

UNIVERSIDAD CARLOS III  
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA



**TESIS DE MÁSTER**

**CONTROLADOR CARTESIANO PARA EL BRAZO  
LWR-UC3M-1 DEL ROBOT MANFRED CON  
DETECCIÓN DE CONTACTO**

Autor: David Álvarez Sánchez  
Director: Dr. Luís Moreno Lorente

MÁSTER OFICIAL EN  
ROBÓTICA Y AUTOMATIZACIÓN

LEGANÉS, MADRID  
OCTUBRE 2011



UNIVERSIDAD CARLOS III  
MÁSTER OFICIAL EN ROBÓTICA Y AUTOMATIZACIÓN

El tribunal aprueba la tesis de Master titulada "**Controlador cartesiano para el brazo LWR-UC3M-1 del robot MANFRED con detección de contacto**" realizado por **David Álvarez Sánchez** como requisito para la obtención del Máster Oficial en **Robótica y Automatización**.

Fecha: Octubre 2011

Tribunal:

---

Dra. Dolores Blanco Rojas

---

Dr. Santiago Garrido Bullón

---

Dr. Higinio Rubio Alonso



*A mi familia, por apoyarme incondicionalmente en las decisiones que tomo.*

*A Luís Moreno, mi tutor, por darme la oportunidad de trabajar con Manfred y por su ayuda a lo largo del desarrollo de esta tesis.*

*A mis compañeros de despacho y departamento, por hacer que resulte agradable venir a trabajar, y estar dispuestos a echar una mano siempre que a uno le invaden las dudas. Especialmente a Fran, por tener una respuesta para las innumerables preguntas que me han surgido a lo largo de estos meses.*

*Al grupo de reunión de los viernes, porque siempre es mejor terminar la semana de trabajo entre amigos.*



# Índice general

<b>Índice de figuras</b>	<b>IX</b>
<b>Resumen</b>	<b>XIII</b>
<b>Abstract</b>	<b>XV</b>
<b>1. Introducción y objetivos.</b>	<b>1</b>
1.1. El problema de la manipulación de objetos. . . . .	1
1.2. Objetivos del proyecto. . . . .	3
<b>2. Fundamentos básicos del control del movimiento de un brazo robótico.</b>	<b>5</b>
2.1. Cinemática de un manipulador. . . . .	5
2.1.1. Cinemática directa. . . . .	6
2.1.1.1. Algoritmo de Denavit-Hartenberg. . . . .	7
2.1.2. Cinemática inversa. . . . .	10
2.1.2.1. Métodos geométricos y analíticos. . . . .	11
2.1.3. Puntos singulares de una cadena cinemática. . . . .	12
2.2. Control del movimiento de un manipulador. . . . .	13
2.2.1. Control en el espacio articular. . . . .	13
2.2.2. Control en el espacio operacional. . . . .	15

2.3.	Tipos de trayectorias en el control cinemático. . . . .	16
2.3.1.	Trayectorias punto a punto. . . . .	17
2.3.2.	Trayectoria coordinada. . . . .	17
2.3.3.	Trayectoria continua. . . . .	18
2.4.	Sistemas de detección de colisión. . . . .	18
<b>3.</b>	<b>Plataforma experimental.</b>	<b>21</b>
3.1.	Manfred. . . . .	21
3.1.1.	Estructura del robot. . . . .	23
3.1.2.	Sistema sensorial. . . . .	25
3.1.3.	Sistema locomotor. . . . .	30
3.1.4.	Sistema de alimentación. . . . .	31
3.1.5.	Sistema manipulador LWR-UC3M-1. . . . .	32
3.1.5.1.	Parámetros de D-H. . . . .	34
3.1.6.	Sistema de control. . . . .	36
3.1.6.1.	Tarjeta controladora PMAC2-PCI. . . . .	36
3.1.7.	Sistema de procesamiento. . . . .	41
3.2.	Linux. . . . .	42
3.3.	Matlab. . . . .	43
3.4.	ROS. . . . .	45
3.5.	KDL. . . . .	50
<b>4.</b>	<b>Solución propuesta.</b>	<b>51</b>
4.1.	Controlador cartesiano. . . . .	51
4.1.1.	Jacobiana inversa. . . . .	53
4.1.2.	Implementación en ROS. . . . .	53
4.1.2.1.	Primera solución propuesta. . . . .	54
4.1.2.2.	Controlador de trayectoria coordinada. . . . .	59
4.1.2.3.	Controlador de trayectoria continua. . . . .	62
4.2.	Librería para el sensor JR3 . . . . .	63

4.2.1. Ejemplo aplicación para detectar contacto. . . . .	65
<b>5. Resultados experimentales.</b>	<b>69</b>
5.1. Primera solución propuesta. . . . .	69
5.2. Controlador de trayectoria coordinada y continua. . . . .	71
5.3. Detector de contacto con el sensor JR3. . . . .	76
<b>6. Conclusiones y futuras líneas de trabajo.</b>	<b>81</b>
6.1. Conclusiones. . . . .	81
6.2. Mejoras. . . . .	82
6.3. Futuras líneas de trabajo. . . . .	83
<b>Referencias</b>	<b>85</b>



# Índice de figuras

2.1.	Parámetros de D-H de una articulación giratoria. . . . .	9
2.2.	Esquema básico de control en el espacio articular. . . . .	14
2.3.	Esquema básico de control en el espacio operacional. . . . .	16
3.1.	Manipulador móvil Manfred. . . . .	22
3.2.	Vista lateral de Manfred. . . . .	24
3.3.	Telémetro láser Hokuyo UTM-30LX. . . . .	26
3.4.	Telémetro láser bidimensional PLS de Sick. . . . .	27
3.5.	Cámaras de color de Manfred. . . . .	28
3.6.	Cámara time-of-flight. . . . .	28
3.7.	Sensor de fuerza/par JR3. . . . .	29
3.8.	Base del manipulador móvil. . . . .	31
3.9.	Sistema de alimentación de Manfred. . . . .	32
3.10.	Brazo de Manfred. . . . .	33
3.11.	Sistemas de referencia según algoritmo D-H. . . . .	35
3.12.	Tarjeta controladora PMAC2-PCI. . . . .	37
3.13.	Accesorio 8E para la interconexión con entre la PMAC y los drivers de potencia. . . . .	38
3.14.	Arquitectura del sistema software de Manfred. . . . .	42
3.15.	Manipulador LWR-UC3M-1 simulado con Matlab. . . . .	44

4.1.	Diagrama de bloques del algoritmo de cinemática inversa. . . . .	54
4.2.	Implementación bajo ROS del diagrama de la figura 4.1. . . . .	54
4.3.	Diagrama de bloques de la alternativa propuesta. . . . .	59
4.4.	Perfiles de posición, velocidad y aceleración respectivamente. . .	62
4.5.	Diagrama de bloques del detector de contacto. . . . .	67
5.1.	Primera posición de la trayectoria, $x = 0,75, y = 0,25, z = 0,25$ . . . . .	70
5.2.	Posición intermedia incorrecta, $x = 0,81, y = 0,14, z = 0,25$ . . . . .	70
5.3.	Tercera posición de la trayectoria, $x = 0,72, y = 0,27, z = 0,25$ . . . . .	71
5.4.	Evolución de la trayectoria coordinada en el eje x. . . . .	73
5.5.	Evolución de la trayectoria coordinada en el eje y. . . . .	73
5.6.	Evolución de la trayectoria continua en el eje x. . . . .	74
5.7.	Evolución de la trayectoria continua en el eje y. . . . .	74
5.8.	Evolución de la trayectoria en el eje y. . . . .	75
5.9.	Evolución de la trayectoria en el eje y. . . . .	75
5.10.	Medidas del sensor en el eje x. . . . .	77
5.11.	Medidas del sensor en el eje y. . . . .	77
5.12.	Medidas del sensor en el eje z. . . . .	77
5.13.	Medidas del sensor en el eje x después del filtro de mediana. . . .	78
5.14.	Medidas del sensor en el eje y después del filtro de mediana. . . .	79
5.15.	Medidas del sensor en el eje z después del filtro de mediana. . . .	79

# Resumen

Esta tesis fin de máster se desarrolla en el marco de un proyecto denominado: "Técnicas de aprendizaje y planificación para manipulación diestra en manipuladores móviles (Dex-arm)", centrado en la investigación sobre técnicas de aprendizaje y manipulación de objetos, y el desarrollo de un conjunto brazo-mano que permitan acercar progresivamente las capacidades de los robots actuales a las capacidades de manipulación que ofrece el sistema brazo-mano de los seres humanos.

Una de las líneas de trabajo del proyecto corresponde con el diseño y desarrollo de métodos de planificación y manipulación que permitan combinar la actuación en situaciones predefinidas, basada en las capacidades aprendidas, junto con las situaciones no previstas detectadas mediante los sistemas sensoriales del robot. Dentro de esta línea de investigación, el proyecto trata los aspectos que tienen que ver con la ejecución de trayectorias por parte del brazo robótico de un manipulador móvil y el uso de un sensor de fuerza/par situado en su muñeca.

El proyecto ha sido desarrollado utilizando el robot Manfred (Man Friendly), un manipulador móvil dotado del brazo robótico LWR-UC3M-1, de seis grados

de libertad, que le permite realizar tareas de manipulación en un entorno humano. El objetivo de este proyecto es desarrollar un sistema de control cinemático para el brazo robótico antes mencionado de manera que pueda ejecutar las trayectorias requeridas, así como utilizar un sensor de fuerza/par situado en su extremo para detectar el contacto del efecto final con objetos.

**Palabras clave:** robot, manipulación, sensor fuerza/par, ROS, control de movimiento, detección de contacto

# Abstract

This master thesis is involved in a bigger research project called: "Learning and planification techniques for dexterous manipulation of mobile manipulators (Dex-arm)", whose main research lines are techniques for learning and manipulating objects, and the development of a robotic arm-hand kinematic chain which allows to get closer to the nowadays manipulation capabilities offered by the human arm-hand system.

More specifically, one of the research lines dissertation about the design and development of planification and manipulation methods that permit a joint actuation in predefined situations, based on the learned abilities, and non expected situations that will be detected by the sensorial system of the robot. In this line, the project deals with several topics such as: the execution of trajectories by the robotic arm of a mobile manipulator and the use of a force/torque sensor located at its wrist.

This project has been developed using the robot Manfred (Man Friendly), a mobile manipulator equipped with the robotic arm LWR-UC3M-1, which has 6 degrees of freedom, that allows the robot to do manipulation tasks in a human environment. The goal of this project is to develop a kinematic control system for

the robotic arm afore mentioned so that it is able to execute the requiered trajectories, and to use the force/torque sensor located at the end effector of the arm to detect the contact with an object.

**KEYWORDS:** robot, manipulation, force/torque sensor, ROS, movement control, contact detection

# Capítulo 1

## Introducción y objetivos.

### 1.1. El problema de la manipulación de objetos.

Una de las principales características de la robótica es la constante interacción del robot con el entorno, se recoge información, a través de los sensores, que nos ayuda a reconocer sus características y se ejecutan las acciones que resulten más convenientes usando los actuadores.

Un manipulador móvil consta de al menos un brazo manipulador y de una base móvil sobre la que éste se asienta. Un manipulador móvil autónomo es aquel que es capaz de realizar tareas sin la intervención directa de un operador humano. Para la realización de estas tareas es frecuente que sea necesario una manipulación diestra de objetos, ya que al fin y al cabo, el ser humano manipula objetos para casi cualquier acción de su vida diaria, por lo que éste es un campo de investigación importante en la robótica.

En ausencia de obstáculos, el estudio de manipuladores móviles concierne principalmente al problema de coordinar locomoción y manipulación. Sin embargo, en presencia de obstáculos, deben considerarse simultáneamente los problemas de evitar los obstáculos y la coordinación del movimiento. Este es el caso de los manipuladores móviles que se van a desenvolver dentro de entornos humanos. Además deben ser capaces de interactuar de forma precisa con los objetos que se encuentran en ese entorno, para lo que suelen reunir ciertas características comunes que les permiten alcanzar en la mayor medida posible sus objetivos:

1. Características antropomórficas en cuanto al tamaño y al espacio de trabajo del manipulador.
2. Electrónica de control que permite controlar de manera coordinada los grados de libertad del robot para conseguir una buena calidad en la ejecución del movimiento.
3. Sistema sensorial avanzado que permite una buena extracción de las características del entorno de trabajo.
4. Capacidades para la toma de decisiones basadas en la información sensorial recogida, como pueden ser algoritmos de localización o de planificación de movimientos.

Cualquier tarea de manipulación normalmente puede dividirse en dos pasos: alcanzar la posición y orientación objetivo con la que queremos comenzar la manipulación, y los movimientos necesarios que requiera dicha manipulación (Prats, 2009). En el primero, en ausencia de obstáculos, el objetivo se definirá usando variables cartesianas y por tanto el control del movimiento se realizará en base a la posición del manipulador, o de su velocidad. En cambio, al entrar en contacto con un objeto fijo, el manipulador forma una cadena cinemática

cerrada que produce fuerzas de reacción en el extremo. Así, el control del manipulación en posición no es suficiente debido a la gran velocidad con la que estas fuerzas pueden crecer provocando fácilmente daños, ya sea en el robot o en el objeto que se quiere manipular (Shujun, Chung, y Velinsky, 2005). Por lo tanto, se utilizará un control diferente, conocido como control de fuerza, que permite realizar la manipulación limitando los riesgos. Por otro lado, los entornos de trabajo reales contienen obstáculos y son dinámicos, de modo que el manipulador móvil debe comportarse de manera reactiva respecto al entorno mientras lleva a cabo la tarea. Debido a esto la seguridad se convierte en un factor relevante a la hora elegir la manera en la que se van a ejecutar las tareas para las que el robot haya sido programado, siendo crucial una rápida coordinación entre la información recibida sensorialmente y la toma de decisiones.

## **1.2. Objetivos del proyecto.**

Los objetivos finales del proyecto consisten en desarrollo de ciertas aplicaciones que puedan ser utilizadas en la plataforma de investigación Manfred. Debido a que este robot tiene instalado un sistema operativo para el control de su sistema de procesamiento conocido como ROS (*Robot Operating System*), el primer objetivo consistirá en el estudio de sus características y de las distintas opciones que ofrece para facilitar el desarrollo de aplicaciones. Será necesario estudiar tanto la información general del funcionamiento del sistema operativo, como el código fuente de aplicaciones desarrolladas para otros robots (como el Care-o-bot o el PR2) de manera que se pueda aprovechar la experiencia que ya tienen otros grupos de investigación en el desarrollo bajo este sistema operativo.

A continuación se desarrollan los objetivos principales que se pretenden alcanzar en cada uno de los puntos de la tesis de máster.

1. Implementar un control cartesiano para el movimiento del brazo manipulador LWR-UC3M-1 instalado en Manfred. Para ello será necesario elegir una librería que nos proporcione la posibilidad de realizar la cinemática inversa de un sistema articulado de 6 ejes, ya sea a través de la posición o de la velocidad. Además, será necesario realizar un estudio de las repercusiones dinámicas que tiene el movimiento del brazo robótico para diseñar un controlador que se adecúe a las características para las que Manfred ha sido diseñado. Por último será necesario elegir entre las distintas APIs (*Application Program Interface*) que proporciona ROS para el desarrollo del controlador, de acuerdo con las características de ejecución que tenga cada uno de los bloques en los que se divide el controlador elegido.
2. Generar una API para el sensor de fuerza/par JR3 para facilitar su futuro uso. En ese caso se cuenta con un driver para Linux y una interfaz gráfica del sensor JR3 para Windows realizada por J. Pires. Por lo tanto el primer objetivo es probar la aplicación de Windows para poder decidir cuales son las funciones mínimas necesarias que requiere el uso de este sensor y el segundo estudiar el funcionamiento del driver para comprender cual es la manera de interactuar con el hardware. Además, la interfaz gráfica proporcionada por Pires nos permitirá evaluar de forma rápida y sencilla si el funcionamiento ofrecido por la API que se pretende desarrollar es correcto.
3. Realizar una aplicación ejemplo que une los dos puntos anteriores. Como aplicación se ha decidido realizar un algoritmo capaz de detectar el contacto del efecto final del brazo (una pinza) con un objeto. Para ello será necesario hacer uso de la API desarrollada para el sensor bajo el sistema operativo de ROS, así como introducir el algoritmo dentro del controlador de movimiento del brazo, siendo además deseable el desarrollo de alguna actuación como respuesta al contacto de la pinza con el objeto.

# Capítulo 2

## Fundamentos básicos del control del movimiento de un brazo robótico.

### 2.1. Cinemática de un manipulador.

La cinemática de un manipulador estudia el movimiento del mismo con respecto a su sistema de referencia. La cinemática se interesa por la descripción analítica del movimiento espacial del manipulador como una función del tiempo, y en particular, por la localización del extremo final del manipulador con los valores que toman sus coordenadas. Existen dos problemas a la hora de abordar la cinemática de un manipulador:

1. El problema cinemático directo se reduce a de determinar la localización del extremo final del manipulador respecto a su sistema de referencia.

## **6 Fundamentos básicos del control del movimiento de un brazo robótico.**

---

2. El problema cinemático inverso se encarga de conseguir los valores que tienen que adoptar las articulaciones del manipulador para poder localizar su extremo en una posición y orientación determinada.

A continuación se detallan con más detalle los dos problemas mencionados y los métodos o algoritmos para la obtención de la solución.

### **2.1.1. Cinemática directa.**

Usando el álgebra vectorial y matricial, se representa y se describe la localización de un objeto en el espacio. Se considera que un manipulador es una cadena cinemática compuesta por objetos sólidos rígidos, eslabones, unidos entre ellos mediante articulaciones o ejes. Tomando como referencia el principio del primer eslabón de la cadena cinemática, se pueden describir una serie de relaciones entre cada uno de los eslabones con respecto al anterior, o respecto a la base de referencia. De esta forma, se deduce que:

“El problema cinemático directo se reduce a encontrar la localización del extremo final del robot cuando las variables de sus articulaciones toman valores determinados.”(Barrientos, Peñín, Balaguer, y Aracil, 1997)

La forma más común de resolución de la cinemática directa es la basada en cambios de los sistemas de referencia acoplados en cada articulación. En concreto, mediante el método de las matrices de transformación homogénea, y gracias al álgebra matricial, el problema cinemático directo se reduce a:

“Encontrar una matriz de transformación homogénea que relacione la localización del extremo del robot respecto del sistema de referencia fijo situado en la base del mismo.” (Barrientos y cols., 1997)

Un robot de  $n$  grados de libertad (g.d.l.) está compuesto por  $n$  articulaciones, de forma que cada articulación-eslabón forma un g.d.l. A cada eslabón se le puede asociar un sistema de referencia, y mediante las transformaciones homogéneas se puede representar la localización relativa entre los diferentes eslabones. La matriz  $T$  que relaciona la localización relativa entre dos eslabones consecutivos de la cadena cinemática se denomina  $_{i-1}A^i$ . Así,  $_0A^1$  describe la localización del sistema de referencia solidario al eslabón 1 con respecto al sistema de referencia del eslabón 0 o base (base del manipulador). De esta manera se puede expresar la cadena cinemática entre dos eslabones cualesquiera mediante el producto de estas matrices.

#### 2.1.1.1. Algoritmo de Denavit-Hartenberg.

El método propuesto por Denavit-Hartenberg (D-H en adelante) establece que, para poder realizar los cálculos de la cinemática directa e inversa, y poder así realizar las mediciones necesarias de las trayectorias, necesitamos hallar la matriz de transformación homogénea que relaciona el sistema de referencia colocado en el anclaje del manipulador, con el sistema de referencia situado en el extremo de éste. Para esto es necesario hallar los parámetros D-H del manipulador (Denavit y Hartenberg, 1955). La definición de estos parámetros y la forma correcta de operar con ellos se definen a continuación:

**D-H 1:** Numerar los eslabones comenzando con 1 (primer eslabón móvil de la cadena) y acabando con el  $n$ -enésimo (último eslabón móvil). Se numerará como eslabón 0 al que va directamente anclado a la base fija del manipulador.

**D-H 2:** Numerar cada articulación comenzando por la 1<sup>a</sup> (la correspondiente al primer grado de libertad) y acabando en la  $n$ -enésima.

**D-H 3:** Localizar el eje de cada articulación. Si ésta es rotativa, el eje será su propio eje de giro. Si es prismática, será a lo largo del cual se produce el desplazamiento.

**D-H 4:** Para i de 0 a n-1 situar el eje  $z_i$  sobre el eje de la articulación i+1.

**D-H 5:** Situar el origen del sistema de la base  $S_0$  en cualquier punto del eje  $z_0$ . Los ejes  $x_0$  e  $y_0$  se situarán de modo que formen un sistema dextrógiro con  $z_0$ .

**D-H 6:** Para i de 1 a n-1, situar el sistema  $S_i$  (solidario al eslabón i) en la intersección del eje  $z_i$  con la línea normal común a  $z_{i-1}$  y  $z_i$ . Si ambos ejes se cortasen se situaría  $S_i$  en el punto de corte. Si fuesen paralelos  $S_i$  se situaría en la articulación i+1.

**D-H 7:** Situar  $x_i$  en la línea normal común a  $z_{i-1}$  y  $z_i$ .

**D-H 8:** Situar  $y_i$  de modo que forme un sistema dextrógiro con  $x_i$  y  $z_i$ .

**D-H 9:** Situar el sistema  $S_n$  en el extremo del robot de modo que  $z_0$  coincida con la dirección de  $z_{n-1}$  y  $x_n$  sea normal a  $z_{n-1}$  y  $z_n$ .

**D-H 10:** Obtener i como el ángulo que hay que girar en torno a  $z_{i-1}$  para que  $x_{i-1}$  y  $x_i$  queden paralelos.

**D-H 11:** Obtener  $d_i$  como la distancia, medida a lo largo de  $z_{i-1}$ , que habría que desplazar  $S_{i-1}$  para que  $x_i$  y  $x_{i-1}$  quedasen alineados.

**D-H 12:** Obtener  $a_i$  como la distancia medida a lo largo de  $x_i$  (que ahora coincidirá con  $x_{i-1}$ ) que habría que desplazar el nuevo  $S_{i-1}$  para que su origen coincidiese con  $S_i$ .

**D-H 13:** Obtener  $\alpha_i$  como el ángulo que habría que girar en torno a  $x_i$  (que ahora coincidiría con  $x_{i-1}$ ), para que el nuevo  $S_{i-1}$  coincidiese totalmente con  $S_i$ .

**D-H 14:** Obtener las matrices de transformación  ${}^{i-1}A_i$ .

**D-H 15:** Obtener la matriz de transformación que relaciona el sistema de la base con el del extremo del robot de la siguiente manera:  $T = {}^0A_1, {}^1A_2, \dots, {}^{n-1}A_n$ .

Los cuatro parámetros de D-H ( $\theta_i, d_i, a_i, \alpha_i$ ) dependen únicamente de las características geométricas de cada eslabón y de las articulaciones que le unen con el anterior y el siguiente. En concreto su representación se puede observar perfectamente en la siguiente figura.

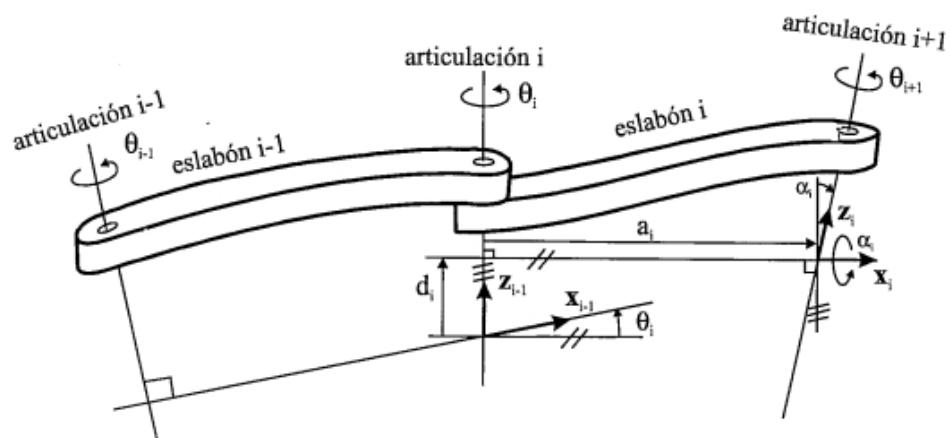


Figura 2.1: Parámetros de D-H de una articulación giratoria.

## 10 Fundamentos básicos del control del movimiento de un brazo robótico.

---

$\theta_i$ : Es el ángulo que forman los ejes  $x_{i-1}$  y  $x_i$  medido en un plano perpendicular al eje  $z_{i-1}$ , utilizando la regla de la mano derecha. Se trata de un parámetro variable en articulaciones giratorias.

$d_i$ : Es la distancia a lo largo del eje  $z_{i-1}$  desde el origen del sistema de coordenadas (i-1)-ésimo hasta la intersección del eje  $z_{i-1}$  con el eje  $x_i$ . Se trata de un parámetro variable en articulaciones prismáticas.

$a_i$ : Es la distancia a lo largo del eje  $x_i$  que va desde la intersección del eje  $z_{i-1}$  con el eje  $x_i$  hasta el origen del sistema i-ésimo, en el caso de articulaciones giratorias. En el caso de articulaciones prismáticas, se calcula como la distancia más corta entre los ejes  $z_{i-1}$  y  $z_i$ .

$\alpha_i$ : Es el ángulo de separación del eje  $z_{i-1}$  y el eje  $z_i$ , medido en un plano perpendicular al eje  $x_i$ , utilizando la regla de la mano derecha.

Con estos parámetros resulta más sencillo encontrar la matriz **A** que define la orientación (submatriz de rotación) y posición (submatriz de traslación) del extremo de un manipulador, referido a la base de éste, en función de las n coordenadas articulares.

### 2.1.2. Cinemática inversa.

El problema cinemático inverso consiste en encontrar los valores que deben adoptar las coordenadas articulares del manipulador para que su extremo se localice según una determinada posición y orientación espacial.

Para la resolución de este problema se han desarrollado algunos procedimientos genéricos que permiten a un ordenador obtener una solución a partir

del conocimiento de la cinemática del robot. Sin embargo, suelen ser métodos numéricos iterativos muy lentos y en los que su convergencia no está garantizada (Goldenberg, Benhabib, y Fenton, 1985). Además, el procedimiento de obtención de las ecuaciones que relacione la localización del extremo con los parámetros físicos del robot para poder obtener los valores articulares es fuertemente dependiente de la configuración del robot. Por lo tanto resulta mucho más interesante encontrar una solución cerrada, ya que ofrece algunas ventajas como:

- En muchas aplicaciones, el problema cinemático inverso se tiene que resolver en tiempo real.
- La solución no es única, existiendo diferentes conjuntos de coordenadas que localizan el extremo del robot en la localización deseada.

#### 2.1.2.1. Métodos geométricos y analíticos.

Los métodos geométricos son muy interesantes en robots sencillos (hasta 3 g.d.l.) o en aquellos en los que se pueda realizar un desacople cinemático, de manera que se puede calcular por separado la posición y orientación que se desea alcanzar. Además, también se suelen utilizar en las aplicaciones en las que no existen restricciones en cuanto a la orientación con la que se alcanza la posición objetivo.

En el caso del manipulador acoplado en el robot Manfred, no se cumple ninguna de las premisas anteriores, por lo que no será posible utilizar un método geométrico para hallar la solución de la cinemática inversa.

Los métodos analíticos tratan de obtener una solución al problema cinemático inverso de un robot a partir del conocimiento de modelo cinemático directo.

## **12 Fundamentos básicos del control del movimiento de un brazo robótico.**

---

Esta tarea no resulta nada trivial ya que resuelto a partir de la matriz de transformación homogénea de un manipulador de 6 g.d.l., se tienen 12 ecuaciones, mientras que se buscan tan sólo 6 relaciones. Esto indica que existirán ciertas dependencias entre las ecuaciones y que será muy difícil escoger aquellas con las que se va a trabajar.

Si en vez de considerar la relación entre las variables articulares y la posición y orientación del robot, consideramos sus respectivas derivadas, esta relación se obtiene a través de la denominada matriz jacobiana, y es relativamente fácil de calcular. Si ahora consideramos la relación contraria, el paso del espacio articular al espacio operacional, tendremos que invertir la matriz anterior. Aunque el planteamiento resulta sencillo, la inversión de una matriz de  $6 \times 6$  elementos (caso de un manipulador de 6 g.d.l.) compuestos por funciones trigonométricas resulta de una gran complejidad. Sin embargo, si la posición y orientación del efecto final son especificadas con un mínimo número de parámetros en el espacio operacional, existen técnicas analíticas que operando sobre la matriz jacobiana del manipulador permiten encontrar la relación entre las velocidades del espacio cartesiano y del espacio articular (Sciavicco y Siciliano, 2005).

### **2.1.3. Puntos singulares de una cadena cinemática.**

Se denomina configuración singular de un robot a aquellas configuraciones en las que el determinante de la matriz jacobina se anula, por lo que en estas configuraciones no existe la jacobiana inversa (Craig, 2006). Esto hace que un incremento infinitesimal de las coordenadas cartesianas, suponga un incremento infinito de las coordenadas articulares, lo que se traduce en que en las inmediaciones de las configuraciones singulares, producir un movimiento del extremo del robot con velocidad cartesiana constante obliga a movimientos de las articulaciones a velocidades inabordables por los actuadores. En una configuración

singular se pierde alguno de los grados de libertad del robot, siendo imposible que su extremo se mueva en una determinada dirección cartesiana. Para evitar la aparición de configuraciones singulares debe considerarse su existencia desde la propia fase de diseño, imponiendo restricciones al movimiento del robot (Barrientos y cols., 1997). Además, se debe tener especial atención a su localización para que sean tenidas en cuenta en la generación de trayectorias.

## 2.2. Control del movimiento de un manipulador.

En una acción de manipulación, el robot se mueve hasta una posición para coger un objeto, transporta el objeto hasta otro punto y lo vuelve a dejar. Estas acciones generales forman parte de casi cualquier tarea de manipulación de alto nivel, como puede ser pintar. Las especificaciones para estas tareas, la trayectoria deseada del efecto final, normalmente son dadas en lo que se denomina el espacio de la tarea, mientras que el control del movimiento del manipulador se realiza en el espacio articular. Este hecho lleva a que existan dos métodos generales para realizar el control del movimiento, un esquema para el control en el espacio articular y otro para el espacio operacional.

### 2.2.1. Control en el espacio articular.

El principal objetivo del control en el espacio articular es diseñar un control por realimentación de manera que las coordenadas articulares  $q(t)$  sigan el movimiento deseado  $q_d(t)$  de la mejor manera posible. Para esto, consideraremos las ecuaciones de movimiento de una manipulador de n grados de libertad expresado en el espacio articular:

$$H(q)\dot{q} + C(q, \dot{q}) + \tau_g(q) = \tau \quad (2.1)$$

## 14 Fundamentos básicos del control del movimiento de un brazo robótico.

donde  $H(q)$  es la matriz de inercia ( $n \times n$ ),  $C(q, \dot{q})$  es el vector de Coriolis y fuerzas centrífugas ( $n \times 1$ ),  $\tau_g(q)$  es el vector de fuerza de gravedad ( $n \times 1$ ) y  $\tau$  es el vector de entradas de control articular ( $n \times 1$ ) que se quiere diseñar. Las fricciones y las entradas de ruido no son consideradas.

En este caso, al ser las entradas de control pares de los motores, el control ha de realizarse en el espacio articular. Sin embargo, el usuario especificará el movimiento en términos de coordenadas del efecto final, por lo que resulta necesario realizar una estrategia como la propuesta en la figura 2.2 (Siciliano, Khatib, y cols., 2008), que muestra el esquema básico de los métodos de control en el espacio articular. Se puede ver como, primero se convierte el movimiento deseado, descrito en coordenadas relativas al efecto final, a su correspondiente trayectoria articular usando la cinemática inversa del manipulador. Despues el controlador por realimentación determina el par motor necesario para mover el manipulador a lo largo de la trayectoria especificada en coordenadas articulares basándose en la posición actual de las articulaciones.

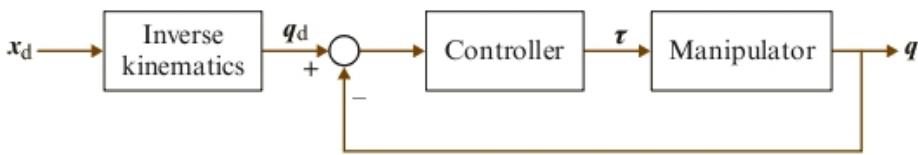


Figura 2.2: Esquema básico de control en el espacio articular.

Como siempre se asume que la tarea está descrita como una secuencia de posiciones de las articulaciones con respecto al tiempo, los esquemas de control en el espacio articular resultan adecuados cuando la tarea a ejecutar por el manipulador puede ser planeada con antelación y apenas se necesitan ajustes de la trayectoria en tiempo de ejecución. Típicamente, la cinemática inversa se calcula para puntos contiguos más o menos cercanos de la trayectoria de la tarea, y la trayectoria articular se interpola usando las soluciones calculadas. Aunque la

trayectoria en el espacio de la tarea consista en movimientos en línea recta entre puntos de interpolación consecutivos, el movimiento articular resultante consistirá en segmentos curvilíneos que coincidirán con la trayectoria del efecto final en los puntos de interpolación.

### 2.2.2. Control en el espacio operacional.

En entornos más complicados y menos seguros, el movimiento del efecto final está sujeto a modificaciones en tiempo de ejecución, de manera que la trayectoria pueda adaptarse a cambios en el entorno detectados por los sensores. Existen numerosas tareas en las que este tipo de problemas pueden aparecer, en general, en todas aquellas en las que controlar la interacción entre el manipulador y el entorno resulta importante.

Como la tarea a desarrollar está normalmente expresada en términos del espacio operacional y requiere de un control preciso en el movimiento del efecto final, es necesario hacer uso de esquemas de control basados en la dinámica expresada en el espacio operacional. Supongamos que la matriz jacobiana,  $J(q) \in R_{nxn}$ , transforma la velocidad de las articulaciones,  $\dot{q} \in R^n$ , en velocidades cartesianas,  $\dot{x} \in R^n$ , de acuerdo con:

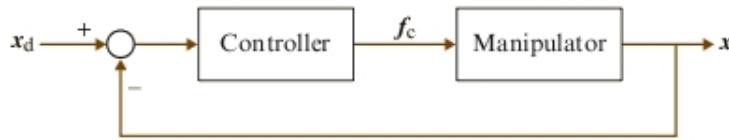
$$\dot{x} = J(q)\dot{q} \quad (2.2)$$

Si, además, asumimos que es invertible, la dinámica en el espacio operacional queda de la siguiente manera:

$$f_c = \Lambda(q)\ddot{x} + \Gamma(q, \dot{q})\dot{x} + \eta(q) \quad (2.3)$$

donde  $f_c$  denota las fuerzas comandadas al robot en el espacio operacional.

El principal objetivo del control en el espacio operacional es diseñar un control por realimentación que permita ejecutar el movimiento del efecto final  $x(t) \in R_{nxn}$  siguiendo el movimiento deseado  $x_d(t)$  de la mejor manera posible. Considerando la ecuación 2.3, el siguiente esquema muestra el diagrama general de los métodos de control del espacio operacional (Siciliano y cols., 2008). Algunas de las ventajas de éstos surgen debido a que el empleo del lazo de realimentación minimiza el error final en el espacio operacional. La cinemática inversa debe ser calculada de manera explícita ya que el algoritmo de control integra el nivel de velocidad en la cinemática directa.



**Figura 2.3:** Esquema básico de control en el espacio operacional.

### 2.3. Tipos de trayectorias en el control cinemático.

El control cinemático establece cuales son las trayectorias que debe seguir cada articulación del robot a lo largo del tiempo para lograr alcanzar los distintos puntos de la trayectoria en el espacio cartesiano. De manera genérica, el control cinemático debe muestrear la trayectoria cartesiana para obtener un numero finito de puntos en ésta; convertir cada uno de estos puntos en una configuración articular alcanzable por el manipulador; e interpolar estos puntos de manera

que se obtengan  $q_i(t)$  trayectorias articulares que debe realizar cada eje del manipulador, pasando lo más cerca posible de ellos. Existen diversas maneras de encontrar estas trayectorias  $q_i(t)$ , aunque se pueden clasificar en tres tipos: trayectorias punto a punto, trayectorias coordinadas y trayectorias continuas.

### 2.3.1. Trayectorias punto a punto.

En este tipo de trayectoria, cada articulación evoluciona desde su posición inicial a la final sin realizar consideración alguna sobre el estado o evolución de las demás articulaciones (Barrientos y cols., 1997). Existen dos tipos de trayectorias punto a punto:

1. Movimiento eje a eje: cada uno de los ejes del manipulador ejecuta la trayectoria en momentos distintos en el tiempo, empieza la primera articulación, y cuando ésta haya alcanzado su destino final, comienza la segunda y así sucesivamente. El tiempo de ejecución de la trayectoria es muy elevado, aunque el consumo instantáneo del potencia es mínimo.
2. Movimiento simultáneo de ejes: en este caso todos los actuadores comienzan simultáneamente a mover las articulaciones del robot a una velocidad específica para cada una de ellas. Como la distancia a recorrer y la velocidad es diferente, cada articulación acabará su movimiento en un instante diferente. El tiempo de ciclo es menor que en el caso anterior, y será el de la articulación que llegue última a su destino.

### 2.3.2. Trayectoria coordinada.

Para evitar que unos actuadores fuerzen sus velocidades de manera innecesaria teniendo que esperar después a que termine el más lento, antes de comenzar el movimiento se calcula cuál es la velocidad que corresponde a cada eje de

manera que todos empiecen y acaben en el mismo momento. Así, se asegura que no se pidan velocidades y aceleraciones elevadas a ningún motor, aunque la trayectoria cartesiana descrita por el efecto final entre cada punto a alcanzar no sea significativa.

### **2.3.3. Trayectoria continua.**

Cuando se pretende que la trayectoria del extremo sea conocida por el usuario (en el espacio cartesiano), es necesario calcular de manera continua las trayectorias articulares, que presentarán una compleja evolución temporal. Aunque el resultado obtenido para articulación puede parecer algo caótico, el resultado del movimiento conjunto de los ejes será que el extremo del robot describirá una trayectoria muy parecida a la deseada por el usuario.

## **2.4. Sistemas de detección de colisión.**

Debido a que los manipuladores móviles están pensados para trabajar en entornos poco estructurados y dinámicos, y además pueden compartir espacio de trabajo con humanos y colaborar con ellos en ciertas tareas, la seguridad se convierte en un área de estudio muy importante. Se debe tratar de evitar las colisiones accidentales que puedan dañar al operario, anticipándose ante situaciones peligrosas. En caso de no ser evitables, se debe asegurar que los efectos de este accidente puedan ser minimizados debido a una correcta reacción del robot que permita recuperar el estado de seguridad.

La investigación en interacción humano-robot trata diferentes aspectos relacionados con la fiabilidad: el diseño mecánico intenta reducir el peso y la inercia

del manipulador; se introducen componentes flexibles que reducen la severidad de los impactos; se usan sensores externos que permiten una detección más rápida de la proximidad del operario; se intentan mejorar las estrategias de planificación del movimiento y el control en su ejecución para disminuir los riesgos de colisión. Cada uno de estos aspectos es relevante en alguna de las fases de la interacción humano-robot.

En la fase previa al impacto, el principal objetivo es evitar la colisión y requiere, al menos, del conocimiento del entorno local y técnicas de planificación de movimiento computacionalmente intensas. Además, algunos fabricantes de robots proveen herramientas hardware y software que permiten describir áreas de seguridad, con variables cartesianas, que no deberían ser utilizadas por el robot en ningún momento. Así mismo, se suelen usar sensores externos como por ejemplo cubiertas sensitivas (Lumelsky y Cheung, 1993), sensores de fuerza o sistemas de visión para tratar de anticipar las colisiones (Ebert y Henrich, 2002).

Una vez ocurrido el impacto, las fuerzas resultantes producidas por éste pueden ser minimizadas intentando diseñar robots más ligeros (Hirzinger, Albuschäffer, Höhnle, Schaefer, y Sporer, 2001), añadiendo cubiertas viscoelásticas a los eslabones del manipulador o introduciendo flexibilidad en el sistema de accionamiento de manera que se pueda desacoplar mecánicamente la inercia del motor de la inercia del eslabón (Zinn, Khatib, Roth, y Salisbury, 2005). También se pueden utilizar actuadores de rigidez variable, de manera de cuando se quiera ejecutar movimientos muy precisos a baja velocidad se endurecerán las articulaciones y cuando se ejecuten movimientos a mayor velocidad se relajen (Tonietti, Schiavi, y Bicchi, 2005).

En la fase posterior al impacto, lo primero y más importante es ser capaces de detectar la colisión, que puede haber ocurrido en cualquier punto a lo largo

del brazo robótico. Una vez detectada, el controlador debe activar una estrategia adecuada, siendo la más sencilla simplemente parar el robot. La detección del contacto puede realizarse con el uso de sensores, ya sean internos o externos al robot.

Entre los sensores internos más corrientes encontramos los medidores de corriente. Su esquema básico de funcionamiento consiste en medir la corriente que entregan los drivers a los motores para poder conocer la fuerza que está desarrollando, y se compara con la fuerza necesaria que debiera realizar ese motor para ejecutar el movimiento deseado según el modelo dinámico del robot (Luca, Hirzinger, Albu-Schäffer, y Haddadin, 2006). Este modelo tiene algunos inconvenientes como son: requiere un buen conocimiento del modelo dinámico del robot; aún cuando el modelo es perfectamente conocido, el cálculo de la dinámica inversa requiere hacer estimaciones de la aceleración, que introducen ruido debido diferenciación numérica de los datos de posición o velocidad; resulta bastante complicado establecer un límite para determinar si hay o no colisión debido a las grandes variaciones dinámicas que producen diferente movimientos del robot; y por último introduce un evidente retardo debido al procesamiento digital.

Entre los sensores externos se pueden usar: sensores de fuerza con los que se pueden buscar picos inesperados en las medidas de fuerza, tienen los inconvenientes de que no es fácil establecer el límite de los picos, y que la detección del contacto empeora al alejarnos del punto donde se coloca el sensor; acelerómetros que se usan igual que los sensores de fuerza pero analizando en este caso las medidas de las aceleraciones del brazo robótico; o cámaras 2D/3D a partir de las cuales se puede hacer un modelado geométrico del entorno (incluido el brazo) para determinar cuando el modelo del brazo ha entrado en contacto con otro objeto.

Capítulo **3**

## Plataforma experimental.

### 3.1. Manfred.

El manipulador móvil Manfred se presenta en la Figura 3.1. Se trata de un manipulador con 8 g.d.l. Consta de una base móvil de tipo diferencial y 2 grados de libertad, junto con un brazo ligero de 6 grados de libertad y diseño antropomórfico. Manfred está diseñado para ser capaz de desenvolverse de forma autónoma por entornos diseñados para personas como: pasillos, salas, despachos, etc., donde en ocasiones resulta necesario abrir y atravesar puertas, evitar obstáculos o coger y manipular objetos (Blanco, Ansari, Castejón, Boada, y Moreno, 2005). Todo esto requiere que el robot integre los siguientes aspectos:

- Todas las capacidades básicas de un robot móvil para desplazarse de forma segura y autónoma por el entorno.
- La coordinación motora entre base y manipulador.
- La coordinación sensorial necesaria para poder manipular objetos.



**Figura 3.1:** Manipulador móvil Manfred.

Cualquier sistema robótico está formado por una serie de subsistemas que posibilitan, mediante su interconexión, el cumplimiento de los objetivos para los cuales ha sido diseñado el robot. Estos módulos utilizan la información del entorno de trabajo para generar datos que son utilizados al final para dotar de capacidad de movimiento, tanto al brazo manipulador como a la base móvil. A continuación se describen con más detalle los principales componentes de los sistemas que constituyen el manipulador móvil propuesto, que son:

1. Estructura del robot.
2. Sistema sensorial.
3. Sistema locomotor.
4. Sistema de alimentación.
5. Brazo manipulador LWR-UC3M-1.
6. Sistema de procesamiento.

### 3.1.1. Estructura del robot.

Manfred está formado por una estructura metálica que permite integrar todos los componentes que necesita para su funcionamiento y que se puede dividir principalmente en tres partes:

- Una base móvil.

La base está constituida por dos plataformas de acero con un diámetro de unos 61 cm y una altura en torno a 65 cm. Está equipada con ruedas que le permiten desplazarse. Además en la base se aloja el sistema de baterías que generan la alimentación necesaria para que el robot pueda funcionar de manera autónoma.

- Un torso fijo.

Sobre la base móvil se ha montado una estructura (bastidor) que hace de cuerpo del robot y que aloja: todo el cableado para la conexión del brazo manipulador con el ordenador, el cableado para la transmisión de potencia desde la base a los motores del brazo, los servoamplificadores asociados con los motores del manipulador y todo el cableado correspondiente a los sensores externos. Esta estructura sirve también como soporte para

el anclaje del brazo, el sensor láser y las cámaras para el sistema de visión. También se sitúa en el bastidor el PC de abordo encargado de proporcionar la inteligencia al robot, que lleva instalada una tarjeta controladora de 8 motores PMAC2-PCI que permite controlar conjuntamente los grados de libertad correspondientes a la base y al brazo manipulador.



**Figura 3.2:** Vista lateral de Manfred.

- Un brazo manipulador ligero.

El brazo manipulador LWR-UC3M-1 constituye el elemento fundamental del robot manipulador Manfred. Está compuesto por elementos rígidos conectados por medio de articulaciones de revolución que le permiten alcanzar un total de 6 g.d.l. Está diseñado para dotar a Manfred de flexibilidad para realizar tareas de agarre y desplazamiento de objetos, combinando los diversos grados de libertad de los que dispone (Ansari, 2011).

### 3.1.2. Sistema sensorial.

El sistema sensorial permite transformar las variables físicas que caracterizan el entorno en un conjunto de datos que serán procesados por otros módulos como por ejemplo los encargados de la localización, del sistema de seguridad, el planificador de movimiento, etc., dando resultado a acciones control que gobernan el comportamiento del robot. Esta información será suministrada por el sistema sensorial del robot, que consta de los siguientes elementos:

1. Subsistema de telemetría láser.

Su objetivo es proporcionar al robot la información necesaria sobre la distancia a los objetos del entorno, para realizar un modelado del espacio de trabajo que rodea al manipulador móvil. Esta información se utiliza principalmente en las tareas de navegación y localización del robot durante sus desplazamientos en un espacio de trabajo amplio. Se pueden utilizar datos de telemetría 2D o 3D según la complejidad y el grado de ocupación del espacio de trabajo. Este subsistema consta a su vez de los siguientes elementos:

- a) Telémetro láser Hokuyo UTM-30LX de 270° de apertura, situado en la parte posterior del vehículo. Tiene un rango de distancia de detección de 100mm hasta 30.000mm y un período de escaneo mínimo de 25msec y cuenta con una resolución de 0.25°. Se conecta al ordenador mediante una interfaz USB2.0. Su consumo de potencia es 12V y 700mA lo que le hace apto para ser utilizado en sistemas alimentados con baterías como es nuestro caso.



**Figura 3.3:** Telémetro láser Hokuyo UTM-30LX.

- b) Telémetro láser bidimensional PLS de Sick de  $180^{\circ}$  de apertura que permite la adquisición de datos en 2D. Para lograr la portabilidad a tres dimensiones está instalado sobre un chasis motorizado que permite su movimiento vertical dentro de un rango de  $45^{\circ}$ . Durante la fase de navegación en entornos poco ocupados, como pueden ser pasillos, se utiliza telemetría 2D de forma que el escáner realice un barrido en un plano paralelo al suelo. Las características técnicas de este láser se resumen en la Tabla 3.1.

Rango máximo	80 metros
Resolución angular	0.25/0.5/1 ° (seleccionable)
Tiempo de respuesta	26 ms
Resolución de la medida	10 mm
Tasa de transferencia	500 kbaud
Alimentación	24 V / 6 A

**Tabla 3.1:** Características técnicas del láser Sick.

Este sensor está preparado para capturar 361 medidas en un barrido horizontal de  $180^{\circ}$  (se toman medidas cada  $0.5^{\circ}$ ). El PLS garantiza unos errores en la medida de profundidad que no superan los 20 mm.

Este error se ve influenciado por dos parámetros, la distancia a medir y el ángulo de disparo del haz láser (de  $0^\circ$  a  $180^\circ$ ).



**Figura 3.4:** Telémetro láser bidimensional PLS de Sick.

## 2. Subsistema de visión.

Se pretende que el robot sea capaz de manipular objetos situados en un entorno 3D por lo que es necesario reconocer el objeto a manipular, determinar su posición y orientación relativa con respecto al manipulador móvil, así como determinar el punto y orientación adecuado para su manipulación. Las tareas de manipulación previstas consisten en la apertura de puertas y paso a través de las mismas, pulsación de interruptores o coger objetos simples de una mesa. Para poder llevar a cabo estas tareas el subsistema de visión consta de los siguientes elementos:

- a) Cámaras de color. El sistema incorpora una cámara Sony EVI-D100 para reconocer los objetos y estimar su posición respecto al robot. Esta cámara se sitúa en la parte frontal del cuerpo del manipulador móvil. El robot también incorpora una minicámara Sony B/N XC-ES50CE,

Figura 3.5, situada sobre la muñeca del manipulador. Esta cámara se utiliza en las tareas de manipulación cuando el brazo se encuentra situado delante del objeto a manipular de forma que interfiere en el campo de visión de la cámara Sony.



**Figura 3.5:** Cámaras de color de Manfred.

- b) Cámara 3D. El robot también incorpora una cámara de tecnología *time-of-flight* que le permite obtener una imagen 3D que consiste en una matriz de distancias a los distintos objetos que se encuentran delante. La información de la cámara 3D y la cámara de color convencional se fusiona para poder realizar una segmentación de objetos más precisa, facilitando así la manipulación de éstos.



**Figura 3.6:** Cámara time-of-flight.

### 3. Sensor fuerza/par.

Para la interacción con el entorno en las tareas de manipulación el robot Manfred porta en el extremo del brazo un sensor fuerza/par JR3 modelo 67M25A-U560, Figura 3.7. El sensor se sitúa entre el extremo del brazo y la pinza o elemento terminal. Este dispositivo tiene como características relevantes una capacidad de carga de hasta 11Kg, un peso de 175g, una frecuencia máxima de funcionamiento de 8KHz (JR3, 1994) y proporciona medidas de fuerza y par en 3 ejes que se pueden utilizar en el lazo de control de fuerza del manipulador móvil. El sensor se basa en un sistema de galgas extensiométricas y un sistema de adquisición mediante DSP (*Digital Signal Processor*) que permite obtener medidas de un elevado ancho de banda, con una relación señal a ruido elevada. El sistema de adquisición de datos se encuentra en una tarjeta bus PCI y soportable bajo los sistemas operativos más comunes como pueden ser Linux o Windows. El objeto principal de la utilización de este sensor es poder realizar tareas de manipulación basadas en control de fuerza o par, como por ejemplo apertura de puertas, pulsación de interruptores, recogida de objetos, etc.



**Figura 3.7:** Sensor de fuerza/par JR3.

### 4. Sensores cinemáticos.

La principal función de los sensores cinemáticos es aportar información sobre la posición, pudiendo obtener en el sistema de control las variables

derivadas (velocidad y aceleración). Estos sensores se encuentran principalmente acoplados a los ejes de los motores proporcionando información del giro de éstos en forma de cuentas de encoder. Esta información puede ser interpretada de dos formas, se obtiene la posición relativa o absoluta del sistema acoplado a dicho motor así como sus variables derivadas. En el caso de los motores del brazo, para complementar a los sensores cinemáticas se incorporan uno sensores de detección de *home* que permiten a través de una rutina de arranque inicial establecer las posiciones absolutas de cada articulación del brazo. Los sensores cinemáticos utilizados son encoders ópticos de alta resolución de la casa HP con referencia HEDS550 cuyas características principales se detallan en la Tabla 3.2. En cuanto a los sensores de *home* se han utilizado sensores inductivos de 3 milímetros de diámetro y una distancia de detección de 1 milímetro, alojados en cada articulación. El principio básico de estos sensores inductivos se basa en la detección de materiales ferromagnéticos mediante la variación de flujo que provoca su presencia cerca del área de detección del sensor.

Resolución	1024 cnt/rev
Alimentación	5 V
Marca de índice	Si

**Tabla 3.2:** Características técnicas de los encoders del brazo.

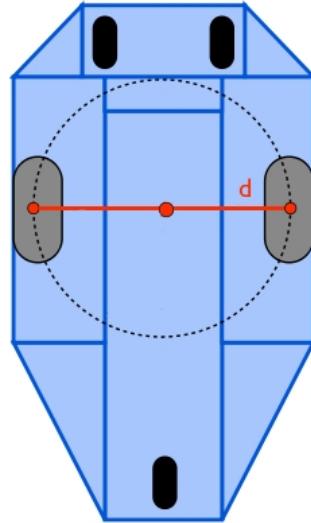
### 3.1.3. Sistema locomotor.

El sistema locomotor del manipulador móvil está constituido por la base móvil citada anteriormente, a la que se incorporan cinco ruedas: tres ruedas de apoyo que mejoran la estabilidad del robot y facilitan su movimiento, y dos ruedas motrices asociadas con motores sin escobillas y sus correspondientes servoamplificadores. La ruedas motrices dan lugar a un desplazamiento de tipo

diferencial que permite al robot girar sobre si mismo. La cinemática directa e inversa de su movimiento se expresa en las siguientes expresiones:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1/2 & -1/2 \\ 1/d & 1/d \end{bmatrix} \cdot \begin{bmatrix} v_r \\ v_l \end{bmatrix} \quad (3.1)$$

donde  $v$  y  $\omega$  son las velocidades lineal y angular de la base respectivamente,  $v_r$  y  $v_l$  las velocidades de las ruedas derecha e izquierda (r: right, l: left), y  $d$  la distancia entre los ejes de ambas ruedas ( $d = 54$  cm).



**Figura 3.8:** Base del manipulador móvil.

### 3.1.4. Sistema de alimentación.

La base móvil alberga el sistema de alimentación compuesto por baterías que dotan de autonomía a todo el robot. Consiste una conexión serie de 4 baterías de 12 V para proporcionar una tensión de alimentación de 48V en continua. Las

baterías seleccionadas son el modelo LC-X1242P de Panasonic, Figura 3.9, que proporcionan una tensión de salida de 12V y una capacidad de 42 A/h. Además, como sistema de seguridad, lleva instalado un sistema de monitorización de las baterías a través de un micro-controlador PIC16F818, que permite medir en todo instante la tensión proporcionada por las baterías y la corriente que circula por ellas. Este sistema permite comunicar continuamente el estado de la alimentación al ordenador de control, así como realizar una parada automática controlada de los motores del robot en caso de baja tensión de alimentación o de superar una corriente umbral.



**Figura 3.9:** Sistema de alimentación de Manfred.

### **3.1.5. Sistema manipulador LWR-UC3M-1.**

Con objeto de realizar las tareas de manipulación, Manfred está equipado con un brazo robótico ligero de seis grados de libertad (Figura 3.10), el cual ha sido íntegramente diseñado en la Universidad Carlos III de Madrid. Las características principales de este brazo manipulador son:

1. Redundancia cinemática similar al brazo humano.
2. Masa del conjunto de 18 kilogramos.
3. Capacidad de carga máxima de 4,5 kilogramos en el extremo del brazo.
4. Relación carga/peso entre 1:3 y 1:4.
5. Alcance en torno a 955 milímetros.



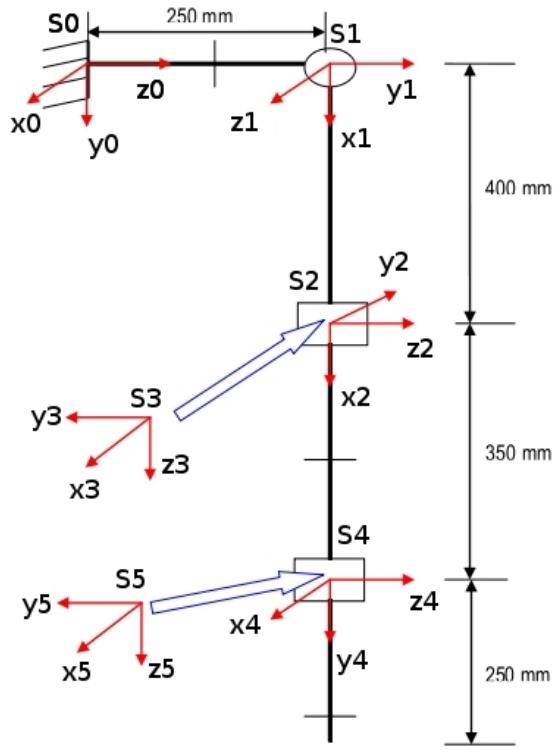
**Figura 3.10:** Brazo de Manfred.

Dicho brazo se monta lateralmente de forma que el sistema de visión y la telemetría láser 3D puedan percibir lo que se desea manipular sin ser obstaculizados por el brazo. Las articulaciones del brazo están compuestas por motores sin escobillas *brushless* de corriente continua que permiten alimentar directamente en continua a los accionadores, una etapa de reducción de velocidad y aumento de par de tipo Harmonic Drive, más concretamente de la compañía alemana Harmonic Drive AG en su categoría HFUC-2UH que incorporan rodamientos de salida integrados.

Debido a que los encoders instalados en los motores del brazo son de tipo relativo, que nos facilitan información sobre la posición actual del motor con respecto a una posición inicial o *home*, se requiere que al encender el robot se ejecute una función conocida como *homing* que nos permite establecer cual es la posición de partida del brazo y nos facilitará la conversión de la información de posición de los motores de relativa a absoluta. Esta función ha sido diseñada utilizando el lenguaje de programación propio de la tarjeta de control PMAC PCI y establece la posición inicial del manipulador aquella en la que todos sus eslabones están estirados apuntando hacia el suelo. Esta posición ha sido elegida debido a que requiere un bajo consumo de energía ya que la mayoría de los motores no están realizando ningún trabajo.

### **3.1.5.1. Parámetros de D-H.**

Si aplicamos las primeras ocho reglas expuestas en el capítulo segundo para lo obtención de los parámetros de D-H al manipulador LWR-UC3M-1, tendremos que los sistemas de referencia de las distintas articulaciones quedan como se muestra en la figura 3.11.



**Figura 3.11:** Sistemas de referencia según algoritmo D-H.

A partir de este esquema, aplicando las últimas siete reglas del algoritmo, se obtienen los parámetros de Denavit-Hartenberg que se recogen en la siguiente tabla 3.4, que nos servirán para la representación de Manfred tanto en el simulador proporcionado por la toolbox de Peter Cork de Matlab, como para la definición del manipulador utilizando la librería de cinemática de KDL.

Articulación	$\alpha_i$ (rad)	$a_i$ (mm)	$\theta_i$ (rad)	$d_i$ (mm)	Tipo
1	$\pi/2$	0	0	250	Revolución
2	$-\pi/2$	400	0	0	Revolución
3	$-\pi/2$	0	0	0	Revolución
4	$\pi/2$	0	0	350	Revolución
5	$-\pi/2$	0	0	0	Revolución
6	0	0	0	250	Revolución

**Tabla 3.3:** Parámetros D-H del manipulador de Manfred.

### 3.1.6. Sistema de control.

Manfred tiene 8 motores que le permiten mover tanto su base (2 motores) como su brazo robótico (6 motores). Para que estos motores se muevan correctamente, es necesario que se realice un control continuo sobre ellos. En este caso el control lo lleva a cabo la tarjeta controladora PMAC2-PCI.

#### 3.1.6.1. Tarjeta controladora PMAC2-PCI.

La tarjeta controladora de múltiples ejes PMAC2-PCI (*Programmable Multi-Axis Controller*) de la compañía Delta Tau Data Systems, Inc. es un dispositivo de alto rendimiento capaz de comandar hasta 8 ejes simultáneamente con un alto nivel de precisión (Delta Tau Data Systems, 2003). Gracias a sus más de 1000 variables de configuración y a la capacidad de cómputo de sus procesadores digitales de señales actuales, la tarjeta PMAC2-PCI puede ofrecer una relación precio-rendimiento muy satisfactoria. El DSP que incorpora la tarjeta PMAC2-PCI es el modelo DSP56002 de 24 bits y frecuencia de trabajo de 40 MHz de la firma Motorola.

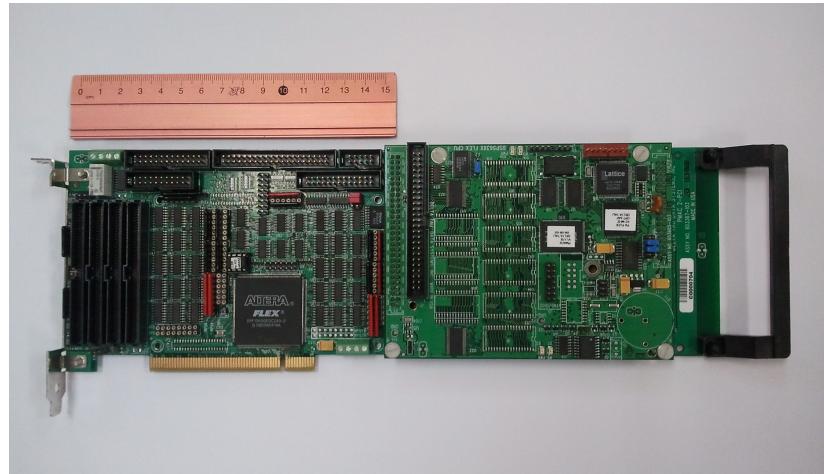


Figura 3.12: Tarjeta controladora PMAC2-PCI.

La PMAC2-PCI dispone de múltiples modos de controlar los motores, sin embargo, no está diseñada para ser conectada directamente a los drivers que alimentan a los motores y a los encoders que se usan en los motores para averiguar en qué posición se hayan estos. Dependiendo de la forma en que se quiera comandar a los motores se deberá elegir un conjunto de tarjetas adicionales que hagan de interfaz entre la PMAC2-PCI y estos dispositivos. Estas tarjetas adicionales son ofrecidas también por Delta Tau Data Systems, Inc.

En el caso del robot Manfred el modelo de tarjeta interfaz adicional que se usa se llama accesorio 8E. Se necesitan 4 de estas tarjetas para comandar los 8 motores de Manfred, ya que cada una de ellas tiene la lógica necesaria para interactuar con dos de ellos. Cada accesorio 8E se conecta a la PMAC2-PCI a través de un bus plano de 100 pines denominado JMACH. Cada accesorio 8E dispone de cuatro conversores de 18-bit D/A para comandar dos drivers de entrada analógica y debe alimentarse con una tensión de  $\pm 15V$ . El accesorio 8E dispone también de entradas para la lectura de 2 encoder y 5 entradas por eje para poder capturar diferentes tipos de eventos como:

- Señal de error.
- Señal de *home*. En el robot Manfred esta señal es indispensable para los motores del brazo, ya que al usar encoder incrementales es necesario tener un dispositivo (un sensor inductivo en este caso) que avise cuando pasa el motor por un punto predeterminado. De esta forma, con esta señal se puede programar a la controladora para que lleve los 6 motores del brazo a una posición de inicio.
- Dos señales para los límites del motor.
- Una señal cualquiera a definir por el usuario. Esta señal permite capturar un evento externo en caso de que fuera necesario para alguna aplicación.



**Figura 3.13:** Accesorio 8E para la interconexión con entre la PMAC y los drivers de potencia.

Entre los accesorios y cada motor, manfred tiene instalados 8 drivers que corresponden al modelo BE25A20 de la firma Advanced Motion Controls. Este driver es el encargado de conmutar las 3 fases del motor asociado para conseguir una velocidad de giro de su eje proporcional al nivel de tensión recibido del accesorio 8E. Para realizar adecuadamente la conmutación de las fases del motor se emplean sensores de efecto hall. El driver también es el encargado de aportar la alimentación necesaria al sensor inductivo y el encoder usados en cada motor.

La configuración de la tarjeta PMAC2-PCI es una tarea muy laboriosa, y para ello se dispone de dos manuales, el *software reference manual* y el *PMAC2 user manual*, junto con un programa que proporciona el fabricante, llamado PEWIN32 PRO, que se ejecuta bajo sistema operativo Windows (XP, Vista o 7). Este software ofrece un conjunto de herramientas que permiten modificar todos los parámetros de configuración de la PMAC2-PCI de una forma eficaz. Algunas de estas herramientas son:

1. El terminal: mediante esta herramienta se le pueden enviar comandos en codificación ASCII a la tarjeta controladora para que los execute.
2. *Watch window*: se trata de una ventana donde puede visualizar en tiempo real los valores de aquellas variables de la tarjeta controladora que se hayan elegido.
3. *Tunning Pro*: esta herramienta permite configurar parámetros de la PMAC2-PCI tales como controladores PID, filtros, calibración de los DAC, etc.
4. Ventana de posición: permite visualizar el valor en cuentas van tomando los motores, al igual que su velocidad, también en cuentas, y su error de seguimiento.

Las variables de configuración de la tarjeta controladora PMAC2-PCI se denominan I-variables. En total hay 1025 variables que configuran hasta el más mínimo detalle de la tarjeta controladora. El software PEWIN32 PRO será de gran ayuda a la hora de realizar los cambios pertinentes en la configuración de la tarjeta controladora, pero a la hora de asignar valores a las I-variables, no queda más remedio que conocer la funcionalidad de cada una de ellas. La principal

dificultad con la que se encuentra un usuario de esta tarjeta controladora es que muchas de las I-variables se encuentran relacionadas, de forma que la variación de una de ellas modifica el efecto otra/s.

La tarjeta controladora PMAC2-PCI puede ejecutar distintos tipos de movimiento, que permiten hacer distintos tipos de control en la trayectoria a ejecutar (Delta Tau Data Systems, 2004), que son comentados a continuación:

- *Motion Programs*: la tarea más obvia que se espera de una tarjeta controladora es que mueva los motores según una determinada secuencia de órdenes. Un programa de movimiento es ejecutado línea a línea por la tarjeta controladora, y una vez finalizado, no volverá a ser ejecutado hasta que la tarjeta controladora reciba un comando que así lo indique. Desde un programa de movimiento, es posible realizar llamadas a otros programas, ejecutar bucles de control e incluso realizar llamadas al terminal para ejecutar comandos de control. La tarjeta controladora PMAC2-PCI tiene capacidad para almacenar y ejecutar hasta 256 programas de movimiento.
- *Programmable Logic Controller (PLC)*: los programas PLC nacen de la necesidad de tener programas que se ejecuten de forma continua, es decir, ejecutando sus tareas de forma asíncrona a los movimientos realizados por los motores. Estos programas se escriben de forma análoga a los programas de movimiento, exceptuando que en la definición de su encabezamiento se definen como PLC. Una vez entran en ejecución, en cada ciclo de la tarjeta controladora, se ejecutará parte de dichos programas según la capacidad disponible por ésta.
- Comandos de movimiento: existe la posibilidad de enviar a la PMAC2-PCI comandos de movimiento a través del terminal. Son comandos sencillos

que permiten mover cada motor por separado, y a pesar de estar pensados para realizar pruebas en los motores, pueden ser utilizados para ejecutar movimientos sencillos dentro de un programa de control.

### 3.1.7. Sistema de procesamiento.

Un manipulador móvil es un sistema de enorme complejidad que debe ser capaz de procesar e interpretar una gran cantidad de información sensorial y cerrar varios lazos de control simultáneamente. Esta complejidad se traduce en crecimiento del software que se desarrolla. Una de las características principales sobre la que se ha estructurado la arquitectura software del sistema es la modularidad, de manera que se favorece el desarrollo y el mantenimiento del sistema software del robot. Para ello, se han desarrollado módulos funcionales que puedan trabajar conjuntamente mediante el intercambio de datos pero que se ejecuten como procesos completamente independientes. Con esta estructura modular se facilita el desarrollo, comprobación y actualización de los diferentes algoritmos. Cada uno de los módulos funcionales se diseña con capacidad para el auto-diagnóstico, de esta forma se dispone de un primer control de fallos en cada uno de los módulos.

La figura 3.14 muestra un esquema de la arquitectura de control diseñada para el manipulador robótico Manfred (Ansari, 2011). Al tratarse de módulos de ejecución independientes, cada tarea o cada fase de una tarea definirá el conjunto de módulos necesarios para su desarrollo. El computador asignado al sistema de control coordinado se encarga de las funciones básicas de control del sistema y en lleva incorporado una tarjeta controladora de ocho ejes con la que se comunica vía memoria de doble puerto. Dicha tarjeta permite cambiar dinámicamente los parámetros y el programa de control que se ejecuta para disponer de un sistema flexible que implemente técnicas avanzadas de control. Esta tarjeta realiza el control de la base (2 g.d.l.) y del brazo (6 g.d.l.).

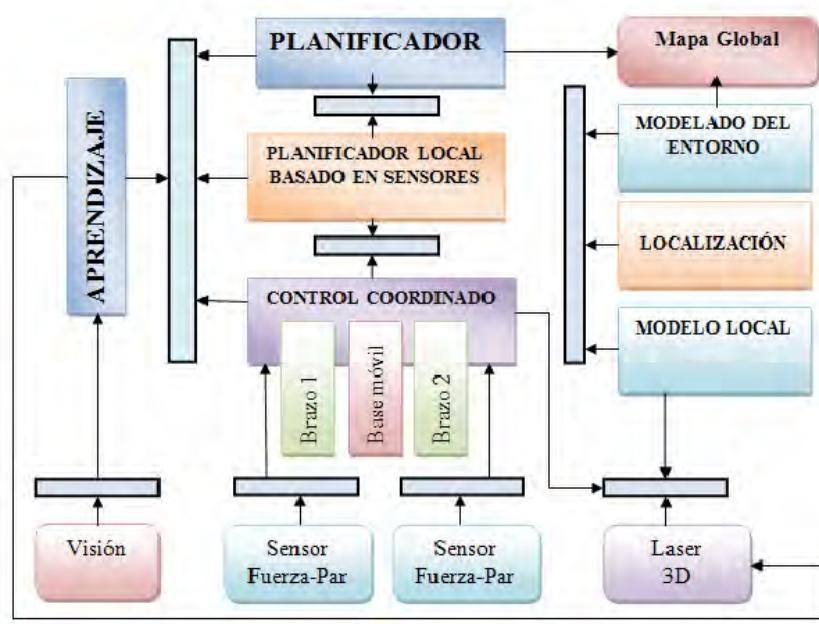


Figura 3.14: Arquitectura del sistema software de Manfred.

### 3.2. Linux.

El entorno elegido para el desarrollo de la tesis ha sido el Linux por tratarse de un sistema operativo de gran estabilidad y compatible con todas las aplicaciones necesarias para su elaboración. Además es el sistema operativo instalado en el robot.

En concreto se ha optado por Ubuntu, que se trata de una distribución GNU/-Linux, cuya licencia es libre y de código abierto. Otras ventajas de este sistema operativo son:

1. Multitarea, multiplataforma, multiusuario y multiprocesador.
2. Carga selectiva de programas según la necesidad.

3. Protección de la memoria para hacerlo más estable frente a caídas del sistema.
4. Uso de bibliotecas enlazadas tanto estáticamente como dinámicamente.

### 3.3. Matlab.

Matlab (abreviación de *MATrix LABoratory*) es un entorno de computación y desarrollo de aplicaciones totalmente integrado, orientado para llevar a cabo proyectos en los que se encuentren implicados elevados cálculos matemáticos y la visualización gráfica de los mismos.

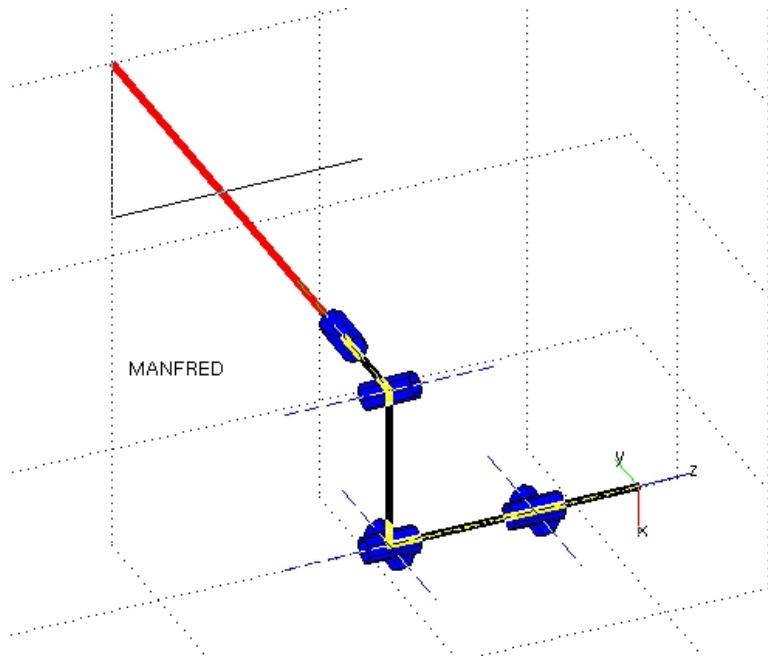
Matlab integra análisis numérico, cálculo matricial, procesado de señal y visualización gráfica, en un entorno completo donde los problemas y sus soluciones son expresados del mismo modo en que se escribirían normalmente, sin necesidad de hacer uso de la programación tradicional.

Matlab dispone también en la actualidad de un amplio abanico de programas de apoyo especializados, denominados *toolbox*, que extienden significativamente el número de funciones incorporadas en el programa principal. Estos *toolbox* cubren prácticamente casi todas las áreas principales en el mundo de la ingeniería y la simulación, señal, control robusto, estadística, análisis financiero, matemáticas simbólicas, redes neuronales, lógica difusa, identificación de sistemas, simulación de sistemas dinámicos, etc.

Para este proyecto se ha utilizado la *toolbox* de robótica de Peter Cork. Esta librería proporciona funciones útiles en diferentes áreas del campo de la robótica como pueden ser la cinemática, la dinámica o la generación de trayectorias (Corke, 2001). Se puede utilizar tanto para simulación como para el análisis de

datos resultantes de pruebas con robots reales. Se basa en un método general de representación de la cinemática y dinámica de cadenas de eslabones y articulaciones, cuyos parámetros están encapsulados en objetos de Matlab. Además ofrece funciones para la manipulación de tipos de datos como vectores, matrices de transformación o cuaternios que serán necesarios para la representación de la posición y la orientación de estas cadenas.

Con los parámetro de D-H recogidos en la tabla 3.3 y utilizando esta *toolbox*, podemos dibujar el manipulador. En la figura 3.15 se puede observar el brazo robótico con todas sus articulaciones en la posición inicial según la definición de D-H. Esta herramienta se utiliza para la simulación de las trayectorias introduciendo los distintos valores de posición de cada una de las articulaciones, de manera que observamos la posición cartesiana que corresponde a dicha configuración articular.



**Figura 3.15:** Manipulador LWR-UC3M-1 simulado con Matlab.

### 3.4. ROS.

ROS es un meta-sistema operativo para robots de código abierto desarrollado por la empresa Willow Garage. Proporciona los servicios típicos de cualquier sistema operativo como: abstracción del nivel hardware, control de bajo nivel de dispositivos, envío de mensajes entre procesos o manejo de paquetes, y además aporta herramientas y librerías para escribir, compilar y ejecutar código en distintos ordenadores. Es similar en algunos de estos aspectos a otros entornos de trabajo como Player, YARP, Oros, Carmen o Microsoft Robotics Studio (Garage, 2011).

El esquema de trabajo de ROS es una red de procesos (nodos) *peer-to-peer* que se ejecutan de manera individual y se acoplan entre sí utilizando la infraestructura de comunicación proporcionada por ROS, pudiendo ser ésta de distintas maneras: comunicación síncrona en modo cliente-servidor a través de los servicios, comunicación asíncrona por envío continuo de datos a través de *topic* o almacenamiento de datos en servidores de parámetros. Estos procesos pueden estar agrupados en paquetes o pilas que pueden ser compartidos a través de repositorios que permiten una colaboración distribuida.

Aunque ROS no es un sistema de tiempo real, se puede integrar código de tiempo real en el entorno de ROS. Por ejemplo, el robot PR2 de Willow Garage utiliza un sistema llamado *pr2-etherCAT* que transporta mensajes de ROS hacia y desde procesos de tiempo-real. Además soporta una perfecta integración con la herramienta de tiempo real de Oros.

Las características más destacables de las ofrecidas por ROS son:

1. Está diseñado de manera que sea lo más ligero posible, y que el código que se escribe para ROS pueda ser utilizado con pocos cambios en otras plataformas. Como ejemplo tenemos que ya se encuentra integrado con OpenRAVE, Orocó y Player.
2. Es independiente del lenguaje de programación utilizado, permitiendo el uso de la mayoría de los lenguajes más comunes en la actualidad. A día de hoy existen nodos implementados en Python, C++ o Lisp, y existen librerías experimentales en Java y Lua.
3. Tiene una unidad de trabajo orientada a las pruebas denominada *rostest* que facilita la corrección de errores.
4. Es apropiado para sistemas grandes en los que se necesita desarrollar numerosos módulos y luego ejecutarlos a la vez.

Actualmente ROS sólo funciona con plataformas basadas en Unix. Está ampliamente probado en Ubuntu (el sistema operativo de Manfred) y Mac OS, además a través de la comunidad de usuarios de ROS se da soporte a Fedora, Gentoo, Arch Linux y otras plataformas de Linux.

Los conceptos sobre los que se desarrolla ROS pueden agruparse en tres niveles: el sistema de archivos, el esquema de ejecución y el nivel de la comunidad. A continuación veremos una breve introducción de estos tres niveles.

El nivel del sistema de archivos lo componen los recursos que se pueden encontrar en el disco duro, como por ejemplo:

1. *Packages*: son la unidad principal en la que se organiza el software en ROS. Por ejemplo un paquete puede contener procesos ejecutables, librerías dependientes de ROS o archivos de configuración.
2. *Manifest*: proporcionan información sobre un *package*, incluyendo su tipo de licencia de uso y publicación, sus dependencias e información específica de compilación. También existen los *stack manifest*, cumplen la misma función que los anteriores, pero asociados a un *stack*.
3. *Stack*: conjuntos de paquetes que proporcionan una funcionalidad concreta. Además es la manera en la que el software de ROS es publicado y tienen números de versión asociados.
4. *Messages*: descripciones de mensajes en las que se define su estructura de datos.
5. *Services*: descripciones en las que se define la estructura de datos de la petición y la respuesta de un servidor de *Services* concreto de ROS.

El nivel de esquema de ejecuciones es una red *peer-to-peer* de procesos de ROS ejecutándose a la vez. Sus conceptos básicos son los siguientes:

1. *Nodes*: son las unidades básicas de procesado. Debido a que ROS pretende ser modular, el sistema de control de un robot normalmente requiere

de varios *nodes*. Se codifican utilizando una librería cliente de ROS, como roscpp o rospy.

2. *Master*: Es el *node* principal de ROS. Proporciona servicios de registro y búsqueda de nombres dentro del resto del esquema de ejecución. Sin él, el resto de *nodes* no serían capaces de encontrarse, intercambiar mensajes o solicitar servicios.
3. *Parameter server*: permite almacenar datos de manera centralizada, forma parte del *master*.
4. *Messages*: *nodes* que se comunican entre ellos intercambiando mensajes. Un mensaje es simplemente una estructura cuyos campos son tipos de datos. Estos campos pueden ser tanto tipos estándar primitivos (enteros, lógicos, en coma flotante, etc), arrays de éstos o combinaciones de datos en forma de estructuras.
5. *Topics*: el *topic* es simplemente un nombre que se usa como identificador. Los datos pertenecientes a un *topic* son encaminados por el *master* en la forma publicador/suscriptor. El nodo que envía los datos los publica en el *topic* adecuado, y todos los nodos suscritos recibirán los datos que han sido publicados de manera asíncrona. . En el mismo *topic* puede haber varios publicadores y/o suscriptores, y entre ellos no necesitan saber de su existencia, de manera que la compartición de información se produce de forma desacoplada.
6. *Services*: el modelo de comunicación cliente/servidor resulta muy flexible y se implementa a través de los servicios. Éstos se definen con un par de

estructuras de datos: petición y respuesta. Para su uso se requiere que un nodo se encargue de ofrecer el servicio (a través del *master*) bajo un nombre concreto y se ocupe de responder las peticiones que le envíe un cliente.

7. *Bags*: sirven para guardar datos que después pueden ser utilizados por algoritmos de prueba o de visualización.

El *master* se encarga de proporcionar un servicio de servidor de mensajes dentro del esquema de ejecución de ROS. Guarda los nombres de los *topics* y los registros a *services* para los demás nodos en ejecución de ROS. Éstos se comunican con el máster para recibir información sobre los registros de otros nodos y poder establecer las conexiones correctas. Además, el *master* se comunicará con los nodos cuando exista algún cambio en esta información de manera que se pueden crear nuevas conexiones de forma dinámica. En cualquier caso, los nodos se conectan unos a otros directamente, el *master* sólo actúa como servidor de información, como un servidor DNS. La conexión entre nodos se establece utilizando un protocolo de ROS llamado TCPROS que está basado en los sockets del estándar TCP/IP.

Esta arquitectura permite un modo de funcionamiento desacoplado, siendo los nombres de los *nodes*, *services* y *messages* la base sobre la que se construyen sistemas complejos y son el principal modo de diferenciación entre ellos. Todas las librerías de ROS soportan un remapeado de nombres a través de línea de comandos, lo que permite que un programa compilado pueda ser modificado en tiempo de ejecución para que opere según una topología diferente dentro del esquema de ejecución.

### 3.5. KDL.

KDL, acrónimo de *Kinematics and Dynamics Library*, es una librería gratuita en C++ que establece un marco de trabajo para el modelado y computación de cadenas cinemáticas, como puede ser un robot, modelos biomecánicos, etc. Proporciona un conjunto de clases generales para descripción de objetos geométricos con los que se construyen las cadenas cinemáticas, e incorpora otras clases de más alto nivel con las que se pueden realizar las especificaciones de movimiento: definición de trayectorias, interpolación entre puntos, etc.

En esta tesis se va a utilizar la librería para la resolución de los problemas de cinemática directa e inversa del brazo robótico de Manfred. La solución del problema cinemático directo se basa en el cálculo de la matriz de transformación que relaciona la posición del anclaje del brazo con la localización del efecto final, haciendo uso de los parámetros D-H vistos anteriormente. En el caso de la cinemática inversa, utiliza un algoritmo basado en la matriz jacobiana pseudo-inversa generalizada para realizar la transformación de velocidad del espacio cartesiano al espacio articular.

# Capítulo 4

## Solución propuesta.

De acuerdo con lo expuesto en el capítulo segundo, se ha implementado un control en el espacio operacional debido fundamentalmente a dos razones: las especificaciones de la trayectoria la tenemos en el espacio de la tarea, y además, permite introducir de manera más sencilla correcciones de la trayectoria en tiempo de ejecución, algo que es frecuente en los entornos de trabajo dinámicos para los que Manfred ha sido diseñado.

### 4.1. Controlador cartesiano.

El objetivo del controlador cartesiano consiste en ejecutar las trayectorias de la pinza, situada en el extremo del brazo, que son entregadas por el nivel de planificación. Estas trayectorias estarán especificadas en variables cartesianas, mientras que el robot debe ser controlado mediante variables articulares, ya sea en posición o en velocidad. Para realizar esta transformación, se hace uso de la jacobiana analítica. El cálculo de las velocidades de las articulaciones se realiza usando la inversa de la jacobiana y evaluando la posición de las articulaciones en el instante de tiempo anterior de la siguiente manera:

$$q(t_{k+1}) = q(t_k) + J^{-1}(q(t_k))v(t_k)\Delta t \quad (4.1)$$

Debido a que la reconstrucción de la posición de las articulaciones se realiza a través de la integración numérica de la velocidad, la posición del efecto final real no coincidirá de manera exacta con la posición deseada. Este problema puede ser resuelto utilizando un esquema de control que tenga en cuenta el error de posición y orientación final del brazo. Si consideramos este error:

$$e = x_d - x = x_d - k(q) \quad (4.2)$$

Además, podemos expresar su derivada en el tiempo de acuerdo con la cinemática diferencial de la siguiente manera:

$$\dot{e} = \dot{x}_d - \dot{x} = \dot{x}_d - J_A(q)\dot{q} \quad (4.3)$$

Cuando se formula el problema de la cinemática inversa de manera algorítmica, las variables articulares  $q$  correspondientes a la posición y orientación objetivo deben alcanzarse de manera muy precisa, por debajo de un error límite preestablecido. El tiempo de establecimiento del sistema dependerá de las características dinámicas del error de la ecuación diferencial. La elección en la relación entre  $\dot{q}$  y  $e$  permite ajustar el funcionamiento del algoritmo de la cinemática inversa.

### 4.1.1. Jacobiana inversa.

Si asumimos que la matriz jacobiana analítica del manipulador es cuadrada y no singular, la elección de la ecuación 4.4 lleva a un sistema lineal equivalente (Sciavicco y Siciliano, 2005) tal que se cumple 4.5:

$$\dot{q} = J_A^{-1}(q)(\dot{x}_d + \mathbf{K}e) \quad (4.4)$$

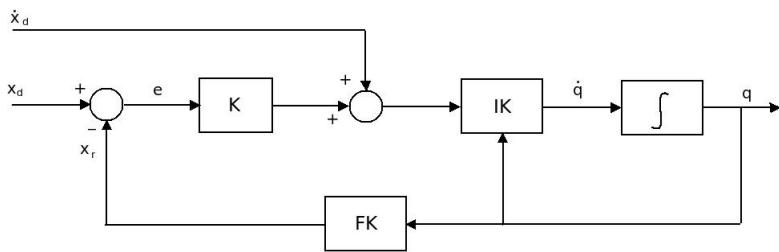
$$\dot{e} + \mathbf{K}e = 0 \quad (4.5)$$

Si  $\mathbf{K}$  es una matriz positiva (normalmente diagonal), el sistema es asintóticamente estable. Por lo tanto, el error tiende a cero a lo largo de la trayectoria, siendo su tiempo de convergencia dependiente de los valores propios de la matriz  $\mathbf{K}$ ; cuanto mayores sean éstos más rápida será la convergencia. Además, ya que el sistema está pensado para implementarse en el tiempo discreto, se sabe que los valores propios van a tener un límite superior por el debajo del cual se garantiza la estabilidad asintótica y que estará limitado por el tiempo de muestreo del sistema (Sciavicco y Siciliano, 2005).

### 4.1.2. Implementación en ROS.

La primera solución que ha sido probada es la propuesta en (Sciavicco y Siciliano, 2005), y se puede observar en la figura 4.1. Se puede apreciar un bloque de integración en el lazo directo, necesario para el cálculo de las posiciones articulares a partir de su velocidad, calculada por el bloque de cinemática inversa

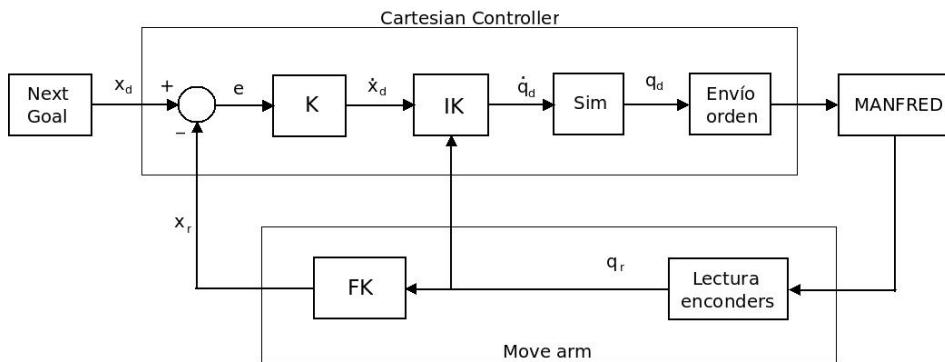
(IK). El bloque de la cinemática directa (FK), se encarga de calcular la posición cartesiana real del efecto final. Además, el esquema asegura que para una referencia de velocidad cartesiana constante ( $\dot{x}_d = 0$ ), se garantiza error nulo en régimen transitorio. La acción de realimentación provocada por  $\dot{x}_d$  para una referencia variable en el tiempo asegura que el error se hace cero a lo largo de la trayectoria, independientemente del tipo de referencia  $x_d(t)$  que se tenga.



**Figura 4.1:** Diagrama de bloques del algoritmo de cinemática inversa.

#### 4.1.2.1. Primera solución propuesta.

El esquema mostrado anteriormente ha sido implementado bajo el sistema operativo para robots ROS, usado en el manipulador móvil Manfred. El diagrama de bloques resultante podemos verlo en la figura 4.2. Los diferentes bloques de ROS pertenecientes al diagrama se explican a continuación.



**Figura 4.2:** Implementación bajo ROS del diagrama de la figura 4.1.

- Next goal: este bloque se encarga de proporcionar los nuevos puntos objetivo por los que debe pasar el efecto final. La posición será indicada en coordenadas cartesianas  $x$ ,  $y$ ,  $z$ , y la orientación según la convención  $roll$ ,  $pitch$ ,  $yaw$  (alabeo, cabeceo y guiñada). Para su implementación bajo ROS, se ha creado un *node* que proporciona un servidor de *service* que atiende a las peticiones de un cliente devolviendo la información sobre el siguiente punto a alcanzar. Además de la posición y orientación objetivo, se incluye un campo que sirve para indicar cuando se ha llegado al último punto de una trayectoria, y otro con el que se puede indicar si es necesario que el siguiente punto se alcance una orientación concreta, o si por el contrario vale con que se alcance la posición objetivo sin importar con que orientación se llega.
  
- IK (Inverse Kinematics): este bloque se encarga de realizar el cálculo de la cinemática inversa utilizando la librería de cinemática y dinámica de Orocos, KDL. El algoritmo de la cinemática inversa se basa en la jacobiana pseudo-inversa generalizada, y utiliza el cálculo de la descomposición de los valores singulares para la transformación de las velocidades de cartesianas a articulares (Leuven, 2011). Bajo ROS se ejecuta como un *node* que contiene un servidor de *service*. Cuando un cliente quiere hacer una petición, debe adjuntar en ella información sobre la posición actual de las articulaciones, y las velocidades de traslación y rotación objetivo, el servidor contestará con las velocidades articulares correspondientes.
  
- FK (Forward Kinematics): este bloque es muy similar al anterior, pero en este caso se encarga de realizar el cálculo de la cinemática directa utilizando la misma librería. Bajo ROS se ejecuta como un *node* que contiene un servidor de *service*. Cuando un cliente quiere hacer una petición, debe adjuntar

en ella información sobre la posición actual de las articulaciones y el servidor contestará con la posición y orientación actuales usando coordenadas cartesianas y los ángulos *roll*, *pitch*, *yaw*.

- sim: este bloque se encarga de hacer la integración en el tiempo de las velocidades devueltas por el bloque de cinemática inversa, de manera que se obtiene la posición que alcanzarían las articulaciones al ejecutar el movimiento calculado durante un tiempo igual al periodo de ejecución del bucle de control. Se integra en ROS como una función a la que llamará el *node* que ejecuta el bloque *cartesian controller*.
- cartesian controller: es el bloque principal del controlador. Se ocupa de pedir una nueva posición objetivo al servidor *Next goal*, y comparándola con la posición actual del robot, calcula el error en posición y orientación, para después obtener las velocidades de traslación y rotación objetivo. Con esta información se ejecutan, de manera secuencial, los bloques *IK* y *sim*, obteniendo la posición que debe alcanzar cada articulación del brazo por medio de la integración de las velocidades de las articulaciones. Estas posiciones deben ser enviadas a la tarjeta de control del robot para que se encargue de la ejecución de los movimientos correspondientes. Este bloque se ejecuta en un *node* como un cliente de *action* que se comunica con el bloque *move arm*, que hace de servidor, al que enviará la posición objetivo de cada una de las articulaciones para que se encargue de controlar cuando se han alcanzado. Además ejecuta clientes de *service* para comunicarse con los nodos *IK*, *sim* y *FK*.
- move arm: este bloque se ejecuta en un *node* que crea un servidor de *action*. Este servidor utiliza un paquete de ROS, *actionlib*, que proporciona una

interfaz estándar para la ejecución de acciones que pueden ser reemplazadas o canceladas desde el cliente mientras el servidor se está ejecutando. Su modo de funcionamiento es parecido a un servidor de *services*, pero mientras que en éstos el cliente debe esperar su respuesta para proseguir (funcionan de manera bloqueante), en el caso de las acciones el cliente proseguirá con sus funciones mientras el servidor se encarga de las acciones que le correspondan, y los objetivos iniciales enviados por el cliente pueden ser modificados o cancelados en cualquier momento. Esta diferencia ofrece ventajas importantes en tareas que pueden tardar mucho tiempo en ejecutarse, o cuyos objetivos pueden cambiar en tiempo de ejecución. En el sistema propuesto, el servidor de *action* se encarga comprobar periódicamente si la posición actual de los motores del brazo coincide con su objetivo, en coordenadas cartesianas. En caso de no haber sido alcanzadas, comunica la posición actual al cliente del bloque *cartesian controller* y continúa con el chequeo de la posición del brazo. En caso de haber sido alcanzado el objetivo, manda parar a los motores y envía la posición final real alcanzada (que no tiene por qué coincidir exactamente con la objetivo) al bloque *cartesian controller*, que se encargará de realizar las acciones necesarias para seguir con la ejecución de la trayectoria correspondiente. Es importante señalar que el límite de error aceptado en la posición final real del efector final es de 1 centímetro (en cada eje de referencia) para la posición y de 2 grados (en cada eje de giro) para la orientación. Además, el controlador ha sido diseñado de manera que se pueda establecer que para ciertos puntos de una trayectoria no sea importante la orientación con la que se alcancen, en este caso la orientación no será tenida en cuenta a la hora de comprobar si el robot ha alcanzado la posición final objetivo.

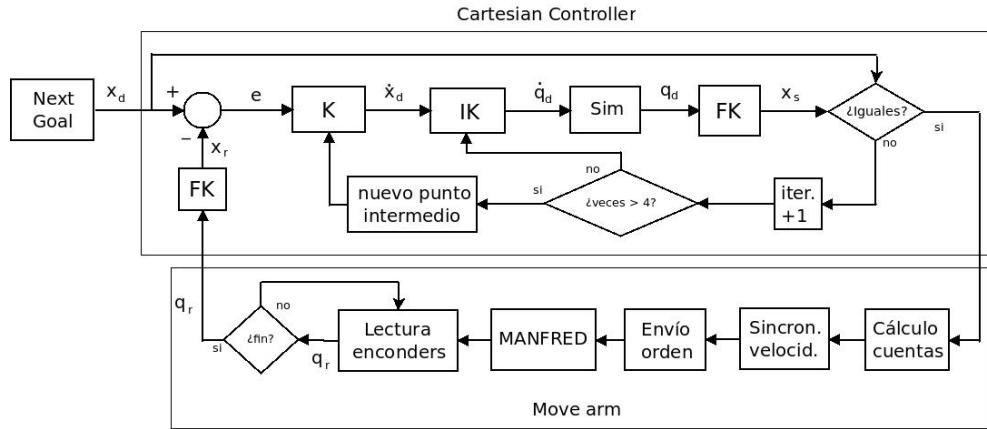
Antes de ser utilizado en el robot, el algoritmo ha sido probado mediante la *toolbox* de robótica de Matlab. Para ello se han añadido al bloque *move arm*

dos funciones: una se encarga de simular las posiciones por las que iría pasando el robot según las órdenes enviadas a éste, y la otra permite guardar en un archivo cada punto de posición del robot que es simulado por la función anterior a lo largo de la ejecución de una trayectoria completa. Una vez completada la trayectoria, se ha utilizado la interfaz de simulación de la *toolbox* de robótica de Matlab, que se mostró en el capítulo tercero, para que ejecute de manera secuencial los puntos grabados con una separación de 0,5 segundos entre cada dos puntos. De esta manera se ha podido hacer una comprobación visual de los movimientos aproximados que haría el robot con las órdenes de control calculadas con el controlador cartesiano que se ha implementado.

Una vez se ha realizado la comprobación descrita en el párrafo anterior, cuyas imágenes más importantes se pueden ver en el capítulo siguiente, se ha obtenido la siguiente conclusión: aunque el efecto final alcanza la posición y orientación final deseada y pasa por todos los puntos intermedios de la trayectoria, en la simulación se puede observar cómo entre algunas parejas de puntos objetivo, el controlador manda órdenes que dan lugar a movimientos muy bruscos del brazo robótico. Esto es debido a que cuando el brazo se encuentra en una posición singular (como la posición inicial del brazo) o cerca de ella, un movimiento pequeño en el espacio cartesiano requiere velocidades muy elevadas de los motores, provocando que el brazo se salga de la trayectoria esperada. También ocurre cuando el movimiento articular que se ha de ejecutar entre dos puntos consecutivos de la trayectoria es muy largo. Este comportamiento es producido por el bloque *IK*, más concretamente en la función de la librería de KDL que se encarga del cálculo de las velocidades angulares de los motores en función de la posición inicial y las velocidades de translación y rotación deseadas.

### 4.1.2.2. Controlador de trayectoria coordinada.

Debido a este problema, el esquema propuesto por Sciavicco no funciona. Como solución, se ha propuesto el esquema que se puede ver en la figura 4.3, en el que se han introducido algunos cambios que explicaremos a continuación.



**Figura 4.3:** Diagrama de bloques de la alternativa propuesta.

La solución adoptada consiste en introducir un nuevo bucle de control dentro del que ya había, de manera que se asegura que no se van a enviar al robot órdenes que provoquen que éste realice movimientos bruscos. Además, se ha añadido en este nuevo bucle la capacidad para detectar cuando dos puntos consecutivos de la trayectoria están demasiado separados, haciendo necesario calcular un punto intermedio para facilitar la ejecución de ésta. Por último, se ha introducido un bloque que sincroniza las velocidades de los motores, según la ecuación 4.6, de manera que entre cada dos puntos, todos los ejes comienzan y terminan su recorrido simultáneamente.

$$\omega_i = q_i / q_{qimax} * \omega_{max} \quad (4.6)$$

Debido a que la mayoría de los bloques utilizados realizan exactamente la misma función que la comentada anteriormente, a continuación se explican los cambios que han sido introducidos en los dos componentes principales del controlador: el servidor y el cliente de *action*, o lo que es lo mismo, los bloques *move arm* y *cartesian controller* respectivamente:

- *cartesian controller*: este es el bloque en el que se han introducido la mayor parte de los cambios. La secuencia de acciones que ejecuta es la siguiente: calcula la velocidad de traslación y rotación entre dos puntos consecutivos de la trayectoria, con esta información, utilizando los bloques *IK* y *sim* calcula la posición que alcanzaría cada uno de los motores del brazo del robot en el periodo de ejecución del bucle. En este punto, en vez de directamente enviar las órdenes al robot, se hace uso de la cinemática directa para comprobar que la posición dada por el controlador nos lleva hasta el punto deseado. En caso contrario, se vuelve a ejecutar esta parte del bucle de control, estableciendo como posición inicial del robot la posición simulada anteriormente. La repetición de este bucle se realizará un máximo de 4 veces. En el caso de que después de la cuarta repetición no se consiga alcanzar el punto deseado, el controlador considerará que la distancia entre el punto el punto de partida y el siguiente punto de la trayectoria es demasiado grande, por lo que se procede a calcular un punto intermedio que se establecerá como nuevo destino objetivo y se volverá al punto inicial del bucle. En el momento en que el punto simulado y el objetivo son suficientemente cercanos (esta decisión se toma en base a los mismo límites que se comentaron anteriormente) el bloque *cartesian controller* envía el punto objetivo, definido en posiciones articulares para cada motor, al bloque *move arm* y acaba su ejecución hasta que no le sea enviado un mensaje con la posición final alcanzada por el manipulador.

- *move arm*: según el nuevo esquema, este bloque resulta ahora más sencillo. Una vez ha recibido las posiciones angulares objetivo, se encarga de calcular cual es la diferencia entre la posición actual de cada motor y la posición a la que se quiere llegar (en cuentas de encoder), construye la orden que se debe enviar a la tarjeta de control para alcanzar la posición final y se la envía. Por último el bloque queda en espera hasta que todos los motores han alcanzado su objetivo, momento en el cual enviará al bloque *cartesian controller* la posición real que han alcanzado los motores para proseguir con la ejecución de la trayectoria.

Al igual que con la solución anterior, primero se ha probado el controlador guardando las posiciones por las que se manda pasar al robot de acuerdo con las instrucciones que se le envían, y ejecutándolas después en el interfaz de simulación de la *toolbox* de Matlab. En este caso se observa que los puntos que se van alcanzando consecutivamente corresponden con los que han sido calculados por el módulos planificador de trayectorias, necesitando en algunos casos del cálculo de puntos intermedios para pasar de un punto a otro de la trayectoria. La exactitud con la que se ejecuta la trayectoria será comentada en el siguiente capítulo.

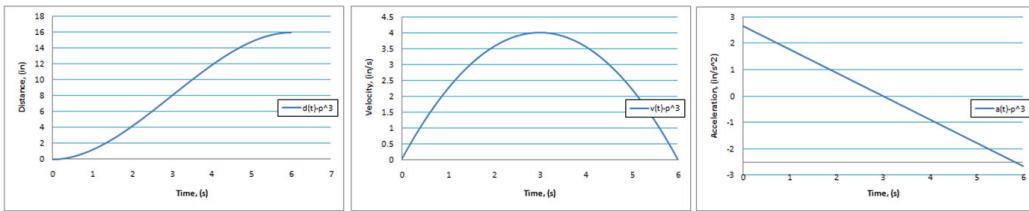
El siguiente paso consiste en probar el controlador en el robot Manfred. Para cada punto de la trayectoria articular se le envían al robot dos órdenes: una para configurar la velocidad de cada motor, de manera que éstos empiecen y acaben simultáneamente, y otra para ejecutar una orden de movimiento que corresponde con los grados de giro del motor. Entre cada punto de la trayectoria, cada motor acelera hasta la velocidad fijada para el movimiento, y al final decelera para quedar totalmente parado.

El resultado es una trayectoria ejecutada de manera coordinada, de modo que al igual que pasaba con la simulación en Matlab, el efecto final del robot

pasa por los puntos especificados en dicha trayectoria. Sin embargo, y como es de normal en este tipo de trayectorias, el movimiento realizado por el efecto final no coincide con una trayectoria cartesiana que pudiera resultar lógica a un ser humano, dando lugar en algunos casos, a movimientos que se salen de la trayectoria esperada.

#### 4.1.2.3. Controlador de trayectoria continua.

Para corregir este problema es necesario que entre punto y punto de la trayectoria, el brazo no se pare. Para esto, las órdenes de movimiento del robot van a formar parte de un *motion program* de la tarjeta controladora, que ejecutará estas órdenes en modo PVT (*position, velocity, time*). En este modo, hay que enviar a la tarjeta tres datos: posición final del motor, tiempo a invertir en el movimiento, y velocidad con la que debe finalizar ese movimiento. Con estos datos, la PMAC2-PCI calcula la única trayectoria de tercer grado, en posición, que cumple las especificaciones dadas, dando como resultado un perfil de aceleración lineal, un perfil de velocidad parabólico y un perfil de posición cúbico (Delta Tau Data Systems, 2003), como los que se observan en la figura 4.4.



**Figura 4.4:** Perfiles de posición, velocidad y aceleración respectivamente.

En esta caso, no es necesario probar la trayectoria en Matlab, ya que los puntos finales por los que pasa son los mismos que en la simulación anterior, y el cambio se producen en la trayectoria que describe el efecto final entre punto y

punto. Se ha probado este modo de control con una trayectoria horizontal y vertical para comprobar que, como esperábamos, la trayectoria cartesiana obtenida es muy parecida a la que cabía esperar.

## 4.2. Librería para el sensor JR3

Como se menciona en el segundo capítulo, la captación de información del entorno por parte de los sensores es esencial para que se puedan tomar las decisiones correctas. En el caso del sensor de JR3, éste nos proporciona medidas de fuerza y par en 3 ejes. Esta información podrá ser utilizada en distintas aplicaciones como: detectar el contacto del efecto final con un objeto, cerrar el lazo de control de fuerza para interactuar con objetos del entorno o calcular el peso de un objeto que previamente haya sido cogido con la pinza.

Para facilitar el desarrollo de futuras aplicaciones, se ha implementado una librería en C++ que permite tanto el acceso a la información que proporciona el sensor, como la configuración de las distintas opciones disponibles en el dispositivo, de manera que la información extraída sea más útil. A continuación se van a explicar las principales funcionalidades que ofrece la librería:

1. abrir y cerrar: permiten el comienzo y finalización de la comunicación entre la aplicación y el driver que nos da acceso a la memoria de la tarjeta PCI que controla el sensor.
  
2. get\_full\_scales: devuelve el valor de los *full scale* de cada uno de los ejes del sensor.

3. get\_offsets: devuelve el valor de *offset* que se aplica a cada uno de los ejes del sensor.
4. info\_sensor: devuelve la siguiente información sobre el sensor: fecha del software que tiene el DSP, número de versión de la tarjeta, número de bits del conversor, anchura del sensor y unidades de medida utilizadas.
5. jr3Filter: devuelve las medidas del sensor JR3. Al llamar a la función, es necesario decirle cuál es la medida que se desea, ya que la tarjeta PCI ofrece 7 medidas diferentes. Si elegimos *FILTER0*, nos devolverá las medidas desacopladas y con el *offset* quitado. Entre el *FILTER1* y el *FILTER6* la única diferencia es la frecuencia de corte de los filtros que se aplican a los datos. En nuestro sensor, estas frecuencias son: 500Hz, 125Hz, 31,25Hz, 7,813Hz, 1,953Hz, 0,4883Hz respectivamente (JR3, 1994).
6. read\_dir: devuelve el valor de la posición de memoria indicada en la llamada.
7. reset\_offsets: pone a cero el valor de los *offsets* que se aplican a las medidas de los sensores.
8. set\_offsets: configura el valor de los *offsets* que se aplican a las medidas de los sensores con el valor indicado por el usuario.
9. set\_RMaxFScales y set\_RmimFScales: configura el valor de *full scale* para cada eje con el valor máximo o mínimo recomendado.

10. set\_user\_fs: configura el valor de *full scale* para cada eje con el valor indicado por el usuario.
11. set\_vector\_axes: configura la tarjeta para que devuelva el valor de los vectores de fuerza y par del modo deseado.

#### 4.2.1. Ejemplo aplicación para detectar contacto.

Para mostrar la utilidad de la librería, se ha realizado una aplicación de ejemplo que funciona bajo el sistema operativo de ROS. La aplicación consiste en la detección del contacto del brazo de Manfred con objetos usando las medidas de fuerza del sensor JR3. Este contacto deberá producirse en una zona próxima al sensor para que sea posible su detección, en concreto nos interesa detectar el contacto cuando éste se produce con la pinza acoplada al brazo.

Para ello, ha sido necesario añadir al controlador de trayectorias un par de nuevos *nodes*. El primero de ellos se comporta como un publicador de información en un *topic*. Su funcionamiento es muy simple, está constantemente leyendo los datos que proporciona el sensor JR3, con una frecuencia de 100 Hz, y los publica según le van llegando. La información leída es la proporcionada por FILTER3, de manera que los datos que obtenemos están filtrados a 31,25Hz, eliminando así buena parte del ruido. El segundo nodo es el encargado de hacer la detección del contacto. Por un lado se suscribe al *topic* en el que se publican los datos medidos por el sensor JR3, de manera que cada vez que hay un dato nuevo, se ejecuta una función de *callback* que guarda el dato para su posterior procesado. Cada uno de estos datos recibidos está formado por las 3 componentes de la fuerza y el par, aunque para esta aplicación sólo nos interesan las correspondientes a la fuerza. El procedimiento por el cual se quiere detectar el contacto es muy simple, se va a buscar un cambio brusco en la fuerza medida en

cualquiera de las tres componentes, ya que en principio, no tenemos por qué saber ni la dirección, ni el lugar de la pinza con el que se producirá el contacto.

Resulta necesario determinar un umbral con el que se va a tomar la decisión de si se ha producido un contacto con un objeto. Para esto, hay que tener en cuenta que las fuerzas provocadas por un choque dependerán de la velocidad con la que se produce, y de la resistencia al movimiento del objeto involucrado. Además, en la elección del umbral, debemos tener en cuenta que éste debe ser mayor que las fuerzas provocadas en el extremo del brazo por las aceleraciones y deceleraciones de las distintas articulaciones del brazo robótico.

Por lo tanto, para facilitar esta detección, se ha decidido procesar los datos del sensor con un filtro, el filtro de mediana de cinco datos. Éste consiste en que, cada cinco datos recibidos, éstos se colocan de menor a mayor, devolviendo el filtro el dato central. De esta manera, se consigue que los picos producidos por los cambios de fuerza en el extremo del brazo se puedan detectar más fácilmente ya que se aumenta la diferencia existente entre dos puntos consecutivos.

Para introducir esta aplicación en el sistema de control del robot, se han introducido dos nuevos *nodes* al controlador de trayectoria coordinada, de manera que el diagrama de bloques queda como en la siguiente figura:

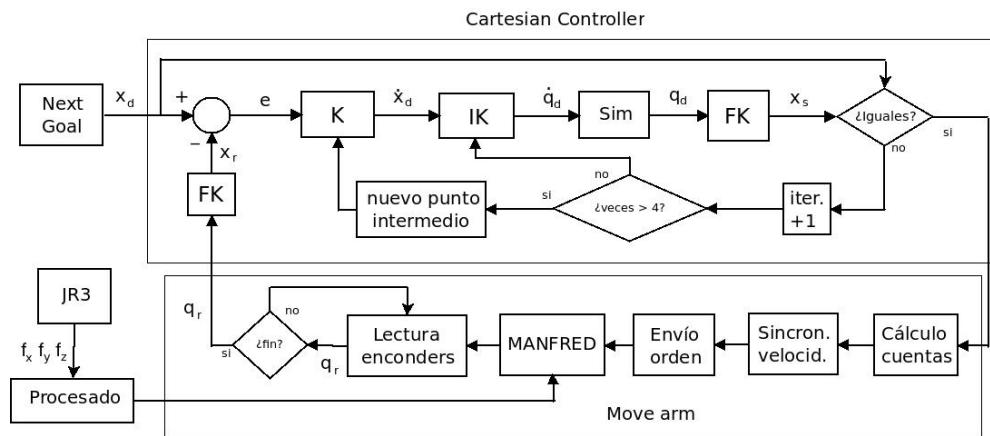


Figura 4.5: Diagrama de bloques del detector de contacto.

El *node* denominado JR3 se encarga de hacer la comunicación con el driver del sensor utilizando la librería que ha sido implementada. Hace la inicialización del sensor, y posteriormente comienza la lectura de datos y su publicación en un *topic*. El *node* denominado procesado, se suscribe al *topic* publicado por JR3, realiza el filtro de mediana de los datos recibidos, y comprueba si se ha producido contacto comparando datos consecutivos devueltos por el filtro. En caso de que sea detectado, manda parar a todos los motores del brazo de Manfred, y le comunica al *node* move arm que se ha producido un contacto a través de un *topic*. Éste detiene su ejecución inmediatamente para que no se envíen nuevas órdenes de movimiento, y a través del mensaje de realimentación proporcionado por el servidor de *action*, comunica al *node* cartesian controller que ha de abortar la trayectoria en curso, que se dará por finalizada. Debido a que no se tiene implementación para la replanificación de trayectorias, o la modificación *on-line* de la trayectoria en ejecución, el sistema completo termina su ejecución en ese momento.



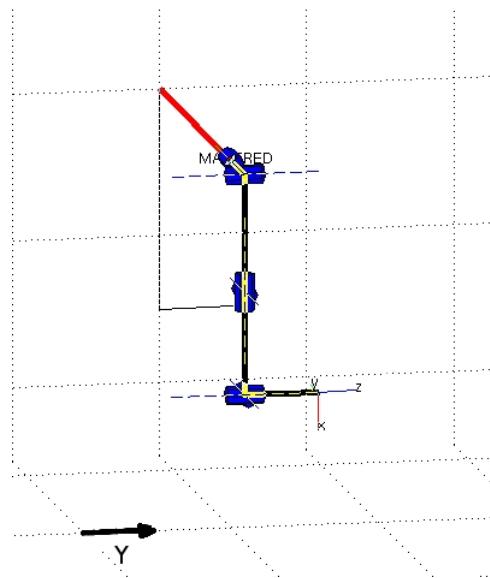
Capítulo **5**

## Resultados experimentales.

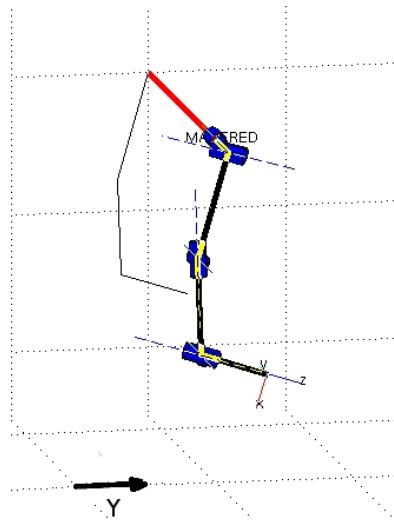
A continuación se procede a exponer los resultados que se han obtenido en las diferentes fases de la tesis, así como su análisis.

### 5.1. Primera solución propuesta.

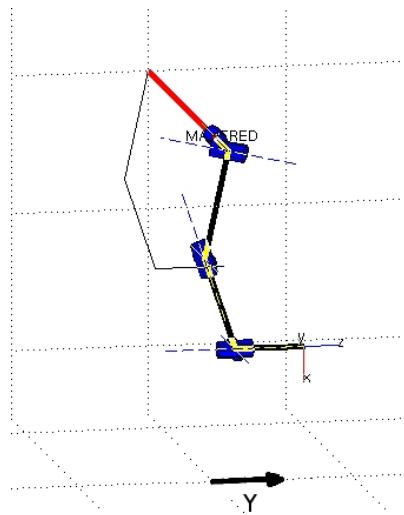
Como ya se dijo, esta solución tenía problemas en el cálculo de la cinemática inversa, haciendo que en ciertas ocasiones el robot tenga que salirse de su trayectoria original, como puede observarse en las siguientes figuras. En ellas, tenemos tres posiciones consecutivas comandadas por el controlador para una trayectoria, siendo la trayectoria original consiste en un movimiento en línea recta en el eje Y. Sin embargo, se puede observar como, entre la primera y la tercera posición, que sí cumplen con la trayectoria, el robot realiza un movimiento en el sentido contrario al esperado, y modificando su orientación.



**Figura 5.1:** Primera posición de la trayectoria,  $x = 0,75$ ,  $y = 0,25$ ,  $z = 0,25$ .



**Figura 5.2:** Posición intermedia incorrecta,  $x = 0,81$ ,  $y = 0,14$ ,  $z = 0,25$ .



**Figura 5.3:** Tercera posición de la trayectoria,  $x = 0,72$ ,  $y = 0,27$ ,  $z = 0,25$ .

Para evitar que se envíen al robot órdenes que le hagan pasar y parar por puntos que en realidad están fuera de la trayectoria, se ha añadido otro bucle de control que genera que se asegura de mandar sólo las órdenes que lleven al brazo robótico a los puntos definidos en la trayectoria.

## 5.2. Controlador de trayectoria coordinada y continua.

Los resultados de la ejecución de la trayectoria de los dos controladores se presentan en la misma sección para resaltar las diferencias existentes.

En ambos casos la trayectoria a ejecutar va a ser la misma, la definición de ésta se puede ver en la tabla 5.1, en la que los datos de posición están en centímetros, y los de orientación en radianes. Consiste en un desplazamiento vertical hacia arriba, desde el punto de comienzo, en el que no se modifica la orientación del efecto final, y prosigue con un movimiento de avance en horizontal en el que la orientación del efecto final cambia a la vez que éste se produce.

<b>X</b>	<b>Y</b>	<b>Z</b>	<b>Roll</b>	<b>Pitch</b>	<b>Yaw</b>
0,9268	0,1768	0,25	-PI/2	0	-PI/4
0,8768	0,1768	0,25	-PI/2	0	-PI/4
0,8268	0,1768	0,25	-PI/2	0	-PI/4
0,7768	0,1768	0,25	-PI/2	0	-PI/4
0,7268	0,1768	0,25	-PI/2	0	-PI/4
0,6768	0,1768	0,25	-PI/2	0	-PI/4
0,6768	0,2268	0,25	-PI/2	0	-PI/4
0,6768	0,2768	0,25	-PI/2	0	-0,635398
0,6768	0,3268	0,25	-PI/2	0	-0,485398
0,6768	0,3768	0,25	-PI/2	0	-0,335398
0,6768	0,4268	0,25	-PI/2	0	-0,185398

**Tabla 5.1:** Definición cartesiana de la trayectoria a ejecutar.

Los resultados de la ejecución se muestran en las gráficas siguientes, que muestran la evolución temporal de los ejes x e y en ambos casos. El eje z no merece la pena mostrarlo ya que se mantiene constante a lo largo de toda la trayectoria. La evolución de la orientación no se muestra ya que no ofrece ninguna información adicional que no pueda observarse en la evolución de la posición. En ambos casos los datos han sido obtenidos monitorizando la posición de cada articulación, con una frecuencia de 100 Hz, mientras ejecutaban la trayectoria descrita en el párrafo anterior.

En las primeras dos gráficas podemos observar como evoluciona la posición de los ejes x e y al ser ejecutada la trayectoria con el controlador coordinado. Podemos observar como el recorrido del efecto final entre puntos consecutivos de la trayectoria no es el esperado, que sería una línea recta.

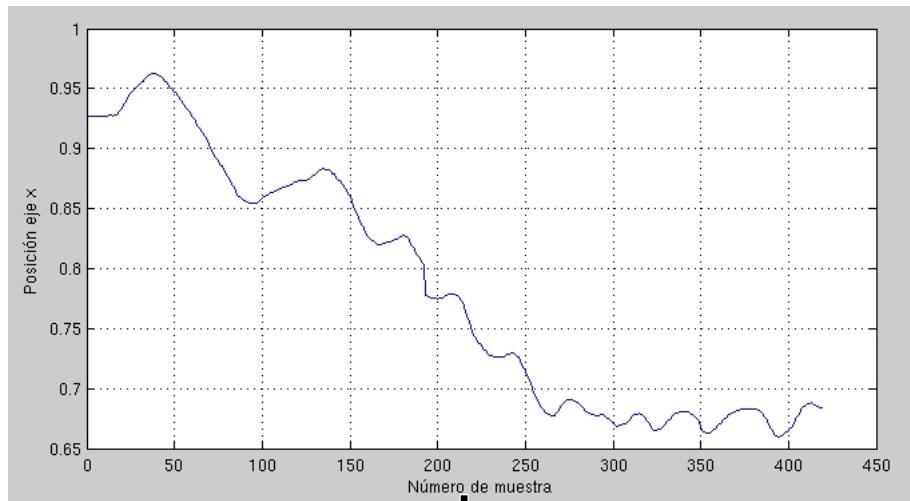


Figura 5.4: Evolución de la trayectoria coordinada en el eje x.

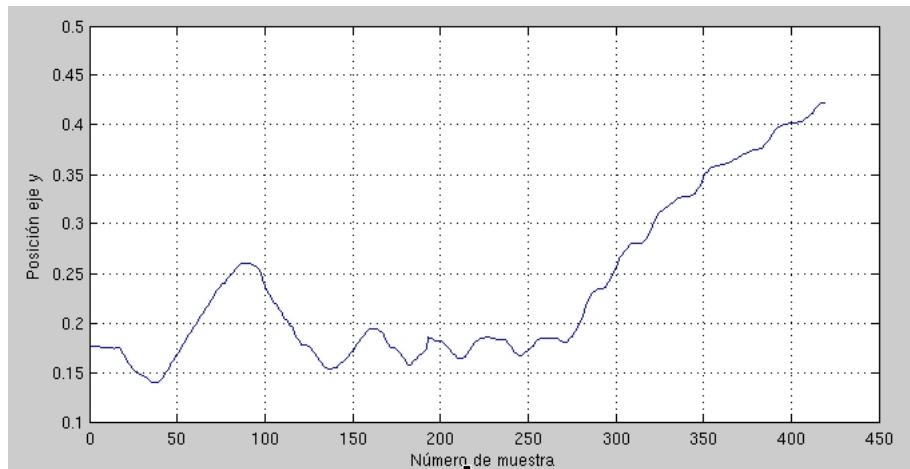
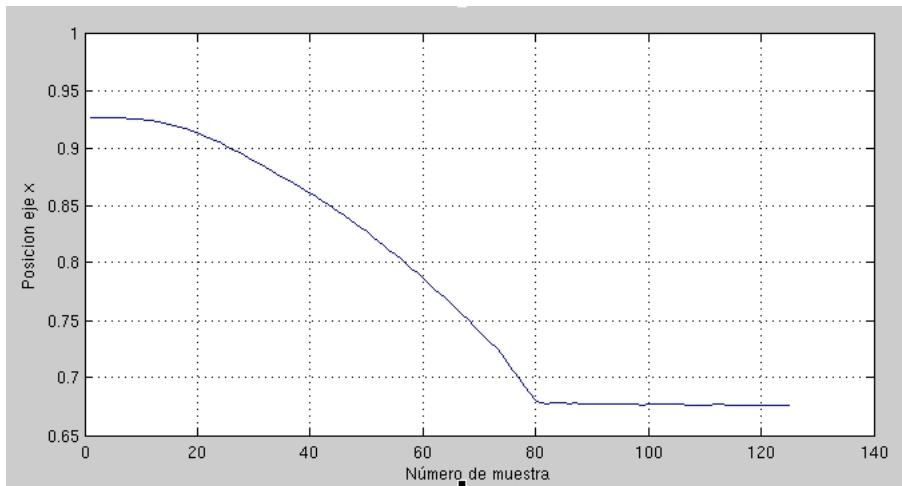
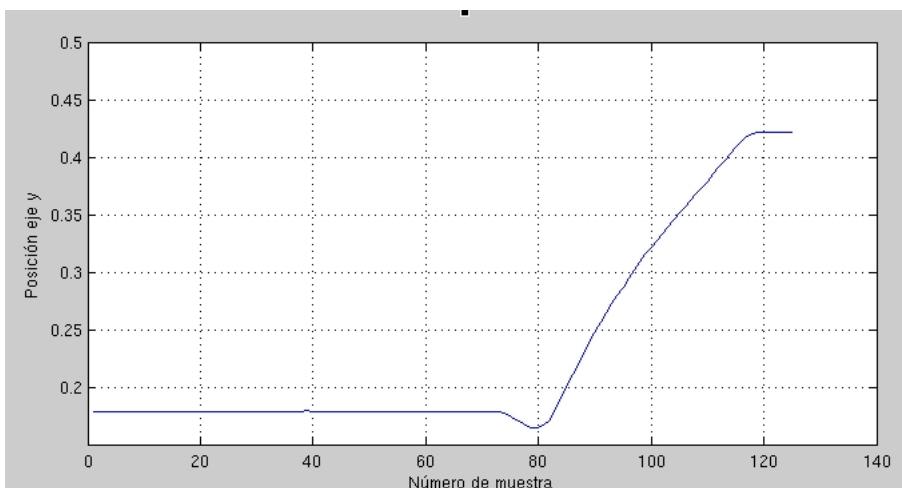


Figura 5.5: Evolución de la trayectoria coordinada en el eje y.

En el caso de la trayectoria continua, figuras 5.6 y 5.7, se observa como tanto la evolución en el eje x como en el y muestran un perfil cúbico, como se anticipó en el capítulo 4. Esto permite que la trayectoria real sea mucho más parecida a la que se espera al hacer la planificación, y evita cambios bruscos en la velocidad.



**Figura 5.6:** Evolución de la trayectoria continua en el eje x.



**Figura 5.7:** Evolución de la trayectoria continua en el eje y.

Las siguientes dos figuras muestran la evolución de la trayectoria en los ejes x e y conjuntamente para la trayectoria coordinada y continua respectivamente. Los cruces rojos representan los puntos objetivo de la trayectoria, mientras que la línea azul es la trayectoria real seguida.

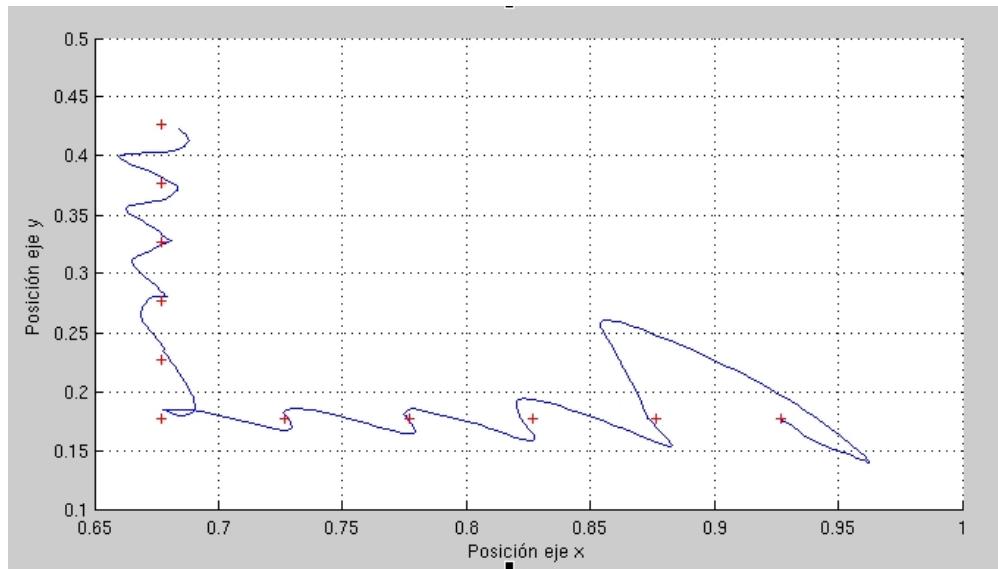


Figura 5.8: Evolución de la trayectoria en el eje y.

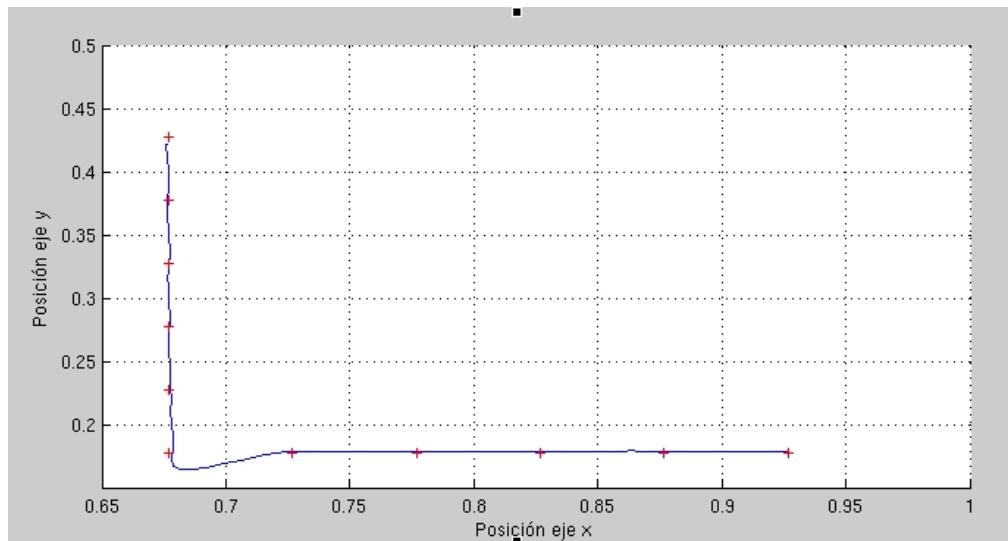


Figura 5.9: Evolución de la trayectoria en el eje y.

Se puede observar como la trayectoria continua resulta casi igual a la esperada cuando se hace la definición de ésta, mientras que en la trayectoria coordinada, en ciertos momentos, la posición del efecto final se aleja bastante de la

trayectoria que cabría esperar. Además los puntos objetivo se alcanzan con mayor precisión en el caso de la continua.

### **5.3. Detector de contacto con el sensor JR3.**

Como ya se explicó en el capítulo 4, se ha utilizado el sensor de fuerza/parrilla JR3 para detectar el contacto del efecto final del brazo de Manfred con objetos. Para poder realizar la detección, se precisó de la utilización de dos filtros, un filtro paso bajo proporcionado por la electrónica del sensor para eliminar ruido, y un filtro de mediana para facilitar la detección en base a diferencias entre muestras consecutivas. En las siguientes figuras se muestra la evolución de las medidas tomadas por el sensor a lo largo del esquema de procesado que se ha seguido.

En las siguientes tres imágenes podemos observar las medidas tomadas por el sensor para cada uno de los ejes x, y, z. En cada una de ellas hay dos gráficas, la de la izquierda corresponde con las medidas sin filtrar, y la de la derecha son las medidas después de pasar por el filtro paso bajo de 31,25Hz. Para cada una de ellas tenemos, en azul, las medidas originales, y en rojo, las diferencias entre cada medida y la anterior. Los datos que se muestran han sido tomados haciendo que la pinza del brazo choque con una caja de cartón de unos 500g mientras ejecutan una trayectoria similar a la mostrada en la sección anterior.

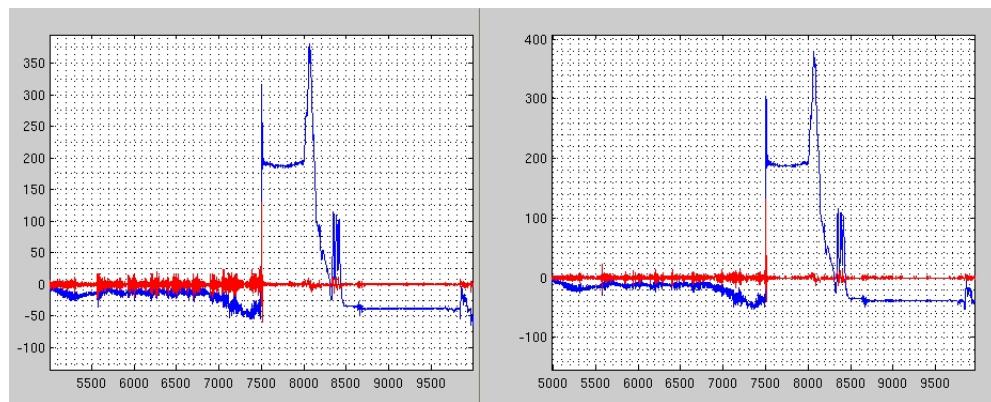


Figura 5.10: Medidas del sensor en el eje x.

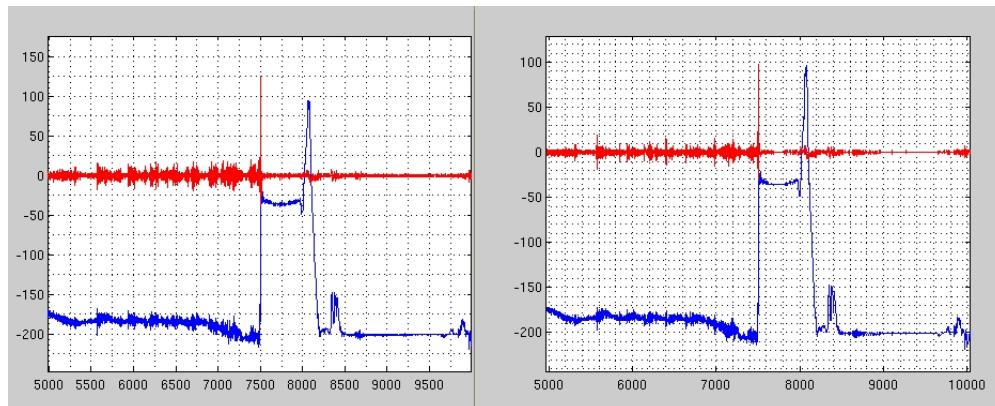


Figura 5.11: Medidas del sensor en el eje y.

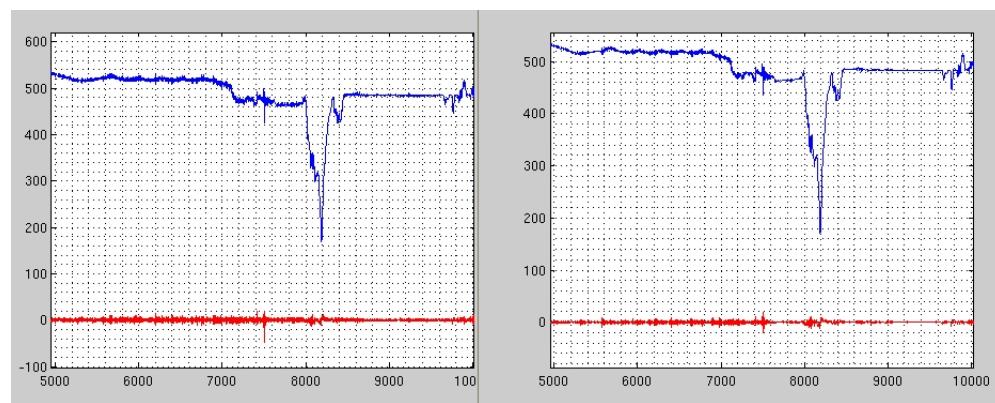
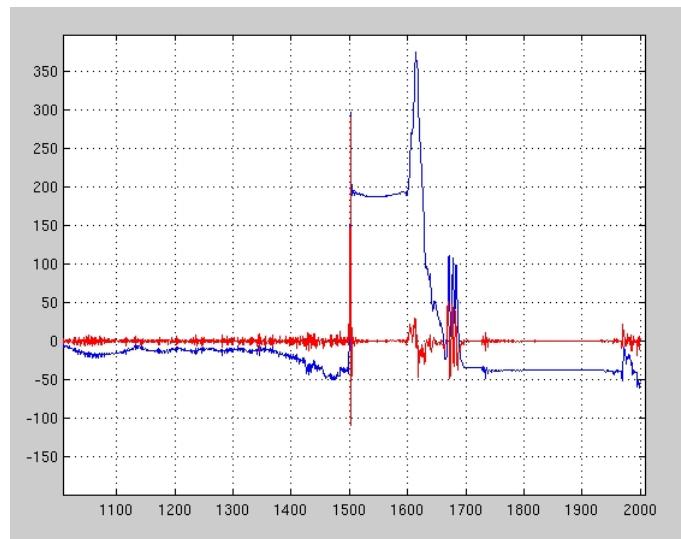


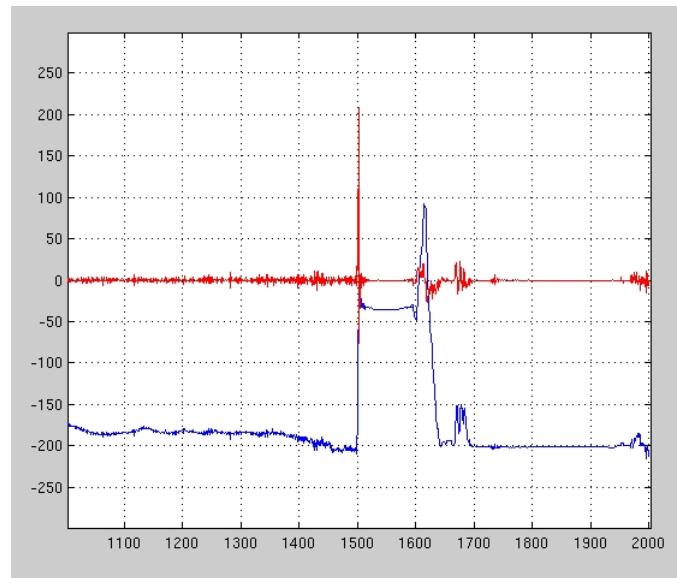
Figura 5.12: Medidas del sensor en el eje z.

Se puede observar que en las imágenes de la derecha hay menor cantidad de ruido ya que se eliminan las componentes de alta frecuencia. Además, se suavizan un poco los picos producidos, lo que ayuda a disminuir los picos producidos por los cambios de velocidad en las articulaciones.

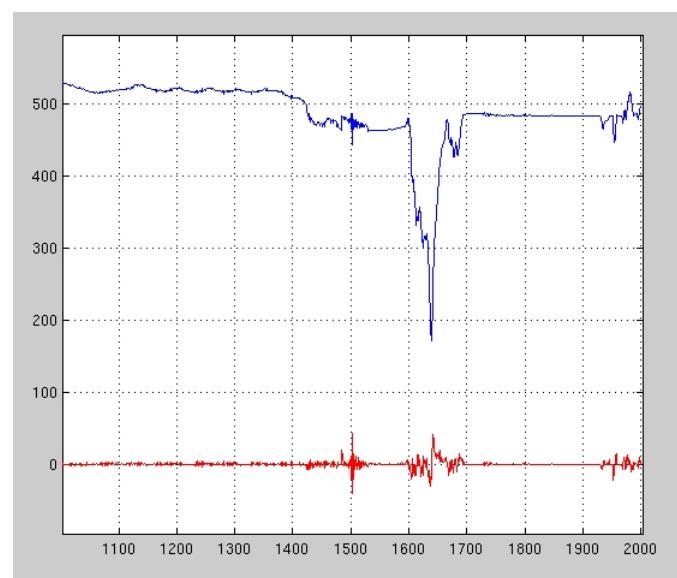
A continuación los datos obtenidos del filtro paso bajo se pasan por un filtro de mediana, los datos obtenidos se muestran a continuación. Al igual que en las gráficas anteriores los datos en azul son los corresponden con la salida del filtro, y los datos en rojo son la diferencia con la muestra anterior.



**Figura 5.13:** Medidas del sensor en el eje x después del filtro de mediana.



**Figura 5.14:** Medidas del sensor en el eje y después del filtro de mediana.



**Figura 5.15:** Medidas del sensor en el eje z después del filtro de mediana.

Podemos ver como al pasar por este filtro, se facilita la detección de los picos debido a un aumento bastante considerable en la señal que corresponde con la diferencia entre medidas consecutivas. Por tanto, a pesar de que se produce un retardo en la detección de contacto producido por el paso de los datos por los sucesivos filtros, ha sido ésta la señal utilizada ya que ofrece una mayor robustez. Además, este retardo nunca es mayor de 80ms.

Capítulo **6**

# Conclusiones y futuras líneas de trabajo.

## 6.1. Conclusiones.

Se ha podido implementar un controlador cartesiano para el brazo robótico LWR-UC3M-1 instalado en Manfred de manera satisfactoria gracias a las APIs ofrecidas por ROS y a la librería de cinemática y dinámica de OROCOS. Con este controlador, se han desarrollado dos tipos de trayectorias: coordinada y continua.

Se ha comprobado la dificultad que entraña comenzar a programar bajo el sistema operativo de ROS. Sin embargo, también conviene destacar que una vez se tienen claros los conceptos básicos y se adquiere costumbre de las características de programación requeridas, aumenta considerablemente la velocidad de desarrollo. Además, cabe destacar la posibilidad de poder consultar del código fuente de aplicaciones parecidas y de reutilización de parte de ese código.

Se ha comprobado que la utilización de la matriz jacobiana inversa para el cálculo del cambio de variable, de velocidad cartesiana a velocidad articular, añade una problemática al control del movimiento debido a sus características de funcionamiento en los puntos singulares y su entorno. Aunque, por otra parte, se ha dado una solución sin añadir mucha dificultad al controlador.

Se ha desarrollado una API en C++ para el sensor JR3 que facilitará su futuro uso en aplicaciones más complejas como la manipulación de objetos.

Se ha desarrollado una pequeña aplicación de detección de contacto con objetos que ha permitido comprobar, tanto la utilidad de la API del sensor de fuerza/par, como las prestaciones que ofrece el sistema operativo ROS a la hora de comunicar y ensamblar diferentes módulos del sistema de procesamiento del robot. Se ha comprobado que la detección de contacto con objetos no resulta sencilla utilizando solamente la información proporcionada por un sensor de fuerza, siendo casi imposible si los objetos a detectar son muy livianos o apenas presentan resistencia al movimiento del brazo robótico.

## **6.2. Mejoras.**

Como mejoras sobre el trabajo realizado se podrían plantear:

1. Buscar o implementar otra librería para la resolución de la cinemática inversa que no tenga la misma problemática que la KDL en los puntos singulares.
2. Desarrollar una cinemática inversa que sea capaz de trabajar con sistemas redundantes, de manera que se puedan incorporar al espacio de soluciones los dos grados de libertad que ofrece el movimiento de la base.

3. Realizar una interfaz gráfica que facilite la utilización de la API del sensor JR3 y que permita la visualización en tiempo real de las medidas que se realizan.
4. Añadir el sistema de detección de contacto al controlador que realiza la trayectoria continua.
5. Desarrollar otros protocolos de respuesta ante el contacto de objetos, como por ejemplo el agarre del objeto con el que se ha contactado.
6. Implementar otras aplicaciones usando el sensor de fuerza/par, como por ejemplo el cálculo del peso de un objeto agarrado con la pinza.

### 6.3. Futuras líneas de trabajo.

El trabajo desarrollado en esta tesis de máster sienta las bases para el desarrollo de aplicaciones más complejas en el campo de la manipulación de objetos con el robot Manfred.

Por una parte, para poder empezar a realizar manipulación real de objetos es necesario añadir un nuevo módulo de control de fuerza que sustituya al control cinemático cuando la manipulación se está llevando a cabo. Por tanto, será necesario un estudio de los diferentes tipos de control de fuerza existentes, así como su posible implementación bajo ROS.

Además, como parte del proyecto "Técnicas de aprendizaje y planificación para manipulación diestra en manipuladores móviles (Dex-arm)", dentro del cual se engloba esta tesis de máster, se va a sustituir el efecto final del brazo robótico de Manfred, que pasará a ser una mano robótica de 16 grados de libertad que permitirá realizar manipulación de objetos de manera más diestra. Este hecho obliga a investigar la manera de realizar tanto la planificación como el

control de movimiento de tan complejo mecanismo. A su vez será necesario el uso de nuevos sensores de fuerza localizados a lo largo de la palma de la mano para poder cerrar el lazo de control en el proceso de manipulación.

# Referencias

- Ansari, S. A. (2011). *Desarrollo de un manipulador móvil autónomo con características antropométricas*. Tesis Docotral, Universidad Carlos III de Madrid.
- Barrientos, A., Peñín, L. F., Balaguer, C., y Aracil, R. (1997). *Fundamentos de robótica*. McGraw-Hill, Interamericana de España, S.A.
- Blanco, D., Ansari, S. A., Castejón, C., Boada, B. L., y Moreno, L. E. (2005). Manfred: Robot antropomórfico de servicios fiable y seguro para operar en entornos humanos. *Revista Iberoamericana de Ingeniería Mecánica*, 9(3), 33–48.
- Corke, P. I. (2001, Abril). Robotics toolbox for matlab (release 6) [Manual de software informático].
- Craig, J. (2006). *Robótica* (3<sup>a</sup> ed.). Pearson Educación.
- Delta Tau Data Systems, I. (2003). Pmac2: User manual [Manual de software informático].
- Delta Tau Data Systems, I. (2004). Pmac2: Software manual [Manual de software informático].
- Denavit, J., y Hartenberg, R. (1955, Junio). A kinematic notation for lower-pair mechanisms based on matrices. *Journal of applied mechanics*.

- Ebert, D. M., y Henrich, D. D. (2002). Safe human-robot-cooperation: Image-based collision detection for industrial robots. En *Ieee/rsj int. conf. on intelligent robots and systems* (pp. 1826–1831).
- Garage, W. (Ed.). (2011, Consultada durante: mar-sep). Ros wiki documentation. [Manual de software informático]. Disponible en [www.ros.org/wiki/](http://www.ros.org/wiki/)
- Goldenberg, A. A., Benhabib, B., y Fenton, R. (1985). A complete generalized solution to inverse kinematics of robots. *IEEE Journal of robotics and automation, RA-1*, 1.
- Hirzinger, G., Albu-Schäffer, A., Höhnle, M., Schaefer, I., y Sporer, N. (2001). On a new generation of torque controlled light-weight robots. En *Ieee int. conf. on robotics and automation* (pp. 3356–3363).
- JR3, I. (1994). Jr3: Installation manual for force-torque sensors with internal electronics [Manual de software informático].
- Leuven, K. U. (Ed.). (2011, Consultado durante: feb-sep). The orocos project. [Manual de software informático]. Disponible en [www.orocos-project.org](http://www.orocos-project.org)
- Luca, A. D., Hirzinger, G., Albu-Schäffer, A., y Haddadin, S. (2006). *Collision detection and safe reaction with the dlr-iii lightweight manipulator arm*. IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing China.
- Lumelsky, V., y Cheung, E. (1993). Real-time collision avoidance in teleoperated whole-sensitive robot arm manipulator. *IEEE Trans. on Systems, Man, and Cybernetics.*, 23(1), 194–203.
- Prats, M. (2009). *Robot physical interaction through the combination of vision, tactile and force feedback*. Tesis Doctoral, Universidad Jaume I, Castellón.
- Sciavicco, L., y Siciliano, B. (2005). *Modelling and control of robot manipulators*. Springer-Verlag London Limited.

- Shujun, L., Chung, J. H., y Velinsky, S. A. (2005, Abril). *Human-robot collision detection and identification based on wrist and base force/torque sensors*. IEEE International Conference on Robotics and Automation Barcelona, Spain.
- Siciliano, B., Khatib, O., y cols. (2008). *Springer handbook of robotics*. Springer-Verlag Berlin Heidelberg.
- Tonietti, G., Schiavi, R., y Bicchi, A. (2005). Design and control of a variable stiffness actuator for safe and fast physical human/robot interaction. En *Ieee int. conf. on robotics and automation* (pp. 528–533).
- Zinn, M., Khatib, O., Roth, B., y Salisbury, J. K. (2005). A new actuation approach for human-friendly robot design. *Int. J. of Robotics Research*, 23(4/5), 379–398.