(k) = 0$ la recompensa es de 10. Con esta ecuación se pretende penalizar al agente ante la existencia de error de posición, incrementos elevados de la acción de control e incrementos altos de error.

7. **Condición de parada del entrenamiento**: Como condición de parada se establece la obtención de la tabla optima $Q_K^*(s, a)$. Esta condición se establece para cada una de las tablas de los parámetros. La actualización de ellas no se realiza si se cumple:

$$\begin{aligned} \Delta Q_{K_p} &= \left\| Q_{K_p}\,(k) - Q_{K_p}\,(k-1) \right\| \leq \text{Umbral} \\ \Delta Q_{K_d} &= \left\| Q_{K_d}\,(k) - Q_{K_d}\,(k-1) \right\| \leq \text{Umbral} \\ \Delta Q_{K_i} &= \left\| Q_{K_i}\,(k) - Q_{K_i}\,(k-1) \right\| \leq \text{Umbral} \end{aligned} \tag{5.6}$$

el algoritmo cancela el entrenamiento si se cumplen las tres condiciones.

5.3. Entrenamiento del controlador ΔQPID

El controlador es incializado sólo una vez con los valores de K_p, K_d y K_i dados en 4.1.1 y se le hace recorrer una trayectoria que es aleatoria entre cuatro tipos distintos. Estando los motores del AGV controlados por los PI con ganancias obtenidas en , el AGV empieza a recorrer la trayectoria durante un periodo de tiempo máximo de 50s. Un nuevo episodio comienza si el robot se sale de la trayectoria o consigue mantenerse en ella durante el tiempo máximo establecido. En cada reinicio de episodio, el AGV es llevado nuevamente al punto inicial de la trayectoria correspondiente en ese momento pero con una pequeña variación de la coordenada del eje y para tratar de introducir más variabilidad, sobre todo al inicio de cada episodio ya que el AGV parte de una velocidad inicial nula. La configuración de los parámetros de la estrategia *ε-greedy* es:

- ε_{min} =0.02
- ε_{max}=1
- $\varepsilon = 1$
- ε_{decay}=0.001

En lo que respecta a la actualización de los valores $q(s, a)$, viene dada en cada paso que el agente realiza, es decir, *on-line*. Para un cambio de estado y una recompensa recibida durante el recorrido de una trayectoria, el valor q correspondiente se actualiza estando así más cerca de su valor optimo q^*. Aparte de este método elegido, existen otros métodos de actualización que aportan resultados distintos como por ejemplo guardar todas las recompensas y estados y al final del episodio se actualizar cada tabla.

Tras un largo entrenamiento, el controlador va ajustando los parámetros del PID de manera correcta presentando un buen comportamiento a lo largo de las trayectorias, pero no se llega a conseguir obtener una parada del algoritmo de entrenamiento por obtención de los valores óptimos de las tres matrices $Q^*_{K_p}$, $Q^*_{K_d}$ y $Q^*_{K_i}$. Se requiere mucho más tiempo de simulación ya que el controlador realiza un paso cada 100ms (tiempo de muestreo para el control de seguimiento). Teniendo en cuenta que el software debe realiza cálculos para el sistema dinámico entre otros, no resulta factible el tratar de entrenar el algoritmo de RL de manera indefinida hasta alcanzar los valores óptimos. En su defecto, se observa el comportamiento del control sobre el AGV y cuando este se considere aceptable, se para el entrenamiento.

Capítulo 6

Resultados

Se realizan distintas combinaciones sobre los controladores diseñados y se someten a prueba mediante el seguimiento de una trayectoria para poder comprobar su funcionamiento y analizar los resultados. Dicha trayectoria es la siguiente:

$$tr = 40 + 1{,}5sin(0{,}3x) \tag{6.1}$$

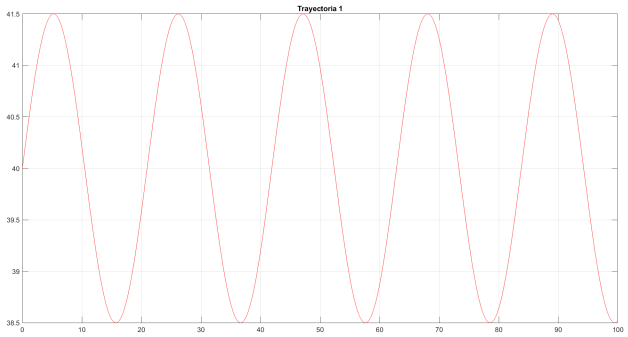

Figura 6.1: Trayectoria nominal

Es una trayectoria senoidal en donde el *off-set* de 40 es fijado únicamente para mantener al robot en el primer cuadrante. Todos los controles se ajustaron durante su proceso de diseño siguiendo dicha trayectoria.

Se prueba inicialmente el control PID para seguimiento de la trayectoria y PI para las ruedas *PID-PI*, posteriormente el control PID para seguimiento junto con el control neuronal para las velocidades *PID-NNC*, a continuación el control con PID incremental mediante Q-Learning para seguimiento y controladores PI para las velocidades $\Delta QPID\text{-}PI$ y por último la combinación de los controles de aprendizaje por refuerzo y neuronal $\Delta QPID\text{-}NNC$. Todos los experimentos de simulación se realizan durante un periodo de 100s, el cual se considera suficiente para analizar los resultados. La velocidad de crucero es fijada en todos los casos para $0.5m/s$.

Con el objetivo de comparar el comportamiento del AGV con los distintos controladores, se toma como métrica el error cuadrático medio (MSE) para los errores de seguimiento y de velocidad de crucero.

6.1. Control PID-PI

6.1.1. Trayectoria senoidal

- **Trayectoria** En fig. 6.2 se puede ver como el control clásico PID-PI con sintonización mediante AG sigue de manera bastante precisa la trayectoria al mismo tiempo que el centro de la unidad de tracción queda aceptablemente alineado con el centro del sensor magnético. Si se necesitara mejorar dicha alineación se podría conseguir mediante un reajuste de las ganancias del control PID. Para ello, en la función de coste del AG un aumento del peso para error del angulo podría conseguir tal objetivo. Por otro lado, en lo que respecta la alineación del cuerpo del AGV con la trayectoria seguida, al final de la simulación se ve que esta alineación no es correcta. Esto es debido a que no se contempló inicialmente esta alineación siendo sólo objetivo la de la unidad de tracción. Aunque ello no corresponda aquí, para mejorar dicha alineación habría que añadir en las funciones de coste y/o recompensa otra variable que relacionara el punto de trayectoria en cada instante con la posición del eje trasero del cuerpo del AGV. Esto último seria en caso de no poder cambiar la ubicación del sensor magnético, ya que de poder hacerlo, la solución más rápida y sencilla pasaría por ubicar dicho sensor en la parte frontal del cuerpo del AGV. Pruebas iniciales sobre la cinemática hechas para este trabajo así lo corroboran a pesar de que no se muestren por no objeto del trabajo.

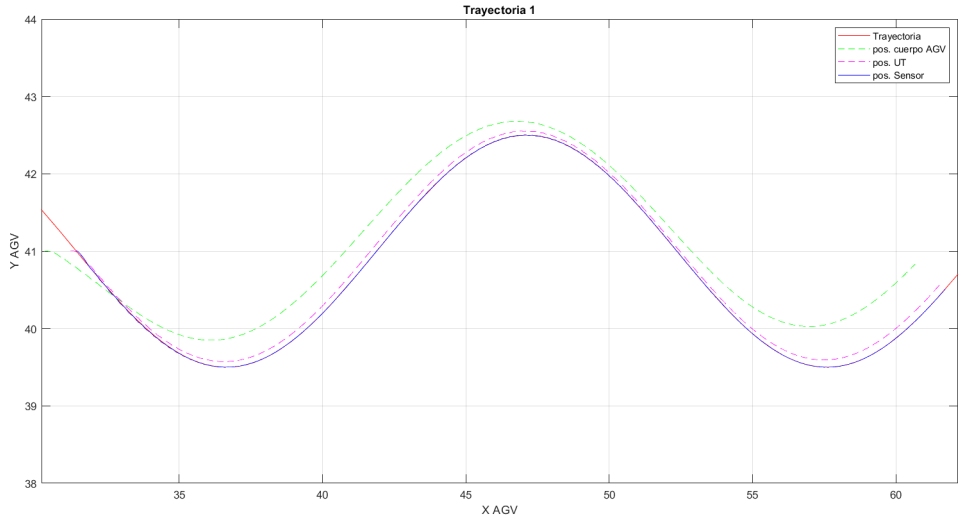

Figura 6.2: PID-PI: Seguimiento de trayectoria

- **Error de seguimiento** A pesar de que la trayectoria parece ser seguida casi de manera ideal, analizando la gráfica fig. 6.3 del error de seguimiento vemos como se producen ligeras oscilaciones alrededor del centro del sensor. Estas oscilaciones son del orden de 1 o 2 cm y van disminuyendo en el tiempo. La causa de esto es la corrección del control PID cuando el AGV se desvía ligeramente de la trayectoria. A pesar de ser pequeñas, podrían ser mejoradas mediante un ajuste más suave tanto para la respuesta del PID como de los controles de velocidad PI.

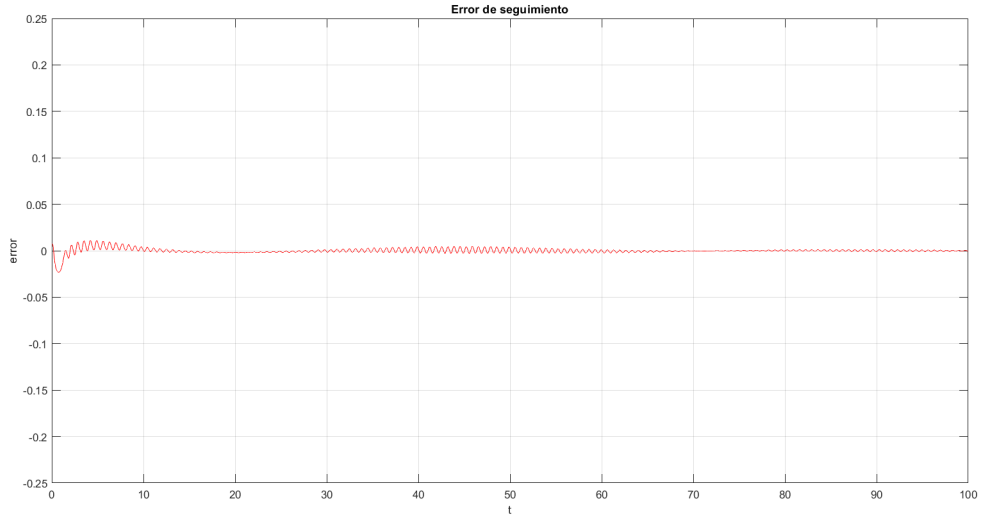

Figura 6.3: PID-PI: Error de seguimiento

- **Velocidad crucero y error**: En la figura 6.4 se comprueba como el objetivo de velocidad de crucero de $0.5m/s$ se cumple satisfactoriamente. El robot inicialmente parte de con velocidad nula y la combinación de los controles hacen que esta quede fija durante el recorrido del AVG independientemente de las pequeñas desviaciones sobre la trayectoria. En la gráfica del error se comprueba que rápidamente converge a un error prácticamente nulo y manteniendo a este durante todo el recorrido.

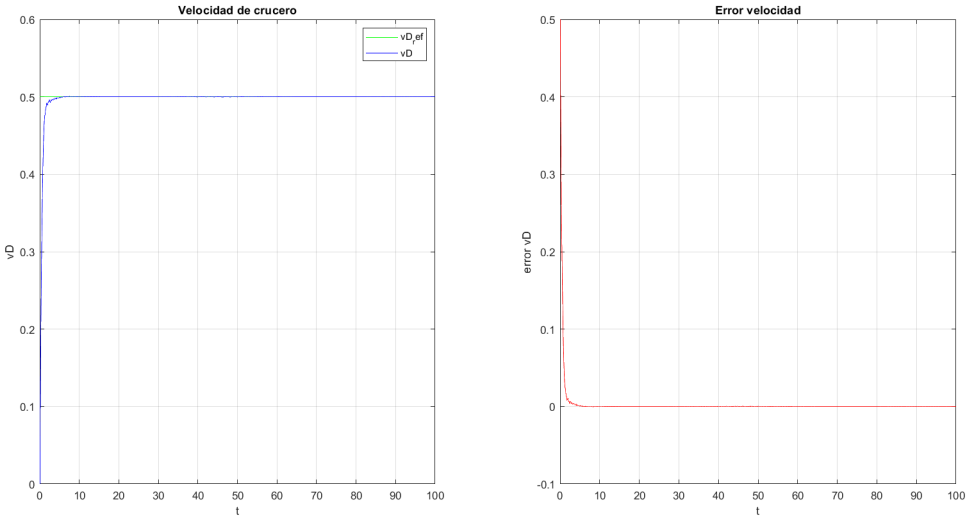

Figura 6.4: PID-PI: Velocidad de crucero y su error

- **Errores de velocidad**: Cuando el control PID hace una corrección de trayectoria, este impone unas nuevas velocidades de referencia para cada rueda y que los controles PI deben conseguir. Estas velocidades de referencia varían cada 100 milisegundos, lo cual es imposible de alcanzar para los motores dada la dinámica más lenta de estos. Los errores mostrados en las gráficas fig. 6.5 muestran esta diferencia entre la velocidad de referencia marcada por el PID y la que se consigue obtener por el control PI para ese corto intervalo de tiempo.

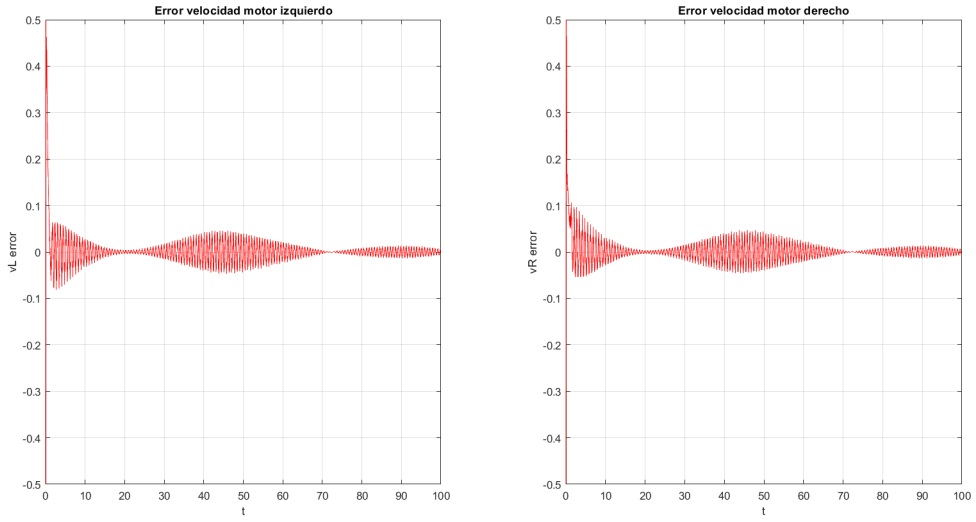

Figura 6.5: PID-PI: Error de velocidad en ruedas

- **Gráficas motores**:

1. *Velocidades angulares y par motor*: Analizando también la respuesta de los motores para comprobar que los valores se encuentran entre valores válidos se comprueba que tanto las velocidades angulares de las ruedas como los pares que provocan dichas velocidades se encuentran en rangos de valor adecuado. Dado que el robot tiene velocidad inicial cero, se produce un par motor elevado, lo es una respuesta típica de estos motores fig. 6.6.

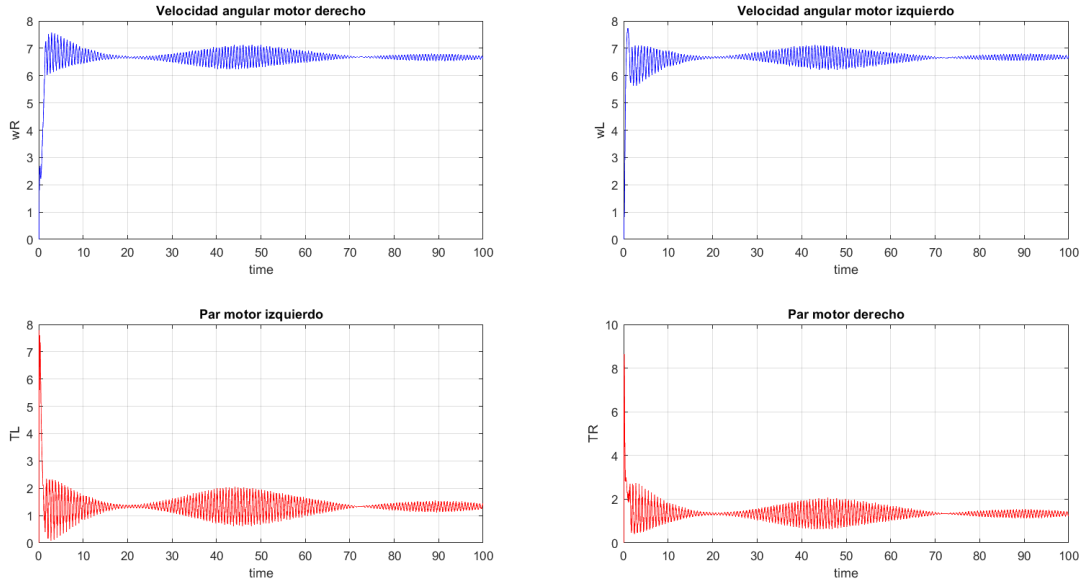

Figura 6.6: PID-PI: Velocidades angulares y par motor

2. *Intensidades motor*: Siguiendo con el análisis de los motores, y que el par tiene una relación proporcional a la corriente, en la siguiente imagen se puede apreciar tal relación si se compara con las anteriores. La corriente de arranque es elevada al inicio y posteriormente fluctúa debido a las correcciones de las velocidades fig. 6.7 .

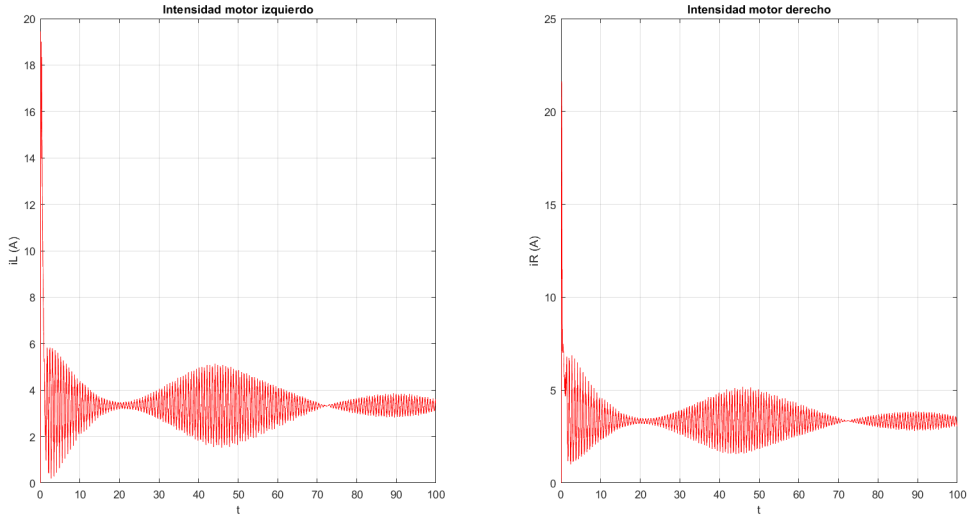

Figura 6.7: PID-PI: Intensidades motores

6.2. Control PID-NNC

6.2.1. Trayectoria senoidal

- **Trayectoria** Simulando para un periodo de 100 milisegundos el control de seguimiento PID (sin modificación de sus parámetros) pero el controlador neuronal con modelo de referencia, la siguiente gráfica muestra el trazado de la trayectoria nominal. En ella se pueden observar que se producen oscilaciones notables (aunque no muy amplias) durante el recorrido. Aun así, el robot consigue permanecer satisfactoriamente en la trayectoria y completar el recorrido sin salirse fig. 6.8.

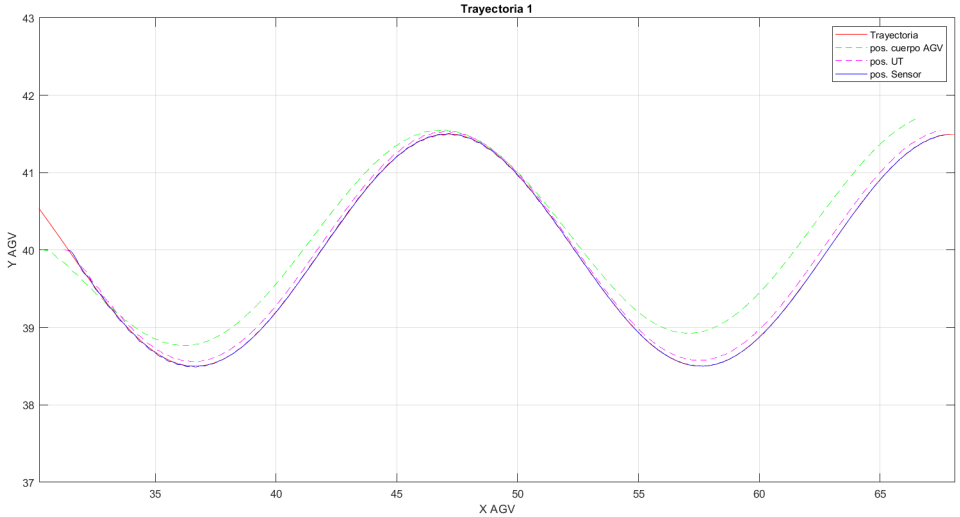

Figura 6.8: PID-NNC:Seguimiento de trayectoria

- **Error de seguimiento** En la figura 6.9 sobre el error de la trayectoria, las oscilaciones observadas en la imagen anterior, se aprecian de una manera más notable que en el apartado anterior con el control PI. A pesar de su mayor amplitud, son del orden de 2 a 3 centímetros.

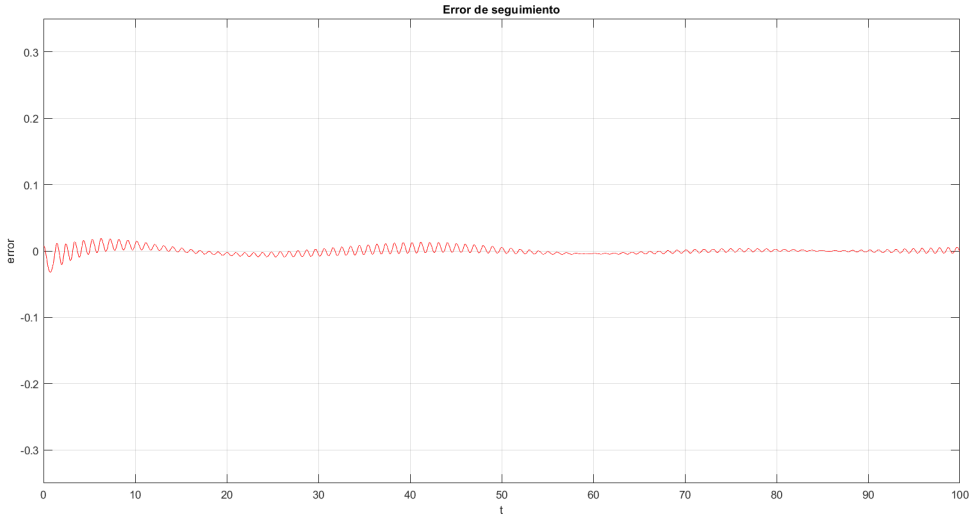

Figura 6.9: PID-NNC:Error de seguimiento

- **Velocidad crucero y error**: Observando la gráfica fig. 6.10 vemos que se obtiene un resultado inesperado y es que la velocidad no consigue mantenerse en los $0.5m/s$ deseados. Existen oscilaciones notables aunque de baja amplitud que provocan un error no uniforme y con tendencia a cero. A pesar de extraño, es comprensible debido a que el controlador PID mantiene los valores de los parámetros ajustados para sistema controlado por controles PI, pero dado que con el controlador neuronal con modelo de referencia estamos forzando a que la respuesta del AGV pase de ser la de un sistema multivariable acoplado a un sistema multivariable desacoplado, se requiere un reajuste de las ganancias del PID con el control neuronal.

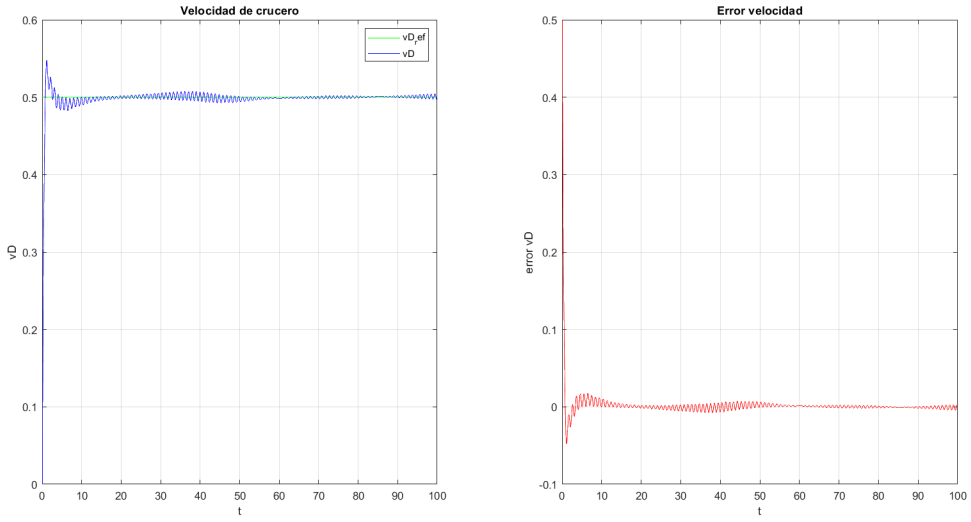

Figura 6.10: PID-NNC:Velocidad de crucero y su error

- **Errores de velocidad**: Respecto a las velocidades de las ruedas, se observa en fig. 6.12 una mayor amplitud en las oscilación mostradas en el control PID-PI, cuyo motivo viene dado por lo comentado en la imagen anterior. Aparte de esto, el comportamiento de estas velocidades es normal.

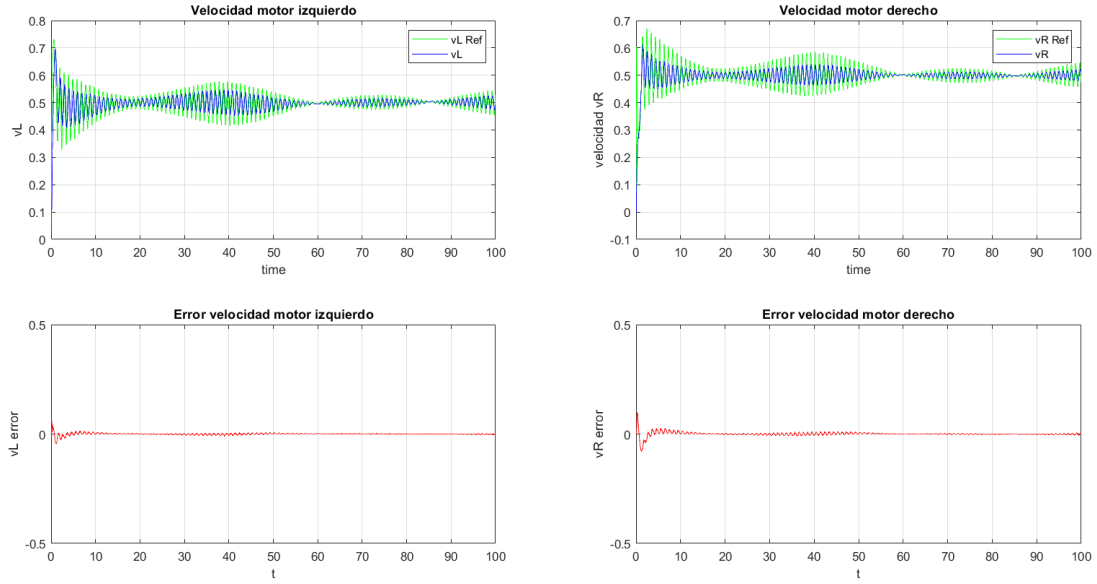

Figura 6.11: PID-NNC:Error de velocidad en ruedas

- **Aproximación al modelo de referencia**: La figura 6.12 muestra el correcto funcionamiento del controlador neuronal con modelo de referencia ya que podemos observar claramente (tras el entrenamiento de las redes) como la red neuronal para el modelo (azul) aproxima muy bien el sistema dinámico (azul). El controlador neuronal consigue llevar al sistema dinámico a seguir la dinámica del modelo de referencia (rojo), el cual tiene una respuesta lineal y desacoplada.

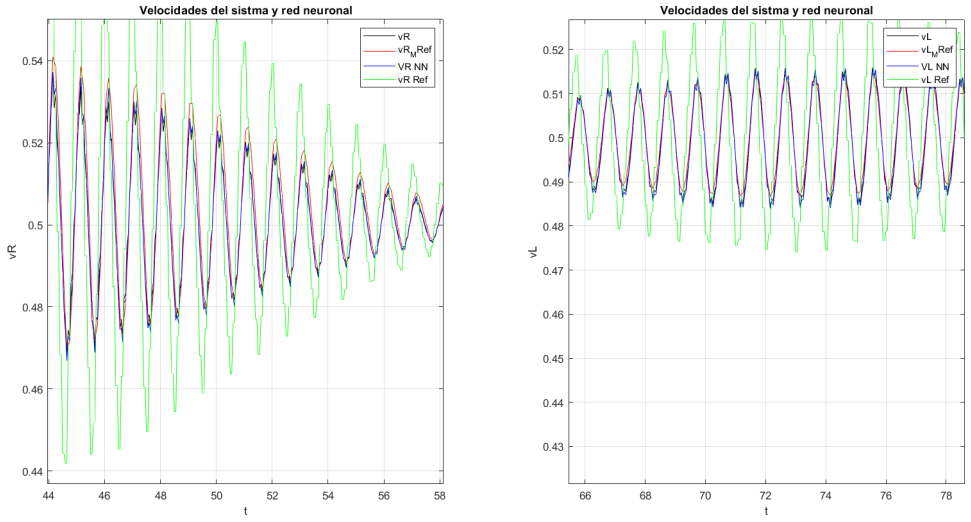

Figura 6.12: PID-NNC:Aproximación a Modelo de Referencia

- **Gráficas motores**:

1. *PID-NNC:Velocidades angulares y par motor*: Respecto al comportamiento de las velocidades angulares y pares motor, se observa en fig. 6.13 un comportamiento normal por parte de los motores. El cambio de respuesta del AGV respecto al apartado con control PID-PI se refleja también en las velocidades angulares de los motores y sus respectivos pares, dando como resultado una mayor amplitud de estas. La siguiente figura muestra tal observación:

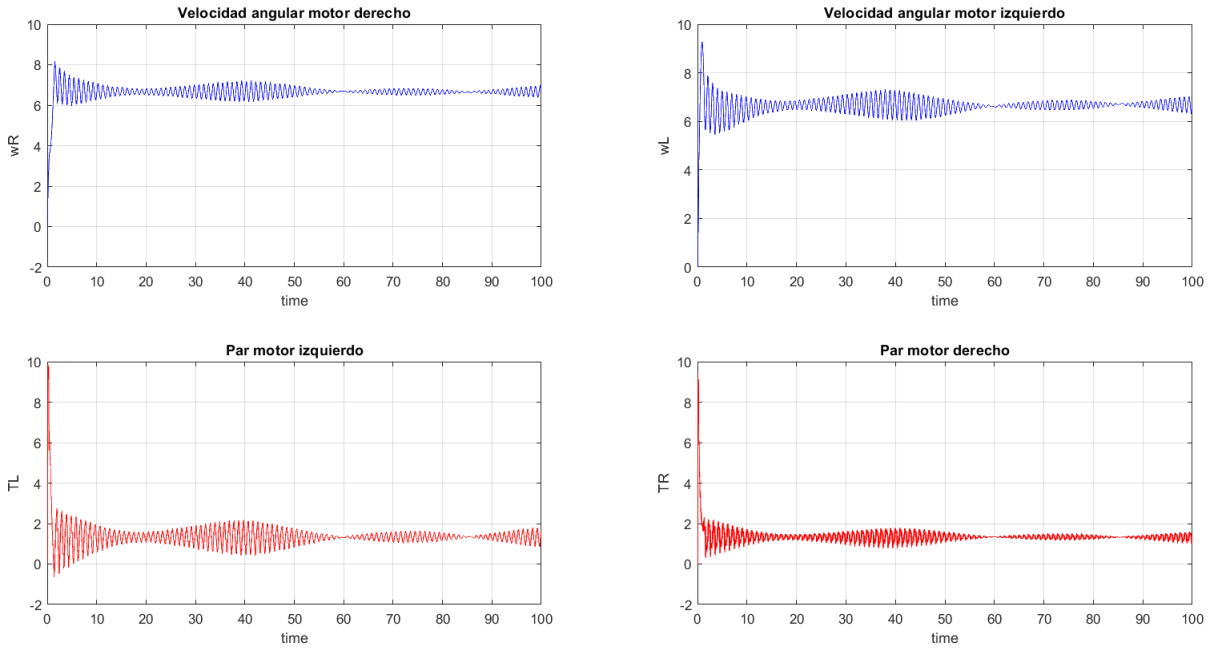

Figura 6.13: PID-NNC:Velocidades angulares y par motor

2. *Corrientes de los motores*: Consecuentemente con lo comentado en la figura anterior, la amplitud y frecuencia de las corrientes es aumentada fig. 6.15:

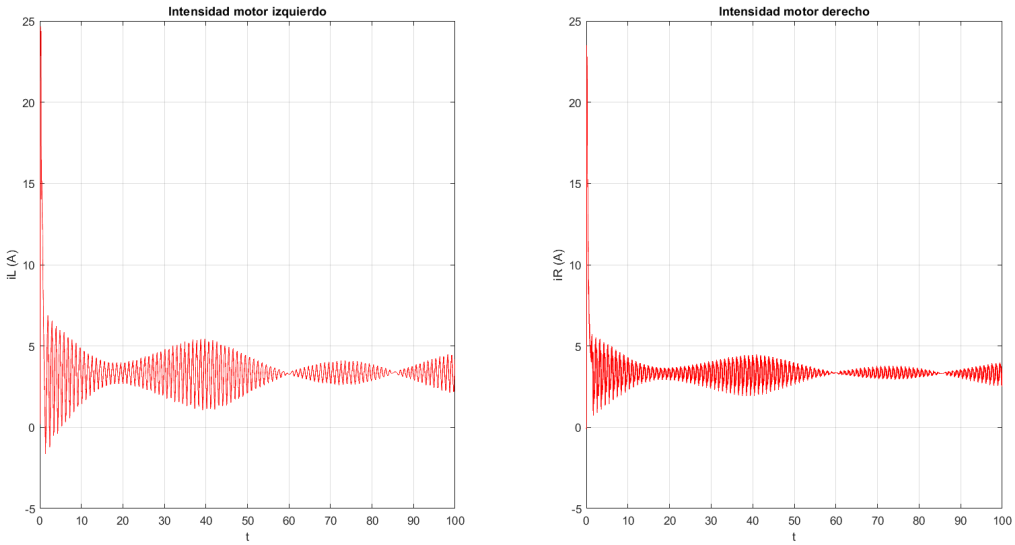

Figura 6.14: PID-NNC:Intensidades motores

3. *Voltaje motores*: El cambio de respuesta observado nos lleva a observar que las tensiones que son aplicadas a los motores en la figura 6.15. En esta se ve los cambios constantes que realiza el control de velocidad pero comprobamos que se encuentran en un rango de amplitud normal y aceptable.

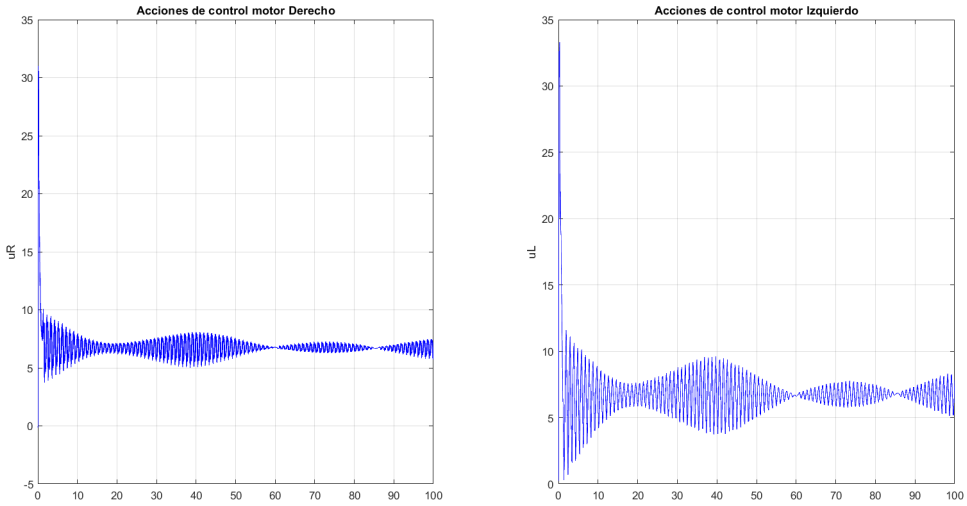

Figura 6.15: PID-NNC:Tensiones motores

6.3. Control ΔQPID-PI

6.3.1. Trayectoria senoidal

- **Trayectoria** Comprobando que el controlador basado en Aprendizaje por Refuerzo funciona adecuadamente, re-ajustando así los parámetros del control PID, levemente se puede observar que existen ciertas oscilaciones durante el seguimiento de la trayectoria. Sin embargo el control acaba el recorrido completo sin que el AGV se salga de la trayectoria. La figura 6.16 así lo muestra:

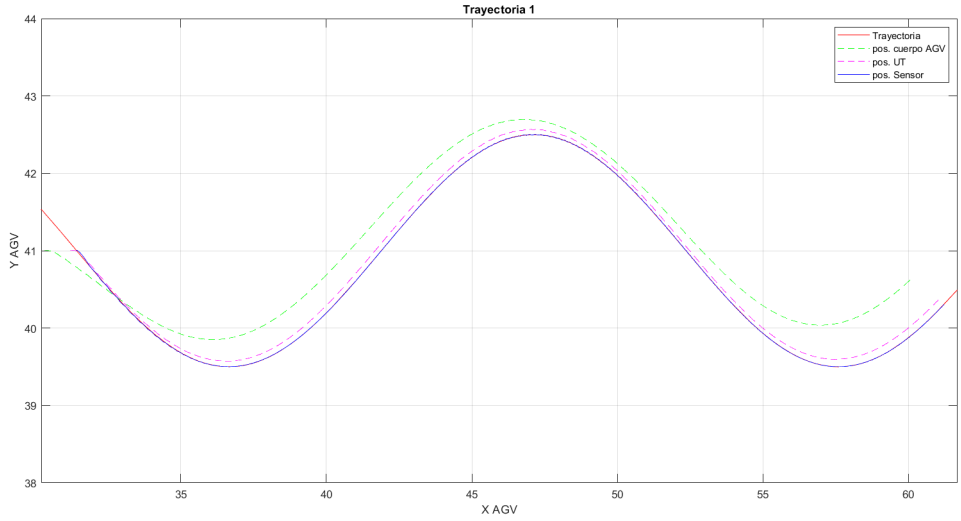

Figura 6.16: ΔQPID-PI:Seguimiento de trayectoria

Tras la simulación, se observa que los valores finales sintonizados por el algoritmo RL son: $Kp = 6{,}186631173324591$, $Kd = 6{,}970929863884416$ y $Ki = 0{,}280162519502034$.

- **Error de seguimiento** Al observar de manera más detallada en la figura 6.17 dichas oscilaciones, se aprecia que no son tanto como parecen, ya que si comparamos con la misma gráfica para el control PID-PI, la amplitud de las oscilaciones es prácticamente la misma, siendo la frecuencia de estas levemente mayor en este caso:

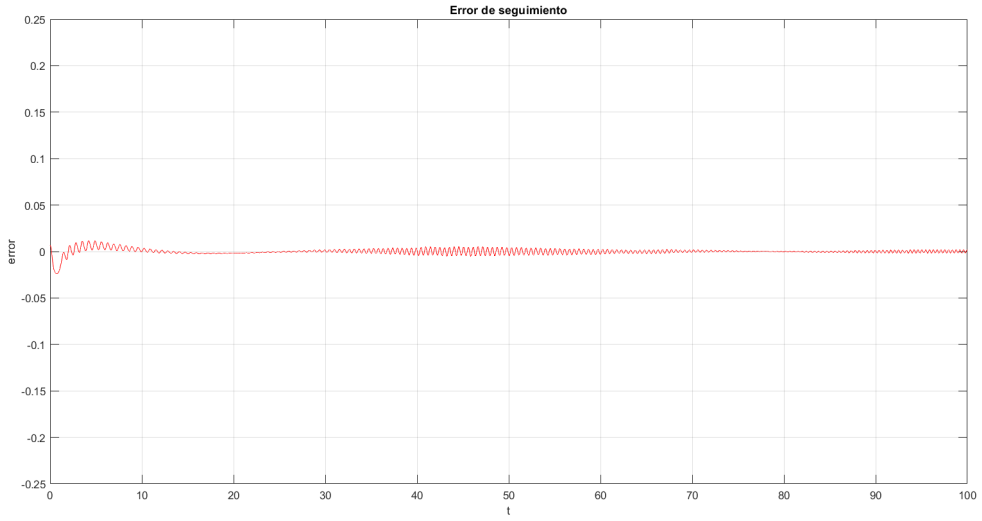

Figura 6.17: ΔQPID-PI:Error de seguimiento

- **Velocidad crucero y error**: Tras observar la velocidad de crucero y su error, se aprecian ciertas fluctuaciones en mitad de su recorrido pero que carecen de mayor importancia ya que la velocidad objetivo de $0.5m/s$ se alcanza de una manera satisfactoria. La gráfica fig. 6.18 indica la buena tendencia de este a cero:

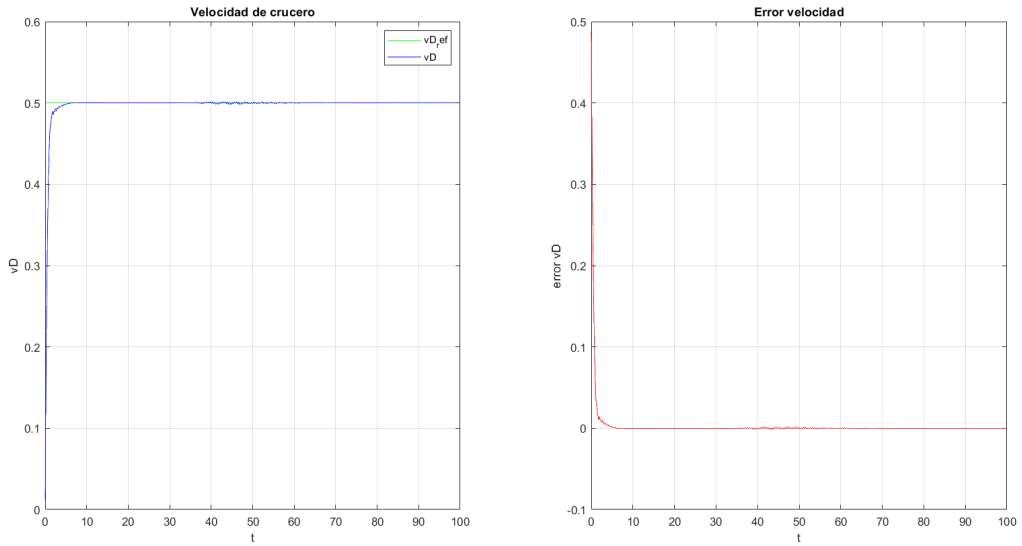

Figura 6.18: ΔQPID-PI:Velocidad de crucero y su error

- **Errores de velocidad**: La figura 6.19 muestra las velocidades de las ruedas respecto a las de referencia y el error entre estas. En la zona central es donde las velocidades adquieren mayor amplitud y es justamente la zona donde la velocidad de crucero presenta unas ligeras fluctuaciones que provocan un pequeño error. Conforme avanza la simulación, estas amplitudes disminuyen ya que el control RL está continuamente actualizándose:

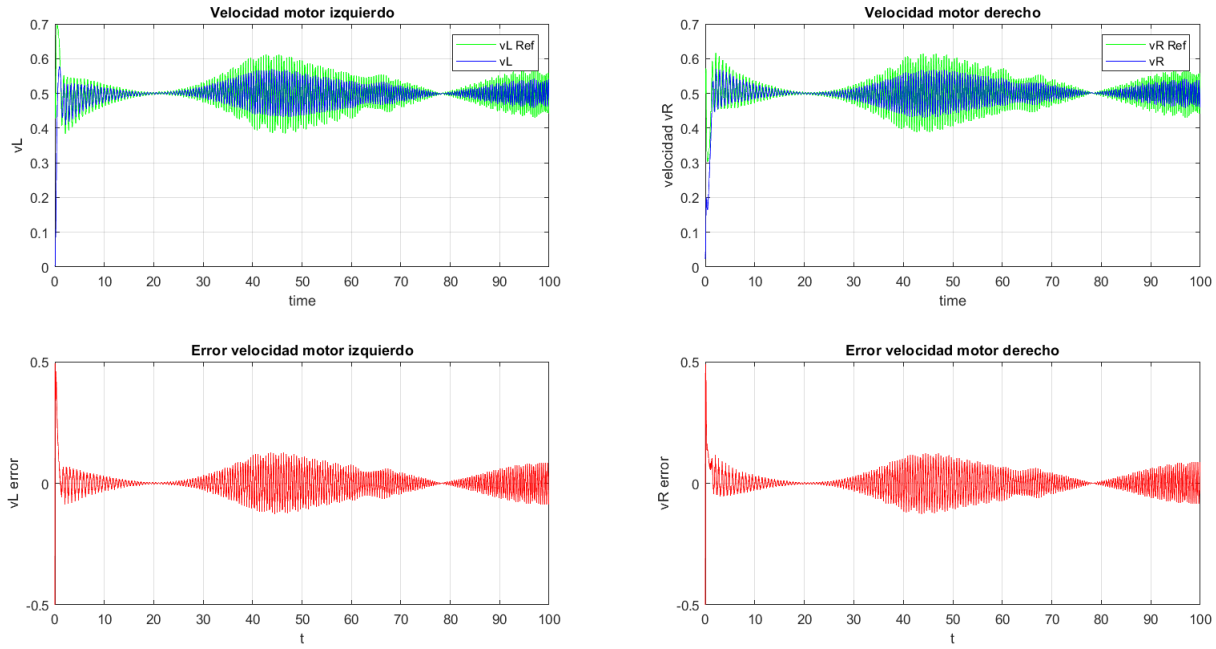

Figura 6.19: ΔQPID-PI:Error de velocidad en ruedas

- **Gráficas motores**: En el comportamiento de los motores, mostrado en fig. 6.20, fig. 6.21 y fig. 6.22, se ven reflejas las variaciones comentadas, tanto en pares-motor como en corrientes aplicadas. Todo ello hace pensar que este control acaba empeorando el comportamiento del control original PID y en cierta manera es posible ya que el controlador PID es continuo utiliza información continua para obtener una acción de control que también es continua. Sin embargo, con el control basado en RL, dicha información de entrada continua, se ha tenido que discretizar para reducir el tiempo de cómputo, de la misma manera que las acciones de control (aplicada a los parámetros del PID) también están discretizadas. Con ello el controlador no puede incrementar o disminuir tales parámetros de una manera ideal ya que existirán valores intermedios más óptimos. Una posible solución para mejorar las acciones de control sobre los parámetros K_p, K_d y K_i pasaría por calcular tales acciones basadas en distintas ecuaciones que fueran función del valor continuo leído directamente del sensor, es decir, del error. Con esto, el controlador podría aprender que ecuación es mejor usar para cada estado. Por otro lado, se puede dar más peso a Δ_{w_D} en la ecuación 5.5 para tratar de suavizar la respuesta del AGV. Con ello se podría conseguir una mejora en cuanto a la manera de actuar sobre los parámetros, aunque aun quedaría pendiente mejorar la entrada de información al controlador. En [8] se puede encontrar información sobre como aplicar las técnicas de aprendizaje por refuerzo a sistemas continuos. Por último, solo comentar que tras un entrenamiento largo no se consiguió la convergencia de las matrices Q, por lo que mejores resultados a este se pueden conseguir si se extiende dicho entrenamiento.

1. Δ*QPID-PI:Velocidades angulares y par motor*:

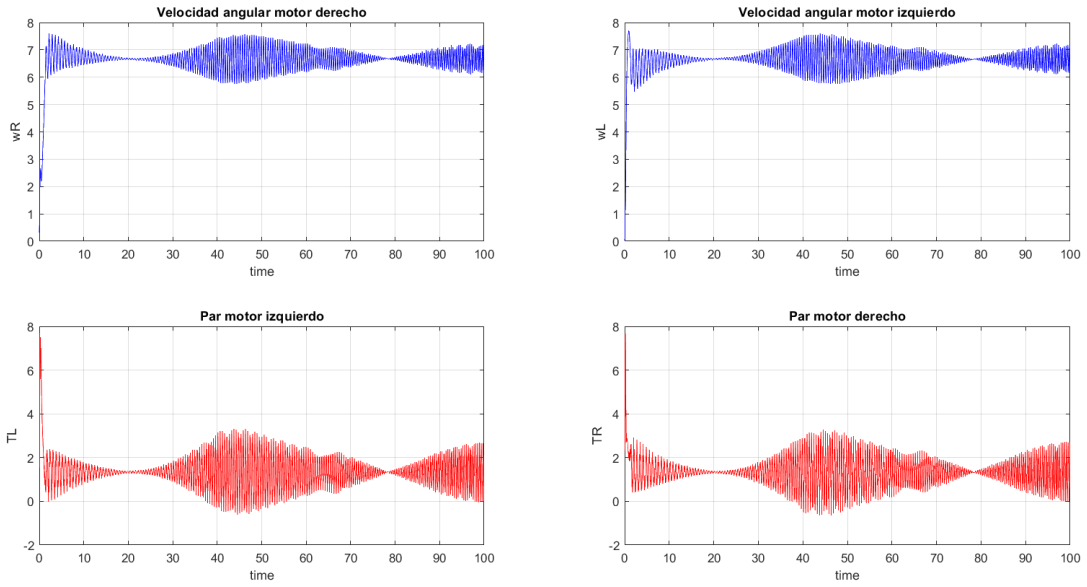

Figura 6.20: ΔQPID-PI:Velocidades angulares y par motor

2. *Corrientes motor*:

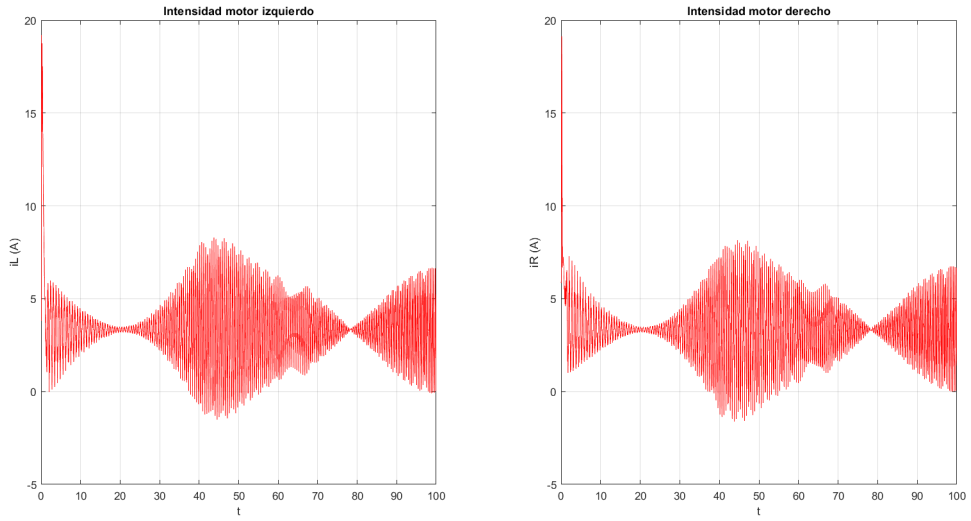

Figura 6.21: ΔQPID-PI:Intensidades motores

3. *Voltaje motores*:

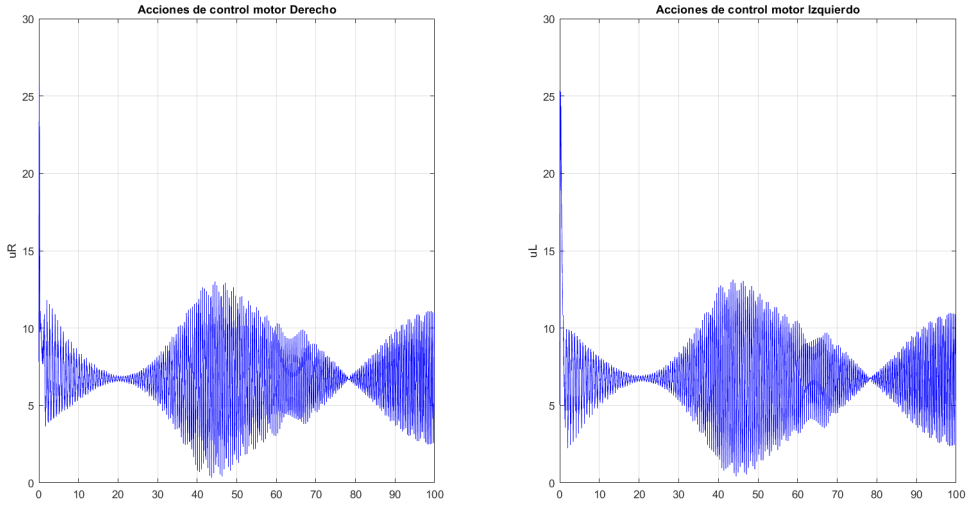

Figura 6.22: ΔQPID-PI:Tensiones motores

- **Tablas de valores** $Q_K(s, a)$: Con la intención de poder ver las matrices Q finales, las imágenes fig. 6.23, fig. 6.24 y fig. 6.25 muestran dichas matrices en donde los colores blancos representan un valor q elevado y los oscuros un valor bajo. En ellas se puede ver en que regiones de pares estado-acción se ha trabajado más. Las zonas en las que se han empleado más muestras son aquellas que presentan un cambio de contraste gradual y suave, mientras las zonas con cambios de contraste brusco indican zonas levemente exploradas.

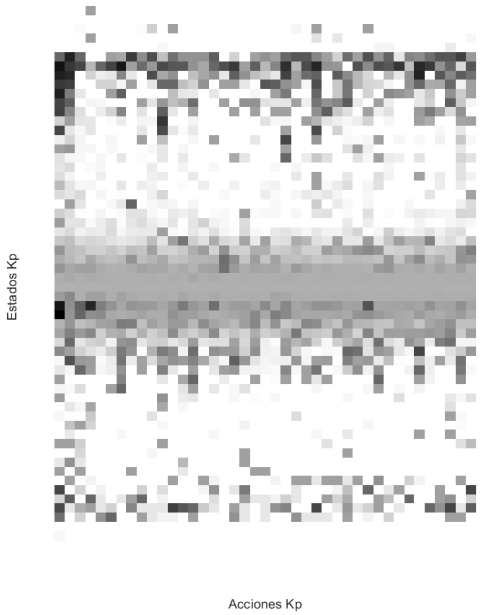

Figura 6.23: ΔQPID-PI:Kp Q-Table

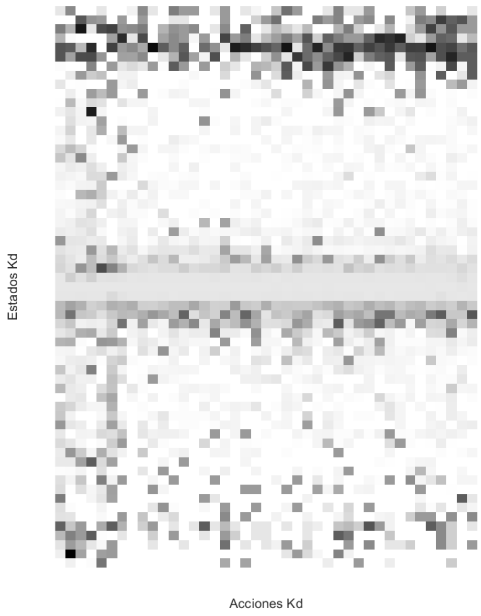

Figura 6.24: ΔQPID-PI:Kd Q-Table

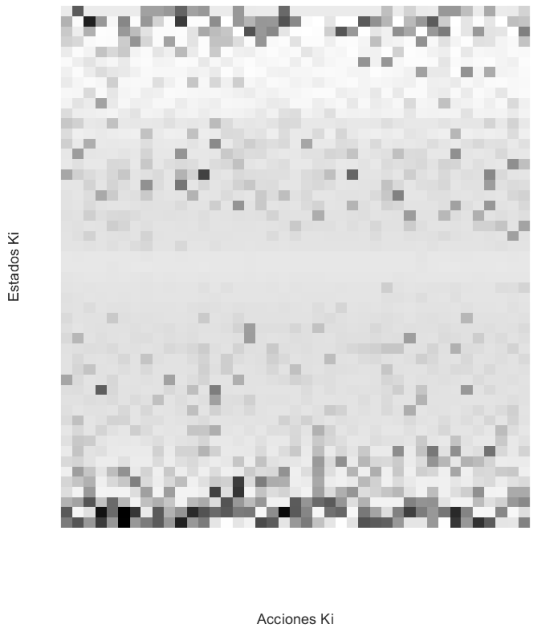

Figura 6.25: ΔQPID-PI:Ki Q-Table

6.4. Control ΔQPID-NNC

6.4.1. Trayectoria senoidal

- **Trayectoria**:En fig. 6.26 se puede ver como el control combinado ΔQPID-NNC traza de una manera correcta la trayectoria. Inicialmente presenta oscilaciones sobre la trayectoria pero que conforme avanza la simulación va desapareciendo hasta conseguir trazarla de una manera bastante precisa. La conclusión sobre este comportamiento es que el control ΔQPID está funcionando de manera esperada, mejorando así la sintonización de los parámetros del PID conforme pasa el tiempo.

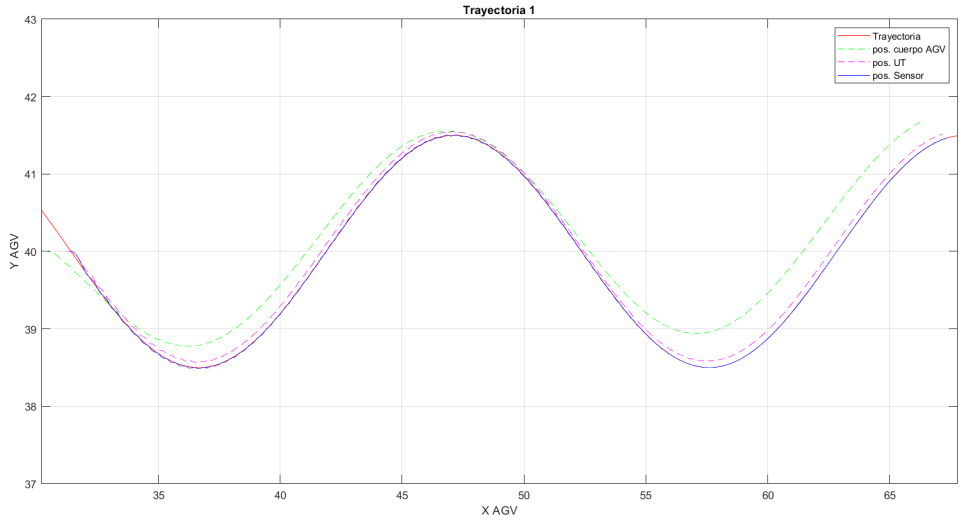

Figura 6.26: ΔQPID-NNC:Seguimiento de trayectoria

- **Error de seguimiento** si observamos el error de seguimiento en fig. 6.27, parece corroborarse lo comentado en la imagen anterior y es que claramente se ve como dicho error va disminuyendo de manera notable. Únicamente, cerca del final del tiempo de simulación parece ser que vuelve a aumentar pero no con la misma amplitud que las oscilaciones iniciales.

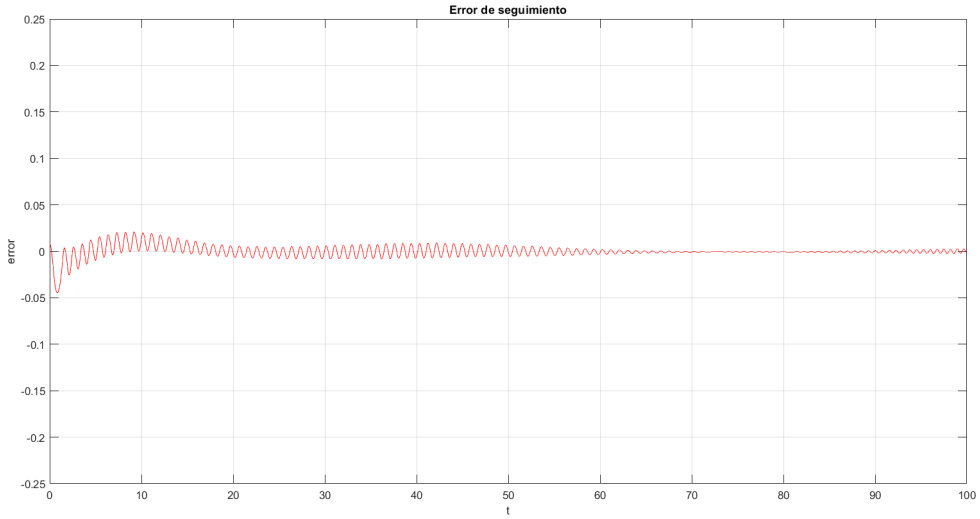

Figura 6.27: ΔQPID-NNC:Error de seguimiento

- **Velocidad crucero y error**: En lo que respecta al comportamiento de la velocidad de crucero, en la figura 6.28 se ve el mismo comportamiento que el observado en el control PID-NNC. El control RL no es capaz por de ajustar el PID de manera que estas desaparezcan. Esto da a pensar que el PID no puede conseguir ese objetivo ni aun con un reajuste, como se comentó anteriormente, mediante el algoritmo genético, en el que se añadiría la velocidad de crucero como penalización en la función de coste. Esto resulta ser obvio ya que el PID debe centrarse en la corrección de la trayectoria y no de la velocidad. Con todo ello, muy probablemente no se haya escogido el modelo de referencia adecuado o las redes neuronales estén ajustadas adecuadamente. Aun así, el AGV consigue llegar al final de la trayectoria con una velocidad de crucero apropiada.

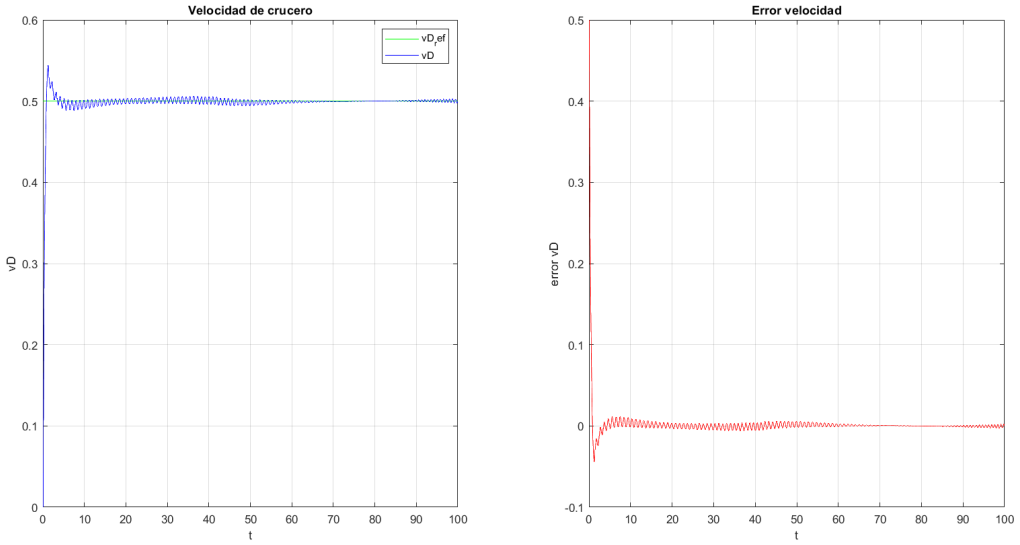

Figura 6.28: ΔQPID-NNC:Velocidad de crucero y su error

- **Aproximación al modelo de referencia**: Después analizar el resultado de la figura anterior y comprobando en la figura 6.29 que el control neuronal lleva la salida del modelo dinámico a la del modelo de referencia de una manera bastante aproximada, se llega a la conclusión de que el comportamiento de este controlador debería de ser analizado más detalladamente ya que, aunque válidos, los resultados no son los esperados.

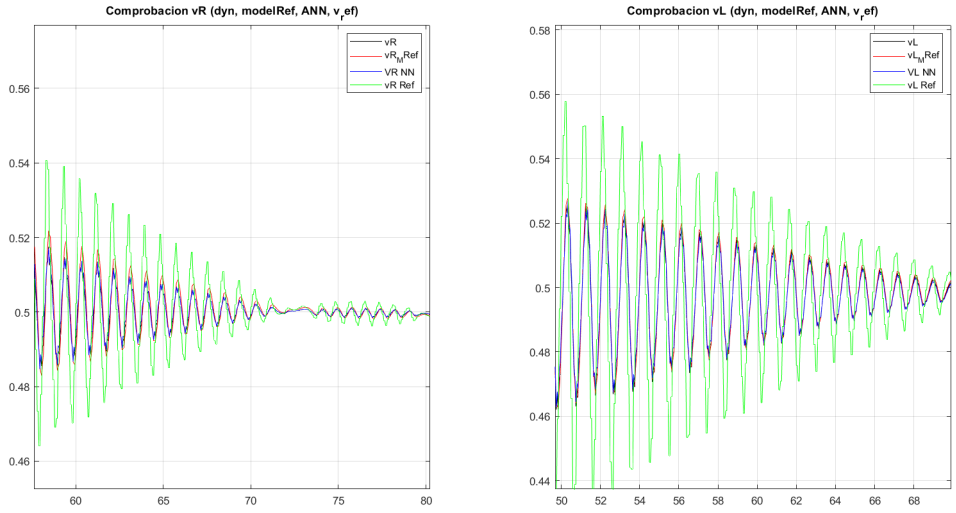

Figura 6.29: ΔQPID-NNC:Aproximación a Modelo de Referencia

- **Gráficas motores**: En lo que respecta a las velocidades angulares y pares-motor, en la figura 6.30 se observa como la respuesta con la combinación de estos motores presentan unas variaciones relativamente suaves en comparación del control ΔQPID-PI con el que las variaciones son bastantes bruscas. Las respuestas mostradas para este control se asemejan bastante con el control PID-PI y el control PID-NNC. Este mismo comportamiento se aprecia en las tensiones de la figura 6.32.

1. *ΔQPID-NNC:Velocidades angulares y par motor*:

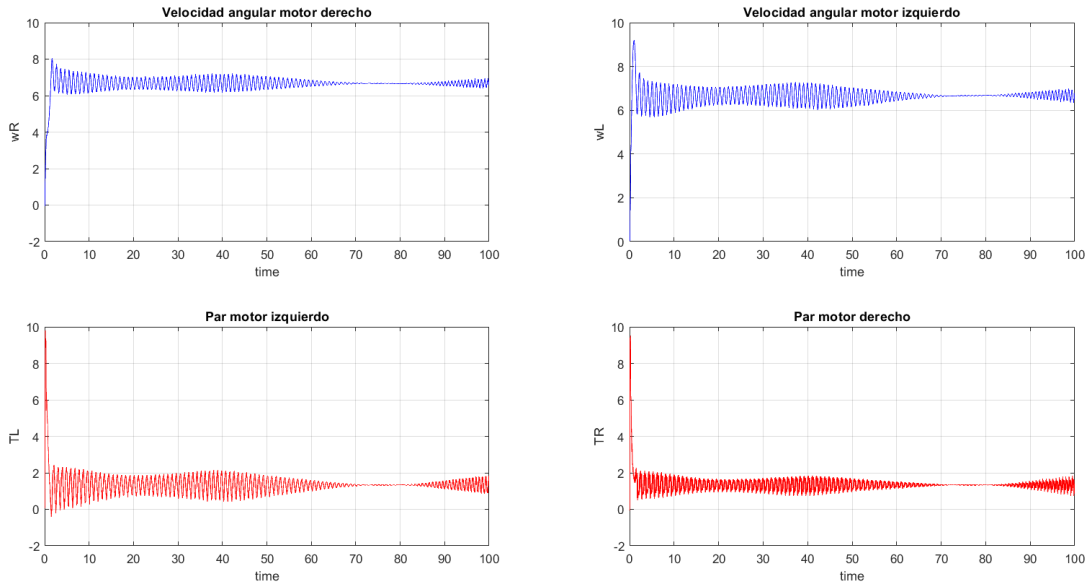

Figura 6.30: ΔQPID-NNC:Velocidades angulares y par motor

2. *Velocidades angulares y par motor*:

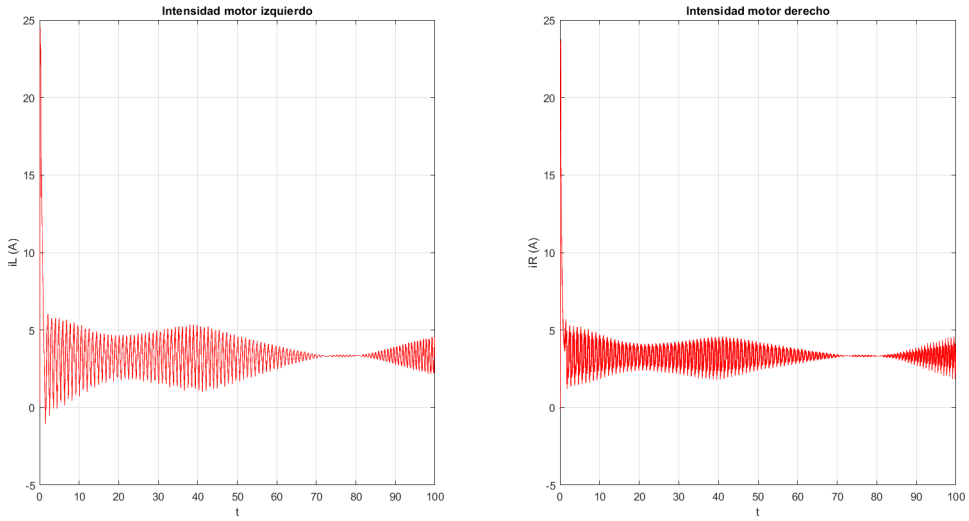

Figura 6.31: ΔQPID-NNC:Intensidades motores

3. *Voltaje motores*:

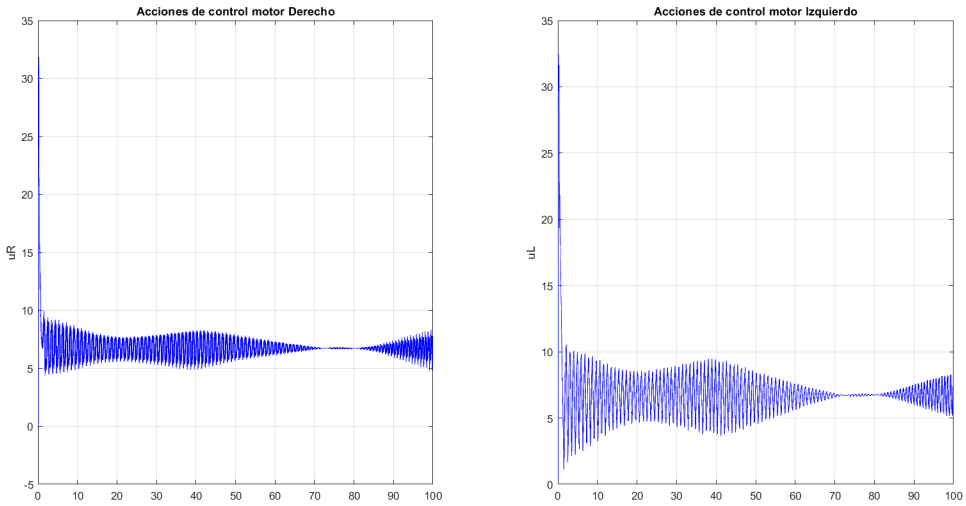

Figura 6.32: ΔQPID-NNC:Tensiones motores

- **Tablas de valores** $Q_K(s, a)$: En lo que respecta a las imágenes sobre las tablas Q, no se considera necesario mostrarlas dado que después de analizarlas se comprobó que son prácticamente igual a las mostradas en fig. 6.23, fig. 6.24 y fig. 6.25. Esto es debido a que no se volvió a entrenar el control por un largo periodo de tiempo. Aun así, la variación hubiera sido difícil de apreciar ya que el control neuronal se aproxima bastante a los controles PI y los valores de las tablas Q mostrados en tales figuras fueron obtenidos en simulaciones para el control ΔQPID-PI.

6.5. Comparación de los distintos controles

En cada una de las pruebas realizas se calcula el valor del error cuadrático medio (MSE) con el se comparará del rendimiento de los diferentes controles aplicados al AGV. Las siguientes tablas recoge tal información:

	Controlador PID-PI: Trayectoria senoidal	**MSE**
1.	Error de seguimiento	8.247e-6
2.	Error de velocidad crucero	5.0019e-04

Tabla 6.1: Evaluación del controlador PID-PI

	Controlador PID-NNC: Trayectoria senoidal	**MSE**
1.	Error de seguimiento	4.8014e-05
2.	Error de velocidad crucero	6.4966e-04

Tabla 6.2: Evaluación del controlador PID-NNC

	Controlador ΔQPID-PI: Trayectoria senoidal	MSE
1.	Error de seguimiento	7.39e-6
2.	Error de velocidad crucero	6.954e-4

Tabla 6.3: Evaluación del controladorΔQPID-PI

	Controlador ΔQPID-NNC: Trayectoria senoidal	MSE
1.	Error de seguimiento	2.652e-5
2.	Error de velocidad crucero	6.2734e-4

Tabla 6.4: Evaluación del controladorΔQPID-NNC

En lo que respecta al error de seguimiento, el control ΔQPID-PI es el que mejor se comporta, lo cual es normal ya que este control usa, inicialmente, las ganancias del PID ajustadas para esta trayectoria, y sobre las cuales, el algoritmo RL realiza un ajuste fino, pero faltaría por comprobar que para trayectorias distintas, este ajuste fino se mantiene igual de bien. Comparando ahora el error de la velocidad de crucero, el control que presenta el mejor rendimiento es el PID-PI, aunque el resto de controles están muy cerca de dicha métrica. Dado que el sintonizado se realizó para esta trayectoria, no es de extrañar el resultado dado el buen rendimiento de control proporcional integral derivativo.

Respecto a los resultados del control neuronal para la velocidad de las ruedas, hay que decir que no eran los esperados a obtener al inicio del trabajo. Aun así, y dada la buena aproximación del sistema al modelo de referencia, hay bastantes indicios de que con una mejora en el modelo de referencia, se podrían obtener los resultados esperados.

Capítulo 7

Conclusiones y trabajos futuros

7.1. Conclusiones

En este trabajo se ha desarrollado e implementado el modelado y control de un robot comercial de la empresa ASTI Mobile Robotics en el que se partía de la información sobre el modelo mostrado en [1]

Se ha desarrollado un modelo dinámico de este AGV mediante el análisis de fuerzas y momentos, añadiendo también motores de corriente continua para completo desarrollo. Con esto se ha obtenido un modelo completo en espacio de estados para su posterior simulación.

Como control clásico para el control del AGV se ha diseñado un control PID-PI con parámetros del PID sintonizados mediante un algoritmo genético.

Se ha diseñado un control neuronal con modelo de referencia para el control de velocidad, con que se aproxima la respuesta del AGV a la de un sistema multivariable lineal y el cual tiene la capacidad de adaptar su acción de control en función de variaciones que se puedan producir sobre el sistema dinámico. Para el control de seguimiento con este control, se diseña un control PID cuyos parámetros son sintonizados mediante un algoritmo genético.

Mejorada la capacidad de adaptación del control de velocidad, se ha diseñado un control incremental basado en PID y técnicas de Aprendizaje por Refuerzo (Q-Learning), con objetivo de mejorar la adaptabilidad del AGV ante la presencia de nuevas trayectorias distintas a la considerada como nominal. A este control se diseñó junto a dos controles PI para el control de velocidad.

Se ha empleado para el desarrollo y diseño de todo el trabajo en el entorno de MATLAB, en el cual se ha realizado una implementación basada en programación orientada a objetos con la intención de simplificar el código creado y facilitar la reutilización del código generado.

Con todo ello, se han realizado pruebas sobre distintas combinaciones de los controles diseñados en los que se aprecian mejoras en cuanto a seguimiento de trayectoria cuando se hace uso del controlador ΔQPID-PI. No se ha conseguido obtener una mejora significativa en la velocidad ni usando el control con red neuronal con modelo de referencia en combinación con un PID (PID-NNC), ni con la combinación de las técnicas de aprendizaje por refuerzo y redes neuronales ΔQPID-NNC. La causa de ello se cree que esta en la necesidad de mejorar el modelo de referencia del control neuronal. Aún así, se ha podido comprobar que el diseño funciona correctamente dado que se consigue mantener la velocidad de crucero alrededor de su valor de referencia mientras que el AGV traza su trayectoria de manera satisfactoria.

Dados los experimentos de simulación y sus resultados, se llega a la conclusión que el control ΔQPID-PI es la mejor opción a implementar sobre el AGV dado que es capaz de reajustar las ganancias del PID durante la trayectoria y la velocidad de crucero se mantiene constante. No hay que olvidar que todas las pruebas realizadas han sido sobre la misma trayectoria y, dado que el

control proporcional integral derivativo empeora su rendimiento conforme se aleja de su punto de trabajo (trayectoria senoidal usada), con una mejora en el control neuronal para corregir las oscilaciones observadas en los resultados, muy posiblemente el control ΔQPID-NNC podría ser la mejor opción a implementar en el AGV.

7.2. Trabajos futuros

Tras encontrar el trabajo aquí desarrollado bastante interesante en lo que refiere a control mediante redes neuronales y aprendizaje por refuerzo, se deja como linea futura de trabajo el extender lo visto aquí con control también basado en aprendizaje por refuerzo pero que incluye redes neuronales como es Deep Q-Learning, ya que se considerada muy interesante y cuyo objetivo sería el mejorar los resultados aquí obtenidos ya que suele ser bastante empleado para sistemas continuos. Por otro lado, también un análisis más profundo sobre el controlador neuronal con modelo de referencia.

Bibliografía

[1] Roberto Sanchez Martinez, (2020) *Trabajo fin de Master: Modelado y simulacion de un AGV híbrido triciclo-diferencia*,UNED-UCM

[2] Alfonso Urquia Moraleda y Carla Martin Villalba, (2018), *Modelado orientado a objetos y simulacion de sistemas fisicos*, Dpto. de Informatica y Automatica Escuela Tecnica Superior de Ingenieria Informatica, UNED, Madrid

[3] Gregor Klancar, Andrej Zdesar, Saso Blazic y Igor Skrjanc, (2017), *Wheeled Mobile Robotics 1st Edition*, Butterworth-Heinemann, Oxford, UK

[4] Agata NAWROCKA, *Syntesis of Neural Network Controller with a Reference Model*, (2010), Department of process control, AGH University of Science and Technology, Krakow, Poland

[5] Sabrine Slama, Ayachi Errachdi y Mohamed Benrejeb,(2017), *A Neural Model Reference Adaptive Controller Algorithm for Nonlinear Systems*, Automation Research Laboratory, Tunis El Manar University, Tunissa

[6] Martin T. Hagan, Howard B Demuth, Mark H Beale, Oklahoma y Orlando De Jesús (2014) *Neural Network Design 2nd Edtion*, Martin Hagan

[7] J. Enrique Sierra-Garcia y Matilde Santos (2020) *Mechatronic Modelling of Industrial AGVs: A Complex System Architecture*, WILEY, Department of Electromechanical Engineering, University of Burgos and Institute of Knowledge Technology, Complutense University of Madrid, Spain

[8] Richard S. Sutton y Andrew G. Barto (2014-2015) *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, Massachusetts, London, England

[9] RAnand Deshpande y Manish Kumar (2018) *Artificial Intelligence for Big Data*, Packt, Birmingham-Mumbai

[10] GONZALO FARIAS, GONZALO GARCIA, GUELIS MONTENEGRO, ERNESTO FABREGAS, SEBASTIÁN DORMIDO-CANTO, y SEBASTIÁN DORMIDO (2020) *Reinforcement learning for position control problem of a mobile robot*, IEE Access, Escuela de Ingeniería Eléctrica, Pontificia Universidad Católica de Valparaíso, Ocean and Mechanical Engineering Florida Atlantic University (FAU), Departamento de Informática y Automática Universidad Nacional de Educación a Distancia (UNED)

[11] Yunxiao Qin, Weiguo Zhang, Jingping Shi y Jinglong Liu (2018) *Improve PID controller through reinforcement learning*, ResearchGate, Shanghai Institute of Aerospace System Engineering, China

[12] [Online] Disponible: *http://en.wikipedia.org*

[13] [Online] Disponible: *https://www.automation.com/*

[14] [Online] Disponible: *https://mathworks.com/*

[15] Cristian Troncoso G. y Alejandro Súarez S. (2017) *Control del Nivel de Pulpa en un Circuito de Flotación Utilizando una Estrategia de Control Predictivo.*, Revista Iberoamericana de Automática e Informática industrial, Departamento de Electrónica de la Universidad Técnica Federico Santa María, Av. España 1680, Valparaíso, Chile

[16] K. KrishnaKumar (1994) *ADAPTIVE NEURO-CONTROL FOR SPACECRAFT ATTITUDE CONTROL* , Department of Aerospace Engineering The University of Alabama, Tuscaloosa, AL 35487-0280, USA.

[17] Gwo-Ching Chang, Jer-Junn Luh, Gon-Der Liao, Jin-Shin Lai, Cheng-Kung Cheng, Bor-Lin Kuo, y Te-Son Kuo (1997) *A Neuro-Control System for the Knee Joint Position Control with Quadriceps Stimulation*,IEEE.

[18] Jorg Heitkotter y David Beasley, (1999), *The Hitch-Hiker's Guide to Evolutionary Computation*, UUnet Deutschland GmbH y University of Bath, Germany, UK

[19] Jesús M. de la Cruz y José A. López Orozco, (1999), *Introducción a los Algoritmos Genéticos*, Dpto. Informática y Automática, UCM, España

[20] Sanchez, R, Sierra-García, J.E y Santos, M, (2021), *Modelado de un AGV híbrido* , Revista Iberoamericana de Automática e Informática industrial

[21] Sierra-García, J. E, Santos, C y Sierra-García, J. E, (2020), *Transporte multi-AGV de una carga: estado del arte y propuesta centralizada* , Revista Iberoamericana de Automática e Informática industrial, 18(1), 82-91.

[22] Molledo, V. R y García, J. E. S, (2021), *Simulation Tool for Hybrid AGVs based on IEC-61131*, IEEE Latin America Transactions, 100(XXX)

[23] Zamora-Cadenas, L, Velez, I y Sierra-García, J. E, (2021), *UWB-Based Safety System for Autonomous Guided Vehicles Without Hardware on the Infrastructure*, IEEE Access, 9, 96430-96443.

[24] Sierra-García, J. E y Santos, M, (2020), *Control of Industrial AGV Based on Reinforcement Learning*, In International Workshop on Soft Computing Models in Industrial and Environmental Applications (pp. 647-656). Springer, Cham.

Appendices

Apéndice A

Causalidad computacional del sistema

El sistema de ecuaciones con la causalidad computacional asignada queda de la siguiente manera:

$$\dot{i}_L = \frac{u_L}{L_m} - \frac{R_m\, i_L}{L_m} - \frac{K_b\, w_L}{L_m} \tag{A.1}$$

$$\dot{w}_L = \frac{\sigma_3}{J_L} \tag{A.2}$$

$$\dot{i}_R = \frac{u_R}{L_m} - \frac{R_m\, i_R}{L_m} - \frac{K_b\, w_R}{L_m} \tag{A.3}$$

$$\dot{w}_R = \frac{\sigma_2}{J_R} \tag{A.4}$$

$$T_{\mathrm{mR}} = \frac{\sigma_2}{f} \tag{A.5}$$

$$F_R = \frac{T_{\mathrm{mR}}}{r} \tag{A.6}$$

$$T_{\mathrm{mL}} = \frac{\sigma_3}{f} \tag{A.7}$$

$$F_L = \frac{T_{\mathrm{mL}}}{r} \tag{A.8}$$

$$\dot{w}_D = -\frac{B\,(-F_R + F_L)}{2\,I_D} \tag{A.9}$$

$$\dot{\beta} = w_D \tag{A.10}$$

$$R_x = F_R + F_L \tag{A.11}$$

$$R_y = F_{\mathrm{cD}} \tag{A.12}$$

$$F_{\mathrm{Tx}} = R_x\, \cos(\beta) - R_y\, \sin(\beta) \tag{A.13}$$

$$F_{\mathrm{Ty}} = R_y\, \cos(\beta) + R_x\, \sin(\beta) \tag{A.14}$$

$$\dot{w}_T = \frac{F_{\mathrm{Tx}}\, \mathrm{cmy}_T + F_{\mathrm{Ty}}\,(L - \mathrm{cmx}_T)}{\mathrm{IT}} \tag{A.15}$$

$$\dot{v}_{Tx} = \frac{F_{\mathrm{Tx}}}{m_2} \tag{A.16}$$

$$v_{\mathrm{Ty}} = \frac{\sigma_1\,(F_{\mathrm{Ty}} + F_{\mathrm{cT}})}{m_2\,(w_T\, \cos(\Omega) + w_T\, \sin(\Omega))} \tag{A.17}$$

$$\dot{v}_{T0x} = \dot{v}_{Tx}\, \cos(\Omega) - \sigma_5 - v_{\mathrm{Tx}}\, w_T\, \sin(\Omega) - \frac{\sin(\Omega)\;(\sigma_5 + \sigma_4)}{\sigma_1} \tag{A.18}$$

$$\dot{v}_{T0y} = \dot{v}_{Tx}\, \sin(\Omega) + v_{\mathrm{Tx}}\, w_T\, \cos(\Omega) - \sigma_4 + \frac{\cos(\Omega)\;(\sigma_5 + \sigma_4)}{\sigma_1} \tag{A.19}$$

$$\dot{x}_T = v_{\mathrm{T0x}} \tag{A.20}$$

$$\dot{y}_T = v_{\mathrm{T0y}} \tag{A.21}$$

con:

$$\sigma_1 = \cos(\Omega) - \sin(\Omega) \tag{A.22}$$
$$\sigma_2 = K_m\, i_R - B_m\, w_R \tag{A.23}$$
$$\sigma_3 = K_m\, i_L - B_m\, w_L \tag{A.24}$$
$$\sigma_4 = v_{\mathrm{Ty}}\, w_T\, \sin(\Omega) \tag{A.25}$$
$$\sigma_5 = v_{\mathrm{Ty}}\, w_T\, \cos(\Omega) \tag{A.26}$$

donde los parámetros correspondientes al AGV y los motores se muestran de nuevo para facilitar la lectura del lector:

	Descripción	**Valores**
dD	Distancia del eje del bloque diferencial al sensor	$0.2m$
L	Distancia entre trasero y delantero	$0.9m$
B	Distancia entre centros de ruedas	$0.5m$
Ws	Longitud del sensor	$0.5m$
d	Distancia entre centro del eje diferencial y extremo frontal	$0.58m$
cmx_T	Coordenada x del centro de gravedad del AGV	$0.45m$
cmy_T	Coordenada y del centro de gravedad del AGV	$0m$
m_2	Peso del AGV	$200kg$
R_m	Resistencia	1.3Ω
L_m	Inductancia	$0.00005H$
K_b	Constante fuerza contra electromotriz	$0.36V/rad/s$
K_m	Constante de la armadura	$0.5Nm/A$
B_m	Fricción viscosa	$0.58N\ m\ s$
f	Coeficiente de reducción	1
r	Radio de las ruedas	$0.075m$
J_m	Momento del inercia del conjunto motor-reductor (fabricante)	$0.0640kg\ mm^2$

Tabla A.1: Valores de parámetros empleados

el resto de variables que aparecen en las ecuaciones corresponden a variables desconocidas cuyo significado se puede encontrar en el capitulo de modelado del AGV.

Apéndice B

Velocidades de referencia motores

A continuación se muestra el calculo de las velocidades de referencia de cada una de las ruedas de tracción para el controlador de velocidad:

$$
\begin{aligned}
&w_{\text{Dref}} = w_D + \Delta w \\
&v_{\text{Lref}} = v_{\text{Dref}} - w_{\text{Dref}}\ 0.5\ B \\
&v_{\text{Rref}} = w_{\text{Dref}}\ B + v_{\text{Lref}}
\end{aligned}
\tag{B.1}
$$

donde :
$\Delta w \longrightarrow$ accion de control del controlador PID
$v_{\text{Dref}} \longrightarrow$ consigna de velocidad de crucero
$w_{\text{Dref}} \longrightarrow$ velocidad angular de referencia que debe adquirir la unidad de traccion
B $\longrightarrow$ distancia entre ruedas motrices

Apéndice C

Objetos creados en MATLAB

Para resolver el problema de control de este trabajo se ha usado programación orientada a objetos en el entorno de MATLAB. Se ha tratado de encapsular en medida de lo posible el código creado en las clases correspondientes a cada uno de los elementos de los elementos del problema de control. Un breve resumen de estos objetos junto con sus estructuras y métodos de cada uno de ellos se pueden ver en este apartado:

- **CAGVDynamics**: Contiene el sistema dinámico del AGV para cuya simulación se hace uso de la función de MATLAB *ode45*;
- **CMotorRueda**: Contiene un modelo para el conjunto motor-rueda simulado mediante *ode45* y su creación es unicamente para componer otros objetos mas complejos. Este objeto no es usado para simular el sistema del AGV ya que
- **CMagneticSensor**: encapsula todo el código necesario para crea un modelo que reproduzca el comportamiento de un sensor magnético usado en AGVs filoguiados. Con el se obtiene la distancia al punto de corte entre la trayectoria y el sensor.
- **CPIController**: Objeto para el modelo de un controlador PI discreto.
- **CPIDController**: Objeto para el modelo de un controlador PID discreto.
- **CLocalizacionAGV**: este objeto es unicamente creado para la obtención y visualización de la estructura del cuerpo del AGV, empleando para ello matrices homogéneas de transformación, ya que en la programación se añade una opción de animación dinámica en la que se puede ver al robot trazando las trayectorias. Su único propósito es del de depuración ya que unicamente con gráficas es difícil de apreciar el movimiento real que el robot realiza.
- **CLocalizacionDiferencial**: objeto similar al anterior pero para la unidad de tracción.
- **CReferenceModel**: encapsula el modelo de referencia a usar en el controlador neuronal. Emplea también *ode45* para la simulación e incluye la posibilidad de modificar parámetros de este. No es usado por si solo.
- **CANNController**: objeto que agrupa toda la estructura del controlador neuronal con modelo de referencia. Es el objeto mas complejo de todos y con mas métodos ya que se compone de: una red neuronal para el modelo, otra para el controlador (ambas con configuración ajustable), métodos de adaptación $on - line$ para esta y el modelo de referencia.

CAGVDymanics

`Constructor`

Class Details

Superclasses	handle
Sealed	false
Construct on load	false

Constructor Summary

CAGVDymanics	Constructor

Method Summary

	addLoad	Añade carga/s extra a transportar por el AGV
	addlistener	Add listener for event. Help for CAGVDymanics/addlistener is inherited from superclass HANDLE
	applyDynamics	Simulacion del sistema dinamico desde t(k) hasta t(k)+Delta_t (hace uso de ode45)
	calcAGVCenterOfMass	Calcula la nueva ubicacion del centro de masas tras añadir una carga
	calcCentripetalForces	Calcula las fuerzas centripetas (no usa al estar estas fuerzas en las ecuaciones dinamicas)
	calcDistributionLoads	Calcula las cargas verticales sobre las ruedas
	calcInertias	Calcula las inercias de las ruedas y unidad de traccion y cuerpo del AGV
	delete	Destructor
	eq	== (EQ) Test handle equality. Help for CAGVDymanics/eq is inherited from superclass HANDLE
	findobj	Find objects matching specified conditions. Help for CAGVDymanics/findobj is inherited from superclass HANDLE
	findprop	Find property of MATLAB handle object. Help for CAGVDymanics/findprop is inherited from superclass HANDLE
	ge	>= (GE) Greater than or equal relation for handles. Help for CAGVDymanics/ge is inherited from superclass HANDLE
	getAGVCenterOfMass	Leer la posicion actual del centro de masas del AGV
	getAGVTotalMass	Leer el peso total (carga incluida) del AGV
	getAngularSpeeds	Leer las velocidades angulares del AGV
	getDistributionLoads	Leer la distribucion de cargas verticales actual
	getInertias	Leer las inercias del AGV
	getMotorCurrents	Leer las corrientes de los motores
	gt	> (GT) Greater than relation for handles. Help for CAGVDymanics/gt is inherited from superclass HANDLE
Sealed	isvalid	Test handle validity. Help for CAGVDymanics/isvalid is inherited from superclass HANDLE
	le	<= (LE) Less than or equal relation for handles. Help for CAGVDymanics/le is inherited from superclass HANDLE
	listener	Add listener for event without binding the listener to the source object. Help for CAGVDymanics/listener is inherited from superclass HANDLE
	lt	< (LT) Less than relation for handles. Help for CAGVDymanics/lt is inherited from superclass HANDLE
	ne	~= (NE) Not equal relation for handles. Help for CAGVDymanics/ne is inherited from superclass HANDLE
	notify	Notify listeners of event. Help for CAGVDymanics/notify is inherited from superclass HANDLE
	setAGVCenterOfMass	Fijar el centro de masas inicial
	setInitialConditions_AGV	Condiciones iniciales del cuerpo del AGV
	setInitialConditions_MOTORS	Condiciones iniciales de los motores
	setInitialConditions_UT	Condiciones iniciales de la unidad de traccion
	setParameters	Fijar los parametros del AGV

Event Summary

ObjectBeingDestroyed	Notifies listeners that a particular object has been destroyed. Help for CAGVDymanics/ObjectBeingDestroyed is inherited from superclass HANDLE

Figura C.1: Objeto CAGVDynamics

CANNController

Constructor

Class Details

Superclasses	handle
Sealed	false
Construct on load	false

Constructor Summary

CANNController	Constructor

Method Summary

	adaptController	Actualizar el Controlador Neuronal
	adaptModel	Actualizar el Modelo Neuronal
	addlistener	Add listener for event. Help for CANNController/addlistener is inherited from superclass HANDLE
	computeReferenceModel	Simular el modelo de referencia desde t(k) hasta t(k)+Tiempo_muestreo
	delete	Destructor
Static	der_f	Cacula los valores de las derivadas de las funciones de activacion de las neuronas
	eq	== (EQ) Test handle equality. Help for CANNController/eq is inherited from superclass HANDLE
Static	evalFunction	Cacula los valores de las funciones de activacion de las neuronas
	findobj	Find objects matching specified conditions. Help for CANNController/findobj is inherited from superclass HANDLE
	findprop	Find property of MATLAB handle object. Help for CANNController/findprop is inherited from superclass HANDLE
	ge	>= (GE) Greater than or equal relation for handles. Help for CANNController/ge is inherited from superclass HANDLE
	getControllerBiases	Leer valores de los biases del Controlador Neuronal
	getControllerWeights	Leer valores de los pesos del Controlador Neuronal
	getModelBiases	Leer valores de los biases del Modelo Neuronal
	getModelWeights	Leer valores de los pesos del Modelo Neuronal
	gt	> (GT) Greater than relation for handles. Help for CANNController/gt is inherited from superclass HANDLE
Sealed	isvalid	Test handle validity. Help for CANNController/isvalid is inherited from superclass HANDLE
	le	<= (LE) Less than or equal relation for handles. Help for CANNController/le is inherited from superclass HANDLE
	listener	Add listener for event without binding the listener to the source object. Help for CANNController/listener is inherited from superclass HANDLE
	lt	< (LT) Less than relation for handles. Help for CANNController/lt is inherited from superclass HANDLE
	ne	~= (NE) Not equal relation for handles. Help for CANNController/ne is inherited from superclass HANDLE
	notify	Notify listeners of event. Help for CANNController/notify is inherited from superclass HANDLE
	predictController	calcular prediccion Controlador Neuronal
	predictModel	calcular prediccion Model Neuronal
	resetInitialConditions	Resetear las condiciones iniciales del modelo de referencia
	setControlLambdaL2Regu	Fijar de regularizacion (Lambda) para el Controlador Neuronal. Weight Decay. No implementado
	setControllerLearningRate	Fijar el valor de Learning Rate para el Controlador Neuronal
	setControllerMomemtum	Fijar cantidad de momento (Beta) para la actualizacion del Controlador Neuronal
	setControllerNet	Fijar la configuracion de la red neuronal del controlador
	setControllerWeightsBiases	Fijar valores de pesos y biases del Controlador Neuronal
	setModelLambdaL2Regu	Fijar de regularizacion (Lambda) para el Modelo Neuronal. Weight Decay. No implementado
	setModelLearningRate	Fijar el valor de Learning Rate para el Modelo Neuronal
	setModelMomemtum	Fijar cantidad de momento (Beta) para la actualizacion del Modelo Neuronal
	setModelNet	Fijar la configuracion de la red neuronal del modelo
	setModelWeightsBiases	Fijar valores de pesos y biases del Modelo Neuronal
	setReferenceModel	Fijar los parametros del modelo de referencia
	setdEduPossitions	Fijar posiciones de las acciones de control dentro de la tupla de entrada para el Modelo Neuronal

Figura C.2: Objeto CANNController

CLocalizacionAGV

Constructor

Class Details

Superclasses	handle
Sealed	false
Construct on load	false

Constructor Summary

CLocalizacionAGV	Constructor

Method Summary

	addlistener	Add listener for event. Help for CLocalizacionAGV/addlistener is inherited from superclass HANDLE
	calcSpatialLocalization	Localizacion espacial de los puntos de interes del cuerpo del AGV
	delete	Destructor
	eq	== (EQ) Test handle equality. Help for CLocalizacionAGV/eq is inherited from superclass HANDLE
	findobj	Find objects matching specified conditions. Help for CLocalizacionAGV/findobj is inherited from superclass HANDLE
	findprop	Find property of MATLAB handle object. Help for CLocalizacionAGV/findprop is inherited from superclass HANDLE
	ge	>= (GE) Greater than or equal relation for handles. Help for CLocalizacionAGV/ge is inherited from superclass HANDLE
	gt	> (GT) Greater than relation for handles. Help for CLocalizacionAGV/gt is inherited from superclass HANDLE
Sealed	isvalid	Test handle validity. Help for CLocalizacionAGV/isvalid is inherited from superclass HANDLE
	le	<= (LE) Less than or equal relation for handles. Help for CLocalizacionAGV/le is inherited from superclass HANDLE
	listener	Add listener for event without binding the listener to the source object. Help for CLocalizacionAGV/listener is inherited from superclass HANDLE
	lt	< (LT) Less than relation for handles. Help for CLocalizacionAGV/lt is inherited from superclass HANDLE
	ne	~= (NE) Not equal relation for handles. Help for CLocalizacionAGV/ne is inherited from superclass HANDLE
	notify	Notify listeners of event. Help for CLocalizacionAGV/notify is inherited from superclass HANDLE
	setParameters	Fijar los parametros del AGV:

Event Summary

ObjectBeingDestroyed	Notifies listeners that a particular object has been destroyed. Help for CLocalizacionAGV/ObjectBeingDestroyed is inherited from superclass HANDLE

Figura C.3: Objeto CLocalizacionAGV

CLocalizacionDiferencial

Constructor

Class Details

Superclasses	handle
Sealed	false
Construct on load	false

Constructor Summary

CLocalizacionDiferencial	Constructor

Method Summary

	addlistener	Add listener for event. Help for CLocalizacionDiferencial/addlistener is inherited from superclass HANDLE
	calcSpatialLocalization	Calcula la localizacion espacial de los puntos de interes de la unidad de traccion
	delete	Destructor
	eq	== (EQ) Test handle equality. Help for CLocalizacionDiferencial/eq is inherited from superclass HANDLE
	findobj	Find objects matching specified conditions. Help for CLocalizacionDiferencial/findobj is inherited from superclass HANDLE
	findprop	Find property of MATLAB handle object. Help for CLocalizacionDiferencial/findprop is inherited from superclass HANDLE
	ge	>= (GE) Greater than or equal relation for handles. Help for CLocalizacionDiferencial/ge is inherited from superclass HANDLE
	gt	> (GT) Greater than relation for handles. Help for CLocalizacionDiferencial/gt is inherited from superclass HANDLE
Sealed	isvalid	Test handle validity. Help for CLocalizacionDiferencial/isvalid is inherited from superclass HANDLE
	le	<= (LE) Less than or equal relation for handles. Help for CLocalizacionDiferencial/le is inherited from superclass HANDLE
	listener	Add listener for event without binding the listener to the source object. Help for CLocalizacionDiferencial/listener is inherited from superclass HANDLE
	lt	< (LT) Less than relation for handles. Help for CLocalizacionDiferencial/lt is inherited from superclass HANDLE
	ne	~= (NE) Not equal relation for handles. Help for CLocalizacionDiferencial/ne is inherited from superclass HANDLE
	notify	Notify listeners of event. Help for CLocalizacionDiferencial/notify is inherited from superclass HANDLE
	setParameters	Fijar parametros del AGV

Event Summary

ObjectBeingDestroyed	Notifies listeners that a particular object has been destroyed. Help for CLocalizacionDiferencial/ObjectBeingDestroyed is inherited from superclass HANDLE

Figura C.4: Objeto CLocalizacionDiferencial

CMagneticSensor

Constructor

Class Details

Superclasses	handle
Sealed	false
Construct on load	false

Constructor Summary

CMagneticSensor	Constructor

Method Summary

	addlistener	Add listener for event. Help for CMagneticSensor/addlistener is inherited from superclass HANDLE
	delete	Destructor
	eq	== (EQ) Test handle equality. Help for CMagneticSensor/eq is inherited from superclass HANDLE
	findobj	Find objects matching specified conditions. Help for CMagneticSensor/findobj is inherited from superclass HANDLE
	findprop	Find property of MATLAB handle object. Help for CMagneticSensor/findprop is inherited from superclass HANDLE
	ge	>= (GE) Greater than or equal relation for handles. Help for CMagneticSensor/ge is inherited from superclass HANDLE
	getMeasurement	Obtener lectura del sensor. Distancia del centro del al punto de corte con la trayectoria
	getSearchRange	Leer rango de espacio de busqueda
	getSensorRecEquation	Leer ecuacion de la recta del sensor
	gt	> (GT) Greater than relation for handles. Help for CMagneticSensor/gt is inherited from superclass HANDLE
Sealed	isvalid	Test handle validity. Help for CMagneticSensor/isvalid is inherited from superclass HANDLE
	le	<= (LE) Less than or equal relation for handles. Help for CMagneticSensor/le is inherited from superclass HANDLE
	listener	Add listener for event without binding the listener to the source object. Help for CMagneticSensor/listener is inherited from superclass HANDLE
	lt	< (LT) Less than relation for handles. Help for CMagneticSensor/lt is inherited from superclass HANDLE
	ne	~= (NE) Not equal relation for handles. Help for CMagneticSensor/ne is inherited from superclass HANDLE
	notify	Notify listeners of event. Help for CMagneticSensor/notify is inherited from superclass HANDLE
	setSearchRange	Rango para el espacio de busqueda alrededor del punto actual
	setSensorLength	Fijar longitud del sensor
	setTrajectoryEquation	Fijar ecuacion de la trayectoria

Event Summary

ObjectBeingDestroyed	Notifies listeners that a particular object has been destroyed. Help for CMagneticSensor/ObjectBeingDestroyed is inherited from superclass HANDLE

Figura C.5: Objeto CMagneticSensor

CMotorRueda

Constructor

Class Details

Superclasses	handle
Sealed	false
Construct on load	false

Constructor Summary

CMotorRueda	Constructor

Method Summary

	addlistener	Add listener for event. Help for CMotorRueda/addlistener is inherited from superclass HANDLE
	applyDynamicModel	Simulacion del sistema dinamico desde t(k) hasta t(k)+Delta_t
	delete	Destructor
	eq	== (EQ) Test handle equality. Help for CMotorRueda/eq is inherited from superclass HANDLE
	findobj	Find objects matching specified conditions. Help for CMotorRueda/findobj is inherited from superclass HANDLE
	findprop	Find property of MATLAB handle object. Help for CMotorRueda/findprop is inherited from superclass HANDLE
	ge	>= (GE) Greater than or equal relation for handles. Help for CMotorRueda/ge is inherited from superclass HANDLE
	gt	> (GT) Greater than relation for handles. Help for CMotorRueda/gt is inherited from superclass HANDLE
Sealed	isvalid	Test handle validity. Help for CMotorRueda/isvalid is inherited from superclass HANDLE
	le	<= (LE) Less than or equal relation for handles. Help for CMotorRueda/le is inherited from superclass HANDLE
	listener	Add listener for event without binding the listener to the source object. Help for CMotorRueda/listener is inherited from superclass HANDLE
	lt	< (LT) Less than relation for handles. Help for CMotorRueda/lt is inherited from superclass HANDLE
	ne	~= (NE) Not equal relation for handles. Help for CMotorRueda/ne is inherited from superclass HANDLE
	notify	Notify listeners of event. Help for CMotorRueda/notify is inherited from superclass HANDLE
	resetInitialConditions	Resetear condiciones iniciales a cero
	setInitialConditions	Fijar condiciones iniciales
	setLoadMass	Fijar carga soportada por la rueda

Event Summary

ObjectBeingDestroyed	Notifies listeners that a particular object has been destroyed. Help for CMotorRueda/ObjectBeingDestroyed is inherited from superclass HANDLE

Figura C.6: Objeto CMotorRueda

CPIController

```
CPIController is a class.
    this = CPIController
```

Class Details

Superclasses	handle
Sealed	false
Construct on load	false

Constructor Summary

CPIController	

Method Summary

	addlistener	Add listener for event. Help for CPIController/addlistener is inherited from superclass HANDLE
	calcControlAction	Calcular accion de control dadas las ganancias Kp y Ki
	delete	Delete a handle object. Help for CPIController/delete is inherited from superclass HANDLE
	eq	== (EQ) Test handle equality. Help for CPIController/eq is inherited from superclass HANDLE
	findobj	Find objects matching specified conditions. Help for CPIController/findobj is inherited from superclass HANDLE
	findprop	Find property of MATLAB handle object. Help for CPIController/findprop is inherited from superclass HANDLE
	ge	>= (GE) Greater than or equal relation for handles. Help for CPIController/ge is inherited from superclass HANDLE
	gt	> (GT) Greater than relation for handles. Help for CPIController/gt is inherited from superclass HANDLE
Sealed	isvalid	Test handle validity. Help for CPIController/isvalid is inherited from superclass HANDLE
	le	<= (LE) Less than or equal relation for handles. Help for CPIController/le is inherited from superclass HANDLE
	listener	Add listener for event without binding the listener to the source object. Help for CPIController/listener is inherited from superclass HANDLE
	lt	< (LT) Less than relation for handles. Help for CPIController/lt is inherited from superclass HANDLE
	ne	~= (NE) Not equal relation for handles. Help for CPIController/ne is inherited from superclass HANDLE
	notify	Notify listeners of event. Help for CPIController/notify is inherited from superclass HANDLE
	resetInitialConditions	Resetear condiciones iniciales a cero (integral del error)
	setSampleTime	Fijar tiempo de muestreo del controlador PI

Event Summary

ObjectBeingDestroyed	Notifies listeners that a particular object has been destroyed. Help for CPIController/ObjectBeingDestroyed is inherited from superclass HANDLE

Figura C.7: Objeto CPIController

CPIDController

`CPIDController is a class.`
`this = CPIDController`

Class Details

Superclasses	handle
Sealed	false
Construct on load	false

Constructor Summary

CPIDController	

Method Summary

	addlistener	Add listener for event. Help for CPIDController/addlistener is inherited from superclass HANDLE
	calcControlAction	Calcular accion de control dadas las ganancias Kp, Kd y Ki
	delete	Delete a handle object. Help for CPIDController/delete is inherited from superclass HANDLE
	eq	== (EQ) Test handle equality. Help for CPIDController/eq is inherited from superclass HANDLE
	findobj	Find objects matching specified conditions. Help for CPIDController/findobj is inherited from superclass HANDLE
	findprop	Find property of MATLAB handle object. Help for CPIDController/findprop is inherited from superclass HANDLE
	ge	>= (GE) Greater than or equal relation for handles. Help for CPIDController/ge is inherited from superclass HANDLE
	gt	> (GT) Greater than relation for handles. Help for CPIDController/gt is inherited from superclass HANDLE
Sealed	isvalid	Test handle validity. Help for CPIDController/isvalid is inherited from superclass HANDLE
	le	<= (LE) Less than or equal relation for handles. Help for CPIDController/le is inherited from superclass HANDLE
	listener	Add listener for event without binding the listener to the source object. Help for CPIDController/listener is inherited from superclass HANDLE
	lt	< (LT) Less than relation for handles. Help for CPIDController/lt is inherited from superclass HANDLE
	ne	~= (NE) Not equal relation for handles. Help for CPIDController/ne is inherited from superclass HANDLE
	notify	Notify listeners of event. Help for CPIDController/notify is inherited from superclass HANDLE
	resetInitialConditions	Resetar condiciones iniciales (integral del error)
	setSampleTime	Fijar tiempo de muestreo del controlador PID:

Event Summary

ObjectBeingDestroyed	Notifies listeners that a particular object has been destroyed. Help for CPIDController/ObjectBeingDestroyed is inherited from superclass HANDLE

Figura C.8: Objeto CPIDController

CReferenceModel

`Constructor`

Class Details

Superclasses	handle
Sealed	false
Construct on load	false

Constructor Summary

CReferenceModel	Constructor

Method Summary

	addlistener	Add listener for event. Help for CReferenceModel/addlistener is inherited from superclass HANDLE
	applySystem	Simulacion del sistema de referencia desde t(k) a t(k)+Delta_t
	delete	Delete a handle object. Help for CReferenceModel/delete is inherited from superclass HANDLE
	eq	== (EQ) Test handle equality. Help for CReferenceModel/eq is inherited from superclass HANDLE
	findobj	Find objects matching specified conditions. Help for CReferenceModel/findobj is inherited from superclass HANDLE
	findprop	Find property of MATLAB handle object. Help for CReferenceModel/findprop is inherited from superclass HANDLE
	ge	>= (GE) Greater than or equal relation for handles. Help for CReferenceModel/ge is inherited from superclass HANDLE
	gt	> (GT) Greater than relation for handles. Help for CReferenceModel/gt is inherited from superclass HANDLE
Sealed	isvalid	Test handle validity. Help for CReferenceModel/isvalid is inherited from superclass HANDLE
	le	<= (LE) Less than or equal relation for handles. Help for CReferenceModel/le is inherited from superclass HANDLE
	listener	Add listener for event without binding the listener to the source object. Help for CReferenceModel/listener is inherited from superclass HANDLE
	lt	< (LT) Less than relation for handles. Help for CReferenceModel/lt is inherited from superclass HANDLE
	ne	~= (NE) Not equal relation for handles. Help for CReferenceModel/ne is inherited from superclass HANDLE
	notify	Notify listeners of event. Help for CReferenceModel/notify is inherited from superclass HANDLE
	resetInitialConditions	Resetear condiciones iniciales a cero.
	setControllerGains	Fijar las ganancias Kp y Ki de los controladores PI
	setMass	Fijar el peso de la carga sobre las ruedas

Event Summary

ObjectBeingDestroyed	Notifies listeners that a particular object has been destroyed. Help for CReferenceModel/ObjectBeingDestroyed is inherited from superclass HANDLE

Figura C.9: Objeto CReferenceModel