Universidad Miguel Hernández de Elche

MASTER UNIVERSITARIO EN ROBÓTICA

"Estudio de redes neuronales para estimar el modo de ensamblado de robots paralelos"

Trabajo de Fin de Máster

Curso 22/23

Autor: Antonio Manuel García Pérez
Tutor: Adrian Peidro Vidal

AGRADECIMIENTOS

En la era actual la robótica es una disciplina puntera y esencial que ha renovado totalmente nuestra manera de interactuar con el mundo o el entorno que nos rodea. Este Trabajo de Fin de Máster (TFM) es el resultado de un apasionante proceso de investigación, en el que diversas personas han aportado su granito de arena haciendo que todo esto sea posible.

En primer lugar, me gustaría mostrar mi agradecimiento a mi tutor durante este proyecto, Adrián Peidro Vidal, para el cual tengo la suerte también de trabajar en el "Laboratorio de robots y mecanismos paralelos" desarrollando un proyecto de investigación relacionado con este mismo Trabajo de Fin de Máster. Sin él, su implicación y dedicación, nada de esto hubiera sido posible.

No puede ser pasado por alto la presencia incondicional brindada por mi familia y seres queridos a lo largo de todo mi recorrido académico, sin su apoyo emocional nada de esto hubiera sido viable, gracias de corazón a los que quedamos y a los que ya no están.

Fuera de los agradecimientos individuales me gustaría agradecer a la Universidad Miguel Hernández la oportunidad de formarme y aprender cada día, y a la fundación ValgrAI por las ayudas que brinda a los estudiantes del Máster en Robótica de la UMH.

Sin todos y cada uno de los mencionados eslabones no hubiera sido posible avanzar hacia la consecución del presente proyecto. Con la mayor de las gratitudes y el entusiasmo propio de la ocasión presento el aquí documentado TFM.

Índice

Índice de figuras

1. Introducción

1.1. Situación general

En la actualidad los sistemas robóticos han concentrado una gran importancia, estos sistemas no solo permiten realizar tareas que para el humano por si solo le resultarían imposibles, sino que también permiten mejorar el desempeño de los humanos en la realización de ciertas tareas históricamente realizadas por ellos en solitario.

Al hablar de robótica se tiende a pensar en los robots más tradicionales y como consecuencia en las aplicaciones clásicas de estos, pero desde hace ya bastantes años la robótica viene rompiendo barreras, siendo su rango de influencia prácticamente ilimitado.

La robótica ya no solo busca optimizar ciertos procesos industriales ya existentes sino que también pretende romper con los límites conocidos y establecidos por los humanos hasta la fecha.

En lo que a robótica industrial refiere los robots han aumentado su nivel de sofisticación, precisión y seguridad, siendo mucho más útiles y menos peligrosos que sus antecesores.

En el ámbito de la medicina los robots ya no solo son robots destinados al transporte de material o la asistencia básica al personal, sino que hoy en día los robots forman parte activa del equipo o incluso podría decirse, salvando las distancias, que forman parte del personal médico. Esto se debe a que existen por ejemplo equipos robóticos destinados a la rehabilitación de pacientes, como los exoesqueletos que hacen que el personal especializado tenga que realizar menores esfuerzos físicos, centrándose únicamente en la supervisión y el asesoramiento al paciente durante el proceso de rehabilitación. Esto permite que el número de empleados implicados o requeridos en el proceso sea menor al que sería en ausencia de estos equipos citeref1.

En los últimos tiempos ya no solo se ha presenciado una disrupción relacionada con la evolución mecánica o tecnológica de los equipos, sino que esta también va ligada a la forma en la que la humanidad percibe las posibles aplicaciones de estos sistemas.

En el pasado la robótica no tenía relación alguna con la educación o el entretenimiento y a día de hoy esta relacuón resulta una realidad, los robots educativos permiten introducir a los más pequeños en el mundo de la tecnología o la programación, siendo este proceso lo más dinámico y entretenido posible, existe una segunda vertiente destinada únicamente al puro entretenimiento, podrían destacar en este campo los robots informativos e interactivos empleados en parques temáticos.

Hay que tener en cuenta que estas aplicaciones no serían tan potentes si no fuese por la capacidad de procesar información de los robots, en un pasado reciente esta capacidad de apariencia racional era totalmente debida a una correcta programación que aunque eficiente era limitada. En los últimos años se ha empezado a hablar del concepto inteligencia artificial y a pesar de ser este cada vez más conocido no todo el mundo tiene claro su verdadero significado.

¿Qué es realmente la inteligencia artificial? En función de la fuente podemos encontrar definiciones de todo tipo, pero todas ellas se podrían resumir o sintetizar en la siguiente: "Es la disciplina que mediante el uso de algoritmos logra dotar de un conjunto de capacidades cognitivas a una unidad, posibilitando que esta mediante un proceso de aprendizaje o entrenamiento obtenga una inteligencia similar a la humana" [24].

Con esta novedosa disciplina se consigue dotar en cierta medida a los robots de lo único que carecen, capacidad racional. Por lo que sus anteriores limitaciones podrían llegar a desaparecer si esta inteligencia artificial se sigue desarrollando al ritmo actual.

1.2. Objetivos generales

Los objetivos del presente proyecto consisten en evolucionar y mejorar los métodos de observabilidad empleados en robots de tipo paralelo, venciendo ciertas problemáticas no resueltas y optimizando los sistemas robóticos en lo que al uso de sensores refiere.

Hasta la fecha existen ciertas dificultades que no han logrado solucionarse. Destaca entre ellas la observabilidad ligada a robots de tipo paralelo en los que existe cierta ambigüedad posicional, siendo hasta la fecha la única solución el empleo de numerosos sensores o el conocimiento previo de ciertos datos. La citada problemática consiste en que para un único estado posicional de las articulaciones activas existen diversas posiciones del robot compatibles.

En el presente trabajo plantearemos un método que careciendo de sensores externos al propio robot será capaz de ayudarnos a deducir cual es la configuración posicional que el robot presenta. Concretamente el método consistirá en ejecutar ciertas trayectorias aleatorias y registrar ciertas variables ligadas a su seguimiento. Concretamente han de registrarse las fuerzas aplicadas por las articulaciones activas (dependerán del robot), siendo estas las causantes del seguimiento de la trayectoria, como consecuencia de estas fuerzas se producirán ciertos desplazamientos de los cuales registraremos los asociados a las articulaciones activas (podrán ser desplazamientos lineales o angulares según el robot).

Ya registrados dichos esfuerzos y variaciones posicionales habrá de generarse una red neuronal que mediante la introducción de estos datos ligados a cada trayectoria y de forma adicional la posición de partida del efector final sea capaz de descifrar la relación existente para ese robot entre las variables mencionadas y la posición de partida aportada de forma adicional.

Con esto lo que se pretende es que para ese robot en el que se han lanzado las trayectorias, facilitándole a la red las variables asociadas a dicho seguimiento y de forma adicional para los casos usados como entrenamiento o aprendizaje la posición de partida del efector final, pueda conocerse la relación entre el punto de partida y las variables generadas durante el seguimiento. Pudiendo así deducir la posición de partida con tan solo la ejecución de una trayectoria aleatoria.

Salvando las distancias podemos asemejar la red a una ecuación, en la que deduciendo ciertos parámetros se consigue relacionar las variables A (variables ligadas a la trayectoria) y B (posición de partida del efector final). Conociendo ciertos ejemplos de A ligados a ciertas B podrían deducirse los parámetros de la ecuación que nos permitan conocer B para todos y cada uno de los casos en los que se conozca A.

Para ello se van a emplear tecnologías novedosas como la inteligencia artificial y más concretamente la ligada a las redes neuronales como se mencionaba anteriormente. A grandes rasgos los objetivos serían los mostrados a continuación:

- Generar modelos robóticos: Se pretende generar en Matlab el modelo de comportamiento de ciertos robots de interés.

- Generación de datasets: Para poder generar los datos de interés se lanzarán los modelos haciéndolos seguir ciertas trayectorias aleatorias, durante el seguimiento de las trayectorias se irán registrando ciertas variables de interés, prestando especial atención a la posición de partida del efector final.

- Diseño de redes neuronales: Construir diferentes modelos, empleando en cada uno de ellos ciertas arquitecturas de redes neuronales. Prestando especial atención al tratamiento de la información de entrada y salida.

- Optimización y ajuste de redes neuronales: Optimizar los parámetros de la red en busca de unos

resultados lo más próximos posibles a los esperados.

- Evaluación de resultados: Estudiar los niveles de acierto en los resultados y tratar de justificar el porqué de los mismos en función de cada caso.
- Planteamiento futuro: Plantear posibles mejoras del actual estudio o propuestas para futuros estudios.

1.3. Estructura del documento

El presente Trabajo de Fin de Master (TFM) se organiza en distintos apartados que tratan de proporcionar una visión integral del proceso seguido a la hora de generar, optimizar y evaluar un sistema novedoso de gran utilidad ante las problemáticas de observabilidad dados en robots paralelos. A continuación se detalla la estructura del presente trabajo:

1.Introducción
En este primer apartado se introducirá el proyecto, explicando el contexto general en el que este será desarrollado y los objetivos que se pretenden cumplir a grandes rasgos. A pesar de no realizar una profundización se introducirán conceptos de vital importancia para la comprensión del presente documento.

2.Antecedentes generales
En este apartado se pretende generar una descripción detallada de como y cual es el escenario sobre el que el proyecto se va a desarrollar, profundizando en tecnologías de actualidad y especialmente en aquellas estrechamente relacionadas con el presente proyecto. De igual modo que sucedía en la introducción se introducirán ciertos conceptos que resultarán importantes en futuros apartados.

3.Antecedente específicos
En este capítulo se pretende focalizar la atención sobre la problemática a enfrentar y sobre aquellas herramientas o información de la que se dispone para ello. Se prestará especial atención a las metodologías explicadas en esa documentación existente de forma previa y en los planteamientos se han ido impulsando frente a la problemática en el pasado.

4.Planteamiento resolutivo
Dejando atrás el contexto y los antecedentes se entra en materia clarificando el enfoque de la resolución propuesta e introduciendo los objetivos específicos a alcanzar durante del desarrollo del proyecto.

5.Entrenamiento y optimización
En cuanto al entrenamiento resultará clave comprender de forma general el contenido de cada etapa de entrenamiento, explicando los programas y ficheros que se pretenden emplear en cada una de ellas. Se explicarán primeramente estas de forma general y posteriormente se detallará el procedimiento seguido para cada uno de los robots, comentando los inconvenientes surgidos y cómo estos se han resuelto.

6.Conclusiones y futuros proyectos

Tras realizar el entrenamiento se analizarán los resultados del proceso y el propio proceso o procesos seguidos para alcanzar los mismos en busca de conclusiones.

El presente proyecto representa una opción novedosa de enfrentar las problemáticas ligadas a la observabilidad en robots paralelos, estando este enmarcado en el contexto de un TFM universitario tendrá un alcance limitado, conocido esto y conociendo el posible potencial de la solución se propondrán ciertas pinceladas orientativas para futuros proyectos relacionados o simplemente futuras fases de una posible continuación de este mismo proyecto.

2. Antecedentes generales

2.1. Relevancia de los sistemas robóticos

Resulta interesante estudiar por qué los sistemas robóticos han ganado tanta importancia, prestando especial atención a dos campos:

- Comenzando por el sector industrial las aplicaciones son casi ilimitadas por lo que las ventajas dependerán del caso concreto, podrían destacarse la capacidad de carga, la flexibilidad, la precisión o la velocidad.

- En lo que a la medicina y más concretamente a la cirugía concierne se tiene un abanico algo menos amplio de ventajas, ya que las aplicaciones finales son todas similares. Las ventajas más destacables vienen relacionadas con la escalabilidad, esta escalabilidad no solo puede aplicarse a los desplazamientos del efector final, como por ejemplo un bisturí que ha de desplazarse 10 veces menos de lo que lo hace el controlador o maestro, sino que también puede extrapolarse a las fuerzas con las que este efector interacciona con el entorno en base a la referencia generada por el operador.

  Aclarando esto último lo que se pretende es que el cirujano pueda operar con una precisión mucho mayor de la que dispondría si realizase la operación de forma manual o convencional, para ello se le dota de una realimentación visual rica en información y de una realimentación de fuerzas también precisa. No siempre se pretende que el cirujano sienta fielmente la fuerza real que está aplicando el efector final, sino que esta puede ser escalada tanto positiva como negativamente. Por ejemplo si el cirujano trabaja sobre tejidos blandos y ha de diferenciar cuando el corte está pasando de una capa a otra resulta conveniente que las fuerzas se le realimenten de una forma exagerada, ya que de por sí los tejidos blandos oponen una pobre resistencia al corte. De modo similar ocurre con los desplazamientos, por muy buen pulso que tenga un cirujano este siempre cometerá pequeños errores, si por el contrario emplea un sistema escalado puede hacer que sus movimientos se den reducidos en el efector final, de tal modo que el avance no sea tan rápido pero los errores de dirección durante el corte sean prácticamente despreciables.

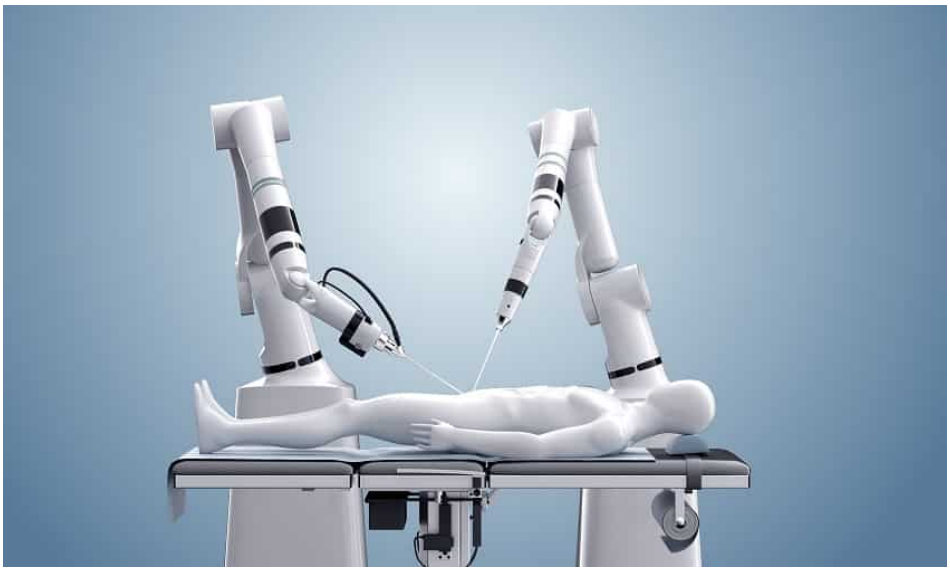

Figura 1: Robot médico empleado en cirugía [1]

Volviendo a hablar de los robots de forma genérica ha de decirse que si de por sí sus ventajas de funcionamiento o mecánicas tradicionales resultaban suficientes para colocar a los robots en una posición más que competitiva, con la incorporación de las ya citadas inteligencias de tipo artificial los robots podrían tener una presencia mucho más notable y variada en los distintos sectores.

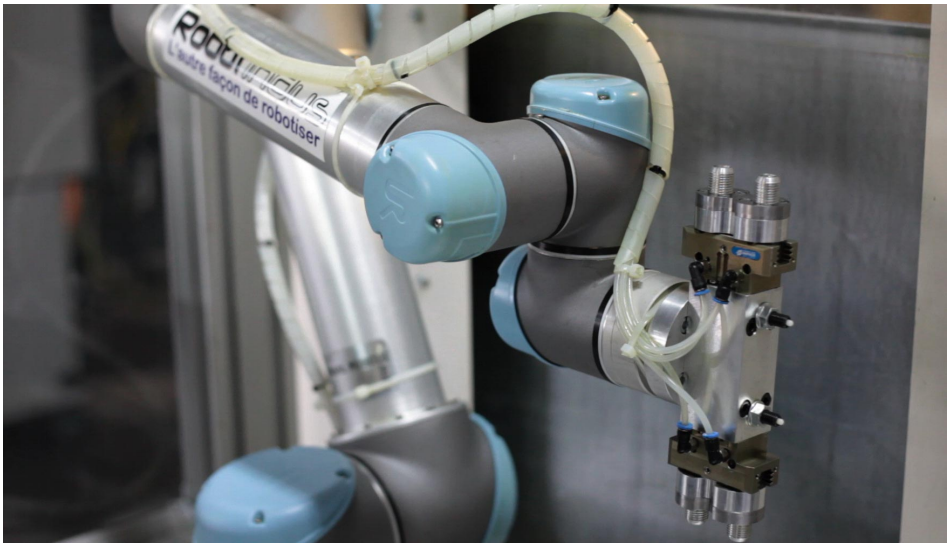

Figura 2: Robot convencional UR5 [2]

2.2. Elementos constructivos en robótica

A nivel constructivo los robots son cada vez conjuntos menos definidos, la gran variedad de tipologías de robot existentes en la actualidad dificultan generar una lista fija de componentes. Conocida la dificultad de esta tarea lo que se pretende a continuación no es generar un listado cerrado de componentes sino hablar de forma general sobre los componentes o tipos de componentes más habituales y relevantes [25].

Estos son los tipos de componentes más empleados en la actualidad:

- Estructura portante: Es la base sobre la que se ubican el resto de componentes del robot, su labor es estructural y es la encargada de soportar gran parte de los esfuerzos mecánicos, especialmente los debidos a las masas ubicadas sobre ella. En ocasiones las estructuras son fijas, pero en el caso de los robots móviles estas se desplazarán junto al robot en todo momento.

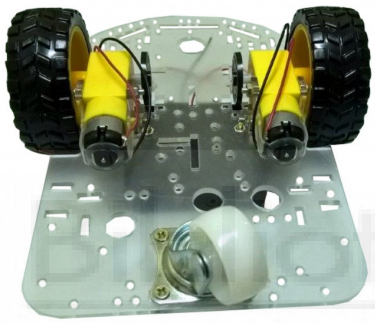

Figura 3: Chasis robot móvil [3]

- Actuadores: Son los elementos responsables de la generación de desplazamientos y/o esfuerzos. Si hablamos de actuadores se nos puede venir a la mente algo similar a lo mostrado a continuación:

Figura 4: Actuador eléctrico [4]

Pero los motores eléctricos no son los únicos actuadores empleados en la actualidad, existe una gran variedad de tipos de actuadores, como por ejemplo los pistones que generan desplazamientos y esfuerzos lineales independientemente de si estos son neumáticos, eléctricos o hidráulicos.

Los motores eléctricos se emplean comúnmente ya que permiten generar desplazamientos muy precisos y controlados. Además de esta destacable precisión estos se alimentan con un cierto voltaje y una cierta intensidad, siendo estos parámetros fácilmente controlables y los cuales pueden aportar gran información sobre que par o fuerza esta ejerciendo el motor en cada instante.

- Sensores: El set de sensores de un robot podría asemejarse a los sentidos humanos por los que nosotros logramos percibir estímulos de vital importancia del entorno. En el pasado los sets de sensores robóticos eran poco sofisticados, pero con la creación de nuevas tipologías de robots que comparten espacios de trabajo con humanos, y como consecuencia del desarrollo tecnológico, estos se han visto forzados a evolucionar. En la actualidad existen infinidad de sensores, incluso diversos tipos de sensores para un misma función que serán seleccionados según escenario o las condiciones de trabajo.

  Destacan los sensores de posición angular como los encoders, los de rango como los LIDAR, los giroscopios o acelerómetros presentes en unidades IMU, entre muchos otros...

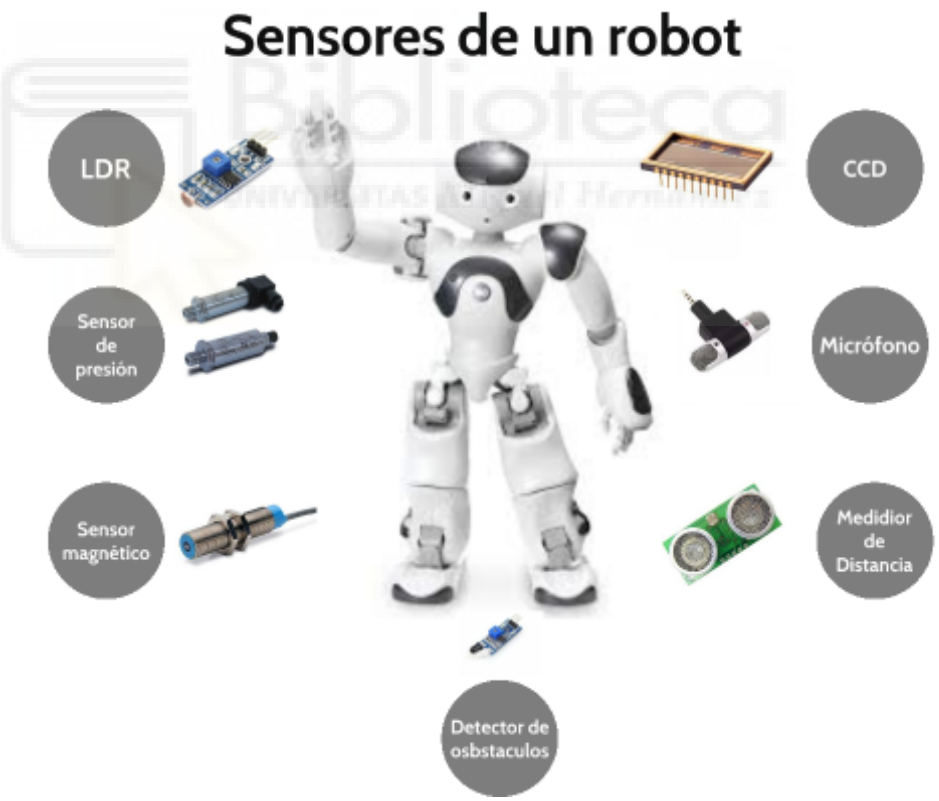

Figura 5: Sensores del robot [5]

- Unidad de control: Bien es cierto que la correcta selección del set de sensores resulta clave, pero no es lo único relevante.

  ¿Que sentido tendrían los sentidos humanos sin un cerebro que sepa interpretar la información recibida? Pues la realidad es que sin el cerebro los sentidos serían inservibles, en robótica sucede

lo mismo: resulta tan importante tener un buen set de sensores como tener una buena unidad de control o procesamiento.

La unidad de control es la encargada de interpretar la información percibida por los sensores y generar ordenes de control apropiadas, tanto para la realización de la tarea como para garantizar la seguridad del propio robot y la de su entorno.

Del mismo modo que la sensorización no sirve sin unidad de control, la unidad de control es inútil si no contiene un correcto software de control. El software de control es algo bastante sofisticado y que ha de tener en cuenta múltiples factores.

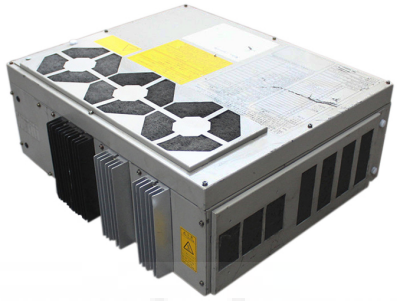

Figura 6: Unidad de control del robot [6]

- Fuente de energía o de alimentación: Los robots han de disponer de energía para poder alimentar los componentes, especialmente los actuadores, los sistemas de control y en la gran mayoría de casos los sensores (independientemente de que estos últimos sean activos o pasivos).

Figura 7: Fuente de alimentación del robot [7]

- Ruedas o sistema locomotor: En el caso de que el robot sea móvil la estructura portante no quedará fija por lo que esta habrá de disponer de ciertos elementos para poder desplazarse. En función del terreno se dispondrá de un tipo de sistema locomotor, pueden ser ruedas convencionales, cadenas similares a las de algunas excavadoras u otros tipos como por ejemplo patas.

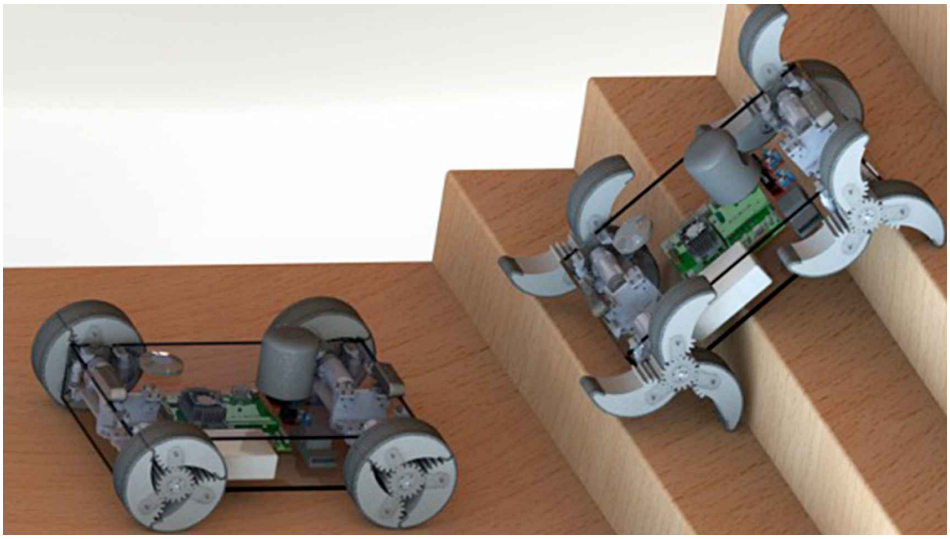

Figura 8: Sistema locomotor del robot [8]

- Mecanismo articulado: No todos los robots cuentan con eslabones al uso, en ocasiones las piezas presentan ciertas formas pero a nivel práctico no dejan de poder concebirse como eslabones, al fin y al cabo lo que se suele tener es un mecanismo en el que un conjunto de piezas se unen con articulaciones de todo tipo. Las articulaciones pueden ser esféricas, telescópicas entre muchos otros tipos que permiten al conjunto con ayuda de los actuadores realizar ciertas tareas para las que el mecanismo se ha diseñado.

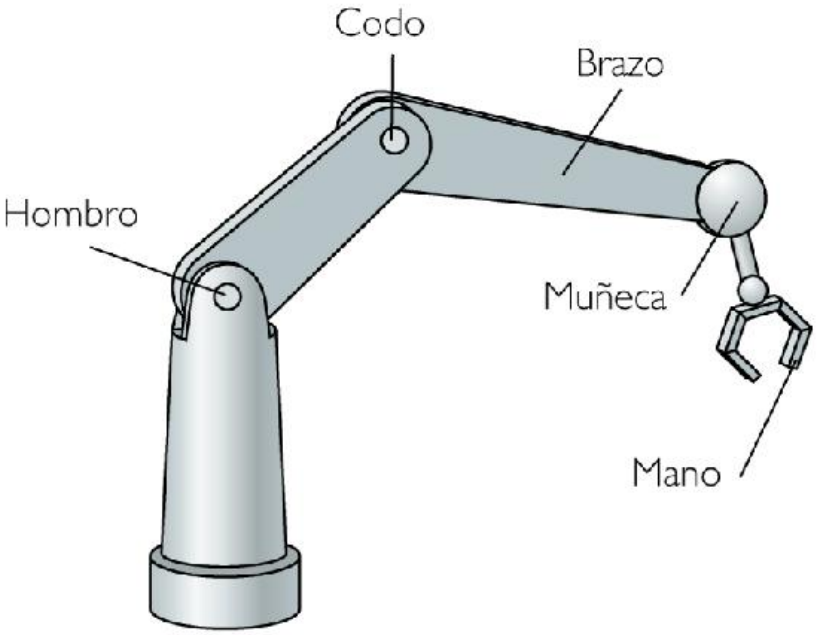

Figura 9: Esqueleto articulado del robot [9]

- Efectores finales: Por norma general al final de la cadena cinemática se ubica una herramienta, esta herramienta depende mucho de la aplicación o las aplicaciones para las que se haya diseñado el robot, bien es cierto que un mismo robot puede estar diseñado para diversas tareas requiriendo este un efector final para cada una de ellas. Años atrás las sustituciones de efectores finales se realizaban de forma manual, en la actualidad los cambios pueden realizarse de forma automatizada y rápida, incluso dentro de una cadena de montaje sería viable que un mismo robot realizase múltiples tareas consecutivas gracias a la rápida sustitución del efector final y la flexibilidad de los propios robots.

Figura 10: Efector final del robot [10]

- Sistemas de comunicación y/o supervisión: En ocasiones en el control de robots no automatizados el operador requiere de cierta información para poder controlar el robot de una forma fiable, la información requerida puede ir desde una realimentación de esfuerzos esclavo-maestro, siendo el esclavo el robot controlado y el maestro el sistema que es movido por el operador para el control, hasta información numérica como por ejemplo la temperatura de la superficie sobre la que se esta realizando un proceso de soldadura.
  Existe otro tipo de comunicación que aunque menos evidente es igual o más relevante, esta es la comunicación entre robots que en ocasiones determina el éxito en la coordinación de estos.

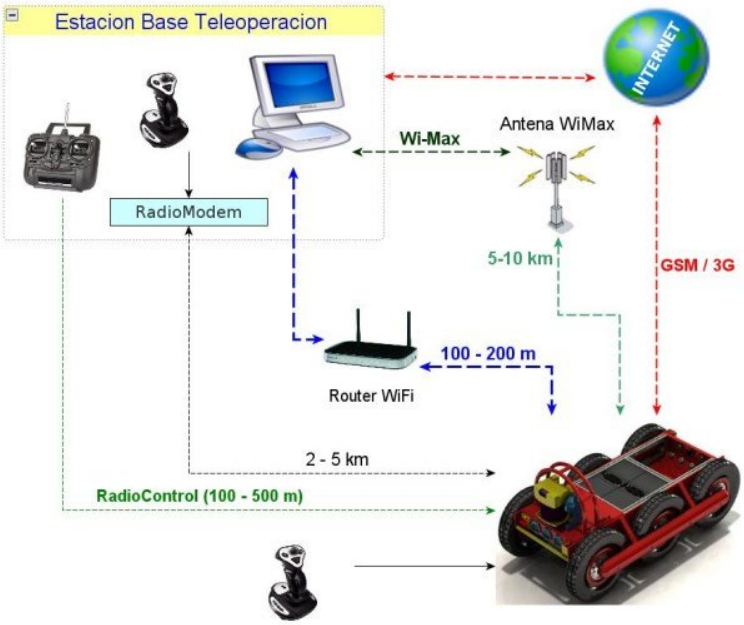

Figura 11: Sistema de coordinación [11]

- Materiales de revestimiento: Aunque no se trate de un componente demasiado evidente en ocasiones los robots se exponen a ambientes insoportables para un humano promedio, esto no sería posible sin ciertos recubrimientos protectores. Los ejemplos más típicos son presión bajo el mar, temperatura o radiación por ejemplo. Si no fuera por estos recubrimientos o revestimientos los robots no serían capaces de explorar esos entornos que para el humano resulta complejo de hacer.

  Al hablar de materiales de revestimiento o recubrimiento se nos vienen a la mente algunos textiles o similares, pero la robótica esta alcanzando límites que en el pasado e incluso en el presente resultarían o resultan difíciles de imaginar o asimilar, resultando en aplicaciones muy exigentes en las que los revestimientos se convierten en complejos sistemas multicapa. A pesar de no ser un componente al uso es un elemento clave en la robótica moderna, especialmente en aquellos entornos donde la exposición ante algun fenómeno natural o no natural resulta inevitable.

Figura 12: Sistema de comunicación del robot[12]

2.3. Tipologías constructivas en robótica

2.3.1. Definiciones

Si nos centramos estrictamente en la tipología constructiva pueden diferenciarse dos tipos de robots.En primer lugar los robots más empleados a lo largo de la historia industrial, que son los robots de tipo serie, estos robots se caracterizan por tener una única cadena cinemática. Para facilitar la comprensión podría decirse que un robot es de tipo serie si se puede llegar desde el chasis o bastidor hasta el efector final pasando por todas y cada una de las piezas o eslabones móviles.

Estos robots o brazos manipuladores en su mayoría suelen asemejarse a la fisiología humana por poseer algunos elementos comunes y por ser idénticos en cuanto a su funcionamiento.

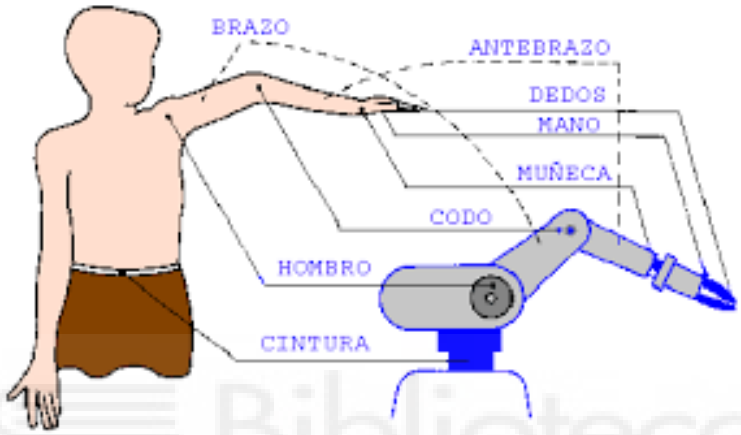

Figura 13: Robot de tipo serie [13]

En segundo lugar hablaremos de una tipología algo menos común en industria pero no por ello menos importante, los robots paralelos. En la gran mayoría de casos en industria no debe considerarse algo peor o mejor, sencillamente suelen existir diferentes versiones ideadas para distintas aplicaciones específicas por lo que dichas versiones no serán mejores ni peores sino que serán óptimas o subóptimas para según que aplicaciones. En este caso sucede lo mismo, los robots de tipo paralelo presentan ciertas ventajas e inconvenientes respecto a un robot de tipo serie y deberán ser usados en consecuencia.

De igual modo que en el caso anterior, se tratará de definir de forma simple un robot paralelo como aquel robot en el que desde el chasis o bastidor existen diversos caminos o trazados que a través de piezas móviles nos permiten alcanzar el efector final.

Figura 14: Robot de tipo paralelo [14]

2.3.2. Ventajas e inconvenientes

Como se comentaba anteriormente cada tipología constructiva presenta sus propias ventajas e inconvenientes, a continuación se muestra una imagen comparativa y empleándola como guía se procederá a hablar sobre las ventajas e inconvenientes.

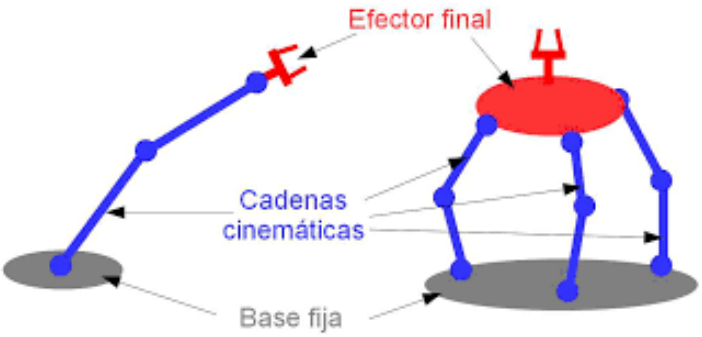

Figura 15: Comparativa robot paralelo y robot serie [15]

- Mayor rigidez: Los robots de tipo paralelo presentan una gran relación rigidez/peso esto se debe a que suelen contar con una serie de brazos que se unen a una estructura base en ciertos puntos fijos, dando lugar a una distribución o configuración paralela de elevada rigidez.
- Reparto de esfuerzos no uniforme: Los esfuerzos no se reparten tampoco de forma uniforme a lo largo de toda la cadena cinemática de un robot serie, ya que cuanto más cerca de la base existen unos momentos mayores debido a que esa zona esta soportando los esfuerzos generados por toda la cadena cinemática, en cambio, un actuador ubicado en el centro de la cadena solo deberá soportar los elementos que existen entre dicho punto y la carga portada por el efector final.

  En los robots de tipo paralelo las cadenas cinemáticas suelen ser similares o incluso idénticas por lo que los actuadores de estas percibirán esfuerzos similares durante su funcionamiento.
- Capacidad de carga: Ligado a lo anteriormente expuesto resulta importante hablar acerca de la capacidad de carga, dado que es una de las cualidades más importantes en robots de tipo industrial.

  Los robots de tipo paralelo presentan una mejor relación carga/peso, es decir, ante igualdad de peso un robot paralelo podría por norma general cargar más peso que un robot serie. Esto es debido a que en los robots paralelos suelen emplearse actuadores como elementos mecánicos prescindiendo de algunos elementos estructurales como por ejemplo ciertos eslabones o barras rígidas. Esto sumado a la mejor distribución de esfuerzos hace de los robots paralelos una buena opción en industrias dedicadas al trabajo con cargas pesadas.
- Precisión: En los robots de tipo serie las holguras presentes se van acumulando a lo largo de la cadena cinemática y del mismo modo que afecta a la estabilidad del efector final esas holguras presentes en los actuadores se traducen en ciertos errores que al darse en serie se agravan afectando a la precisión posicional del efector final.

  En los robots de tipo paralelo la afección es diferente ya que al darse en paralelo los errores no se acumulan sino que se promedian haciendo que cada cadena cinemática tienda a corregir a las otras en caso de estar estas en la posición final equivocada.
- Características dinámicas: Al presentar los robots paralelos cadenas cinemáticas cerradas y estar estas coordinadas pueden forzar movimientos más enérgicos, es decir, las aceleraciones pueden ser mayores y aún así estas pueden estar controladas. Los robots paralelos son capaces de alcanzar aceleraciones importantes sin poner en riesgo el desarrollo de la tarea.

Uno de los factores más relevantes en este aspecto es la aparición de diversos puntos de anclaje en robots paralelos, viniendo esto estrechamente ligado a la aparición de cadenas cinemáticas cerradas y con la posibilidad de imprimir mayores aceleraciones.

En este aspecto los robots de tipo serie no son capaces de generar aceleraciones tan elevadas, entre otras cosas por la inercia presente a lo largo de la cadena cinemática.

- Espacio de trabajo: Los robots paralelos venían acumulando ventajas, pero en el espacio de trabajo se encuentra una de sus principales flaquezas, independientemente de que tipo de espacio de trabajo hablemos los robots de tipo paralelo suelen presentar grandes limitaciones pudiendo alcanzar con su efector final un espacio limitado y por tanto su espacio de trabajo será por lo general más reducido.

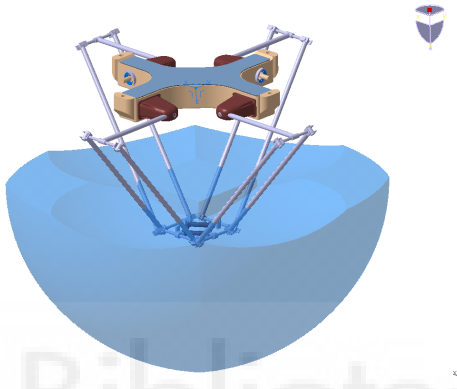

Figura 16: Espacio de trabajo robot paralelo [16]

Existen diferentes tipos de espacio de trabajo, siendo los principales: espacio de trabajo a orientación constante, espacio de trabajo de la muñeca (solo en robots serie con muñeca esférica) y espacio de trabajo alcanzable. No se entrará en detalles ya que no resulta de interés en este punto de la explicación.

Los robots de tipo serie suelen presentar espacios de trabajo algo más generosos, haciéndolos una opción interesante a la hora de flexibilizar cadenas de producción por poder generar trayectorias diferentes y amplias sin necesidad de desplazar el robot.

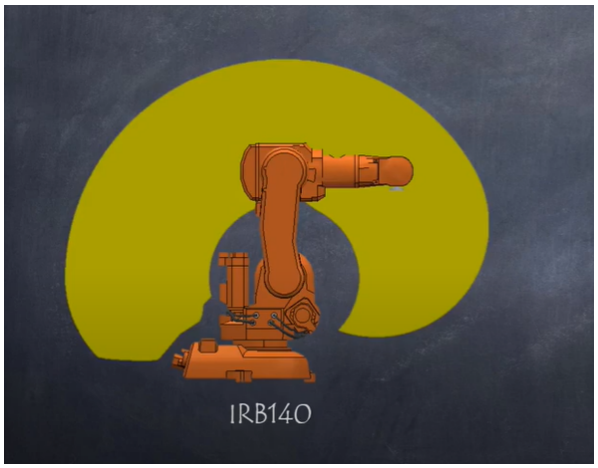

Figura 17: Espacio de trabajo robot serie [17]

Aunque la representación sea bidimensional el espacio de trabajo de este tipo de robots resulta ser esa figura revolucionada en torno al primer eje de giro del robot, este eje de giro es vertical y es fácilmente localizable.

- Singularidades: En los robots paralelos aparece un nuevo tipo de singularidad, la cual se identifica como singularidad de tipo paralelo o de tipo 2. Resultaría conveniente en primer lugar entender que es una singularidad, una singularidad es una configuración que se da cuando el robot alcanza un estado posicional que le impide realizar ciertos movimientos, aplicar ciertas velocidades o controlar un movimiento.

  En robots serie existe un único tipo de singularidad, este tipo de singularidad es conocida como singularidad de tipo serie y se da cuando el determinante de la jacobiana es nulo (la jacobiana es una matriz y permite relacionar velocidades articulares con las velocidades del efector final), que el determinante de dicha matriz sea nulo nos esta indicando que hay ciertos grados de libertad solapados (estamos perdiendo ciertas capacidades de movimiento en el efector final a causa de existir dependencias lineales dentro de la matriz jacobiana).

  Las consecuencias de las singularidades de tipo serie son:

  - 1. Algunos conjuntos de velocidades articulares no producen movimiento en el efector final, esto se debe a que existen combinaciones lineales dentro de las aportaciones de cada articulación a cada uno de los movimientos del efector final, y por ello resulta posible que los movimientos ejecutados se compensen.

  - 2. Algunas velocidades del efector final estan prohibidas, es decir en algunas direcciones se pierde el poder de generar velocidades ya que el robot ha perdido la libertad en dicha dirección por encontrarse las articulaciones en una posición que les impide empujar en la ya mencionada dirección.

  - 3. Ligada a la anterior consecuencia viene una última, esta concretamente afecta a la capacidad de reacción ante esfuerzos en el efector final justo en las direcciones en las que no resultaba posible generar velocidad, es decir perdíamos capacidad de movimiento por lo que en dichas direcciones los esfuerzos se resisten pasivamente, es decir, sin que los actuadores generen esfuerzos.

  Continuando con las singularidades y sin entrar al cálculo de estas pasaremos a hablar de robots paralelos, estos presentan ciertas diferencias constructivas de las que ya se ha hablado anteriormente y como no podía ser de otro modo estas traen consigo ciertas particularidades.

  Los robots paralelos presentan cadenas cinemáticas por lo que pueden seguir presentando singularidades de tipo serie, por poder existir ciertas dependencias, bien es cierto que las relaciones matemáticas en robots paralelos no son tan directas y esto podría dificultar la detección de este tipo de singularidades en primera instancia, ya que las configuraciones singulares ya no dependen solo de las coordenadas articulares, sino también de la posición del efector final .En el caso de darse una singularidad serie en un robot paralelo su afección sería similar a la ya explicada para robots serie [26, 27].

  Además de esta posible afección de singularidades de tipo serie entran en juego las singularidades de tipo paralelo, estas singularidades se dan ya que en robots paralelos la relación entre velocidades articulares y velocidades del efector final no resulta tan directa como en robots serie.

  La ecuación general empleada en ambos casos es la siguiente:

$$A \cdot \dot{x} + B \cdot \dot{q} = 0 \tag{1}$$

En robots de tipo serie el término o matriz A es la identidad por lo que dicho término no tiene afección alguna y desaparece, esto hace que B coincida con la jacobiana por ser la jacobiana la matriz que relaciona velocidades articulares con velocidades del efector final. Quedando la ecuación:

$$\dot{x} = J \cdot \dot{q} \tag{2}$$

En robots paralelos no se corre la misma suerte por lo que para deducir la relación entre la jacobiana y las matrices A y B resulta conveniente despejar de la primera ecuación mostrada las velocidades del efector final, quedando para el caso de la cinemática directa la siguiente ecuación:

$$\dot{x} = -A^{-1} \cdot B \cdot \dot{q} \tag{3}$$

Si nos fijamos en la ecuación general de la cinemática directa para robots serie y la comparamos con esta podemos ver que todo el término que acompaña al vector de velocidades articulares al lado derecho de la ecuación sería nuestra jacobiana en este caso. Como vemos el término que nos indicaría si el determinante de la jacobiana es nulo es B y por tanto estaríamos evaluando nuevamente singularidades de tipo serie, siendo estas dadas en robots paralelos cuando el determinante de B es nulo por implicar esto que la jacobiana también tendrá determinante nulo.

¿Entonces como habrían de ser evaluadas las singularidades de tipo paralelo?
Para ello hemos de recurrir a la cinemática inversa buscando poder así evaluar el término A de forma no inversa. La cinemática inversa no es más que el proceso por el cual podemos hallar velocidades articulares a partir de velocidades del efector final. Para ello ha de despejarse el término correspondiente a la derivada de las posiciones articulares q, en función del resto de parámetros, quedando la ecuación del siguiente modo:

$$\dot{q} = -B^{-1} \cdot A \cdot \dot{x} \tag{4}$$

Como vemos en este caso todo el término que acompaña a la derivada de x o vector de velocidades del efector final correspondería a la jacobiana inversa, bien es cierto que en un robot serie la jacobiana únicamente vendría formada por el término matricial B, por ser A en robots serie la identidad.

Como en este caso estamos evaluando un robot paralelo todo el término ha de ser tomado en cuenta siendo equivalente todo el término a la jacobiana inversa y dándose la singularidad de tipo paralelo cuando esta jacobiana inversa tenga determinante nulo, esto sucederá cuando la matriz A tenga determinante nulo.

Ya conocido el método para determinar si existe singularidad de tipo paralelo resulta importante conocer que consecuencias trae este tipo de singularidad consigo:

- 1. Aunque los actuadores se encuentren bloqueados existe movilidad del efector final en determinadas direcciones.
- 2. Ligada a la anterior consecuencia existe otra, esta segunda hace referencia a que los esfuerzos dados en esa dirección o direcciones en las que existe inestabilidad no podrán ser contrarrestados al menos en dicho instante o estado posicional.

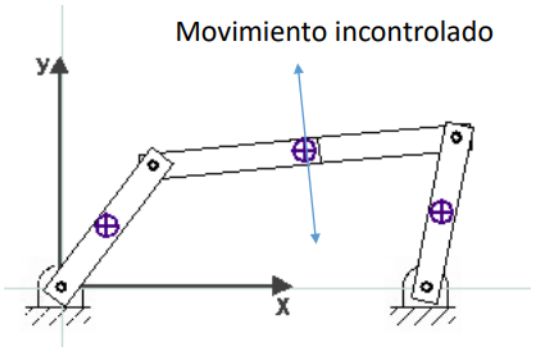

Figura 18: Singularidad de tipo paralelo

En la figura se puede apreciar que dos eslabones están quedando alineados con una articulación intermedia y por mucha tensión o tracción que existiese en esta podría darse un pequeño movimiento en la dirección perpendicular a los eslabones, este fenómeno resulta detectable a simple vista en muchos de los casos, pero resulta conveniente realizar los cálculos durante la fase de diseño para conocer el comportamiento posterior del robot.

- 3. Al depender el movimiento del efector final de varias cadenas cinemáticas, ha de tenerse en cuenta que existen ciertas combinaciones de velocidades articulares incompatibles o prohibidas, por lo que los movimientos articulares han de darse de forma coordinada y respetando ciertas relaciones.

Para concluir resulta importante resaltar que las singularidades no solo traen consigo ciertas afecciones puntuales, sino que son limitantes claros a la hora de trabajar condicionando fuertemente el espacio de trabajo. Las singularidades de tipo serie limitan el espacio de trabajo exteriormente lo que hace que el rango de posiciones alcanzables se reduzca.

Por su lado las singularidades de tipo paralelo son las responsables de que dentro del espacio de trabajo se delimiten zonas o "aspectos" con ciertas fronteras, estas fronteras hacen referencia a aquellas posiciones en las que el robot presenta singularidades de tipo paralelo y normalmente han de evitarse ya que el robot podría quedar atrapado en estas fronteras imposibilitando la consecución de la tarea o peor aún generando daños en el propio robot.

Bien es cierto que en algunos casos se cruzan por necesidad o incluso por temas de diseño pero esto segundo se da solo en casos muy específicos, como por ejemplo el diseño de robots de retorno muy rápido.

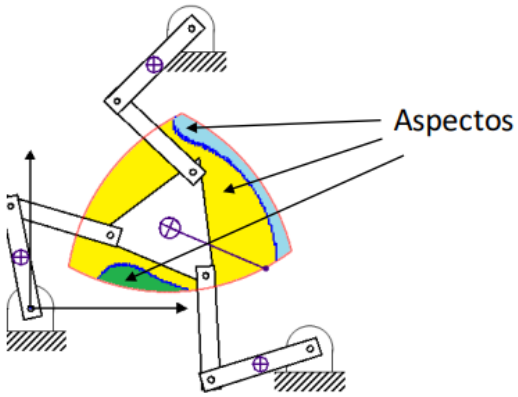

Figura 19: Espacio de trabajo y singularidades robot paralelo

- Complejidad matemática: Como ya se ha podido ir apreciando los robots paralelos suelen presentar cálculos más complejos y unas relaciones no tan intuitivas o directas, lo que hace que los cálculos puedan tornarse algo más pesados.

  Bien es cierto que la cinemática inversa presenta un menor número de soluciones para los robots de tipo paralelo, pero tanto la cinemática directa como el resto de los cálculos dinámicos se tornan más complejos en esta tipología de robots.

	Robot serie con 6 ejes		Robot paralelo
	Muñeca esférica	Muñeca no esférica, caso general	Hexápodo (plataforma de Stewart)
Cinemática directa	1 solución Método DH	1 solución Método DH	40 soluciones No hay método general
Cinemática inversa	8 soluciones	16 soluciones	1 solución

Figura 20: Número de soluciones cinemáticas según tipología

2.4. Inteligencia artificial en robótica

Durante el presente documento y a pesar de no haber sido tratada de forma directa, la inteligencia artificial ha sido mencionada en diversas ocasiones e incluso se ha definido esta brevemente, pero una tecnología tan relevante en la actualidad y con un impacto tan potente sobre la industria y robótica debe de ser atendida con un mayor detenimiento.

La inteligencia artificial existe desde hace décadas y desde sus orígenes han surgido diversas definiciones, a continuación se muestran algunas de ellas.

- John McCarthy en 1955: El objetivo de la Inteligencia Artificial (IA) es desarrollar máquinas que se comporten de forma inteligente.
- Haugeland en 1985: El esfuerzo por hacer a las computadoras pensar creando máquinas con mentes en el sentido amplio y literal.
- Charniak y McDermott en 1985: Estudio de las facultades mentales a través del estudio de modelos computacionales.
- Schalkoff en 1990: Un campo de estudio que busca explicar el comportamiento inteligente en términos de procesos computacionales.
- Rick y Knight 1991: Estudio de como hacer sistemas para la realización de tareas que, de momento, la gente hace mejor.

Formalmente podría decirse que la inteligencia artificial es el conjunto de técnicas que se aplican en el diseño de programas de computador para la resolución de problemas que por su dificultad requieren el uso de un cierto grado de inteligencia.

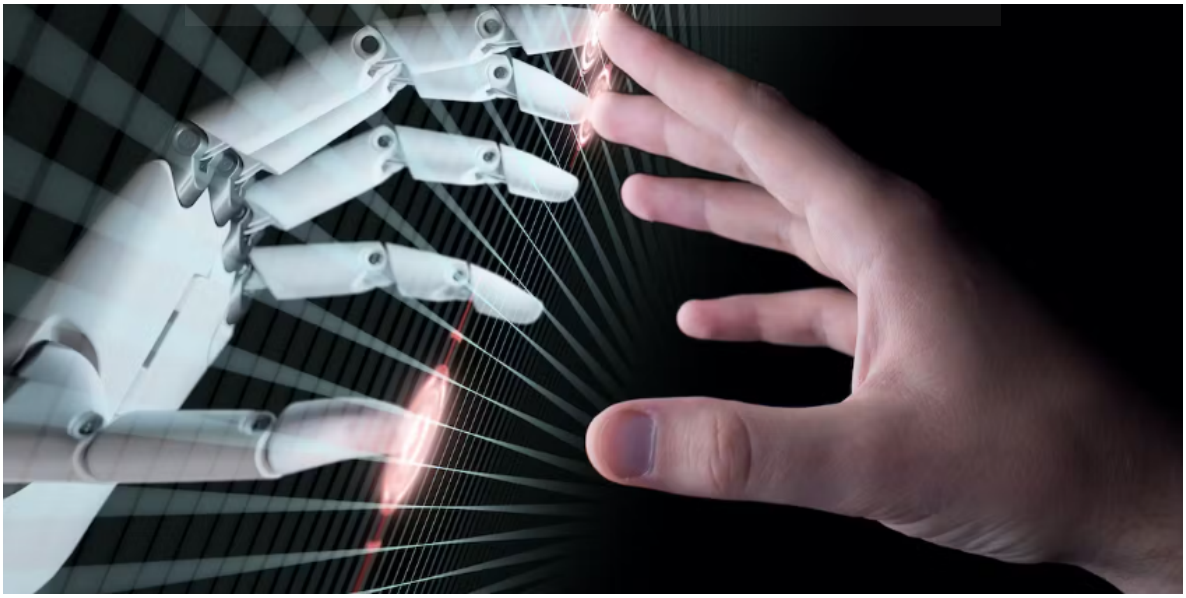

Figura 21: Inteligencia artificial como aproximación humano-máquina [18]

La inteligencia artificial es una técnica o conjunto de técnicas en pleno desarrollo por lo que día tras día nacen nuevos términos relacionados o incluidos dentro de la propia inteligencia artificial, a pesar de que por desconocimiento se tienda a pensar que la inteligencia artificial es algo muy concreto resulta complejo definir la frontera de lo que es y lo que deja de ser inteligencia artificial.

Los términos más destacados son:

- Cloud computing: El cloud computing consiste en brindar recursos computacionales a través de la red, bien es cierto que no es una técnica incluida en la inteligencia artificial pero sí estrechamente relacionada por requerir las empresas o las personas dedicadas a la inteligencia artificial en ocasiones mayores poderes de cómputo para el entrenamiento de redes neuronales.

- Data mining: Esta técnica convierte grandes cantidades de datos en información de utilidad, la dificultad reside en darle un sentido a tanta información y es ahí donde la IA gana importancia.

- Machine learning: El machine learning consiste en guiar una máquina con ejemplos hacia el aprendizaje de patrones, en este proceso los humanos suelen ayudar generando un aprendizaje de tipo supervisado, es decir, el humano le facilita a la red, por ejemplo, 300 imágenes, de las cuales 150 son coches y el resto pelotas, pues esas 300 imágenes vendrán etiquetadas sabiendo el algoritmo durante el aprendizaje qué es cada imagen y pudiendo generar así un cierto patrón que le sirva para imágenes diferentes. Esto no tiene por qué darse con imágenes, podría darse con cualquier otro tipo de dato como palabras o números.

  Los patrones a localizar por los algoritmos de machine learning suelen ser patrones que los humanos somos capaces de comprender, pero estos algoritmos nos ayudan a agilizar el proceso.

- Deep learning: El deep learning emplea redes neuronales con varias capas ocultas, las redes neuronales son programas que tratan de imitar fielmente el cerebro humano y su funcionamiento. En el pasado han existido muchos programas que han tratado de imitar el funcionamiento del cerebro humano, pero la diferencia es que las redes neuronales imitan también la arquitectura del cerebro humano empleando nodos y neuronas entre otros elementos.

  Si algo caracteriza al deep learning es que permite aprendizajes no supervisados en los que es la red neuronal la que se encarga de buscar alguna lógica o patrón aunque un humano no sea capaz de hacerlo. Por ejemplo si se tiene una gran cantidad de imágenes en las que se ven simplemente píxeles de colores aleatorios y no sabemos cómo clasificarlas, la red será capaz de hacerlo siguiendo algún tipo de criterio.

Resulta importante tener claros estos conceptos ya que no será la última vez que se hable de ellos en el presente documento, siendo el mismo una prueba más del gran potencial de la inteligencia artificial y más concretamente de las redes neuronales.

3. Antecedentes específicos

3.1. Problemática a abordar

En robótica se emplean diversos sensores para poder conocer el estado posicional, cinemático y dinámico de los robots, esto resulta clave ya que sin una información de calidad el control del robot resultaría ineficiente.

Esta adquisición de información sobre el robot o su entorno a partir de los sensores que nos sea posible colocar se conoce de forma amplia como observabilidad y es uno de los conceptos a tener en cuenta a la hora de diseñar un robot. Por norma general el término de observabilidad se reserva para aquellas ocasiones en las que la información sensorial es incompleta y han de ser deducidas ciertas variables de interés.

En ocasiones resulta suficiente con solo monitorizar el movimiento o la posición de las articulaciones activas, es decir, aquellas que poseen actuadores, ya que en algunos robots como los de tipo serie no existe ambigüedad posicional.

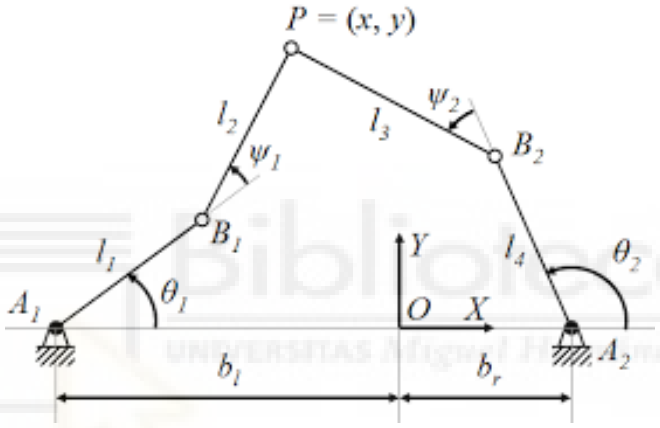

Figura 22: Robot paralelo plano [19]

Por ejemplo si en el robot plano mostrado las articulaciones activas fuesen las dos ubicadas en los apoyos y estas fuesen las únicas controladas mediante encoders podrían darse para un mismo conjunto de posiciones articulares activas dos estados posicionales diferentes.

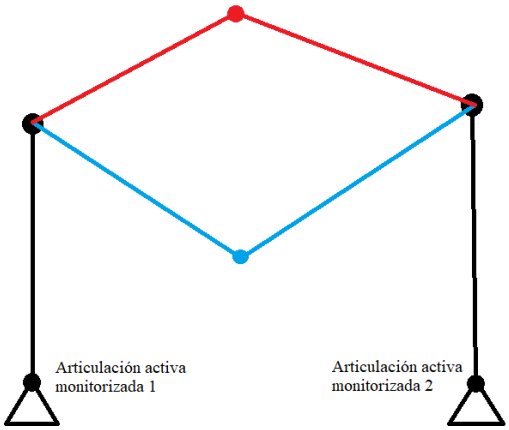

Figura 23: Ambigüedad posicional en robots paralelos

En la imagen anterior hablando de forma más concreta se puede ver representada en azul una posible combinación posicional de los dos eslabones más próximos al efector final (articulación central) para una posición de los actuadores en la que el primero de los eslabones de cada cadena cinemática se encuentra en posición vertical. Como vemos sin variar esa posición de los primeros eslabones de las cadenas cinemáticas podría darse también el estado posicional dibujado en rojo.

A priori esta problemática relacionada con la ambigüedad posicional en un instante puede parecer sencilla de resolver y así sería si se contasen con sensores en otros puntos el propio robot o simplemente sensores externos que brindasen alguna información posicional extra, y aunque añadir sensores sea una opción viable y sea esta empleada habitualmente resultaría conveniente hacerse una pregunta.

¿Es posible prescindir de sensores extra y aún así poder conocer en que estado posicional nos encontramos? Esto resultaría conveniente ya que se ahorraría bastante capital en la fabricación de robots por ser en ocasiones los sensores los elementos más costosos y pudiendo así hacer robots algo más asequibles. Por otro lado y si se estuvieran usando sensores montados sobre los elementos móviles del robot el prescindir de ellos se traduciría en una notable reducción de peso en dichas zonas y esto podría derivar en ahorro nuevamente por generar una disminución de los requerimientos o necesidades demandadas a los actuadores, pudiendo montar motores menos potentes y por tanto más económicos.

3.2. Documentación existente

3.2.1. Planteamiento

Sobre la mencionada problemática existe poca documentación, ciertas problemáticas similares se han tratado de abordar matemáticamente, a continuación se introducirán algunos de ellos.

Entre los artículos existentes cabe destacar en primer lugar un interesante artículo publicado en junio de 2020 en el cual se aborda una problemática similar, este se titula "Dynamics-Based Algorithm for Reliable Assembly Mode Tracking in Parallel Robots" y hace referencia a un proyecto de investigación llevado a cabo por Koessler, Goldsztejn, Briot y Bouton [28].

En dicho articulo se introduce la problemática haciendo referencia a algo de lo que ya se ha hablado en el presente documento, si recordamos el momento en el que hablábamos de las ventajas y desventajas de los robots serie y paralelo, concretamente en el apartado 2.3.2, existía un apartado dedicado a la complejidad matemática en el cual se introducía una tabla en la que aparecía el número de soluciones que presentaba la cinemática directa e inversa según algunos ejemplos de las dos tipologías de robot más comunes. En esta tabla se ponía un ejemplo de robot paralelo, concretamente el robot hexápodo o más comúnmente conocido como plataforma de Stewart. Este robot presenta una única solución para su cinemática inversa pero a la hora de resolver la cinemática directa conociendo las posiciones de las articulaciones activas existirían un total de 40 soluciones compatibles.

Figura 24: Plataforma de Stewart [20]

Este tipo de robot es ampliamente empleado en industria, se constituye de 6 articulaciones prismáticas o pistones siendo estos comúnmente hidráulicos, por lo que es un robot con gran capacidad de carga. Existen opciones alternativas, entre ellas destacan los actuadores lineales eléctricos.

El punto en común entre el artículo y la citada referencia del presente documento es que en el artículo se habla de modos de ensamblaje, estos modos de ensamblaje no dejan de ser las posibles soluciones o combinaciones posicionales que obtendríamos ante la resolución o el cálculo de la cinemática directa.

Podrían definirse formalmente los modos de ensamblaje como las distintas soluciones de la cinemática directa para un valor dado de las articulaciones actuadas. Por ejemplo, en el siguiente robot doble SCARA se conocen las posiciones articulares actuadas θ_1 y θ_2, para estos existen dos posibles modos de ensamblaje.

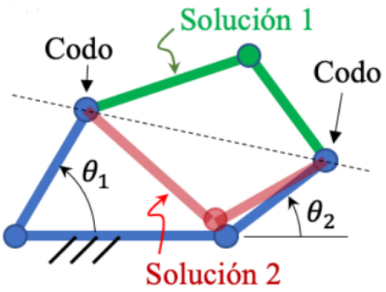

Figura 25: Explicación modos de ensamblaje [21]

Los métodos planteados hasta la fecha para determinar cual es el modo de ensamblado actual del robot (de entre los varios posibles) adoptan dos tendencias, en [29] se plantea la lectura directa de la posición del efector final mediante el uso de cámaras, no siendo esta la única opción, ya que como se indica en [30] también resulta viable medir la posición de las articulaciones pasivas.

Por otro lado tenemos la propuesta expuesta en los artículos [31, 28], plantea la necesidad de predisponer de una estimación de la posición del efector final o en su defecto conocer el modo de ensamblaje (al fin y al cabo ambas opciones aportan la misma información), tras mover el efector final se va corrigiendo la estimación de la posición del efector final, mejorando así esta hasta conocer con cierta fiabilidad la posición de dicho efector. Siendo más concisos [28] plantea un método mejorado que será detallado posteriormente.

En el artículo se mencionan algunos contratiempos a los que los métodos clásicos no logran enfrentarse de forma adecuada. A la hora de diseñar un robot puede hacerse de tal manera que el modo de ensamblaje nunca cambie, pero con esto se perdería gran parte del espacio de trabajo que se tendría si esto no fuese así.

Los métodos existentes por lo general asumen que el modo de ensamblaje no cambia y que el robot nunca se acerca a singularidades de tipo paralelo, lo cual podría ser así si se cuidan las trayectorias a realizar pero para maximizar el potencial de un robot puede diseñarse este para que sea capaz de cambiar de modo de ensamblaje de forma controlada. Si nos centramos en el robot doble SCARA de la figura anterior en este caso al pasar de un modo de ensamblaje al otro el efector final pasaría por la línea discontinua virtual y móvil que se encuentre entre ambos codos siendo justo esa posición una singularidad de tipo paralelo. Lo que pretende decirse con esto ultimo es que el cambio de modo de ensamblaje y el paso por singularidades de tipo 2 puede venir asociado en algunas tipologías concretas de robot (en determinadas tipologías denominadas "cuspidales" [32], el modo de ensamblado puede cambiarse sin atravesar sngularidades del tipo 2).

En ciertas aplicaciones el volumen o la superficie (si el robot es plano) del espacio de trabajo requerido no resulta ser demasiado, en estos casos resulta fácil poder optimizar el robot para que solo funcione en un modo de ensamblaje como el robot mostrado a continuación por ejemplo.

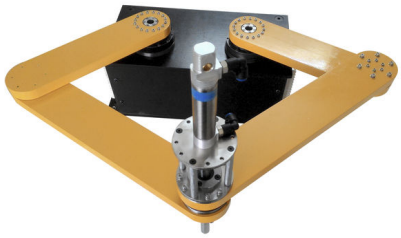

Figura 26: Robot SCARA con un único modo de ensamblaje [22]

En este caso concreto las longitudes de los eslabones y el diseño general del robot hace de cambiar de modo de ensamblaje una tarea imposible, ya que para que esto pudiese suceder los primeros eslabones de ambas cadenas cinemáticas deberían igualar en longitud de los más cercanos al efector final y si esto no fuera suficiente deberían trabajar ambos grupos de eslabones en planos los suficientemente diferenciados para que en ese cambio el efector final no colisionara con otros elementos mecánicos.

¿Exime esto al robot de la problemática expuesta haciéndolo válido para la resolución?
La realidad es que no ya que por muy pequeño que sea el espacio de trabajo este puede estar dividido en zonas, dichas zonas son denominadas aspectos y fueron explicados en el apartado 2.3.2. Los aspectos en sí no son problemáticos, la problemática esta en las fronteras entre ellos que es justamente donde se encuentran los estados posicionales considerados singularidades de tipo 2 o paralelas.

¿Resulta posible diseñar robots con un espacio de trabajo ininterrumpido?
La realidad es que sí, existe la posibilidad de crear robots libres de singularidades pero esto en ocasiones implica que la rigidez sea menor por razones del propio diseño. Si se quiere un robot lo más polivalente posible resulta mejor opción estudiar como enfrentar el paso entre aspectos o paso por singularidades de tipo dos, y los cambios de modo de ensamblaje durante el funcionamiento que diseñar robots con espacios de trabajo reducidos y libres de singularidades.

Volviendo a hablar de las dos tendencias existentes, a la hora de abordar la problemática cada una de ellas ofrece ciertas ventajas e inconvenientes que no deben ser pasadas por alto, estas se destacarán a continuación:

- Medición directa: Mediante el uso de sensores como cámaras, encoders u otros sensores se logrará deducir cual es el modo de ensamblaje actual. Con esto, la información posicional de las articulaciones activas y las medidas del robot resulta sencillo poder conocer la posición del efector final.
  - Ventajas
    - Rápido y directo: al tratarse de medidas directas resulta sencillo conocer mediante cálculos sencillos cualquier información posicional deseada del robot.
    - Fiabilidad: Si el sistema se encuentra correctamente montado y calibrado rara vez fallará, haciendo que los resultados generados por norma general sean fiables.
  - Inconvenientes
    - Sensores redundantes: En esta alternativa se plantea emplear sensores redundantes, esto quiere decir que la información brindada por estos sensores es similar o igual a la ya conocida en ausencia de estos. Bien es cierto, que sin la presencia de estos resultaría

imposible conocer el modo de ensamblaje y por tanto las informaciones conocidas serían insuficientes para poder discriminar entre las diversas posiciones compatibles del efector final.

- Coste: Si nos centramos en robots sencillos en los que las posiciones del efector final o modos de ensamblaje compatibles son escasas, resulta sencillo determinar cual es el modo de ensamblaje con un set de sensores no especialmente puntero. El problema está en que no se suele trabajar con robots sencillos sino que los robots empleados en industria u otros sectores presentan múltiples soluciones compatibles, por ello habrá de instalarse un set de sensores de alta precisión que hagan posible la consecución del marcado objetivo, es decir, deducir cual es el modo de ensamblaje actual del robot.

  El coste no solo deriva de la calidad de los sensores, sino que en ocasiones los robots paralelos presentan características difíciles de medir, ya sea porque la articulación no es del todo accesible o porque esta es de tipologías difícilmente medibles como las esféricas o universales.

- Conocimiento previo del modo de ensamblaje: Esta solución plantea partir de una medida inicial de cuál es el modo de ensamblaje del robot, y a medida que el robot se mueve se va actualizando la estimación del modo de ensamblaje del robot resolviendo así la cinemática directa. La actualización se va realizando de modo iterativo mediante el uso del método de Newton-Raphson, partiendo cada iteración de la estimación final generada en el instante anterior Esto es lo que plantean Koessler y sus colaboradores en el artículo de 2020 [28], pero como se mencionaba anteriormente ellos lo hacen incluso de forma mejorada usando una técnica llamada "aritmética de intervalos", que permite tener en cuenta incertidumbres o errores en las medidas. Además ellos consiguen que este método siga funcionando incluso al atravesar singularidades de tipo 2 (de tipo paralelo), donde los métodos anteriores al suyo fallaban y por eso evitaban tener en cuenta la posibilidad de cruzar singularidades de tipo 2.

  - Ventajas
    - Singularidades paralelas: Este método concretamente en su versión mejorada planteada en el ya mencionado artículo, presenta un buen comportamiento incluso al atravesar singularidades de tipo 2.
  - Inconvenientes
    - Requerimientos previos: El método a pesar de presentar una importante ventaja requiere de una estimación inicial para poder funcionar, siendo esta su mayor flaqueza. En ausencia de esta estimación previa resultará imposible deducir la posición del efector final o el modo de ensamblaje actual por medio de este método.

3.2.2. Enfoque propuesto en este trabajo

Dado que las alternativas existentes requieren de cierta información adicional a la generada por las articulaciones activas, se ha generado una nueva alternativa, la cual será introducida en la presente sección.

A grandes rasgos la idea consiste en emplear una red neuronal para que tras la ejecución de una trayectoria aleatoria se pueda deducir el punto desde el que esa trayectoria partió y por tanto así conocer la posición inicial del efector final. Para clarificar esto a continuación será introducido el robot que se pretende emplear para estudiar la bondad de los resultados generados por la presente alternativa, siendo este por tanto el robot con el que se experimentará. Este robot es conocido como 2RPR ya que presenta dos cadenas cinemáticas idénticas compuestas por tres articulaciones en serie de tipologías rotacional, prismática y rotacional respectivamente.

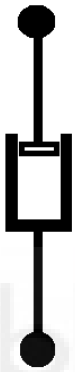

Figura 27: Cadena cinemática RPR

En cada cadena cinemática existirá una articulación activa, siendo para este robot la prismática en ambos casos. A la hora de supervisar, controlar o registrar seguimientos de trayectoria resulta imprescindible conocer la longitud de dicha articulación prismática activa.

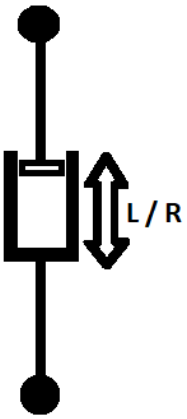

Figura 28: Variables cadena cinemática RPR

En la mostrada representación puede verse que aparece L/R esto hace referencia a que la variable de interés será llamada L o R en función de en que cadena cinemática nos encontremos, concretando

L hará referencia al pistón de la cadena cinemática izquierda (Left) y R al de la cadena cinemática de la derecha (Right).

Por simplicidad no serán L o R medidas como las longitudes de los respectivos conjuntos compuestos por pistón y cilindro ya que esto resultaría poco clarificador, L o R será la longitud de la articulación prismática incluyendo las barras de sus extremos que sirven de enlace con las articulaciones rotacionales y por tanto dichas longitudes coincidirán con la distancia entre las dos articulaciones rotacionales ubicadas a cada lado de la articulación activa o pistón.

No serán registradas solo las longitudes L y R durante el seguimiento de la trayectoria generada aleatoriamente sino que también resultarán datos clave los esfuerzos imprimidos por ambas articulaciones activas para el seguimiento de dicha trayectoria. Al tratarse de un movimiento con cierta extensión en el tiempo habrán de irse registrando los valores adoptados por L y R junto a las fuerzas aplicadas (siendo fl la denominación para la fuerza del pistón izquierdo y fr la respectiva al pistón derecho) en un gran número de instantes repartidos a lo largo de la trayectoria.

Este tipo de registro se llevará a cabo para diversas trayectorias y puntos de partida generados al azar, bien es cierto que será un azar condicionado ya que habrán zonas del espacio de trabajo que será mejor evitar debido a una problemática que será explicada próximamente. Las trayectorias serán generadas de tal forma que sean suaves y cumplan ciertos requisitos que se explicarán posteriormente.

El método se apoya principalmente en el comportamiento dinámico de los robots (especialmente en los casos en los que la gravedad es tomada en cuenta). Si tomamos un robot, como por ejemplo el robot 2RRR mostrado en la figura 25, podemos observar que en su representación aparecen dos variables, θ_1 y θ_2, siendo estas las posiciones angulares marcadas por las dos articulaciones activas de tipo rotacional presentes en dicho robot. Vemos que si se bloqueasen las articulaciones activas, e independientemente de en que ángulo se bloqueen estas, seguirían existiendo dos modos de ensamblado compatibles como los mostrados en la figura 23. Aunque pueda parecer complejo, el hecho de diferenciar el modo de ensamblaje únicamente con información relacionada con las articulaciones activas resulta viable y lograrlo es la principal pretensión del presente proyecto.

Para poder llevar a cabo esto ha de prestarse especial atención al comportamiento dinámico del robot, ya que este será el aspecto diferencial, no siendo el mismo en función de que punto se encuentre el efector final al comienzo de la ejecución de la trayectoria. Es decir, si definimos una trayectoria A y la ejecutamos desde el modo de ensamblaje en el que el efector final se encuentra en la parte alta de la representación, para unas posiciones de las articulaciones activas definidas, las fuerzas que tendrán que imprimir las articulaciones activas no serán las mismas, que sí de en su simétrico nos encontrásemos (siendo este el otro modo de ensamblaje compatible con las posiciones articulares).

Además de las fuerzas también resulta importante tener en cuenta la evolución posicional de las articulaciones activas (θ_1 y θ_2) durante el seguimiento de la trayectoria. Ya que la información dinámica por si sola no resultaría suficiente para conocer el modo de ensamblado actual del robot. A su vez la evolución posicional nos ayudará a ubicar virtualmente los eslabones de la manera adecuada pudiendo así permitir que junto a la información dinámica se pueda deducir la posición del efector final de forma relativamente precisa.

Esta idea de estimar si el robot partía de una solución u otra leyendo la evolución de las coordenadas articulares activas y de las fuerzas que los actuadores ejercen sobre dichas articulaciones, es un concepto conocido de teoría de control, llamado observabildad (determinar la configuración del robot a partir de las mencionadas variables). Para sistemas lineales el problema de observabilidad está perfectamente resuelto. Pero para sistemas no-lineales como es el caso de los robots paralelos, no existe una expresión general para estimar el estado inicial a partir de las mencionadas señales. Por ello, proponemos aproximar esa solución mediante una red neuronal. La tipología de red neuronal empleada y su configuración serán explicadas próximamente. Ensayaremos nuestra propuesta con un robot 2RPR simulado.

Toda red neuronal requiere de cierta información de entrada por lo que, ejecutadas múltiples tra-

yectorias y registrada una cantidad notable de datos (L, R ,fr y fl), estos serán introducidos a la red neuronal. En primera instancia se le introducirán casos resueltos a modo de entrenamiento, es decir se introducirá L, R, fr y fl, y por separado y para cada uno de los registros se introducirá la posición del efector final al comienzo del seguimiento de la trayectoria (una posición por registro o trayectoria).

La idea es que tras el entrenamiento y para casos en los que se desconozca la solución (casos reales) o simplemente esta no le sea facilitada a la red (ensayos de verificación), la red sea capaz de determinar desde que punto partió el efector final al inicio del seguimiento de la trayectoria, a su vez con ello se podrá conocer el modo de ensamblado actual del robot, siendo este el verdadero objetivo del método.

Aunque la estimación de la posición del efector final no fuera exacta, si esta es lo suficientemente acertada para permitir diferenciar el modo de ensamblado, resultaría sencillo calcular posteriormente con dicha información y la información posicional de las articulaciones (L y R), la posición exacta del efector final.

4. Planteamiento resolutivo

4.1. Concepto

Ante la problemática expuesta anteriormente se ha tratado de generar un nuevo enfoque resolutivo, en este se ha tratado de aprovechar al máximo los recursos disponibles.
Como se comentaba en el apartado 3.1 reducir el número de sensores presentes siempre trae consigo ventajas, especialmente si estos son sensores montados sobre el propio robot.

¿Si se prescinde de los sensores como se obtendrá la información necesaria para poder solucionar el problema de observabilidad?
Bien es cierto que la falta de sensores puede dificultar las cosas a la hora de conocer el estado posicional de un robot, pero no por ello imposibilita la tarea. Si pretendiésemos conocer el estado posicional de una forma directa resultaría difícil, pero existe una información de gran utilidad oculta en el propio funcionamiento de los actuadores.

Sean los actuadores de tipo eléctrico o de otro de los tipos explicados en el apartado 2.2 puede extraerse información a partir de la alimentación de estos. Si nos centramos por ejemplo en los actuadores eléctricos puede deducirse información sobre el par (motor giratorio) o la fuerza (motor giratorio que acaba generando una trayectoria lineal) que el actuador esta generando, a su vez en caso de ser un actuador de tipo lineal podría conocerse la longitud instantánea del eslabón y si por el contrario es un actuador giratorio podría deducirse la posición angular.

Hasta aquí todo parece sencillo, pero si el robot para esas características concretas presenta dos o más estados posicionales compatibles habría de poder discriminarse cual es el estado real.

¿Cómo podría llevarse a término esta discriminación?
Para poder discriminar entre diversos estados posicionales compatibles para un cierto instante se generará una trayectoria aleatoria a partir de dicho punto registrando la longitud o posición angular de los actuadores y sus respectivas fuerzas o pares (deducibles a partir del voltaje y la intensidad de alimentación) con estos registros a lo largo de un total de 1001 instantes que compondrán la trayectoria (pasos de la simulación) podrá deducirse en la gran mayoría de los casos en que estado posicional se encuentra el sistema.

¿Que algoritmos serán los que generen la discriminación?
En este caso se optará por el uso de las redes neuronales de las cual se ha venido hablando a lo largo del presente documento, estas redes serán capaces de deducir cual es la posición del efector final sin conocer cuales son los posibles estados compatibles, solo se tomará como entrada la longitud y fuerza o par de los actuadores a lo largo de una cierta trayectoria aleatoria.

4.2. Objetivos específicos

Para poder llevar a cabo con éxito la experimentación resulta importante definir ciertos objetivos que nos permitirán ir midiendo qué tan cerca del propósito final nos encontramos, estos objetivos para el presente proceso de investigación son los mismos que se comentaban en el apartado 1.1, pero en la presente sección se explicarán de forma desglosada y detallada todos y cada uno de ellos. A continuación se muestran dichos objetivos con su correspondiente explicación:

- 1. Generar modelos robóticos: En Matlab y más concretamente en Simulink se generarán esquemas de bloques que modelarán el comportamiento dinámico y cinemático del robot de interés. Estos bloques generarán respuestas ante ciertas entradas teniendo en cuenta la aparición o no aparición de ciertos efectos como la gravedad.

- 2. Programación de red en Python: En Python habrá de programarse una red neuronal de tal modo que tomando los datos registrados a lo largo de ciertas trayectorias simuladas como entradas sea capaz de deducir de que punto partió el efector final, siendo este el punto de interés. Resultará clave definir una arquitectura de red correcta (número de capas, neuronas por capa, tipo de red etc...) y unos parámetro internos o hiperparámetros que permitan a la red converger acertadamente. En esta fase se realizará el entrenamiento de la red pudiendo así evaluar qué tan bien funciona con cada configuración.

- 3. Fase de test: Dado que la mencionada fase de verificación se realiza discriminando un conjunto de ciertos datos del dataset (conjunto de todos los datos disponibles) e introduciendo estos como entrada sin haber sido usados en el entrenamiento, ha de tenerse en cuenta que estos datos pueden guardar aún así cierta interrelación con el resto. Es por esto por lo que resulta conveniente verificar manualmente cada modelo de red entrenado empleando ciertas trayectorias generadas de forma específica respetando estas un patrón de generación diferente al empleado en la generación del dataset y definiendo un punto de partida concreto y conocido que nos permita contrastar si es este el devuelto por la red.
- 4. Optimización: En este tipo de tareas no existe un método directo para conocer la configuración óptima sino que han de ir comprobándose manualmente muchas de ellas en busca de un conjunto de hiperparámetros que hagan converger correctamente a la red para la gran mayoría de los casos. Hay que tener en cuenta que para cada arquitectura los hiperparámetros se comportarán de modo diferente y es por esto que habrá de tomarse cada arquitectura como un lienzo en blanco sin poder descartar configuraciones que antes no funcionaron en otra arquitectura.

5. Entrenamiento y optimización

5.1. Robot 2RPR con gravedad

Tras un primer estudio de los posibles robots simples a utilizar, se optó por emplear el robot 2RPR el cual se compone de dos cadenas cinemáticas, estando cada una de ellas compuesta por una articulación rotacional, prismática y rotacional de nuevo en dicho orden. En la figura 29 se muestra una imagen esquemática del robot 2RPR:

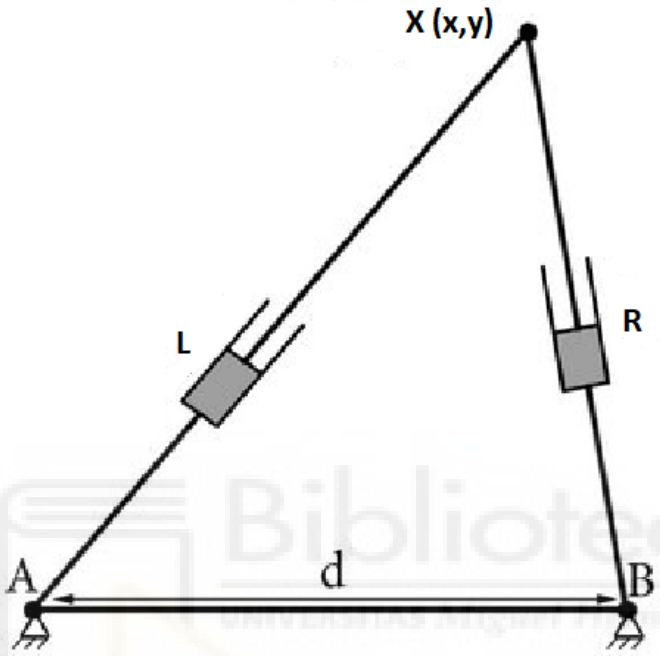

Figura 29: Robot 2RPR

En este robot los esfuerzos de los actuadores y la posición de los mismos resultan fácilmente monitorizables, especialmente en el caso simulado por ser estos directamente los requeridos para seguir la trayectoria, en un caso real habrían de evaluarse los voltajes empleados en la alimentación de los propios actuadores para a partir de estos deducir los esfuerzos aplicados de forma real y adicionalmente usar al menos algún sensor básico (encoder o potenciómetro) para conocer las posiciones instantáneas de las articulaciones activas (en este caso particular serían posiciones lineales o longitudes almacenadas en las variables L y R).

Conocido el robot a emplear resulta clave aclarar el porqué se ha de comenzar por el caso con gravedad en lugar de comenzar por el caso sin gravedad siendo este segundo más simple al menos sobre el papel. Esto se debe a que según desde que posición parta el efector final los actuadores experimentarán unos esfuerzos diferentes en el caso con gravedad, pero en el caso sin gravedad veremos que no será así.

Las dos posiciones del efector final compatibles con los datos posicionales de las articulaciones activas (L y R) resultan ser simétricas, es por esto que la propia longitud de los actuadores no es lo únicamente necesario para poder discriminar cual es la realmente adoptada sino que será la información dinámica adicional (fr y fl) la que nos permita generar esta distinción. Por si solos ni los datos posicionales ni los dinámicos resultan ser suficientes para poder seleccionar cual es el modo de ensamblaje adoptado, sino que será la agrupación de toda esta información la que nos brinde dicha posibilidad.

5.1.1. Procedimiento específico

A la hora de abordar la problemática para el robot 2RPR los pasos a seguir serán los ya descritos con anterioridad, bien es cierto que cada robot presenta ciertas particularidades que marcan la diferencia en el procedimiento. Dado que el procedimiento puede variar según el robot este será explicado de forma detallada y específica.

- **1.Modelo del robot y su controlador**

  Si se pretenden generar modelos robóticos existe una herramienta potente llamada Simulink, esta forma parte del programa Matlab y permite representar relaciones u operaciones matemáticas mediante elementos visuales.

  Concretamente Simulink es una herramienta de modelado y simulación basada en el uso de bloques, mediante los cuales se convierte en un modelo visual lo que antes eran sistemas dinámicos o procesos. Para poder llevar esto a cabo se van definiendo conexiones entre los bloques generando en su conjunto un diagrama.

  Aprovechando el potencial del mencionado software se modelará el suavizado de trayectorias, el robot y su controlador asociado necesario para el seguimiento de las trayectorías.

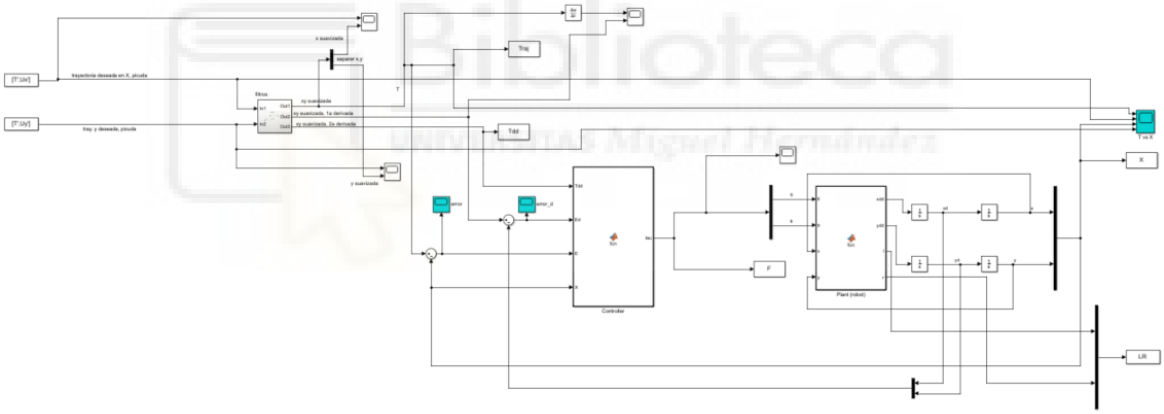

Figura 30: Esquema de bloques completo Simulink

En la figura anterior se muestra el esquema de Simulink completo, este resulta importante pero no muy claro por ello a continuación se irá abordado el mismo parte por parte. Las partes son fácilmente diferenciables ya que sus funciones son notablemente diferentes.

En primer lugar hablaremos de la sección encargada del suavizado de las trayectorias a seguir, como se explicará posteriormente las señales generadas presentan ciertas problemáticas y por ello requieren de un suavizado.

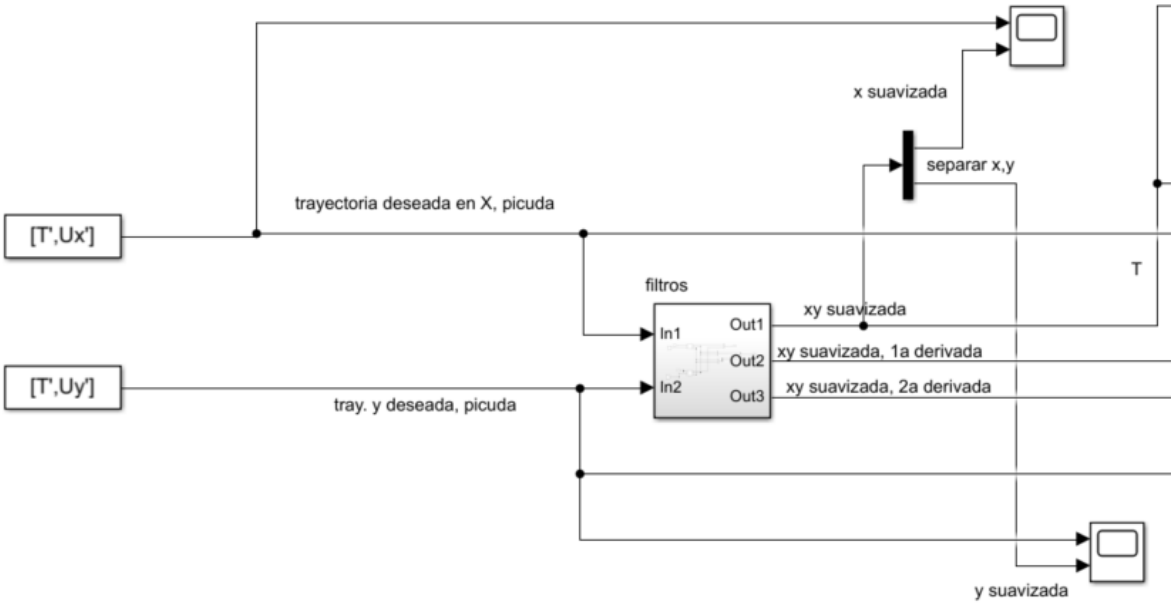

Figura 31: Sección suavizado del esquema de Simulink

En esta primera sección se toman como entradas las trayectorias a seguir, tomando por separado sus componentes x e y. Tras la lectura de dicha información esta es introducida en un bloque con ciertos filtros que no son más que la aplicación de una función de transferencia entre entrada y salida, este bloque es el encargado de suavizar la señal y contiene otro esquema de bloques de menor dimensión:

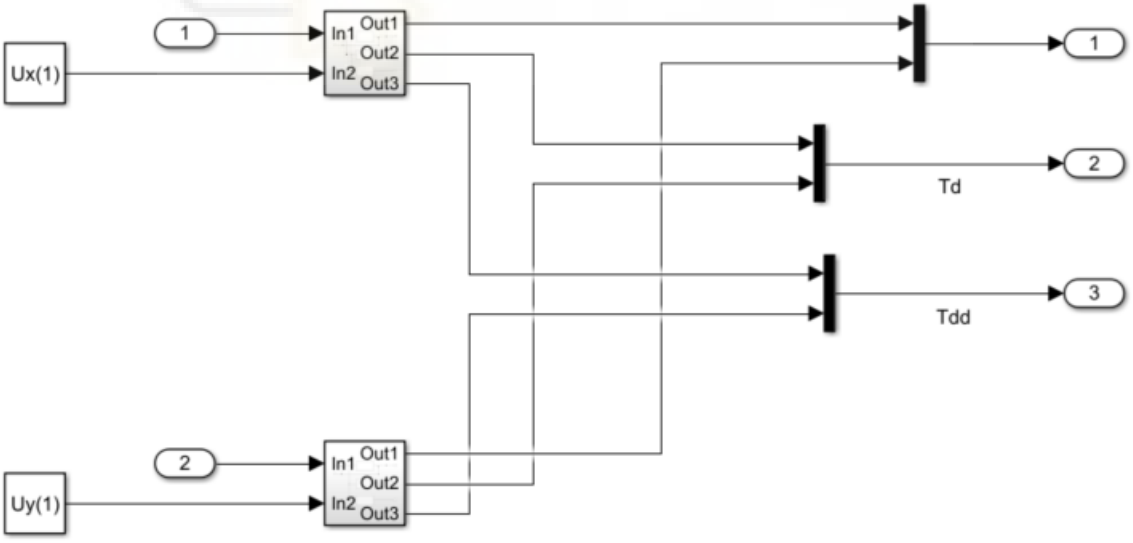

Figura 32: Subesquema suavizado de trayectorias

Estos mostrados bloques contienen un filtro paso-bajo con frecuencia de corte de 0,5 Hz, con la función de transferencia mostrada a continuación:

$$\frac{1}{\frac{1}{(2\cdot\pi\cdot 0{,}5)^2}\cdot s^2+\frac{1}{2\cdot\pi\cdot 0{,}5}\cdot s+1} \tag{5}$$

En el mostrado esquema se lleva a cabo el suavizado de las señales, como bien se indicaba anteriormente a este bloque le llegan las componentes X (Ux) e Y (Uy) de las trayectorias no suavizadas, estas entradas son suavizadas por separado y posteriormente se reconstruyen las señales a partir de las componentes suavizadas. El suavizado de las componentes no es la única tarea realizada sino que se suavizan sin derivar, derivando y derivando doblemente, generando trayectorias en posición, velocidad y acelaración.

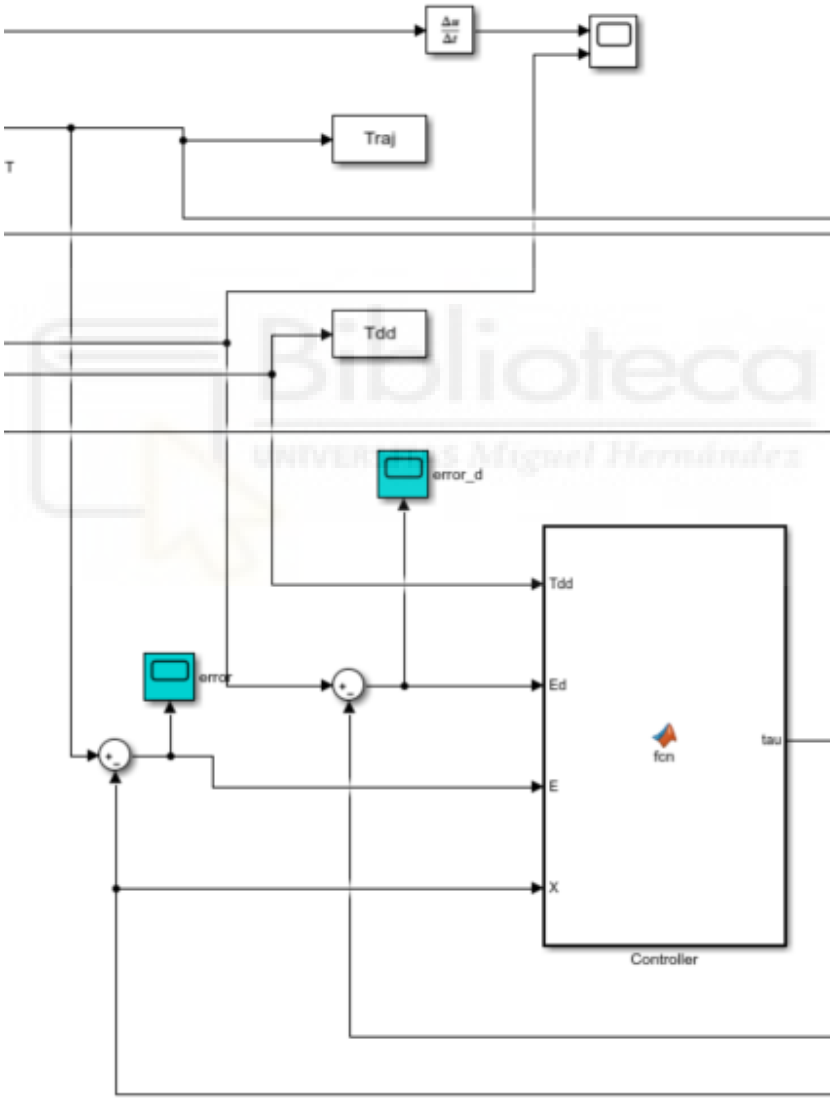

Figura 33: Sección controlador del esquema de Simulink

El controlador mostrado es el encargado de generar las señales de control lanzadas como órdenes al robot, como es lógico este control tiene que poder recibir información sobre el estado instantáneo del seguimiento de la trayectoria.

Si nos fijamos en la imagen anterior vemos que al bloque de control recibe un total de 4 entradas, estas son Tdd (aceleraciones deseadas durante el seguimiento de la trayectoria), Ed (error en velocidad entre la deseada y la real instantánea), E (error posicional instantáneo) y X (posición real instantánea del efector final). A partir de estos datos lo que se pretende obtener es un conjunto de fuerzas (variable tau) para los actuadores de tal modo que ejecutando estas el seguimiento de la trayectoria sea el más fiel posible reduciendo el error, para ello dentro del bloque principal se encuentra un programa dedicado a realizar un control de par computado, siendo este último uno de los controles existentes más eficaces. A continuación se muestra el código:

```
function tau  = fcn(Tdd,Ed,E,X)

% Aplicamos control de par computado (en ingles: Computed Torque Control)

% Calcula el modelo dinamico del robot, y lo compensa para que el error
    de
% seguimiento tienda a cero, o que la trayectoria se siga con fidelidad.

% Model ideal parameters. Unit mass.
m = 1;
b = 1;

x = X(1);
y = X(2);

l=sqrt( x^2 + y^2 );
r=sqrt( (x-b)^2 + y^2 );

Kp = 100*eye(2); % ganancia proporcional
Kd =  20*eye(2); % ganancia derivativa
% Las ganancias se eligen de modo que el seguimiento sea mas o menos
% rapido.
M = m/b * [ l , (b-x)*l/y ; -r , x*r/y ];
% tau = M*( [xdd;ydd] + [0;g] )
tau = M * ( Tdd + Kd*Ed + Kp*E + [0;9.81] );
% tau = [ fl ; fr ]

% si calculamos las fuerzas de control segun las ecuaciones anteriores,
    se
% garantiza el seguimiento de la trayectoria deseada.
```

En este método es empleado control por par computado el cual es extensamente empleado en robótica, sobre este método puede encontrarse información adicional en el citado artículo [33].

En este caso y como se mencionaba con anterioridad el control de par computado genera señales de par o fuerza, estas en un caso real sería aplicadas al robot y en consecuencia se iría reduciendo el error mientras se trata de seguir la trayectoria de interés. Estas señales resultan de gran utilidad pero en nuestro caso al encontrarnos en un entorno simulado no sirven de nada sin la presencia de un modelo del robot capaz de ejecutar las ordenes recibidas, es por esto por lo que aparece una tercera sección en el esquema.

Al no tener un robot real lo aquí contemplado es el modelo generado para imitar lo más fielmente posible el comportamiento del robot real, en sí lo más relevante es comprender a grandes rasgos lo que sucede en esta sección. Al robot le llegan fr y fl siendo estas las fuerzas contenidas en la

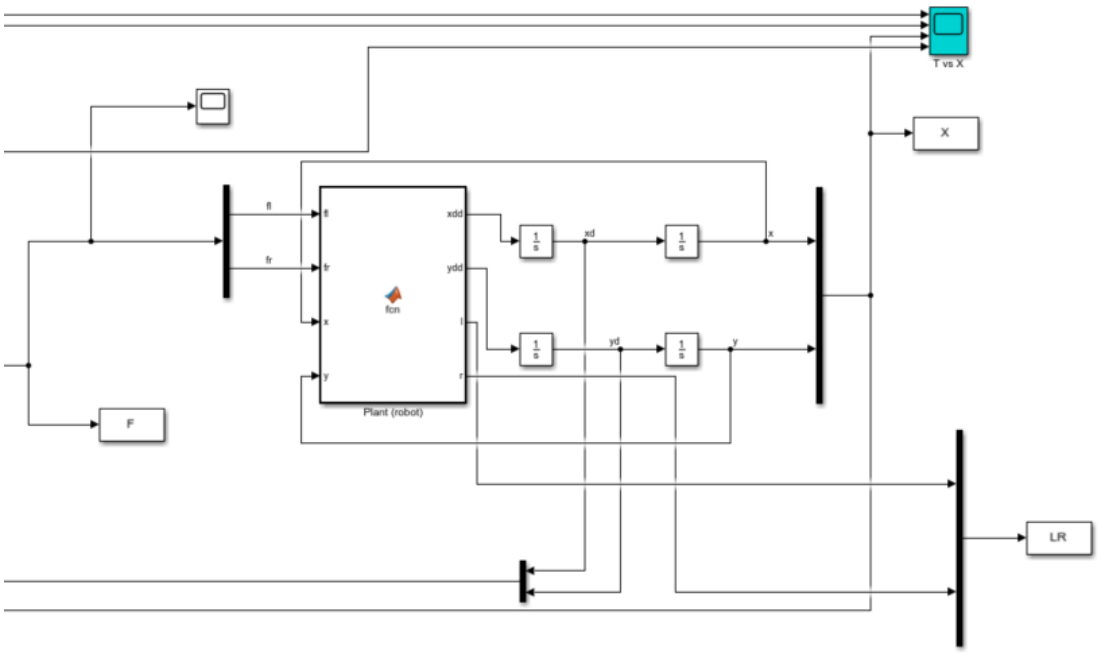

Figura 34: Sección modelo robótico del esquema de Simulink

variable tau, este generará un movimiento en consecuencia y variará la posición en X e Y, las velocidades en dichas componentes (xd e yd) y las longitudes de los actuadores por ser lineales en el presente caso (l y r). Las salidas posicionales (x e y) son realimentadas para el cálculo del error empleado por el control para generar las respuestas esperadas o adecuadas.

En la zona superior derecha del esquema puede apreciarse un scope, siendo este un tipo de bloque que permite dibujar varias señales superpuestas, lo que se pretende es representar la evolución de la trayectoria deseada frente a la trayectoria realmente seguida por el efector final, pudiendo ver qué tan bien esta funcionando el control y en general el sistema.

Volviendo a hablar del bloque central puede apreciarse que sus salidas son aceleraciones por esto son integradas en dos consecutivas ocasiones recuperando así la información posicional al uso contenida en la variable X, y conteniendo realmente tanto la coordenada x como la y del efector final. Esta posición es la que se le introduce al scope anteriormente mencionado y a su vez es la única característica retroalimentada.

A continuación se muestra el contenido del bloque que simula el comportamiento dinámico del robot, es decir, la relación entre las fuerzas fl,fr ejercidas por los pistones y el movimiento que adquiere su efector final (x,y):

```
    function [xdd,ydd,l,r] = fcn(fl,fr,x,y)

m = 1;
b = 1; % Separation between actuators.

tau = [fl;fr];

l=sqrt( x^2 + y^2 );
r=sqrt( (x-b)^2 + y^2 );

M = m/b * [ l , (b-x)*l/y ; -r , x*r/y ];
```

```
accelerations = inv(M)*tau;

g = 9.81;

xdd = accelerations(1);
ydd = accelerations(2) - g;

% tau = M*( [xdd;ydd] + [0;g] )
```

Se pueden encontrar más detalles sobre este modelo dinámico y su funcionamiento en la siguiente referencia [34].

- **2.Archivos clave en la generación de datos**

Comenzaremos explicando el contenido de uno de los códigos clave, este es el contenido en el archivo "simulate_trajectory.m", concretamente se mostrará en primer lugar el código completo y a posteriori este se irá desgranando para explicarlo de una forma más detallada:

```
x_min = -1.5;
x_max =  1.5;
y_min = -1.5;
y_max =  1.5;
y_safety = 0.4;

while 1
    y0 = y_min + (y_max-y_min)*rand;
    if abs(y0) > y_safety
        break
    end
end
X0 = [ x_min + (x_max-x_min)*rand ; y0 ];

% Ahora genera dos puntos intermedios y otro final, que tengan mismo
   signo
% para coordenada Y que el punto inicial.

while 1
    y1 = y_min + (y_max-y_min)*rand;
    if abs(y1) > y_safety && y1*X0(2) > 0 % Si tienen mismo signo
       break
    end
end
X1 = [ x_min + (x_max-x_min)*rand ; y1 ];
t1 = 2.5 + 5*rand;

while 1
    y2 = y_min + (y_max-y_min)*rand;
    if abs(y2) > y_safety &&  y2*X0(2) > 0 % Si tienen mismo signo
       break
    end
end

X2 = [ x_min + (x_max-x_min)*rand ; y2 ];
t2 = t1 + 2.5 + 5*rand;

while 1
    y3 = y_min + (y_max-y_min)*rand;
    if abs(y3) > y_safety && y3*X0(2) > 0 % Si tienen mismo signo
       break
    end
end

X3 = [ x_min + (x_max-x_min)*rand ; y3 ];
t3 = t2 + 2.5;
t3 = round(t3*100)/100;

dt = 0.01;
T = [0:dt:10];
i = randi([10,round(length(T)/2)]);
j = randi([i+1,length(T)-10]);
indices_tiempo = [i,j];
```

```
T1 = T(1:i);
T2 = T(i+1:j);
T3 = T(j+1:end);

U1x = X0(1) + (X1(1)-X0(1))/(T1(end)-T1(1))*(T1-T1(1));
U2x = X1(1) + (X2(1)-X1(1))/(T2(end)-T2(1))*(T2-T2(1));
U3x = X2(1) + (X3(1)-X2(1))/(T3(end)-T3(1))*(T3-T3(1));


U1y = X0(2) + (X1(2)-X0(2))/(T1(end)-T1(1))*(T1-T1(1));
U2y = X1(2) + (X2(2)-X1(2))/(T2(end)-T2(1))*(T2-T2(1));
U3y = X2(2) + (X3(2)-X2(2))/(T3(end)-T3(1))*(T3-T3(1));


Ux = [U1x,U2x,U3x];
Uy = [U1y,U2y,U3y];


% close all
% subplot(1,2,1)
% plot(T,Ux)
% xlim([0,T(end)])
% subplot(1,2,2)
% plot(T,Uy)
% xlim([0,T(end)])
% hold on
% plot([0,T(end)],[1,1]*y_safety*sign(y0));
% if y0>0
%     ylim([0,y_max])
% else
%     ylim([y_min,0])
% end
```

Este código es el responsable de la generación de trayectorias no suavizadas a partir de ciertos puntos generados a partir de azar condicionado, es decir estos puntos se generan de forma aleatoria pero dentro de un espacio restringido por algunas condiciones.

El código se explicará punto a punto a continuación, pudiendo así comprender mucho mejor este:

- Definición del espacio de trabajo

```
x_min = -1.5;
x_max =  1.5;
y_min = -1.5;
y_max =  1.5;

y_safety = 0.4;
```

En este primer tramo se definen los límites del espacio de trabajo tanto en el eje X como en el Y, además de estos límites se puede observar una definición adicional, esta hace referencia a la mínima distancia que puede tolerarse entre la recta horizontal que une los dos puntos de apoyo y el efector final, con esto se evitan configuraciones posicionales de tipo singular.

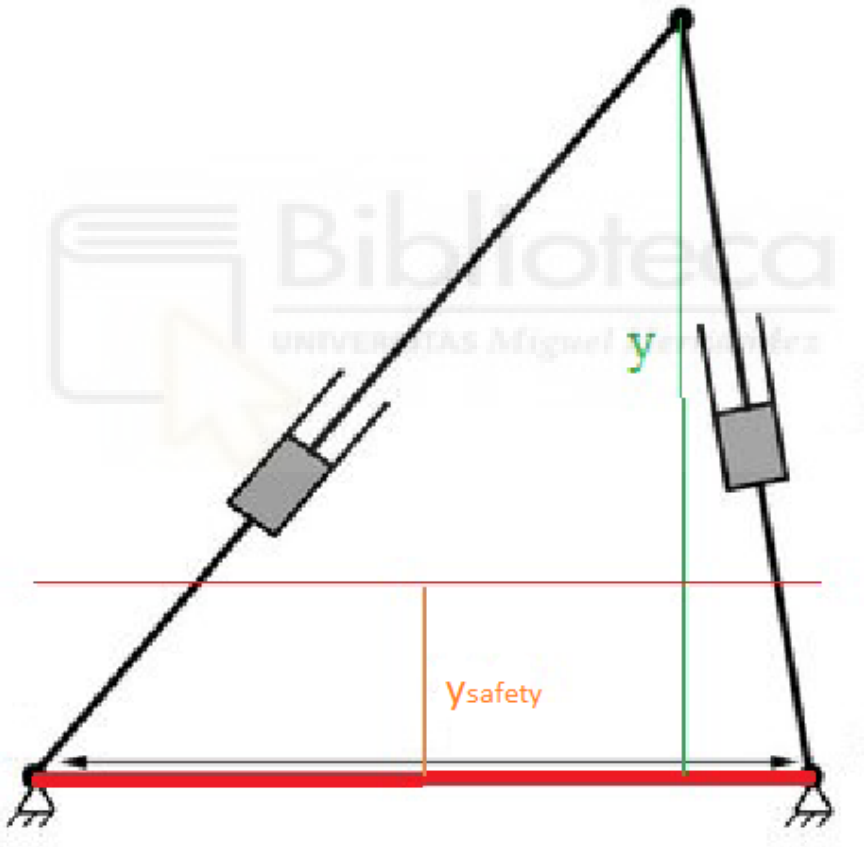

Figura 35: Margen tolerable en el eje y de las trayectorias

En la imagen anterior se explica de forma esquemática que distancia es la que debe respetarse. Aunque en la imagen se muestre únicamente una configuración esta distancia también debe respetarse en la configuración simétrica, es decir ese limíte definido se trata de una distancia en y en valor absoluto que ha de respetarse tanto en el sentido positivo como negativo del eje Y.

- Definición del primer punto de la trayectoria

```
    while 1
    y0 = y_min + (y_max-y_min)*rand;
    if abs(y0) > y_safety
        break
    end
end
X0 = [ x_min + (x_max-x_min)*rand ; y0 ];
```

Lo que se realiza en esta segunda etapa es generar el primer punto a incluir en la trayectoria, para ello se realiza un bucle en el que se genera una coordenada Y aleatoria pero condicionada a estar dentro del espacio de trabajo, posteriormente se verifica si cumple con el margen de seguridad. En caso de ser así el bucle concluye dando lugar a un punto X0 en el que su componente X es generada dentro del espacio de trabajo y la componente Y será la creada y verificada anteriormente. El eje X no resulta problemático y es por esto por lo que no se requieren verificaciones extra.

- Generación de los restantes puntos

```
% Ahora genera dos puntos intermedios y otro final, que tengan mismo
    signo
% para coordenada Y que el punto inicial.

while 1
    y1 = y_min + (y_max-y_min)*rand;
    if abs(y1) > y_safety && y1*X0(2) > 0 % Si tienen mismo signo
       break
    end
end
X1 = [ x_min + (x_max-x_min)*rand ; y1 ];
t1 = 2.5 + 5*rand;

while 1
    y2 = y_min + (y_max-y_min)*rand;
    if abs(y2) > y_safety &&  y2*X0(2) > 0 % Si tienen mismo signo
       break
    end
end

X2 = [ x_min + (x_max-x_min)*rand ; y2 ];
t2 = t1 + 2.5 + 5*rand;

while 1
    y3 = y_min + (y_max-y_min)*rand;
    if abs(y3) > y_safety && y3*X0(2) > 0 % Si tienen mismo signo
       break
    end
end

X3 = [ x_min + (x_max-x_min)*rand ; y3 ];
```

```
t3 = t2 + 2.5;
t3 = round(t3*100)/100;
```

En esta tercera fase se concretan los puntos restantes requeridos para la generación de la trayectoria, el procedimiento es muy similar al explicado en la etapa anterior con la única diferencia de que en la componente Y de estos puntos no solo se verifica el margen mínimo sino que también debe mantenerse el signo de dicha componente entre los diferentes puntos. Con esto se asegura que en ningún punto de la trayectoria ese margen en Y se deje de respetar ya que la trayectoria estará compuesta por una gran cantidad de puntos que nos permitirán unir los 4 anteriormente generados y si dos de ellos tienen signos diferentes parte de los puntos se acercarán al origen en Y, lo que supondría poder llegar a cruzar una singularidad lo cual resulta indeseable.

- Definición de la distribución de tiempos durante la trayectoria

  La trayectoria en sí se compondrá de 1001 puntos de los cuales el primero será X0 y el último será X3, bien es cierto que estos no han de distribuirse de forma equitativa ente los diferentes puntos principales sino que interesará que haya más puntos donde la trayectorias es más lenta, es aquí donde entran en juego los tiempos.

```
dt = 0.01;
T = [0:dt:10];
i = randi([10,round(length(T)/2)]);
j = randi([i+1,length(T)-10]);
indices_tiempo = [i,j];
T1 = T(1:i);
T2 = T(i+1:j);
T3 = T(j+1:end);
```


En este tramo del código se buscan definir tres tiempos, siendo estos los tiempos en los que deberán alcanzarse los puntos X1, X2 y X3, en el caso de X0 no será necesario un tiempo ya que al ser el punto de partida será este el que se ubique el efector final en el instante inicial.
Los tiempos clave son i y j, siendo estos los dos tiempos intermedios que como resulta lógico no pueden ser coincidentes ya que eso supondría la necesidad de alcanzar una velocidad y aceleración infinita que permita desplazar el efector final de un punto al siguiente en 0 segundos, lo cual resulta irreal e imposible. Es por esto por lo que han de cuidarse los valores i y j.
Ya conocidos i y j resulta simple definir T1, T2 y T3, siendo estos tiempos los que a su vez definirán las velocidades y aceleraciones en los tramos de unión entre los puntos principales.

- Cálculo de pendientes por tramos

  En esta última sección se calculan las pendientes de cada tramo de recta mediante la ecuación de la recta, logrando así generar puntos intermedios dentro de la línea recta que une un punto y su consecutivo

```
U1x = X0(1) + (X1(1)-X0(1))/(T1(end)-T1(1))*(T1-T1(1));
U2x = X1(1) + (X2(1)-X1(1))/(T2(end)-T2(1))*(T2-T2(1));
U3x = X2(1) + (X3(1)-X2(1))/(T3(end)-T3(1))*(T3-T3(1));


U1y = X0(2) + (X1(2)-X0(2))/(T1(end)-T1(1))*(T1-T1(1));
U2y = X1(2) + (X2(2)-X1(2))/(T2(end)-T2(1))*(T2-T2(1));
```

```
U3y = X2(2) + (X3(2)-X2(2))/(T3(end)-T3(1))*(T3-T3(1));


Ux = [U1x,U2x,U3x];
Uy = [U1y,U2y,U3y];
```

Tras este código existe una sección adicional pero esta se encuentra comentada ya que solo genera ciertas representaciones gráficas que sirven para verificar la generación de trayectorias.

Al ejecutar este código al completo se nos devuelve la variable T y las variables Ux y Uy referentes a las posiciones deseadas a lo largo de la trayectoria en los mencionados ejes y siendo T concretamente la trayectoria no suavizada, que se introducirá al sistema de suavizado, control y ejecución de Simulink explicado anteriormente.

- **3.Generación de datos**

Teniendo todos los ficheros listos solo queda emplear estos para la generación de datos en grandes cantidades, para ello se usará un fichero complementario encargado de la coordinación y ejecución del resto de ellos.
Este fichero se titula "generate_training_data.m" y a continuación se muestra su contenido:

```
clear all

l = [];
r = [];
fl = [];
fr = [];
x = [];
y = [];

tic

iteraciones = 1;
Nmax = 100000;
l = zeros(1001,Nmax);
r = zeros(1001,Nmax);
x = zeros(1001,Nmax);
y = zeros(1001,Nmax);
fl = zeros(1001,Nmax);
fr = zeros(1001,Nmax);

while 1
    simulate_trajectory;
    if ~max(isnan(Ux)) && ~max(isnan(Uy))
        sim('classic_CTC');
        l(:,iteraciones) = LR(:,1);
        r(:,iteraciones) = LR(:,2);
        x(:,iteraciones) = X(:,1);
        y(:,iteraciones) = X(:,2);
        fl(:,iteraciones) = F(:,1);
        fr(:,iteraciones) = F(:,2);
        iteraciones = iteraciones + 1
    end
    if iteraciones>Nmax
        break
    end
end

tiempo = toc

M = [l',r',fl',fr',x(1,:)',y(1,:)'];

csvwrite('datos_100k',M)
%-------------------------------------------------------------
%% Animar una trayectoria cualquiera, elegida aleatoriamente

close all

sim_index = randi([1,size(l,2)]);

ang = [0:0.1:6.3];
```

```
for i=1:4:size(l,1)
    plot([0,1],[0,0],'k')
    hold on
    plot([0,x(i,sim_index)],[0,y(i,sim_index)],'r','LineWidth',2);
    plot([1,x(i,sim_index)],[0,y(i,sim_index)],'b','LineWidth',2);
    plot(x(i,sim_index),y(i,sim_index),'or');
    cl = l(i,sim_index)*[cos(ang);sin(ang)];
    cr = r(i,sim_index)*[cos(ang);sin(ang)] + [1;0];
    plot(cl(1,:),cl(2,:),'r')
    plot(cr(1,:),cr(2,:),'b')
    plot(x(:,sim_index),y(:,sim_index),'k')
    hold off
    xlim([-2,2])
    ylim([-2,2])
    set(gca,'DataAspectRatio',[1 1 1])
    pause(0.01);
end

%-------------------------------------------------------------
%% Ver las trayectorias del efector a la vez:
close all

sim_index = randi([1,size(l,2)]);
plot([0,1],[0,0],'k')
hold on

skip_factor = 100;

for i=1:skip_factor:size(l,2)
    plot(x(:,i),y(:,i),'k')
end

plot([0,x(1,sim_index)],[0,y(1,sim_index)],'r','LineWidth',2);
plot([1,x(1,sim_index)],[0,y(1,sim_index)],'b','LineWidth',2);
plot(x(1,sim_index),y(1,sim_index),'or');

xlim([-2,2])
ylim([-2,2])
set(gca,'DataAspectRatio',[1 1 1])
```

Este código cuenta con cuatro tramos y tres secciones de ejecución claramente diferenciadas, estas se separan con dos símbolos de porcentaje consecutivos y pueden ser ejecutadas de forma independiente. En este caso la importancia reside en la primera de las secciones ya que es la que genera el dataset siendo las dos secciones restantes secciones destinadas a realizar comprobaciones y representaciones.

A pesar de ser las últimas dos secciones prescindibles estas se explicarán junto a la primera de las mismas con la intención de clarificar lo realizado en el fichero:

Tramo 1 : Inicialización de variables:

```
clear all

l = [];
r = [];
fl = [];
fr = [];
x = [];
y = [];

tic

iteraciones = 1;
Nmax = 100000;
l = zeros(1001,Nmax);
r = zeros(1001,Nmax);
x = zeros(1001,Nmax);
y = zeros(1001,Nmax);
fl = zeros(1001,Nmax);
fr = zeros(1001,Nmax);
```

En este primer tramo de código simplemente se inicializan las variables de interés y aparece un comando llamado "tic" este se usa para comenzar a registrar un tiempo de ejecución y contará hasta que llegue a un punto del código en el que se utilice el comando "toc".

Por otro lado, aparece "Nmax" e "iteraciones", siendo "Nmax" el número de registros que se desean generar e "iteraciones" simplemente la variable empleada como contador.

Tramo 2 : Generación de datos

En el segundo de los tramos se pone en marcha un bucle que genera tantos datos como valor tenga "Nmax", a continuación se muestra el código empleado.

```
    while 1
    simulate_trajectory;
    if ~max(isnan(Ux)) && ~max(isnan(Uy))
        sim('classic_CTC');
        l(:,iteraciones) = LR(:,1);
        r(:,iteraciones) = LR(:,2);
        x(:,iteraciones) = X(:,1);
        y(:,iteraciones) = X(:,2);
        fl(:,iteraciones) = F(:,1);
        fr(:,iteraciones) = F(:,2);
        iteraciones = iteraciones + 1
    end
    if iteraciones>Nmax
        break
    end
end

tiempo = toc

M = [l',r',fl',fr',x(1,:)',y(1,:)'];

csvwrite('datos_100k',M)
```

Lo que aquí se realiza es llamar a la función "simulate_trajectory" y tras comprobar si la trayectoria no presenta ninguna velocidad infinita o problemática se simula el modelo robótico de Simulink, "classic_CTC" para seguir esta primera trayectoria generada por la función "simulate_trajectory" , los resultados de interés del seguimiento serán los almacenados en las mostradas variables dentro del bucle. Si nos fijamos estas variables almacenan en forma de columnas la información respectiva a cada itercación, por lo que tendrán una dimensión 1001 x Nmáx ya que este bucle se repetirá hasta que el número de iteraciones alcance a Nmax, cuando lo supere se cortará la ejecución y se saldrá del bucle.

Al salir del bucle se almacenarán todas las variables traspuestas dentro de la variable M y se almacenará la misma en un archivo CSV, aunque en algunas pruebas se ha empleado un archivo .mat y es perfectamente válido.

Como se ha mencionado antes al aparecer un comando "toc" se finalizará la cuenta temporal registrando el tiempo en la variable con ese mismo nombre, pudiendo así conocer cuanto tiempo se ha requerido para generar Nmáx datos.

Tramo 3 : Animación de trayectorias

En tercer lugar y formando por sí solo una sección de ejecución independiente dentro del fichero tenemos el siguiente código:

```
        %% Animar una trayectoria cualquiera, elegida aleatoriamente

close all

sim_index = randi([1,size(l,2)]);

ang = [0:0.1:6.3];

for i=1:4:size(l,1)
    plot([0,1],[0,0],'k')
    hold on
    plot([0,x(i,sim_index)],[0,y(i,sim_index)],'r','LineWidth',2);
    plot([1,x(i,sim_index)],[0,y(i,sim_index)],'b','LineWidth',2);
    plot(x(i,sim_index),y(i,sim_index),'or');
    cl = l(i,sim_index)*[cos(ang);sin(ang)];
    cr = r(i,sim_index)*[cos(ang);sin(ang)] + [1;0];
    plot(cl(1,:),cl(2,:),'r')
    plot(cr(1,:),cr(2,:),'b')
    plot(x(:,sim_index),y(:,sim_index),'k')
    hold off
    xlim([-2,2])
    ylim([-2,2])
    set(gca,'DataAspectRatio',[1 1 1])
    pause(0.01);
end
```

Con este código lo que se logra es animar una de las trayectorias generadas anteriormente, para esto habrá de lanzarse el código completo o al menos cargar un archivo de trayectorias previamente generadas ejecutando posteriormente a la carga de dichas trayectorias esta sección.

Si se realiza esto correctamente se mostrará en pantalla la animación 2D del robot siguiendo la trayectoria con las velocidades y aceleraciones pertinentes.
Se mostraría en dicho caso una ventana emergente dinámica similar como la mostrada a continuación:

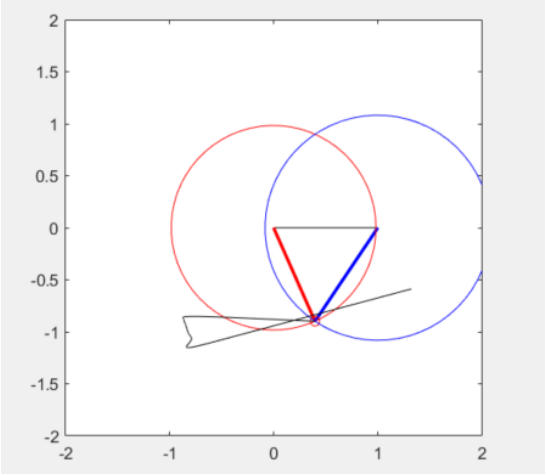

Figura 36: Animación de trayectorías en Matlab

Profundizando en el código destaca en primer lugar el bucle "for i=1:4:size(l,1)" que va recorriendo los puntos de la trayectoria de 4 en 4, haciendo así menos pesado el proceso y perdiendo poca resolución temporal. Para cada uno de estos se realizarán diversas acciones, en primer lugar con "plot([0,1],[0,0],'k')" se dibujará una línea horizontal entre los puntos 0,0 y 0,1, gracias a hold on al realizar una nueva acción no se perderá la ventana sino que se mantendrán todas las representaciones en esta única y primera ventana. Continuando con las acciones lo que se hace además de esto en el bucle es dibujar todos los elementos del robot, el trazado de la trayectoria completo y unos círculos de radio igual a la longitud del actuador en cada momento.
En cada punto seleccionado de la trayectoria se actualizará la representación, entrando al bucle con la posición de interés y representando nuevamente toda la información visual necesaria.

Tramo 4 : Representación de trayectorias generadas

En la última de las secciones del fichero aparece el código encargado de la representación de parte de las trayectorias generadas, para ello se emplea el siguiente código:

```
    %% Ver las trayectorias del efector a la vez:
close all

sim_index = randi([1,size(l,2)]);
plot([0,1],[0,0],'k')
hold on

skip_factor = 100;

for i=1:skip_factor:size(l,2)
    plot(x(:,i),y(:,i),'k')
end

plot([0,x(1,sim_index)],[0,y(1,sim_index)],'r','LineWidth',2);

plot([1,x(1,sim_index)],[0,y(1,sim_index)],'b','LineWidth',2);

plot(x(1,sim_index),y(1,sim_index),'or');

xlim([-2,2])
ylim([-2,2])

set(gca,'DataAspectRatio',[1 1 1])
```


A grandes rasgos lo que se hace es seleccionar en primer lugar un índice aleatorio el cual es almacenado en la variable "sim_index" , este será utilizado posteriormente para la representación del robot ya que si se dibujara el robot para todos los puntos de todas las trayectorias tendriamos un grave problema de claridad en la representación.

Lo que se hace tras el for es concretamente representar el primer punto de la trayectoria coincidente con el número almacenado en dicha variable.

Si vemos no es la única variable que se define al comienzo del código sino que también destaca la variable "skip", esta se emplea para realizar una purga de trayectorias de tal modo que solo 1 de cada 100 trayectorias es representada, pudiendo así liberar carga de computo y a la vez clarificar un poco la representación, de lo contrario se observarían las zonas de interés casi totalmente negras, lo cual dificultaría la comprobación visual.

En el bucle for lo que se realiza es la representación de alguna de las trayectorias completas, concretamente se van omitiendo 100 de estas por cada una que se representa.

Lo que se logra con esto es poder ver de un solo vistazo si las trayectorias presentan ciertas tendencias comunes o si realmente están siendo lo suficientemente aleatorias, a su vez también puede observarse si el espacio de trabajo esta siendo el adecuado. La ejecución de la explicada sección generaría una ventana emergente similar a la siguiente:

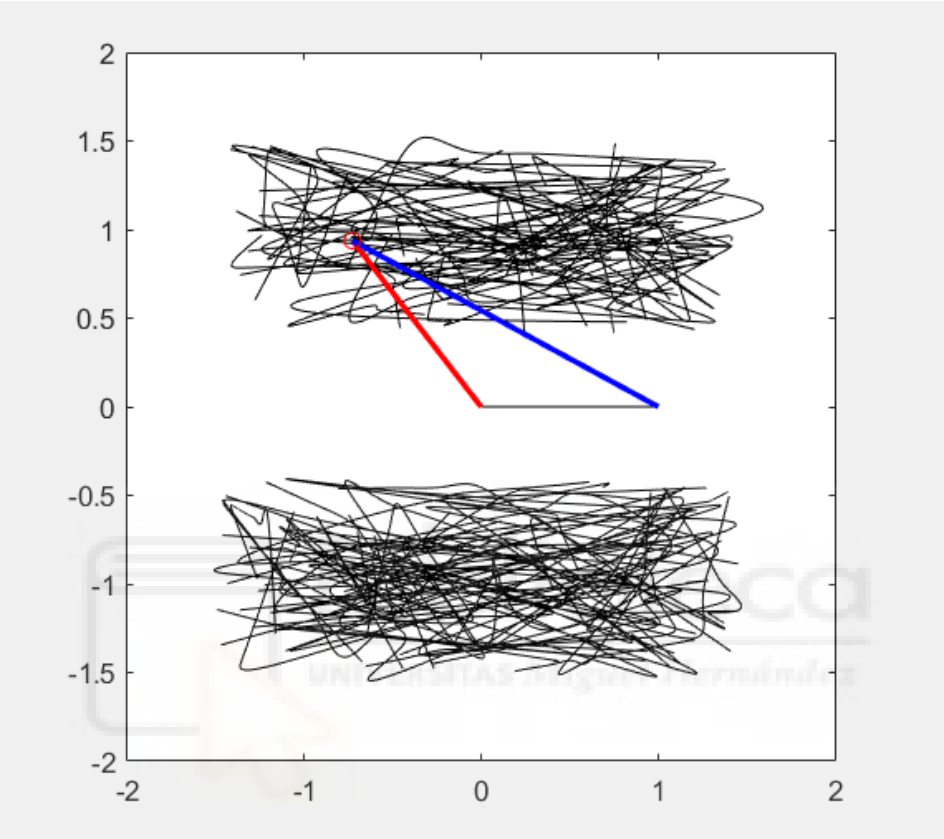

Figura 37: Representación de trayectorias generadas

Como se explicaba antes se representan ciertas trayectorias, viniendo el número de estas definido por el "skip factor", si nos fijamos se forman dos marañas de líneas claramente diferenciables, esta no son más que la representación que ese cierto número de trayectorias. Si no existiese "skip factor" la maraña sería una masa casi totalmente o totalmente negra.

4.Programación y entrenamiento de la red

Ya conocidos los datos ha de generarse y prepararse la red para el entrenamiento, para esto dejaremos atrás Matlab y pasaremos a usar Python, siendo en este segundo programa donde emplearemos los datos generados en Matlab y verificados visualmente para entrenar la red neuronal generada o las redes generadas en caso de requerirse diversas comprobaciones. Tras barajar la opción de usar varios tipos de redes y como ya se comentó anteriormente se ha optado por una red unidireccional. Posteriormente y ya sabiendo que se pretendía entrenar una red unidireccional se optó por la opción de emplear 4004 entradas correspondientes a los 1001 valores de cada uno de los 4 vectores de entrada, estos son correspondientes al par de fuerzas y de longitudes de los actuadores, siendo más concisos los 4004 valores son el resultado de concatenar dichas variables (L, R, fr y fl) para cada una de las señales o iteraciones generadas en el dataset, es decir, existe una entrada de 4004 valores por cada trayectoria registrada.

No resulta casual que sean 1001 valores sino que esta cifra viene determinada por el número de registros o instantes registrados para todas y cada una de las variables, la ejecución de la trayectoria tiene una duración de 10 segundos muestreados cada 0,01 segundos, es por esto por lo que cada vector presenta 1000+1 valores discretos.

En lo que a la salida refiere nos encontramos una decisión similar pero algo más simple, ya que en esta primera estimación solo nos interesan dos valores, los dos valores que definen la posición de partida del efector final (X0 e Y0). Esto quiere decir que en lugar de optar por las 2002 salidas correspondientes a los 1001 valores adoptados por X e Y a lo largo de una trayectoria concreta solo se requieren de 2 salidas, siendo estas los dos primeros valores de dichas secuencias.

A continuación se muestra el código completo realizado en python y posteriormente las respectivas explicaciones:

```
    RED NEURONAL PYTHON
import numpy as np
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split

# Scipy me posibilita importar datos .mat (Matlab)
import scipy.io


#Configuraci n de hiperpar metros
ARCHITECTURE = (4004,100,100,2)
BATCH_SIZE = 800 #Referencia->1000
ACTIVATION_FUNCTION = 'tanh' #Referencia->tanh
EPOCHS =350
LEARNING_RATE = 0.01 #Referencia->0.01
TEST_SIZE=0.2 #Referencia->0.2

# Se leen los datos del fichero .mat
data = scipy.io.loadmat('datos_100k_mat.mat')

l = data['l']
r = data['r']

fr = data['fr']
fl = data['fl']

x = data['x']
y = data['y']

x = x.transpose()
y = y.transpose()
```

```
l = l.transpose()
r = r.transpose()
fr = fr.transpose()
fl = fl.transpose()


# Obtener las dimensiones de fr, viene a ser lo mismo para cada registro
de entrada (es com n en todos)

num_registros = fr.shape[0]
num_instantes = fr.shape[1]


formax=x.shape
formay=y.shape
print('X tiene dimensiones',formax)
print('Y tiene dimensiones',formay)
x0=x[:,1]
y0=y[:,1]
formax0=x0.shape
formay0=y0.shape
print('X0 tiene dimensiones',formax0)
print('Y0 tiene dimensiones',formay0)
x0 = x0[:, np.newaxis]
y0 = y0[:, np.newaxis]

#Uno tambi n las salidas
output_data = np.concatenate((x0, y0), axis=1)
dimensionessal=output_data.shape
print('Las dimensiones de la salida son',dimensionessal)

# Dividir los datos en conjuntos de entrenamiento y prueba
input_data_train, input_data_test, output_data_train,
output_data_test =
train_test_split(input_data, output_data, test_size=TEST_SIZE, random_state
    =42)


#Creo la red unidireccional
model=Sequential()

model.add(tf.keras.Input(shape=(4004,)))
model.add(tf.keras.layers.Dense(100,activation='tanh'))
model.add(tf.keras.layers.Dense(100,activation='tanh'))
model.add(tf.keras.layers.Dense(2,activation='linear'))

model.summary()


#OPTIMIZADORES
#adam_optimizer = tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE)
#model.compile(optimizer=adam_optimizer, loss='mean_squared_error',
metrics=['accuracy'])

#sgd_optimizer = tf.keras.optimizers.SGD(learning_rate=LEARNING_RATE)
#model.compile(optimizer=sgd_optimizer, loss='mean_squared_error',
metrics=['accuracy'])

#rmsprop_optimizer = tf.keras.optimizers.RMSprop(learning_rate=LEARNING_RATE)
#model.compile(optimizer=rmsprop_optimizer, loss='mean_squared_error',
metrics=['accuracy'])
```

```
#adagrad_optimizer = tf.keras.optimizers.Adagrad(learning_rate=LEARNING_RATE)
#model.compile(optimizer=adagrad_optimizer, loss='mean_squared_error',
metrics=['accuracy'])

#adadelta_optimizer = tf.keras.optimizers.Adadelta(learning_rate=1.0)
# No se utiliza learning_rate directamente en Adadelta

#model.compile(optimizer=adadelta_optimizer, loss='mean_squared_error',
metrics=['accuracy'])

nadam_optimizer = tf.keras.optimizers.Nadam(learning_rate=LEARNING_RATE)
model.compile(optimizer=nadam_optimizer, loss='mean_squared_error',
metrics=['accuracy'])



history = model.fit(input_data_train, output_data_train, epochs=EPOCHS,
validation_data=(input_data_test, output_data_test),batch_size=BATCH_SIZE)

# Se evalua el modelo con los datos de test
output_pred=model.predict(input_data_test)

# Calcula el error absoluto medio (MAE)
from sklearn.metrics import mean_absolute_percentage_error
mae = mean_absolute_percentage_error(output_data_test, output_pred)
print('Mean Absolute Percentage Error:', mae)

size1=output_pred.shape
size2=output_data_test.shape
print('La predicci n es', output_pred)
print('Su tama o es:',size1 )
print('La predicci n esperada es', output_data_test)
print('Su tama o es:',size2 )

#Evaluaci n adicional del error:
errorx=[]
errory=[]

#Finalmente lo he automatizado por simplicidad
for i in range(size1[0]):
    errorx.append(abs(output_data_test[i, 0] - output_pred[i, 0]))
    errory.append(abs(output_data_test[i, 1] - output_pred[i, 1]))

errorxmed=sum(errorx)/len(errorx)
print('Deber a poner 0,2xNumRegistros->',len(errorx))
errorymed=sum(errory)/len(errory)
print('Deber a poner 0,2xNumRegistros->',len(errory))


print('Siendo su error medio en x0',errorxmed)
print('Siendo su error medio en y0',errorymed)

#Guardo el modelo de red entrenado
if EPOCHS>70:
    model.save('modelo_entrenadoExcel1.h5')
```

Mostrado el código va a ser analizado este punto por punto:

- Importar paquetes:

```
        RED NEURONAL PYTHON
import numpy as np
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split

# Scipy me posibilita importar datos .mat (Matlab)
import scipy.io
```

En esta primera parte del código se importan los paquetes de utilidad necesarios para el correcto funcionamiento de la red.

En primer lugar tenemos el paquete numpy importado bajo el nombre np por posterior comodidad de uso, este paquete permite optimizar ciertos cálculos y operaciones haciendo más eficientes los códigos.

En segundo lugar los paquetes tensorflow y keras son paquetes de gran utilidad a la hora de generar redes neuronales o sistemas relacionados con el aprendizaje automático, estos paquetes nos permitirán entrenar y constituir la red neuronal de una forma sencilla. Si nos fijamos se mencionan subpaquetes como Sequential o Dense siendo estos referentes a redes unidireccionales y capas de neuronas de redes unidireccionales respectivamente.

En tercer lugar tenemos Scikit-learn o sklearn, dicho paquete nos permetirá entrenar la red y preparar los datos de entrenamiento y test de forma previa.

Por último y a pesar de no ser el único método viable se importó scipy con la intención de poder usar datos en formato .mat de Matlab sin necesidad de exportarlos en otros formatos reconocibles por python como CSV. Con scipy puede leerse comodamente toda la información contenida en forma de variables dentro del .mat y utilizar solo ciertas variables de interés si fuera conveniente.

- Definición de parámetros de la red:

```
#Configuraci n de hiperpar metros
ARCHITECTURE = (4004,100,100,2)
BATCH_SIZE = 800 #Referencia->1000
ACTIVATION_FUNCTION = 'tanh' #Referencia->tanh
EPOCHS =350
LEARNING_RATE = 0.01 #Referencia->0.01
TEST_SIZE=0.2 #Referencia->0.2
```

A la hora de hacer funcionar la red existen ciertos parámetros que como se ha comentado previamente resultan claves, estos parámetros podrían ser empleados de forma directa dentro del código sin su precedente definición pero a la hora de optimizar la red y realizar pruebas resulta conveniente que cada parámetro quede almacenado en una variable pudiendo variar rápidamente los valores que se requiera [31].

- Lectura y tratamiento de datos:

```
        # Se leen los datos del fichero .mat
data = scipy.io.loadmat('datos_100k_mat.mat')
```

```
l = data['l']
r = data['r']

fr = data['fr']
fl = data['fl']

x = data['x']
y = data['y']

x = x.transpose()
y = y.transpose()
l = l.transpose()
r = r.transpose()
fr = fr.transpose()
fl = fl.transpose()
```

En este tramo de código se lee el fichero .mat procedente de la generación de datos en Matlab y se etiqueta cada dato leído en su correspondiente variable, por último se transponen los datos para que presenten las dimensiones esperadas.

- Estudio de dimensiones y verificación de las mismas

```
# Obtener las dimensiones de fr, viene a ser lo mismo para cada registro
de entrada (es com n en todos)
num_registros = fr.shape[0]
num_instantes = fr.shape[1]

#Uno todo lo que quiero introducir como entrada
input_data = np.concatenate((fr, fl, r ,l), axis=1)
dimensiones=input_data.shape
print('Las dimensiones de la entrada son',dimensiones)
```

A la hora de trabajar con redes neuronales resulta clave verificar que las dimensiones de cada una de las entradas sean las correctas de lo contrario la red generaría errores.

En el código mostrado se certifica en primer lugar que las dimensiones de las variables leídas del archivo .mat sean las esperadas, siendo estas las mismas para todas y cada una de las variables por lo que con verificar una es suficiente.

En segundo lugar se concatenan los vectores de entrada componiendo la variable "input_data", representando y conteniendo esta todas las entradas de la red en un único vector.

- Evaluacion de datos de salida

```
formax=x.shape
formay=y.shape
print('X tiene dimensiones',formax)
print('Y tiene dimensiones',formay)
x0=x[:,1]
y0=y[:,1]
formax0=x0.shape
formay0=y0.shape
print('X0 tiene dimensiones',formax0)
print('Y0 tiene dimensiones',formay0)
x0 = x0[:, np.newaxis]
y0 = y0[:, np.newaxis]
```

Al igual que sucedía con la entrada los datos de salida han de ser correctamente preparados y comprobados, en este código se estudian las dimensiones de x e y para posteriormente extraer su valor inicial, siendo este el correspondiente a x0 e y0, a estos segundos valores se les hará una nueva verificación dimensional, asegurando así el correcto funcionamiento del código.

Al final lo que se realiza es dotar a ese par de vectores de valores iniciales de carácter matricial para poder continuar con la ejecución y posteriormente poder concatenarlos.

- Preparación de datos de salida

```
#Uno tambi n las salidas
output_data = np.concatenate((x0, y0), axis=1)
dimensionessal=output_data.shape
print('Las dimensiones de la salida son',dimensionessal)
```

En este tramo lo que se realiza es exactamente lo mismo que se hizo con los datos de entrada, se concatenan los vectores de salida de tal modo que la red vaya leyendo dos a dos las salidas, a lo largo de ambos vectores de longitud igual al número de datos generados. Esto no sería posible sin lo realizado en la parte final del anterior tramo de código, ya que no se pueden concatenar vectores sino que solo es válido el comando si se trabaja con matrices. Nuevamente se comprobarán las dimensiones, pudiendo así llevar cierto control de lo realizado.

- División de datos disponibles

```
# Dividir los datos en conjuntos de entrenamiento y prueba
input_data_train, input_data_test, output_data_train, output_data_test
=train_test_split(input_data, output_data, test_size=TEST_SIZE,
random_state=42)
```

Como se ha explicado en diversas ocasiones las redes se entrenan a partir de un cierto número de ejemplos resueltos que suelen suponer entre el 80 % y 90 % de los ejemplos o datos disponibles, el resto de los datos se usan para verificar el entrenamiento pudiendo así tener una referencia fiable ya que si se usan datos de entrenamiento para el test puede que la red solo acierte por haberlos memorizado y no tanto por el ajuste óptimo de los pesos y parámetros internos. Esto último se conoce como sobreajuste de una red y se debe a que ajusta la red para acertar exactamente todos esos resultados pero ante otros ejemplos no funcionaría bien, por lo que la red carecería de una utilidad real.

- Creación de la red

```
#Creo la red unidireccional
model=Sequential()

model.add(tf.keras.Input(shape=(4004,)))
model.add(tf.keras.layers.Dense(100,activation='tanh'))
model.add(tf.keras.layers.Dense(100,activation='tanh'))
model.add(tf.keras.layers.Dense(2,activation='linear'))

model.summary()
```

En esta parte de código se genera el modelo de red unidireccional y se le van añadiendo las capas convenientes con sus respectivas funciones de activación, bien es cierto que en la capa de entrada no aparece ninguna función de activación, cuando esto sucede es equivalente a estar usando una

función de tipo lineal como sucede en la capa de salida.

Si se presta atención aparece una última línea de código en la que lo que aparece no es más que una llamada para imprimir en pantalla un resumen de la red generada.

Para comprender mejor los sucesivos apartados resulta clave aclarar que parámetros son los clave de una red y que son cada uno de estos:

- Número de capas: Cuantas capas ocultas se pretenden emplear.
- Número de neuronas en cada capa: Cuantas neuronas se ubica en cada capa, especialmente en las ocultas.
- Número de epochs de entrenamiento: Cuantas veces se repite el proceso de entrenamiento.
- Batch size: Los datos disponibles para el entrenamiento (80 % o 90 % de los datos conocidos) se han de pasar a la red uno a uno, pero en paralelo podrían darse múltiples ejecuciones a la vez, es por esto por lo que el tamaño de lote o batch size resulta importante por permitir que el entrenamiento se de de un modo más rápido, un tamaño de lote excesivo podría hacer que la actualización de pesos se de de forma menos frecuente por lo que la red podría perder generalidad o flexibilidad.
- Learning rate: Como de fuerte es la variación de los pesos y parámetros internos de la red según el error en las salidas durante el entrenamiento, resulta importante que este valor no sea excesivo ya que un error puntual podría tener demasiada incidencia.
- Test size: Como se ha comentado con anterioridad no todos los datos conocidos se han de emplear en el entrenamiento ya que la red podría memorizar todos y no se dispondría de más datos para contrastar los resultados. Es por esto por lo que se reserva un porcentaje en tanto por uno para el test, este suele ser 0,1 o 0,2 aunque podría ser otro el valor.
- Optimizador: A la hora de ajustar los parámetros internos de la red como pesos o ciertos sesgos de interconexión neuronal se hace siguiendo un cierto tipo de optimizador que trata de minimizar la función de perdida que no es más que la función que modeliza el error. La gran mayoría de optimizadores están más que comprobados y funcionan correctamente.
- Función de activación: Se emplean para introducir no linealidad dentro de la red de tal modo que se realicen operaciones no lineales con los datos que una neurona recibe a partir de otras y estos sean su salida, por ejemplo funciones de tipo sigmoidal o tangente hiperbólica entre otras.

- Selección de optimizador

```
#OPTIMIZADORES
#adam_optimizer = tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE)
#model.compile(optimizer=adam_optimizer, loss='mean_squared_error',
metrics=['accuracy'])

#sgd_optimizer = tf.keras.optimizers.SGD(learning_rate=LEARNING_RATE)
#model.compile(optimizer=sgd_optimizer, loss='mean_squared_error',
metrics=['accuracy'])

#rmsprop_optimizer = tf.keras.optimizers.RMSprop(learning_rate=
    LEARNING_RATE)
#model.compile(optimizer=rmsprop_optimizer, loss='mean_squared_error',
```

```
metrics=['accuracy'])

#adagrad_optimizer = tf.keras.optimizers.Adagrad(learning_rate=
    LEARNING_RATE)
#model.compile(optimizer=adagrad_optimizer, loss='mean_squared_error',
metrics=['accuracy'])

#adadelta_optimizer = tf.keras.optimizers.Adadelta(learning_rate=1.0)
# No se utiliza learning_rate directamente en Adadelta
#model.compile(optimizer=adadelta_optimizer, loss='mean_squared_error',
metrics=['accuracy'])

nadam_optimizer =
tf.keras.optimizers.Nadam(learning_rate=LEARNING_RATE)
model.compile(optimizer=nadam_optimizer, loss='mean_squared_error',
metrics=['accuracy'])
```

En las mostradas líneas se selecciona el modelo de optimización, los paquetes para la creación de redes neuronales incluyen diversos modelos de optimización los cuales son empleados para la reducción del error durante el entrenamiento.

Todos funcionan correctamente, pero en ciertos casos específicos podrían existir diferencias. Conocido esto resulta necesario poner a prueba algunos de ellos para nuestro caso específico, es por esto por lo que todos aparecen comentados y se descomentará únicamente uno siendo ese el empleado en dicha ejecución.

En esas líneas también se define el tipo de error cuantificado, pudiendo usar este como referencia posteriormente, el error cuadrático medio es el tipo de error seleccionado en este caso.

- Ejecución del entrenamiento

```
history = model.fit(input_data_train, output_data_train, epochs=EPOCHS,
validation_data=(input_data_test, output_data_test),batch_size=BATCH_SIZE
    )
```

A pesar de ser breve el citado código tiene una importancia vital ya que ordena al modelo realizar el entrenamiento cumpliendo los parámetros previamente establecidos. Permitiendo así el ajuste de pesos y parámetros internos correctos para el funcionamiento de la red.

- Se evalua la red entrenada con los datos de test

```
# Se evalua el modelo con los datos de test
output_pred=model.predict(input_data_test)

# Calcula el error absoluto medio (MAE)
from sklearn.metrics import mean_absolute_percentage_error
mae = mean_absolute_percentage_error(output_data_test, output_pred)
print('Mean Absolute Percentage Error:', mae)

size1=output_pred.shape
size2=output_data_test.shape
print('La predicci n es', output_pred)
print('Su tama o es:',size1 )
print('La predicci n esperada es', output_data_test)
print('Su tama o es:',size2 )
```

Tras entrenar la red se emplearán los datos de test para evaluar si las salidas generadas para estos datos son las esperadas o la red presenta problemas en su ajuste.
Después de generar las salidas con la red se estudia el error de los datos de salida generados respecto a los esperados y este será mostrado en pantalla junto a otros parámetros de interés en la verificación como por ejemplo las dimensiones de dichas variables previamente comparadas.

- Verificaciones extra

```
#Evaluaci n adicional del error:
errorx=[]
errory=[]

#Finalmente automatizado por simplicidad
for i in range(size1[0]):
    errorx.append(abs(output_data_test[i, 0] - output_pred[i, 0]))
    errory.append(abs(output_data_test[i, 1] - output_pred[i, 1]))

errorxmed=sum(errorx)/len(errorx)
print('Deber a poner 0,2xNumRegistros->',len(errorx))
errorymed=sum(errory)/len(errory)
print('Deber a poner 0,2xNumRegistros->',len(errory))


print('Siendo su error medio en x0',errorxmed)
print('Siendo su error medio en y0',errorymed)
```

Ya que para comparar entre diferentes modelos entrenados de la red los errores preestablecidos no son muy clarificadores se optó por emplear un error medio solo aplicado a las variables x0 e y0 de forma separada, con estos errores medios se puede obtener una valiosa información sobre el funcionamiento de la red.

Por otro lado y al igual que en ocasiones anteriores se comprueban ciertas dimensiones de utilidad para certificar que el funcionamiento es bueno.

- Guardo el modelo entrenado

```
#Guardo el modelo de red entrenado
if EPOCHS>70:
    model.save('modelo_entrenadoExcel1.h5')
```

Ya entrenado el modelo este se almacena para testearlo de forma más concreta con casos especialmente conocidos o simplemente para poder utilizar la red para resolver la problemática de interés. Ya que a veces se realizan pruebas puntuales del modelo con pocas iteraciones o epochs se ha definido que si estas no superan las 70 no se almacene el modelo, evitando así la sobreescritura indeseada de datos.

- **5.Test del modelo y pruebas**

Cuando se tiene un modelo entrenado no basta con estudiar la calidad o el acierto de los datos generados sobre el porcentaje de datos iniciales destinados a test sino que han de verificarse los resultados de algún otro modo que nos aseguren que el sistema funciona bien para otros datos y no solo para una toma de ellos. Como ya se ha comentado a lo largo del documento existe una problemática ligada a esto conocida como sobreentrenamiento de la red, a lo que esto hace referencia es a aquel proceso que por excesivo resulta en un entrenamiento deficiente en el que la red opta por memorizar secuencias demasiado concretas y solo logra generar buenos resultados de salida para los datos que se le han introducido previamente.

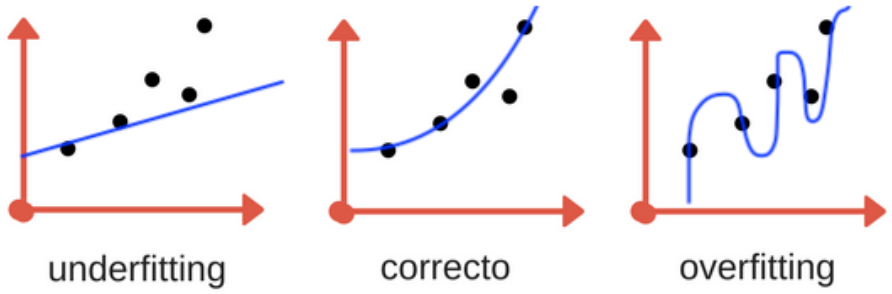

Figura 38: Sobreentrenamiento en redes neuronales [23]

La imagen insertada se compone de tres representaciones y resulta de gran utilidad a la hora de comprender el fenómeno del sobreentrenamiento.

Esta imagen diferencia tres entrenamientos, en el primero de ellos la cantidad de iteraciones ha sido escasa y por ello la red aún no logra comprender la lógica o los patrones que relacionan a los datos de entrada con los de salida porque modela mal dichas relaciones, podría decirse que el eje X son las entradas y el Y las salidas.

En la segunda representación puede verse un seguimiento bastante mejor al anterior, pero este no logra acercarse del todo a lo esperado, esto no quiere decir que sea malo, de hecho, quizás sea el mejor que la red puede obtener.

Por último vemos que se muestra un seguimiento perfecto, pero este no es real ni útil. La razón es que no esta realizando un seguimiento generando salidas basadas en una lógica elaborada, sino que simplemente ha memorizado qué salidas corresponden a ciertas entradas y las está plasmando. Esto último parece ser ideal y reflejaría un acierto total si introdujésemos como entrada ciertos datos empleados durante el entrenamiento, pero si introdujésemos datos diferentes la red generaría unos resultados incluso peores que los que obtendriamos con una red subentrenada.

Para poder saber si un modelo ha sido sobreentrenado resulta interesante generar ciertas trayectorias las cuales sigan una forma conocida y evaluar qué tan bien se estiman los puntos de partida para dichas trayectorias.

Estas trayectorias se generarán empleando un fichero de título "new_test_trajectoryMAT.m", a continuación se explicará el mismo.

El código incluido en el mencionado fichero nos permite seleccionar un punto de partida y a partir de él generar una trayectoria lemniscata (siendo este el nombre técnico de las trayectorias

en forma de infinito), la forma no ha sido escogida al azar sino que se ha tomado esta ya que presenta movimientos arriba, abajo, a derechas y a izquierdas, por lo que incluirá información útil a la hora de deducir el punto de partida.
Para comprender mejor el fichero se muestra a continuación el código:

```
load diez_mil_trayectorias

X0 = [-1,-1]; NAME='TESTdata-1-1';
%X0 = [0,-1]; NAME='TESTdata0-1';
%X0 = [1,-1]; NAME='TESTdata1-1';
%X0 = [-1,1]; NAME='TESTdata-11';
%X0 = [0,1]; NAME='TESTdata01';
%X0 = [1,1]; NAME='TESTdata11';

close all

sim_index = randi([1,size(l,2)]);
plot([0,1],[0,0],'k')
hold on

skip_factor = 100;

for i=1:skip_factor:size(l,2)
    plot(x(:,i),y(:,i),'k')
end

xlim([-2,2])
ylim([-2,2])
set(gca,'DataAspectRatio',[1 1 1])

T = [0:dt:10];

Ux = X0(1) + 0.25*sin(2.4*T);
Uy = X0(2) - 0.2*sin(1.2*T+0.3);

sim('classic_CTC');
l_validate = LR(:,1);
r_validate = LR(:,2);
x_validate = X(:,1);
y_validate = X(:,2);
fl_validate = F(:,1);
fr_validate = F(:,2);

plot(x_validate,y_validate,'g','LineWidth',2)
plot([0,x_validate(1,:)],[0,y_validate(1,:)],'r','LineWidth',2);
plot([1,x_validate(1,:)],[0,y_validate(1,:)],'b','LineWidth',2);
plot(x_validate(1,:),y_validate(1,:),'or');


save(NAME,'fr_validate','fl_validate','r_validate','l_validate')
```

Al ser un código relativamente simple este no se explicará paso a paso sino que se explicará de un modo general. A grandes rasgos lo que este código realiza es en primer lugar, una lectura de uno de los archivos de datos disponibles, sea este un fichero de 100.000, 10.000 datos u otra cantidad no resultaría problemático ya que este solo será empleado para leer ciertas trayectorias y representarlas. Si nos fijamos al comienzo del código aparecen multiples líneas comentadas, esto aparece así por simplicidad ya que solo se descomentará la línea que contenga el punto inicial deseado en cada momento y el nombre apropiado para el posterior registro de la trayectoria generada para dicho punto de partida.

Las trayectorias importadas anteriormente serán representadas teniendo en cuenta un factor de omisión como ya sucedía en ocasiones anteriores, este será de 100 trayectorias omitidas por cada una representada. Hecho esto quedará por representar la trayectoria única, esta deberá seguir una cierta forma que vendrá definida por las posiciones dependientes del tiempo definidas en Ux y Uy. Posteriormente se simulará la trayectoria y se registrarán los datos generados durante el seguimiento de la misma. Hecho esto solo quedará añadir a la representación la trayectoría generada y almacenar todos los datos en un archivo, nuevamente este archivo podría ser .csv pero en este caso se ha empleado un registro en formato .mat.

Si se ejecuta el presente código se debe obtener una representación similar a la aquí mostrada:

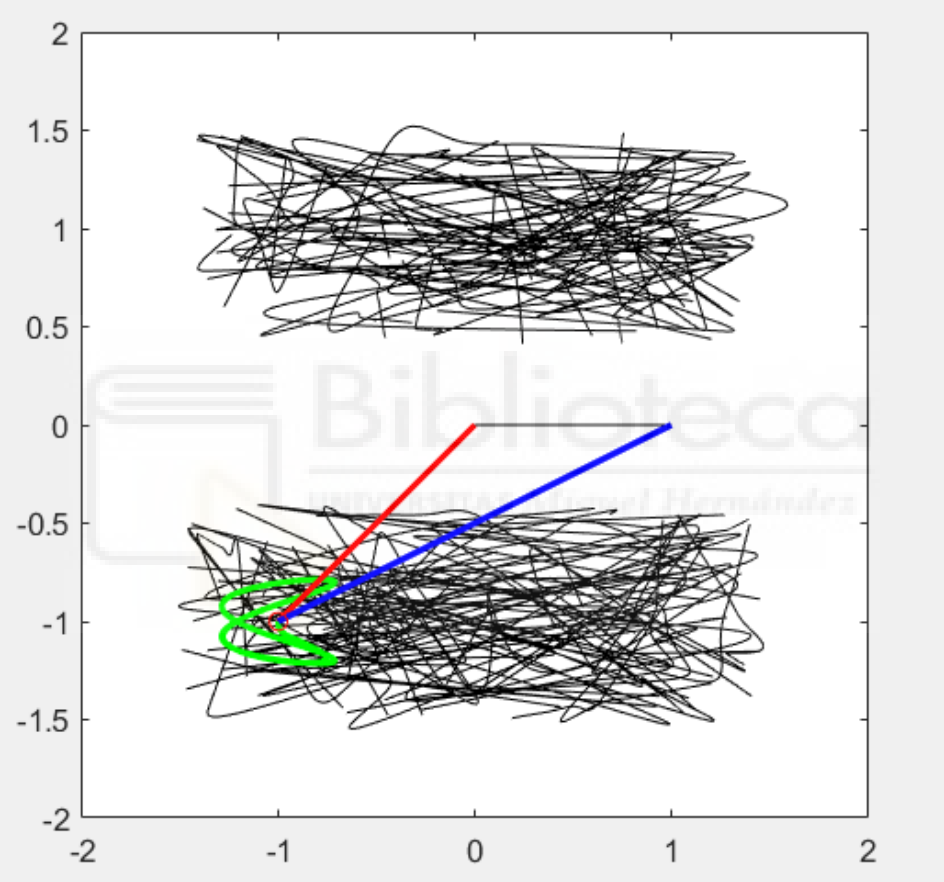

Figura 39: Ejemplo trayectoria de test

Resulta inteligente generar todas las trayectorias desde los diferentes puntos de partida clave dejando almacenados sus respectivos archivos de datos, de este modo tras cada entrenamiento de la red se podrán realizar seis test adicionales, los puntos de partida de interés serán los seis puntos más sencillos,(-1,-1), (0,-1), (1,-1), (-1,1), (0,1) y (1,1). Siendo todos estos puntos en torno al origen, se descartarán aquellos que se encuentren en su misma horizontal por atravesar singularidades si se tratan de alcanzar dichos puntos (-1,0), (0,0), (1,0). Por ser estos puntos relativamente sencillos sus trayectorias no dejarán de ser una gran herramienta para evaluar cada entrenamiento.

Por nuestra percepción esos números nos parecen sencillos por ser enteros, pero para una red neuronal 1 es una valor numérico y 1,98 no deja de ser otro valor numérico, por lo que no dejará de ser un valor más que la red deberá deducir, siendo más concretos una coordenada compuesta

por dos valores sencillos.

Ya conocidas dichas trayectorias habrá de ponerse a prueba la red, para ello se empleará nuevamente un fichero alternativo. El cual tiene por nombre "RedUnidireccionalTestMAT.py" y se ha desarrollado también en lenguaje python. Dicho fichero lee la trayectoria que se quiera y la introduce a la red para que esta prediga el punto del que partió el efector final. El código en cuestión es el aquí destacado:

```
import numpy as np
from keras.models import load_model
# Scipy me posibilita importar datos .mat (Matlab)
import scipy.io

# Lista para almacenar los datos del CSV
RUTAarchivo="TESTdata11"

# Abrir el archivo MAT y leer los datos
data = scipy.io.loadmat(RUTAarchivo)

fr = data['fr_validate']
fl = data['fl_validate']

l = data['l_validate']
r = data['r_validate']


fr = fr.transpose()
fl = fl.transpose()
l = l.transpose()
r = r.transpose()

#Reorganizo
vectorentrada= np.concatenate((fr, fl, r ,l), axis=1)


forma=vectorentrada.shape
print('La dimensi n del input es: ',forma)


# Especifica la ruta del archivo .h5 que contiene el modelo entrenado
modelo_archivo = 'modelo_entrenadoExcel23.h5'
# Carga el modelo desde el archivo
modelo_cargado = load_model(modelo_archivo)

# Predicci n para el conjunto de datos del par ntesis
predicciones = modelo_cargado.predict(vectorentrada)

print('Predicci n:')
print(predicciones)
```

Con la ejecución de este fichero lo que se realiza en primer lugar es definir el nombre del archivo de test que se pretende introducir, para que esto fuera sencillo todos los archivos comparten el nombre "TESTdata**" donde los "*" representan las coordenadas X e Y de partida, por ejemplo el archivo guardado como "TESTdata-1-1" será el correspondiente a la trayectoria iniciada en el punto (-1,-1). Tras la lectura de este archivo se extraen los datos de interés relacionados con la trayectoria preparando los mismos para ser introducidos como entrada.

Ya teniendo los datos resulta importante cargar el modelo de red entrenado deseado, con el paso

de las pruebas se irán almacenando más y más modelos por lo que estos serán numerados, concretamente el nombre tendrá el formato "modelo_entrenadoExcelK" donde "K" hace referencia al número de registro. Como es lógico, al introducir los datos de test uno a uno resulta sencillo saber si la salida es la esperada, ya que se tiene claro cual es la deseada, siendo esta justo la correspondiente a la numeración presente en el nombre del primer fichero cargado.

5.1.2. Entrenamiento y optimización

Tras conocer el procedimiento a nivel teórico se ha de llevar a cabo el mismo, realizando múltiples pruebas y ajustes que nos permitan ver como varían los resultados según que parámetros sean modificados. Será en esta fase en la que se compruebe la efectividad del método y el nivel de fiabilidad de este.

Se mostrará un caso guiado de entrenamiento y a posteriori se mencionarán cada una de las pruebas realizadas explicando la influencia de cada variación. Al trabajar con un gran número de cofiguraciones y pruebas resulta relevante generar un registro de resultados, en nuestro caso este será efectuado en excel por ser esta una gran herramienta para este tipo de casos.

Para dar comienzo a la experimentación se empleará una configuración de referecia determinada a partir de múltiples pruebas previas, en las que se ha logrado llegar a una primera configuración de compromiso. Esta primera configuración empleará 4004 neuronas de entrada (neuronas en la capa de entrada) con una función de activación lineal, 100 neuronas en ambas capas ocultas y ambas capas con funciones de activación de tipo tangente hiperbólica y por último una capa de salida con 2 neuronas y función de activación lineal. Esto definiría la arquitectura de la red pero no son los únicos parámetros a tener en cuenta.

Conocida la estructura de la red han de definirse el resto de parámetros, en primer lugar resulta importante establecer el "BATCH SIZE" o tamaño de lote,en este caso 1000, este parámetro como ya se explicó hace referencia a cuantos datos se procesan de forma simultánea.

En segundo lugar el número de "EPOCHS" o etapas de entrenamiento, este parámetro será clave para evitar un entrenamiento poco trabajado o el caso contrario en el que aparecería el ya explicado fenómeno del sobreentrenamiento, en este caso se fijará el número de EPOCHS en "100".

En tercer lugar el ratio de corrección, dureza del ajuste o "LEARNING RATE", este parámetro podría ser el encargado de que la red converja o no logre hacerlo, para lograrlo es muy importante no usar un valor excesivo, en este caso se empleará un valor de 0,01.

En cuarto lugar ha de definirse qué parte de los datos se destinará al test mediante el parámetro "TEST SIZE", en este caso se reservarán un 20 % de los datos. A la hora de converger no solo importa el "LEARNING RATE" sino que el mecanismo optimizador u "OPTIMIZER" también será relevante, en este caso los ajustes de pesos e hiperparámetros de la red se realizará mediante el uso del optimizador ADAM.

Por último ha de definirse qué tipo de error será el que se mida, este no será del todo relevante ya que se usará comparativamente y casi cualquier tipo sería válido, en este caso no será el único mecanismo de supervisión del error sino que el entrenamiento será evaluado de diversos modos por ser estos errores predeterminados poco clarificadores. El tipo de error seleccionado es el error cuadrático medio o MSE.

Al ejecutar el código, siendo este idéntico al mostrado cuando se explicó el entrenamiento punto por punto, debería mostrarse algo similar a lo siguiente en la parte inferior:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               400500

 dense_1 (Dense)             (None, 100)               10100

 dense_2 (Dense)             (None, 2)                 202

=================================================================
Total params: 410,802
Trainable params: 410,802
```

Figura 40: Muestra de descripción red tras ejecución

Además de la mostrada descripción de la red en la consola se irá mostrando información en directo sobre el estado del entrenamiento y específicamente sobre cada época de entrenamiento.

A continuación se muestran las últimas 5 etapas de entrenamiento y su respectiva información tanto de entrenamiento como de test:

```
Epoch 95/100
8/8 [==============================] - 2s 261ms/step - loss: 0.0691 - accuracy: 0.9220 - val_loss: 0.0776 - val_accuracy: 0.9220
Epoch 96/100
8/8 [==============================] - 1s 134ms/step - loss: 0.0682 - accuracy: 0.9252 - val_loss: 0.0759 - val_accuracy: 0.9160
Epoch 97/100
8/8 [==============================] - 2s 262ms/step - loss: 0.0688 - accuracy: 0.9215 - val_loss: 0.0756 - val_accuracy: 0.9110
Epoch 98/100
8/8 [==============================] - 1s 189ms/step - loss: 0.0716 - accuracy: 0.9187 - val_loss: 0.0781 - val_accuracy: 0.9125
Epoch 99/100
8/8 [==============================] - 2s 259ms/step - loss: 0.0703 - accuracy: 0.9219 - val_loss: 0.0761 - val_accuracy: 0.9170
Epoch 100/100
8/8 [==============================] - 1s 166ms/step - loss: 0.0699 - accuracy: 0.9197 - val_loss: 0.0799 - val_accuracy: 0.9225
```

Figura 41: Información en directo tras ejecución

Al finalizar la ejecución, en este caso tras la etapa o Epoch 100, se mostrará información de utilidad para comprobar qué tan bien se ha realizado el entrenamiento, qué errores se han cometido y algunos parámetros de control como ciertas dimensiones de variables.

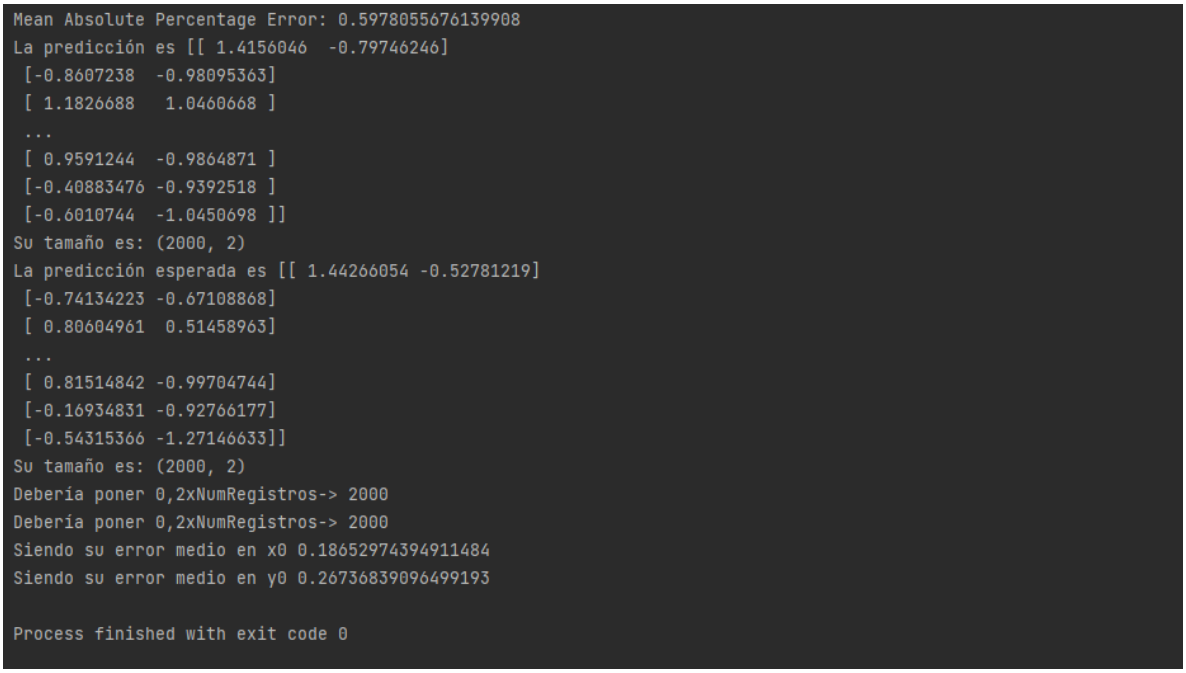

Figura 42: Información mostrada tras finalizar ejecución

De los mostrados datos cabe destacar el error cuadrático medio, y los errores medios para X0 e Y0, a su vez y como método de control se anotarán ciertos datos mostrados con anterioridad como la perdida, la precisión y los mismos valores para la etapa de test de la última de las iteraciones.

Para que quede más claro se mostrarán a continuación los valores de interés que se anotarían tras la presente ejecución:

- Loss o pérdida en la última epoch del entrenamiento: 0,0699
- Precisión en la última epoch del entrenamiento: 0,9197
- Loss o pérdida en la etapa de test de la última epoch: 0,0799
- Precisión en la etapa de test de la última epoch: 0,9225
- Error cuadrático medio global: 0,5978
- Error global medio en X0: 0,1865
- Error global medio en Y0: 0,2674

Como se comentaba con anterioridad estos valores resultan importantes pero no por ello son suficientes para verificar si el entrenamiento ha resultado exitoso, por ello resulta relevante generar una etapa de test manual en el que se proporcionarán como entradas las trayectorias controladas de la ya explicada generación controlada, siendo esta la generación de trayectorias individuales y controladas a partir de un punto de partida concreto y conocido.

Para la presente configuración la red será verificada con los 6 puntos clave ya detallados, a continuación se explicará brevemente el procedimiento y se hará especial incapie en las salidas generadas para dichas entradas de contraste.

Las salidas generadas fueron las siguientes:

- Para X0: -1 e Y0: 1 $->$ -0,9180 0,9351
- Para X0: 0 e Y0: 1 $->$ 0,1352 0,9468
- Para X0: 1 e Y0: 1 $->$ 1,1522 0,9731
- Para X0: -1 e Y0: -1 $->$ -0,9280 -0,9551
- Para X0: 0 e Y0: -1 $->$ -0,2845 -0,9488
- Para X0: 1 e Y0: -1 $->$ 0,8030 -0,9547

Como puede apreciarse los resultados no son del todo acertados, por ello habrán de realizarse más pruebas con diferentes ajustes, aún así estos resultados ya serían de gran utilidad porque nos permitirían discriminar entre las dos posibles soluciones en cada uno de los casos. Deducido el modo de ensamblaje, si se conociese las posiciones articulares L y R, y se tuviera cierta información dimensional del robot sería sencillo deducir la posición exacta del efector final.

Aunque las soluciones generadas no sean del todo exactas existe la posibilidad de a partir de estas soluciones estimadas y aplicando el método de Newton-Raphson, siendo este un método numérico empleado para resolver sistemas de ecuaciones, en el que se parte de un valor que no es solución (pero está suficientemente cerca de ella), y se van realizando unos cálculos iterativos usando la matriz Jacobiana de las ecuaciones que se quieren resolver (la Jacobiana es la matriz de derivadas parciales de dichas ecuaciones), hasta que converge a la solución exacta o se aproxima mucho a esta. Aplicado este procedimiento se tendría la posición "exacta" del efector final y con esta podríamos discriminar sin lugar a error cual es el modo de ensamblado actual del robot. Quizás en un robot sencillo no parezca importante, pero en robots complejos donde existen múltiples modos de ensamblado compatibles para unas mismas posiciones articulares activas, puede resultar imprescindible que la posición sea lo más exacta posible.

5.1.3. Resultados

En este proceso experimental se seguirá una secuencia definida, para un conjunto de datos conocido se realizarán diversos entrenamientos variando en cada uno de ellos ciertos parámetros, eso sí para estudiar mejor la influencia de la variación de cada uno de ellos se modificarán uno a uno. Para cada una de las modificaciones se realizará un entrenamiento en el que un cierto porcentaje de los datos se destina a realizar el test devolviendo ciertos datos sobre el rendimiento de la red, adicionalmente se generarán ciertas trayectorias con puntos de partida conocidos y se introducirán como entrada estudiando la efectividad de la red.

Ya explicado y ejecutado el primero de los ejemplos se procederá a mostrar los resultados obtenidos en cada uno de los experimentos y pruebas generadas, para ello en primer lugar se mostrarán ciertos parámetros los cuales se mantendrán invariantes a lo largo de la experimentación, estos no han sido definidos aleatoriamente, sino que previo a los siguientes ensayos, se realizaron decenas de ellos a modo de aproximación, siendo esta la arquitectura adecuada para el presente caso:

Normalización	Neuronas IN	Activación IN	Neuronas 1	Activación 1
Off	4004	Lineal	100	Tangente hiperbólica

Neuronas 2	Activación 2	Neuronas OUT	Activación OUT
100	Tangente hiperbólica	2	Lineal

Estos serán los parámetros que permanezcan invariantes siendo otros los que vayan siendo modificados en busca de estudiar su influencia y sobre todo tratando de mejorar los resultados obtenidos, pudiendo así acercarnos al modelo ideal. Dicho esto se mostrarán a continuación los resultados obtenidos en distintas tablas, separando estos por grupos y aportando una explicación por cada grupo de estos citados resultados.

En primer lugar se trabajará con el **BATCH SIZE**. Partiendo de 1000 se irá variando este, durante el proceso no se seguirá un orden de variaciones ascendente ni descendente sino que se irá modificando en busca de óptimos locales en los resultados. Las modificaciones y sus respectivos resultados serán mostrados a continuación en el orden en el que estas se han ido ejecutando. Debido al gran número de datos las tablas se construirán verticalmente, es decir cada columna será una de las pruebas.

Las tablas se organizarán por páginas pudiendo así estas ser diferenciadas claramente.

DATOS GENERALES			
Título	Entrenamiento 1	Entrenamiento 2	Entrenamiento 3
Número de datos	10000	10000	10000
Batch size	1000	500	100
Epochs	100	100	100
Learning rate	0,01	0,01	0,01
Test size	0,2	0,2	0,2
Optimizer	ADAM	ADAM	ADAM
Error type	MSE	MSE	MSE
ÚLTIMA ITERACIÓN			
Loss	0,0716	0,0800	0,1106
Accuracy	0,9125	0,9151	0,8982
Test loss	0,0800	0,0853	0,1230
Test accuracy	0,9005	0,9090	0,8875
MEDIAS GLOBALES ENTRENAMIENTO			
MSE	0,6141	0,6042	0,9358
Error medio X0	0,1949	0,2016	0,2724
Error medio Y0	0,2618	0,2666	0,2803
TEST DE VERIFICACIÓN CONTROLADO			
VALOR ESPERADO	VALORES GENERADOS		
X0: -1	-0,9180	-0,8445	-0,5310
Y0: 1	0,9351	0,9632	0,9406
Error línea recta	0,1046	0,1598	0,4728
X0: 0	0,2845	0,0634	0,0499
Y0: -1	-0,9488	-1,0084	-0,8364
Error línea recta	0,2891	0,0639	0,1711
X0: 1	0,8030	1,4134	1,2025
Y0: -1	-0,9547	-1,0847	-1,0675
Error línea recta	0,2022	0,4220	0,2134
X0: -1	-0,9280	-0,3941	-0,0765
Y0: -1	0,9551	-0,9813	-0,7862
Error línea recta	0,0848	0,6061	0,9480
X0: 1	0,1352	-0,0841	-0,4022
Y0: 1	0,9468	0,9341	0,9252
Error línea recta	0,1453	0,1069	0,4091
X0: 1	1,1522	0,5184	1,2766
Y0: 1	0,9731	1,1004	0,9445
Error línea recta	0,1546	0,4919	0,2821
Error medio recta	0,1634	0,3084	0,4161

DATOS GENERALES			
Título	Entrenamiento 4	Entrenamiento 5	Entrenamiento 6
Número de datos	10000	10000	10000
Batch size	1500	2000	2500
Epochs	100	100	100
Learning rate	0,01	0,01	0,01
Test size	0,2	0,2	0,2
Optimizer	ADAM	ADAM	ADAM
Error type	MSE	MSE	MSE
ÚLTIMA ITERACIÓN			
Loss	0,0800	0,0735	0,0799
Accuracy	0,9114	0,9185	0,9035
Test loss	0.0856	0,0854	0,0887
Test accuracy	0.9045	0,9065	0,8990
MEDIAS GLOBALES ENTRENAMIENTO			
MSE	0,7373	0,6318	0,7060
Error medio X0	0,2055	0,2015	0,2122
Error medio Y0	0,2690	0,2707	0,2754
TEST DE VERIFICACIÓN CONTROLADO			
VALOR ESPERADO	VALORES GENERADOS		
X0: -1	-0,8083	-0,8866	-0,8465
Y0: 1	1,0125	1,0024	0,9753
Error línea recta	0,1921	0,1134	0,1555
X0: 0	0,1791	0,0655	0,0252
Y0: -1	-0,9950	-1,0185	-0,9904
Error línea recta	0,1792	0,0681	0,0269
X0: 1	1,3164	0,8277	1,0822
Y0: -1	-0,9103	-1,1163	-1,0528
Error línea recta	0,3289	0,2079	0,0977
X0: -1	-0,8584	-1,1055	-0,3415
Y0: -1	-0,9535	-0,9574	-0,9984
Error línea recta	0,1490	0,1138	0,6585
X0: 0	-0,0750	0,1289	-0,1925
Y0: 1	1,0048	1,0384	1,0105
Error línea recta	0,0752	0,1345	0,1928
X0: 1	0,8750	1,1875	0,8153
Y0: 1	0,8040	0,8236	1,0447
Error línea recta	0,2325	0,2574	0,1900
Error medio rectas	0,1928	0,1492	0,2202

DATOS GENERALES		
Título	Entrenamiento 7	Entrenamiento 8
Número de datos	10000	10000
Batch size	4004	2250
Epochs	100	100
Learning rate	0,01	0,01
Test size	0,2	0,2
Optimizer	ADAM	ADAM
Error type	MSE	MSE
ÚLTIMA ITERACIÓN		
Loss	0,0871	0,0807
Accuracy	0,8920	0,9069
Test loss	0,1016	0,0898
Test accuracy	0,8755	0,8975
MEDIAS GLOBALES ENTRENAMIENTO		
MSE	0,8028	0,6198
Error medio X0	0,2374	0,2172
Error medio Y0	0,2764	0,2738
TEST DE VERIFICACIÓN CONTROLADO		
VALOR ESPERADO	VALORES GENERADOS	
X0: -1	-0,6518	-0,7348
Y0: 1	0,9554	0,9631
Error línea recta	0,3510	0,2678
X0: 0	0,2266	0,4391
Y0: -1	-1,1510	-0,9693
Error línea recta	0,2723	0,4402
X0: 1	1,1537	1,1633
Y0: -1	-1,2544	-0,9337
Error línea recta	0,2972	0,1763
X0: -1	-0,8232	-0,9601
Y0: -1	-0,9923	-0,9388
Error línea recta	0,1769	0,0730
X0: 0	0,0147	-0,0384
Y0: 1	0,9774	1,0477
Error línea recta	0,0269	0,0612
X0: 1	1,2705	0,6722
Y0: 1	0,8640	1,0391
Error línea recta	0,3028	0,3301
Error medio rectas	0,2379	0,2248

A la hora de comparar resultados no todo resulta tan claro, ya que no siempre los mejores resultados de entrenamiento vienen acompañados por el mejor de los resultados en las verificaciones controladas, es por esto por lo que ambos caminos han de ser considerados. En este caso los resultados más destacados se han resaltado en verde dentro de la tabla.

En este caso los entrenamientos más destacados han resultado ser el primero y el quinto con tamaños de batch 1000 y 2000 respectivamente. Ambos han resultado destacados, pero no por las mismas causas, uno de ellos gracias a los resultados en el entrenamiento y otro gracias a los resultados en la posterior verificación. Teniendo en cuenta que ambos resultados son importantes cabe destacar que el entrenamiento 1 presenta un mejor equilibrio en sus resultados y es por esto por lo que se considera más fiable, fijando el tamaño de batch en 1000, siendo este el valor que empleamos también en el ejemplo guiado.

Esto no es casualidad ya que como se comentaba anteriormente de manera previa a la presente experimentación se realizaron decenas de pruebas orientativas, permitiendo estas estimar una confi-

guración de partida adecuada, no siempre tendrán que ser estos valores iniciales los correctos pero sí estarán cerca de serlo.

Aunque ahora se tome por ganadora una de las configuraciones esto no quiere decir que a lo largo de las siguientes pruebas y tras modificar algún otro parámetro no se pueda volver a modificar el batch size, de hecho este será modificado con total seguridad en busca de nuevas configuraciones óptimas. Bien es cierto que en caso de modificarse este se congelarán el resto de modificaciones, ya que si se modificaran varios parámetros a la vez perderíamos claridad a la hora de estudiar la influencia de cada una de estas modificaciones.

Tras comprobar los resultados con diversos valores del batch size, se fijará temporal o definitivamente este en 1000. Fijado el valor del batch size y los valores invariantes anteriormente detallados se procederá a variar el número de **EPOCHS** partiendo con una variación al alza de 150 desde las 100 fijadas anteriormente.

DATOS GENERALES			
Título	Entrenamiento 9	Entrenamiento 10	Entrenamiento 11
Número de datos	10000	10000	10000
Batch size	1000	1000	1000
Epochs	250	350	500
Learning rate	0,01	0,01	0,01
Test size	0,2	0,2	0,2
Optimizer	ADAM	ADAM	ADAM
Error type	MSE	MSE	MSE
ÚLTIMA ITERACIÓN			
Loss	0.0674	0,0575	0,0493
Accuracy	0.9234	0,9261	0,9291
Test loss	0.0743	0,0649	0,0518
Test accuracy	0.9180	0,9135	0,9320
MEDIAS GLOBALES ENTRENAMIENTO			
MSE	0,5931	0,4587	0,4543
Error medio X0	0,1776	0,1534	0,1627
Error medio Y0	0,2516	0,2454	0,1904
TEST DE VERIFICACIÓN CONTROLADO			
VALOR ESPERADO	VALORES GENERADOS		
X0: -1	-0,9500	-0,8678	-0,7817
Y0: 1	0,9681	0,9459	0,9857
Error línea recta	0,0593	0,1429	0,2187
X0: 0	0,1187	0,1648	0,2376
Y0: -1	-0,9103	-0,9881	-0,9670
Error línea recta	0,1488	0,1652	0,2399
X0: 1	0,8078	1,0859	1,0946
Y0: -1	-1,0475	-1,0006	-0,8549
Error línea recta	0,1980	0,0859	0,1732
X0: -1	-0,5371	-1,0750	-0,9926
Y0: -1	-0,9236	-0,9288	-0,8142
Error línea recta	0,4692	0,1034	0,1859
X0: 0	-0,1060	0,0092	0,0586
Y0: 1	1,1182	1,0431	0,7308
Error línea recta	0,1588	0,0441	0,2755
X0: 1	1,0178	0,8464	0,8077
Y0: 1	1,1425	1,3141	1,0034
Error línea recta	0,1436	0,3497	0,1923
Error medio recta	0,1963	0,1485	0,2143

En este caso destacan los entrenamientos 10 y 11, esto resulta lógico ya que 100 iteraciones era un valor algo escaso y era poco probable que estuviera cerca de los valores de sobreentrenamiento. Al aumentar por tanto las iteraciones o epochs de entrenamiento vemos que los resultados obtenidos mejoran, pero no por aumentar más estos tienen que ser necesariamente mejores, si nos fijamos en el caso actual vemos que los resultados son mas sólidos para un número de iteraciones intermedio, haciendo que con 350 epoch el sistema converja correctamente tanto durante el entrenamiento como en la fase de verificación. Dicho esto serán 350 epochs las tomadas como referencia.

Que el aumento de epochs no venga necesariamente ligado a una mejora de los resultados suele deberse a que la red está mostrando tendencias hacia sobreentrenamiento dejándose guiar por lo conocido previamente y no tanto por el entendimiento de la problemática.

Fijado el valor de epochs en los ya mencionados 350 y el batch en 1000 se realizará una prueba idéntica a una de las realizadas en la anterior fase, concretamente una prueba idéntica al entrenamiento 10 que ha resultado ser el más favorable pero incrementando el número de **DATOS** de los 10.000 a los 100.000 y dando paso al uso de este nuevo dataset para próximas pruebas.

DATOS GENERALES	
Título	Entrenamiento 12
Número de datos	100000
Batch size	1000
Epochs	350
Learning rate	0,01
Test size	0,2
Optimizer	ADAM
Error type	MSE
ÚLTIMA ITERACIÓN	
Loss	0,0340
Accuracy	0,9359
Test loss	0,0355
Test accuracy	0,9303
MEDIAS GLOBALES ENTRENAMIENTO	
MSE	0,5273
Error medio X0	0,1182
Error medio Y0	0,1693
TEST DE VERIFICACIÓN CONTROLADO	
VALOR ESPERADO	VALORES GENERADOS
X0: -1	-0,8773
Y0: 1	0,7960
Error línea recta	0,2381
X0: 0	0,0286
Y0: -1	-0,9551
Error línea recta	0,0533
X0: 1	1,0932
Y0: -1	-1,2898
Error línea recta	0,3045
X0: -1	-0,7885
Y0: -1	-0,9129
Error línea recta	0,2288
X0: 0	-0,0103
Y0: 1	0,9180
Error línea recta	0,0826
X0: 1	1,0252
Y0: 1	0,8614
Error línea recta	0,1408
Error medio recta	0,1747

En este entrenamiento el dataset es mucho más amplio lo que aleja la posibilidad de generar sobreentrenamiento para un mismo número de epochs. Si nos fijamos en los resultados estos mejoran bastante durante el entrenamiento en comparación a los obtenidos para el entrenamiento número 10, pero empeoran ligeramente en la fase de validación controlada, al solo evaluarse controladamente con 6 puntos, puede que concretamente para estos la red genere peores resultados, pero en este caso la generalidad de esta y su rango de aplicación debería ser mayor. Debido a que el nivel de generalidad debería ser mayor y a que las mejoras presentan mayor magnitud en balance respecto a los resultados empeorados, se continuará trabajando con este último dataset de 100.000 datos.

Como se comentó anteriormente la modificación de un parámetro no implica que este no pueda ser modificado nuevamente, en la actual fase dado que las condiciones han variado se volverá a tratar de optimizar la red modificando el batch size. En la primera de las fases se modificó el batch size teniendo 10.000 datos en el dataset leído y 100 epochs. Actualmente se tiene un dataset con 100.000 datos y 350 epochs por lo que las situaciones son totalmente diferentes.

DATOS GENERALES			
Título	Entrenamiento 13	Entrenamiento 14	Entrenamiento 15
Número de datos	100000	100000	100000
Batch size	10000	800	600
Epochs	350	350	350
Learning rate	0,01	0,01	0,01
Test size	0,2	0,2	0,2
Optimizer	ADAM	ADAM	ADAM
Error type	MSE	MSE	MSE
ÚLTIMA ITERACIÓN			
Loss	0,0163	0,0434	0,0556
Accuracy	0,9545	0,9372	0,9239
Test loss	0,0169	0,0455	0,0578
Test accuracy	0,9546	0,9397	0,9202
MEDIAS GLOBALES ENTRENAMIENTO			
MSE	0,4253	0,6495	0,7157
Error medio X0	0,0907	0,1307	0,1328
Error medio Y0	0,1126	0,2016	0,2369
TEST DE VERIFICACIÓN CONTROLADO			
VALOR ESPERADO	VALORES GENERADOS		
X0: -1	-0,8212	-0,9881	-0,8508
Y0: 1	0,8778	0,9321	0,9454
Error línea recta	0,2165	0,0689	0,1589
X0: 0	0,1470	-0,0969	0,1233
Y0: -1	-0,8526	-1,1100	-0,8212
Error línea recta	0,2082	0,1466	0,2171
X0: 1	1,2036	1,1097	1,0097
Y0: -1	-1,2543	-0,9748	-0,7494
Error línea recta	0,3258	0,1126	0,2508
X0: -1	-0,9082	-0,8610	-0,8102
Y0: -1	-1,0055	-1,1245	-0,9172
Error línea recta	0,0920	0,1865	0,2071
X0: 0	-0,0936	-0,0222	0,1077
Y0: 1	0,8398	0,9495	1,1840
Error línea recta	0,1856	0,0552	0,2132
X0: 1	0,8842	1,0755	0,9863
Y0: 1	1,2026	1,0649	0,7565
Error línea recta	0,2334	0,0996	0,2438
Error medio recta	0,2102	0,1116	0,2152

DATOS GENERALES		
Título	Entrenamiento 16	Entrenamiento 17
Número de datos	100000	100000
Batch size	850	750
Epochs	350	350
Learning rate	0,01	0,01
Test size	0,2	0,2
Optimizer	ADAM	ADAM
Error type	MSE	MSE
ÚLTIMA ITERACIÓN		
Loss	0,0415	0,0486
Accuracy	0,9352	0,9329
Test loss	0,0415	0,0475
Test accuracy	0,9259	0,9326
MEDIAS GLOBALES ENTRENAMIENTO		
MSE	0,5603	0,6575
Error medio X0	0,1221	0,1342
Error medio Y0	0,1906	0,2020
TEST DE VERIFICACIÓN CONTROLADO		
VALOR ESPERADO	VALORES GENERADOS	
X0: -1	-0,9018	-0,9408
Y0: 1	0,8418	1,1139
Error línea recta	0,1862	0,1284
X0: 0	0,1667	0,4790
Y0: -1	-1,1328	-1,0871
Error línea recta	0,2132	0,4869
X0: 1	1,0382	1,0683
Y0: -1	-1,3341	-1,1138
Error línea recta	0,3363	0,1327
X0: -1	-0,7688	-0,9089
Y0: -1	-1,0858	-0,9397
Error línea recta	0,2466	0,1093
X0: 0	0,0267	0,0147
Y0: 1	0,9279	1,0445
Error línea recta	0,0768	0,0468
X0: 1	1,0633	0,9606
Y0: 1	1,0174	1,0390
Error línea recta	0,0657	0,0555
Error medio rectas	0,1875	0,1599

En este caso destacan los entrenamientos 13 y 14, nuevamente cada uno de ellos destaca por un motivo diferente. Bien es cierto que el primero de ellos emplea un batch size exagerado, siendo este concretamente de 10.000, aumentando el coste computacional. Por su lado el segundo presenta un valor bastante más habitual, esto hace de que el coste computacional sea bastante más asumible por casi cualquier unidad informática, como desventaja podría decirse que al procesar 800 datos en paralelo en lugar de 10.000 los tiempos de procesamiento serán algo mayores.

Analizados los resultados se ha tomado como mejor opción los referentes al entrenamiento 14.

Tras varias fases de entrenamiento se han logrado obtener unos **valores de compromiso** entre tiempos de ejecución, precisión y fiabilidad. Concretando la configuración seleccionada se muestra en la siguiente tabla:

DATOS GENERALES	
Título	Valores de referencia actualizados
Número de datos	100000
Batch size	800
Epochs	350
Learning rate	0,01
Test size	0,2
Optimizer	ADAM
Error type	MSE

Como puede apreciarse hay ciertos valores o parámetros que no han sido modificados durante la fase de optimización, esto se debe a que de forma previa se realizaron multitud de pruebas y en paralelo al proceso de optimización se han seguido realizando muchas otras, mediante estas se ha podido determinar que el learning rate de 0,01 es el apropiado por ser suficientes sus correcciones y en ningún caso excesivas, por su lado el test size hace que el 20 % de los datos se destine a la fase de test, teniendo así una cantidad de valores suficiente para ratificar los resultados de cada iteración o epoch del entrenamiento.

Fuera de los ya mencionados existe otro parámetro que hasta el momento se ha mantenido invariante, este es el modelo de optimización o el **OPTIMIZER**, que ha sido en todo caso ADAM por ser este extendidamente usado, pero como aquí no existe una opción indiscutible se ha optado por probar diversos modelos. Las pruebas han arrojado los siguientes resultados:

DATOS GENERALES			
Título	Entrenamiento 18	Entrenamiento 19	Entrenamiento 20
Número de datos	100000	100000	100000
Batch size	800	800	800
Epochs	350	350	350
Learning rate	0,01	0,01	0,01
Test size	0,2	0,2	0,2
Optimizer	SGD	RMSprop	Adagrad
Error type	MSE	MSE	MSE
ÚLTIMA ITERACIÓN			
Loss	0,0090	0,0718	0,0064
Accuracy	0,9762	0,9152	0,9757
Test loss	0,0101	0,0665	0,0073
Test accuracy	0,9763	0,9131	0,9758
MEDIAS GLOBALES ENTRENAMIENTO			
MSE	0,2348	0,5721	0,2509
Error medio X0	0,0641	0,1431	0,0540
Error medio Y0	0,0891	0,2645	0,0741
TEST DE VERIFICACIÓN CONTROLADO			
VALOR ESPERADO	VALORES GENERADOS		
X0: -1	-0,8001	-1,0717	-0,9538
Y0: 1	0,7311	0,9713	0,9196
Error línea recta	0,3351	0,0772	0,0927
X0: 0	0,1763	0,1213	0,2382
Y0: -1	-1,2351	-1,0474	-1,1528
Error línea recta	0,2938	0,1302	0,2830
X0: 1	0,8790	0,9578	0,8910
Y0: -1	-1,268	-1,0889	-1,2338
Error línea recta	0,2949	0,0984	0,2580
X0: -1	-0,7997	-0,9518	-0,4739
Y0: -1	-1,3687	-1,0104	-1,3974
Error línea recta	0,4196	0,0493	0,6593
X0: 0	0,1018	-0,0623	0,0272
Y0: 1	0,7375	0,9615	0,8796
Error línea recta	0,2816	0,0733	0,1234
X0: 1	0,9901	0,9443	0,9834
Y0: 1	0,8232	0,9687	0,8877
Error línea recta	0,1771	0,0639	0,1136
Error medio recta	0,3004	0,0821	0,2550

DATOS GENERALES		
Título	Entrenamiento 21	Entrenamiento 22
Número de datos	100000	100000
Batch size	800	800
Epochs	350	350
Learning rate	No usa	0,01
Test size	0,2	0,2
Optimizer	Adadelta	Nadam
Error type	MSE	MSE
ÚLTIMA ITERACIÓN		
Loss	0,0109	0,0428
Accuracy	0,9718	0,9359
Test loss	0,0152	0,0424
Test accuracy	0,9639	0,9315
MEDIAS GLOBALES ENTRENAMIENTO		
MSE	0,3146	0,5721
Error medio X0	0,0807	0,1197
Error medio Y0	0,1090	0,1969
TEST DE VERIFICACIÓN CONTROLADO		
VALOR ESPERADO	VALORES GENERADOS	
X0: -1	-0,6720	-0,8439
Y0: 1	0,6640	0,9127
Error línea recta	0,4695	0,1789
X0: 0	0,0952	0,1069
Y0: -1	-1,2793	-1,3228
Error línea recta	0,2951	0,3400
X0: 1	0,8781	1,1183
Y0: -1	-1,0170	-1,2978
Error línea recta	0,1230	0,3204
X0: -1	-1,0526	-0,7717
Y0: -1	-1,4416	-1,2448
Error línea recta	0,4448	0,3348
X0: 0	0,0980	-0,0659
Y0: 1	0,8368	1,0089
Error línea recta	0,1904	0,0665
X0: 1	1,0365	0,8374
Y0: 1	0,8477	0,6872
Error línea recta	0,1566	0,3525
Error medio rectas	0,2799	0,2655

Observando las tablas puede apreciarse una mejora con ciertos optimizadores, pero hay algo a tener en cuenta y es la regularidad, al ejecutar varias veces los entrenamientos con estos nuevos optimizadores los resultados variaban mucho, además la diferencia obtenida en cada ejecución entre los resultados de test del entrenamiento y los de la posterior validación son notablemente diferentes, es decir no ambos tienen porque ser buenos. Si volvemos al entrenamiento 14 que era el generado con estos mismos parámetros y optimizados ADAM puede apreciarse que los resultados son bastante estables en toda fase. Por esto podría tomarse el optimizador ADAM como el más apropiado para la presente aplicación.

5.1.4. Conclusiones específicas

Como se ha podido apreciar la generación, programación y el entrenamiento de los modelos ha resultado un éxito, resolviendo con esto la planteada problemática de una forma eficaz, contundente y fiable.

Aunque resulte importante encontrar una configuración óptima hay que tener en cuenta que al partir de una base definida a partir de centenares de pruebas empleadas como estimación, todos los resultados obtenidos durante los mostrados ajustes han resultado ser más que aceptables. Dado que el sistema se ha diseñado con la intención de ayudarnos a discriminar soluciones entre varias posibles con el peor de los niveles de precisión obtenidos esto resultaría perfectamente realizable. Bien es cierto que el poder estimar los valores con mayor precisión hace que el conocer las posibles soluciones no sea tan necesario, liberándonos así de ciertos cálculos matemáticos.

Los resultados podrían afinarse aún más en busca de una configuración aún más cercana de la óptima absoluta, pero al ser realizada la optimización obligatoriamente mediante un proceso de ensayo y error esto tomaría mucho tiempo, dado que los resultados finales han resultado ser bastante buenos se ha decidido interrumpir aquí el proceso siendo estos resultados totalmente válidos para discriminar entre las diversas soluciones compatibles o modos de ensamblaje compatibles (en este caso dos).

5.1.5. Caso sin gravedad

La anterior resolución se encontraba centrada en el caso de un robot 2RPR ubicado en posición vertical y teniendo en cuenta la acción gravitatoria, gracias a esto existían ciertas diferencias en lo que a esfuerzos de los actuadores refiere entre trayectorias generadas desde un punto o desde su simétrico.

¿Que sucedería si la gravedad no fuese tomada en cuenta o si el robot se ubicase en disposición horizontal?
En ese caso como el robot 2RPR presenta dos posibles soluciones para cada par de longitudes de los actuadores, siendo estas simétricas la inercia generada en el efector final y sus esfuerzos asociados en los actuadores al seguir una trayectoria aleatoria no brindarían información suficiente para diferenciar ambos modos de ensamblaje, ya que todas las trayectorias aleatorias generadas a partir de un punto brindarían resultados idénticos a los que se generarían si se partiese de su simétrico.

Existen robots en los que las soluciones o modos de ensamblaje para unas longitudes o posiciones determinadas de las articulaciones activas no presentan una simetría exacta, y por ello la información generada al seguir una trayectoria partiendo de dos puntos de partida asociados a unas mismas longitudes sí podría ser determinante a la hora de discriminar la posición de partida, incluso sin gravedad.

6. Conclusiones y futuros proyectos

6.1. Conclusiones

El presente trabajo de investigación nos ha ayudado a solventar una problemática presente en los robots de tipo paralelo, que consiste en determinar cuál de las posibles soluciones del robot es la adoptada por éste, conociendo el desplazamiento o elongación de sus articulaciones actuadas. Este problema ha sido abordado por pocos trabajos hasta ahora, y en este trabajo proponemos una nueva solución en base a las fuerzas que producen movimiento en las articulaciones del robot, y en el movimiento que dichas articulaciones adquieren.

Hasta el momento a la hora de enfrentar la problemática existían dos tendencias. En la primera de ellas se debía conocer una solución de partida y a medida que el robot realizaba movimientos se trataba de concretar la verdadera ubicación reduciendo el error. Esto resulta problemático ya que se requiere de una primera estimación posicional.

Por otro lado, la segunda de las tendencias se centraba en emplear sensores como cámaras o sensores colocados en articulaciones pasivas, que permiten ubicar el efector final de forma directa en cada instante. El problema de esta alternativa reside en que los sensores suelen presentar error en sus medidas y si se pretende reducir este hay que invertir en un set de sensores de alta precisión y elevado coste.

El método aquí propuesto a diferencia de los anteriores no requiere de información previa, ni de sensores externos más allá de los de los sensores propioceptivos (miden cualidades del propio robot) asociados a cada uno de los actuadores. Si recordamos en el explicado método se generaban ciertas trayectorias aleatorias registrando las fuerzas (fr y fl) empleadas para seguir cada una de estas trayectorias y registrando la evolución longitudinal (L y R) que estas fuerzas generan a lo largo del tiempo. Estos datos se almacenan y concatenan generando un único vector de entrada de 4004 componentes (1001 por cada variable), tendremos tantos vectores de entrada como trayectorias se hayan ejecutado, estos datos de entrada junto a sus asociadas salidas esperadas, que no son más que los puntos de partida de dichas trayectorias nos permitirán entrenar la red. Estando esta ya entrenada solo tendremos que lanzar una trayectoría aleatoria y desconociendo el punto de partida será la propia red neuronal la que lo deduzca.

A pesar de haber realizado pruebas en un robot relativamente sencillo este proceso resulta totalmente extrapolable a otros tipos de robot, para cada robot habrá de generarse un modelo de actuación y habrá de configurarse la red para procesar los datos que dicho modelo genere al seguir ciertas trayectorias. Con estos ajustes la red debería ser capaz de entrenarse sin problemas y con ello podrá emplearse para la ya mencionada determinación posicional.

Existe también la posibilidad de que el dataset se genere a raíz de la ejecución de trayectorias en un robot real pero esto sería algo más complejo, quizás esto sería adecuado para fases de alta madurez en las que la red ya se encuentre ajustada y se quiera reentrenar esta teniendo en cuenta que el comportamiento del robot real puede variar ligeramente según la unidad.

6.2. Posibles mejoras

- Nuevas configuraciones: Probar nuevas configuraciones de la red en busca de unos resultados aún más fiables y estables.

- Nuevas arquitecturas: A pesar de haber empleado diversas arquitecturas como redes recurrentes, convolucionales o unidireccionales en las pruebas preliminares existen múltiples tipos de arquitecturas con las que se podría experimentar. En este caso se optó por una red unidereccional por ser la más afín a la aplicación deseada.

- Mayor amplitud del dataset: Hacer que el dataset contenga más de 100.000 trayectorias, esto resultaría viable pero habría de disponerse de bastante espacio en el disco duro ya que los datasets resultan pesados.

- Generación de trayectorias: En cuanto a la generación de trayectorias hasta el momento se ha optado por generar trayectorias aleatorias. Dado que lo que nos interesa es hallar el punto de partida de dichas trayectorias podría resultar interesante optar por una generación de trayectorias de tipo controlado, definiendo una o varias trayectorias invariantes las cuales se ejecuten a partir de diferentes puntos de partida, siendo estos segundos los que vayan variando, propiciando así que se vaya reuniendo la información que conformará el dataset. Con esto podría ganarse poder de diferenciación ya que una misma trayectoria ejecutada a partir de diferentes puntos generaría esfuerzos diferentes pero seguirían un patrón común, en este caso el signo del esfuerzo aportaría una información mucho más determinante que en el caso actual.

6.3. Aplicaciones similares

- Acierto total: A pesar de poder ser considerada una mejora, lo explicado a continuación presenta una utilidad diferente, la idea es lograr mejorar los resultados mediante las ya explicadas opciones con la idea de que los resultados no busquen o permitan discriminar entre varias soluciones, sino que ellos mismos sean las soluciones.

  Una de las opciones viables para realizar esto matemáticamente consiste en usar los resultados generados por la red como semillas iniciales del método Newton-Raphson, siendo este un método iterativo que nos permitiría resolver la cinemática directa convergiendo a la verdadera configuración con un error nulo.

- Nuevos robots: Dada la precisión obtenida no sería problema el replicar este sistema para otros robots de tipo paralelo. A pesar de presentar estos mayor número de soluciones resultaría interesante ver donde están los límites del método en caso de existir estos. A futuro los límites vendrán totalmente definidos por la optimización de la red y no tanto por el número de soluciones de los robots, ya que los datos de trayectorias 3D y sus respectivos puntos de partida podría incluir incluso más información que los hasta ahora vistos.

Referencias

[1] Robot cirujano con un paciente de cirugía en la mesa de operaciones. https://www.istockphoto.com/es/foto/robot-cirujano-hace-un-paciente-de-cirug%C3%ADa-en-la-mesa-de-operaciones-gm1127504033-297185439. Accessed: 2023-09-14.

[2] Robot ur5 de universal robots. https://www.interempresas.net/Robotica/Articulos/155087-Automatizar-el-proceso-de-mecanizado-para-mejorar-la-productividad.html. Accessed: 2023-09-14.

[3] C-9879 chasis universal para robots 2wd. https://fadisel.com/es/robots-para-arduino/1691-chasis-universal-para-robots-2wd.html. Accessed: 2023-09-14.

[4] Motor dc de engranaje de 12 dientes y dos velocidades para taladradora eléctrica industrial. https://www.amazon.es/Engranaje-velocidades-taladradora-el%C3%A9ctrica-Industrial/dp/B08DTBR9JW. Accessed: 2023-09-14.

[5] Sensores de un robot. https://prezi.com/p/kcpnlthtxjdy/sensores-de-un-robot/. Accessed: 2023-09-14.

[6] Armario de control de robot, utilizado en buenas condiciones, rc5-hme4bb-cp, 410000-9980. https://es.aliexpress.com/item/1005001927968148.html. Accessed: 2023-09-14.

[7] Pccase ep-500 unidad de - fuente de alimentación. https://www.amazon.es/PC-Case-Gear-EP-500-alimentaci%C3%B3n/dp/B012OETUXM. Accessed: 2023-09-14.

[8] Ruedas que se convierten en patas: los robots cambiarán para siempre. https://www.elespanol.com/omicrono/tecnologia/20201103/idean-ruedas-convierten-patas-robots-cambiaran-siempre/532197722_0.html. Accessed: 2023-09-14.

[9] Brazo robótico serie. https://crearticos.blogspot.com/p/blog-page.html. Accessed: 2023-09-14.

[10] Pinzas y gripper para robots industriales. https://innovacion-tecnologia.com/robotica/pinzas-y-gripper-para-robots-industriales/. Accessed: 2023-09-14.

[11] Sistemas de comunicacion del robot. https://www.researchgate.net/figure/Figura-5-Sistemas-de-Comunicacion-del-Robot_fig4_236023822. Accessed: 2023-09-14.

[12] El "perro robot": un robot autónomo con patas mapea la radiación en torno a la unidad 4 de chernóbil. https://www.iaea.org/es/bulletin/robots-ia-y-modelos-3d. Accessed: 2023-09-14.

[13] Estructura de un robot industrial. http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/robotica/sistema/morfologia.htm. Accessed: 2023-09-14.

[14] Packaging parallel robot. https://www.turbosquid.com/es/3d-models/packaging-parallel-robot-3d-model-1587405. Accessed: 2023-09-14.

[15] Partes principales de la estructura de un robot manipulador típico. https://arvc.umh.es/label/prac1_es.pdf. Accessed: 2023-09-14.

[16] Espacio de trabajo del prototipo araba ii. https://www.interempresas.net/Alimentaria/Articulos/33092-Manipuladores-paralelos-para-operaciones-de-pick-and-place.html. Accessed: 2023-09-14.

[17] Robot irb140. https://www.youtube.com/watch?v=Q0epkrwlr88. Accessed: 2023-09-14 Minuto 3:16.

[18] Ilustración digital: Realidad vs ia. https://www.viajes24horas.com/se-esta-perdiendo-lo-humano-en-favor-de-lo-tecnologico/. Accessed: 2023-09-14.

[19] Robot paralelo 5r. https://arvc.umh.es/label/prac1_es.pdf. Accessed: 2023-09-14.

[20] Plataforma de stewart. https://addi.ehu.es/bitstream/handle/10810/29472/NICOL%C3%81S%20DEZA%20D%C38DAZ%2C%20MEMORIA%20TFG.pdf?sequence=1&isAllowed=y. Accessed: 2023-09-14.

[21] Modos de ensamblado robot paralelo 5r. https://ruc.udc.es/dspace/bitstream/handle/2183/31395/2022_Peidro_Adrian_Practicas_de_control_de_robots_paralelos.pdf?sequence=3&isAllowed=y. Accessed: 2023-09-14.

[22] Overfitting y underfitting machine learning. https://www.directindustry.es/prod/gridbots-technologies-private-limited/product-169302-1930631.html. Accessed: 2023-09-14.

[23] Overfitting y underfitting machine learning. https://carlosjuliopardoblog.wordpress.com/2018/01/04/overfitting-underfitting/. Accessed: 2023-09-14.

[24] Definición inteligencia artificial. https://es.wikipedia.org/wiki/Inteligencia_artificial. Accessed: 2023-09-14.

[25] Componentes de los sistemas robóticos. http://www.udesantiagovirtual.cl/moodle2/mod/book/tool/print/index.php?id=24908. Accessed: 2023-09-14.

[26] Apuntes de la asignatura cinemática de robots del máster en robótica de la universidad miguel hernández. https://www.umh.es/contenido/Estudios/:asi_m_4357/datos_es.html. Accessed: 2023-09-14.

[27] Apuntes de la asignatura dinámica y simulación de robots manipuladores del máster en robótica de la universidad miguel hernández. https://www.umh.es/contenido/Estudios/:asi_m_4358_CSig/datos_es.html. Accessed: 2023-09-14.

[28] Adrien Koessler, Alexandre Goldsztejn, Sébastien Briot, and Nicolas Bouton. Dynamics-based algorithm for reliable assembly mode tracking in parallel robots. *IEEE Transactions on Robotics*, 36(3):937–950, 2020.

[29] F. C. Can, M. Hepeyiler, and Ö. Başer. A novel inverse kinematic approach for delta parallel robot. *International Journal of Materials, Mechanics and Manufacturing*, 6 Nº5:321–326, 2018.

[30] H. Saafi, M. A. Laribi, and S. Zeghloul. Forward kinematic model improvement of a spherical parallel manipulator using an extra sensor. *Mechanism and Machine Theory*, pages 102–119, 2015.

[31] R. F. Abo-Shanab. An efficient method for solving the direct kinematics of parallel manipulators following a trajectory. *Journal of Automation and Control Engineering*, 2(3):228–233, 2014.

[32] P. Wenger and D. Chablat. A review of cuspidal serial and parallel manipulators. *Journal of Mechanisms and Robotics*, 15(4):040801, 2023.

[33] J. Villa : E. Yime, J. Roldán. Computed torque control of a 2-rr planar parallel robot. *Dinámica paralela*, 15:85–95, 2017.

[34] A. Peidró, A. Quijada-Fernández, D. Úbeda, R. Puerto, L. Payá, and Ó. Reinoso. Imperfect dynamic modeling of parallel robots eases the crossing of type-ii singularities. *2022 IEEE 17th International Conference on Advanced Motion Control (AMC)*, pages 124–131, 2022.