

TRABAJO FIN DE MASTER

CONTROL CINEMATICO DE ROBOT HIPERREDUNDANTE MEDIANTE RED NEURONAL



“En tiempos oscuros, la esperanza es algo que te das a ti mismo. Ese es el significado de la verdadera fuerza interior”

- Iroh

AGRADECIMIENTOS

En primer lugar, quiero agradecerle a mi director, Antonio Barrientos, la paciencia que ha mostrado conmigo y el entusiasmo con el que ha enfocado este trabajo. Ha habido muchos momentos en los que parecía que habíamos llegado a un callejón sin salida, pero en cada uno Antonio tuvo una reunión conmigo y juntos encontramos una salida o un atajo.

Quiero dar las gracias a los estudiantes de doctorado que trabajan en el laboratorio donde he hecho las pruebas, especialmente a David, Chrystian y Silvia. Gracias por echarle 80 vistazos a mi robot y a mi código cuando no funcionaban, y muchas gracias por el apoyo. Gracias también a Elena, Jaime y Diego, estudiantes de TFG y TFM que han hecho otros trabajos sobre el mismo tema recientemente, por aclararme aspectos de sus trabajos que no entendía

Gracias a todos mis amigos, en especial a Miguel, Lucía, María y Raquel, por el apoyo prestado durante este trabajo y por tratarme como a vuestra familia, aunque no compartamos sangre. Gracias a mi amiga Lucía por discutir conmigo qué red neuronal era mejor para mis datos y aclararme algunos conceptos. Quiero darle las gracias también a Asbjørn por escuchar atentamente mis problemas y sugerir soluciones creativas; y a Miguel porque pasea a mi perro para que yo pueda escribir y entregar a tiempo.

Y por último, quiero agradecerle a mi hermana Estefanía que me haya acompañado al otro lado del teléfono durante todo este trabajo. Gracias por escucharme y por animarme. Gracias también por todos los consejos y las discusiones sobre programación y matemáticas. Y gracias a Abril, por ser el mejor perro del mundo y acompañarme durante todas las horas que paso trabajando frente al ordenador.

RESUMEN

Los avances que venimos viendo en las últimas décadas demuestran que la robótica va mucho más allá de los brazos robóticos que es habitual ver en la industria. Cada vez existen más robots diseñados para áreas como por ejemplo la medicina, la seguridad o los robots personales, capaces de mejorar la calidad de vida de las personas o incluso salvar vidas. Los robots de tipo continuo hiperredundantes como el robot Pylori-I que se emplea en este proyecto, abren la puerta a la posibilidad de automatizar tareas de inspección como endoscopias o búsqueda de supervivientes entre escombros en tareas de rescate; reduciendo así los errores humanos o los riesgos que corren los rescatadores. Sin embargo, este tipo de aplicaciones requieren gran precisión y estos robots presentan gran complejidad en su control debido al gran número de parámetros que deben considerarse en su modelo cinemático para que este tenga un buen desempeño.

De entre todos los métodos que se han planteado para resolver este problema, las redes neuronales cuentan con la ventaja de que funcionan como una caja negra en el sentido de que no requiere especificar qué variables que influyen en el modelo, más allá de las de entrada y salida, ni su relevancia para obtener buenos resultados; por lo que son una buena alternativa cuando encontrar las ecuaciones que describen el movimiento del robot resulta demasiado complejo.

Este trabajo pretende comprobar si es viable emplear redes neuronales para mejorar el desempeño del método de control previamente implementado en Matlab para el robot Pylori-I del CAR (Centro de Automática y Robótica) de la ETSII (Escuela Técnica Superior de Ingenieros Industriales de la UPM), que se muestra en la figura 0.1 y servir de base para futuras mejoras de su control y del de otros robots del CAR usando este método.



Figura 0.1: Robot Pylori-I del CAR. (Muñoz Sánchez, 2022a)

Para ello, el trabajo comienza presentando el diseño y funcionamiento del robot Pylori-I con el que se va a trabajar, tal y como lo construyó Elena Muñoz (Muñoz Sánchez, 2022a). Este se trata de un robot hiperredundante de tipo serpantino. Es decir, que si bien sus eslabones son elementos rígidos, su alto número y su disposición a lo largo del brazo robótico tal que un solo actuador mueve varios de ellos cada vez, hacen que el comportamiento del robot sea como el de uno continuo. En el caso del Pylori, los eslabones son discos que, debido al acabado en cuña en una de sus caras, pueden pivotar en una única dirección respecto al anterior al colocarlos sucesivamente. Esos se colocan repartidos en cuatro secciones de distinta longitud, donde todos los discos se colocan de tal modo que solo pueden pivotar en una de dos direcciones perpendiculares distintas para cada sección, lo que otorga dos grados de libertad a cada una y 8 a todo el robot como se aprecia en la figura 0.2. En el tercer capítulo se exponen en detalle tanto los elementos que lo componen como la forma en la que se logra su accionamiento mediante

cables tensores con motores paso a paso y el software disponible para su control al inicio del proyecto.

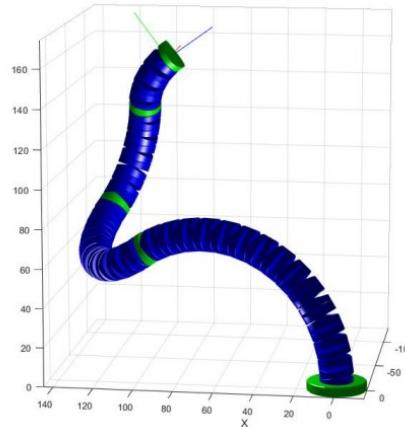


Figura 0.2: Simulación del comportamiento del robot Pylori-I. (Bravo Algaba, 2021)

Debido a una caída sufrida por el robot antes del comienzo del proyecto, se continúa explicando las reparaciones y mejoras que se han hecho al hardware y el software para su puesta a punto.

El siguiente paso en el trabajo que se da una vez que se ha devuelto el robot a un estado funcional, es la toma de datos. Se expone como, tras elegir entre varios sistemas de captura de movimiento, se ha procedido a medir el comportamiento del robot cuando se mueven los motores directamente mediante comandos enviados a través de Matlab al serial de Arduino. Para ello se han tomado medidas con Optitrack, del que se explica cómo distribuido el hardware, como se ha calibrado y puesto a punto, y cómo se ha conectado a Matlab mediante ROS (Robot Operating System) para poder recibir y almacenar los datos en un formato más cómodo.

A continuación, se enumeran y justifican los pasos tomados para procesar, limpiar y almacenar los datos obtenidos para que tengan suficiente calidad como para que resulte viable entrenar con ellos dos redes neuronales; una para mejorar el modelado de la cinemática directa del robot y otra para mejorar el de la cinemática inversa.

Posteriormente, indica el tipo de red neuronal empleado y se detallan los parámetros y características que se han elegido para implementarla y las razones para cada una. Sin embargo, la hiperredundancia del robot implica que los datos con los que se ha entrado la cinemática directa no se pueden emplear directamente para entrenar la cinemática inversa, por lo que se desarrolla el método empleado para obtener un nuevo conjunto de datos válido para esta empleando la primera red neuronal.

Finalmente, se presentan y analizan los resultados del trabajo en el apartado de conclusiones. De estos extrae que es posible emplear redes neuronales para mejorar el control del robot, no solo para el control del extremo, sino también en un futuro para el control de la posición de todo el brazo robótico. Además, hacerlo de esta forma permitiría reentrenar las redes con facilidad en caso de añadir mejoras o modificaciones en el robot. Sin embargo, también queda patente la dificultad de encontrar un buen método de control si no existe un cierto grado de robustez en el diseño y estructura del robot, y la gran susceptibilidad de este método ante errores no sistemáticos en comportamiento mecánico del robot. En este apartado también plantean las posibles líneas futuras del trabajo.

La memoria finaliza con un breve análisis del impacto del proyecto; y con el presupuesto y planificación temporal (constituida por la EDP y el diagrama de Gantt), el glosario y el enlace al repositorio del código desarrollado para el proyecto, incluidos en los anexos.

Códigos UNESCO

330491 - Robótica 331101 - Tecnología de la automatización 331102 - Ingeniería de control
120304 - Inteligencia artificial

Palabras clave

Automática, control, robótica, robots hiperredundantes, inteligencia artifical, red neuronal, Arduino.

ÍNDICE

AGRADECIMIENTOS	III
RESUMEN	v
Códigos UNESCO	VII
Palabras clave	VII
ÍNDICE	IX
ÍNDICE DE TABLAS	xii
ÍNDICE DE FIGURAS	xiv
1. INTRODUCCIÓN	1
1.1. Motivación	1
1.2. Objetivos	1
1.3. Estructura del proyecto	2
2. ESTADO DEL ARTE	3
2.1. Clasificación de robots hiperredundantes	3
2.2. Antecedentes	4
3. FUNCIONAMIENTO DEL PYLORI-I Y CÓDIGO PREVIO	6
3.1. Diseño mecánico	6
3.1.1. Brazo robótico	6
3.1.2. Base y armario eléctrico	8
3.2. Código previo	10
4. REPARACIÓN Y MEJORAS	12
4.1. Reparación y mejoras del diseño mecánico	12
4.2. Aportaciones al código	15
5. TOMA DE DATOS	18

ÍNDICE

5.1. Herramientas	18
5.2. Diseño de los experimentos	23
5.3. Procesamiento y limpieza de datos	24
6. REDES NEURONALES	28
6.1. Cinemática directa	28
6.2. Cinemática inversa	32
7. CONCLUSIONES Y LÍNEAS FUTURAS	34
8. IMPACTO DEL TRABAJO	36
9. BIBLIOGRAFÍA	37
ANEXO I: PLANIFICACIÓN TEMPORAL Y ESTUDIO ECONÓMICO	40
A. Presupuesto	40
B. Estructura de Descomposición del Proyecto (EDP)	42
C. DIAGRAMAS DE GANTT	43
ANEXO II: GLOSARIO Y ABREVIATURAS	44
ANEXO III: CÓDIGO	45

ÍNDICE DE TABLAS

3.1. Valores los parámetros de Arduino. Modificado de (Muñoz Sánchez, 2022a).	10
3.2. Descripción de los comandos disponibles. Modificado de (Muñoz Sánchez, 2022a).	10
A.1. Costes del Proyecto.	41

ÍNDICE DE FIGURAS

0.1. Robot Pylori-I del CAR. (Muñoz Sánchez, 2022a)	v
0.2. Simulación del comportamiento del robot Pylori-I. (Bravo Algaba, 2021)	vi
2.1. Diagrama de Venn para la clasificación de robots. (Martín-Barrio et al., 2018) . .	3
2.2. Robot MACH del CAR. (Márquez Alcolea, 2021)	4
2.3. Robot Kyma del CAR. (Martínez Martín, 2017)	5
2.4. Robot blando actuado mediante SMA del CAR. (Guzmán Merino, 2020)	5
3.1. (Muñoz Sánchez, 2022a)	6
3.2. Modelo TCPJ. Disposición de los discos. (Muñoz Sánchez, 2022a)	7
3.3. Disco de final de sección.	7
3.4. Secciones del brazo robótico.	8
3.5. Montaje de los motores y las poleas en los soportes. (Muñoz Sánchez, 2022a) . .	8
3.6. Disposición de los motores	9
3.7. Interior del armario eléctrico. (Muñoz Sánchez, 2022a)	9
4.1. Detalles del daño en los soportes tras la caída.	12
4.2. Soportes nuevos. Recién impreso en resina a la izquierda e impreso en PLA y listo para montar, a la derecha.	13
4.3. Cable y conectores para la conexión entre los motores y el armario. (Muñoz Sánchez, 2022a)	13
4.4. Robot reparado.	14
4.5. Polea rota.	14
4.6. Nuevo diseño de la polea.	15
4.7. Disposición de la polea en el soporte.	15
4.8. Simulación fallida de la posición del robot.	16
4.9. Simulación del movimiento del robot para una trayectoria circular.	17
5.1. Marcadores y cámaras del Optitrack.	18
5.2. Montaje de los marcadores en el extremo del brazo robótico.	19
5.3. Brazo robótico recalibrado con el peso de los marcadores.	19
5.4. Distribución de las cámaras para la toma de datos.	20

ÍNDICE DE FIGURAS

5.5. Toma de puntos para la calibración	20
5.6. Calibración del Optitrack.	21
5.7. Herramientas de calibración.	21
5.8. Visualización de los trackables y su medida en Optitrack.	22
5.9. Panel de configuración de (OptiTrack, 2012).	22
5.10. Esquema de los sistemas de referencia	25
5.11. Distribución de los datos en el plano XY	26
6.1. Red neuronal de tipo MLP con una capa oculta. (Pedregosa et al., 2011)	29
6.2. Efecto del coeficiente de la regularización L^2 sobre la tendencia de la red a cometer overfitting. (Goodfellow et al., 2016)	29
6.3. Evolución del error en el training set y el validation set sin early stopping.(Goodfellow et al., 2016)	30
6.4. Validación cruzada: particiones de los datos para entrenar la red neuronal. (Pedregosa et al., 2011)	31
6.5. Resultado de entrenar la red neuronal para la cinemática directa. Cada punto azul representa un punto y la línea negra discontinua representa la recta predicción=real.	31
6.6. Malla de puntos de 2 mm de los que se generan datos para entrenar la cinemática inversa, con el punto de partida del algoritmo indicado en naranja.	32
6.7. Resultado de entrenar la red neuronal para la cinemática inversa. Cada punto azul representa un punto y la línea negra discontinua representa la recta $x=y$	33
B.1. Estructura de Descomposición del Proyecto (EDP)	42
C.2. Diagrama de Gantt del proyecto.	43

1. INTRODUCCIÓN

1.1. Motivación

Desde que se empezaron a implantar en la industria hace décadas, ha quedado patente la capacidad de los robots de realizar tareas que resultan peligrosas, muy pesadas o tediosas para un humano o que requieren una precisión muy alta. Sin embargo, los avances recientes en robótica ponen de manifiesto que estas capacidades pueden aplicarse a otras muchas otras áreas relevantes como por ejemplo la medicina, la seguridad, las relaciones públicas, la limpieza o la construcción. En este sentido, los robots de tipo continuo hiperredundantes como el que se emplea en este proyecto, presenta varias ventajas frente a los robots de tipo discreto tradicionalmente empleados en la industria, que los hacen ideales para tareas de inspección en áreas como la medicina (endoscopios o laparoscopios), la investigación o tareas de defensa y rescate (búsqueda de supervivientes entre escombros). Su alto número de grados de libertad y su alto grado de flexibilidad permiten controlar no solo la posición del extremo si no la del robot entero, permitiendo el acceso a lugares estrechos sin dañar el entorno. Sin embargo, esto requiere un alto grado de control y precisión sobre el control del robot, que debido a la complejidad de su cinemática, no suele resultar sencillo.

Existen muchas posibles soluciones para resolver la cinemática que van desde la modelización de la estructura del robot mediante ecuaciones hasta distintos métodos de aprendizaje automático y combinaciones de ambas. Además, el método más adecuado puede variar significativamente en función del mecanismo del robot. De modo que no es extraño que para el mismo robot se planteen distintas opciones o modificaciones. Por lo que en este tipo de proyectos es especialmente relevante implementar un buen sistema de transferencia de conocimiento, y realizar un diseño que permita mejoras en etapas posteriores. Si bien no se trata en absoluto de una necesidad exclusiva de este tipo de proyectos, y todas las mejoras que se consigan en este sentido pueden extrapolarse a otros proyectos.

En este marco, este Trabajo Fin de Grado pretende plantear una mejora del control del robot Pilory-I del CAR (Centro de Automática y Robótica) de la UPM partiendo de dos proyectos previos sobre su diseño y montaje (Muñoz Sánchez, 2022b) y sobre el planteamiento de su cinemática mediante un algoritmo genético (Bravo Algaba, 2022).

1.2. Objetivos

El objetivo principal de este Trabajo Fin de Máster es entrenar una red neuronal que permita mejorar el control desde Matlab de un robot hiperredundante de tipo serpantino partiendo de un proyecto previo. Se pretende que la red permita controlar la posición del extremo con mayor precisión que el algoritmo basado en la cinemática del robot implementado previamente al proyecto.

Para conseguirlo, se plantean los siguientes objetivos secundarios como hitos intermedios en el desarrollo del proyecto que permiten evaluar su avance y desempeño:

1. Reparar los daños en el hardware del robot tras la caída sufrida antes del comienzo del proyecto. E implementar mejoras si es posible en aquellos aspectos del diseño que lo requieran.
2. Reparar y mejorar el software de control del robot disponible al inicio del proyecto, de

1. INTRODUCCIÓN

modo que el robot sea funcional y el código se pueda ejecutar en un futuro desde las nuevas versiones de Matlab.

3. Seleccionar y disponer un método de captura de movimiento válido para la toma de datos
4. Tomar suficientes datos del comportamiento del robot ante los posibles comandos de los motores, tales que permitan entrenar las redes neuronales.
5. Entrenar una red neuronal que mejore el desempeño de la cinemática directa.
6. Entrenar una red neuronal que mejore el desempeño de la cinemática inversa.

Además, se espera que sirva de punto de partida para conseguir en un futuro un control más preciso de la posición de todo el robot, aprovechando su calidad de hiperredundante para aplicaciones como las mencionadas en el apartado anterior.

1.3. Estructura del proyecto

Esta memoria se ha estructurado siguiendo el desarrollo de los objetivos mencionados a lo largo del proyecto. A continuación se explica brevemente como se han ordenado los distintos capítulos y su contenido para ello.

En primer lugar, se introduce brevemente el estado del arte. Con ello se introducen algunos conceptos relevantes para la mejor comprensión del proyecto y mostrando algunos ejemplos tanto de robots hiperredundantes como de los distintos métodos que existen para su control.

A continuación, se expone el funcionamiento del hardware y el software que se heredan de proyectos anteriores como punto de partida del proyecto, procediendo en el tercer capítulo a explicar las reparaciones y mejoras que se han realizado.

El cuarto capítulo comienza con la explicación de como se ha elegido, dispuesto y preparado el sistema para medir el comportamiento del robot. En él también se explican cómo se han diseñado y ejecutado los experimentos para la toma de datos, así como cómo se han procesado y limpiado los datos obtenidos.

Entonces se procede a explicar como se han entrenado las bases de datos a partir de los datos disponibles, tanto en el caso de la cinemática directa como de la indirecta.

Finalmente, se discuten los resultados y se analizan las conclusiones más relevantes del proyecto, así como cuáles son sus posibles líneas futuras. Por último se incluyen anexos con el cronograma, la planificación y el presupuesto, además de un breve glosario y los enlaces al código desarrollado a lo largo del proyecto.

2. ESTADO DEL ARTE

2.1. Clasificación de robots hiperredundantes

Como se ha mencionado, la robótica hiperredundante permite llevar las ventajas de la robótica a nuevas áreas y funcionalidades. Los robots hiperredundantes se diferencian de los convencionales en que tienen un número de grados de libertad (GdL) mayor del necesario para controlar la posición del extremo del robot. Esto abarca un conjunto muy amplio de robots, ya que existen muchas formas distintas de diseñar un robot para lograr esta cualidad. En este apartado se presenta brevemente los distintos tipos en los que se puede clasificar un robot redundante según (Martín-Barrio et al., 2018), como se muestra en la figura 2.1.

- **Según el número de GdL:** Se puede considerar hiperredundante cualquier robot que tenga al menos el doble de grados de libertad que los estrictamente necesarios para posicionar el extremo, por lo que el rango de GdL que puede tener un robot de este tipo es muy amplio.
- **Según el número de articulaciones:** Pueden ser o bien discretos (tienen uniones rígidas, por lo que cada grado de libertad se acciona desde un solo punto del robot) o continuos (en los que un actuador puede provocar que toda una sección del robot se deforme de forma continua, controlando así un grado de libertad). Hay ciertos robots, como es el caso del Pylori-I, que si bien estrictamente hablando están compuestos por elementos rígidos, su estructura y accionamiento está diseñada de tal forma que presentan un movimiento similar a los robots continuos y se pueden modelizar como tal. Por esta razón se los suele clasificar como de tipo continuo
- **Según funcionalidad:** Pueden ser manipuladores (emplean una herramienta para manipular objetos y suelen estar montados sobre una base fija), móviles (capaces de desplazarse), o híbridos.
- **Según el tipo de actuadores:** Existe gran variedad de actuadores que se pueden emplear para accionar un robot, además de motores, tales como actuadores neumáticos, hidráulicos, SMA (Aleaciones con Memoria de Forma), piezoelectrónicos o músculos artificiales.

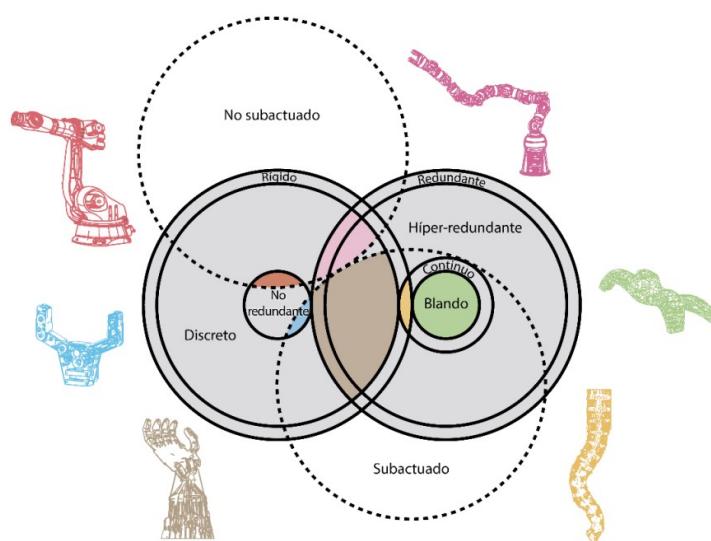


Figura 2.1: Diagrama de Venn para la clasificación de robots. (Martín-Barrio et al., 2018)

- **Según sus materiales:** Fundamentalmente rígido o blandos. Cabe mencionar que los robots blandos suelen ser de tipo continuo, al igual que el Pylori.

El robot empleado en este proyecto se puede clasificar como un robot manipulador, hiperredundante, de tipo continuo. Y si bien es un robot actuado mediante motores, dado que los elementos que realmente transmite el movimiento son los cables que tensan o destensan los motores y que esto afecta considerablemente al movimiento del robot, se lo suele clasificar como actuado mediante cables.

2.2. Antecedentes

Como ya se ha mencionado, las aplicaciones más habituales de los robots hiperredundantes son en tareas de inspección para acceder a lugares delicados o de difícil acceso. Así, existen ejemplos de robot de este tipo dedicados a tareas de inspección, puesta a punto de turbinas de gas de aviones o cohetes (Dong et al., 2017), o de búsqueda y rescate como el expuesto en (Wolf et al., 2003). En el campo de la medicina existen también varios ejemplos empleados en endoscopias para distintas áreas de la medicina como la neumología (Mitros et al., 2021) o la cirugía mínimamente invasiva para eliminación de tumores cerebrales (Granna et al., 2018).

Además, en los últimos años se han desarrollado varios robots hiperredundantes en el Centro de Automática y Robótica (CAR) de la ETSII aparte del Pylori-I:

- **MACH:** Con un funcionamiento muy similar al del Pylori a pesar de ser discreto, es un robot inspirado por la trompa del elefante y accionado mediante cables. Fue construido por Iván Rodríguez (Rodríguez Rodríguez, 2019) y después David Márquez Alcolea (Márquez Alcolea, 2021) y Diego Cerrillo Vacas (Cerrillo Vacas, 2022) contribuyeron a su control.

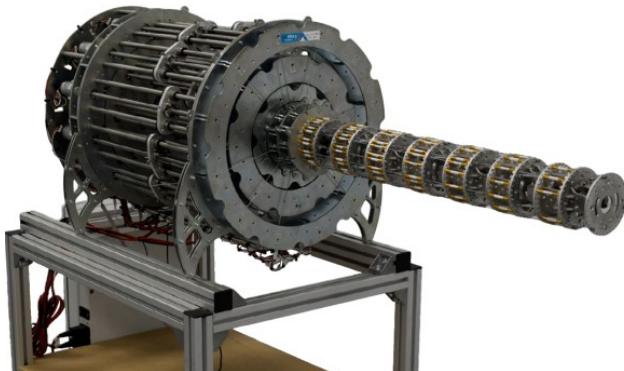


Figura 2.2: Robot MACH del CAR. (Márquez Alcolea, 2021)

- **Kyma:** Robot blando y continuo diseñado y construido por Cecilia Martínez Martín (Martínez Martín, 2017). También está accionado por cables controlados por motores, pero estos actúan sobre fuelles, aportando 3 GdL para cada una de las cuatro secciones.

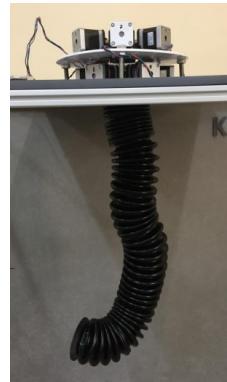


Figura 2.3: Robot Kyma del CAR. (Martínez Martín, 2017)

- **Robot blando accionado con SMA:** Se trata de otro robot de tipo blando y continuo desarrollado por Miguel Guzmán Merino (Guzmán Merino, 2020). Está fabricado con silicona y accionado con aleaciones con memoria de forma (SMA).



Figura 2.4: Robot blando actuado mediante SMA del CAR. (Guzmán Merino, 2020)

3. FUNCIONAMIENTO DEL PYLORI-I Y CÓDIGO PREVIO

Como ya se ha mencionado, el objetivo principal de este proyecto es entrenar una red neuronal que ayude a mejorar el control de unos de los robots hiperredundantes accionados con cables del CAR. En concreto del robot conocido como Pylori I de tipo Serpentino. Para lograr este objetivo, sobre todo teniendo en cuenta que el robot había sufrido una caída que había causado daños justo antes de comenzar el proyecto, el primer paso fue comprender el funcionamiento y diseño de robot. Este paso resultó imprescindible para poder repararlo, ponerlo en funcionamiento y entender los requisitos que debía cumplir la red neuronal y los datos para entrenarla.

Así pues, en este apartado se presenta el diseño y funcionamiento del robot Pylori al inicio del proyecto, tal y como lo diseñó mi compañera Elena Muñoz Sánchez (Muñoz Sánchez, 2022a) junto con el código escrito para su control y modelizado entre ella y Jaime Bravo Algaba (Bravo Algaba, 2021). Para entender en mayor detalle el diseño y el código puede referirse a sus respectivos trabajos.

3.1. Diseño mecánico

Cómo se aprecia en la figura 3.1, la estructura del Pylori está compuesta de dos partes: un brazo de un brazo robótico compuesto por discos pivotantes y una base posterior que contiene los motores. Estos últimos, conectados mediante cables al armario eléctrico que permite alimentar y controlar el robot.

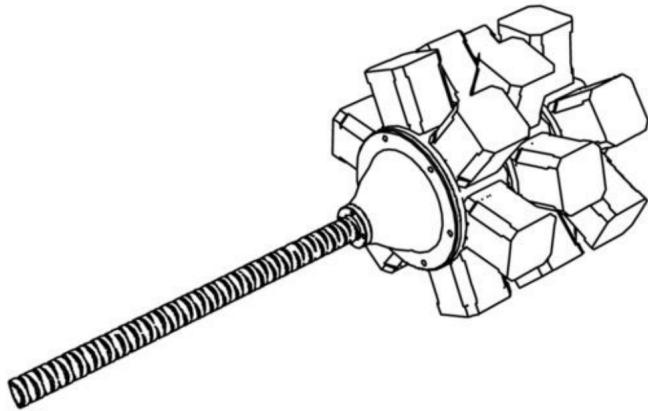


Figura 3.1: (Muñoz Sánchez, 2022a)

A continuación se explica en más detalle el diseño de estas partes.

3.1.1. Brazo robótico

El brazo robótico, de una longitud de 260 mm, está compuesto por 52 discos y por los 16 cables que controlan su movimiento. A pesar de ser elementos rígidos, estos discos son los responsables

de que el robot presente una cinemática propia de los de tipo continuo gracias a su montaje de tipo *twin-pivot complaint joints* (Barrientos-Diez et al., 2021).

La implementación de este modelo en el caso del Pylori se consigue mediante el diseño y disposición de los discos. Como se puede apreciar en la figura 3.2, su forma permite que, al colocar dos discos sucesivos con un desfase de 90° entre sí, existan dos puntos de apoyo diametralmente opuestos que generan un único eje sobre el que un disco puede girar un máximo de 0.23 radianes respecto al otro.

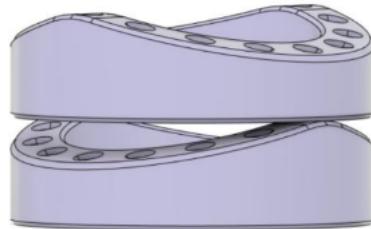


Figura 3.2: Modelo TCPJ. Disposición de los discos. (Muñoz Sánchez, 2022a)

Al colocar más discos siguiendo este patrón, se consigue un segmento del brazo capaz de girar en dos direcciones perpendiculares entre sí. Es decir, que el segmento cuenta con dos grados de libertad definidos por dos ejes de giros perpendiculares entre sí.

El accionamiento de este movimiento se consigue mediante cables que pasan por los orificios de los discos que se aprecian en la figura 3.2 hasta llegar al último disco del segmento. Este disco presenta orificios adicionales en los laterales, tal y como muestra la figura 3.3, para poder pasar los extremos de los cables por ellos y hacer un nudo, de modo que queden fijados a este disco sin afectar el movimiento del siguiente.

Así, al tirar desde el otro extremo de un cable que pasa por alguno de los dos orificios situados en cualquiera de los ejes de giro de los discos, se consigue que el segmento entero gire (con un ángulo aproximadamente constante) en torno al eje contrario en el sentido del cable tensionado. Por lo que bastan 4 cables para controlar los dos grados de libertad de este segmento.

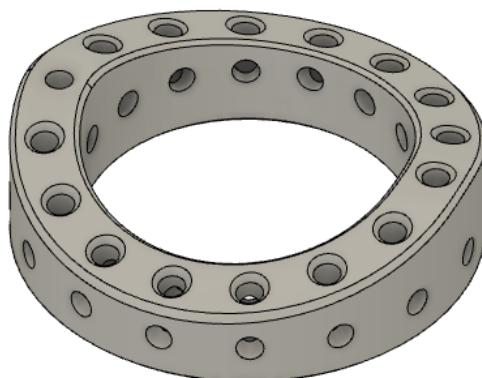


Figura 3.3: Disco de final de sección.

3. FUNCIONAMIENTO DEL PYLORI-I Y CÓDIGO PREVIO

Los discos del robot se han distribuido en 4 de estos segmentos, denominados secciones en adelante. Estas secciones constan de 20, 12, 12 y 8 discos respectivamente, como se aprecia en la figura 3.4, donde se ha marcado el último disco de cada sección en verde. Con el fin de dotar al robot de más grados de libertad, cada una de las secciones está montada con un desfase de $\pi/8$ rad entre sí. Lo que dota al brazo de 8 grados de libertad controlados mediante 16 cables.

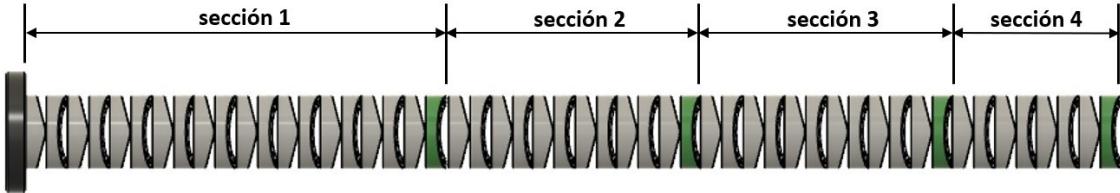


Figura 3.4: Secciones del brazo robótico.

Para que los cables fuesen capaces de soportar tanto que se les someta a tensiones y torsiones al tirar de ellos, como el rozamiento con los discos, se han empleado dos tipos distintos de hilo de pescar.

3.1.2. Base y armario eléctrico

Tratándose de un robot de tan poco diámetro y tamaño con un comportamiento de robot continuo, colocar el peso de los actuadores en el propio brazo, tal y como se suele hacer en los motores discretos más robustos, es inviable. Así pues, los cables se pasan no solo por la sección que controlan, sino también por los orificios diseñados para ello de los discos de las secciones previas hasta llegar hasta los motores situados en la parte posterior.

Los 16 motores paso a paso se montan sobre tres piezas hexagonales a las que denominaremos soportes, y en sus ejes se acopla una polea, como se observa en la figura 3.5. El otro extremo del cable se ata a la polea, de modo que al girar el eje en una dirección u otra se enrolle o desenrolle el cable, acortando o alargando así su longitud y accionando el brazo robótico.



Figura 3.5: Montaje de los motores y las poleas en los soportes. (Muñoz Sánchez, 2022a)

Los tres soportes se atornillan entre sí con un desfase de 20 grados cada uno para evitar en la medida de lo posible que los cables se crucen o enreden afectando a su movimiento. Teniendo en

cuenta que queda espacio sin usar para dos motores más en caso de que en un futuro se quisiera añadir utilaje al extremo, los motores se disponen como en la figura 3.6.

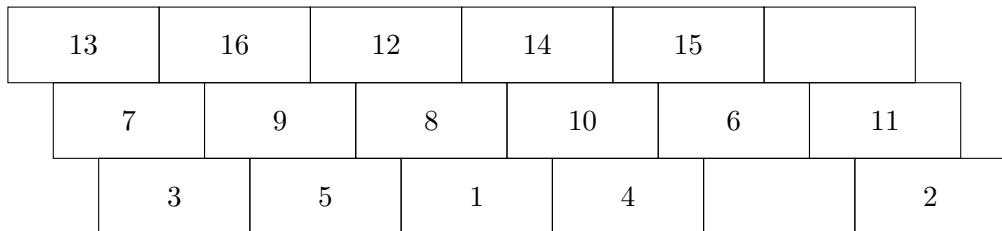


Figura 3.6: Disposición de los motores

Sabiendo esto, se establece como referencia el motor 1, que se coloca siempre en posición vertical cuando se emplea el robot. Así, en adelante, el eje Z es el que viene marcado por la longitud del brazo en su posición de origen, siendo el eje y el vertical. O lo que es lo mismo, el que sigue la posición del motor 1.

Para su control, los motores paso a paso (PaP) se conecta al armario eléctrico. En el interior del armario se establece la conexión de cada motor con un driver TB6600 siguiendo la distribución que se muestra en la figura 3.7.

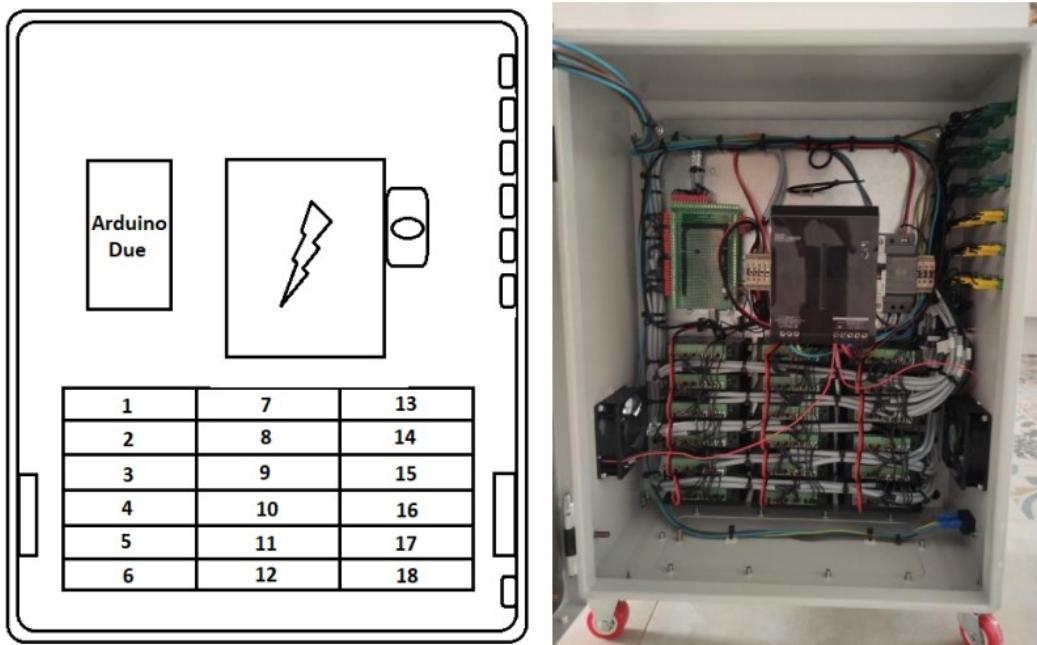


Figura 3.7: Interior del armario eléctrico. (Muñoz Sánchez, 2022a)

Estos, a su vez, se controlan mediante un Arduino Due al que se le envían los comandos de control mediante Serial. Los micropasos y la corriente de los drivers está configurada mediante los switches laterales para conseguir un microstepping de 8 y una corriente de 2 A. Gracias a esto, los motores tienen 1600 pulsos por revolución.

3.2. Código previo

Para poder escribir en el serial, se dispone del archivo Programa_Control.ino que hay que cargar en el arduino antes de empezar. Al cargarlo y establecer la conexión con el Arduino Due habiendo configurado los parámetros a los valores que se muestran en la tabla 3.1 este permite enviar comandos a los motores siguiendo siempre la siguiente sintaxis: **[motor;modo;dato1;dato2]**.

Tabla 3.1: Valores los parámetros de Arduino. Modificado de (Muñoz Sánchez, 2022a).

parámetro	valor
Baudrate	250000
DataBits	8
StopBits	1
Parity	None

Donde *motor* indica a que motor se le envía el comando, *modo* indica el tipo de comando y *dato1* y *dato2* son valores de velocidad o posición según sea necesario para cada modo. Los valores que pueden tomar se detallan en la tabla 3.2.

Tabla 3.2: Descripción de los comandos disponibles. Modificado de (Muñoz Sánchez, 2022a).

modo	dato1	dato2	comando
0	Disable		Deshabilita el motor correspondiente
1	Speed mode	velocidad (steps/s)	El motor se mueve a la velocidad indicada por dato1. El signo indica el sentido de giro.
2	Servo mode	velocidad (steps/s)	El motor se mueve hasta la posición dato1 a la velocidad indicada por dato1.
3	Set position	posición (pulsos)	Establece la posición actual al valor que se le indica. La posición cada vez que se restablece la conexión es 0.

Una vez que se conoce como se deben escribir los comandos en el serial de Arduino para mover los motores, es sencillo hacerlo desde Matlab, plataforma que permite también hacer la simulación de la cinemática y que facilita el desarrollo de código que permita realizar el control en base a ello. Razones por las cuales el resto del código disponible fue desarrollado en Matlab por mis compañeros Elena y Jaime. Este código está disponible en sus respectivos repositorios de Github (Muñoz Sánchez, 2022b) y (Bravo Algaba, 2022) y dada su extensión a continuación se explica muy brevemente su contenido, haciendo hincapié únicamente en aquellas partes relevantes para la comprensión del desarrollo de este proyecto.

El envío de comandos a los motores se realiza mediante dos funciones que escriben en el serial, `calibrarArduino` y `enviarArduino`. La función `calibrarArduino` permite establecer la posición del robot o de un motor como el 0, mover cualquier motor hasta una posición dada a velocidad de 150, o devolver el robot a la posición 0. Mientras que la función `enviarArduino` permite mover el robot a un punto o una trayectoria a partir de los datos obtenidos con la cinemática. Como cada vez que se reinicia la conexión con Arduino desde Matlab se pierden los valores de posición del robot, es necesario usar la función `calibrarArduino` para devolver el robot al origen antes de empezar (si no lo está ya) e indicar que está en la posición 0.

En cuanto a la cinemática inversa, para poder obtener la posición de los motores para mover el extremo del robot a un punto del espacio de trabajo, se emplea un método iterativo. Partiendo de unos ángulos iniciales de los discos del robot, en cada iteración se calcula la posición aproximada del extremo mediante la cinemática directa, e introduciendo una corrección en función del error cometido respecto a la posición deseada, se calcula el ángulo de los discos mediante la cinemática inversa.

Para el cálculo de la cinemática directa, mi compañero Jaime Bravo Algaba, 2022 usó un sistema de clases que le permite simular la estructura del robot a partir de los elementos para calcular el comportamiento del robot. Partiendo de una clase para recrear los discos del robot y su movimiento con respecto a los adyacentes, se usan luego las clases `HRRSection` y `HRRTree` (que heredan de la clase `rigidBodyTree` de la Robotics System Toolbox) para generar las secciones a partir de los discos y el robot apartir de estas. Finalmente, emplea la clase `HRRobot`, hija de `HRRTree`, que además permite representar el robot y evaluar su comportamiento.

Para resolver la cinemática inversa se emplearon métodos que emplean tanto la matriz jacobiana como el descenso de gradiente. Estos se implementaron como métodos de la clase `HRRTree` que devuelven el ángulo girado por cada disco para un punto dado del espacio.

Se dispone, también, de funciones que permiten pasar del ángulo de los discos a su posición (`config2joints`), del ángulo de los discos a la longitud que hay que alargar o acortar cada cable (`ang2long`) y de la longitud a los pulsos que debe girar cada motor (`long2pulsos`). Y finalmente, las funciones `Demo`, `Demo_Trayectorias` y `Demo_Colision_Discos` engloban en un solo script los pasos necesarios para calcular las posiciones del robot para ir a un punto, para trazar una trayectoria y para comprobar si el punto al que se desea llevar el robot es imposible de alcanzar debido a colisiones entre los discos.

Es relevante mencionar que la función (`ang2long`) tiene en cuenta que la posición de las secciones previas a la que controla el cable también afecta a la elongación total que debe tener el cable. Sin embargo, el modelo que usa para este cálculo es bastante simple, ya que solo tiene en cuenta el alargamiento que supone el ángulo girado por las secciones anteriores, suponiendo ángulo constante y sin tener en cuenta el efecto de los cables (aparte de los 4 diseñados para controlarla) sobre la posición real de las secciones.

4. REPARACIÓN Y MEJORAS

Debido a que el robot Pylori-I había sufrido una caída que había provocado daños significativos, el primer paso de la ejecución de este proyecto fue la reparación de la estructura del propio robot. Además, también fue necesario modificar y reescribir parte del código disponible.

En esta sección se explica como se llevaron a cabo las reparaciones, así como las mejoras que se incluyeron para dejar el robot a punto para poder proceder a la toma de datos.

4.1. Reparación y mejoras del diseño mecánico

Como se aprecia en la figura 4.1 bajo estas líneas, la caída causó daños significativos en dos de los tres soportes de los motores del Pylori-I. Incluyendo una grieta, una grieta que atravesaba el último soporte a la altura del motor 3 y daños en los elementos que permiten la unión entre los dos soportes, tales como los agujeros roscados del segundo soporte, o los agujeros pasantes del tercer soporte así como flexiones en los respectivos tornillos que impedían no solo su correcto funcionamiento, sino también extraerlos.



Figura 4.1: Detalles del daño en los soportes tras la caída.

Por esta razón se consideró que era inviable repararlos y que había que sustituir estas piezas, por lo que se procedió a desmontar los elementos dañados, imprimir dos soportes nuevos y volver a montar el robot. Con el objetivo de tener las piezas disponibles en el menor tiempo posible, se imprimieron a la vez. La asociación RESET se encargó de imprimir el segundo soporte en PLA turquesa, mientras que el tercer soporte se imprimió en resina gris, como se puede ver en la figura 4.2

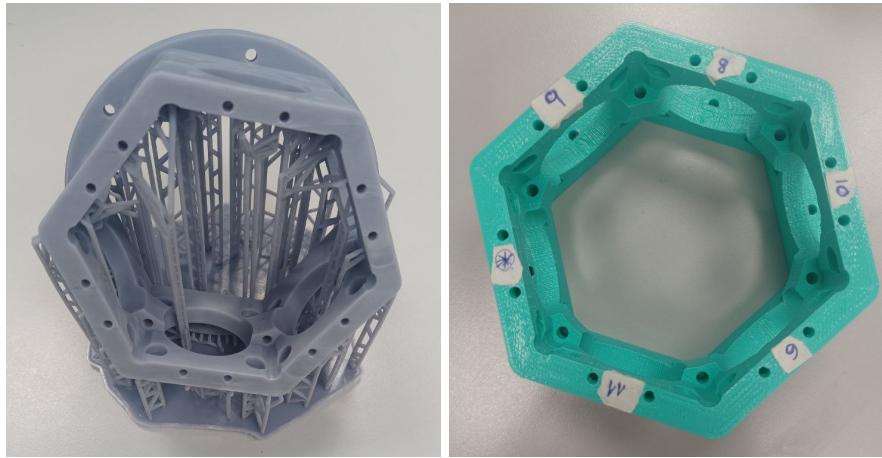


Figura 4.2: Soportes nuevos. Recién impreso en resina a la izquierda e impreso en PLA y listo para montar, a la derecha.

Durante el desmontaje del robot se observó que varios de los cables de conexión entre los motores y el armario eléctrico habían sufrido daños en alguno de los dos extremos (ver figura 4.3). Estos extremos de cable habían sido fabricados desde cero por mi compañera Elena Muñoz cuando diseñó y montó el robot. De modo que, tras comprobar con un multímetro que cables se habían soltado, se procedió a soldarlos y atornillarlos según lo necesitasen siguiendo el diseño original.

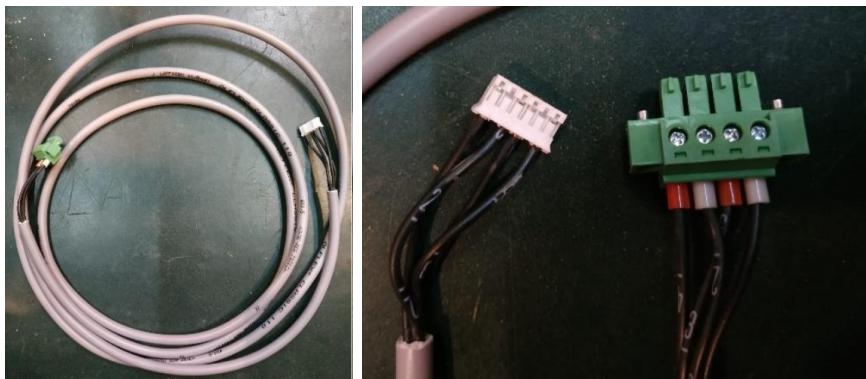


Figura 4.3: Cable y conectores para la conexión entre los motores y el armario. (Muñoz Sánchez, 2022a)

En la figura 4.4 se observa el robot ya montado y con todos los cables debidamente conectados, con la ventaja añadida sobre el original de que los distintos colores de los soportes facilitan ahora referirse a ellos.



Figura 4.4: Robot reparado.

Una vez que se puso en funcionamiento el robot se advirtió un último problema. Las poleas sobre las que se enrosca el hilo de pesca para lograr el alargamiento de los cables se rompían constantemente, dejando de transmitir la acción de los motores al robot. Después de cambiar varias por repuestos, se observó que se fracturaban siempre del mismo modo, como se aprecia en la figura 4.5.

Debido a la forma del orificio para la rosca, la fuerza que ejerce el tornillo al presionar sobre el eje del motor evitando que la polea deslice y la que ejerce la tuerca para mantener el tornillo en su lugar, se concentran en las esquinas del orificio para la tuerca y en perímetro del agujero pasante. Añadido al escaso grosor del plástico entre la tuerca y la cabeza del tornillo en una pieza tan pequeña, basta con que el cable ejerza pequeñas fuerzas de torsión para que la pieza empiece a romperse por esta zona.



Figura 4.5: Polea rota.

Para corregir este problema se procedió a rediseñar la pieza, siendo el resultado final el de la figura 4.6

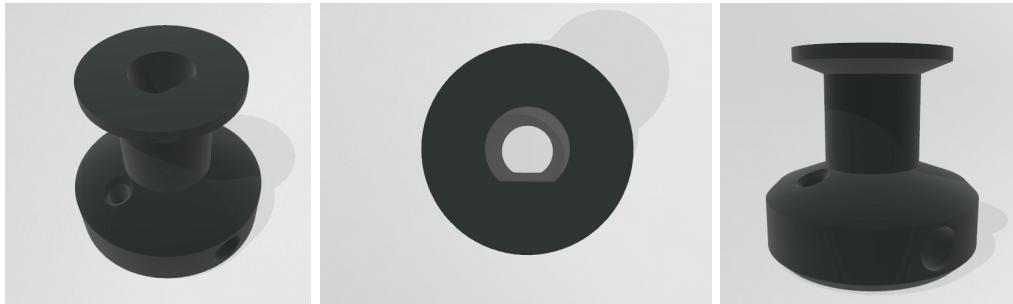


Figura 4.6: Nuevo diseño de la polea.

De los dos modelos de motor paso a paso empleados, uno de ellos tiene un eje en D, mientras que a los demás ya se les había mecanizado el eje previamente para que lo tuviesen. Esto permitió modificar el diseño original de la polea para que al sustituir el agujero cilíndrico para el eje que atraviesa toda la pieza por uno en D, hubiese una segunda superficie (significativamente mayor) que impidiese el deslizamiento de la polea sobre el eje soportando la torsión debida a la tensión del cable. Así, el tornillo y la tuerca transmiten menos tensiones, lo que permite tensar más sin que haya roturas y, además, complica que haya deslizamientos, incluso aunque la pieza presentase fracturas.

La segunda modificación que se introdujo en la pieza fue la reducción de la altura total de la pieza, reduciendo ligeramente la altura de la parte sobre la que se enrolla el cable. Debido al escaso espacio que queda en el interior de los soportes de motores, para que las poleas no rozasen unas con otras en el extremo superior, una vez montados todos los motores era necesario introducir la polea hasta el final del eje. Lo que, como se aprecia en la figura 4.7, suponía que la base de la polea quedase dentro del orificio del soporte para el motor, dificultando considerablemente el acceso al tornillo. Reducir la altura de la polea permite, por tanto, introducir menos la polea, dando mejor acceso al tornillo y facilitando así el montaje y desmontaje de estas piezas en caso de que haya que volver a sustituirlas en un futuro.



Figura 4.7: Disposición de la polea en el soporte.

4.2. Aportaciones al código

Una vez que se hubo terminado de reparar el hardware del robot, se intentó emplear el código disponible para mover el robot y comprobar la calidad del control del robot de la que partía el

4. REPARACIÓN Y MEJORAS

proyecto. Sin embargo, se encontró que el código disponible no funcionaba, por lo que en este apartado se explica los pasos que se tomaron para corregirlo y las modificaciones y mejoras que se introdujeron al código.

El código previo a este proyecto se había escrito simultáneamente por varias personas que habían tenido que coordinarse para distintas tareas en distintos momentos, y además se había desarrollado código para resolver varias tareas muy distintas entre sí. Por esta razón no solo había varias versiones del código con distintas funcionalidades, sino que algunos de los archivos necesarios para ciertas funcionalidades se habían guardado aparte y no se había escrito documentación sobre como ejecutarlo o en qué orden. El primer paso para obtener un código funcional fue averiguar que versión del código se quería usar para cada funcionalidad que se iba intentando poner en funcionamiento, así como que archivos requería y donde se encontraban. Por esta razón se decidió crear un repositorio de Github (cuyo enlace está disponible en el 9) que hereda código del de mi compañera Elena y emplea control de versiones y ramas para ir desarrollando código nuevo e integrando el de otras versiones. Con esto no solo resulta más sencillo reparar los errores existentes, sino que se facilita la comprensión y acceso al proyecto a quien lo pueda continuar en un futuro.

Ya que no es posible comprobar si la cinemática funciona correctamente solo con la simulación sin mover el robot, se decidió empezar por reparar el código que permite mover el brazo robótico de forma no automatizada, motor a motor. Al intentar ejecutar el código previo, una vez que se disponía de todos los archivos necesarios y se conocía en qué orden debían ejecutarse, se comprobó que varias de las funciones de Matlab que se usaban repetidamente se habían retirado de las versiones más actuales de Matlab debido a que producían errores con frecuencia. Así pues, se decidió reescribir los scripts correspondientes, y aprovechar para añadir mejoras. Por ejemplo, el código original dispone de un solo script que permite mover el robot sin usar la cinemática denominado *calibrarArduino*, que permite establecer la posición actual del robot o de un motor como la posición 0, moverlo el robot hasta el origen o mover un motor a una posición concreta para calibrarlo (a una velocidad preestablecida; mientras que al actualizar el código se han implementado dos funciones distintas para mover el robot sin la cinemática. Por un lado, se ha escrito la función *calibrarRobot* que permite establecer como posición de origen la posición actual de un motor o del robot, o devolver el robot a la posición 0; y por otro, la función *mover_motor* que permite mover el motor elegido tanto a una posición y velocidad concretadas por el usuario, como a la velocidad a elegir durante el tiempo que indique el usuario (más intuitivo para los casos en los que el robot se ha descalibrado y las posiciones están mal definidas o para cuando se desconoce a qué posición se desea llegar). Entre otras cosas, se modificaron algunas líneas del código que provocaban errores en la conexión con arduino al ejecutar varios scripts seguidos y se generó documentación sobre como ejecutarlos.

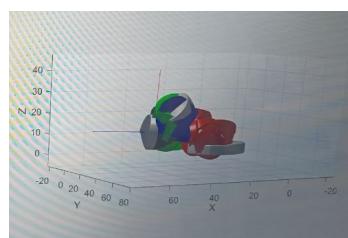


Figura 4.8: Simulación fallida de la posición del robot.

Una vez que se consiguió que el robot se moviera, se pasó a intentar integrar la cinemática en el control del movimiento. Después de actualizar las mismas funciones que daban problemas para mover motor a motor del código escrito por mi compañera Elena y de escribir un script para

indicarle a la cinemática que moviese el robot en una trayectoria aproximadamente circular (ya que se había determinado que es la óptima para la posterior toma de datos) se consiguió que el código simulase correctamente el movimiento y la posición del robot al trazar el círculo y que enviase comandos al robot para moverlo. La figura 4.9 muestra las posiciones del robot que generaba la simulación de la cinemática para los puntos de la trayectoria tras corregir el código, mientras que la figura 4.8 muestra la posición que devolvía la simulación al principio de este proyecto para un punto próximo al de inicio de la trayectoria circular.

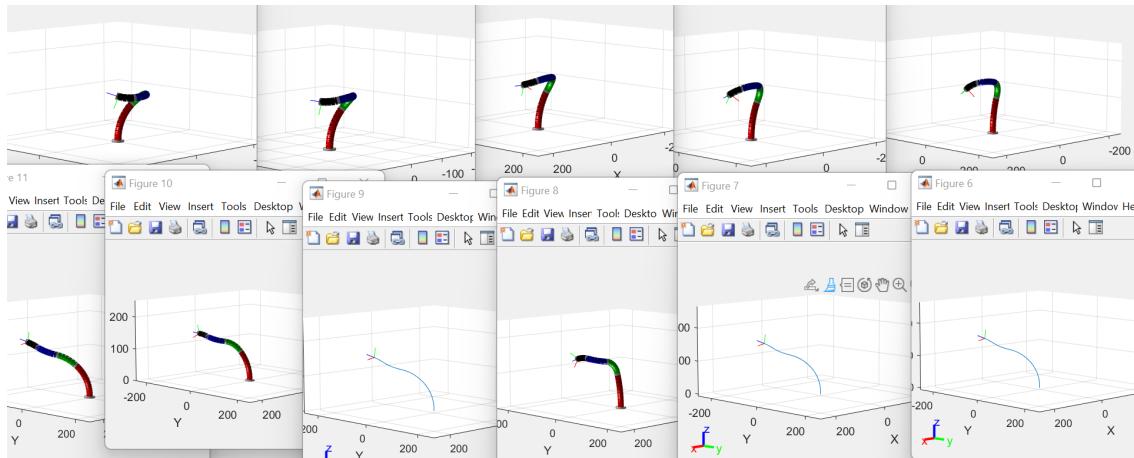


Figura 4.9: Simulación del movimiento del robot para una trayectoria circular.

Sin embargo, el comportamiento real del robot no se correspondía con el simulado, produciendo movimientos muy alejados de los deseados. Se han introducido varias mejoras al código para tratar de remediar este fallo. Como por ejemplo asegurar que los motores se mueven en un orden que garantiza que siempre se destensa un motor antes de tensar su opuesto (reduciendo así los efectos no deseados de las tensiones y los rozamientos de los cables tanto sobre la sección que controlan como sobre las demás) o cambiar a la versión de la cinemática recomendada por mi compañero Jaime (en vez de la que se empleaba originalmente en el código de mi compañera Elena), y corregir varios errores menores que se encontraron en el código que permite generar los comandos para mover el robot a partir de la simulación. Y si bien todo ello mejoró ligeramente el comportamiento del robot, no se ha llegado a conseguir que funcione correctamente.

Dado que el modelo matemático funciona correctamente, como se comprueba con la simulación, la causa de que el sistema real no se comporte del mismo modo debe ser debido a aspectos no considerados en el cálculo del alargamiento de los cables a partir de la posición simulada del robot, tales como el rozamiento, la tensión de los cables o la calibración. Es por ello por lo que se llega la conclusión de que el modelo formal matemático tal y como está implementado no es válido y debe ser aplicado otro tipo de metodología, como la que se propone basada en redes neuronales.

5. TOMA DE DATOS

5.1. Herramientas

Para la toma de datos se propusieron varias herramientas de captura y análisis de movimiento, pero finalmente se optó por Optitrack tanto por precisión y facilidad de uso comparado con los otros; como por su disponibilidad dado que había dos en el departamento. Este sistema consta tanto de hardware como de software propio. De los dos Optitracks disponibles, si bien el primero por el que se optó tiene una versión más actualizada del software, solo disponía de 4 cámaras que resultaron no ser suficientes para captar el movimiento correctamente. Por lo que, finalmente, se optó por la segunda opción disponible con 6 cámaras y la versión **Tracking Tools** para la toma de datos.

En este apartado se expone tanto como se ha empleado el hardware y el software de Optitrack para la toma de datos, como el modo en que se ha conectado mediante ROS a Matlab para poder capturar sus medidas y guardarlas.

El hardware de Optitrack consiste fundamentalmente en un conjunto de cámaras como las de la figura 5.1 que se conectan mediante switches a un ordenador, sus trípodes y las herramientas de calibración. Y finalmente, marcadores (que consisten en bolas reflectantes como las que se muestran en la figura 5.1), cuya posición puede captarse por las cámaras con precisión mediante triangulación.



Figura 5.1: Marcadores y cámaras del Optitrack.

Para que las cámaras puedan seguir el movimiento de un punto sólido, denominados trackables, se deben colocar al menos 3 marcadores sobre el objeto en una posición que deje el punto en cuestión en su centro geométrico. El primer obstáculo que se encontró ya al probar con el primer Optitrack es que, debido al escaso diámetro del brazo robótico de apenas 17 mm, las cámaras no eran capaces de distinguir los marcadores que se colocaban en el extremo entre sí. Y debido al tamaño y estructura del robot, colocar demasiado peso en el extremo afecta considerablemente a su comportamiento e introduce esfuerzos apreciables sobre algunos cables ya en su posición de reposo. Lo que a su vez afecta a su rango de movimientos y al deterioro de los cables y las poleas. Teniendo esto en cuenta, la solución a la que se llegó fue adherir al extremo un triángulo equilátero recortado en cartón con cuadrados en sus puntas sobre los que fijar los marcadores (ver figura 5.2).

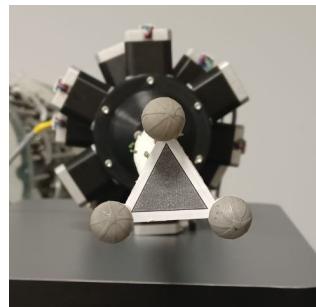


Figura 5.2: Montaje de los marcadores en el extremo del brazo robótico.

A pesar de haber escogido un material ligero, el peso del cartón junto al de los marcadores provocó que hubiese que tensar más algunos cables (sobre todo los correspondientes a los motores 1 y 5) y contribuyó a que el robot se descalibrase con mayor facilidad durante el movimiento, como se verá más adelante. Sin embargo, el peso no fue tanto como para que no se pudiese volver a la posición de reposo, como se aprecia en la figura 5.3.



Figura 5.3: Brazo robótico recalibrado con el peso de los marcadores.

El siguiente paso fue instalar las cámaras, distribuyéndolas de modo que pudieran ver en todo momento los dos puntos que se querían medir: el extremo de robot y la base del robot. Como se explica en el manual de Tracking Tools (OptiTrack, 2012), es necesario que se den varias condiciones para ello.

Para empezar, se eliminaron en la medida de lo posible todas las fuentes de luz infrarroja (por ejemplo, tapando las ventanas) u objetos altamente reflectantes del campo de visión de las cámaras, ya que son muy sensibles a las interferencias que causan. En los casos de las fuentes de luz que no era posible eliminar, tales como otras cámaras o los motores del propio robot, se garantizó que no variaran su posición a lo largo de los experimentos para que el programa pudiera enmascararlas. Además, dado que los datos se toman por triangulación, las cámaras deben situarse de modo que los dos puntos a medir queden siempre dentro del rango de visión de al menos tres de ellas. A ser posible desde ángulos bastante diferentes para mejorar la calidad de la medida.

Considerando todo esto y con el espacio del que se disponía en la sala de prácticas del CAR, la disposición final quedó como se ve en la figura 5.4 donde el robot se coloca en la plataforma negra que queda entre las cámaras.

5. TOMA DE DATOS



Figura 5.4: Distribución de las cámaras para la toma de datos.

Tras conectar todas las cámaras a los switches y estos al ordenador, el programa permite hacer una calibración para poder conocer la posición de las cámaras y así poder situar los puntos a medir correctamente en su sistema de coordenadas. Moviendo la vara de la figura 5.7b por el campo visual de las cámaras, se recogen medidas en forma de una nube de puntos distinta para cada cámara (ver figura 5.5). Para un grupo de 6 cámaras, a partir de 1000 puntos por cámara empieza a ser suficiente para que la calibración se pueda llevar a cabo (OptiTrack, 2012), pero para conseguir una calibración de mayor calidad se ha comprobado que es necesario tomar al menos 10000 datos para las cámaras con peores ángulos que son las que menos datos suelen captar.

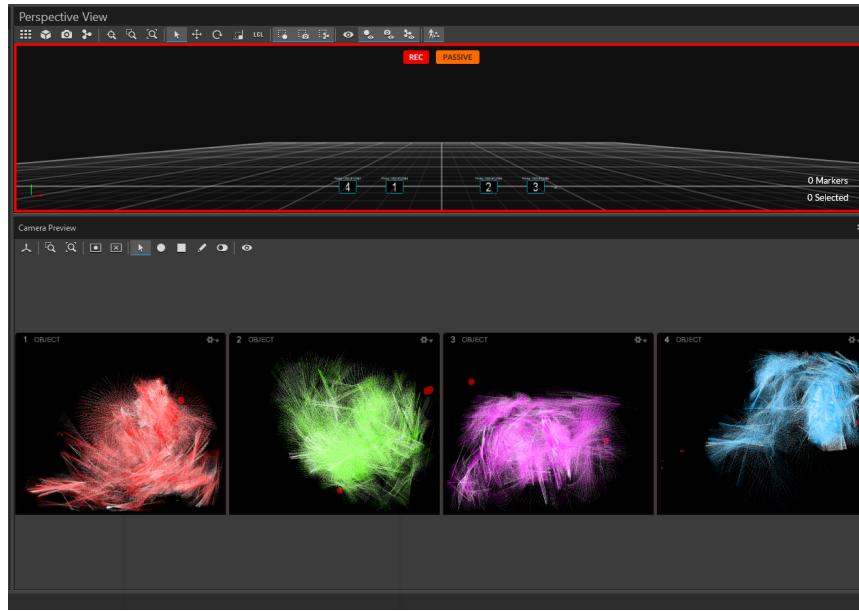


Figura 5.5: Toma de puntos para la calibración

Cuando se dispone de suficientes datos, el programa realiza un cálculo iterativo mediante la triangulación de aquellos puntos que han captado varias cámaras para estimar la posición real de estas, como se observa en la figura 5.6. En esta misma figura se puede ver el panel que va mostrando la calidad y el error promedio de la calibración lograda en cada momento hasta que se decide detener el cálculo y aplicar el resultado. A partir de entonces, el panel denominado

perspective view muestra la distribución en el espacio de las cámaras y de los puntos que se midan.

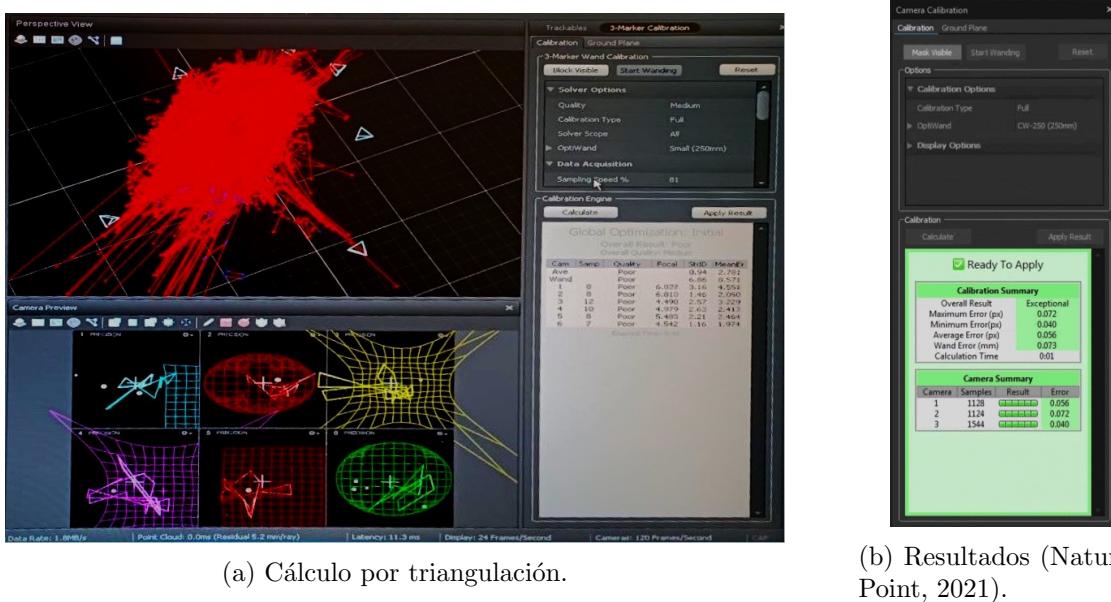


Figura 5.6: Calibración del Optitrack.

El último paso para terminar la calibración y que Optitrack pueda medir la posición de los puntos es proporcionar la posición del origen y los ejes del sistema de referencia, para lo que se coloca el L-frame mostrado en la figura 5.7a en el punto deseado. Entonces se genera un archivo con la calibración para poder restablecerla siempre y cuando no se hayan movido las cámaras ni hayan variado las condiciones del entorno.

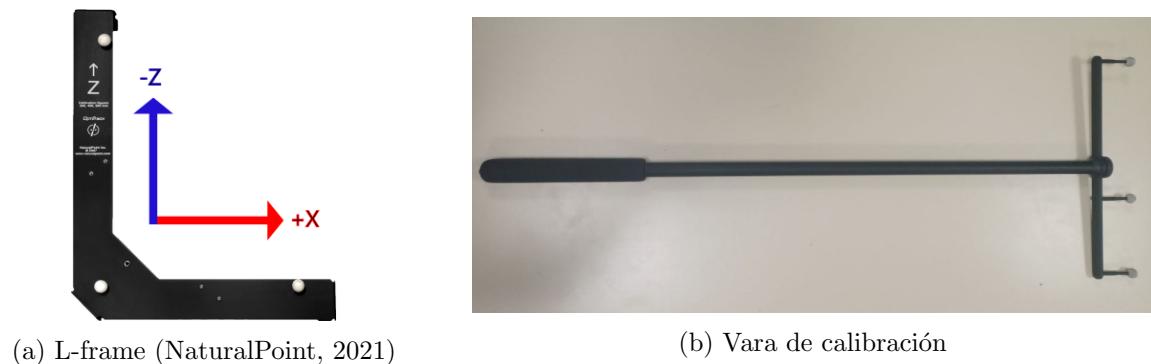


Figura 5.7: Herramientas de calibración.

Una vez calibrado se vuelven a colocar los marcadores de los puntos a medir (3 para cada uno de los dos puntos, en este caso) y desde el panel perspective view se seleccionan los que corresponden a cada uno de los puntos por separado y se definen como trackables. Optitrack mide la posición, orientación y error de posición estimado de todos los trackables y muestra en pantalla los de aquel que esté seleccionado (ver figura 5.8)

5. TOMA DE DATOS

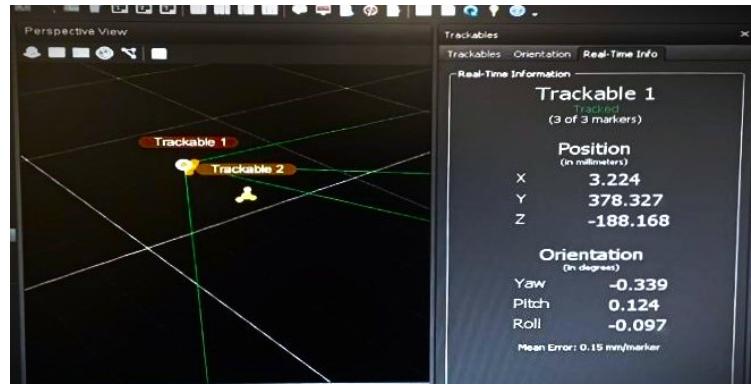


Figura 5.8: Visualización de los trackables y su medida en Optitrack.

Optitrack es capaz de tomar datos de los trackables cada 0,2 segundos y exportarlos a un archivo .csv. Sin embargo, para este proyecto solo son útiles los datos de las posiciones del robot una vez que ha llegado a la posición requerida y no del movimiento, por lo que es necesario sincronizar la toma de datos desde el programa que envía comandos al robot, en este caso Matlab.

Dado que el ordenador que contenía la licencia y el programa de Optitrack no reunía los requisitos para instalar la versión necesaria de Matlab para mover el robot, se realizó una conexión mediante ROS (Robot Operating System) entre ambos ordenadores empleando un cable Ethernet.

Optitrack permite transmitir datos por varios medios que se pueden configurar desde el panel Streaming Properties, similar al de la figura 5.9.

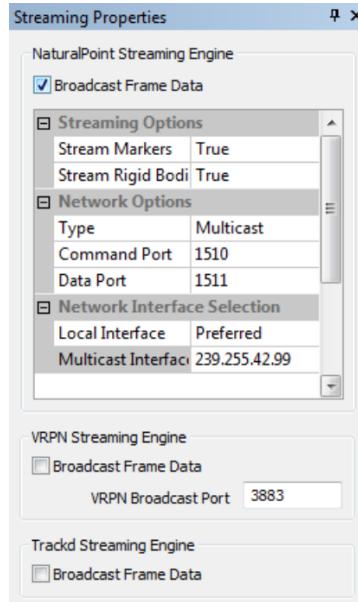


Figura 5.9: Panel de configuración de (OptiTrack, 2012).

El paquete de ROS recomendado por Open Robotics para la transmisión de datos desde Optitrack es `vrpn_client_ros` (Quigley et al., 2009),(Robotics, 2017) disponible para la versión `melodic` de ROS. Fue necesario instalar esta versión de ROS sobre el sistema operativo Ubuntu 18.04 (bionic) para el que se desarrolló originalmente (Robotics, 2018), ya que las versiones que se han sacado para otros sistemas operativos no tienen todas las funcionalidades, incluyendo

algunas de las requeridas en este caso.

Una vez seleccionada la opción de transmisión de datos por VRPN en Optitrack y configurada correctamente la IP del ordenador que recibe los datos, `vrpn_client_ros` establece un tópico de tipo `<tracker name>/pose` por cada trackable de Optitrack. Los datos de la posición y orientación de cada uno se transmiten a través de su tópico correspondiente con un formato `geometry_msgs/PoseStamped` (Robotics, 2017),(Quigley et al., 2009). Consiste en una estructura con un encabezado (de tipo `std_msgs/Header.msg`) en el que se incluye el momento en que se ha tomado el dato y el nombre del trackable, y una estructura de tipo pose (`geometry_msgs/Pose.msg`) que incluye una variable con los valores de la posición en cartesianas y otra con la orientación en quaternios.

Gracias a la ROS Toolbox y conociendo la ip del ordenador master (desde el que se ejecuta Optitrack en este caso), se puede establecer una conexión mediante ROS y guardar en una variable los datos que se reciben a través del tópico correspondiente en un momento concreto. Para ello, primero se crea un objeto con la función `rossubscriber` en el que se establece de qué tópico se van a recibir datos, el tipo de dato que se recibe y que el dato debe almacenarse en una variable de tipo estructura. Y después, cada vez que se quiera recoger el dato del tópico, basta con llamar a la función `receive` para este objeto (indicando el tiempo de espera que puede tardar en recibir el dato), y volcarlo en una variable.

5.2. Diseño de los experimentos

Una vez establecida la conexión entre Matlab y Optitrack se procedió a seleccionar que datos se iban a tomar para entrenar la red neuronal posteriormente. Esta decisión requirió considerar varios factores.

Para empezar, para que el desempeño de la red neuronal sea el mejor posible, se debe disponer de medidas para un número suficiente de puntos del espacio de trabajo del robot distribuidos de forma lo más uniforme posible. Sin embargo, debido a la facilidad con la que las tensiones altas dañan las poleas y los cables (con el factor añadido de la tensión extra que ya soportaban algunos cables para compensar el peso de los marcadores del Optitrack en el extremo del robot), el área en el que se podía mover el robot sin riesgo de roturas o comportamientos erróneos se redujo considerablemente. A lo que se añade los efectos en las tensiones de los cables debidas a los movimientos de otras secciones. De modo que el acortamiento máximo de los cables no podía ser muy alto, siendo aún más restrictivo cuantos más motores se mueven al mismo tiempo.

Otra dificultad a la hora de diseñar los experimentos es el efecto del rozamiento de los cables con los discos que, combinado con la falta de unión fija entre los discos más allá de los propios cables, permite que los discos deslicen ligeramente unos respecto a otros si se mueve el robot durante un tiempo prolongado. Este efecto provoca que el ángulo de giro de las secciones por las que pasan menos cables se deforme a lo largo de la sección y esto a su vez provoca tensiones nuevas y que el brazo no se desplace del mismo modo que recién calibrado aún enviando los mismos comandos. Para corregirlo hay que detener el movimiento, corregir la posición de los discos manualmente y volver a calibrar. Por lo tanto, el número de datos que se toman en un solo experimento debe ser limitado y las direcciones en la que se mueve el robot en cada experimento lo más simétricas posibles para evitar que el efecto del deslizamiento de los discos llegue a una magnitud tal que los datos que se recojan sean erróneos.

Finalmente, es necesario considerar que cada vez que se tensa un cable, hay que destensar en la misma medida el cable opuesto (es decir, el que controla el movimiento en la misma dirección pero

sentido opuesto para esa misma sección) para conseguir el movimiento deseado y no provocar tensiones que el robot sea incapaz de soportar. De modo que para los experimentos solo que pueden considerar como variables independientes 8 motores.

Teniendo en cuenta todo lo expuesto, se decidió hacer varias tomas de datos más cortas para poder parar a recalibrar con frecuencia y escoger conjuntos de datos que estuviesen lo más uniformemente distribuidos en torno al origen posible sin tensar excesivamente los cables. Los experimentos realizados son:

- Tensar solo un motor cada vez llevándolo a posiciones entre 200 y 2200 pulsos en intervalos de 200 pulsos.
- Mover todos los motores 500 y 800 pulsos a la vez para todas las posibles combinaciones en las que se garantiza que para cada motor que se tensa su opuesto se destensa lo mismo.
- Mover un motor de cada sección a la vez, tensándolos todos 500 pulsos, 800 pulsos o bien combinaciones de ambos valores.
- Mover todos los motores de una única sección a la vez, tensándolos o destensándolos 500 pulsos, 800 pulsos o bien combinaciones de ambos valores.
- Mover todos los motores de dos secciones a la vez, tensándolos o destensándolos 500 pulsos, 800 pulsos o bien combinaciones de ambos valores.
- Mover todas las combinaciones posibles de dos motores no opuestos, tensándolos todas las posibles combinaciones de valores de 200 a 1600 pulsos en intervalos de 200 pulsos.

Para cada uno de estos experimentos se escribió un script que establece la conexión con ROS y automatiza la captura de los datos de la posición y orientación del extremo y de la base del robot cada vez que el robot termina de moverse y lo vuelca en un .csv junto al comando de los motores que le corresponde. Además, al inicio del script se captura y vuelca la posición inicial del robot. Esto sirve para dos propósitos: identificar que experimentos han sido fallidos por no haber corregido correctamente la deformación causada en la posición de los discos en experimentos previos; y disponer datos para realizar el cambio de sistema de referencia en el procesamiento de datos.

Cada vez que se terminaba un experimento o que había que interrumpirlo debido a algún fallo, se corregía la posición de los discos y se recalibraba el robot. Sin embargo, no siempre se pudo hacer con precisión, ya que en ambos casos se hace manualmente.

5.3. Procesamiento y limpieza de datos

Ha sido necesario tanto procesar los datos obtenidos como limpiar ruido para poder emplearlos para entrenar la red neuronal.

Como ya se ha comentado, Optitrack transmite la posición y orientación de los puntos medidos expresados respecto de su propio sistema de referencia, cuyo origen se define fuera del robot, mientras que la cinemática que se ha empleado para simular y controlar el robot define la posición del extremo respecto de un sistema de coordenadas con origen en la base, eje Z en la dirección del brazo en su posición de inicio y eje Y según el motor 1. Por tanto, el primer paso es realizar una transformación de coordenadas.

Para ello, con la colaboración de mi compañero Diego Cerrillo, hubo que desarrollar un script que, a partir de los datos disponibles, siguiese los pasos que se explican a continuación.

Para que la explicación resulte más sencilla de entender, en la figura 5.10 se han representado los puntos b (base del robot) y e (extremo del robot) expresados en ambos sistemas de referencia (el del Optitrack en rojo y el de la cinemática en negro), junto con sus respectivos puntos de origen (O para el Optitrack y la propia base del robot para la cinemática) y el vector \bar{v} que va de la base al extremo.

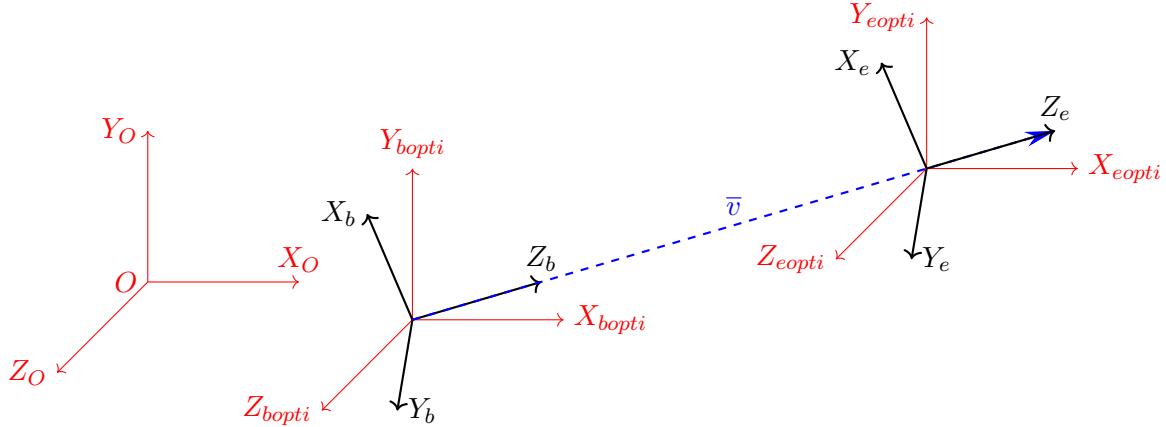


Figura 5.10: Esquema de los sistemas de referencia

Como durante los experimentos se han tomado medidas de ambos puntos en reposo antes de empezar a mover el robot, se conocen la posición y orientación de los dos puntos expresados según el sistema de referencia del Optitrack, b_{opti} y e_{opti} . Además, conocemos el módulo del vector \bar{v} y sabemos que su sentido en el sistema de referencia de la cinemática es según el eje Z, por lo que conocemos el vector en este sistema (\bar{v}_b). Partiendo de esta información se quiere llegar a la matriz de transformación homogénea (de traslación y rotación) entre ambos sistemas: $MTH_{b,O}$. Tal y como se explica en (Barrientos et al., 2007), conociendo la posición en cartesianas (expresadas en un vector que denominamos P) y su orientación en ángulos de Euler (que permite obtener de forma inmediata su matriz de rotación respecto al origen, R), se conoce la matriz de transformación homogénea (en adelante MTH) de los puntos respecto al sistema de referencia del Optitrack tal y como se muestra en la ecuación 5.1 para la base, y de forma análoga para el extremo.

$$MTH_{O,b_{opti}} = \left[\begin{array}{c|c} R_{b_{opti}} & P_{b_{opti}} \\ \hline 0 & 1 \end{array} \right] \quad (5.1)$$

Lo que nos permite plantear la siguiente ecuación:

$$MTH_{b,O} = MTH_{b,b_{opti}} \cdot MTH_{b_{opti},O} \quad (5.2)$$

por lo que basta con calcular $MTH_{b,b_{opti}}$ para obtener $MTH_{b,O}$. Se empieza por calcular la MTH de un punto a otro, expresados en el sistema de referencia de Optitrack a partir de sus MTH respecto al origen de este sistema que habíamos calculado según la ecuación 5.1.

$$MTH_{b_{opti},e_{opti}} = MTH_{O,b_{opti}} \cdot MTH_{b_{opti},O} \quad (5.3)$$

La cuarta columna de la matriz $MTH_{b\text{-}opti\text{-}eopti}$ que acabamos de calcular se corresponde con el vector que define la traslación entre ambos puntos expresada según el sistema de referencia del Optitrack, es decir $\bar{v}_{b\text{-}opti}$. Conociendo el vector \bar{v} en ambos sistemas de referencia podemos calcular el eje y ángulo de rotación de la matriz $MTH_{b\text{-}opti}$ que estamos buscando. Donde la dirección del eje se calcula como la norma del producto vectorial de los vectores $\bar{v}_{b\text{-}opti}$ y \bar{v}_b , y el ángulo se calcula como el arcocoseno de su producto vectorial. Y ahora contamos con los datos necesarios para aplicar la fórmula de rotación de Rodrigues con la función *Rodrigues* de Matlab y obtener la matriz de rotación $R_{b\text{-}opti}$.

Una vez obtenida, se calcula la matriz $MTH_{b\text{-}opti\text{-}b}$ como en la ecuación 5.4 y finalmente se sustituye en la ecuación 5.2.

$$MTH_{b\text{-}opti} = \left[\begin{array}{c|c} R_{b\text{-}opti} & 0 \\ \hline 0 & 1 \end{array} \right] \quad (5.4)$$

Debido al formato de dato que emplea ROS para Optitrack la orientación se recibe en cuaternios, pero los cálculos anteriores y el diseño de la cinemática se ha realizado empleando los ángulos de Euler. Por lo que se convierte la orientación de cuaternios a ángulos de Euler de todos los datos al principio del script, antes de realizar los cálculos para encontrar la MTH.

Además, durante la toma de datos se observó que debido a la proximidad de los puntos medidos y del ángulo en el que los ven las cámaras cuando se distribuyen para minimizar el número de cámaras que se ven entre sí, Optitrack confunde ocasionalmente a qué trackable corresponden los marcadores. Aunque estos errores se dan con poca frecuencia y durante poco tiempo, es necesario detectar que datos del extremo y la base están mal tomados e intercambiarlos.

Para cada experimento se ha ejecutado un script que ejecuta los cálculos para encontrar la matriz de transformación del sistema de referencia a partir de los datos del robot antes de comenzar el experimento, luego aplica la transformación a todos los datos, intercambia los datos de la base y el extremo cuando es necesario y finalmente los vuelca en un archivo .xls junto con sus comandos de motores correspondientes. Con esto termina el procesamiento de datos.

Una vez que se disponía de los datos correctamente procesados, se representaron para poder tener una primera idea de la calidad de los datos obtenidos.

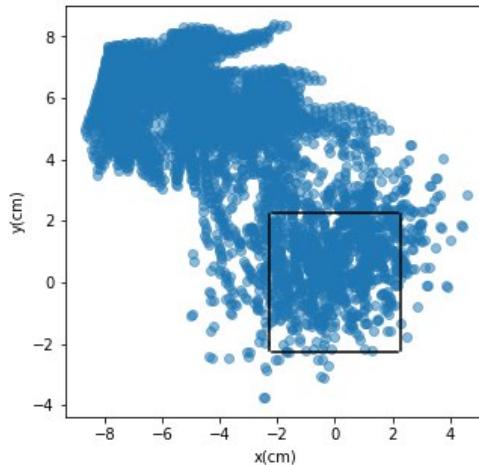


Figura 5.11: Distribución de los datos en el plano XY

Como ya se ha comentado en el apartado anterior, los experimentos realizados deberían proporcionar puntos distribuidos de forma aproximadamente simétrica alrededor del origen en el plano XY. Sin embargo, en la figura 5.11 se aprecia que esto solo se cumple para algunos conjuntos de datos. Al estudiarlo en más detalle, se observa que los datos que se encuentran desplazados hacia el cuadrante superior izquierdo de la gráfica corresponden a los mismos experimentos. Generalmente, los que se realizaron más tarde, de modo que el error cometido al recalibrar y recolocar el brazo robótico se había ido acumulando. Ante la imposibilidad de tomar nuevos datos, se eliminaron los correspondientes a estos experimentos y se procedió a entrenar la red neuronal.

6. REDES NEURONALES

Una vez que se disponía de todos los datos limpios, se procedió a entrenar las dos redes neuronales (NN). Una para la cinemática directa, es decir, que dadas las posiciones de los motores en pulsos devuelve la posición del extremo en coordenadas cartesianas; y otra para la cinemática inversa, que devuelva los comandos que se le deben enviar a cada motor en pulsos para llevar el extremo a una posición de su espacio de trabajo.

La solución escogida para ambas redes neuronales es muy similar, pero debido a la naturaleza hiperredundante del robot, la NN (red neuronal) para la cinemática inversa no se podía entrenar directamente sobre los datos obtenidos con los experimentos. Así pues, en los siguientes apartados se exponen las características de la NN escogida y sus hiperparámetros para la cinemática directa; y la solución alcanzada para los datos en el caso de la cinemática inversa, junto con los hiperparámetros de la NN para esta.

6.1. Cinemática directa

Este tipo de problemática de regresión requiere una red neuronal hacia delante o MLP (Multi-Layer Perceptron) que se ha entrenado en Python empleando la implementación de sickit-learn (Pedregosa et al., 2011)). Las decisiones sobre las características e hiperparámetros que se han escogido para implementarla se han tomado usando como libro de referencia *Deep Learning* (Goodfellow et al., 2016), por lo que se invita al lector a acudir a él si requieren más detalles sobre cómo se han escogido.

Para empezar, hay que escoger cuidadosamente las variables de entrada y salida de la NN. Al haber tenido que descartar los datos que se alejaban apreciablemente de la posición de origen del robot y dada la precisión del Optitrack, se ha observado que se puede despreciar la coordenada Z de la posición del extremo para nuestro conjunto de datos. Además, como ya se ha explicado, los valores que toman los motores pares dependen directamente de los motores impares, por lo que en realidad solo hay 8 variables independientes de entre los 16 motores. Así se define la variable x como la posición del extremo en el plano (X,Y) y la variable y como un vector en \mathbb{R}^8 con la posición de los motores impares.

Es necesario tener en cuenta que las MLP son muy sensibles al rango de las variables, por lo que se recomienda reescalar los datos que se le proporcionan a $[-1,1]$ y deshacer el rescalado una vez obtenida la variable de salida. Así pues, las variables de entrada y salida de la NN se definen en relación con las variables x e y como sigue:

$$x_t = 2(x - x_{min}) / (x_{max} - x_{min}) - 1 \quad (6.1)$$

$$y_t = 2(y - y_{min}) / (y_{max} - y_{min}) - 1 \quad (6.2)$$

Donde ./ y .* indican que se trata de una división y una multiplicación elemento a elemento, y x_{min} , x_{max} , y_{min} y y_{max} representan el valor más alto y más bajo que toman las variables x e y .

Las ecuaciones que describen la estructura de la red neuronal son:

$$h_i = g(W_i h_{i-1} + b_i) \quad i = 1, 2, \dots, N \quad (6.3)$$

$$y_t = W_{N+1} h_N + b_{N+1}, \quad (6.4)$$

donde N es el número de capas escondidas, h_i es el vector que representa el estado de la i -ésima capa oculta, $h_0 = x_t$, W_i son matrices con los pesos de las capas, b_i son los sesgos y g es la función de activación. Un ejemplo de la estructura de una red neuronal con una capa escondida se puede ver en la figura 6.1.

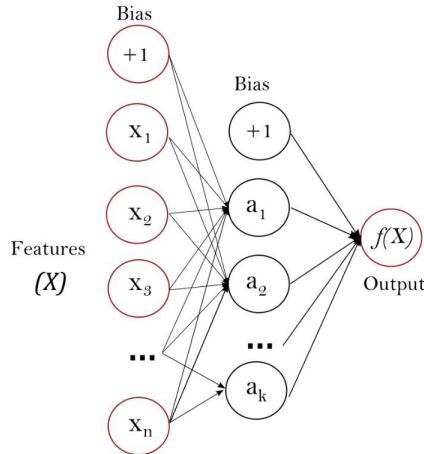


Figura 6.1: Red neuronal de tipo MLP con una capa oculta. (Pedregosa et al., 2011)

Después de probar con varias posibles funciones de activación para comprobar cuál produce mejores resultados, se ha escogido la función ReLU (Rectified Linear Unit):

$$g(t) = \max(t, 0) \quad (6.5)$$

Como función de coste de la red neuronal se ha utilizado el error cuadrático medio con regularización L^2 (también conocida como weight decay). Por tanto, la función de coste resultante, \tilde{J} , se puede escribir de la siguiente forma (Goodfellow et al., 2016):

$$\tilde{J}(w; x_t; y_t) = \frac{\lambda}{2} w^T w + J(w; x_t; y_t) \quad (6.6)$$

Donde w es un vector que contiene los pesos y los sesgos (W_i , b_i y de las ecuaciones 6.3 y 6.4), J el error cuadrático medio y λ es el coeficiente de la regularización L^2 y uno de los hiperparámetros de la NN que es necesario determinar. Cuanto mayor es este parámetro, más se reduce la velocidad con la que aumentan los valores de los pesos, reduciendo así la tendencia a que se dé overfitting como se aprecia en la figura 6.2, por lo que el valor que se le da debe escogerse con cuidado.

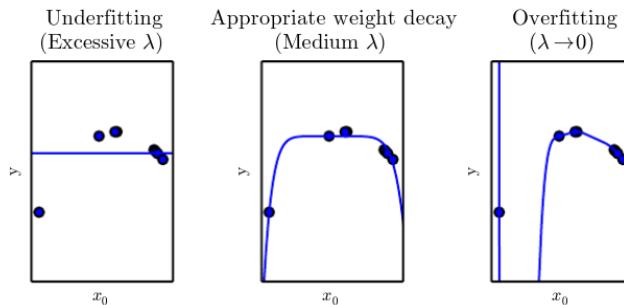


Figura 6.2: Efecto del coeficiente de la regularización L^2 sobre la tendencia de la red a cometer overfitting. (Goodfellow et al., 2016)

El optimizador empleado para reducir esta función de coste es el optimizador Adam (Kingma y Ba, 2014). Para usar este optimizador en la implementación de la MLP de scikit-learn (Pedregosa et al., 2011) hay que indicar el valor del paso del optimizador, también conocido como tasa de aprendizaje (o learning rate) que controla cuanto se modifica el modelo con cada iteración. Valores demasiado pequeños o demasiado grandes de este hiperparámetro provocan que el optimizador no encuentre el mínimo, por lo que el paso del optimizador debe elegirse cuidadosamente para obtener resultados óptimos.

Para controlar la tendencia al overfitting mientras se entrena la NN, además de la regularización L^2 , también se emplea el método conocido como early stopping (Goodfellow et al., 2016). La figura 6.3 muestra la evolución del error cometido, para el training set y para el validation set para un ejemplo de red neuronal parecida a la empleada en este trabajo. En esta figura, el tiempo se mide en iteraciones del algoritmo de optimización (indicado como epochs en la figura). Al principio ambos errores se reducen con cada iteración, reduciendo así el underfitting de la NN. Sin embargo, llega un momento en que aunque el error del training set sigue reduciéndose, el cometido para el validation set empieza a aumentar cada vez más; lo que significa que lo más probable es que se esté empezando a producir overfitting.

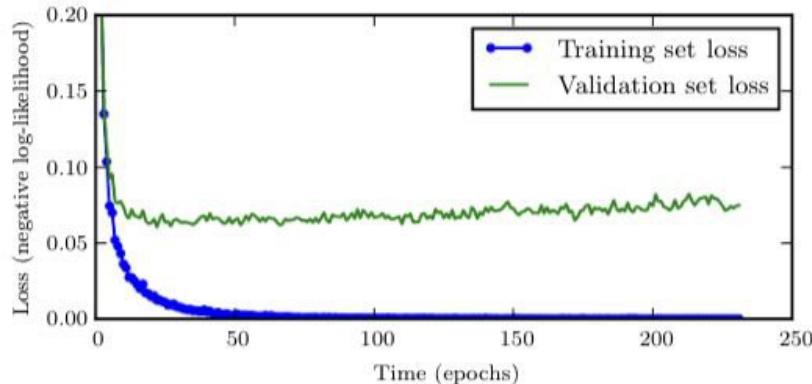


Figura 6.3: Evolución del error en el training set y el validation set sin early stopping.(Goodfellow et al., 2016)

El método early stopping consiste en almacenar los parámetros del modelo para el mejor valor del error en el validation set hasta el momento y dejar de entrenar la red en el momento en que se deja de apreciar ninguna mejora respecto a estos o cuando se alcanza un número predeterminado de iteraciones (Goodfellow et al., 2016).

Finalmente, solo queda explicar como se han escogido los valores de los tres hiperparámetros de esta red neuronal, que son: el parámetro λ del weight decay del que ya se ha hablado; la tasa de aprendizaje (o learning rate), y el tamaño de cada capa oculta (número de unidades que tiene) y número de capas ocultas.

Su valor se ha determinado mediante validación cruzada (o cross validation), para lo que es necesario dividir los datos empleados para encontrar los parámetros (o training data) en particiones, como se muestra en la figura 6.4. En el caso de la red neuronal de la cinemática directa se han dividido en 5 particiones. Hay que mencionar que debido a la escasez de datos disponibles, se ha decidido usar todos los datos como training set en vez de hacer un test data y, en su lugar, comprobar la calidad de los resultados observando directamente el comportamiento del robot al emplear la red neuronal, ya que la naturaleza del proyecto lo permite.

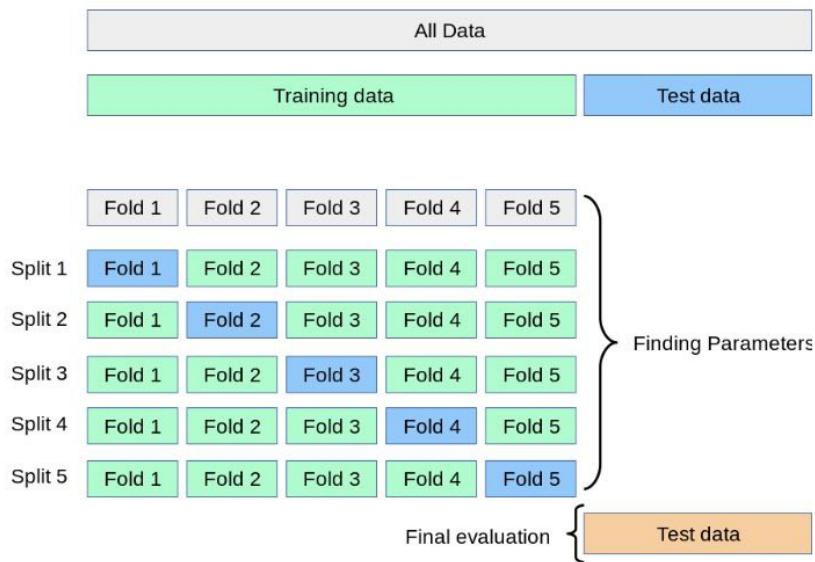


Figura 6.4: Validación cruzada: particiones de los datos para entrenar la red neuronal. (Pedregosa et al., 2011)

Los valores encontrados para los hiperparámetros de la red neuronal de la cinemática directa son 0.005 para el coeficiente de la regularización L^2 (λ), 0.01 para la tasa de aprendizaje y el número de capas ocultas son dos con 100 unidades cada una.

El resultado final se muestra en la figura 6.5: cada subfigura representa una de las dos variables a predecir, el eje de las x de cada subfigura muestra un valor medido con el Optitrack y el eje de las y , el valor que predice la red neuronal para esa combinación de motores. Estas predicciones se han obtenido por validación cruzada, es decir, se muestra la predicción para cada punto que corresponde a la partición en la que ese dato no se ha empleado para el entrenamiento. El coeficiente de determinación de la regresión estimado por validación cruzada es de $r^2 = 0,86$

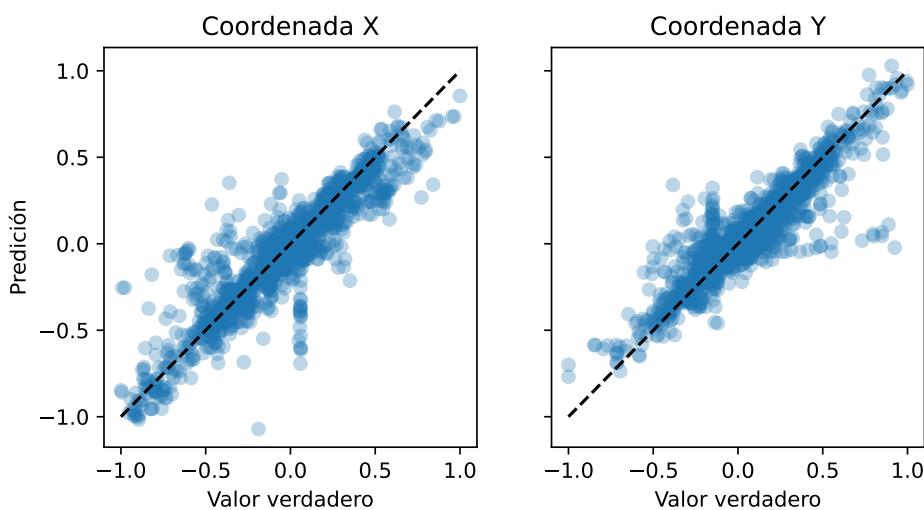


Figura 6.5: Resultado de entrenar la red neuronal para la cinemática directa. Cada punto azul representa un punto y la línea negra discontinua representa la recta predicción=real.

6.2. Cinemática inversa

Al ser el Pylori-I hiperredundante, si bien para cada posición de los motores existe una sola posición del extremo, para cada posición del extremo del robot (aunque no de robot en sí) en el espacio existe más de una combinación de posiciones de motores válida. No es posible entrenar una red neuronal que para una única entrada encuentre más de un valor de la variable de salida. De modo que hay que aplicar algún algoritmo que permita elegir no solo una sola posición de los motores para cada punto del espacio de la zona de la que se disponen datos, sino que además elija la que garantice un movimiento suave del robot al ir de un punto a otro para que no sufra daños al moverse en trayectorias.

Para empezar, de toda el área del espacio de trabajo de la que se dispone suficiente densidad de datos como para poder entrenar una red neuronal capaz de interpolar la posición del robot en ella, se escoge un cuadrado que quede contenido en ella y tenga la mayor superficie posible. Que en este caso, debido a la escasez de datos de la que se dispone, es un cuadrado de 2 cm de lado con centro en el origen del plano XY. Entonces se ha empleado una malla para dividirlo en cuadrados de 2 mm de lado, ya que esa es precisión promedio de las que indicaba el Optitrack para los experimentos realizados con las calibraciones de mejor calidad. Por tanto, es razonable asumir que todos los datos que quedan dentro de la misma cuadrícula se pueden considerar el mismo punto.

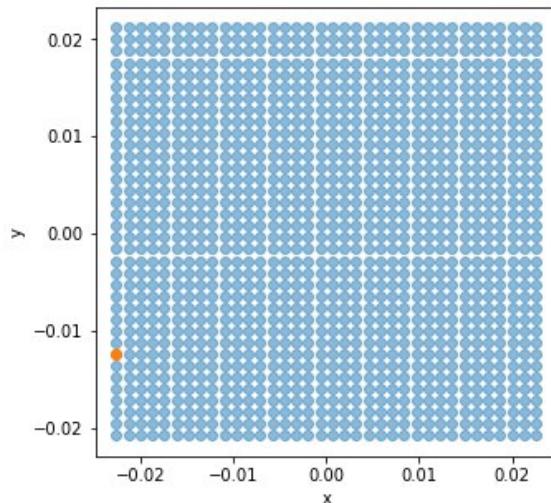


Figura 6.6: Malla de puntos de 2 mm de los que se generan datos para entrenar la cinemática inversa, con el punto de partida del algoritmo indicado en naranja.

Ahora el problema se puede redefinir como que partiendo de un punto (X_0, Y_0) de la malla (para el que conocemos qué combinación de motores lo produce), queremos encontrar qué combinación de motores produce un punto (X_1, Y_1) adyacente en la malla al punto de partida y queremos ir de uno a otro de forma suave, dando pasos pequeños. Así pues, para encontrar una inversa de la función definida por la red neuronal de la cinemática directa, hay que buscar la posición de los motores que minimiza la distancia a X_1, Y_1 . Que se puede lograr encontrando una solución para la fórmula

$$f(\text{motores}) = (X(\text{motores}) - X_1)^2 + (Y(\text{motores}) - Y_1)^2, \quad (6.7)$$

para cada punto de la malla mediante un algoritmo de optimización local. El algoritmo de optimización local empleado es el L-BFGS-B (H. Byrd et al., 1995), implementado con la función de Scipy para ello (Virtanen et al., 2020).

El punto desde el que se empieza a ejecutar el algoritmo es el que se ha marcado en naranja en la figura 6.6. Tomando este punto como (X_0, Y_0) , el algoritmo de optimización L-BFGS-B va modificando ligeramente las posiciones de los motores en cada iteración y recalculando la posición con la red de la cinemática directa, hasta que encuentra el valor de los motores más similar al de partida que lleva el extremo a los nuevos puntos (X_1, Y_1) , el inmediatamente superior y el inmediatamente inferior en la malla. El proceso se repite usando estos nuevos puntos como nuevos (X_0, Y_0) hasta que se tiene una solución para todos los puntos de la columna. Y del mismo modo, se usan estos puntos para generar los de su derecha y así sucesivamente hasta que se completa la malla.

Una vez que se dispone de un conjunto de datos válido, se entrena una red neuronal igual a la empleada para la cinemática directa, excepto por los valores encontrados para los hiperparámetros. Estos hipervalores son: 0.0005 para el coeficiente de la regularización L^2 (λ), 0.005 para la tasa de aprendizaje y una sola capa oculta con 100 unidades.

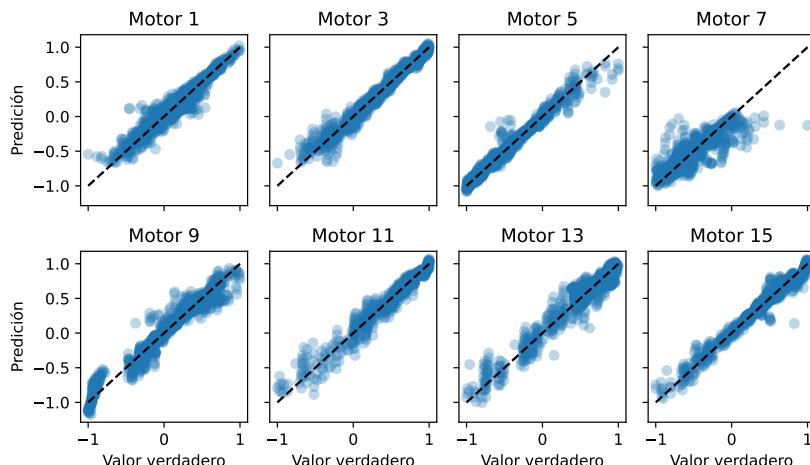


Figura 6.7: Resultado de entrenar la red neuronal para la cinemática inversa. Cada punto azul representa un punto y la línea negra discontinua representa la recta $x=y$.

La figura 6.7 muestra el resultado obtenido, donde cada subfigura representa el resultado para uno de los 8 motores impares que predice la red neuronal. El eje de las x de cada una de ellas muestra un valor del motor para el cual la cinemática directa devuelve el correspondiente punto de la malla; y el eje de las y muestra el valor que predice la red neuronal para esa posición del extremo. Como con la cinemática directa, estas predicciones se han obtenido por validación cruzada, es decir que, como en el caso anterior, la predicción representada para cada punto se corresponde a la partición en la que ese dato no se ha empleado para el entrenamiento. El coeficiente de determinación de la regresión estimado por validación cruzada es de $r^2 = 0,927$

Ambas redes se han entrenado en python, por lo que para poder implementarlas en Matlab para usarlas para controlar el robot hay que exportar las matrices de los pesos y los sesgos obtenidos y resolver las ecuaciones 6.3 y 6.4 usando como x_t el valor reescalado del punto del espacio para al que se desea llevar el robot. Hay que recordar deshacer el reescalado de la solución obtenida y encontrar los valores de los motores pares antes de intentar enviar el comando a los motores.

7. CONCLUSIONES Y LÍNEAS FUTURAS

En este trabajo se pretendía mejorar la calidad del control del robot Pylori-I del CAR mediante redes neuronales y comprobar si estas son un buen método para obtener un control más fino y de mayor calidad en robots con características similares. Para ello se ha empezado por reparar el robot, introduciendo algunas mejoras tanto en el hardware como en el software para devolver el robot a un estado funcional, y después se han tomado datos de su comportamiento. Se ha entrenado una red neuronal hacia delante para resolver ambos sistemas del bucle de control. Es decir, una para la cinemática directa y otra para la cinemática inversa, para la que ha sido necesario generar nuevos datos empleando la red obtenida para la cinemática directa. En ambos casos se ha empleado el optimizador Adam, regularización L^2 y early stopping y se han obtenido los hiperparámetros por validación cruzada; obteniendo redes neuronales con coeficientes de determinación de la regresión de $r^2 = 0,86$ y $r^2 = 0,927$ respectivamente.

Como se ha comprobado directamente sobre el comportamiento del robot para comandos obtenidos mediante las redes, este resultado es suficiente, no solo para implementarlas, sino para mejorar considerablemente el comportamiento del robot respecto al método de control previamente implementado. Incluso a pesar de los efectos inesperados de los problemas mecánicos en el movimiento. Por tanto, se puede considerar que el objetivo principal del trabajo se ha cumplido.

Sin embargo, también se observa que la calidad del control obtenido queda lejos de ser óptima, por lo que es necesario analizar tanto las posibles razones de que así sea, cómo qué implicaciones tienen los resultados obtenidos de cara al uso de redes neuronales en el control de robots similares en el futuro.

Para empezar, la red neuronal de la cinemática inversa se ha entrenado sobre datos obtenidos mediante simulación con la cinemática directa que no se han podido replicar tomando medidas con el Optitrack debido a la avería del ordenador que contenía la licencia. También se podría haber aplicado algún otro método de los que se conocen para compensar la redundancia al mismo tiempo que se entrena la red en este tipo de problemática (Wang et al., 2021). Sin embargo, dado que la principal razón para construir robots hiperredundantes de tipo continuo es que son ideales para aplicaciones que requieren controlar la posición de todo el brazo robótico al llevar el extremo a un punto, puede resultar más conveniente tomar medidas de cada final de sección (en vez de únicamente del extremo) para entrenar una red neuronal capaz de controlar la posición de todo el robot. Dado que la posición del robot es única para cada combinación de posiciones de los motores, esto eliminaría el problema de la redundancia. Y si bien es cierto que aumentaría el número de variables con las que debe trabajar la red neuronal, dada la sencillez del modelo actual, la complejidad añadida al modelo no sería tanta como para que suponga un problema a la hora de entrenarla o implementarla. En este trabajo se realizó un intento de implementar esta solución, pero debido al escaso tamaño del brazo robótico en relación con los marcadores disponibles y a la precisión del Optitrack, no era posible para este distinguir varios puntos tan próximos entre sí.

Otro factor que afecta considerablemente a la calidad de las redes neuronales obtenidas es la calidad y cantidad de datos. Las redes neuronales como las empleadas son muy sensibles al ruido y están diseñadas para interpolar en vez de extrapolar, por lo que no es suficiente con tomar muchos datos, sino que también deben tener suficiente calidad y estar distribuidos por todo el espacio de trabajo. Como se ha comprobado en este proyecto, eso requiere una cierta robustez de la estructura del robot, ya que las redes neuronales son capaces de obtener buenos resultados cuando algún factor mecánico que no se ha considerado en el diseño introduce un error sistemático en el movimiento, pero no cuando se trata de uno aleatorio o que no dependan

de los comandos que se le envían al robot en cada momento. Lo mismo se aplica a los errores que pueda introducir el sistema de medida de unos experimentos a otros.

Por tanto, si se desea seguir mejorando el control del Pylori-I en el futuro, sería necesario encontrar un modo de evitar o reducir el deslizamiento de los discos de las últimas secciones entre sí, y reducir lo suficiente los efectos de las tensiones de los cables sobre las secciones que no controla y sobre otros cables como para poder tomar todos los datos en un solo experimento sin necesidad de recalibrar en medio.

Por último, es perfectamente posible combinar una red neuronal con un modelo que describa la estructura del robot mediante ecuaciones (como el que había diseñado mi compañero Jaime previamente a este proyecto) para lograr el control de robots de tipo continuo. De hecho, partir de uno de estos modelos implementar un método de aprendizaje se ha convertido en una práctica común (Wang et al., 2021), por lo que mejorar el desempeño de este modelo puede mejorar el control significativamente. En este caso, dado que no se han encontrado errores aparentes en la simulación de la estructura y movimiento del robot, se ha llegado a la conclusión de que lo más probable es que la principal fuente de error se encuentre en el modelo empleado para traducir los ángulos de giro de los discos y secciones al alargamiento de cada cable. Como línea futura, se propone implementar un modelo más elaborado que sea mejor prediciendo el efecto de las tensiones de los cables sobre todas las secciones, como por ejemplo los que se presentan en (Jones y Walker, 2006) o (III y Jones, 2010).

8. IMPACTO DEL TRABAJO

Cualquier proyecto de ingeniería debe considerar, no solo el contenido técnico, sino también reflexionar sobre su impacto en la sociedad y el planeta a la hora de considerar tanto su viabilidad como su éxito. Especialmente, en un momento en el que temas como la sostenibilidad medioambiental o la ética en la robótica y el aprendizaje automático han cobrado tanta relevancia en la sociedad que nos rodea y a la que se pretende contribuir positivamente.

Cómo ya se ha mencionado al principio de esta memoria, los robots de tipo continuo hiperredundantes son ideales para automatizar tareas de inspección en áreas como la medicina o labores de rescate, entre otras. Los beneficios de usarlos para tareas como endoscopias o búsqueda de supervivientes entre escombros son claros. Al poder controlar la posición de todo el brazo con precisión, se podrían realizar operaciones y cirugías menos invasivas y más rápidas que si las realizase, solamente un humano, mejorando la tasa de éxito y el tiempo de recuperación. Mientras que en el caso de labores de rescate, permitiría el acceso remoto a zonas de escombros en las que o bien no cabe una persona o bien existe riesgo de derrumbamiento, reduciendo así el riesgo sobre la vida de los rescatadores y aumentando la probabilidad de encontrar supervivientes a tiempo. Además, al contrario que en el caso de muchos otros robots y aplicaciones de inteligencia artificial, este tipo de actividades deben ser supervisadas por un humano, por lo que no generan problemas de destrucción de trabajos.

En cuanto al impacto económico, se puede dividir en los recursos materiales y energéticos consumidos, así como su impacto a largo plazo en la economía. Como se aprecia en el desglose del presupuesto incluido en el 9 la cantidad de recursos empleados y su coste no es muy alto. En cuanto a la cantidad de energía consumida, si bien cualquier algoritmo de aprendizaje automático requiere tiempo de computación tanto para la toma de datos como para entrenar el algoritmo, la red empleada es bastante sencilla, lo que reduce considerablemente el consumo; y aunque en este trabajo no se ha conseguido, una vez que se dispone de una red neuronal de suficiente calidad, no es necesario volver a entrenarla ni a simular cada vez que se quiere mover el robot, por lo que el consumo energético a largo plazo se reduce considerablemente frente al método previamente implementado. Finalmente, hay que considerar que los robots hiperredundantes aún no están ampliamente implantados en la industria al no ser una tecnología lo suficientemente madura y desarrollada para su uso comercial en la mayoría de los campos, por lo que cualquier contribución a la robustez de su diseño o control es un paso hacia su uso comercial que podría suponer un gran desarrollo económico.

9. BIBLIOGRAFÍA

Referencias

- Barrientos, A. et al. (2007). *Fundamentos de robótica*. Biblioteca Hernán Malo González.
- Barrientos-Diez, J., Dong, X., Axinte, D. y Kell, J. (2021). «Real-Time Kinematics of Continuum Robots: Modelling and Validation». En: *Robotics and Computer-Integrated Manufacturing* 67, pág. 102019. ISSN: 0736-5845. DOI: <https://doi.org/10.1016/j.rcim.2020.102019>. URL: <https://www.sciencedirect.com/science/article/pii/S0736584520302301>.
- Bravo Algaba, J. (sep. de 2021). «Optimización del diseño de un robot hiper-redundante de cables mediante algoritmo genético». URL: [https://github.com/JaimeBravoAlgaba/Optimizaci-n-de-CDHRR-de-Discos-Pivotantes-Mediante-Algoritmo-Gen-tico](https://github.com/JaimeBravoAlgaba/Optimizaci-n-de-CDHRR-de-Discos-Pivotantes-Mediante-Algoritmo-Gen-tico/blob/main/Optimizaci%C3%B3n%20del%20Dise%C3%BAo%20de%20un%20Robot%20Hiperredundante%20Mediante%20Algoritmo%20Gen%C3%A9tico.pdf).
- (ago. de 2022). *Optimizaci-n-de-CDHRR-de-Discos-Pivotantes-Mediante-Algoritmo-Gen-tico*. Ver. 1.0.0. URL: <https://github.com/JaimeBravoAlgaba/Optimizaci-n-de-CDHRR-de-Discos-Pivotantes-Mediante-Algoritmo-Gen-tico>.
- Cerrillo Vacas, D. (jul. de 2022). «Metodología de Modelado de Robots Hiperredundantes Actuados por Cables». URL: <https://oa.upm.es/71552/>.
- Dong, X. et al. (abr. de 2017). «Development of a slender continuum robotic system for on-wing inspection/repair of gas turbine engines». En: *Robotics and Computer-Integrated Manufacturing* 44. DOI: [10.1016/j.rcim.2016.09.004](https://doi.org/10.1016/j.rcim.2016.09.004).
- Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press.
- Granna, J., Nabavi, A. y Burgner-Kahrs, J. (nov. de 2018). «Computer-assisted planning for a concentric tube robotic system in neurosurgery». En: *International Journal of Computer Assisted Radiology and Surgery* 14. DOI: [10.1007/s11548-018-1890-8](https://doi.org/10.1007/s11548-018-1890-8).
- Guzmán Merino, M. (Enero de 2020). «Robot continuo actuado con SMA». URL: <https://oa.upm.es/57837/>.
- H. Byrd, R., Lu, P., Nocedal, J. y Zhu, C. (1995). «A Limited Memory Algorithm for Bound Constrained Optimization». En: *SIAM Journal on Scientific and Statistical Computing* 16.5, págs. 1190-1208.
- III, R. y Jones, B. (nov. de 2010). «Design and Kinematic Modeling of Constant Curvature Continuum Robots: A Review». En: *I. J. Robotic Res.* 29, págs. 1661-1683. DOI: [10.1177/0278364910368147](https://doi.org/10.1177/0278364910368147).
- Jones, B. y Walker, I. (2006). «Kinematics for multisection continuum robots». En: *IEEE Transactions on Robotics* 22.1, págs. 43-55. DOI: [10.1109/TRO.2005.861458](https://doi.org/10.1109/TRO.2005.861458).

- Kingma, D. P. y Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. DOI: 10.48550/ARXIV.1412.6980. URL: <https://arxiv.org/abs/1412.6980>.
- Márquez Alcolea, D. (feb. de 2021). «Control y teleoperación de mach i, un robot manipulador híper-redundante para tareas de inspección». URL: <https://oa.upm.es/66646/>.
- Martín-Barrio, A., Terrile, S., Barrientos, A. y Cerro, J. (sep. de 2018). «Robots Hiper-Redundantes: Clasificación, Estado del Arte y Problemática». En: *Revista Iberoamericana de Automatica e Informatica Industrial (RIAI)* 15. DOI: 10.4995/riai.2018.9207.
- Martínez Martín, C. (jun. de 2017). «Desarrollo de un robot manipulador blando e híper-redundante». URL: <https://oa.upm.es/47557/>.
- Mitros, Z. et al. (mayo de 2021). «Design and Modelling of a Continuum Robot for Distal Lung Sampling in Mechanically Ventilated Patients in Critical Care». En: *Frontiers in Robotics and AI* 8. DOI: 10.3389/frobt.2021.611866.
- Muñoz Sánchez, E. (feb. de 2022a). «Construcción y control de un robot continuo de cables con discos pivotantes». URL: <https://oa.upm.es/69811/>.
- (ago. de 2022b). *Pylori-I*. Ver. 1.0.0. URL: <https://github.com/LnaMzSz/Pylori-I>.
- NaturalPoint, I. D. O. (2021). *Calibration*. https://v22.wiki.optitrack.com/index.php?title=Calibration_pane. Recuperado el 21/08/2022.
- OptiTrack, N. I. D. (feb. de 2012). *Tracking Tools 2.4.0. User's Guide*. NaturalPoint Corporation. 3658 SW Deschutes, Corvallis OR 97333.
- Pedregosa, F. et al. (2011). «Scikit-learn: Machine Learning in Python». En: *Journal of Machine Learning Research* 12, págs. 2825-2830.
- Quigley, M. et al. (mayo de 2009). «ROS: an open-source Robot Operating System». En: *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*. Kobe, Japan.
- Robotics, O. (jun. de 2017). *vrpn_client_ros*. Recuperado el 23/08/2022. ROS.org. URL: http://wiki.ros.org/vrpn_client_ros.
- (ago. de 2018). *ROS Melodic Morenia*. Recuperado el 23/08/2022. ROS.org. URL: <http://wiki.ros.org/melodic>.
- Rodríguez Rodríguez, I. (2019). «MACH I. Un manipulador actuado por cables hiperredundante».
- UPM (2022). *Portal de Transparencia*. <https://transparencia.upm.es/>. Recuperado el 06/09/2022.
- Virtanen, P. et al. (2020). «SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python». En: *Nature Methods* 17, págs. 261-272. DOI: 10.1038/s41592-019-0686-2.

Wang, X., Li, Y. y Kwok, K.-W. (2021). «A Survey for Machine Learning-Based Control of Continuum Robots». En: *Frontiers in robotics and AI* 8, pág. 730330. ISSN: 2296-9144. DOI: 10.3389/frobt.2021.730330. URL: <https://europepmc.org/articles/PMC8527450>.

Wolf, A. et al. (2003). «A mobile hyper redundant mechanism for search and rescue tasks». En: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 3, 2889-2895 vol.3. DOI: 10.1109/IROS.2003.1249309.

«XIX Convenio Colectivo Nacional de Empresas de Ingeniería y Oficinas de Estudios Técnicos» (2019). En: *Boletín Oficial del Estado* 251, págs. 114772-114801.

ANEXO I: PLANIFICACIÓN TEMPORAL Y ESTUDIO ECONÓMICO

En este anexo se incluyen el presupuesto, la Estructura de descomposición del proyecto (EDP) y el diagrama de Gantt.

A. Presupuesto

En este apartado se evalúa del coste del proyecto. Para estimar su valor se ha desglosado en partidas de las que se conoce el valor, que se pueden agrupar en las siguientes categorías: costes de personal, costes de los materiales y costes de amortización. Los costes totales calculados para cada partida se incluyen en la tabla A.1.

Los costes de personal se pueden dividir a su vez en tres partidas: trabajo del alumno, trabajo del profesor y trabajo de los estudiantes de doctorando que han ayudado con el proyecto. La normativa de la universidad determina una dedicación de entre 25 y 30 horas por crédito, que para un Trabajo Fin de Máster de 12 créditos significa entre 300 y 360 horas. Sin embargo, se han calculado 400 horas de dedicación para este trabajo, es decir, un exceso de 40 horas. Como referencia para el coste se ha empleado el XIX Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos publicado en el BOE («XIX Convenio Colectivo Nacional de Empresas de Ingeniería y Oficinas de Estudios Técnicos» 2019) que establece en la tabla salarial (actualizada en 2020 por última vez) un sueldo anual de 26323.57 € para una jornada laboral de 1792 horas anuales. Lo que se traduce por 14,69 €/h.

Los costes horarios del trabajo del profesor y de los estudiantes de doctorando se han estimado a partir de los valores que proporciona el portal de transparencia de la UPM (UPM, 2022). Así, con un sueldo de 49033.42 € anuales para un profesor catedrático a tiempo completo (1792 h anuales), para 22 horas de dedicación estimadas resulta un coste de 601.97 €. En el caso de los doctorandos se ha estimado una dedicación de unas 60 h en conjunto, que mediante un cálculo análogo al caso anterior para su sueldo, suponen 817,65 €.

En la partida de Amortización se incluye la amortización tanto del software como del hardware. La licencia de Matlab 65 € anuales, pero dado que la proporciona gratuitamente la UPM a sus estudiantes y que el programa ya se había instalado para realizar otras tareas antes del comienzo del proyecto, no se ha tenido en cuenta para el coste del proyecto. Tampoco se ha incluido coste para el programa Gantt Project, ya que es de libre distribución.

Los costes de material se dividen en el coste de impresión de las piezas que ha habido que sustituir o modificar y el coste de los tornillos y materiales empleados para reparar y soldar los cables rotos. Las piezas se han impreso o bien en el laboratorio o por la asociación de estudiantes RESET, y su coste se ha estimado en 98 €. Mientras que el coste de los tornillos, conectores y metal de soldadura se ha estimado a partir del presupuesto del proyecto de diseño del robot de mi compañera Elena Muñoz (Muñoz Sánchez, 2022a) por valor de 9,13 €.

El ordenador personal del alumno es un ordenador Xiaomi Laptop Air 13.3. Su precio aproximado fue de 825 € y se le estima un periodo de amortización de 6 años, un número de 320 horas empleadas en el TFG y un empleo medio de 6 horas diarias en promedio. Se le ha denominado ordenador personal nuevo para distinguirlo del ordenador antiguo del alumno en el que se instaló Ubuntu, Ros y Matlab que se empleó para poder enviar los datos de Optitrack a Matlab sin tener que instalar un sistema operativo distinto en el ordenador que se estaba empleando. El

ordenador personal antiguo es un Asus TP300LA que se compró en 2015 por 699 €. Si bien tiene más antigüedad que los años de amortización que se le suponían, se considera que aún tenía cierto valor residual que se ha empleado todo en el proyecto, ya que se averió al final de la toma de datos. En cuanto al ordenador del laboratorio y al Optitrack, si bien la versión del software no era reciente y el ordenador tenía ya cierta antigüedad, al haberse averiado se considera que se ha gastado todo su valor residual. Mientras que el hardware de Optitrack puede emplearse con la nueva licencia que se ha comprado para el laboratorio en otro ordenador. Suponiendo una amortización a 15 años, y basándose en el coste actual de las cámaras que vende Optitrack (2199€ por cámara y 56 € el paquete de 10 marcadores) y considerando que se ha empleado durante aproximadamente 3 semanas, quedan 55.21 €.

Finalmente, considera un sobrecoste de un 5 % sobre el valor total calculado, donde se incluyen desplazamientos, uso de electricidad y otros gastos.

Tabla A.1: Costes del Proyecto.

Partida	Concepto	Coste
Personal	Coste de trabajo del estudiante	5876 €
	Coste de trabajo del profesor	601.97 €
	Coste de trabajo de los doctorandos	817,65 €
Materiales	Coste de impresión de los soportes y poleas	98 €
	Coste de material de soldadura y tornillos	9,13 €
Amortización	Amortización del ordenador personal nuevo	19.20 €
	Amortización del ordenador personal antiguo	15 €
	Amortización del ordenador del laboratorio	200 €
	Amortización de Optitrack	55.21 €
Sobrecoste	Desplazamientos, electricidad y otros gastos	384.61 €
Total		8076.77 €

B. Estructura de Descomposición del Proyecto (EDP)

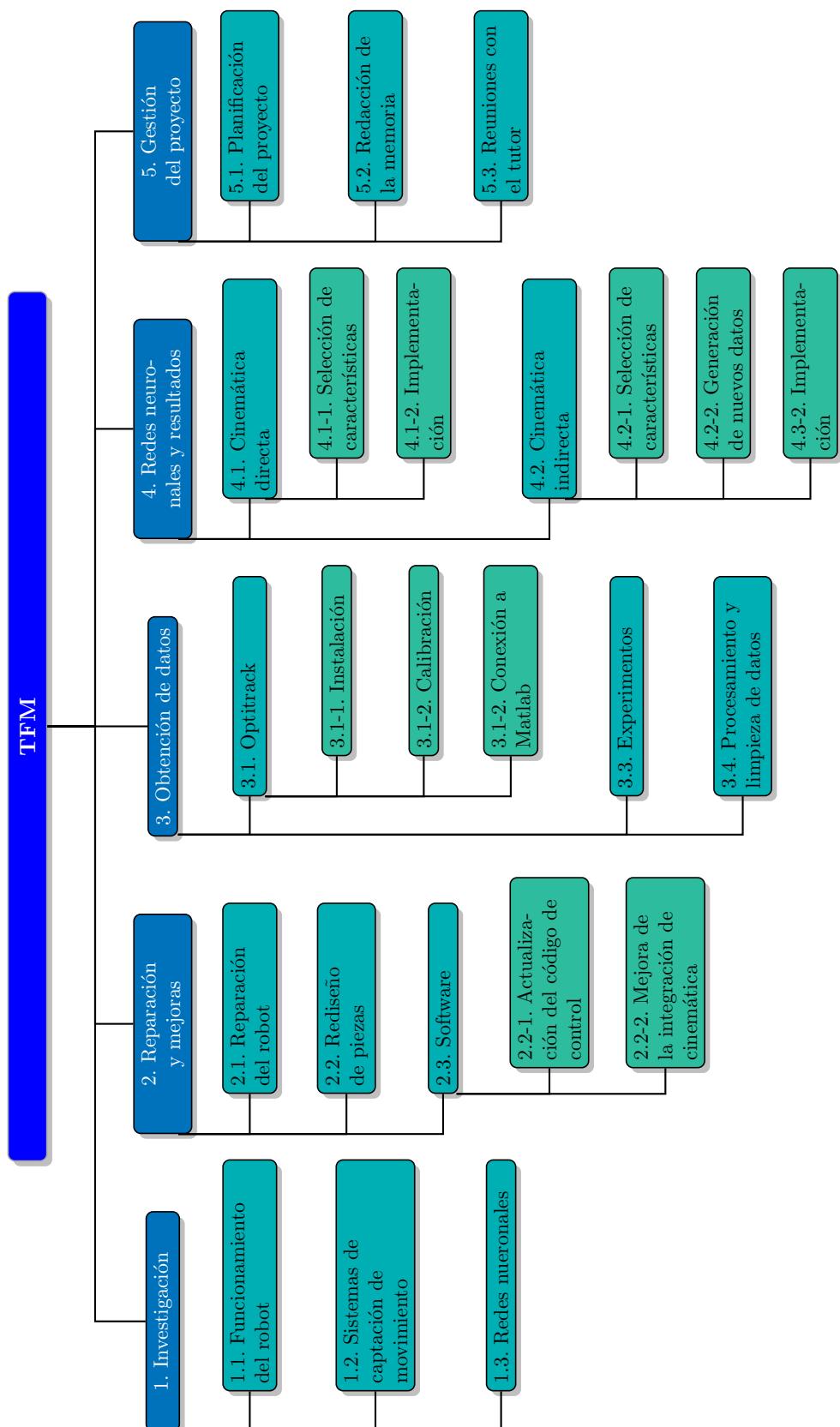


Figura B.1: Estructura de Descomposición del Proyecto (EDP)

C. DIAGRAMAS DE GANTT

El siguiente diagrama de Gantt se ha realizado con el programa GanttProject.

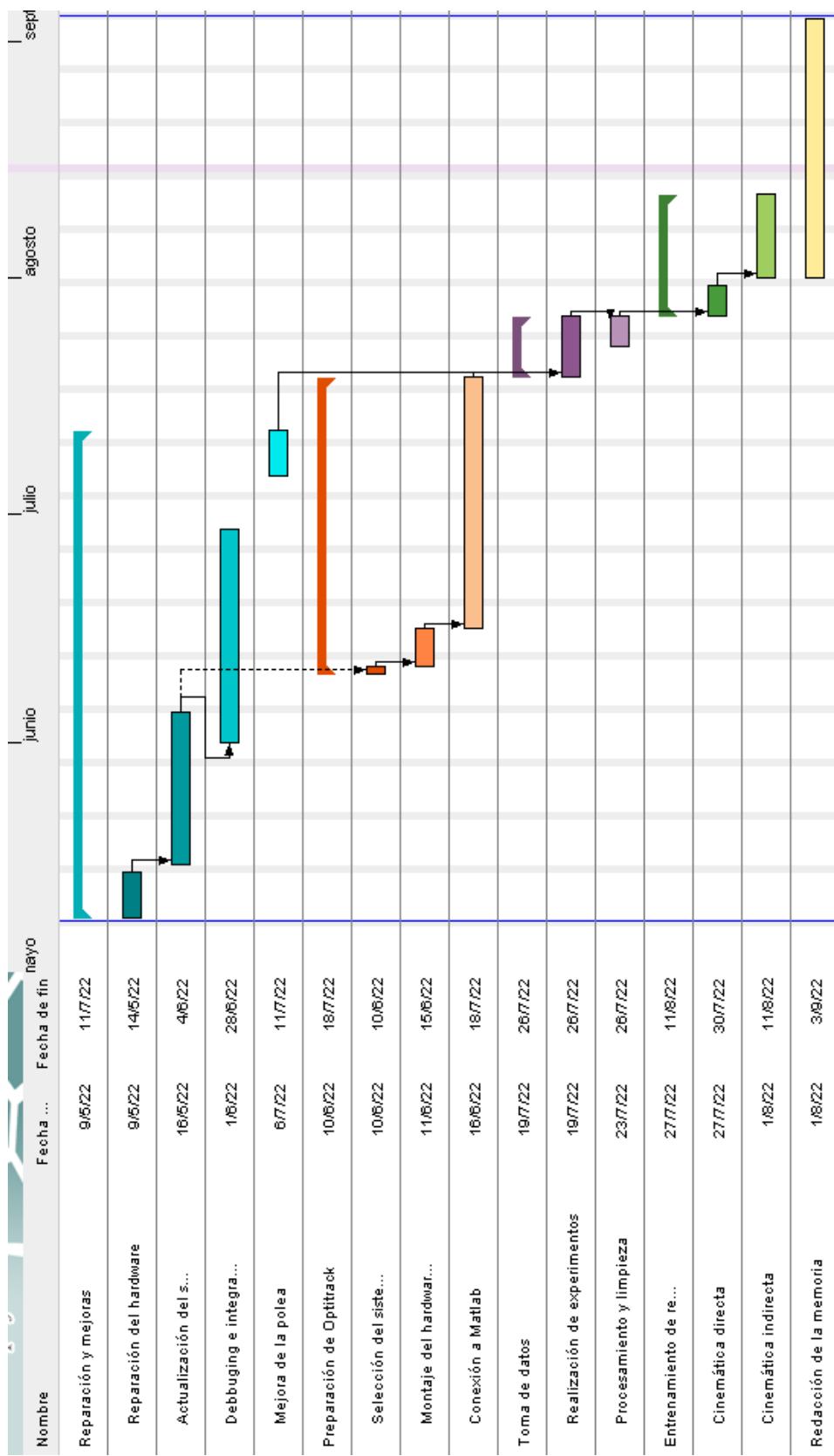


Figura C.2: Diangrama de Gantt del proyecto.

ANEXO II: GLOSARIO Y ABREVIATURAS

- **CAR:** Centro de Automática y Robótica
- **GdL:** Grados de libertad.
- **L-BFGS-B:** Algoritmo de optimización local de tipo quasi-Newton
- **MPL:** Multilayer perceptron o red neuronal hacia delante.
- **MTH:** Matriz de Transformación Homogénea.
- **NN:** Red neuronal
- **ROS:** Robot Operating System. Conjunto de librerías y herramientas de software creado para facilitar el desarrollo de software para robots
- **VRPN:** Virtual Reality Peripheral Network.

ANEXO III: CÓDIGO

Dada la longitud del código escrito durante el desarrollo de este proyecto y la gran cantidad de script en la que se organiza, en vez de plasmarlo en esta memoria, se ha optado por proporcionar indicar el repositorio de Github donde se recoge para que el lector pueda acceder a él libremente.

El enlace al repositorio es el siguiente: <https://github.com/CeliadelRio/Pylori-I/>



POLITÉCNICA

**ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES
UNIVERSIDAD POLITÉCNICA DE MADRID**

José Gutiérrez Abascal, 2. 28006 Madrid
Tel.: 91 336 3060
info.industriales@upm.es

www.industriales.upm.es