REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Universidad Católica de Santa María

Facultad de Ciencias e Ingenierías Físicas y Formales

Escuela Profesional de Ingeniería Electrónica

"DESARROLLO DE UN SISTEMA DE GENERACIÓN DE MOVIMIENTO PARA ROBOTS HUMANOIDES"

Tesis presentada por el Bachiller:

Ramos Ponce, Oscar Efraín

para optar el Título Profesional de:

Ingeniero Electrónico

Asesor:

Ing. Delgado Barra, Lucy

AREQUIPA - PERÚ
2018

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

UNIVERSIDAD CATOLICA DE SANTA MARIA
URB. SAN JOSE S/N - UMACOLLO

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

INFORME DICTAMEN DE BORRADOR DE TESIS

Los que suscriben:

Ing Lucy Delgado Barra

Ing Eduardo Esquivel Zenteno

Dictaminadores del Borrador de Tesis titulado:

Desarrollo de un Sistema de Generación de Movimiento para Robots Humanoides

Nombrados con Decreto Nro.: 006-DEPIE-2018 **, de fecha:** 15 noviembre del 2018

Presentado por el (la) los Bachiller (es):

3. Oscar Efrain Ramos Ponce
4. _____

Cumplen con emitir su Dictamen:

- No registra observaciones, **es procedente** (X)
- Registra observaciones que deben ser subsanadas (ver hoja adjunta) ( )

Arequipa, 10 Diciembre 2018

Lucy Delgado Barra
Código Nro: 1546

EDUARDO ESQUIVEL
Código Nro.: 1264

(5154) 382038 (5154) 252542 ucsm@ucsm.edu.pe http://www.ucsm.edu.pe 0010080

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Dedicatoria:

A mis padres, por su gran apoyo y cariño, por motivarme a seguir adelante, y estar pendientes de mí en cada momento.

Agradecimiento:

A mi asesora de tesis, por sus consejos, sugerencias y tiempo durante la elaboración de la presente tesis.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Presentación

Los robots manipuladores tradicionales poseen típicamente seis grados de libertad articular, los cuales les permiten alcanzar las tres posibles posiciones y orientaciones dentro de su espacio de trabajo. Sin embargo, cuando el espacio de trabajo presenta restricciones, estos robots ven reducida su capacidad de movimiento y se requiere robots con más grados de libertad, llamados redundantes, los cuales imitan el brazo humano. Más aún, en este proceso de imitación surgen los llamados robots humanoides, los cuales imitan cinemáticamente la estructura de un humano y, por tanto, son altamente redundantes.

Debido a la redundancia, estos robots requieren un tratamiento especial para la generación de movimiento: las técnicas de control utilizadas para robots manipuladores tradicionales tienen que ser extendidas para aprovechar la estructura cinemática de manera adecuada. Más aún, estos robots poseen una base flotante debido a lo cual se debe además generar movimiento para todo el cuerpo; es decir, el robot debe coordinar el movimiento de todas sus articulaciones en simultáneo, y no funcionar como cuatro manipuladores separados. El problema consiste en la generación y coordinación adecuada de movimiento que no esté enfocada en un solo robot, sino que pueda ser aplicada a diferentes robots redundantes con diferentes estructuras, siendo los robots manipuladores un caso particular.

El control adecuado de robots humanoides es un tema complicado tanto a nivel algorítmico como a nivel tecnológico. Por este motivo, los sistemas de control y generación de movimiento se encuentran generalmente orientados hacia robots específicos. Algunos sistemas son de código cerrado y están restringidos al laboratorio donde fueron desarrollados. Otros sistemas son de código abierto pero están basados en estructuras complejas de software desarrolladas para un robot en particular o para el uso propio del laboratorio de investigación. Debido a esto, dichos middlewares normalmente no se encuentran debidamente documentados y es complicado su uso.

En esta tesis se desarrollará un sistema genérico e independiente de generación de movimiento cinemático para robots humanoides. El sistema base no dependerá de estructuras complejas de software y podrá controlar cualquier robot humanoide y, por extensión, cualquier manipulador redundante o no, cuyo modelo matemático esté disponible. El sistema será verificado y probado con diversos modelos de robots humanoides y redundantes.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Resumen

Recientemente los robots humanoides, y en general los robots redundantes con base flotante, se han incrementado en universidades y labobratorios de investigación alrededor del mundo debido a las capacidades potenciales que presentan. Sin embargo, el control de movimiento de estos sistemas es diferente al control que se realiza para robots manipuladores debido a la gran cantidad de grados de libertad (motores) que poseen, los cuales requieren coordinación para alcanzar un movimiento adecuado. En esta tesis se presenta el desarrollo de un sistema cinemático de generación de movimiento para este tipo de robots, desde el diseño del sistema hasta la implementación del mismo y la aplicación en diversos modelos simulados.

El diseño del sistema se basa en el uso de tareas cinemáticas, las cuales a su vez toman como referencia conceptos de cinemática diferencial. Tradicionalmente la cinemática diferencial es aplicada a robots manipuladores con base fija, pero esto no es suficiente para el control de robots humanoides. En esta tesis se muestra paso a paso cómo pasar de un esquema tradicional a un esquema que considera el movimiento de la base flotante del robot. Esta descripción teórica se basa en el uso de coordenadas Cartesianas para la posición, y de cuaterniones para la orientación, con el fin de evitar singularidades de la configuración. Se aprovecha, además, la redundancia del sistema usando esquemas de control cinemático basados en ponderación y en jerarquía de tareas, los cuales permiten realizar varias tareas al mismo tiempo.

La implementación del sistema se realiza en C++ utilizando clases y conceptos de polimorfismo, y se basa en la lectura de un modelo URDF que especifica las características físicas de un robot. Este sistema es independiente de middlewares y solo depende de algunas librerías de código abierto. El sistema desarrollado es además de código abierto. Se emplea una interface con Python para poder describir el movimiento usando Python, debido a que es mucho más sencillo realizarlo así. El sistema es luego validado mediante una interfaz con ROS (Robot Operating System) para poder visualizar los resultados, usando modelos de robots manipuladores, de un robot arbitrario diseñado, y de dos robots humanoides. Así, se demuestra la aplicabilidad del sistema para cualquier tipo de robot con extremidades y con base fija o flotante, asumiendo que cada robot dispone de un control de bajo nivel de los motores, que es, en la práctica, un caso usual.

Palabras clave: robótica, control cinemático, generación de movimiento

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Abstract

Recently, humanoid robots and, in general, redundant robots with a floating base, have increased in universities and research labs around the world due to the potential capabilities that they have. However, their motion control is different from the control that is usually performed on robot manipulators due to the large number of degrees of freedom (motors) that they have, which require coordination to achieve proper motion. This thesis presents the development of a kinematic system for motion generation applied to this type of robots, ranging from the design of the system to its implementation and application in different simulated models.

The system design is based on kinematic tasks, which in turn use the concepts of differential kinematics. Traditionally, differential kinematics is applied to robot manipulators with a fixed base, but this is not enough for the control of humanoid robots. This thesis shows step by step how to extend the traditional scheme to a scheme that considers the motion of the floating base. This theoretical description is based on the use of Cartesian cooridnates for position, and quaternions for orientation, in order to avoid configuration singularities. The redundancy of the system is also considered using kinematic control schemes based on tasl weighting and hierarchy, which allow the achievement of several tasks at the same time

The implementation of the proposed system is done in C++ using classes and concepts of polymorphism, and is based on the parsing of a URDF model that specifies the physical characteristics of a robot. This system is independent of middlewares and it does only depend on some open source libraries. The developped system is also open source. An interface with Python was also developed since it is easier to describe motion using a script language. The system is then validated through an interface with ROS (Robot Operating System) to visualize the results, using models of robot manipulators, of an arbitrarily designed robot, and humanoid robots. Thus, the aplicability of the system to any type of robot with limbs and fixed or floating base is shown, assuming that each robot has a low level motor control, which is in practice a usual case.

Keywords: robotics, kinematic control, motion generation

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Índice general

Presentación IV

Resumen V

Abstract VI

Índice de Figuras X

Símbolos y Abreviaciones XII

1. Introducción 1
- 1.1. Descripción del Problema . . . 1
- 1.2. Justificación . . . 2
- 1.3. Antecedentes . . . 3
- 1.4. Hipótesis . . . 3
  - 1.4.1. Hipótesis Principal . . . 3
  - 1.4.2. Hipótesis Secundarias . . . 4
- 1.5. Objetivos . . . 4
  - 1.5.1. Objetivo General . . . 4
  - 1.5.2. Objetivos Específicos . . . 4
- 1.6. Alcances . . . 4
- 1.7. Estructura de la Tesis . . . 5

2. Fundamentos para la Generación de Movimiento 7
- 2.1. Representación Espacial de Cuerpos Rígidos . . . 7
  - 2.1.1. Posición de Cuerpos Rígidos . . . 7
  - 2.1.2. Orientación de Cuerpos Rígidos . . . 8
  - 2.1.3. Posición y Orientación . . . 12
  - 2.1.4. Velocidad de Cuerpos Rígidos . . . 12
- 2.2. Representación de un Robot Humanoide . . . 13
  - 2.2.1. Representación Mediante Coordenadas Generalizadas . . . 13
  - 2.2.2. Representación en el Espacio Operacional . . . 15
- 2.3. Cinemática de Robots . . . 16
  - 2.3.1. Cinemática Directa . . . 16
  - 2.3.2. Cinemática Inversa . . . 17
  - 2.3.3. El Jacobiano . . . 19
  - 2.3.4. Cinemática Diferencial . . . 21
- 2.4. Redundancia en Robótica . . . 23

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

2.4.1. Redundancia Cinemática . . . 23
2.4.2. Cinemática Diferencial con Redundancia . . . 24
2.4.3. Redundancia en Robots con Base Flotante . . . 25
2.5. Generación de Movimiento . . . 26
2.5.1. Control Cinemático de Robots Humanoides. . . . 26
2.5.2. Otras Formas de Generación de Movimiento en Robótica . . . 27

3. Diseño del Sistema Cinemático de Control **30**
3.1. Función de Tarea . . . 30
3.2. Jacobiano de un Robot Humanoide . . . 31
3.2.1. Efecto del Twist de la Base Flotante . . . 32
3.2.2. Relación entre Velocidades de la Base Flotante . . . 33
3.2.3. Jacobiano Geométrico de un Robot Humanoide . . . 35
3.3. Generación Cinemática de Movimiento . . . 36
3.3.1. Componentes Cinemáticos . . . 37
3.3.2. Control Cinemático . . . 39
3.3.3. Integración de la Velocidad Articular Generalizada . . . 41
3.4. Esquemas de Control Cinemático . . . 42
3.4.1. Solución usando Ponderación de Tareas . . . 43
3.4.2. Solución Proyectando en el Espacio Nulo . . . 45
3.4.3. Solución con Jerarquía de QPs . . . 48
3.5. Tareas Cinemáticas . . . 52
3.5.1. Tarea de Posición . . . 52
3.5.2. Tarea de Orientación . . . 53
3.5.3. Tarea de Posición y Orientación . . . 55
3.5.4. Tarea de Postura . . . 56
3.5.5. Otras Tareas de Desigualdad . . . 56

4. Implementación del Sistema de Generación de Movimiento **58**
4.1. Herramientas utilizadas en el Sistema . . . 58
4.1.1. Manejo de Álgebra Lineal . . . 59
4.1.2. Modelamiento del Robot . . . 60
4.1.3. Cinemática Básica del Robot . . . 61
4.1.4. Resolución de QPs . . . 62
4.2. Estructura General del Sistema . . . 62
4.2.1. Generalidades de la Implementación en C++ . . . 66
4.2.2. Generalidades de la Interfaz con Python . . . 69
4.2.3. Características Principales del Sistema Desarrollado . . . 71
4.3. Estructura Detallada del Sistema . . . 72
4.3.1. Modelo del Robot y Cinemática Básica. . . . 73
4.3.2. Arquitectura de Tareas . . . 79
4.3.3. Control Cinemático . . . 86
4.3.4. Funciones de Ayuda. . . . 91
4.4. Interfaces de Uso . . . 91
4.4.1. Interfaz con C++ . . . 91
4.4.2. Interfaz con Python . . . 96
4.5. Integración con ROS (*Robot Operating System*) . . . 100
4.5.1. Generalidades de ROS . . . 100

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

4.5.2. Generación de Movimiento en ROS 102

5. Pruebas y Resultados **103**
5.1. Movimiento para Robots Manipuladores 103
5.1.1. Robot KUKA KR-5 (seis grados de libertad) 104
5.1.2. Robot Sawyer (siete grados de libertad) 111
5.2. Robot Simple de Base Flotante 117
5.2.1. Especificaciones del robot 118
5.2.2. Análisis Cinemático Analítico 119
5.2.3. Generación de Movimiento 122
5.3. Movimiento para Robots Humanoides 126
5.3.1. Robot NAO 127
5.3.2. Robot Valkyrie 138

6. Conclusiones y Recomendaciones **145**

A. Cuaterniones **148**
A.1. Operaciones con cuaterniones 148
A.2. Cuaterniones Unitarios y Rotaciones 149

B. Modelo de un Robot de Base Flotante en URDF **152**

C. Extractos de Código **157**
C.1. Modelo del Robot 157
C.2. Tareas Cinemáticas 158
C.3. Control Cinemático 159
C.4. Funciones para Posición y Orientación 163

D. Funciones y Documentación **167**

Bibliografía **171**

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Índice de figuras

2.1. Relaciones entre los espacios nulo y de columnas de J. 25

3.1. Solución de (3.25) para varios valores de λ. 38
3.2. Ejemplo de variación de la ganancia con el error, según (3.26). 39
3.3. Esquema del sistema de control cinemático: nivel conceptual. 40

4.1. Repositorio en github del paquete implementadoq, llamado *oscr*. 63
4.2. Página principal de la documentación usando Doxygen. 69
4.3. Ejemplo de documentación de una función usando Doxygen. 70
4.4. Esquema de la obtención de la cinemática directa y del Jacobiano. 73
4.5. Diagrama de herencia para el modelo del robot y la cinemática directa. 74
4.6. Arquitectura de una tarea cinemática genérica. 80
4.7. Diagrama de herencia para las clases que implementan tareas cinemáticas 81
4.8. Diagrama de la generación cinemática de movimiento basada en tareas. 86
4.9. Diagrama de herencia para las clases del control cinemático. 87
4.10. Salida de la lectura con C++ del modelo de un robot y su cinemática. 92
4.11. Salida del control cinemático con C++ para un robot de base flotante. 94
4.12. Posición alcanzada y articulaciones usadas para el movimiento de ejemplo 96
4.13. Salida de la lectura con Python del modelo de un robot y su cinemática. 98

5.1. Robot KR-5 de KUKA: real (izquierda), en ROS (derecha). 104
5.2. Algunas características del robot KR-5 de KUKA. 105
5.3. Posición (izquierda) y orientación (derecha) del efector final. 105
5.4. Valores articulares en función del tiempo para el Robot KR-5. 106
5.5. Trayectoria cartesiana de la posición inicial a la final. 107
5.6. Tiempo de cálculo para KR-5 usando HQP. 107
5.7. Trayectoria Cartesiana de seguimiento de referencia lineal. 108
5.8. Posición y orientación para el seguimiento de trayectoria lineal 108
5.9. Valores articulares en función del tiempo para el Robot KR-5 109
5.10. Trayectoria en y para distintos valores de ganancia λ. 110
5.11. Posición y orientación del efector final para una trayectoria circular. 111
5.12. Valores articulares de KR-5 para la trayectoria circular. 111
5.13. Trayectoria Circular en el plano Cartesiano. 112
5.14. Robot Sawyer: real (izquierda), modelo en ROS (derecha). 112
5.15. Movimiento del codo manteniendo el efector final 113
5.16. Posición del codo y del efector final de Sawyer 114
5.17. Trayectoria de cada una de las articulaciones del robot. 114
5.18. Desplazamiento en 3D del codo del robot. 115
5.19. Sawyer desplazando objetos en Gazebo 116

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

5.20. Trayectoria de cada articulación para el movimiento en Gazebo. 117
5.21. Trayectoria en x, y, z para el desplazamiento de objetos en Gazebo. 118
5.22. Modelo del robot de 6 grados de libertad de base flotante 119
5.23. Asignación de coordenadas para el robot de 6 grados de libertad 119
5.24. Movimiento del robot de base flotante de 6 gdl 123
5.25. Posición y orientación de la extremidad izquierda con movimiento. 124
5.26. Posición y orientación de la extremidad derecha (fija). 125
5.27. Posición y orientación de la base flotante. 126
5.28. Configuración articular en función del tiempo. 126
5.29. Robot NAO de SoftBank Robotics: real (izquierda), en ROS (derecha). 127
5.30. Movimiento de ambos brazos del robot NAO 128
5.31. Trayectoria de las 26 articulaciones de NAO en función del tiempo. 129
5.32. Posición y orientación de la mano derecha. 130
5.33. Posición y orientación de la mano izquierda. 130
5.34. Posición y orientación de la base del robot. 131
5.35. Tiempo de computación para la generación de cada ciclo de control 131
5.36. Ejemplos de movimiento del robot NAO 133
5.37. Movimiento de la base del robot NAO 134
5.38. Movimiento con tareas conflictivas usando WQP y HQP 135
5.39. Movimiento acrobático del robot NAO 137
5.40. Secuencia de tareas para el movimiento acrobático de NAO. 138
5.41. Robot Valkyrie de NASA-JSC: real (izquierda), en ROS (derecha). 139
5.42. Ejemplos de movimiento del robot Valkyrie 140
5.43. Movimiento acrobático del robot Valkyrie 141
5.44. Evolución de la posición de ambas manos. 142
5.45. Evolución de la posición de la pelvis y del pie izquierdo. 143
5.46. Evolución de la configuración articular. 144
5.47. Tiempo de computación para el movimiento acrobático. 144

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Símbolos y Abreviaciones

gdl (*dof*)	Grados de Libertad (*Degrees of Freedom*)
SO(3)	Espacio ortogonal especial en $\mathbb{R}^3$ (*Special Orthogonal*)
SE(3)	Espacio Euclideano en $\mathbb{R}^3$ (*Special Euclidean*)
so(3)	Álgebra de Lie asociado a SO(3)
se(3)	Álgebra de Lie asociado a SE(3)
QP	Problema Cuadrático de Optimización (*Quadratic Program*)
$\boldsymbol{q}$	Configuración articular generalizada (incluyendo base flotante)
$\boldsymbol{q}_a$	Configuración articular actuada
$\boldsymbol{x}$	Representación de la posición y orientación
$\boldsymbol{p}$	Vector de posición
$\boldsymbol{o}$	Representación genérica de la orientación
$\boldsymbol{R}$	Matriz de rotación
$\boldsymbol{Q}$	Cuaternión $(w, \boldsymbol{\epsilon})$
$\boldsymbol{\epsilon}$	Parte vectorial de un cuaternión $(\epsilon_x, \epsilon_y, \epsilon_z)$
$\mathbb{H}$	Espacio Hamiltoniano de los cuaterniones
$\boldsymbol{T}$	Matriz de transformación homogénea
$\boldsymbol{v}$	Velocidad lineal
$\boldsymbol{\omega}$	Velocidad angular
$\hat{\boldsymbol{\omega}}$	Matriz antisimétrica asociada con $\boldsymbol{\omega}$
$\boldsymbol{\xi}$	Twist (velocidad lineal y angular)
J	Jacobiano analítico
$J^{\#}$	Pseudo-inversa (usualmente de Moore-Penrose) de J
J_G	Jacobiano geométrico con componentes J_v, J_ω
E	Matriz que relaciona el Jacobiano analítico con el geométrico
T	Inversa de E correspondiente a la orientación (E_o^{-1})
c_θ	$\cos\theta$
s_θ	$\sin\theta$
$\boldsymbol{e}$	Función de tarea (error)

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Capítulo 1

Introducción

En este capítulo se presentará, describirá y justificará el problema que busca resolver la presente tesis, indicando además los antecedentes, los objetivos y los alcances de la misma. De igual modo, se dará a conocer de manera general el enfoque utilizado para solucionar el problema, mostrando los aportes de la tesis y puntualizando la estructura del trabajo por capítulos.

1.1. Descripción del Problema

Los robots manipuladores utilizados para tareas industriales altamente automatizadas poseen típicamente seis grados de libertad articular. Estos grados de libertad les permiten alcanzar las tres posibles posiciones y orientaciones dentro de su espacio de trabajo. Sin embargo, cuando el espacio de trabajo presenta ciertas restricciones, como por ejemplo obstáculos, estos robots ven reducida su capacidad de movimiento. Por este motivo, se desarrollaron robots manipuladores de siete grados de libertad, donde el grado adicional brinda redundancia espacial. Estos manipuladores redundantes, a veces denominados antropomórficos, se inspiran en el movimiento redundante del brazo humano. Al imitar no solo el brazo sino toda la estructura antropomórfica, se llega a los llamados robots humanoides, que fueron desarrollados como robots de servicio para ayudar a las personas en ambientes cotidianos. Estos robots poseen un aproximado de treinta grados de libertad, lo cual los convierte en robots altamente redundantes en el espacio tridimensional.

Los robots redundantes requieren un tratamiento especial para la generación de movimiento: las técnicas de control utilizadas para robots manipuladores tradicionales tienen que ser extendidas para aprovechar la estructura cinemática de manera adecuada. Más aún, los robots humanoides poseen una base flotante debido a lo cual, además de resolver la redundancia, se debe generar movimiento de cuerpo completo. Esto es, el robot debe coordinar el movimiento de todas sus articulaciones en simultáneo, y no funcionar como cuatro manipuladores separados (por ejemplo, para desplazar

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

su centro de masa). El problema aquí consiste en la generación y coordinación adecuada de movimiento que no esté enfocada en un solo robot, sino que pueda ser aplicada a diferentes robots redundantes con diferentes estructuras, siendo los robots manipuladores un caso particular.

En esta tesis se desarrollará un sistema genérico e independiente de generación de movimiento cinemático para robots humanoides. El sistema base no dependerá de estructuras complejas de software y podrá controlar cualquier robot humanoide y, por extensión, cualquier manipulador redundante o no, cuyo modelo matemático esté disponible. El sistema será verificado y probado con diversos modelos de robots humanoides y redundantes.

1.2. Justificación

El control adecuado de robots humanoides es un tema complicado de investigación actual tanto a nivel algorítmico como a nivel tecnológico. La estructura cinemática y dinámica de cada robot depende de cada fabricante, universidad o laboratorio de investigación que lo desarrolla. Por este motivo, los sistemas de control y generación de movimiento se encuentran generalmente orientados hacia robots específicos. Algunos sistemas son de código cerrado y están restringidos al laboratorio donde fueron desarrollados. Otros sistemas son de código abierto pero están basados en estructuras complejas de software ("middlewares") que fueron desarrolladas para un robot en particular o para el uso propio del laboratorio de investigación. Debido a esto, dichos middlewares normalmente no se encuentran debidamente documentados y para comprender su funcionamiento se requiere bastante dedicación (para la ingeniería inversa) y conocimiento bastante avanzado del lenguaje de programación utilizado.

La dependencia de los sistemas de generación de movimiento existentes de estructuras mucho más complejas hace que, incluso para utilizar una parte pequeña del sistema, se requiera instalar todo el "framework". Así, la utilización requiere la comprensión del framework, algo que no constituye parte esencial del sistema de control. Esto complica, por ejemplo, el desarrollo, implementación, comparación o prueba de diversas técnicas de generación de movimiento. Más aún, limita el uso en sistemas embebidos donde el espacio de memoria suele ser crítico, y hasta el uso educacional y académico. Además de esto, el soporte a robots para los cuales no fueron diseñados resulta bastante engorroso por el problema de la falta de documentación y estructura compleja.

Actualmente no existe un sistema de generación de movimiento para robots humanoides (y por extensión, redundantes) que sea genérico a cualquier robot y que sea independiente de sistemas complejos innecesarios. Debido a este problema, el aporte de esta tesis es el desarrollo de un sistema completo autocontenido que soporta la generación de movimiento para cualquier robot humanoide

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

(o manipulador) y es independiente de middlewares externos. El sistema será además de código abierto para facilitar la adaptación, extensión, colaboración, benchmarking y hasta educación.

1.3. Antecedentes

La generación de movimiento para robots humanoides es un actual área de investigación activa en robótica. En el Perú no existen antecedentes oficiales de ningún sistema que haya sido desarrollado para controlar robots humanoides. Tampoco existen antecedentes de sistemas de control genéricos para robots redundantes con base fija. A nivel nacional, algunas tesis de pregrado y posgrado presentan el desarrollo de prototipos de brazos robóticos manipuladores (o alguna parte específica de los mismos), pero estos prototipos son de cinco grados de libertad o menos [1–4], muy por debajo de los grados de libertad necesarios para un robot redundante. Los métodos de control cinemático o dinámico de estos trabajos se basan en análisis exhaustivo ad-hoc al sistema desarrollado. Sin embargo, cada sistema es incompatible con el otro; el desarrollo de un trabajo no puede ser aprovechado por otro debido a la especificidad de la metodología y a la falta de una descripción unificada del modelo matemático del robot.

Sistemas genéricos de generación de movimiento pueden ser encontrados en trabajos de investigación de universidades y laboratorios principalmente de Estados Unidos, Francia, Alemania, Italia, España, Corea y Japón. Muchos de estos sistemas están restringidos al propio laboratorio de investigación y solo se conoce de su existencia por sus resultados en publicaciones científicas o por información personal de quienes trabajan allí. Algunos de los sistemas disponibles libremente (*open source*) más conocidos que permiten la generación de movimiento para robots humanoides son: *Stack of Tasks*, desarrollado en el LAAS-CNRS (Francia), *OpenSOT*, desarrollado en el IIT (Italia), *Drake*, desarrollado en el MIT, *Stanford Whole-body Control Framework*, desarrollado en la Universidad de Stanford, *ControlIt*, desarrollado en la Universidad de Texas, *Klampt*, desarrollado en Duke University, OpenRAVE, entre otros. Sin embargo, como se mencionó anteriormente, estos sistemas son dependientes de una estructura computacional (*framework*) bastante grande, lo cual complica su uso así como el desarrollo de nuevos esquemas de control.

1.4. Hipótesis

1.4.1. Hipótesis Principal

El sistema de generación de movimiento para robots humanoides y redundantes de base fija permitirá mover de manera coherente y deseada diversos robots siempre que su modelo matemático se encuentre adecuadamente especificado.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

1.4.2. Hipótesis Secundarias

- El control basado principalmente en tareas y en cinemática inversa puede generar movimiento de cuerpo completo para robots humanoides.
- Un sistema de generación de movimiento no necesita depender de estructuras complejas de software para poder generar movimiento.
- Es posible utilizar el sistema desarrollado con diversos robots redundantes, sean de base fija o de base flotante, así como con manipuladores tradicionales.
- Es suficiente la descripción matemática de un robot en formato URDF (Unified Robot Description Framework) para poder ser utilizado con el sistema propuesto.

1.5. Objetivos

1.5.1. Objetivo General

Desarrollar e implementar un sistema de generación de movimiento basado en tareas para robots humanoides y robots redundantes, incluyendo aquellos con base fija.

1.5.2. Objetivos Específicos

- Diseñar un sistema de control cinemático basado en tareas.
- Implementar esquemas de control cinemático basados tanto en ponderación de tareas como en prioridad de tareas.
- Realizar la integración del sistema desarrollado con ROS (*Robot Operating System*) que actualmente el casi estándar *de facto* en robótica.
- Probar el sistema desarrollado con modelos de diversos robots humanoides y redundantes.
- Realizar movimientos diversos utilizando el sistema de control desarrollado.

1.6. Alcances

Esta tesis busca el desarrollo de un sistema que permita generar movimiento para robots humanoides, comprendiendo también a manipuladores tradicionales y redundantes, utilizando principalmente control cinemático. Esta forma de control en robótica tiene como salida final la configuración

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

articular, la velocidad articular o la aceleración articular del robot y por tanto asume la existencia de algún tipo de servomecanismo de bajo nivel que controla directamente los motores.
En particular, se utilizará como señal de salida la configuración articular y se asumirá que cada motor del robot se comporta como un servomotor que controla directamente la posición articular. Así, la señal de salida podrá ser aplicada directamente a cualquier robot controlado en posición, lo cual no es una mayor limitación ya que la gran mayoría de robots industriales y robots humanoides son controlados en posición. Sin embargo, no se podrá controlar robots cuyo control sea el torque del motor, dado que para dicho caso se requeriría alguna forma de control dinámico. De igual modo, el control directo de fuerza e interacciones no será considerado.

1.7. Estructura de la Tesis

Esta tesis está conformada por los siguientes capítulos.

- *Capítulo 1: Introducción*. En este capítulo se brinda una introducción al problema a tratar dando a conocer la descripción del mismo así como la justificación y los antecedentes de la tesis. De igual manera, se presenta los objetivos de la presente tesis y los alcances de la misma.

- *Capítulo 2: Fundamentos para la Generación del Movimiento*. En este capítulo se da a conocer los conceptos fundamentales y los métodos existentes de generación de movimiento para robots altamente redundantes, con énfasis en robots humanoides, desde un punto de vista cinemático.

- *Capítulo 3: Diseño del Sistema Cinemático de Control*. En este capítulo se presenta la estructura cinemática utilizada para el sistema desarrollado, desde un punto de vista teórico así como de aplicación. Así, se introducen los fundamentos teóricos y metodológicos sobre los cuales se basan los siguientes capítulos de esta tesis.

- *Capítulo 4: Implementación del Sistema de Generación de Movimiento*. Este capítulo presenta los detalles de implementación del sistema propuesto en la presente tesis desde el punto de vista operativo; es decir, se muestra el detalle del software utilizado y de la estructura desarrollada. Esta implementación está basada en las bases teóricas del capítulo 3.

- *Capítulo 5: Pruebas y Resultados*. Este capítulo presenta los experimentos realizados de generación de movimiento a nivel de simulación cinemática y dinámica usando el sistema cinemático propuesto. Estos experimentos de desarrollan utilizando ROS (Robot Operating

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

System) por conveniencia, pero el núcleo del sistema desarrollado puede ser utilizado con cualquier middleware de robótica.

- *Capítulo 6: Conclusiones y Recomendaciones*. En este capítulo se presenta las principales conclusiones a las que se llegó durante la elaboración de esta tesis, así como algunas recomendaciones.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Capítulo 2

Fundamentos para la Generación de Movimiento

En este capítulo se presentará de manera sucinta los conceptos básicos relacionados con la cinemática de robots manipuladores y su extensión a robots redundantes. De igual manera, se describirá algunas técnicas actualmente utilizadas para el control cinemático de este tipo de robots.

2.1. Representación Espacial de Cuerpos Rígidos

La estructura mecánica de cualquier robot está constituida principalmente por eslabones unidos a través de articulaciones. Cada eslabón puede ser modelado como un cuerpo rígido y, por tanto, para poder especificar algún objetivo de control ligado a dicho cuerpo rígido es necesario poder representar su posición y orientación en el espacio. La excepción se da en la denominada robótica blanda (*soft robotics*) dado que dichos sistemas cuentan con eslabones flexibles; sin embargo, esta tesis considera solamente robots con eslabones rígidos. Para fines de control cinemático se ignora además el efecto de los sistemas de transmisión de movimiento.

2.1.1. Posición de Cuerpos Rígidos

La posición de un cuerpo rígido está representada por coordenadas Cartesianas, cilíndricas o esféricas. La representación más usual consiste en el uso de coordenadas Cartesianas tridimensionales asociadas a un punto fijo del cuerpo rígido; es decir, un vector $\boldsymbol{p} \in \mathbb{R}^3$ perteneciente a un espacio vectorial, dado por

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.1}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

donde $x, y, z \in \mathbb{R}$ son las coordenadas Cartesianas del punto en cuestión. Dicho punto se mueve de manera solidaria con el cuerpo rígido y describe completamente su posición con respecto a algún sistema de referencia.

2.1.2. Orientación de Cuerpos Rígidos

Para representar la orientación, es necesario asociar un sistema de coordenadas (también llamado sistema de referencia) al cuerpo rígido. De esta forma, la orientación consiste en describir este sistema de coordenadas con respecto a un sistema de coordenadas de referencia. Esta descripción se realiza de manera natural a través de una matriz de rotación $\boldsymbol{R} \in SO(3)$, donde $SO(3)$ denota el grupo de Lie ortogonal especial de dimensión 3 (las siglas provienen del inglés *Special Orthogonal* [5]). Formalmente, este grupo se define como

$$SO(3) = \{\boldsymbol{R} \mid \boldsymbol{R}\boldsymbol{R}^T = I \text{ y } \det(\boldsymbol{R}) = +1\} \tag{2.2}$$

donde I es la matriz identidad de 3×3 y $\det$ representa al determinante.
Considérese un sistema de referencia $\{\mathcal{C}\}$ asociado al cuerpo rígido, y considérese un sistema de referencia fijo o inercial $\{\mathcal{F}\}$. La orientación del sistema $\{\mathcal{C}\}$ se describe con respecto al sistema fijo a través de la matriz de rotación

$${}^{F}\boldsymbol{R}_{C} = \begin{bmatrix} | & | & | \\ {}^{F}\boldsymbol{x}_{C} & {}^{F}\boldsymbol{y}_{C} & {}^{F}\boldsymbol{z}_{C} \\ | & | & | \end{bmatrix} \tag{2.3}$$

donde cada columna de la matriz representa a los ejes unitarios del sistema $\{\mathcal{C}\}$ expresados en el sistema $\{\mathcal{F}\}$ (los subíndices denotan el sistema actual y los superíndices el sistema de referencia). La obtención matemática de esta matriz se basa en proyecciones ortogonales de los ejes y puede encontrarse en [6], [7].

Rotaciones Canónicas. Las matrices de rotación de un ángulo ϕ alrededor de los ejes x, y, o z se denominan rotaciones canónicas y tienen formas conocidas dadas por

$$\boldsymbol{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix}, \; \boldsymbol{R}_y(\phi) = \begin{bmatrix} c_\phi & 0 & s_\phi \\ 0 & 1 & 0 \\ -s_\phi & 0 & c_\phi \end{bmatrix}, \; \boldsymbol{R}_z(\phi) = \begin{bmatrix} c_\phi & -s_\phi & 0 \\ s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.4}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

donde c_ϕ y s_ϕ representan el coseno y seno del ángulo ϕ, respectivamente. Usando combinaciones de estas matrices de rotación se puede obtener cualquier rotación.

Un problema con el uso de matrices de rotación para representar orientación consiste en que se utiliza 9 elementos a pesar de que solamente existen 3 grados de libertad para la orientación. Debido a la ortonormalidad de sus columnas y filas, existen 6 elementos que brindan información redundante. Por este motivo, su utilización computacional no es eficiente. Además, la distancia entre dos matrices de rotación no se encuentra adecuadamente definida, dificultando la definición de un error de orientación [6].

Debido a los motivos anteriores, resulta más conveniente utilizar parametrizaciones de la rotación, las cuales utilizan menos elementos para representar la orientación. Algunas de estas parametrizaciones se describen a continuación.

Ángulos de Roll, Pitch, Yaw. Estos ángulos son denominados en español como alabeo, cabeceo y guiñada, pero el uso de los términos *roll*, *pitch*, *yaw* es bastante común en robótica. Existen diversas convenciones de ángulos de roll, pitch, yaw. La normalmente utilizada en robótica humanoide es la siguiente:

$$\boldsymbol{R}(\phi_r, \phi_p, \phi_y) = \boldsymbol{R}_z(\phi_y)\boldsymbol{R}_y(\phi_p)\boldsymbol{R}_x(\phi_r) \tag{2.5}$$

donde ϕ_r es el ángulo de roll, ϕ_p el ángulo de pitch y ϕ_y el ángulo de yaw. Realizando los productos matriciales, usando (2.4), la matriz de rotación equivalente es

$$\boldsymbol{R}(\phi_r, \phi_p, \phi_y) = \begin{bmatrix} c_{\phi_p}c_{\phi_y} & s_{\phi_r}s_{\phi_p}c_{\phi_y} - c_{\phi_r}s_{\phi_y} & s_{\phi_r}s_{\phi_y} + c_{\phi_r}s_{\phi_p}c_{\phi_y} \\ c_{\phi_p}s_{\phi_y} & s_{\phi_r}s_{\phi_p}s_{\phi_y} + c_{\phi_r}c_{\phi_y} & c_{\phi_r}s_{\phi_p}s_{\phi_y} - s_{\phi_r}c_{\phi_y} \\ -s_{\phi_p} & s_{\phi_r}c_{\phi_p} & c_{\phi_r}c_{\phi_p} \end{bmatrix} \tag{2.6}$$

donde c y s representan las funciones coseno y seno, respectivamente. De manera inversa, si se da una matriz de rotación $\boldsymbol{R}$ con componentes r_{ij}, estos ángulos están dados por

$$\phi_r = \text{atan2}\left(r_{32}, r_{33}\right), \quad \phi_p = \text{atan2}\left(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}\right), \quad \phi_y = \text{atan2}(r_{21}, r_{11}) \tag{2.7}$$

donde la función atan2 es el arco tangente que considera los ángulos en los cuatro cuadrantes. Con esta definición se tiene los siguientes rangos: $\phi_r \in [-\pi, \pi]$, $\phi_p \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ y $\phi_y \in [-\pi, \pi]$. Se debe notar que (2.7) solo es válido cuando $\phi_p \neq \{-\frac{\pi}{2}, \frac{\pi}{2}\}$ debido a que si es igual a dichos valores se tendrá $c_{\phi_p} = 0$ y por tanto ϕ_r y ϕ_y no estarán definidos (a lo que se conoce como singularidad de representación). En estos casos solo se puede recuperar $\phi_r + \phi_y$ o $\phi_r - \phi_y$. Analizando cada caso singular en (2.6), se puede demostrar que una alternativa es asignar $\phi_y = 0$, y usar $\phi_r =$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

atan2(r_{12}, r_{22}) si $\phi_p = \frac{\pi}{2}$, o $\phi_r = -\text{atan2}(r_{12}, r_{22})$ si $\phi_p = -\frac{\pi}{2}$. Los ángulos de roll, pitch, yaw, a pesar de ser ángulos bastante intuitivos presentan el problema de singularidades de configuración; es decir, existen tanto saltos como configuraciones que no se encuentran bien definidas [6].

Representación Eje/ángulo. La rotación de un ángulo ϕ alrededor de un eje unitario $\boldsymbol{u}$ está dada por la llamada *fórmula de Rodrigues* la cual se expresa como [5]:

$$\boldsymbol{R}(\phi, \boldsymbol{u}) = e^{\hat{\boldsymbol{u}}\theta} = I + \hat{\boldsymbol{u}} \sin\theta + \hat{\boldsymbol{u}}^2(1 - \cos\theta) \tag{2.8}$$

donde $\hat{\boldsymbol{u}}$ es la *matriz antisimétrica* (*skew-symmetric*) asociada con el vector $\boldsymbol{u}$ y definida como

$$\hat{\boldsymbol{u}} = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} \quad \text{cuando} \quad \boldsymbol{r} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}. \tag{2.9}$$

De igual manera, una notación equivalente es $\boldsymbol{u}^\wedge = \hat{\boldsymbol{u}}$. Esta matriz antisimétrica tiene un rol importante en robótica. Si se representa la velocidad angular como $\boldsymbol{\omega}$ (que no es necesariamente un vector unitario) y su matriz antisimétrica asociada como $\hat{\boldsymbol{\omega}}$, la relación entre la velocidad angular y la matriz de rotación está dada por

$$\hat{\boldsymbol{\omega}} = \dot{\boldsymbol{R}}\boldsymbol{R}^T \tag{2.10}$$

donde $\dot{\boldsymbol{R}}$ es la derivada de la matriz de rotación [5]. Una de las ventajas principales que presenta la representación eje/ángulo es que permite describir de manera directa cualquier orientación.
El problema inverso consiste en obtener el eje y ángulo dada una matriz de rotación $\boldsymbol{R}$ con componentes r_{ij}. En este caso, el ángulo equivalente es

$$\theta = \text{atan2}\left(\frac{\sqrt{(r_{21} - r_{12})^2 + (r_{31} - r_{13})^2 + (r_{32} - r_{23})^2)}}{2}, \frac{\text{tr}\boldsymbol{R} - 1}{2} \right) \tag{2.11}$$

el cual queda definido entre 0 y π, y el ángulo unitario de giro es:

$$\boldsymbol{u} = \frac{1}{2\sin\theta} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}. \tag{2.12}$$

Estas expresiones se obtienen del análisis de cada término de (2.8); los detalles se pueden encontrar en [5], [8]. Se debe además considerar que $\boldsymbol{u}$ no está bien definida cuando $\theta = 0$, dado que $\sin\theta =$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

0. En este caso, analizando término a término, se puede demostrar que se puede escoger $\boldsymbol{u}$ como:

$$\boldsymbol{u} = \frac{\theta}{\sqrt{3 - \text{tr}\boldsymbol{R}}} \begin{bmatrix} \text{sgn}(r_{32} - r_{23})\sqrt{r_{11} - r_{22} - r_{33} + 1} \\ \text{sgn}(r_{13} - r_{31})\sqrt{r_{22} - r_{33} - r_{11} + 1} \\ \text{sgn}(r_{21} - r_{12})\sqrt{r_{33} - r_{11} - r_{22} + 1} \end{bmatrix} \tag{2.13}$$

Cuaterniones. La rotación de un ángulo ϕ alrededor de un eje unitario $\boldsymbol{r}$ puede parametrizarse mediante 4 elementos denominados también *parámetros de Euler* [9], los cuales forman un *cuaternión* $\boldsymbol{Q} = (w, \epsilon_x, \epsilon_y, \epsilon_z)$ tal que

$$\boldsymbol{Q} = \left(\cos\left(\frac{\phi}{2}\right), \boldsymbol{r}\sin\left(\frac{\phi}{2}\right)\right) \tag{2.14}$$

donde el primer elemento (w) es denominado parte escalar, y los tres restantes $\boldsymbol{\epsilon} = (\epsilon_x, \epsilon_y, \epsilon_z)$ son llamados parte vectorial del cuaternión [6]. Formalmente, se tiene $\boldsymbol{Q} \in \mathbb{H}$, donde $\mathbb{H}$ representa el espacio Hamiltoniano de los cuaterniones. Se puede demostrar que la matriz de rotación equivalente a un cuaternión está dada por:

$$\boldsymbol{R}(w, \boldsymbol{\epsilon}) = \begin{bmatrix} 2(w^2 + \epsilon_x^2) - 1 & 2(\epsilon_x\epsilon_y - w\epsilon_z) & 2(\epsilon_x\epsilon_z + w\epsilon_y) \\ 2(\epsilon_x\epsilon_y + w\epsilon_z) & 2(w^2 + \epsilon_y^2) - 1 & 2(\epsilon_y\epsilon_z - w\epsilon_x) \\ 2(\epsilon_x\epsilon_z - w\epsilon_y) & 2(\epsilon_y\epsilon_z + w\epsilon_x) & 2(w^2 + \epsilon_z^2) - 1 \end{bmatrix}. \tag{2.15}$$

Dada una matriz de rotación $\boldsymbol{R}$, con componentes r_{ij}, el cuaternión equivalente es [9]:

$$\begin{aligned} w &= \frac{1}{2}\sqrt{r_{11} + r_{22} + r_{33} + 1} \\ \boldsymbol{\epsilon} &= \frac{1}{2}\begin{bmatrix} \text{sgn}(r_{32} - r_{23})\sqrt{r_{11} - r_{22} - r_{33} + 1} \\ \text{sgn}(r_{13} - r_{31})\sqrt{r_{22} - r_{33} - r_{11} + 1} \\ \text{sgn}(r_{21} - r_{12})\sqrt{r_{33} - r_{11} - r_{22} + 1} \end{bmatrix} \end{aligned} \tag{2.16}$$

donde $\text{sgn}(x)$ representa la función signo cuyo valor es 1 cuando $x \geq 0$, y -1 cuando $x < 0$. Los cuaterniones serán utilizados exhaustivamente en esta tesis debido a que no presentan configuraciones singulares; es decir, las rotaciones quedan siempre bien definidas. Debido a esta importancia, el Apéndice A resume el álgebra de cuaterniones y algunas de sus propiedades principales.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

2.1.3. Posición y Orientación

Es común representar la posición y la orientación por separados utilizando algunas de las representaciones descritas anteriormente. Una representación compacta está dada por las matrices de transformación homogénea $\boldsymbol{T}$ que contienen la matriz de rotación $\boldsymbol{R}$ asociada a un cuerpo rígido y el vector posición $\boldsymbol{p}$ del mismo.

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{p} \\ \boldsymbol{0} & 1 \end{bmatrix}. \tag{2.17}$$

Formalmente, la representación de posición y orientación de un cuerpo rígido pertenece al grupo de Lie $SE(3)$, denominado grupo Euclideano especial (en inglés *Special Euclidean*) y definido como [5]:

$$SE(3) = \{(\boldsymbol{R}, \boldsymbol{p}) \mid \boldsymbol{R} \in SO(3), \boldsymbol{p} \in \mathbb{R}^3\} = SO(3) \times \mathbb{R}^3. \tag{2.18}$$

La desventaja del uso de las matrices de transformación homogénea consiste en la redundancia de las matrices de rotación que la componen, y por este motivo se suele utilizar orientaciones parametrizadas.

2.1.4. Velocidad de Cuerpos Rígidos

La velocidad total de los cuerpos rígidos que conforman un robot tiene un componente lineal $\boldsymbol{v} \in \mathbb{R}^3$ y un componente angular $\boldsymbol{\omega} \in \mathbb{R}^3$. En robótica es frecuente, además, representar el componente de velocidad angular por la matriz antisimétrica equivalente $\hat{\boldsymbol{\omega}} \in so(3)$ donde $so(3)$ se define como [5]:

$$so(3) = \{\hat{\boldsymbol{\omega}} \in \mathbb{R}^{3\times 3} \mid \hat{\boldsymbol{\omega}} + \hat{\boldsymbol{\omega}}^T = 0\}. \tag{2.19}$$

Matemáticamente, a $so(3)$ se le denomina álgebra de Lie de $SO(3)$ y representa el espacio tangente a la identidad del grupo matricial $SO(3)$.

Para referirse tanto a la velocidad lineal como a la angular a la vez, es común utilizar el término *velocidad espacial* [10] o *twist* [5] $\boldsymbol{\xi} = (\boldsymbol{v}, \boldsymbol{\omega}) \in se(3)$ donde

$$se(3) = \{(\hat{\boldsymbol{\omega}}, \boldsymbol{v}) \mid \hat{\boldsymbol{\omega}} \in se(3), \boldsymbol{v} \in \mathbb{R}^3\}. \tag{2.20}$$

A $se(3)$ se denomina el álgebra de Lie del grupo matricial $SE(3)$ y representa el espacio tangente a la identidad de $SE(3)$. El uso de velocidades espaciales, en lugar de utilizar cada velocidad por separado, ha cobrado auge recientemente en robótica debido a la mayor eficiencia computacional comparada con los enfoques clásicos [11]. La teoría asociada a las velocidades espaciales puede encontrarse en [10].

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

2.2. Representación de un Robot Humanoide

Una de las diferencias principales entre un robot humanoide y un robot manipulador es que el primero se puede mover libremente en su entorno mientras que el último se encuentra fijo en el entorno. Esta habilidad de moverse libremente también genera un gran reto de control conocido como *sub-actuación*: el movimiento en el entorno no puede ser controlado directamente sino de manera indirecta a través del movimiento adecuado de las articulaciones [12]. Por este motivo, la representación de un robot humanoide debe considerar esta sub-actuación además de las coordenadas articulares.

2.2.1. Representación Mediante Coordenadas Generalizadas

Las coordenadas generalizadas son una generalización de las coordenadas articulares que incluyen, además de las articulaciones actuadas, la representación de la base flotante que no posee actuación directa.

Articulaciones actuadas. Las articulaciones actuadas de un robot están normalmente compuestas por dos elementos: articulaciones de revolución y articulaciones prismáticas [7]. Otros tipos de articulaciones actuadas no son comunes en robótica.

1. Una *articulación de revolución* es aquella que gira alrededor de un eje definido, y toma valores en $\mathbb{S}^1$, donde $\mathbb{S}^1$ es la variedad correspondiente al círculo unitario (esfera de dimensión 1). Un conjunto de r articulaciones de revolución toma valores en un toroide de dimensión r en $\mathbb{T}^r$ definido como $\mathbb{T}^r = \mathbb{S}^1 \times \cdots \times \mathbb{S}^1$ (r veces).

2. Una *articulación prismática* es aquella que se desplaza sobre un determinado eje de movimiento, y toman valores en $\mathbb{R}$ a lo largo de dicho eje. Un conjunto de p articulaciones prismáticas genera el espacio $\mathbb{R}^p$.

Un robot con $n = r + p$ grados de libertad, de las cuales r son articulaciones de revolución y p articulaciones prismáticas, posee un espacio articular descrito por una variedad n dimensional $\mathcal{Q}$ tal que $\mathcal{Q} = \mathbb{T}^r \times \mathbb{R}^p$. A dicha variedad $\mathcal{Q}$ se denomina el *espacio articular* o *espacio de configuración*, y contiene todos los posibles valores que puede tomar la variable articular actuada $\boldsymbol{q}_a$.

Base Flotante. La posición y orientación de un robot en su entorno se encuentra asociada con la posición y orientación de alguna parte rígida del robot, la cual es denominada *base flotante* o *raiz* (es también común el término *root*, en inglés). Para un robot humanoide, este cuerpo rígido es

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

típicamente el torso o la cintura [13]. La posición y orientación de la base flotante está representada por

$$\boldsymbol{x}_b = \begin{bmatrix} \boldsymbol{p}_b \\ \boldsymbol{o}_b \end{bmatrix} \in SE(3) \tag{2.21}$$

donde $\boldsymbol{p}_b$ es la posición de la base, usualmente dada por coordenadas Cartesianas ($\boldsymbol{p}_b \in \mathbb{R}^3$), y $\boldsymbol{o}_b$ es la orientación dada por alguna parametrización de $SO(3)$. En general, $\boldsymbol{x}_b$ es una parametrización del grupo $SE(3)$ y presenta 6 grados de libertad en el espacio, pero puede contener más de 6 coordenadas, dependiendo de la representación de orientación utilizada. El *twist* de la base flotante se denotará como

$$\boldsymbol{\xi}_b = \begin{bmatrix} \boldsymbol{v}_b \\ \boldsymbol{\omega}_b \end{bmatrix} \in se(3) \tag{2.22}$$

donde $\boldsymbol{v}_b$, $\boldsymbol{\omega}_b$ son las velocidades lineal y angular, respectivamente. Resulta común encontrar estos valores mediante mediciones obtenidas de la unidad de medida inercial *IMU* (*Inertial Measurement Unit*) que se encuentra usualmente en la base flotante del robot o cerca a la misma (en cuyo caso se debe realizar algunas transformaciones rígidas simples). Sin embargo, estas mediciones no son directas y se debe usar algún estimador, siendo común el uso del filtro de Kalman para la estimación [14].

Coordenadas Generalizadas. Las *coordenadas generalizadas* $\boldsymbol{q}$ representan el estado completo del robot y contienen tanto la base sub-actuada como las articulaciones actuadas [13]. Estas coordenadas generalizadas $\boldsymbol{q}$, junto con el vector de velocidad articular generalizada $\dot{\boldsymbol{q}}$, son

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{x}_b \\ \boldsymbol{q}_a \end{bmatrix} \quad \text{y} \quad \dot{\boldsymbol{q}} = \begin{bmatrix} \dot{\boldsymbol{x}}_b \\ \dot{\boldsymbol{q}}_a \end{bmatrix} \tag{2.23}$$

con cada término definido anteriormente. Se denomina *espacio articular generalizado* al espacio que contiene a $\boldsymbol{q}$ y que está compuesto por un componente de $SE(3)$ y un componente de $\mathcal{Q}$. Este espacio representa por completo la configuración de un robot humanoide, o en general, de un robot con base flotante.

Se debe notar que $\dot{\boldsymbol{q}}$ en (2.23) contiene al vector $\dot{\boldsymbol{x}}_b$ y por tanto es dependiente de la representación que se utilice para $\boldsymbol{x}_b$. Para evitar esta dependencia, en algunos casos resulta conveniente utilizar el twist de la base del robot en lugar de las derivadas de la representación, quedando:

$$\boldsymbol{v}_{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{\xi}_b \\ \dot{\boldsymbol{q}}_a \end{bmatrix}. \tag{2.24}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

La ventaja de esta representación con respecto a la anterior es que el twist $\boldsymbol{\xi}_b$ es independiente de la parametrización de $SE(3)$.

Matriz de Selección. La configuración articular actuada $\boldsymbol{q}_a$ puede ser recuperada a partir de las coordenadas generalizadas $\boldsymbol{q}$ utilizando la denominada *matriz de selección* S que selecciona solamente la parte actuada [15]. Para un robot con una representación b-dimensional de su base flotante y con n grados de libertad actuados, la matriz $S \in \mathbb{R}^{n\times(b+n)}$ es

$$S = \begin{bmatrix} \bar{0} & I \end{bmatrix} \qquad \text{tal que} \qquad \boldsymbol{q}_a = S\boldsymbol{q} \tag{2.25}$$

donde $\bar{0} \in \mathbb{R}^{n\times b}$ es una matriz de ceros y $I \in \mathbb{R}^{n\times n}$ es la matriz identidad. La matriz de selección es también importante para el mapeo de torques generalizados $\boldsymbol{\tau}$ que actúan en las articulaciones actuadas a las coordenadas generalizadas como

$$S^T \boldsymbol{\tau} = \begin{bmatrix} 0 \\ \boldsymbol{\tau} \end{bmatrix} \tag{2.26}$$

donde 0 corresponde a la sub-actuación del sistema, y se origina del hecho que la base $\boldsymbol{x}_b$ no puede ser controlada. Esta expresión para el torque es particularmente útil al analizar el modelo dinámico de un robot humanoide, pero no será usada en esta tesis, dado que el modelo utilizado aquí es puramente cinemático.

2.2.2. Representación en el Espacio Operacional

El *espacio operacional* [16], también denominado *espacio de la tarea* [17], es una variedad matemática que define el espacio en el que se realiza una determinada tarea y puede entenderse como una generalización del espacio Cartesiano dependiente del objetivo de control. Por ejemplo, el vector que contiene la posición y orientación del efector final de un manipulador robótico, está representado en el espacio operacional. De igual forma, si el robot posee una cámara en la cabeza y se desea que ésta enfoque un punto específico, las coordenadas de los píxeles de la cámara forman parte del espacio operacional.

Para robots humanoides no se utiliza el término "efector final", dado que éste usualmente se refiere al final de la cadena cinemática que se desea mover (mano del manipulador), pero en el caso de un humanoide, existen más partes con movimiento posible. En su lugar se utiliza el término *Punto Operacional*, el cual es una designación más genérica de cualquier parte del cuerpo del robot cuya posición y/o orientación se desea controlar. Algunos ejemplos de estos puntos operacionales son: las muñecas o las manos del robot, los tobillos o pies del robot, el pecho del robot, la cintura, la

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

cabeza, etc. Al referirse a puntos operacionales, es común interecambiar los términos muñeca con mano, y tobillo con pie, siempre que el significado esté claro. De igual manera, diversos puntos operacionales pueden ser definidos para otros tipos de robots con base flotante.

2.3. Cinemática de Robots

En robótica, el término cinemática hace referencia al estudio de la posición, orientación, velocidad y aceleración de los componentes que conforman el robot sin considerar las fuerzas o torques involucrados en el movimiento ni las propiedades dinámicas, como la masa o inercia, de los cuerpos que componen el sistema. El estudio cinemático comprende la cinemática directa e inversa, que se encargan solamente de la posición y orientación (algunos prefieren denominarla *geometría del robot*); y la cinemática diferencial, que se encarga de las velocidades y en menor medida de las aceleraciones.

2.3.1. Cinemática Directa

Consiste en encontrar la posición y orientación $\boldsymbol{x} \in SE(3)$ de algún punto operacional dada una determinada configuración articular $\boldsymbol{q} \in \mathcal{Q}$ del robot. Matemáticamente se describe mediante el mapa no lineal

$$f : \mathcal{Q} \to SE(3) \qquad \text{tal que} \qquad \boldsymbol{x} = f(\boldsymbol{q}), \tag{2.27}$$

el cual siempre se encuentra bien definido; es decir, para todo $\boldsymbol{q} \in Q$ siempre habrá un único $\boldsymbol{x} \in SE(3)$. Los métodos más usados para obtener y representar la cinemática directa en robots de cadena abierta son los parámetros de Denavit-Hartenberg y el método geométrico [6].

Método de Denavit-Hartenberg. Los parámetros estándar de Denavit-Hartenberg (DH) fueron introducidos en [18] y son ampliamente utilizados para modelar cinemáticamente robots manipuladores. El método consiste en primero asignar sistemas de referencia a cada eslabón del robot, y luego calcular 4 parámetros por cada grado de libertad usando las relaciones entre los sistemas asignados. Estas asignaciones se basan en una serie de reglas (que pueden ser encontradas, por ejemplo, en [6, 7, 18]) y solamente existe libertad para escoger el sistema de base y el sistema del efector final. Una vez obtenidos los parámetros, se asocia una matriz de transformación homogénea ${}^{i-1}\boldsymbol{T}_i$ a cada grupo de 4 parámetros, la cual relaciona un eslabón i con el eslabón anterior $i-1$. Para un robot de n grados de libertad, la cinemática del efector final queda representada como

$${}^{0}\boldsymbol{T}_n = {}^{0}\boldsymbol{T}_1\,{}^{1}\boldsymbol{T}_2\,{}^{2}\boldsymbol{T}_3 \cdots {}^{n-1}\boldsymbol{T}_n. \tag{2.28}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Se debe notar que existe una variación de estos parámetros, llamada parámetros DH modificados, la cual fue introducida en [19] y es seguida por algunos autores. A grandes rasgos, la diferencia consiste en asignar el sistema de referencia al final del eslabón en lugar de al principio. A pesar de ser muy parecidos, ambos métodos no son compatibles y se debe tener claro cuál se está utilizando.

Método Geométrico. Consiste en asignar arbitrariamente los sistemas de referencia a cada eslabón del robot. Dicha asignación se basa usualmente en conveniencia o facilidad matemática para la representación. La convención suele ser asignar el eje $\boldsymbol{z}$ a lo largo del eje de rotación. Una vez que se tiene los sistemas asignados, se obtiene la matriz de transformación homogénea que relaciona cada sistema con el anterior ${}^{i-1}\boldsymbol{T}_i$ y luego se realiza el mismo producto mostrado en (2.28). Alternativamente, aunque menos usado en la práctica, se puede utilizar teoría de *twists* basados en álgebra de Lie para representar la cadena cinemática [5, 11].

Cinemática Directa en Robots Humanoides. Los robots redundantes con base flotante, como los robots humanoides, presentan cadenas cinemáticas en forma de árbol en donde no es muy común el uso de los parámetros DH. Uno de los principales motivos consiste en que el método DH se enfoca principalmente en el efector final, asignando sistemas de referencia no intuitivos a los eslabones intermedios. Sin embargo, en robots humanoides resulta importante también tener sistemas de referencia intuitivos (por ejemplo, con el eje $\boldsymbol{x}$ apuntando hacia adelante y el eje $\boldsymbol{z}$ hacia arriba) en eslabones intermedios, como las rodillas o los codos, debido a que también se controla el movimiento de estas partes. Por estos motivos, es práctica común utilizar el método geométrico y realizar transformaciones homogéneas específicas entre eslabones contiguos [13]. Así, se describe cada cadena cinemática, como brazos o piernas, con respecto al sistema de la base flotante de forma independiente. Luego, se realiza transformaciones para que todo quede expresado en un sistema de referencia inercial, como se detalla en el Capítulo 3.

2.3.2. Cinemática Inversa

Consiste en encontrar la configuración articular $\boldsymbol{q} \in \mathcal{Q}$ necesaria para poder alcanzar una posición y orientación $\boldsymbol{x} \in SE(3)$ de un determinado punto operacional. En algunos casos específicos puede ser representada por el mapa inverso

$$f^{-1} : SE(3) \to \mathcal{Q} \qquad \text{tal que} \qquad \boldsymbol{q} = f^{-1}(\boldsymbol{x}). \tag{2.29}$$

Debido a la redundancia y alta no linealidad del sistema, en general, $f^{-1}(\boldsymbol{x})$ suele no ser único y podría presentar infinitas soluciones. Además, para configuraciones $\boldsymbol{x}$ que el robot no puede

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

alcanzar, f^{-1} no está definido. Debido a estos problemas, la cinemática inversa es más complicada de calcular que la cinemática directa.

Métodos Analíticos. Para cadenas cinemáticas simples, la cinemática inversa puede ser resuelta de manera analítica mediante los siguientes métodos clásicos:

a. *Métodos geométricos.* Consisten en analizar de manera geométrica la estructura cinemática del robot para encontrar relaciones entre los ángulos y las posiciones u orientaciones del efector final. Son métodos, en general, ad-hoc a cada problema y difíciles de generalizar. Algunas simplificaciones son el desacoplo de articulaciones cuando los últimos tres ejes se intersectan, llamado desacoplo de Peiper [20], o los subproblemas de Paden-Kahan que se basan en el producto de exponenciales.

b. *Métodos algebraicos.* Consisten en la obtención de las variables articulares a partir de la transformación homogénea resultante de la cinemática directa (2.28) usando álgebra. Algunos métodos realizan transformaciones inversas o eliminaciones dialíticas para derivar polinomios que puedan ser resueltos [21]. Otros métodos analizan directamente cada ecuación o realizan multiplicaciones por transformaciones homogéneas a la derecha o a la izquierda para obtener ecuaciones más fácilmente resolubles.

Métodos Numéricos Iterativos. Para cadenas cinemáticas complejas, como robots humanoides, las soluciones analíticas se vuelven tediosas, irresolubles, o de múltiples soluciones debido a la redundancia del sistema y a los términos fuertemente no lineales de la cinemática directa. En estos casos se prefiere enfoques numéricos o el uso de cinemática diferencial [6]. Los enfoques numéricos principalmente utilizados son los siguientes.

a. *Método de Newton.* Consiste en reformular el problema como $\mathbf{F}(\boldsymbol{q}) = \boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{q}) = \mathbf{0}$, donde $\boldsymbol{x}_d$ es la posición/orientación deseada, y calcular los ceros de $\mathbf{F}(q)$ usando el método de Newton multivariable. Esta solución iterativa es:

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + J^{-1}(\boldsymbol{q}_k)(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{q}_k)) \tag{2.30}$$

donde $J(\boldsymbol{q}_k) = \frac{\partial \boldsymbol{f}(\boldsymbol{q}_k)}{\partial \boldsymbol{q}}$, y $\boldsymbol{q}_k$ representa la configuración articular generalizada $\boldsymbol{q}$ para la iteración k-ésima. El criterio de término puede ser un error Cartesiano pequeño o un incremento articular pequeño. La ventaja del método es la rapidez de convergencia, que es de tipo cuadrático. En caso que la inversa J^{-1} no esté definida, se utiliza la pseudo-inversa, pero cerca a configuraciones singulares, donde J pierde rango, se puede usar la pseudo-inversa amortiguada [22].

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

b. *Método del gradiente.* Formula la cinemática inversa como un problema de optimización que busca minimizar la función $g(\boldsymbol{q}) = \frac{1}{2}(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{q}))^T(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{q}))$ utilizando el método de descenso del gradiente. La solución iterativa, en este caso, es:

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \alpha J^T(\boldsymbol{q}_k)(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{q}_k)) \tag{2.31}$$

donde α es una constante arbitraria que determina el tamaño del paso en dirección del gradiente. Este método no presenta problemas cerca a singularidades, cuando J pierde rango, ya que usa la transpuesta, pero posee una convergencia más lenta.

A pesar de que las soluciones de ambos métodos lucen bastante semejantes, se debe tener en cuenta que ambas formulaciones son conceptualmente diferentes y se obtienen de distinta manera. Al utilizar métodos numéricos, es práctica común iniciar con el método del descenso del gradiente para evitar problemas cerca a valores donde J pierde rango, y luego de unas iteraciones cambiar al método de Newton debido a la rápida convergencia de este último.

La principal desventaja del cálculo de cinemática inversa usando métodos analíticos o numéricos consiste en la dificultad de satisfacer restricciones adicionales como los límites articulares, por lo que se debe realizar verificaciones posteriores que pueden afectar la configuración hallada. Una forma de resolver estos problemas es mediante la cinemática inversa, que se basa en el uso del Jacobiano.

2.3.3. El Jacobiano

El Jacobiano tiene un rol bastante importante en robótica. Debido a que la cinemática directa es fuertemente no lineal, es práctica común linealizarla localmente utilizando el Jacobiano. A través de enfoques basados en el Jacobiano se puede además formular restricciones a la cinemática. Por otro lado, es también utilizado para mapear fuerzas externas en fuerzas articulares cuando se analiza la dinámica del robot. Existen dos tipos de Jacobianos en robótica, el analítico y el geométrico [6].

Jacobiano Analítico. Es la matriz Jacobiana $J(\boldsymbol{q})$ de derivadas parciales asociada a la función de cinemática directa $\boldsymbol{f}(\boldsymbol{q})$ que se define de manera general como:

$$J(\boldsymbol{q}) = \frac{\partial \boldsymbol{f}(\boldsymbol{q})}{\partial \boldsymbol{q}}. \tag{2.32}$$

Al Jacobiano analítico también se le conoce como el *Jacobiano de la tarea*. Por ejemplo, si la cinemática directa $\boldsymbol{f}(\boldsymbol{q})$ de un robot de n grados de libertad está representada mediante posición Cartesiana (x, y, z) y orientación dada por los ángulos de roll, pitch yaw (ϕ_r, ϕ_p, ϕ_y), tal que

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

$\boldsymbol{f}(\boldsymbol{q}) = [x \;\; y \;\; z \;\; \phi_r \;\; \phi_p \;\; \phi_y]^T$, el Jacobiano analítico es

$$J(\boldsymbol{q}) = \begin{bmatrix} \frac{\partial \boldsymbol{f}}{\partial q_1} & \frac{\partial \boldsymbol{f}}{\partial q_2} & \cdots & \frac{\partial \boldsymbol{f}}{\partial q_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \cdots & \frac{\partial x}{\partial q_n} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \cdots & \frac{\partial y}{\partial q_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \phi_y}{\partial q_1} & \frac{\partial \phi_y}{\partial q_2} & \cdots & \frac{\partial \phi_y}{\partial q_n} \end{bmatrix} \tag{2.33}$$

donde q_i representa la articulación i-ésima. El cálculo puede realizarse de manera analítica a través de las derivadas parciales de cada elemento. Alternativamente, y debido a la complejidad usual de $\boldsymbol{f}(\boldsymbol{q})$, se puede utilizar diferencias finitas para el cálculo de las derivadas. En este caso, la derivada parcial respecto al ángulo q_i será

$$\frac{\partial \boldsymbol{f}}{\partial q_i} \approx \frac{\boldsymbol{f}(q_1, \cdots, q_i + \Delta q_i, \cdots, q_n) - \boldsymbol{f}(q_1, \cdots, q_i, \cdots, q_n)}{\Delta q_i} \tag{2.34}$$

donde Δq_i es un valor bastante pequeño. Se debe notar que el Jacobiano analítico depende de la representación que se utilice para la posición y para la orientación.

Jacobiano Geométrico. Considerando el twist $\boldsymbol{\xi}$, que contiene la velocidad lineal y angular de algún punto operacional, y la velocidad del vector de configuración $\dot{\boldsymbol{q}}$, este Jacobiano $J_{_G}$ se define como

$$\boldsymbol{\xi} = J_{_G}(\boldsymbol{q})\, \dot{\boldsymbol{q}}. \tag{2.35}$$

donde $J_{_G}\colon \mathcal{T}_q(\mathcal{Q}) \to se(3)$ es una aplicación tangente y $\mathcal{T}_q(\mathcal{Q})$ representa el espacio tangente a $\mathcal{Q}$ en $\boldsymbol{q}$ El Jacobiano geométrico es también denominado *Jacobiano básico* [16] o, en algunos casos, simplemente Jacobiano. Su estructura está dada por

$$J_{_G}(\boldsymbol{q}) = \begin{bmatrix} J_{v_1}(\boldsymbol{q}) & J_{v_2}(\boldsymbol{q}) & \cdots & J_{v_n}(\boldsymbol{q}) \\ J_{\omega_1}(\boldsymbol{q}) & J_{\omega_2}(\boldsymbol{q}) & \cdots & J_{\omega_n}(\boldsymbol{q}) \end{bmatrix} = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix} \tag{2.36}$$

donde J_{v_i} corresponde al efecto que tiene la articulación q_i en la velocidad lineal, y J_{ω_i} al efecto en la velocidad angular. Para robots sin base flotante, utilizando consideraciones de propagación de velocidad, se puede demostrar que para una articulación prismática $J_{v_i} = \boldsymbol{z}_i$, $J_{\omega_i} = 0$, con $\boldsymbol{z}_i$ representando el eje de movimiento de la articulación. Por otro lado, para una articulación de revolución se tiene $J_{\omega_i} = \boldsymbol{z}_i$, $J_{v_i} = \boldsymbol{z}_i \times (\boldsymbol{p}_N - \boldsymbol{p}_i)$ donde $\boldsymbol{p}_N$, $\boldsymbol{p}_i$ representan la posición del efector final y del eslabón i, respectivamente. Todos los vectores deben estar expresados con relación al sistema de referencia base. Una explicación más detallada puede encontrarse en [6, 7]. La generalización al caso de robots con base flotante, como robots humanoides, se encuentra en la sección 3.2.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Debido a que este Jacobiano está relacionado con la velocidad lineal y angular, es independiente de la representación que se utilice para la cinemática directa y siempre será el mismo para una configuración dada de cada punto operacional.

Cambio de Sistema de Referencia. Cuando es evidente el sistema de referencia con respecto al cual se expresa el Jacobiano, se obvia su indicación. En otros casos, o como aclaración, pueden ser incluidos. Considérese el Jacobiano geométrico ${}^{A}J_{G}$ con respecto al sistema de referencia $\{\mathcal{A}\}$. La expresión de este Jacobiano con respecto a un sistema de referencia $\{\mathcal{B}\}$ está dada por [7]:

$$\begin{bmatrix} {}^{B}J_{v} \\ {}^{B}J_{\omega} \end{bmatrix} = \begin{bmatrix} {}^{B}\boldsymbol{R}_{A} & 0 \\ 0 & {}^{B}\boldsymbol{R}_{A} \end{bmatrix} \begin{bmatrix} {}^{A}J_{v} \\ {}^{A}J_{\omega} \end{bmatrix} \tag{2.37}$$

donde ${}^{B}\boldsymbol{R}_{A}$ es la matriz de rotación que relaciona al sistema $\{\mathcal{A}\}$ con el sistema $\{\mathcal{B}\}$. Al analizar término a término se puede observar que la transformación consiste básicamente en multiplicar cada parte del Jacobiano con la matriz de rotación.

2.3.4. Cinemática Diferencial

En robótica, la *cinemática diferencial* o *cinemática instantánea* hace referencia al estudio de la velocidad y aceleración de los componentes que conforman el robot sin considerar las fuerzas o torques que generan el movimiento.

Cinemática Diferencial Directa. Consiste en encontrar el twist $\boldsymbol{\xi} \in se(3)$ de un punto operacional que es producido a causa de las variaciones articulares $\dot{\boldsymbol{q}} \in \mathcal{T}_{q}(\mathcal{Q})$. Esta relación se expresa mediante (2.35) y usa el Jacobiano geométrico $J_{G}(\boldsymbol{q})$. Alternativamente, la cinemática diferencial directa puede ser formulada encontrando las variaciones de primer orden $\dot{\boldsymbol{x}}$ que son producidas por $\dot{\boldsymbol{q}}$, lo cual es expresado como

$$\dot{\boldsymbol{x}} = J(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.38}$$

donde $\boldsymbol{J}(\boldsymbol{q})$ definido en (2.32) es un mapa diferencial denominado Jacobiano analítico o de la tarea. La cinemática diferencial directa se puede considerar como una linealización instantánea de la cinemática directa en el punto $\boldsymbol{q}$. Debido a la relación lineal que la define, permite aprovechar los métodos de álgebra lineal, y las implementaciones algorítmicas eficientes y numéricamente estables, para relacionar las velocidades del efector final con las velocidades articulares e incluso para calcular la cinemática inversa.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Relación entre los Jacobianos. La relación entre las variaciones de primer orden del espacio operacional $\dot{\boldsymbol{x}}$ y el twist $\boldsymbol{\xi}$ está dada por una función $E(\boldsymbol{x})$ como $\dot{\boldsymbol{x}} = E(\boldsymbol{x})\boldsymbol{\xi}$, o más explícitamente como

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{p}}(\boldsymbol{q}) \\ \dot{\boldsymbol{o}}(\boldsymbol{q}) \end{bmatrix} = \begin{bmatrix} E_p(\boldsymbol{p}) & 0 \\ 0 & E_o(\boldsymbol{o}) \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix}. \tag{2.39}$$

donde $\dot{\boldsymbol{p}}(\boldsymbol{q})$ es la derivada de la representación de posición y $\dot{\boldsymbol{o}}(\boldsymbol{q})$ es la derivada de la representación de orientación. La matriz $E_p(\boldsymbol{p})$ relaciona las partes correspondientes a la posición y velocidad lineal, mientras que $E_o(\boldsymbol{o})$ las partes correspondientes a la orientación y velocidad angular. Si se usa coordenadas Cartesianas para representar la posición, como es usual, se tiene $E_p = I$, donde I es la matriz identidad. Para obtener E_o se puede proceder como sigue [6, 23]: (*i*) calcular la matriz de rotación equivalente a la representación de orientación $\boldsymbol{o}$; (*ii*) encontrar $\hat{\boldsymbol{\omega}} = \dot{\boldsymbol{R}}\boldsymbol{R}^T$ y obtener los componentes de la velocidad angular $\boldsymbol{\omega}$ a partir del $\hat{\boldsymbol{\omega}}$ resultante; (*iii*) reordenar los términos y operar algebraicamente para obtener $\dot{\boldsymbol{x}}_o = E_o(\boldsymbol{o})\boldsymbol{\omega}$.

Utilizando (2.39) y las definiciones de los Jacobianos, se obtiene que la relación entre el Jacobiano analítico y el Jacobiano geométrico está dada por [6]:

$$J(\boldsymbol{q}) = E(\boldsymbol{x})J_G(\boldsymbol{q}) \tag{2.40}$$

cuya estructura es

$$\begin{bmatrix} J_p(\boldsymbol{q}) \\ J_o(\boldsymbol{q}) \end{bmatrix} = \begin{bmatrix} E_p(\boldsymbol{p}) & 0 \\ 0 & E_o(\boldsymbol{o}) \end{bmatrix} \begin{bmatrix} J_v(\boldsymbol{q}) \\ J_\omega(\boldsymbol{q}) \end{bmatrix}. \tag{2.41}$$

donde J_p y J_o son la parte correspondiente a la posición y orientación del Jacobiano analítico, y J_v y J_ω son la parte correspondiente a la velocidad lineal y angular del Jacobiano geométrico.

Cinemática Diferencial Inversa. Consiste en encontrar las variaciones articulares $\dot{\boldsymbol{q}}$ que producen una variación de un punto operacional. Dado que (2.38) es una relación lineal, se realiza una inversión del Jacobiano, si éste es cuadrado y no singular. En caso que J sea singular o no sea cuadrado, se usa la inversa generalizada $J^{\#}$ como:

$$\dot{\boldsymbol{q}} = J^{\#}\dot{\boldsymbol{x}}. \tag{2.42}$$

Normalmente $J^{\#}$ es la pseudo-inversa de Moore-Penrose, pero también es común la pseudo-inversa amortiguada en los casos donde J pierde rango. Se debe notar que la inversa generalizada es igual a la inversa cuando J es invertible. Una discusión exhaustiva sobre las diversas inversas generalizadas se puede encontrar en [24, 25].

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

La cinemática diferencial inversa puede ser utilizada para calcular de manera iterativa la cinemática inversa o para generar lo que se denomina *control cinemático*. El control cinemático, introducido inicialmente en [26] como *resolved motion-rate control*, consiste en encontrar los incrementos $\delta \boldsymbol{q}$ adecuados que se usen para actualizar iterativamente el vector articular $\boldsymbol{q}_k = \boldsymbol{q}_{k-1} + \delta \boldsymbol{q}$, partiendo de una configuración inicial $\boldsymbol{q}_0$, con el fin de alcanzar un valor $\boldsymbol{x}_d$ deseado. Evidentemente el proceso se repite hasta que $\boldsymbol{x}(\boldsymbol{q})$ sea lo suficientemente cercano a $\boldsymbol{x}_d$. Algunos de los métodos variacionales clásicos utilizados en robots manipuladores son la transpuesta del Jacobiano, la pseudo-inversa, la descomposición por valores singulares, el método de mínimos cuadrados amortiguados y sus variantes. Una descripción exhaustiva de estos métodos se puede encontrar en [22]. En el caso de robots humanoides se debe además considerar la redundancia existente, lo cual complica la resolución del problema.

2.4. Redundancia en Robótica

Una de las principales diferencias entre robots manipuladores tradicionales y robots articulados móviles o robots humanoides es el número de articulaciones. Los robots manipuladores tradicionales poseen seis o menos articulaciones debido a que el espacio tridimensional posee únicamente seis grados de libertad espaciales (tres para la posición y tres para la orientación). Por otro lado, los robots redundantes poseen más de seis articulaciones y por ende pueden llegar a diversas configuraciones espaciales de infinitas maneras. Por ejemplo, los robots humanoides poseen aproximadamente 30 articulaciones, sin contar las articulaciones de los dedos. Esta redundancia es uno de los motivos de la complejidad de la generación de movimiento.

2.4.1. Redundancia Cinemática

En general, la redundancia en robótica está dividida en redundancia de actuación y redundancia cinemática [23]. La redundancia de actuación solo se presenta en mecanismos que poseen cadena cinemática cerrada, y por tanto no es de interés en la presente tesis ya que se aborda únicamente robots que poseen cadenas cinemáticas abiertas (a pesar de que el contacto con el entorno podría generar cadenas cinemáticas cerradas, pero estas nuevas cadenas no generarían redundancia de actuación). Por este motivo, esta sección se enfocará únicamente en la redundancia cinemática.

La redundancia cinemática se presenta cuando el robot posee más grados de libertad que el mínimo número requerido para ejecutar una tarea. Esto significa que la configuración articular del robot puede ser modificada manteniendo la misma posición y/o orientación del efector final. Así, la redundancia cinemática está relacionada con la tarea específica que se realizará: un robot puede ser redundante con respecto a algunas tareas y no redundante con respecto a otras tareas.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

De manera más formal, considérese una tarea i que implique m_i grados de libertad. Un robot de n grados de libertad articular se dice que es *cinemáticamente redundante* con respecto a la tarea i si $n > m_i$, y el grado de redundancia con respecto a la tarea i es $n - m_i$ [23]. Si existen k tareas simultáneas, cada una con m_i grados de libertad, el sistema robótico es aún cinemáticamente redundante si $n > \sum_{i=1}^{k} m_i$.

A pesar de que la redundancia cinemática le otorga mayor destreza y versatilidad al robot (como mayor rango articular, posibilidad de evitar obstáculos o singularidades, movimiento compliante [27] o minimización del consumo de energía), incrementa la complejidad de la cinemática inversa debido a que existe más de una solución, o soluciones infinitas, en el espacio articular para una posición y orientación deseadas en el espacio operacional. Es común que exista un interés especial en una solución particular que satisface tanto la tarea principal como algunas otras tareas complementarias de manera tan cercana como sea posible. Un ejemplo de este tipo de comportamiento es un manipulador que debe alcanzar una posición determinada evitando colisiones y con una cierta cantidad de objetos en su entorno. En este caso, no todas las soluciones matemáticas son plausibles físiscamente, y algunas restricciones deben ser consideradas.

2.4.2. Cinemática Diferencial con Redundancia

La ecuación que describe la cinemática diferencial (2.38) puede ser caracterizada en términos del espacio de columnas y del espacio nulo del Jacobiano J [6].

- El *espacio de columnas* o *rango* de J se define como $C(J) = \{\dot{\boldsymbol{x}} \mid J(\boldsymbol{q})\dot{\boldsymbol{q}} = \dot{\boldsymbol{x}}\}$ y representa las velocidades del espacio operacional $\dot{\boldsymbol{x}}$ que pueden ser generadas mediante las velocidades articulares $\dot{\boldsymbol{q}}$ en una configuración dada del robot $\boldsymbol{q}$. La dimensión del espacio de las columnas se denomina el rango del Jacobiano J y se representa como: $\rho(J) = \dim\{C(J)\}$.

- El *espacio nulo* de J se define como $N(J) = \{\dot{\boldsymbol{q}} \mid J(\boldsymbol{q})\dot{\boldsymbol{q}} = 0\}$ y representa las velocidades articulares $\dot{\boldsymbol{q}}$ que no producen velocidad alguna en el punto operacional $\dot{\boldsymbol{x}}$, para la configuración articular $\boldsymbol{q}$. La dimensión del espacio nulo se denomina nulidad y se representa como: $\nu(J) = \dim\{N(J)\}$.

La importancia del espacio de columnas y del espacio nulo consiste en caracterizar los posibles movimientos que puede realizar un robot. De todas las posibles velocidades $\dot{\boldsymbol{x}}$, solamente algunas pueden ser físicamente realizables usando $\dot{\boldsymbol{q}}$ y están caracterizadas por $C(J)$, como se esquematiza en la Figura 2.1. Además, los movimientos tales que $\dot{\boldsymbol{q}} \in N(J)$ implican $\dot{\boldsymbol{x}} = 0$ para la tarea asociada con J y por tanto no generan movimiento para el punto operacional $\boldsymbol{x}$. Esto significa que para robots con redundancia cinemática, las configuraciones $\dot{\boldsymbol{q}} \in N(J)$ pueden ser utilizadas para

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

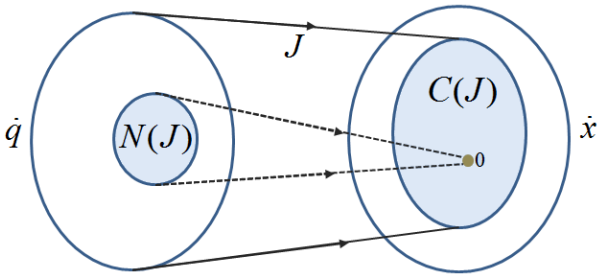


FIGURA 2.1: Relaciones entre los espacios nulo y de columnas de J (adaptado de [6]).

realizar movimientos sin interferir con un movimiento principal. Así, la existencia de un subespacio $N(J)$ permite la determinación de técnicas que se usan para manejar los grados de libertad redundantes. Considerando la redundancia, la solución general de la ecuación de cinemática diferencial está dada por

$$\dot{\boldsymbol{q}} = J^{\#}\dot{\boldsymbol{x}} + (I - J^{\#}J)\boldsymbol{z} \tag{2.43}$$

donde $J^{\#}$ es una inversa generalizada de J, $P = (I - J^{\#}J)$ es un proyector en $N(J)$, y $\boldsymbol{z}$ es un vector arbitrario de dimensiones adecuadas. Es importante notar que $P\boldsymbol{z} \in N(J)$ y, por tanto, $\boldsymbol{z}$ no tiene efecto en la velocidad para el espacio operacional de $\dot{\boldsymbol{x}}$, debido a la definición misma del espacio nulo. La función de este vector $\boldsymbol{z}$ es la de proporcionar libertad adicional para satisfacer restricciones adicionales como posturas diferentes o para satisfacer tareas con características específicas.

2.4.3. Redundancia en Robots con Base Flotante

Los robots con base flotante son aquellos que no se encuentran fijos a ninguna parte de su entorno y pueden desplazarse libremente en el mismo, como los robots humanoides, cuadrúpedos, hexápodos, etc. La principal ventaja que poseen es que pueden desplazarse sin tener la restricción de mantenerse en una sola localización espacial, pudiendo alcanzar diferentes puntos en un entorno determinado. Sin embargo, el problema que presentan es la complejidad que involucra la coordinación de su movimiento.

En el caso de robots con base flotante, adicionalmente a la redundancia tradicional, existen fuerzas en el suelo que deben ser consideradas, principalmente para un análisis dinámico. Más aún, estos robots son por lo general altamente redundantes debido a que superan por mucho los 6 grados de libertad articular. Por otro lado, la base flotante por sí misma ya posee seis grados de libertad, pero sub-actuados, ya que no es posible modificarlos directamente sino a través de la coordinación del movimiento [15].

Para el caso de robots humanoides, la redundancia es inherente. Por ejemplo, un robot humanoide puede desear mover la cabeza, los brazos, el centro de masa y sus piernas, todo al mismo tiempo.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Más aún, hay infinitas formas de realizar una tarea para un robot humanoide. Para ver esto, un robot humanoide puede tener la mano en un lugar, y al mismo tiempo puede colocar la otra mano en infinitas posiciones diferentes (dado que el espacio Cartesiano es continuo). Los diversos esquemas de generación de movimiento solucionan este problema a nivel cinemático o dinámico pero siempre considerando la redundancia y aprovechándola, aunque no es trivial determinar cuál es la mejor configuración para el caso redundante.

2.5. Generación de Movimiento

Un robot es un sistema electro-mecánico actuado por los torques que producen los motores. Sin embargo, es posible considerar un control cinemático (usualmente velocidad) como señal de entrada al sistema, debido a la presencia de control realimentado de bajo nivel en las articulaciones del robot que permiten establecer velocidades de referencia o posiciones de referencia. Estos lazos de control de bajo nivel, realimentados internamente, se encuentran presentes en robots industriales con arquitectura de control cerrada, donde los usuarios pueden solamente especificar los comandos de referencia de tipo cinemático. De esta forma, el rendimiento puede ser satisfactorio siempre que el movimiento deseado no sea muy rápido y no se requiera grandes aceleraciones. De requerirse esto, se tendría que modelar los motores del robot y realizar de alguna forma un control indirecto de los mismos.

2.5.1. Control Cinemático de Robots Humanoides.

A pesar de que la redundancia cinemática ofrece mayor dexteridad y versatilidad a los robots humanoides, como poder evitar singularidades u obstáculos, realizar movimientos compliantes, o minimizar el consumo de energía, también incrementa la complejidad del problema cinemático inverso [27]. En un robot humanoide, para una tarea determinada existen infinitas soluciones en el espacio articular; por tanto, soluciones analíticas no son en general posibles. Más aún, usualmente se desea soluciones que satisfagan a la vez la tarea principal y alguna otra tarea complementaria de forma tan cercana como sea posible. Por ejemplo, se puede desear mover el brazo derecho hacia un lugar determinado y al mismo tiempo mover el brazo izquierdo evitando colisiones con objetos del ambiente, así como girar el torso y la cabeza.

En estos casos complejos se prefiere el uso de métodos numéricos para la resolución de la cinemática. Los métodos numéricos tratan el problema de cinemática inversa como un problema de optimización y es posible añadir algunas restricciones adicionales, lo cual incrementa la aplicabilidad de la solución. Normalmente el cálculo implica hallar un cambio óptimo $\delta \boldsymbol{q}$ para un pequeño

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

cambio $\delta \boldsymbol{x}$, de donde la trayectoria en el espacio articular $\boldsymbol{q}(t)$ se obtiene integrando $\delta \boldsymbol{q}$ en tiempo real.

Los principales esquemas de control cinemático se basan en esquemas ponderación de tareas o en una jerarquía de tareas. Los esquemas basados en ponderación asocian un peso w_i a cada tarea, correspondiendo el peso más alto a la tarea más importante. Algunos ejemplos de este esquema se encuentran en [28], donde se utiliza un método extendido del Jacobiano, o [29] donde se utiliza una variación continua de pesos. Debido a la naturaleza de estos esquemas, existen problemas cuando se presentan tareas conflictivas dado que en esos casos ninguna de las tareas puede ser completamente satisfecha. Los métodos basados en prioridad de tareas resuelven estos conflictos debido a la jerarquía que tienen las tareas: las de menos prioridad se realizan solo en la medida que no interfieran con las de mayor prioridad. Algunos ejemplos de este esquema se encuentran en [30], [31] y las formas usuales consisten en proyecciones sucesivas en el espacio nulo de la tarea anterior o en el uso de programación cuadrática [32].

Control de posición. El control cinemático brinda como salida una trayectoria temporal de velocidad $\dot{\boldsymbol{q}}(t)$ la cual podría ser directamente utilizada para controlar un robot cuyo control de bajo nivel de los motores sea de velocidad. Sin embargo, la mayoría de robots humanoides, así como robots manipuladores redundantes (y robots manipuladores tradicionales), son controlados en posición; esto es, los motores tienen un lazo de control de posición (típicamente un controlador PID, o alguna variación) y se comportan como servomotores. Así, para poder controlar estos robots se requiere una configuración articular $\boldsymbol{q}(t)$. Por este motivo, la señal de control obtenida del sistema de generación de movimiento cinemático se convierte en posición mediante alguna forma de integración, siendo la más común la integración de Euler $\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \dot{\boldsymbol{q}}_k \Delta T$, donde ΔT es el período de control. Es posible utilizar métodos más precisos de integración, pero dado que ΔT es normalmente bastante pequeño (típicamente entre 1 a 10 ms), el error acumulado no es grande, y el esquema es suficientemente preciso en la práctica.

2.5.2. Otras Formas de Generación de Movimiento en Robótica

Alternativamente, las otras formas de generar movimiento para robots manipuladores o robots redundantes con o sin base flotante se basan en enfoques dinámicos, en planificación del movimiento, y en control óptimo, también denominado optimización de trayectoria en robótica debido a su naturaleza numérica.

Control Dinámico. El denominado *control dinámico*, es una forma de control en robótica que usualmente realiza una realimentación linealizante conocida como dinámica inversa con el objetivo

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

de tener un modelo lineal local, aunque esto requiere un modelo dinámico teórico muy próximo al modelo real [11]. Para el control por dinámica inversa se tiene los mismos enfoques que para el control cinemático: esquemas basados en ponderación de tareas, o esquemas basados en jerarquía de tareas [33]. El control dinámico es ventajoso para controlar directamente la fuerza que es aplicada al entorno o para controlar la fuerza y el movimiento de manera simultánea, a lo que se conoce como control híbrido. Para robots humanoides toma en cuenta, además, toda la dinámica del cuerpo y puede prevenir movimientos residuales involuntarios, principalmente debidos al momento angular de todo el cuerpo al realizar movimientos rápidos. Sin embargo, el problema tecnológico existente es que requieren controlar directamente el torque del robot, lo cual no es usualmente posible con robots industriales o cuyos controladores de bajo nivel son cerrados. En estos casos, se utiliza la aceleración (del modelo dinámico), la cual es integrada dos veces para obtener posición, conllevando errores de ruido inherentes al proceso de integración numérica real.

Planificación del Movimiento. Otra modalidad popular para generar el movimiento de un robot se denomina *planificación del movimiento* (*motion planning*, en inglés) y consiste en encontrar de manera automática un camino o trayectoria para ir de una configuración inicial a una configuración final satisfaciendo ciertas restricciones. Por ejemplo, se puede calcular la trayectoria que mueve la mano de su posición inicial a la final evitando colisiones con algún objeto en el entorno o colisiones con el robot mismo (autocolisiones). Para esto, es necesario mapear todo el espacio operacional al espacio de configuración, realizando todo el planeamiento en dicho espacio [34]. Los métodos usuales en este caso se pueden clasificar en hojas de ruta (*roadmaps*) que incluyen gráficos de visibilidad, diagramas de Voronoi, etc., algoritmos basados en descomposición de celdas, campos potenciales, o algoritmos basados en muestreo, como RRT[35], BiRRT y modificaciones. Usualmente estos métodos solamente utilizan la cinemática del robot y dependen completamente del controlador interno de bajo nivel.

Control Óptimo. Otra forma de generación de movimiento en robótica es el control óptimo, también conocido como *optimización de trayectoria*. Consiste en encontrar una trayectoria y su ley de control asociada que satisface ciertos criterios de optimalidad. En términos matemáticos, concierne las propiedades de las funciones de control, las cuales al ser insertadas en una ecuación diferencial dan soluciones que minimizan una medida de costo. La ventaja en robótica es que se puede generar movimientos muy potentes de forma casi automática pero el problema es que son actualmente computacionalmente muy costosos. Algunos ejemplos de control óptimo que se aplican junto con planificación de movimiento son CHOMP (*Covariant Hamiltonian Optimization for Motion Planning*) [36], STOMP (*Stochastic Trajectory Optimization for Motion Planing*), [37] e ITOMP

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

(*Incremental Trajectory Optimization for Real-Time Replanning in Dynamic Environments*) [38]. Métodos de control óptimo emergentes se basan en DDP (*differential dynamic programming*) e iLQR (iterative LQR) para la optimización de trayectorias pero suelen considerar modelos dinámicos simplificados debido al a complejidad computacional de la solución, que impide el control en tiempo real.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Capítulo 3

Diseño del Sistema Cinemático de Control

El sistema de generación de movimiento desarrollado en esta tesis se denomina *OSCR*, las cuales son siglas que provienen del inglés *Operational Space Control for redundant Robots* (que en español sería *control en el espacio operacional para robots redundantes*), debido a que implementa estrategias de control cinemático para robots redundantes, siendo los robots humanoides un caso especial. El sistema desarrollado se encuentra disponible de forma *open source* en https://github.com/oeramosp/oscr. Este capítulo presenta los fundamentos teóricos utilizados en este sistema, siendo el principal aporte la descripción completa de la metodología empleada, con suficiente detalle como para poder replicarla en alguna otra aplicación. Este aporte se debe a que actualmente no existen libros sobre robots redundantes de base flotante, y la información se encuentra solo en artículos científicos. Así, en la actual literatura de robótica muchos de estos conceptos se dan por sobreentendidos, y quienes intentan desarrollar algoritmos de control cinemático se encuentran con el problema de no disponer de fuentes que expliquen cómo se obtienen estos resultados, cómo se aplican, o cómo pueden ser extendidos.

3.1. Función de Tarea

El formalismo de la función de tarea [17] es un esquema que describe las tareas a ser realizadas en términos de funciones de salida específicas escogidas para facilitar la observación y control de la tarea a realizar. Esta metodología también es conocida como el enfoque del Espacio Operacional [16] y consiste en generar movimiento con base en diversas tareas cuyas leyes de control están expresadas en un subespacio de dimensión más pequeña que el espacio de estados completo del robot. Estas leyes de control son luego proyectadas de vuelta al espacio original.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Por ejemplo, si el brazo de un robot humanoide se encuentra dirigido hacia el suelo y la tarea consiste en levantar el brazo, la ley de control puede ser definida en términos de las posiciones y orientaciones espaciales inicial y final de alguna parte del brazo (como la mano), en lugar de ser definido en términos de todo el espacio de configuración. Así, la ventaja consiste en reducir la complejidad del control, hacerlo más intuitivo, y desacoplar las diferentes partes del movimiento con una clara separación de los objetivos.

Sea n el número total de grados de libertad del robot, $\mathcal{Q}$ la variedad de dimensión n que representa el espacio articular generalizado, $\mathcal{M}$ la variedad de dimensión m que representa el espacio operacional o espacio de la tarea, y $\mathcal{U}$ la variedad de dimensión p que representa el espacio de control. Para especificar una tarea cinemática de forma completa, los siguientes tres elementos deben ser definidos [39]:

1. La *función de tarea* $\boldsymbol{e} : \mathcal{Q} \to \mathcal{M}$, la cual describe la tarea.

2. El *comportamiento de referencia* $\boldsymbol{e}^* \in \mathcal{R}$ que es un comportamiento deseado para alcanzar adecuadamente la tarea. En general, $\mathcal{R}$ es una variedad de dimensión m relacionada a $\mathcal{M}$, y en cinemática es la velocidad deseada para la tarea.

3. El *mapa* $\mathcal{J} : \mathcal{U} \to \mathcal{R}$, el cual relaciona a la tarea, en un cierto campo vectorial $\dot{\boldsymbol{e}} \in \mathcal{R}$, con la entrada de control $\boldsymbol{u} \in \mathcal{U}$ como $\dot{\boldsymbol{e}} = J\boldsymbol{u}$. Este mapa está dado por el Jacobiano de la tarea $J(q) = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{q}} \in \mathbb{R}^{m \times n}$, y el control está dado directamente por las velocidades articulares tal que $\mathcal{U} = \mathcal{T}_p(\mathcal{Q})$ o, equivalentemente, $\boldsymbol{u} = \dot{\boldsymbol{q}}$.

En el caso cinemático, la función de tarea está representada por algún tipo de error definido en el espacio operacional, como por ejemplo la diferencia entre la posición actual y la posición deseada. El comportamiento de referencia, en este caso, es definido buscando que el error tienda a cero, a modo de un problema de regulación, y la señal de control en cinemática es la velocidad articular.

3.2. Jacobiano de un Robot Humanoide

Una característica distintiva de un robot humanoide, o en general de un robot móvil articulado, es el poseer una base flotante que se puede desplazar libremente en el espacio tridimensional. Como se mencionó en la sección 2.2.1, la posición y orientación de esta base flotante con respecto a un sistema inercial están dadas por (2.21). En esta tesis se representa la posición de la base mediante coordenadas Cartesianas $\boldsymbol{p}_b \in \mathbb{R}^3$ y la orientación de la base mediante un cuaternión $\boldsymbol{Q}_b \in \mathbb{H}$. De

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

manera explícita, la posición y orientación de la base flotante quedarán representadas por

$$\boldsymbol{x}_b = \begin{bmatrix} \boldsymbol{p}_b \\ \boldsymbol{Q}_b \end{bmatrix} \tag{3.1}$$

con respecto a un sistema inercial $\{\mathcal{I}\}$. Normalmente el sistema inercial es un punto fijo que inicialmente está entre ambos pies del robot. Para representar la orientación se utilizó un cuaternión debido a que los cuaterniones no presentan singularidades de configuración; es decir, se encuentran siempre bien definidos para cualquier orientación.

La libertad espacial que ofrece la base flotante debido a su movimiento libre puede ser modelada como una articulación de 6 grados de libertad en el espacio; es decir, una articulación que no posee restricciones de movimiento. Como cualquier otra articulación, el movimiento de esta nueva articulación (de la base) contribuye al movimiento total del robot, y por ende al movimiento de cada uno de los estabones constituyentes. Así, el twist de la base flotante afectará al twist de cada eslabón. Como consecuencia, el Jacobiano tradicional de un manipulador, que se obtiene con base con respecto a un sistema de referencia fijo, deberá poseer un término adicional que represente dicho efecto.

3.2.1. Efecto del Twist de la Base Flotante

Se representará el twist de la base flotante como $\boldsymbol{\xi}_b = \begin{bmatrix} \boldsymbol{v}_b & \boldsymbol{\omega}_b \end{bmatrix}^T$ donde $\boldsymbol{v}_b$ y $\boldsymbol{\omega}_b$ son la velocidad lineal y angular de la base flotante, respectivamente, como se mencionó en la sección 2.2.1. El movimiento de la base flotante genera un movimiento en cada uno de los eslabones del robot. Considérese el eslabón i-ésimo cuya posición, con respecto a un sistema de referencia inercial, está dada por $\boldsymbol{p}_i \in \mathbb{R}^3$, y cuya velocidad lineal debida al movimiento de la base flotante es $\boldsymbol{v}_i \in \mathbb{R}^3$. Utilizando relaciones básicas de velocidad [10] y suponiendo que ningún otro eslabón del robot tiene movimiento, la velocidad del eslabón i estará dada por

$$\begin{aligned} \boldsymbol{v}_i &= \boldsymbol{v}_b + \boldsymbol{\omega}_b \times (\boldsymbol{p}_i - \boldsymbol{p}_b) \\ &= \boldsymbol{v}_b + (\boldsymbol{p}_b - \boldsymbol{p}_i) \times \boldsymbol{\omega}_b \\ &= \boldsymbol{v}_b + (\boldsymbol{p}_b - \boldsymbol{p}_i)^\wedge \boldsymbol{\omega}_b \end{aligned} \tag{3.2}$$

donde se ha utilizado la siguiente propiedad elemental del producto vectorial: $\boldsymbol{a} \times \boldsymbol{b} = -\boldsymbol{b} \times \boldsymbol{a}$, con $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$. Notar que la notación $.^\wedge$ representa la matriz antisimétrica asociada con un vector, y definida en (2.9), cuya relación con el producto vectorial es: $\boldsymbol{a} \times \boldsymbol{b} = \hat{\boldsymbol{a}}\boldsymbol{b} = (\boldsymbol{a}^\wedge)\, \boldsymbol{b}$. De igual manera, suponiendo que ningún otro eslabón tiene movimiento, la velocidad angular $\boldsymbol{\omega}_i$ del eslabón

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

i debida únicamente a la velocidad de la base estará dada por

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_b \tag{3.3}$$

debido a que todos los puntos de un mismo cuerpo rígido presentan la misma velocidad angular [7]. En este caso el sistema es equivalente a un gran cuerpo rígido dado que ningún otro eslabón o articulación posee movimiento. Utilizando ambas expresiones anteriores, el efecto del twist de la base flotante sobre el twist del eslabón i, representado por $\boldsymbol{\xi}_i = \begin{bmatrix} \boldsymbol{v}_i & \boldsymbol{\omega}_i \end{bmatrix}^T$, es

$$\boldsymbol{\xi}_i = \begin{bmatrix} I & (\boldsymbol{p}_b - \boldsymbol{p}_i)^\wedge \\ 0 & I \end{bmatrix} \boldsymbol{\xi}_b \tag{3.4}$$

donde I es la matriz identidad de 3×3. Un resultado similar se muestra en [40].

3.2.2. Relación entre Velocidades de la Base Flotante

Una forma de representar la velocidad de la base flotante es mediante la razón de cambio de su posición y orientación. Dado que en esta tesis se representa la orientación de la base flotante mediante un cuaternión $\boldsymbol{Q}_b$, la razón de cambio será

$$\dot{\boldsymbol{x}}_b = \begin{bmatrix} \dot{\boldsymbol{p}}_b \\ \dot{\boldsymbol{Q}}_b \end{bmatrix}. \tag{3.5}$$

La señal de control en un esquema de control cinemático es $\dot{\boldsymbol{q}} = (\dot{\boldsymbol{x}}_b, \dot{\boldsymbol{q}}_a)$, como se describe en la sección 3.3.2, y por tanto, $\dot{\boldsymbol{x}}_b$ es obtenido de forma directa usando este esquema. Sin embargo, el efecto del movimiento de la base en el movimiento de cada eslabón se obtuvo en función de $\boldsymbol{\xi}_b$ en (3.4) y, por tanto, es necesario relacionar $\dot{\boldsymbol{x}}_b$ con $\boldsymbol{\xi}_b$. Una forma de dicha relación está dada por (2.39) para una orientación genérica. La relación opuesta se obtiene usando la inversa de $E(\boldsymbol{x}_b)$ como $\boldsymbol{\xi}_b = E(\boldsymbol{x}_b)^{-1}\dot{\boldsymbol{x}}_b$ o, equivalentemente, como

$$\boldsymbol{\xi}_b = \begin{bmatrix} \boldsymbol{v}_b \\ \boldsymbol{\omega}_b \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & T(\boldsymbol{Q}_b) \end{bmatrix} \dot{\boldsymbol{x}}_b \tag{3.6}$$

donde $T(\boldsymbol{Q}_b) = E_o(\boldsymbol{Q}_b)^{-1}$, y $E_p(\boldsymbol{p}_b)^{-1} = E_p(\boldsymbol{p}_b) = I$ debido a que se representa la posición utilizando coordenadas Cartesianas. Notar que la matriz T relaciona la velocidad angular con la derivada del cuaternión como $\boldsymbol{\omega}_b = T(\boldsymbol{Q}_b)\dot{\boldsymbol{Q}}_b$. Por tanto, está íntimamente ligada a la estructura del cuaternión.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Cálculo de T para cuaterniones. Se puede obtener la matriz T a partir del análisis de la orientación, que en este caso es representada mediante un cuaternión. Considérese que el cuaternión unitario $\boldsymbol{Q}$ está dado por $\boldsymbol{Q} = (w, \boldsymbol{\epsilon}) = (w, \epsilon_x, \epsilon_y, \epsilon_z)$, donde w es la parte escalar y $\boldsymbol{\epsilon} = (\epsilon_x, \epsilon_y, \epsilon_z)$ es la parte vectorial. Utilizando las transformaciones entre representaciones de orientación, dado un cuaternión cualquiera es posible encontrar una matriz de rotación que represente exactamente la misma orientación. Esta matriz de rotación se expresa en función del cuaternión como $\boldsymbol{R}(w, \boldsymbol{\epsilon})$ y su expresión término a término está dada por (2.15).

Haciendo uso de la regla de la cadena, la derivada temporal de esta matriz de rotación, en función de las derivadas temporales de cada elemento del cuaternión, es

$$\frac{d}{dt}\boldsymbol{R}(w, \boldsymbol{\epsilon}) = \frac{\partial \boldsymbol{R}(w, \boldsymbol{\epsilon})}{\partial w}\dot{w} + \frac{\partial \boldsymbol{R}(w, \boldsymbol{\epsilon})}{\partial \boldsymbol{\epsilon}}\dot{\boldsymbol{\epsilon}} \tag{3.7}$$

o, de manera más explícita:

$$\frac{d}{dt}\boldsymbol{R}(w, \boldsymbol{\epsilon}) = \frac{\partial \boldsymbol{R}(w, \boldsymbol{\epsilon})}{\partial w}\dot{w} + \frac{\partial \boldsymbol{R}(w, \boldsymbol{\epsilon})}{\partial \epsilon_x}\dot{\epsilon}_x + \frac{\partial \boldsymbol{R}(w, \boldsymbol{\epsilon})}{\partial \epsilon_y}\dot{\epsilon}_y + \frac{\partial \boldsymbol{R}(w, \boldsymbol{\epsilon})}{\partial \epsilon_z}\dot{\epsilon}_z. \tag{3.8}$$

Utilizando (2.15) se puede obtener la expresión analítica de cada uno de los términos anteriores. Por otro lado, la relación entre la matriz de rotación y la velocidad angular está dada por (2.10) y es $\hat{\boldsymbol{\omega}} = \dot{\boldsymbol{R}}\boldsymbol{R}^T$. Por lo tanto, se debe calcular el producto de la derivada de la matriz de rotación con su transpuesta. Para simplificar este producto, las dos siguientes propiedades relacionadas con los elementos del cuaternión son útiles:

$$w^2 + \epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 = 1 \tag{3.9}$$

$$w\dot{w} + \epsilon_x\dot{\epsilon}_x + \epsilon_y\dot{\epsilon}_y + \epsilon_z\dot{\epsilon}_z = 0. \tag{3.10}$$

La primera propiedad se desprende del hecho que el cuaternión es unitario, y la segunda propiedad es simplemente la derivada temporal de la primera. Simplificando la expresión de $\dot{\boldsymbol{R}}\boldsymbol{R}^T$, a través del uso de propiedades trigonométricas y estas dos propiedades adicionales, se llega a

$$\hat{\boldsymbol{\omega}} = 2\begin{bmatrix} 0 & \dot{w}\epsilon_z + \dot{\epsilon}_x\epsilon_y - \dot{\epsilon}_y\epsilon_x - \dot{\epsilon}_z w & -\dot{w}\epsilon_y + \dot{\epsilon}_x\epsilon_z + \dot{\epsilon}_y w - \dot{\epsilon}_z\epsilon_x \\ -\dot{w}\epsilon_z - \dot{\epsilon}_x\epsilon_y + \dot{\epsilon}_y\epsilon_x + \dot{\epsilon}_z w & 0 & \dot{w}\epsilon_x - \dot{\epsilon}_x w + \dot{\epsilon}_y\epsilon_z - \dot{\epsilon}_z\epsilon_y \\ \dot{w}\epsilon_y - \dot{\epsilon}_x\epsilon_z - \dot{\epsilon}_y w + \dot{\epsilon}_z\epsilon_x & -\dot{w}\epsilon_x + \dot{\epsilon}_x w - \dot{\epsilon}_y\epsilon_z + \dot{\epsilon}_z\epsilon_y & 0 \end{bmatrix}$$

Se puede observar que esta matriz es antisimétrica ($\hat{\boldsymbol{\omega}} = -\hat{\boldsymbol{\omega}}^T$), y además cada uno de sus términos representa un componente de velocidad angular. Usando la definición (2.9), se iguala los elementos

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

de la matriz antisimétrica correspondientes a la velocidad angular llegando a

$$\omega_x = 2(-\dot{w}\epsilon_x + \dot{\epsilon}_x w - \dot{\epsilon}_y \epsilon_z + \dot{\epsilon}_z \epsilon_y) \tag{3.11}$$

$$\omega_y = 2(-\dot{w}\epsilon_y + \dot{\epsilon}_x \epsilon_z + \dot{\epsilon}_y w - \dot{\epsilon}_z \epsilon_x) \tag{3.12}$$

$$\omega_z = 2(-\dot{w}\epsilon_z - \dot{\epsilon}_x \epsilon_y + \dot{\epsilon}_y \epsilon_x + \dot{\epsilon}_z w). \tag{3.13}$$

Finalmente, es posible reordenar los términos de la expresión anterior para escribirla de forma matricial como

$$\boldsymbol{\omega} = T(\boldsymbol{Q})\dot{\boldsymbol{Q}} \tag{3.14}$$

con $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)$ y $\dot{\boldsymbol{Q}} = (\dot{w}, \dot{\epsilon}_x, \dot{\epsilon}_y, \dot{\epsilon}_z)$, donde

$$T(\boldsymbol{Q}) = 2\begin{bmatrix} -\epsilon_x & w & -\epsilon_z & \epsilon_y \\ -\epsilon_y & \epsilon_z & w & -\epsilon_x \\ -\epsilon_z & -\epsilon_y & \epsilon_x & w \end{bmatrix}. \tag{3.15}$$

Así, la velocidad angular y la derivada del cuaternión de la base de un robot humanoide quedan relacionadas mediante (3.15), siempre que se utilice el cuaternión de la base $\boldsymbol{Q}_b$ como argumento de T. Un resultado similar a (3.15), pero sin demostración, se puede encontrar en [41].

3.2.3. Jacobiano Geométrico de un Robot Humanoide

El efecto de la velocidad de la base flotante del robot, expresada como la razón de cambio de la posición y orientación de la base, sobre la velocidad lineal y angular de algún eslabón del robot se obtiene reemplazando $\boldsymbol{\xi}_b$ de (3.6) en (3.4) y es igual a [42]:

$$\boldsymbol{\xi}_i = \begin{bmatrix} I & (\boldsymbol{p}_b - \boldsymbol{p}_i)^\wedge T(\boldsymbol{Q}_b) \\ 0 & T(\boldsymbol{Q}_b) \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{p}}_b \\ \dot{\boldsymbol{Q}}_b \end{bmatrix}. \tag{3.16}$$

Jacobiano de la base. El término $\dot{\boldsymbol{x}}_b = (\dot{\boldsymbol{p}}_b, \dot{\boldsymbol{Q}}_b)$ es parte de la razón de cambio del vector de configuración articular generalizada $\dot{\boldsymbol{q}} = (\dot{\boldsymbol{x}}_b, \dot{\boldsymbol{q}}_a)$ y por tanto la expresión anterior relaciona la derivada de parte de la configuración articular $\dot{\boldsymbol{x}}_b$ con el twist $\boldsymbol{\xi}_i$. Se puede ver que esta relación es semejante a la definición del Jacobiano geométrico dada en (2.35). Dado que la expresión anterior solo considera la velocidad correspondiente a la base del robot, se utiliza para definir al *Jacobiano geométrico de la base flotante* del robot, J_b, como

$$\boldsymbol{\xi}_i = J_b\, \dot{\boldsymbol{x}}_b \tag{3.17}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

donde

$$J_b = \begin{bmatrix} I & (\boldsymbol{p}_b - \boldsymbol{p}_i)^\wedge \, T(\boldsymbol{Q}_b) \\ 0 & T(\boldsymbol{Q}_b) \end{bmatrix}. \tag{3.18}$$

Cada uno de los elementos de J_b está expresado con respecto a un sistema inercial $\{\mathcal{I}\}$.

Jacobiano geométrico completo. Las coordenadas generalizadas de un robot humanoide están dadas por (2.23), y en esta tesis la orientación de la base se representa mediante un cuaternión. Así, los componentes de estas coordenadas, y sus derivadas temporales son

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{p}_b \\ \boldsymbol{Q}_b \\ \boldsymbol{q}_a \end{bmatrix} \quad \text{y} \quad \dot{\boldsymbol{q}} = \begin{bmatrix} \dot{\boldsymbol{p}}_b \\ \dot{\boldsymbol{Q}}_b \\ \dot{\boldsymbol{q}}_a \end{bmatrix}. \tag{3.19}$$

Considérese que el eslabón i tiene como Jacobiano geométrico a $\begin{bmatrix} J_v(\boldsymbol{q}_a) & J_w(\boldsymbol{q}_a) \end{bmatrix}^T$, expresado con respecto al sistema de la base del robot $\{\mathcal{B}\}$, el cual se asocia solamente con las coordenadas articulares actuadas $\boldsymbol{q}_a$. Usando (3.18), para un robot humanoide, la relación del twist del eslabón i con la razón de cambio de las coordenadas generalizadas (3.19), está dada por

$$\boldsymbol{\xi}_i = J_G(\boldsymbol{q})\, \dot{\boldsymbol{q}} \tag{3.20}$$

donde

$$J_G(\boldsymbol{q}) = \begin{bmatrix} I & (\boldsymbol{p}_b - \boldsymbol{p}_i)^\wedge \, T(\boldsymbol{Q}_b) & \boldsymbol{R}_b(\boldsymbol{Q}_b)\, {}^bJ_v(\boldsymbol{q}_a) \\ 0 & T(\boldsymbol{Q}_b) & \boldsymbol{R}_b(\boldsymbol{Q}_b)\, {}^bJ_\omega(\boldsymbol{q}_a) \end{bmatrix} = \begin{bmatrix} J_v(\boldsymbol{q}) \\ J_\omega(\boldsymbol{q}) \end{bmatrix} \tag{3.21}$$

es el *Jacobiano geométrico completo* del robot humanoide, y $\boldsymbol{R}_b$ es la matriz de rotación que relaciona la base del robot $\{\mathcal{B}\}$ con el sistema inercial $\{\mathcal{I}\}$. Dicha matriz de rotación es necesaria dado que los Jacobianos correspondientes a los valores actuados se expresan con respecto a la base del robot. La expresión (3.21) es válida, en general, para cualquier robot articulado con base flotante.

3.3. Generación Cinemática de Movimiento

El movimiento de un robot, con configuración articular generalizada $\boldsymbol{q}$ y con n grados de libertad, puede ser especificado mediante un conjunto de tareas, las cuales quedan definidas por funciones de tareas [17] como se describió en la sección 3.1. El objetivo del control cinemático consiste en obtener la velocidad articular $\dot{\boldsymbol{q}}$ necesaria para satisfacer una tarea o un conjunto de tareas.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

3.3.1. Componentes Cinemáticos

Como se describió en la sección 3.1, una tarea cinemática está completamente definida por la función de tarea, el comportamiento de referencia, y el mapa equivalente al Jacobiano. En esta sección se brinda más detalles sobre estos componentes.

Función de Tarea. De manera concreta una función de tarea $\boldsymbol{e}$, a veces denominada simplemente tarea, se define como la diferencia entre la configuración actual $\mathbf{s}$ y la configuración deseada $\mathbf{s}_d$:

$$\boldsymbol{e} = \mathbf{s} - \mathbf{s}_d \tag{3.22}$$

donde $\mathbf{s}$ y $\mathbf{s}_d$ pertenecen a un mismo espacio operacional. Dicho de otro modo, una función de tarea $\boldsymbol{e}$ se define como un *error* en el espacio operacional [39]. Por ejemplo, $\boldsymbol{s}$ puede representar una posición en el espacio: $\mathbf{s} = \boldsymbol{x} \in \mathbb{R}^3$. Casos específicos de funciones de tarea se presentan en la sección 3.5. La derivada temporal de la función de tarea queda representada de manera genérica como $\dot{\boldsymbol{e}} = \dot{\boldsymbol{s}} - \dot{\boldsymbol{s}}_d$. Sin embargo, es usual que el comportamiento de referencia sea constante, de tal modo que la expresión anterior se simplifica a

$$\dot{\boldsymbol{e}} = \dot{\boldsymbol{s}}. \tag{3.23}$$

En el caso de realizar seguimiento de trayectoria, la referencia no es constante; sin embargo, en la práctica esta aparente limitación no genera restricciones importantes debido a que la actualización del ciclo de control se realiza a intervalos muy pequeños, típicamente entre 1 ms a 10 ms.

Jacobiano de la Función de Tarea. Debido a que $\boldsymbol{s}$ representa una configuración en un espacio operacional, la relación existente entre $\dot{s}$ y $\dot{q}$ está dada por el Jacobiano de la tarea J, también denominado Jacobiano analítico. Como se mencionó anteriormente, al asumir una referencia constante se tiene $\dot{\boldsymbol{e}} = \dot{\boldsymbol{s}}$ y por tanto

$$\dot{\boldsymbol{e}} = J(\boldsymbol{q})\, \dot{\boldsymbol{q}} \tag{3.24}$$

donde $\dot{\boldsymbol{q}}$ es la velocidad temporal de la configuración articular del robot. Por tanto, el Jacobiano articular es el Jacobiano que se utiliza para la razón de cambia del error de la tarea.

Comportamiento de Referencia. Al generar movimiento cinemático se desea reducir la función de error, de tal manera que la configuración actual alcance la configuración deseada. Esto es equivalente a un problema de regulación, donde se desea que el error tienda a cero: $\boldsymbol{e} \to \mathbf{0}$. Una forma de resolver el problema de regulación en el caso cinemático consiste en elegir el comportamiento

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

de referencia como

$$\dot{\boldsymbol{e}}^* = -\lambda \boldsymbol{e} \tag{3.25}$$

donde $\boldsymbol{e}$ está definida en (3.22), y $\lambda > 0$, llamada *ganancia cinemática*, garantiza estabilidad asintótica y, en particular, una convergencia exponencial a cero [43]. Esto se puede demostrar resolviendo la ecuación diferencial (3.25) ya que la solución temporal, considerando la condición inicial $\boldsymbol{e}(0)$, tiene la forma $\boldsymbol{e}(t) = \boldsymbol{e}(0)e^{-\lambda t}$, la cual contiene una exponencial que tiende a cero cuando t tiende a infinito, si $\lambda > 0$. Este comportamiento se representa en la Figura 3.1 para diferentes valores de λ. Si la ganancia λ se incrementa, la tarea será alcanzada con mayor rapidez, como se observa en la figura, pero las restricciones mecánicas del robot impiden el uso de ganancias muy altas, ya que se podría generar saturaciones indeseadas.

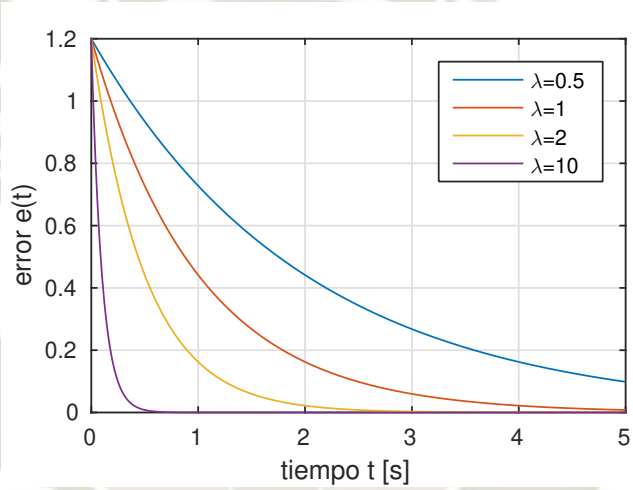


FIGURA 3.1: Solución de (3.25) para varios valores de λ (elaboración propia). A medida que λ (la ganancia cinemática) se incrementa, el error tiende más rápido a cero.

La ganancia cinemática en (3.25) puede ser constante o puede variar con el error. Si es constante, al haber un error pequeño, la acción de control será también muy pequeña y la convergencia será lenta. Si la ganancia varía con el error se puede elegir de tal modo que se incremente cuando el error es menor. Así, la ganancia se puede definir como

$$\lambda = (\lambda_{max} - \lambda_{min})e^{-k\|\boldsymbol{s}-\boldsymbol{s}_d\|} + \lambda_{min} \tag{3.26}$$

donde $k > 0$ es la razón de descenso de la ganancia, λ_{max} es la ganancia máxima y λ_{min} es la ganancia mínima. En este caso, la ganancia se incrementa a medida que el error decrece, alcanzando una convergencia más rápida. Este efecto se muestra en la Figura 3.2, donde se usó $\lambda_{max} = 0.7$, $\lambda_{min} = 0.1$ y 4 diferentes ganancias con valores $k = \{2, 5, 10, 20\}$. En la figura se observa claramente los valores máximos y mínimos de ganancia, y que a medida que el error disminuye, la

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

ganancia aumenta. Esta ganancia variable evita además el uso de valores muy altos al inicio cuando el error es grande, de tal modo que el movimiento sea más suave.

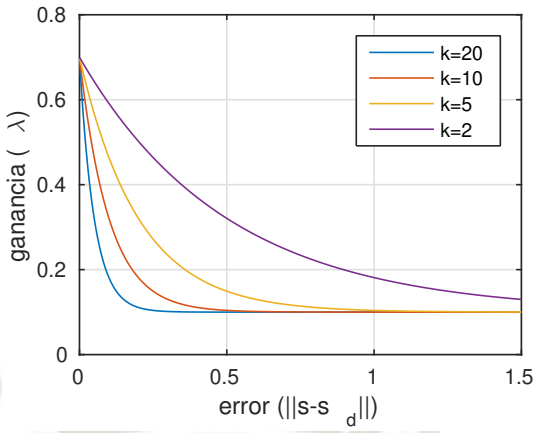


FIGURA 3.2: Ejemplo de variación de la ganancia con el error, según (3.26) (elaboración propia). A medida que el error disminuye, el valor de la ganancia cinemática aumenta, y a medida que el parámetro k disminuye, se tiene una variación de λ más suave.

3.3.2. Control Cinemático

El control cinemático consiste en generar movimiento para el robot a partir de consideraciones puramente cinemáticas, sin considerar ni la dinámica del robot ni la dinámica de los actuadores (motores) del robot. Este control asume que cada actuador tiene un lazo de control de bajo nivel independiente que lo convierte en un servomotor capaz de realizar desplazamientos angulares de manera precisa. La mayoría de robots industriales y de investigación funcionan bajo este principio y reciben posiciones articulares como entradas al sistema. Bajo esta suposición, el control cinemático genera configuraciones articulares variantes en el tiempo $\boldsymbol{q}_a(t)$ que son enviadas al robot.

Con una tarea. A partir de la inversión de (3.24), utilizando (3.25) como comportamiento de referencia para $\dot{\boldsymbol{e}}$, se puede obtener el valor de aceleración articular $\dot{\boldsymbol{q}}$. Esta inversión, para una sola tarea, es

$$\dot{\boldsymbol{q}} = J(\boldsymbol{q})^{\#}\, \dot{\boldsymbol{e}}^{*} \tag{3.27}$$

donde $\dot{\boldsymbol{e}}^{*}$ es el comportamiento de referencia, y $J(\boldsymbol{q})^{\#}$ representa una pseudo-inversa o inversa generalizada de $J(\boldsymbol{q})$, y puede verificarse en [24]. Este resultado puede obtenerse analíticamente a partir de la solución de un problema de optimización que minimiza la norma $\|\dot{\boldsymbol{e}} - J\dot{\boldsymbol{q}}\|_Q$. Cuando $Q = 2$, se tiene la norma L-2 y la solución es la pseudoinversa de Moore-Penrose. La norma L-2 es ampliamente utilizada en robótica y será la única que se considerará en esta tesis.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Con dos tareas. Cuando existen más grados de libertad que los necesarios para satisfacer una tarea, como en el caso de un robot redundante, se puede realizar por lo menos una tarea más de manera simultánea. En este caso, para satisfacer la segunda tarea, se puede aprovechar el espacio nulo del Jacobiano como se muestra en (2.43). Así, se tiene la solución más genérica siguiente

$$\dot{\boldsymbol{q}}^* = J(\boldsymbol{q})^{\#}\dot{\boldsymbol{e}}^* + P_J\,\bar{\dot{\boldsymbol{q}}} \tag{3.28}$$

donde $P_J = I - J^{\#}J$ es un proyector en el espacio nulo de $J(\boldsymbol{q})$, y $\bar{\dot{\boldsymbol{q}}}$ es un vector arbitrario [27]. Debido a que $\bar{\dot{\boldsymbol{q}}}$ es proyectado en el espacio nulo de $J(\boldsymbol{q})$, puede ser utilizado para realizar la segunda tarea sin interferir con la tarea actual $\boldsymbol{e}$. Este esquema de proyección en el espacio nulo de la pseudo-inversa del Jacobiano no es el único existente, pero brinda una idea clara de que existe aún la posibilidad de ejecutar una tarea más además de la primera tarea.

Con más tareas. Si existen más de dos tareas, y los grados de libertad suficientes para alcanzarlas, se puede extender el concepto de proyecciones sobre los espacios nulos de las tareas precedentes, como se mostró para dos tareas. Sin embargo, es posible desarrollar otros esquemas de obtención del vector de velocidad articular $\dot{\boldsymbol{q}}$ final, como se muestra en la sección 3.4. Estos esquemas para más de dos tareas constituyen una parte central de esta tesis.

Esquema General de Control Cinemático. Sea que exista una tarea o varias tareas, lo que el control cinemático obtiene de manera directa es el vector de velocidad articular $\dot{\boldsymbol{q}}$. Sin embargo, para un robot controlado en posición se requiere el vector articular $\boldsymbol{q}$, por lo que es necesario integrar $\dot{\boldsymbol{q}}$. La forma en la que se realiza esta integración se describe con más detalle en la sección 3.3.3. Considerando esta integración, el sistema de control por cinemática inversa en el espacio operacional usado en esta tesis, basado en [23, 43, 44], se muestra en la Figura 3.3, asumiendo un robot controlado en posición.

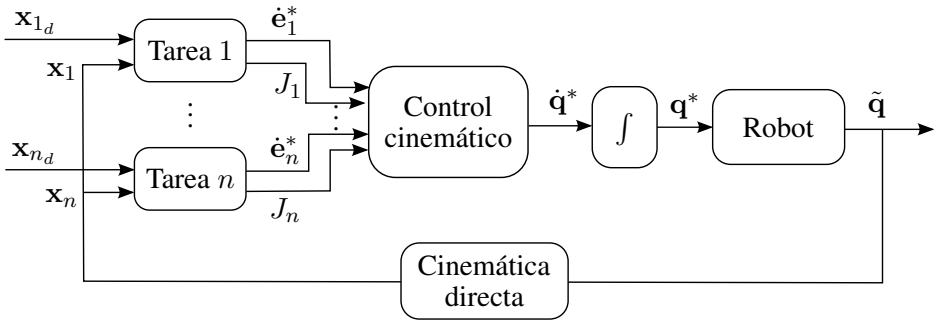


FIGURA 3.3: Esquema general del sistema de control cinemático implementado en esta tesis, a nivel conceptual (elaboración propia).

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Se puede observar que el sistema acepta como entradas varias tareas cinemáticas, para las cuales se define el valor de referencia $\boldsymbol{x}_i^*$. Cada tarea brinda el comportamiento de referencia $\dot{\boldsymbol{e}}_i^*$ y el Jacobiano de la tarea J_i al sistema de control cinemático utilizado (ver sección 3.3.3 para más detalles), el cual calcula el vector de velocidades articulares generalizadas $\dot{\boldsymbol{q}}^*$ que es la señal de control. Esta velocidad luego es integrada para obtener el comando $\boldsymbol{q}_a^*$ que será ingresado al robot controlado en posición. Si el robot es controlado en velocidad, entonces no existe la necesidad de integrar. El vector articular sensado $\tilde{\boldsymbol{q}}$, que se obtiene de un robot real o de un robot simulado, luego se utiliza para determinar el valor actual del espacio operacional de cada tarea y así actualizar cada tarea, repitiéndose el ciclo de control.

En la Figura 3.3, aparece un único bloque de cinemática directa en la realimentación del sistema únicamente con fines ilustrativos. En la implementación real desarrollada en esta tesis, no existe un único el bloque de cinemática directa, sino que este bloque se encuentra embebido en cada tarea. La razón es que cada tarea especifica un punto operacional diferente (por ejemplo, mano derecha o mano izquierda), y por tanto requiere una cinemática diferente. No sería eficiente calcular la cinemática para todos los puntos operacionales, cuando algunas tareas solo utilizan algunos de estos puntos.

3.3.3. Integración de la Velocidad Articular Generalizada

Como se ha mencionado, es necesario integrar la velocidad articular generalizada $\dot{\boldsymbol{q}}$, dada por (2.23), para obtener $\boldsymbol{q}$ y de allí el vector de configuración articular $\boldsymbol{q}_a$ que es enviado como señal de comando al robot. Esta integración tiene dos partes: por un lado se integra la velocidad articular $\dot{\boldsymbol{q}}_a$, y por otro lado se integra la velocidad de la base flotante $\dot{\boldsymbol{x}}_b$. El motivo para esta separación es que la velocidad de la base flotante, dada por (3.5), contiene la razón de cambio del cuaternión de la base $\dot{\boldsymbol{Q}}_b$, y se debe tener especial cuidado al integrar cuaterniones ya que siempre deben ser unitarios para ser descriptores válidos de rotación.

Integración de la Velocidad Articular. La velocidad articular $\dot{\boldsymbol{q}}_a$ puede ser integrada directamente para obtener $\boldsymbol{q}_a$ utilizando la integración de Euler como

$$\boldsymbol{q}_a[k+1] = \boldsymbol{q}_a[k] + \Delta T\, \dot{\boldsymbol{q}}_a[k] \tag{3.29}$$

donde $\boldsymbol{q}_a[k]$ representa el valor de $\boldsymbol{q}_a$ en el instante k, y ΔT es el tiempo de control. Este tiempo típicamente va de 1 ms a 10 ms, y por tanto es suficientemente pequeño como para que esta forma simple de integración no presente notables desviaciones del valor adecuado. La salida obtenida con (3.29) es directamente enviada a las articulaciones del robot controlado en posición.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Integración de la Base Flotante. La parte correspondiente a la posición de la base flotante puede ser integrada utilizando un esquema semejante al mostrado en (3.29) reemplazando $\boldsymbol{q}_a$ por $\boldsymbol{x}_b$, y $\dot{\boldsymbol{q}}_a$ por $\dot{\boldsymbol{x}}_b$. Sin embargo, este esquema de integración no se puede aplicar directamente al cuaternión de la base, dado que si se aplica una simple integración de Euler, el elemento resultante muy probablemente no será un cuaternión unitario y por tanto no representará una orientación válida. Una alternativa de solución consiste en utilizar la representación eje/ángulo como paso intermedio. Para determinar esta representación, la derivada temporal del cuaternión $\dot{\boldsymbol{Q}}_b$ se convierte primero en un vector de velocidad angular $\boldsymbol{\omega}$ utilizando (3.14) como

$$\boldsymbol{\omega} = T(\boldsymbol{Q}_b)\, \dot{\boldsymbol{Q}}_b \tag{3.30}$$

donde $T(\boldsymbol{Q}_b)$ está dado por (3.15). Con base en [6], el vector de velocidad angular obtenido puede utilizarse para calcular la representación eje/ángulo. El eje de giro $\boldsymbol{u}$ queda así determinado por la dirección unitaria que describe $\boldsymbol{\omega}$ y el ángulo $\Delta\theta$ por la magnitud de la velocidad angular; es decir:

$$\Delta\theta = \|\boldsymbol{\omega}\| \qquad \text{y} \qquad \boldsymbol{u} = \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}. \tag{3.31}$$

La representación eje/ángulo así obtenida se utiliza para generar un nuevo cuaternión $\Delta\boldsymbol{Q}$ que representará el incremento de la rotación como

$$\Delta\boldsymbol{Q} = \left(\cos\left(\frac{\Delta\theta\Delta T}{2}\right), \boldsymbol{u}\, \sin\left(\frac{\Delta\theta\Delta T}{2}\right)\right). \tag{3.32}$$

donde ΔT es el tiempo de control. Utilizando este cuaternión de incremento, el cuaternión final será

$$\boldsymbol{Q}[k+1] = \Delta\boldsymbol{Q}[k]\, \boldsymbol{Q}[k] \tag{3.33}$$

donde $\boldsymbol{Q}[k]$ representa la orientación en el tiempo k, y el producto de $\Delta\boldsymbol{Q}[k]$ es por la izquierda debido a que la variación de la rotación (derivada del cuaternión) se da con respecto al sistema inercial o fijo.

3.4. Esquemas de Control Cinemático

Los esquemas que se utilizan para dar solución al problema de cinemática inversa se denominan en general resolutores (en inglés, *solvers*) y generan movimiento basados en el formalismo de las tareas cinemáticas, las cuales determinan los diversos objetivos de movimiento. En esta sección se

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

presenta los esquemas de solución que son integrados en esta tesis, el primero basado en ponderación de tareas, y los siguientes dos basados en prioridad de tareas.

3.4.1. Solución usando Ponderación de Tareas

El resolutor basado en ponderación de tareas encuentra la velocidad articular generalizada $\dot{\boldsymbol{q}}$ asignando pesos a cada una de las tareas, siendo aquellas tareas que tienen un peso mayor las tareas más importantes [43]. Considérese un sistema compuesto por n_t tareas que definen el movimiento deseado, y supóngase que la tarea i-ésima está definida por $\boldsymbol{e}_i$, J_i, y tiene un peso w_i, donde $w_i \geq 0$ (si $w_i = 0$ la tarea n o se considera). El esquema ponderado de solución consiste en resolver el siguiente problema de optimización

$$\min_{\dot{\boldsymbol{q}}} \left\{ \sum_{i=1}^{n_t} w_i \|\dot{\boldsymbol{e}}_i - J_i \dot{\boldsymbol{q}}\|^2 \right\}. \tag{3.34}$$

En este esquema, las tareas con mayor peso serán satisfechas de manera más cercana, pero si existen tareas con conflictos mutuos (que no puedan ser satisfechas a la vez) puede ocurrir que ninguna tarea sea completamente satisfecha sino que exista un compromiso entre ellas [43]. El problema con este esquema es que los valores de $\dot{\boldsymbol{q}}$ pueden ser grandes, lo cual es no deseable. Debido a esto, se añade un término de regularización $\|\dot{\boldsymbol{q}}\|^2$ y el problema se redefine como

$$\min_{\dot{\boldsymbol{q}}} \left\{ \lambda_q \|\dot{\boldsymbol{q}}\|^2 + \sum_{i=1}^{n_t} w_i \|\dot{\boldsymbol{e}}_i - J_i \dot{\boldsymbol{q}}\|^2 \right\} \tag{3.35}$$

donde λ_q es un escalar que se utiliza para determinar la importancia del factor de regularización. El término de regularización minimiza la norma de $\dot{\boldsymbol{q}}$ y permite obtener el valor más pequeño de incremento $\dot{\boldsymbol{q}}$. El argumento de la minimización puede ser escrito de la siguiente manera

$$\begin{aligned}
&\lambda_q \dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}} + \sum_{i=1}^{t} w_i \left(\dot{\boldsymbol{e}}_i - J_i \dot{\boldsymbol{q}}\right)^T \left(\dot{\boldsymbol{e}}_i - J_i \dot{\boldsymbol{q}}\right) \\
&\lambda_q \dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}} + \sum_{i=1}^{t} w_i \left(\dot{\boldsymbol{e}}_i^T \dot{\boldsymbol{e}}_i - \dot{\boldsymbol{e}}_i^T J_i \dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}^T J_i^T \dot{\boldsymbol{e}}_i + \dot{\boldsymbol{q}}^T J_i^T J_i \dot{\boldsymbol{q}}\right) \\
&\lambda_q \dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}} + \sum_{i=1}^{t} w_i \dot{\boldsymbol{q}}^T J_i^T J_i \dot{\boldsymbol{q}} - 2 w_i \dot{\boldsymbol{q}}^T J_i^T \dot{\boldsymbol{e}}_i + w_i \dot{\boldsymbol{e}}_i^T \dot{\boldsymbol{e}}_i \\
&\dot{\boldsymbol{q}}^T \left(\lambda_q I + \sum_{i=1}^{t} w_i J_i^T J_i\right) \dot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^T \left(\sum_{i=1}^{t} -2 w_i J_i^T \dot{\boldsymbol{e}}_i\right).
\end{aligned} \tag{3.36}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Utilizando esta última expresión, se puede formular el problema de minimización como un problema cuadrático:

$$\min_{\dot{\boldsymbol{q}}} \left\{ \dot{\boldsymbol{q}}^T W \dot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^T \boldsymbol{p} \right\} \tag{3.37}$$

donde

$$W = \lambda_q I + \sum_{i=1}^{t} w_i J_i^T J_i \qquad \text{y} \qquad \boldsymbol{p} = -2 \sum_{i=1}^{t} w_i J_i^T \dot{\boldsymbol{e}}_i. \tag{3.38}$$

Es importante notar que W es una matriz mientras que $\boldsymbol{p}$ es un vector. Además, el término $w_i \dot{\boldsymbol{e}}_i^T \dot{\boldsymbol{e}}_i$ en la expansión de (3.35) ha sido descartado dado que es independiente de $\dot{\boldsymbol{q}}$, que es la variable a ser optimizada, y solo actúa como constante (*bias*) sin alterar la solución óptima. En la implementación de esta tesis, el valor de λ_q es por defecto la unidad pero el valor puede ser modificado.

Consideración de Límites Articulares. Para que el movimiento sea reproducible por un robot real, los límites articulares no deben ser excedidos y por tanto el esquema debe considerarlos. Sean $\underline{\boldsymbol{q}}$, $\overline{\boldsymbol{q}}$ los límites articulares inferior y superior, respectivamente, y sean $\underline{\dot{\boldsymbol{q}}}$, $\overline{\dot{\boldsymbol{q}}}$ los límites de velocidad inferior y superior de cada articulación. Estos límites deben ser expresados con respecto a la variable de optimización que es $\dot{\boldsymbol{q}}$ [39]. Con este fin, se puede utilizar la expansión de Taylor

$$\boldsymbol{q}(t + \Delta T) = \boldsymbol{q}(t) + \dot{\boldsymbol{q}}(t)\Delta T + \mathcal{O}(\Delta T^2) \tag{3.39}$$

donde t es el tiempo, ΔT es un incremento, y $\mathcal{O}(\Delta T^2)$ representa los términos de orden superior. Resolviendo para $\dot{\boldsymbol{q}}(t)$, la expresión anterior puede ser escrita como

$$\dot{\boldsymbol{q}}(t) = \frac{\boldsymbol{q}(t + \Delta T) - \boldsymbol{q}(t)}{\Delta T}. \tag{3.40}$$

Dado que se desea que el valor articular en $t + \Delta T$ satisfaga los límites articulares, se utilizará los valores inferior o superior $\underline{\boldsymbol{q}}$ y $\overline{\boldsymbol{q}}$ para este valor en dicho instante de tiempo. Para el límite inferior se utiliza el límite más restrictivo, es decir, el máximo del límite articular $\underline{\boldsymbol{q}}$ y del límite de velocidad $\underline{\dot{\boldsymbol{q}}}$; de igual manera, se toma el límite más restrictivo para el límite superior. El problema de optimización dado en (3.37), considerando los límites articulares, puede ser escrito en forma del siguiente problema de optimización cuadrático (QP, del inglés *Quadratic Program*) como:

$$\min_{\dot{\boldsymbol{q}}} \left\{ \dot{\boldsymbol{q}}^T W \dot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^T \boldsymbol{p} \right\}$$

sujeto a

$$\text{máx} \left\{ \frac{\underline{\boldsymbol{q}} - \boldsymbol{q}}{\Delta T}, \underline{\dot{\boldsymbol{q}}} \right\} \leq \dot{\boldsymbol{q}} \leq \text{mín} \left\{ \frac{\overline{\boldsymbol{q}} - \boldsymbol{q}}{\Delta T}, \overline{\dot{\boldsymbol{q}}} \right\}. \tag{3.41}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

donde ΔT es el tiempo de control entre cada señal de control. Las restricciones de mínimo y máximo en (3.41) son tomadas elemento a elemento y aseguran que tanto los límites articulares como los de velocidades sean satisfechos.

3.4.2. Solución Proyectando en el Espacio Nulo

Esta solución, a diferencia de la anterior, brinda una solución jerárquica; es decir, la solución a la tarea con mayor jerarquía se obtiene primero y luego se obtiene soluciones para las siguientes tareas, las cuales se ejecutan de la manera más cercana posible pero sin interferir con las de mayor jerarquía [27]. A cada tarea se le asocia una prioridad $i \in \mathbb{N}^+$, donde un valor bajo de i indica una mayor prioridad. A continuación se realizará una descripción detallada de este enfoque.

Con una tarea. Considerando una única tarea definida por $\boldsymbol{e}_1$, con referencia deseada $\dot{\boldsymbol{e}}_1^*$, se requiere obtener una solución que satisfaga el sistema $\dot{\boldsymbol{e}}_1^* = J_1\dot{\boldsymbol{q}}$. Este problema se puede plantear como $\min_{\dot{\boldsymbol{q}}} \|\dot{\boldsymbol{e}}_1^* - J_1\dot{\boldsymbol{q}}\|^2$, cuya solución es

$$\dot{\boldsymbol{q}}^* = J_1^{\#}\dot{\boldsymbol{e}}_1^* + P_1\boldsymbol{z}_2 \tag{3.42}$$

donde $J_1^{\#}$ es la pseudo-inversa de Moore-Penrose de J_1, y $P_1 = I - J_1^{\#}J_1$ es un proyector en el espacio nulo de J_1. El valor de z_2 es arbitrario y la adecuada satisfacción'de la tarea dado que está proyectada en el espacio nulo de J_1. Sin embargo, $\boldsymbol{z}_2$ provee el espacio necesario para desarrollar otras tareas. Para una posterior generalización, resulta conveniente representar la expresión anterior como

$$\dot{\boldsymbol{q}}^* = \dot{\boldsymbol{q}}_1^* + P_1\boldsymbol{z}_2 \tag{3.43}$$

donde $\dot{\boldsymbol{q}}_1^* = J_1^{\#}\dot{\boldsymbol{e}}_1^*$ es la solución a la primera tarea.

Con dos tareas. Si existen dos tareas definidas mediante $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, donde $\boldsymbol{e}_1$ es la tarea de mayor prioridad, el valor anterior de $\boldsymbol{z}_2$ puede ser utilizado para encontrar una solución a la segunda tarea. Considerar que la segunda tarea está especificada como

$$\dot{\boldsymbol{e}}_2^* = J_2\dot{\boldsymbol{q}}. \tag{3.44}$$

El valor de $\dot{\boldsymbol{q}}$ que satisface esta segunda tarea debe a su vez ser una solución para la primera tarea, y por tanto se reemplaza la expresión de $\dot{\boldsymbol{q}}$ obtenida en (3.43) para el caso de una tarea. Así se tiene:

$$\dot{\boldsymbol{e}}_2^* = J_2\left(\dot{\boldsymbol{q}}_1^* + P_1\boldsymbol{z}_2\right) = J_2\dot{\boldsymbol{q}}_1^* + J_2P_1\boldsymbol{z}_2. \tag{3.45}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Ahora se observa que el valor de $\dot{\boldsymbol{q}}_1^*$ es constante y no se puede modificar, y la única forma de encontrar la solución para la segunda tarea es escogiendo adecuadamente el valor de $\boldsymbol{z}_2$. Despejando $\boldsymbol{z}_2$ de la expresión anterior, haciendo uso de la pseudo-inversa, se tiene

$$\boldsymbol{z}_2 = (J_2P_1)^{\#}(\dot{\boldsymbol{e}}_2^* - J_2\dot{\boldsymbol{q}}_1^*) + \tilde{P}_2\boldsymbol{z}_3 \tag{3.46}$$

donde $\tilde{P}_2 = I - (J_2P_1)^{\#}(J_2P_1)$ es una matriz que realiza la proyección al espacio nulo de J_2P_1, y $\boldsymbol{z}_3$ es un vector arbitrario que no interfiere con la solución obtenida para la segunda tarea debido a la proyección. Al reemplazar $\boldsymbol{z}_2$ en (3.43) se llega a

$$\dot{\boldsymbol{q}}^* = \dot{\boldsymbol{q}}_1^* + P_1(J_2P_1)^{\#}(\dot{\boldsymbol{e}}_2^* - J_2\dot{\boldsymbol{q}}_1^*) + P_1\tilde{P}_2\boldsymbol{z}_3. \tag{3.47}$$

Existe una propiedad de la pseudo-inversa de Moore-Penrose que establece la siguiente igualdad: $A(BA)^{\#} = (BA)^{\#}$ para matrices A y B de tamaños adecuados. Utilizando esta propiedad, se puede simplificar la expresión anterior como

$$\begin{aligned}\dot{\boldsymbol{q}}^* &= \dot{\boldsymbol{q}}_1^* + (J_2P_1)^{\#}(\dot{\boldsymbol{e}}_2^* - J_2\dot{\boldsymbol{q}}_1^*) + P_1\tilde{P}_2\boldsymbol{z}_3 \\ &= \dot{\boldsymbol{q}}_1^* + \dot{\boldsymbol{q}}_2^* + P_2\boldsymbol{z}_3 \\ &= \dot{\boldsymbol{q}}_{1,2}^* + P_2\boldsymbol{z}_3\end{aligned} \tag{3.48}$$

donde se ha utilizado las siguientes igualdades por conveniencia: $\dot{\boldsymbol{q}}_2^* = (J_2P_1)^{\#}(\dot{\boldsymbol{e}}_2^* - J_2\dot{\boldsymbol{q}}_1^*)$, $P_2 = P_1\tilde{P}_2$, y $\dot{\boldsymbol{q}}_{1,2}^* = \dot{\boldsymbol{q}}_1^* + \dot{\boldsymbol{q}}_2^*$. Se puede observar que con estas sustituciones, la solución obtenida para dos tareas (3.48) tiene una forma similar a la solución obtenida anteriormente para una tarea (3.43).

Con tres tareas. En este caso se considera que existe una tercera tarea $\boldsymbol{e}_3$ descrita por $\dot{\boldsymbol{e}}_3^* = J_3\dot{\boldsymbol{q}}_3$ y que tiene que ser satisfecha de manera tan cercana como sea posible, pero sin interferir ni con la segunda tarea, ni con la primera, ya que las dos primeras tareas tienen mayor prioridad. El procedimiento de solución es similar al utilizado para el caso de dos tareas, pero en este caso se aprovecha el vector $\boldsymbol{z}_3$ que se obtuvo en (3.48) ya que este vector está proyectado en el espacio nulo de las dos primeras tareas mediante P_2. Al igual que en el caso anterior, se puede demostrar que este vector estará dado por

$$\boldsymbol{z}_3 = (J_3P_2)^{\#}(\dot{\boldsymbol{e}}_3^* - J_3\dot{\boldsymbol{q}}_{1,2}^*) + \tilde{P}_3\boldsymbol{z}_4. \tag{3.49}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Reemplazando esta expresión en (3.48) se llega a

$$\begin{aligned}\dot{\boldsymbol{q}}^* &= \dot{\boldsymbol{q}}^*_{1,2} + (J_3P_2)^{\#}(\dot{\boldsymbol{e}}^*_3 - J_3\dot{\boldsymbol{q}}^*_{1,2}) + P_2\tilde{P}_3\boldsymbol{z}_4 \\ &= \dot{\boldsymbol{q}}^*_{1,2} + \dot{\boldsymbol{q}}^*_3 + P_3\boldsymbol{z}_4 \\ &= \dot{\boldsymbol{q}}^*_{1,2,3} + P_3\boldsymbol{z}_4 \end{aligned} \tag{3.50}$$

donde se ha realizado las siguientes sustituciones: $\dot{\boldsymbol{q}}^*_3 = (J_3P_2)^{\#}(\dot{\boldsymbol{e}}^*_3 - J_3\dot{\boldsymbol{q}}^*_{1,2})$, $P_3 = P_2\tilde{P}_3$, $\dot{\boldsymbol{q}}^*_{1,2,3} = \dot{\boldsymbol{q}}^*_{1,2} + \dot{\boldsymbol{q}}^*_3$, y $\tilde{P}_3 = I - (J_3P_2)^{\#}(J_3P_2)$ es una matriz que proyecta al espacio nulo de J_3P_2. Así, la configuración articular $\dot{\boldsymbol{q}}$ que da la solución para la tercera tarea sin interferir con las anteriores está dada por (3.50).

Con n tareas. Se puede observar que las soluciones dadas en (3.43) para una tarea, en (3.48) para dos tareas, y en (3.50) para 3 tareas, son similares y guardan el mismo formato, el cual puede generalizarse a más tareas. Para un número arbitrario de n_t tareas, las expresiones anteriores pueden generalizarse siguiendo el mismo procedimiento descrito. Así, para n_t tareas, la solución jerárquica $\dot{\boldsymbol{q}}$ basada en proyecciones sobre el espacio nulo anterior es

$$\dot{\boldsymbol{q}}^* = \dot{\boldsymbol{q}}^*_{1,2,\cdots,n} + P_n\boldsymbol{z}_{n+1} \tag{3.51}$$

donde

$$\dot{\boldsymbol{q}}^*_{1,2,\cdots,n} = \sum_{k=1}^{n_t}(J_kP_{k-1})^{\#}\left(\dot{\boldsymbol{e}}^*_k - J_k\,\dot{q}^*_{1,2,\cdots,k-1}\right) \tag{3.52}$$

con

$$P_k = P_{k-1}\Big(I - (J_kP_{k-1})^{\#}(J_kP_{k-1})\Big) \tag{3.53}$$

$$P_0 = I \tag{3.54}$$

$$\dot{\boldsymbol{q}}^*_0 = \boldsymbol{0}. \tag{3.55}$$

Una expresión similar se obtiene en [43]. Los valores iniciales mostrados en las dos últimas filas se utilizan únicamente para que el algoritmo de control cinemático mostrado sea general, pero no tienen significado físico aunque, dado que $P_0 = I$, el espacio nulo de la tarea cero es todo el espacio de la tarea y la primera tarea puede ser ejecutada sin restricción.

Consideración de Límites Articulares. Los límites articulares son tareas de desigualdad que tienen que ser consideradas para poder ejecutarse en un robot real. Este enfoque no considera por defecto este tipo de tareas ya que las proyecciones en el espacio nulo son por defecto igualdades.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Sin embargo es posible utilizar igualdades para incorporar desigualdades. Esto se realiza mediante funciones potenciales $\boldsymbol{f}_p(\boldsymbol{q})$ las cuales actúan como campos de atracción o repulsión haciendo que los valores se acerquen o alejen de determinados puntos [45]. Una función potencial se expresa como

$$\dot{\boldsymbol{q}} = \alpha \frac{\partial \boldsymbol{f}_p(\boldsymbol{q})}{\partial \boldsymbol{q}}. \tag{3.56}$$

La desventaja de utilizar funciones potenciales es que imponen restricciones suaves en lugar de restricciones duras; es decir, son satisfechas en la medida de lo posible pero no se puede garantizar que siempre se cumplirán. Usando funciones potenciales, los límites articulares son tomados en cuenta como

$$\boldsymbol{f}_p(\boldsymbol{q}) = \frac{1}{2n} \sum_{i=1}^{n} \left(\frac{\overline{q}_i - \underline{q}_i}{(\overline{q}_i - q_i)(q_i - \underline{q}_i)} \right) \tag{3.57}$$

dando un valor mínimo en $\frac{1}{2}(\overline{q}_i + \underline{q}_i)$ y un potencial alto cuando las articulaciones se acercan al valor límite. La tarea resultante es

$$\boldsymbol{q} = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}^T \tag{3.58}$$

donde

$$\dot{q}_i = \frac{\alpha(\overline{q}_i - \underline{q}_i)}{2n} \left(\frac{1}{(\overline{q}_i - q_i)^2 (q_i - \underline{q}_i)} - \frac{1}{(\overline{q}_i - q_i)(q_i - \underline{q}_i)^2} \right) \tag{3.59}$$

El esquema de prioridad basado en el espacio nulo y dada en (3.51) es a veces denominado esquema de prioridad *directo*, debido a que la solución comienza con la tarea de más alta prioridad y va haciea la tarea de más baja prioridad. Recientemente, otro esquema denominado prioridad *reversa* fue planteado [46], pero no es muy utilizado y no están claras las ventajas que esta metodología presenta. Por este motivo, no ha sido considerada su implementación en esta tesis.

3.4.3. Solución con Jerarquía de QPs

Esta solución consiste en encontrar el control cinemático para un orden jerárquico de tareas a través del uso de una cascada de QPs (programas cuadráticos) [44]. Debido a esta jerarquía, a este tipo de solución también se denomina HQP (*Hierarchical Quadratic Program*). Debido a que un problema de optimización cuadrática (QP) considera por defecto desigualdades, este esquema toma en consideración las tareas de desigualdad de manera natural. Así, para una tarea $\boldsymbol{e}_i$ es necesario indicar sus límites de referencia superior e inferior $\overline{\dot{\boldsymbol{e}}^*}$ y $\underline{\dot{\boldsymbol{e}}^*}$, respectivamente. Para una tarea de igualdad se tendrá que ambos límites de la inecuación son iguales; es decir, $\overline{\dot{e}} = \underline{\dot{e}}$. En este caso, el problema es

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

equivalente a una proyección en el espacio nulo. Por este motivo, este enfoque basado en una cascada de QPs puede ser considerado como una generalización de las proyecciones sobre el espacio nulo donde se incluye desigualdades.

A continuación se brinda los detalles paso a paso de esta implementación. Se asume que existen $n_t = k$ niveles de tareas donde la tarea con nivel 1 tiene mayor prioridad que la tarea 2, y así sucesivamente.

Tarea 1. Considérese la primera tarea $\boldsymbol{e}_1$ de mayor prioridad y dimensión m_1, definida mediante vectores de referencia superior e inferior $\overline{\dot{\boldsymbol{e}}_1^*}$ y $\underline{\dot{\boldsymbol{e}}_1^*}$ respectivamente, y con Jacobiano J_1. La solución a esta tarea se obtiene resolviendo el siguiente problema de minimización, formulado en términos de un programa cuadrático:

$$\begin{aligned} &\min_{\dot{\boldsymbol{q}},\boldsymbol{w}_1} \|\boldsymbol{w}_1\|^2 \\ &\text{sujeto a} \\ &\underline{\dot{\boldsymbol{e}}}_1 \leq J_1\dot{\boldsymbol{q}} + \boldsymbol{w}_1 \leq \overline{\dot{\boldsymbol{e}}}_1 \end{aligned} \tag{3.60}$$

donde $\boldsymbol{w}_1$ es una variable de holgura, la cual brinda una medición de cuán buena es la solución obtenida. Con mayor detalle, este problema ser escrito de la siguiente manera para hacer explícito el hecho que se utiliza un QP:

$$\begin{aligned} &\min_{\dot{\boldsymbol{q}},\boldsymbol{w}_1} \begin{bmatrix} \boldsymbol{w}_1^T & \dot{\boldsymbol{q}}^T \end{bmatrix} \begin{bmatrix} I_{m_1\times m_1} & 0_{m_1\times n} \\ 0_{n\times m_1} & 0_{n\times n} \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_1 \\ \dot{\boldsymbol{q}} \end{bmatrix} \\ &\text{sujeto a} \\ &\underline{\dot{\boldsymbol{e}}}_1 \leq \begin{bmatrix} I_{m_1\times m_1} & J_1 \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_1 \\ \dot{\boldsymbol{q}} \end{bmatrix} \leq \overline{\dot{\boldsymbol{e}}}_1 \end{aligned} \tag{3.61}$$

donde $J_1 \in \mathbb{R}^{m_1\times n}$ y los subíndices indican el tamaño de las matrices, para mayor claridad. Notar que la dimensión de la variable de holgura es la misma que la dimensión de la tarea; es decir, m_1 para este caso. El vector de optimización contiene tanto a $\boldsymbol{w}_1$ como a $\dot{\boldsymbol{q}}$ debido a que dichas variables son las variables de optimización. La solución a este problema de una tarea será el vector $\begin{bmatrix} \boldsymbol{w}_1^* & \dot{\boldsymbol{q}}^* \end{bmatrix}^T$.

Tarea 2. La segunda tarea $\boldsymbol{e}_2$ de dimensión m_2 y menor prioridad que la tarea 1, se define considerando vectores de referencia superior e inferior $\overline{\dot{\boldsymbol{e}}_2^*}$ y $\underline{\dot{\boldsymbol{e}}_2^*}$ respectivamente, y con Jacobiano J_2. La

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

formulación del problema se plantea como la solución al siguiente problema cuadrático de minimización:

$$\min_{\dot{\boldsymbol{q}},\boldsymbol{w}_2} \|\boldsymbol{w}_2\|^2$$

sujeto a (3.62)

$$\underline{\dot{\boldsymbol{e}}}_2 \leq J_2\dot{\boldsymbol{q}} + \boldsymbol{w}_2 \leq \overline{\dot{\boldsymbol{e}}}_2$$

$$\underline{\dot{\boldsymbol{e}}}_1 \leq J_1\dot{\boldsymbol{q}} + \boldsymbol{w}_1^* \leq \overline{\dot{\boldsymbol{e}}}_1$$

donde $\boldsymbol{w}_2$ es una variable de holgura que se desea optimizar, y $\boldsymbol{w}_1^*$ es la solución obtenida en el nivel 1 (tarea 1). Nótese que la restricción que hace referencia a la tarea uno (segunda fila de las restricciones) contiene la variable de holgura $\boldsymbol{w}_1^*$ que ha sido obtenida anteriormente. Esto garantiza que la solución actual será tan buena con respecto a la tarea 1 como lo fue la solución cuando solo existía la tarea 1. Este problema puede ser escrito de la siguiente manera para hacer explícito el hecho que se utiliza un QP.

$$\min_{\dot{\boldsymbol{q}},\boldsymbol{w}_2} \begin{bmatrix} \boldsymbol{w}_2^T & \dot{\boldsymbol{q}}^T \end{bmatrix} \begin{bmatrix} I_{m_2\times m_2} & 0_{m_2\times n} \\ 0_{n\times m_2} & 0_{n\times n} \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_2 \\ \dot{\boldsymbol{q}} \end{bmatrix}$$

sujeto a (3.63)

$$\begin{bmatrix} \underline{\dot{\boldsymbol{e}}}_2 \\ \underline{\dot{\boldsymbol{e}}}_1 - \boldsymbol{w}_1^* \end{bmatrix} \leq \begin{bmatrix} I_{m_2\times m_2} & J_2 \\ 0_{m_1\times m_2} & J_1 \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_2 \\ \dot{\boldsymbol{q}} \end{bmatrix} \leq \begin{bmatrix} \overline{\dot{\boldsymbol{e}}}_2 \\ \overline{\dot{\boldsymbol{e}}}_1 - \boldsymbol{w}_1^* \end{bmatrix}$$

donde $J_2 \in \mathbb{R}^{m_2\times n}$ y la dimensión de $\boldsymbol{w}_2$ es m_2, igual que la dimensión de la tarea 2. La solución al considerar la segunda tarea se representará como el vector $\begin{bmatrix} \boldsymbol{w}_2^* & \dot{\boldsymbol{q}}^* \end{bmatrix}^T$. Notar que en esta formulación, el problema para la tarea 2 incrementa en uno el número de restricciones que tenía el problema cuando solo había una tarea.

Tarea 3. Considerar ahora que existe una tercera tarea $\boldsymbol{e}_3$ de dimensión m_3 cuyo comportamiento de referencia se define mediante el comportamiento de referencia superior e inferior $\overline{\dot{\boldsymbol{e}}_3^*}$ y $\underline{\dot{\boldsymbol{e}}_3^*}$ respectivamente, y cuyo Jacobiano está dado por J_2. Esta tercera tarea tiene menor prioridad que la tarea 2 y que la tarea 1, y su satisfacción no debe interferir con estas otras dos tareas. El problema

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

en este caso se formula como el siguiente problema cuadrático de minimización

$$\min_{\dot{\boldsymbol{q}},\boldsymbol{w}_3} \|\boldsymbol{w}_3\|^2$$

sujeto a (3.64)

$$\begin{aligned}
\underline{\dot{\boldsymbol{e}}}_3 &\leq J_3\dot{\boldsymbol{q}} + \boldsymbol{w}_3 \leq \overline{\dot{\boldsymbol{e}}}_3 \\
\underline{\dot{\boldsymbol{e}}}_2 &\leq J_2\dot{\boldsymbol{q}} + \boldsymbol{w}_2^* \leq \overline{\dot{\boldsymbol{e}}}_2 \\
\underline{\dot{\boldsymbol{e}}}_1 &\leq J_1\dot{\boldsymbol{q}} + \boldsymbol{w}_1^* \leq \overline{\dot{\boldsymbol{e}}}_1
\end{aligned}$$

donde $\boldsymbol{w}_3$ es la variable de holgura, $\boldsymbol{w}_2^*$ es la solución obtenida en el nivel 2 (tarea 2), y $\boldsymbol{w}_1^*$ es la solución obtenida en el nivel 1 (tarea 1). Para hacer explícito el hecho que se utiliza un problema cuadrático (QP), la formulación anterior puede ser escrita de la siguiente manera

$$\min_{\dot{\boldsymbol{q}},\boldsymbol{w}_3} \begin{bmatrix} \boldsymbol{w}_3^T & \dot{\boldsymbol{q}}^T \end{bmatrix} \begin{bmatrix} I_{m_3\times m_3} & 0_{m_3\times n} \\ 0_{n\times m_3} & 0_{n\times n} \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_3 \\ \dot{\boldsymbol{q}} \end{bmatrix}$$

sujeto a (3.65)

$$\begin{bmatrix} \underline{\dot{\boldsymbol{e}}}_3 \\ \underline{\dot{\boldsymbol{e}}}_2 - \boldsymbol{w}_2^* \\ \underline{\dot{\boldsymbol{e}}}_1 - \boldsymbol{w}_1^* \end{bmatrix} \leq \begin{bmatrix} I_{m_3\times m_3} & J_3 \\ 0_{m_2\times m_3} & J_2 \\ 0_{m_1\times m_3} & J_1 \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_2 \\ \dot{\boldsymbol{q}} \end{bmatrix} \leq \begin{bmatrix} \overline{\dot{\boldsymbol{e}}}_3 \\ \overline{\dot{\boldsymbol{e}}}_2 - \boldsymbol{w}_2^* \\ \overline{\dot{\boldsymbol{e}}}_1 - \boldsymbol{w}_1^* \end{bmatrix}$$

donde $J_3 \in \mathbb{R}^{m_3\times n}$. La solución en este caso estará representada mediante el vector $\begin{bmatrix} \boldsymbol{w}_3^* & \dot{\boldsymbol{q}}^* \end{bmatrix}^T$ que contiene tanto el valor mínimo de la variable de holgura, como el valor de velocidad articular necesario. Esta solución será tal que no interferirá con las tareas 2 y 1 debido a que las restricciones de la segunda y tercera fila utilizan las variables de holgura solucionadas en las etapas anteriores.

Formulación Genérica. Se puede notar que el enfoque tiene un patrón determinado y puede generalizarse para $n_t = k$ tareas. En general, usando este enfoque, la función de optimización en el nivel k-ésimo de prioridad se escribe como [44]:

$$\min_{\dot{\boldsymbol{q}},\boldsymbol{w}_k} \|\boldsymbol{w}_k\|^2$$

sujeto a (3.66)

$$\begin{aligned}
\overline{\dot{\boldsymbol{e}}}_k &\leq J_k\dot{\boldsymbol{q}} + \boldsymbol{w}_k \leq \underline{\dot{\boldsymbol{e}}}_k \\
\overline{\dot{\boldsymbol{e}}}_{k-1}^s &\leq J_{k-1}^s\dot{\boldsymbol{q}} + \boldsymbol{w}_{k-1}^{s*} \leq \underline{\dot{\boldsymbol{e}}}_{k-1}^s
\end{aligned}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

donde $\underline{\dot{e}}_k$, $\overline{\dot{e}}_k$ son los límites inferior y superior de la tarea k-ésima, y $\boldsymbol{w}_k$ es una variable de holgura (en inglés denominada *slack*) en el nivel k-ésimo. La prioridad queda considerada utilizando los límites apilados de las tareas previas $\overline{\dot{\boldsymbol{e}}}^s_{k-1}$, los Jacobianos apilados J^s_{k-1} y las variables de holgura optimizadas apiladas $\boldsymbol{w}^{s*}_{k-1}$ cuyos elementos en el caso general están dados por

$$\overline{\dot{\boldsymbol{e}}}^s_{k-1} = \begin{bmatrix} \overline{\dot{\boldsymbol{e}}}_{k-1} \\ \vdots \\ \overline{\dot{\boldsymbol{e}}}_1 \end{bmatrix}, \qquad \underline{\dot{\boldsymbol{e}}}^s_{k-1} = \begin{bmatrix} \underline{\dot{\boldsymbol{e}}}_{k-1} \\ \vdots \\ \underline{\dot{\boldsymbol{e}}}_1 \end{bmatrix}, \qquad J^s_{k-1} = \begin{bmatrix} J_{k-1} \\ \vdots \\ J_1 \end{bmatrix}, \qquad \boldsymbol{w}^{s*}_{k-1} = \begin{bmatrix} \boldsymbol{w}^*_{k-1} \\ \vdots \\ w^*_1 \end{bmatrix}$$

Más específicamente, $\boldsymbol{w}^*_i$ representa la solución obtenida en los pasos previos de optimización. El problema de optimización (3.66) se resuelve primero para la tarea 1 obteniendo $\boldsymbol{w}^*_1$. Luego, se soluciona para la tarea 2 usando $\boldsymbol{w}^*_1$ como una restricción estricta (*hard*) para la tarea 1. Incluyendo $\boldsymbol{w}^*_1$, la solución de la tarea 2 también será óptima con respecto a la tarea 1 alcanzando prioridad entre tareas. De la misma manera se prosigue para todas las tareas.

3.5. Tareas Cinemáticas

En el enfoque basado en tareas, las tareas cinemáticas son fundamentales para describir el movimiento deseado de un robot al considerar únicamente la cinemática. Siguiendo los componentes descritos en la sección 3.1, es necesario especificar la función de tarea, el comportamiento de referencia y el mapa para una tarea determinada. El detalle cinemático de estos componentes se describió en la sección 3.3.1. En esta sección se presentará los componentes para las tareas utilizadas en esta tesis. Algunas de estas tareas, pero para el caso dinámico, se encuentran descritas en [39].

3.5.1. Tarea de Posición

El objetivo de esta tarea consiste en mover algún punto operacional del robot de la posición actual a una posición deseada. En este caso no se considera la orientación y, por tanto, se puede llegar a la posición deseada con cualquier orientación. La representación escogida es la Cartesiana debido a que es la más utilizada y debido a que no existe, para el caso genérico, ventaja en la utilización de otra representación.

Definición del error. Sea $\boldsymbol{x}(\boldsymbol{q}) = (x(\boldsymbol{q}), y(\boldsymbol{q}), z(\boldsymbol{q})) \in \mathbb{R}^3$ la posición actual de algún punto operacional, y sea $\boldsymbol{x}_d = (x_d, y_d, z_d) \in \mathbb{R}^3$ la posición deseada. Se define el error de posición como

$$\boldsymbol{e} = \boldsymbol{x}_d - \boldsymbol{x}(\boldsymbol{q}). \tag{3.67}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Es importante notar que $\boldsymbol{x}(\boldsymbol{q})$ depende de la configuración actual del sistema y se obtiene a través del cálculo de la cinemática directa. Además, se realizará la suposición que $\boldsymbol{x}_d$ es constante y por tanto no tiene dependencia del tiempo, como se observa.

En el caso de tener únicamente la cinemática directa con respecto a la base fija del robot ${}^b\boldsymbol{x}(\boldsymbol{q}_a)$, donde ^{b} indica el sistema de referencia de la base del robot, se puede formular la cinemática directa completa como

$$\boldsymbol{x}(\boldsymbol{q}) = \boldsymbol{p}_b + \boldsymbol{R}_b(\mathbf{Q}_b)\, {}^b\boldsymbol{x}(\boldsymbol{q}_a) \tag{3.68}$$

donde $\boldsymbol{R}_b(\mathbf{Q}_b)$ es la matriz de rotación que representa la orientación de la base del robot en el sistema de referencia inercial, y depende del cuaternión de la base $\mathbf{Q}_b$. Además, $\boldsymbol{p}_b$ es la posición de la base del robot en el mismo sistema inercial. Se puede observar que $\boldsymbol{x}$ es una función de $\boldsymbol{q}$ a diferencia de ${}^b\boldsymbol{x}$ que solo depende de $\boldsymbol{q}_a$ dado que depende además de la base del robot.

Jacobiano de Posición. Para una tarea de posición, el Jacobiano de la tarea, también llamado analítico, es igual al Jacobiano de velocidad lineal dado por la primera fila de (3.21) si se utiliza coordenadas Cartesianas. Como se mencionó anteriormente, en esta tesis se utiliza únicamente coordenadas Cartesianas para representar la posición. Así, el Jacobiano de posición es

$$J(\boldsymbol{q}) = J_v(\boldsymbol{q}) = \begin{bmatrix} I & (\boldsymbol{p}_b - \boldsymbol{p}_i)^{\wedge}\, T(\boldsymbol{Q}_b) & \boldsymbol{R}_b(\boldsymbol{Q}_b)\, {}^bJ_v(\boldsymbol{q}_a) \end{bmatrix} \tag{3.69}$$

donde $\boldsymbol{p}_i = \boldsymbol{x}(q)$ es la posición del punto operacional que se quiere controlar y J_v es el jacobiano de posición del punto operacional en el sistema de la base del robot considerando solamente las articulaciones actuadas.

3.5.2. Tarea de Orientación

El objetivo de esta tarea consiste en girar algún punto operacional del robot de tal modo que alcance una orientación deseada. En este caso no se considera la posición y por tanto ésta puede cambiar; lo único importante es la orientación con respecto a un sistema de referencia dado. Como se mencionó anteriormente, la representación que se utiliza en esta tesis para la orientación es el cuaternión, ya que no presenta singularidades de representación.

Definición del error. Sea $\boldsymbol{Q} = (w, \boldsymbol{\epsilon})$ el cuaternión que representa la orientación actual de algún punto operacional, y sea $\boldsymbol{Q}_d = (w_d, \boldsymbol{\epsilon}_d)$ el cuaternión que representa la orientación deseada. Para definir el error de orientación de $\boldsymbol{Q}$ a $\boldsymbol{Q}_d$ se puede definir el cuaternión $\boldsymbol{Q}_e = \boldsymbol{Q}_d\boldsymbol{Q}^{-1}$ que describe el error entre ambos cuaterniones. Si ambos fuesen iguales, se tendría la identidad $\boldsymbol{Q}_e = (1, 0, 0, 0)$.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Por tanto, el error se define con relación a la identidad. El producto que origina $\boldsymbol{Q}_e$, usando la definición de producto de cuaterniones, es

$$\begin{aligned}\boldsymbol{Q}_e &= (w_d, \boldsymbol{\epsilon}_d)(w, -\boldsymbol{\epsilon}) \\ &= (w_d w + \boldsymbol{\epsilon}_d^T \boldsymbol{\epsilon}, -w_d \boldsymbol{\epsilon} + w\boldsymbol{\epsilon}_d - \boldsymbol{\epsilon}_d \times \boldsymbol{\epsilon}) \\ &= (w_e, \boldsymbol{\epsilon}_e). \end{aligned} \tag{3.70}$$

Dado que se busca la identidad, utilizando la expresión anterior es posible definir el error de orientación $\boldsymbol{e}$ como

$$\boldsymbol{e} = \begin{bmatrix} w_e - 1 \\ \boldsymbol{\epsilon}_e \end{bmatrix} \tag{3.71}$$

donde se hace explícita la diferencia entre $\boldsymbol{Q}_e$ y la identidad para cuaterniones. En caso de tener la cinemática directa con respecto a la base fija del robot, se obtiene el cuaternión de orientación con respecto a esta base fija ${}^b\boldsymbol{Q}$. En este caso, el cuaternión con respecto al sistema inercial se obtiene como

$$\boldsymbol{Q} = \boldsymbol{Q}_b \, {}^b\boldsymbol{Q} \tag{3.72}$$

donde $\boldsymbol{Q}_b$ es el cuaternión que describe la orientación de la base del robot en el sistema de referencia inercial.

Jacobiano de Orientación. Para una tarea de orientación basada en cuaterniones, el Jacobiano analítico describe la relación entre la razón de cambio del cuaternión y la velocidad articular. Para calcularlo se puede utilizar la expresión (2.40) que relaciona este Jacobiano $J_o(\boldsymbol{q})$ con la parte de orientación del Jacobiano geométrico $J_\omega(\boldsymbol{q})$ como $J_o(\boldsymbol{q}) = E_o(\boldsymbol{o})J_\omega(\boldsymbol{q})$. En adelante se utilizará J en lugar de J_o para simplificar la notación. Cuando la orientación $\boldsymbol{o}$ se representa por un cuaternión, $\boldsymbol{o} = \boldsymbol{Q}$, esta expresión se puede expresar como

$$J(\boldsymbol{q}) = T(\boldsymbol{Q})^{\#} J_\omega(\boldsymbol{q}) \tag{3.73}$$

donde $T(\boldsymbol{Q})^{\#} = E_o(\boldsymbol{Q})$ es la pseudo-inversa de $T(\boldsymbol{Q})$, y $T(\boldsymbol{Q})$ está dada por (3.15). Se debe notar que se ha utilizado la pseudo-inversa y no simplemente la inversa debido a que (3.15) no es una matriz cuadrada y por tanto no es invertible. Es posible demostrar que $T(\boldsymbol{Q})$ es de rango completo por filas, de tal modo que $T(\boldsymbol{Q})T(\boldsymbol{Q})^T$ es no singular y la pseudoinversa es $T^{\#} = T^T(TT^T)^{-1}$,

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

donde se omitió la dependencia de $\boldsymbol{Q}$. Usando la expresión anterior se tiene

$$T(\boldsymbol{Q})^{\#} = \frac{1}{2}\begin{bmatrix} -\epsilon_x & -\epsilon_y & -\epsilon_z \\ w & \epsilon_z & -\epsilon_y \\ -\epsilon_z & w & \epsilon_x \\ \epsilon_y & -\epsilon_x & w \end{bmatrix}. \tag{3.74}$$

Por otro lado, como se muestra en (3.21) el Jacobiano geométrico del robot humanoide o con base flotante está dado por

$$J_\omega(\boldsymbol{q}) = \begin{bmatrix} \mathbf{0} & T(\boldsymbol{Q}_b) & \boldsymbol{R}_b(\boldsymbol{Q}_b)\, {}^bJ_\omega(\boldsymbol{q}_a) \end{bmatrix}. \tag{3.75}$$

Utilizando las expresiones anteriores, el Jacobiano analítico para la tarea de orientación queda dado por (3.73).

3.5.3. Tarea de Posición y Orientación

Esta tarea consiste en controlar a la vez tanto la posición como la orientación de algún punto operacional del robot.

Definición del error. En este caso, el error de posición y orientación (llamados *pose* en inglés) se puede representar matemáticamente de manera genérica como

$$\boldsymbol{e} = M^* \ominus M \tag{3.76}$$

donde $M \in SE(3)$ es alguna representación de la posición y orientación, M^* es su valor deseado, y $\ominus$ es un operador de diferencia en $SE(3)$. En esta tesis la posición se representa mediante coordenadas Cartesianas, y la orientación mediante cuaterniones, y por tanto el error queda dado por (3.67) y (3.71).

Jacobiano de Posición y Orientación. En este caso se considera un Jacobiano de siete filas, donde las tres primeras están compuestas por la parte correspondiente a la posición y las cuatro últimas corresponden a la orientación. Es decir, se tendrá:

$$J = \begin{bmatrix} J_{\text{pos}} \\ J_{\text{or}} \end{bmatrix} \tag{3.77}$$

donde J_{pos} está dado por (3.69) y J_{or} por (3.73).

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

3.5.4. Tarea de Postura

Una postura es una configuración específica para algunas articulaciones. Así, esta tarea busca que las articulaciones alcancen una postura específica. Será posible especificar una postura para todas las articulaciones o solamente para algunas de ellas.

Definición del error. Sea $\tilde{\boldsymbol{q}}$ un subconjunto arbitrario del vector de configuración actual tal que $\tilde{\boldsymbol{q}} = \{q_i, q_j, \cdots, q_k\}$, y sea $\tilde{\boldsymbol{q}}_d$ el vector de postura deseado para las articulaciones anteriores tal que $\tilde{\boldsymbol{q}}_d = \{q_{i_d}, q_{j_d}, \cdots, q_{k_d}\}$. El error se define como

$$\boldsymbol{e} = \tilde{\boldsymbol{q}}_d - \tilde{\boldsymbol{q}} \tag{3.78}$$

y su tamaño dependerá de la cantidad de articulaciones que se utilice. Se puede observar que en realidad se trata de una tarea en el espacio de configuración, pero también se nota que el espacio operacional contiene, en cierta forma, al espacio de configuración.

Jacobiano de Postura. Este Jacobiano contendrá ceros en todos los lugares excepto en aquellos relacionados con las articulaciones a controlar, debido a que se desea el movimiento de las mismas. Así, se puede definir como

$$J = \begin{bmatrix} \mathbf{0}_{ff} & J_q \end{bmatrix} \tag{3.79}$$

donde $\mathbf{0}_{ff}$ es una matriz de ceros correspondiente a la parte que contiene la base flotante, y J_q es una matriz que contiene ceros excepto en las posiciones correspondientes a las articulaciones a ser controladas, donde contiene unos. Para el caso descrito en la definición del error, las posiciones con unos serían $(1, i+7), (2, j+7), \cdots, (n, k+7)$, donde se asume que los índices comienzan de 1. El número 7 anterior se debe a que las columnas incluyen además la base flotante.

3.5.5. Otras Tareas de Desigualdad

En algunos casos las tareas de igualdad no son suficientes. Por ejemplo, cuando se requiere que un punto tridimensional se encuentre dentro del campo visual de una cámara localizada en la cabeza del robot, dicha especificación no puede ser representada como igualdad. En estos casos se requiere desigualdades. Una tarea de desigualdad se representa como

$$\underline{\boldsymbol{e}} \leq \boldsymbol{e} \leq \overline{\boldsymbol{e}}$$

donde $\underline{\boldsymbol{e}}$ y $\overline{\boldsymbol{e}}$ son los limites inferior y superior, respectivamente, de la tarea. En el caso cinemático, dicha desigualdad debe ser expresada en función de la velocidad (dado que la cinemática se basa en

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

velocidades) para lo cual es usual utilizar la aproximación de primer orden de Taylor. Estas tareas de desigualdad solo pueden ser utilizadas en esquemas que hacen uso de QPs, como el esquema ponderado y el esquema jerárquico, pero no en esquemas basados en proyecciones sobre el espacio nulo. Para hacer uso de tareas de desigualdad en este último caso se requeriría transformarlas en funciones potenciales, pero esto queda más allá de los alcances de esta tesis. En esta tesis no se utilizará tareas de desigualdad más allá de la validación de los límites articulares que se encuentran en la misma definición de los sistemas de control cinemáticos.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Capítulo 4

Implementación del Sistema de Generación de Movimiento

Los fundamentos teóricos genéricos presentados en el capítulo 2 relacionados con la representación de cuerpos rígidos y la cinemática de robots manipuladores, así como el diseño teórico específico del sistema de generación cinemática del movimiento presentado en el capítulo 3 necesitan ser implementados para poder ser utilizados. Este capítulo presenta en detalle la organización y las partes del sistema implementado, así como las herramientas utilizadas para la implementación. Una de las características principales de esta implementación es el ser independiente de *middlewares*, dependiendo solamente de algunas librerías elementales libres e independientes de la plataforma, así como el hecho de ser *Open Source*. El código de todo el sistema desarrollado se encuentra disponible como código abierto en `https://github.com/oeramosp/oscr`.

4.1. Herramientas utilizadas en el Sistema

Para implementar el sistema propuesto es necesario el uso de algunas herramientas computacionales y de algunos lenguajes de programación. Existen varias alternativas como MATLAB, por ejemplo, pero el objetivo es hacer los cálculos de la manera más rápida y eficiente posible ya que los periodos de control pueden ser tan cortos como 1 ms. Por este motivo se implementó la base del sistema en C++, ya que el código es compilado en binario y puede ser ejecutado de manera eficiente. Para mayor facilidad se realizó además una interface que permite utilizar las librerías implementadas en C++ desde Python. Si bien es posible implementar todos los algoritmos desde cero, se optó por hacer uso de algunas librerías de propósito general que permiten el manejo de matrices y algunas operaciones cinemáticas básicas. La razón es que el objetivo principal de la tesis es la implementación de esquemas de generación de movimiento cinemático y no el desarrollo de aspectos

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

más elementales desde el punto de vista computacional. Además, existen librerías de código abierto altamente especializadas y eficientes, cuya velocidad se quiso aprovechar. Esta sección presenta las herramientas *open source* utilizadas como base para implementar cada uno de los componentes del sistema de generación de movimiento propuesto en esta tesis.

4.1.1. Manejo de Álgebra Lineal

La implementación de cualquier algoritmo en robótica necesita el uso de vectores, matrices y diversas operaciones de álgebra lineal. Algunos lenguajes de programación de tipo *script* como Matlab, Octave, Scilab, o Python (usando la librería NumPy) poseen herramientas inherentes para el manejo de álgebra lineal; es decir, para realizar operaciones sobre vectores y matrices. Sin embargo, ni C ni C++ poseen dentro de sus diversos estándares (incluyendo el más reciente C++17) herramientas para este manejo, por lo que es necesario el uso de librerías adicionales. Para el lenguaje C existen librerías como LAPACK (http://www.netlib.org/lapack) o BLAS (http://www.netlib.org/blas), ambas escritas en Fortran pero con interfaz para C, que permiten el uso de operaciones matriciales. Sin embargo, dado que la implementación de esta tesis se realiza utilizando C++, se prefirió mantener coherencia con este lenguaje, y su formato, y no utilizar librerías desarrolladas para C. En su lugar, se utilizó una librería para C++, escrita también en C++, denominada *Eigen*.

Eigen. Es una librería de C++ basada en plantillas (*templates*) para álgebra lineal: matrices, vectores, resolutores numéricos y algoritmos relacionados [47]. Esta librería se originó como un proyecto local en un laboratorio de investigación de INRIA, en Francia, pero es actualmente desarrollada y soportada por una comunidad cada vez más grande a nivel internacional. La librería es distribuida como software libre y está siendo utilizada en diversas aplicaciones que requieren el manejo de álgebra lineal. La característica de esta librería es que está escrita usando el paradigma conocido como *meta-programación* con base en plantillas de C++ y no requiere de librerías adicionales sino solamente las estándares de C++98. Además, está optimizada para diferentes SIMD (*Single Instruction Multiple Data*) de arquitecturas INTEL, ARM, IBM Power, entre otras, lo cual le permite ser bastate eficiente.

En esta tesis se utiliza Eigen para realizar el manejo de matrices y vectores, así como diversas operaciones básicas sobre ellos, operaciones por filas y columnas, descomposición de matrices (por ejemplo, SVD), inversas, producto punto, producto cruz, etc. Para mayor información sobre Eigen y para una introducción a su uso se puede consultar el tutorial provisto por la comunidad de Eigen en: http://eigen.tuxfamily.org/dox

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

4.1.2. Modelamiento del Robot

Para poder obtener información cinemática o dinámica de un robot es necesario conocer los parámetros que describen su modelo, como las longitudes, masas, momentos de inercia, ejes de giro, uniones eslabón-articulación, etc. Los manipuladores industriales suelen tener su propio formato, el cual o se encuentra en formato binario propietario o solamente es compatible con los programas desarrollados por el propio fabricante. Igualmente los laboratorios de investigación suelen utilizar sus propios formatos, normalmente incompatibles unos con otros. Debido a estos problemas de incompatibilidad que complican el uso de diversos modelos de robots, alrededor del año 2010 surgió un nuevo formato, denominado *URDF*, que de alguna manera trata de dar solución a esta gran variedad de formatos. En esta tesis, los robots están descritos utilizando este formato.

URDF. Este formato, con siglas provenientes del inglés *Unified Robot Description Format*, es un formato basado en XML que se utiliza para describir todos los elementos de un robot (`http://wiki.ros.org/urdf`). Contiene información sobre la estructura cinemática, parámetros dinámicos como masa y momento de inercia de cada eslabón, tipos de articulaciones, estructuras para colisiones, apariencia visual, parámetros para simulación, etc. Este formato se originó como un proyecto en la Universidad de Stanford, en Estados Unidos, pero poco a poco su uso fue extendiéndose, y actualmente gran parte de universidades y laboratorios de investigación con enfoque en robótica lo utilizan o lo soportan. Más aún, varios fabricantes de robots industriales, como Kuka, ABB, Fanuc, Motoman, Stäubli, entre otros, han empezado a utilizarlo y actualmente es posible encontrar manipuladores industriales con este formato. Normalmente, los robots descritos mediante *urdf* pueden ser encontrados en repositorios git o mercurial provistos por el propio fabricante o por laboratorios asociados. Igualmente, cualquier robot, sea manipulador, móvil, humanoide, quadrúpedo, aéreo, etc. puede ser descrito utilizando este formato. El tutorial oficial, que puede ser usado como punto de partida con este formato, se encuentra en: `http://wiki.ros.org/urdf/Tutorials`.
El sistema desarrollado en esta tesis usa como entrada robots descritos mediante URDF, sean estos robots existentes, o robots arbitrariamente creados por el usuario. La lectura o *parsing* de un modelo URDF puede ser realizado mediante las funciones de C++ provistas para este fin, normalmente con la librería `urdfdom`. Sin embargo, en esta tesis por facilidad se utiliza dos sistemas externos (RBDL y Pinocchio), descritos en la sección 4.1.3 para la extracción de información básica del modelo URDF. Un ejemplo de URDF que describe un robot cuadrúpedo arbitrariamente sintetizado se muestra en el apéndice B.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

4.1.3. Cinemática Básica del Robot

Una vez que se conoce tanto los parámetros que describen a un robot, como la configuración que éste tiene, es posible obtener su modelo cinemático. Como se mencionó en la sección 2.3.1 se puede emplear el método de Denavit-Hartenberg o algún otro método geométrico para calcular la cinemática directa. Si bien estos métodos son generales en su procedimiento y aplicación, de forma práctica implican una gran labor manual de modelamiento, la cual varía mucho de robot a robot. Para automatizar este proceso, es posible el uso sistemático de la información cinemática contenida en el modelo URDF del robot, la cual brinda relaciones entre eslabones, dimensiones, posiciones de los ejes articulares y organización de la cadena cinemática. Esto se logra a través del *parsing* adecuado del modelo y del uso de matrices de transformación homogénea. Una vez que se tiene el modelo cinemático se puede realizar de manera sistemática el cálculo de los Jacobianos geométricos, como se menciona brevemente en la sección 2.3.4, para cada punto operacional.
El cálculo de la cinemática directa y de los Jacobianos geométricos es un problema ampliamente conocido en robótica y su solución no aporta algo realmente novedoso. Sin embargo, desde el punto de vista de ingeniería es necesario su obtención para cada robot con el que se trabaja, debido a la información que proveen y a su uso para control cinemático. Por este motivo, y dado que su cálculo no es un objetivo de la tesis, se prefirió utilizar algunas herramientas *open source* existentes y eficientes, las cuales son *RBDL* y *Pinocchio*, y se describirán a continuación. Ambas librerías extraen información del modelo URDF del robot y brindan funciones elementales de cinemática directa y diferencial, entre otras cosas, pero no proveen elementos de control ni a nivel cinemático ni a nivel dinámico, lo cual es el objetivo de la tesis.

RBDL. Son las siglas en inglés de *Rigid Body Dynamics Library* y es una librería altamente eficiente para C++ que contiene algunos algoritmos esenciales de dinámica de cuerpos rígidos como el algoritmo de cuerpos articulados (ABA) para dinámica directa, el algoritmo recursivo de Newton-Euler (RNEA) para dinámica inversa, y el algoritmo de cuerpos rígidos compuestos (CRBA) para el cálculo eficiente de la matriz de inercia. Esta librería, desarrollada por el grupo de investigación en "Optimización en Robótica y Biomecánica" de la Universidad de Heidelberg en Alemania, puede ser obtenida en la página `https://rbdl.bitbucket.io/`. La librería está diseñada para ser utilizada principalmente en Linux, aunque existe forma de compilarla en otros sistemas operativos.

Pinocchio. Es una librería *open source* de C++ para cálculos eficientes de dinámica de cuerpos rígidos, y fue desarrollada en el grupo de investigación de Sistemas Antropomórficos del laboratorio de investigación LAAS-CNRS, en Toulouse, Francia (`https://github.com/stack-of-tasks/`

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

pinocchio). Implementa algoritmos eficientes relacionados con el movimiento de cuerpos rígidos para sistemas poliarticulados basado en el estado del arte de esta área. Esta implementación se fundamenta en *Eigen* para el manejo de álgebra lineal, y en *FCL* para la detección de Colisiones y utiliza meta-programación basada en plantillas de C++, lo que le permite efectuar cálculos de manera eficiente y optimizada. Está diseñada para ser utilizada principalmente en Linux aunque puede ser empleada en otros sistemas operativos.

4.1.4. Resolución de QPs

La solución de algunos de los esquemas de control cinemático mostrados en la sección 3.4, en particular del método basado en ponderación de tareas y del método de jerarquía de QPs, requiere la solución de un problema de optimización cuadrático. Estos problemas, en el caso general, no tienen una solución analítica simple, sino algorítmica. Es posible la implementación manual de estas soluciones pero para que sean eficientes, como se requiere en este caso, se requiere el uso diversas técnicas numéricas. Debido a que el objetivo de esta tesis no es la resolución eficiente de QPs, se utilizó la librería llamada *QPOases* [48] que resuelve este tipo de problemas eficientemente.

QPOases. Es una librería open source para C++ que resuelve problemas de optimización cuadrática (QPs) a través de la implementación de la estrategia de conjuntos activos o *active sets* (https://projects.coin-or.org/qpOASES). Su forma de implementación y los fundamentos teóricos sobre los que se basa lo hacen particularmente adecuado para resolver sistemas que impliquen aplicaciones de control predictivo basado en modelo (MPC). Está, además, implementado de manera numéricamente robusta para poder lidiar con casos de QPs semi-definidos, mal condicionados o degenerados, los cuales necesitan un tratamiento especial. Esta librería fue desarrollada por el grupo de "Optimización en Ingeniería" de la Universidad Católica de Leuven (KU Leuven) en Bélgica, y actualmente es soportada por investigadores de la universidad de Heidelberg, en Alemania, y por el Centro de Investigación Corporativa de ABB, en Suiza.

4.2. Estructura General del Sistema

El sistema desarrollado en esta tesis integra la cinemática de robots humanoides y robots con base flotante altamente redundantes, con diversos esquemas de control cinemático en el espacio operacional descritos en la sección 3.4, así como todas las herramientas subyacentes, en una librería unificada de tipo *open source*. Esta librería se denomina *OSCR* que son siglas en inglés de *Operational Space Control for redundant Robots*, ya que el objetivo principal es el control en el espacio operacional de robots redundantes, de los cuales un robot humanoide es un caso

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

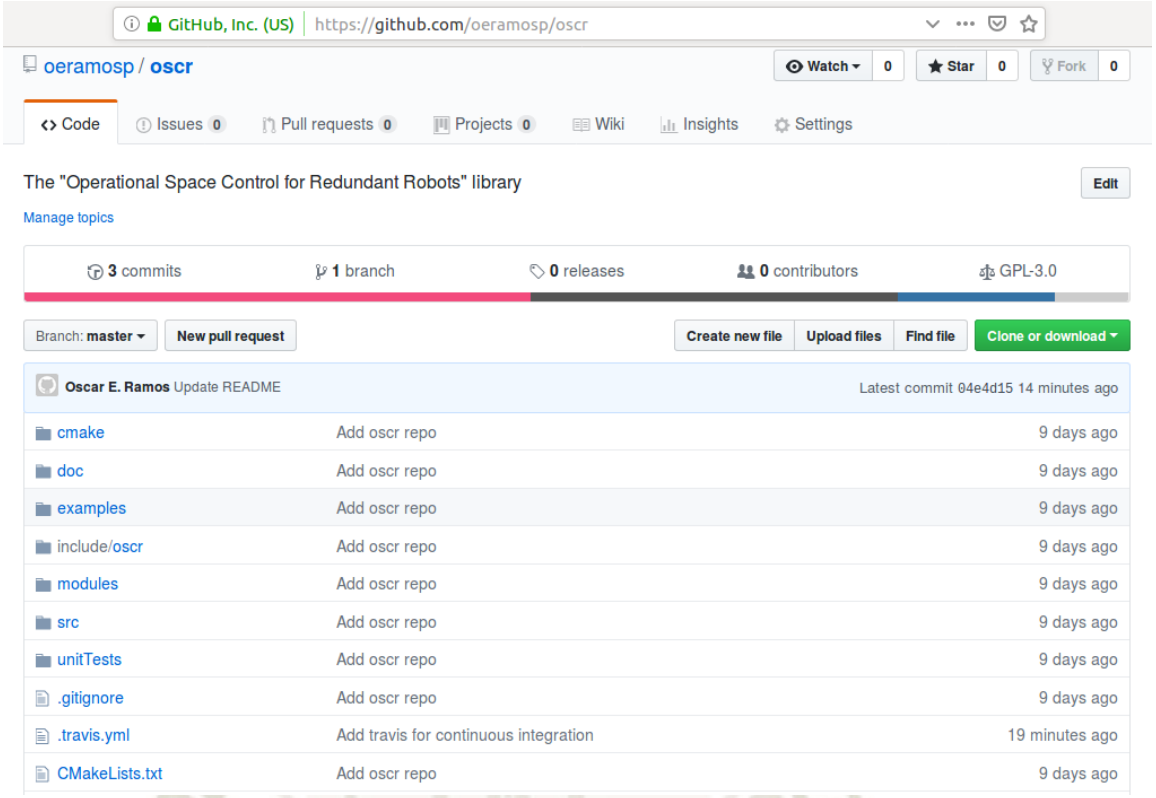


FIGURA 4.1: Captura de pantalla del repositorio en github del paquete implementado (oscr), que se encuentra en https://github.com/oeramosp/oscr (elaboración propia).

especial. Todo el código desarrollado se encuentra como repositorio público de git en https://github.com/oeramosp/oscr, y puede ser clonado a partir de allí (utilizando comandos básicos de git), o inspeccionado en línea. Una captura de pantalla del repositorio se muestra en la Figura 4.1. Además, el repositorio cuenta con integración a *travis* (`https://travis-ci.org/`), un servicio de integración continua que lo compila y ejecuta sobre Linux recién instalado para verificar dependencias y la funcionalidad general. El sistema ha sido enteramente implementado utilizando C++ pero con interfaces para Python, debido a que para prototipar movimiento es más sencillo y rápido usar un lenguaje de tipo script.

El sistema propuesto está diseñado para funcionar sobre Linux. No se utilizó MacOS ni ninguna versión de Windows como plataformas objetivo debido a que las principales aplicaciones en robótica corren en entornos Linux. Incluso tarjetas embebidas de bajo costo como Raspberry Pi, Jetson, BeagleBone, etc., corren Linux, y pueden ser utilizadas para el desarrollo de prototipos en robótica. De igual manera, las diversas herramientas usadas y descritas en la sección 4.1 tienen como plataforma objetivo principal a Linux. Además, el sistema desarrollado es completamente independiente de *middlewares*. Sin embargo, por facilidad para la interacción con modelos gráficos también se desarrolló una interface con ROS (*Robot Operating System*), como se detalla en la sección 4.5.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Contenido del Repositorio (*oscr*). El repositorio implementado, el cual se encuentra disponible online y cuya página principal se observa en la Figura 4.1 contiene las siguientes carpetas y archivos, los cuales han sido escritos de cero para esta tesis.

- *cmake*: contiene el archivo `FindQPOASES.cmake` que se utiliza para encontrar el lugar donde QPOases ha sido instalado.
- *doc*: contiene los archivos llamados `Doxyfile.in`, `Mainpage.dox`, `footer.html` y el archivo `treeview.dox` que determinan la configuración de cómo se generará la documentación con Doxygen. El archivo `Mainpage.dox` contiene el texto de la página principal de la documentación (Figura 4.2).
- *examples*: contiene las siguientes carpetas que contienen ejemplos de uso
  - *c++*: contiene archivos ejecutables de C++ que brindan ejemplos de la lectura y extracción de información del modelo del robot `ff6dof-model.cpp` y de control cinemático simple `ff6dof-kine-control.cpp`, junto con un archivo `CMakeLists.txt` que indica cómo compilar dichos archivos
  - *models*: contiene los siguientes modelos de robots, los cuales están escritos en formato URDF: `humanoid_example.urdf`, `ff6dof.urdf`, `sampleHumanoid.urdf`.
  - *python*: contiene scripts de Python que ejemplifican el uso de la interfaz de Python para leer el modelo del robot `ff6dof_model.py`, y para realizar control cinemático `ff6dof_kine_control.py`.
- *include/oscr*: contiene las cabeceras de las clases utilizadas organizadas como:
  - *model*: contiene las cabeceras de las clases que son utilizadas para la lectura del modelo del robot y la extracción básica de cinemática. Estos archivos son `robot-model.hpp`, `robot-model-pin.hpp`, `robot-model-rbdl.hpp`, `robot-model-ff6dof-hpp`.
  - *ik*: contiene las cabeceras de las clases que están directamente relacionadas con el control cinemático del robot. Los archivos incluidos son las tareas `kine-task.hpp`, `kine-task-pose.hpp`, `kine-task-posture.hpp`, y los esquemas de control cinemático, que son `osik-solver.hpp`, `osik-solver-HQP.hpp`, `osik-solver-NS.hpp` y el `osik-solver-WQP.hpp`, donde cada uno será descrito en la sección 4.3.
  - *tools*: contiene las cabeceras de herramientas adicionales que sirven como apoyo a todo el sistema: `math-utils.hpp`, `vector-utils.hpp`, `vector-logger.hpp`.

Todas estas cabeceras se pueden incluir con el archivo `oscr.hpp`

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

- *modules*: contiene un archivo `CMakeLists.txt` para la compilación de la interface de Python, un archivo `oscr.i` que especifica los archivos que serán pasados a swig para su uso en Python, `swig-eigen` que es una carpeta con la interface entre Eigen y Python, y una carpeta llamada `oscr`.
    - *oscr*: contiene los siguientes archivos escritos en Python que son utilizados para facilitar el uso desde Python ya que realizan varias de las conexiones del sistema de manera automática: `robot_base.py`, `utils.py`. Además, contiene archivos que facilitan la interface con ROS: `ros_robot.py`, `ros_kine_tasks.py`, `ros_kine_sim.py`. También contiene archivos para facilitar la publicación de tópicos en ROS (`ros_pubs.py`), para realizar logs (`ros_logs.py`, y para mostrar marcadores en RViz (`ros_markers`).
- *src*: contiene los archivos que contienen el código fuente de las clases implementadas. Los archivos de ayuda son: `math-utils.cpp`, `model-utils.cpp`, `vector-logger.cpp`, los archivos que implementan la cinemática básica del robot son: `robot-model.cpp`, que es una clase abstracta, `robot-model-pin.cpp`, `robot-model-rbdl.cpp`, y para el caso del robot de ejemplo: `robot-model-ff6dof.cpp`. Para las tareas se tiene: `kine-task.cpp`, `kine-task-pose.cpp`, `kine-task-posture.cpp`. Finalmente para los esquemas de control cinemático se tiene: `osik-solver.cpp`, `osik-solver-WQP.cpp`, `osik-solver-NS.cpp`, `osik-solver-HQP.cpp`. También esta carpeta contiene el archivo `CMakeLists.txt` que indica cómo compilar para generar la librería.
- `unitTests`: contiene archivos que permiten probar partes unitarias del sistema implementado: `testRobotModelRbdl.py`, `testRobotModelPin.py`, `testMathUtils.py`.
- .gitignore: es un archivo que le indica a `git` las extensiones que debe ignorar al momento de almacenar los archivos del repositorio.
- .travis.yml: es un archivo de configuración que indica al servidor de `travis` cómo realizar la integración continua del sistema, para verificar si existe algún problema de compilación o de dependencias externas.
- CMakeLists.txt: es el archivo de configuración de CMake que indica de manera global cómo compilar el sistema.
- INSTALL.md: contiene una guía paso a paso sobre cómo realizar la instalación, en el caso que no se tenga mucho conocimiento del uso de Linux.
- LICENSE: contiene el contenido de la licencia *open source* asociada con el sistema implementado, que en este caso es de tipo *GNU General Public License v3.0*.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

- README.md: es un archivo que contiene la información general sobre el sistema, así como una guía rápida de la instalación (sin mucho detalle).
- oscr.pc.make: contiene la configuración básica que será utilizada por *pkg-config* al momento de establecer dónde se instalará este paquete.

Utilizando el comando `git ls-files | xargs cat | wc -l` en el repositorio del sistema se obtiene que el sistema implementado contiene 17585 líneas de código en total. Debe tenerse en cuenta que este número de líneas solo aplica a este repositorio. Los repositorios adicionales, que fueron utilizados para obtener los resultados, añaden varias líneas más de código escrito.

4.2.1. Generalidades de la Implementación en C++

El sistema de generación cinemática de movimiento desarrollado en esta tesis se encuentra íntegramente implementado utilizando el lenguaje de programación C++ debido a que este lenguaje permite la elaboración de aplicaciones que pueden correr en tiempo real. Esto es importante dado que el sistema debe ser capaz de funcionar en tiempo real para poder controlar robots reales, los cuales presentan intervalos de control bastante pequeños, que típicamente van de 1 ms a 10 ms. Por este motivo se requiere evitar las complicaciones, lentitudes y sobre-cargas innecesarias que presentan los lenguajes script como Matlab o Python. Por otro lado, el uso de C++ para la implementación de algoritmos en robótica es casi estándar en la gran mayoría de universidades y laboratorios de investigación en esta área debido a su rapidez de ejecución, y es realizado en los propios sistemas embebidos del robot, los cuales usualmente corren alguna versión reciente de Linux. En la base misma del sistema desarrollado se hace uso extensivo de la biblioteca STL (*Standard Template Library*) de C++, así como de *Eigen* para el manejo de álgebra lineal (sección 4.1.1).

Compilación. Dado que el sistema propuesto es de tipo *open source* y su plataforma objetivo es Linux, se hace uso de *CMake* (`https://cmake.org`), el cual es una herramienta que permite la configuración del proyecto a través de un archivo llamado `CMakeLists.txt` y genera de manera automática un archivo de tipo Makefile con base en esta configuración. Es este archivo Makefile el que luego es utilizado de forma directa para la compilación, haciendo uso de algún compilador. Como compilador se utiliza g++, que es el compilador GNU de C++, y su uso no requiere de ninguna licencia. Así, en un terminal de Linux, para clonar y compilar el sistema se debe utilizar los siguientes comandos (donde se usa *CMake*):

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Listado 4.1: Comandos para clonar y compilar el código implementado en esta tesis

```
$ git clone https://github.com/oeramosp/oscr
$ cd oscr; mkdir build; cd build; cmake ..
$ make
```

Para mayor información sobre la compilación e instalación del paquete, así como el detalle del archivo `CMakeLists.txt` que se usa para la configuración de CMake, se puede revisar el repositorio de github. Los archivos `README.md`, y `INSTALL.md` proveen información detallada de compilación e instalación. La compilación y el funcionamiento con otros sistemas operativos como Windows o MacOS no ha sido comprobado dado que está fuera del alcance de esta tesis. Sin embargo, debido a que CMake es multi-plataforma, la adaptación a estos sistemas operativos requiere únicamente pequeñas modificaciones en los archivos de configuración CMakeLists.txt.

Manejo de Paquetes. En el sistema implementado se realiza el manejo de paquetes a través de *PkgConfig* (`http://pkg-config.freedesktop.org/`), el cual es un programa que recolecta información sobre las librerías instaladas en un sistema y realiza enlaces con estas librerías. Así, se tiene un archivo llamado oscr.pc.cmake que al instalarse se convierte en oscr.pc, y al añadir su camino a la variable de entorno `PKG_CONFIG_PATH`, el programa PkgConfig puede encontrar todas las dependencias. El contenido del archivo `oscr.pc.cmake` se muestra a continuación:

Listado 4.2: Contenido del archivo oscr.pc.cmake que contiene la configuración de pkg-config que indicará dónde se encuentra instalado el sistema implementado en esta tesis

```
prefix=@CMAKE_INSTALL_PREFIX@
libdir=@CMAKE_INSTALL_FULL_LIBDIR@
includedir=@CMAKE_INSTALL_FULL_INCLUDEDIR@

Name: Operational Space Control for Redundant Robots (OSCR)
Description: Whole-body Motion Generation
URL: github.com/oeramosp-oscr
Version: @PROJECT_VERSION@
Requires: eigen3, rbdl, pinocchio, qpoases
Conflicts:
Libs: -L${libdir} -loscr -Wl
Libs.private:
Cflags: -I${includedir}
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Puede observarse que el lugar donde se va a instalar la librería y los archivos de cabecera (*includes*) se encuentran dados por variables de *CMake* y son automáticamente generados una vez que se instala el sistema. Además, este archivo contiene las dependencias que se tiene, que en este caso son: eigen3, rbdl, pinocchio y qpoases.

Dependencias Externas. El detalle de cada una de las dependencias externas se describe en la sección 4.1. Dichas dependencias pueden ser descargadas o clonadas directamente desde las páginas o repositorios de sus respectivos proyectos, y luego instaladas una por una. Este proceso requiere cierta familiaridad con el manejo de paquetes en Linux y no es complicado para personas habituadas a este entorno. Sin embargo, para personas sin experiencia podría no ser tan directo. Por este motivo se provee un paquete llamado oscr-deps en el repositorio `https://github.com/oeramosp/oscr-deps`, que contiene los principales componentes de estas dependencias y permite su fácil compilación y enlace. La instalación de este paquete de dependencias externas se realiza mediante un script escrito para tal fin, cuyo uso se especifica en un archivo `README.md` del repositorio.

Implementación en C++. La implementación del sistema de generación de movimiento para robots redundantes se basa en clases de C++, utilizando el concepto de polimorfismo para la creación de una interface homogénea. Toda la base del sistema utiliza el *namespace* `oscr`, que a su vez es el nombre de la librería, con el fin de evitar conflictos de nombres en las funciones y como una buena práctica en C++. Otra buena práctica consiste en declarar explícitamente miembros tipo `const` cuando no van a modificarse las variables internas, y de usar variables privadas o protegidas dentro de las clases para el manejo de datos que sean utilizados continuamente durante toda la ejecución, y así evitar declaraciones *on-the-fly* que ralentizan la ejecución. En general, el estilo del código sigue muchos de los lineamientos dados en [49] para código C++ efectivo, así como las diversas recomendaciones del estilo de codificación de Google (`https://google.github.io/styleguide/cppguide.html`). El detalle de cada una de las clases creadas se muestra en la sección 4.3.

Documentación. Debido a que el sistema propuesto ha sido implementado utilizando diversas funciones, clases, y estructuras en C++, sería impráctico realizar y, sobretodo, actualizar manualmente la documentación de cada una de las funciones o métodos de las clases desarrolladas. Es por ello que con el fin de automatizar este proceso, se utilizó la herramienta llamada *Doxygen* (`http://www.doxygen.nl`), que es un generador de documentación para diversos lenguajes de programación, entre ellos C++. Se escogió Doxygen debido a que presenta una integración natural

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

FIGURA 4.2: Página principal de la documentación del sistema implementado, generada automáticamente de las cabeceras usando Doxygen (elaboración propia).

con C++ y genera documentación en formato html con hipervínculos entre funciones y al código mismo. La Figura 4.2 muestra la interfaz principal de la documentación creada con Doxygen. Como se observa, se dispone de algunos ejemplos y a la izquierda se puede ver cada una de las clases implementadas. Al hacer click en alguna de estas clases, el panel derecho mostrará la documentación asociada con esta clase.

Otra ventaja del uso de Doxygen es que permite la escritura de ecuaciones utilizando `LaTeX`, y actualiza las definiciones de las funciones cuando éstas cambian ya que la documentación es extraída directamente del código. Esto permite contar siempre con documentación actualizada. Un ejemplo de documentación de una función llamada `angularJacobian`, que es parte de `RobotModel` (como se detalla en la sección 4.3.1), se muestra en la Figura 4.3. Se puede observar que la descripción de la función utiliza variables matemáticas que han sido definidas usando `LaTeX`. Además, dado que esta es una función puramente virtual, Doxygen automáticamente enlaza a las instancias que implementan la función y a las funciones que hacen referencia a la misma.

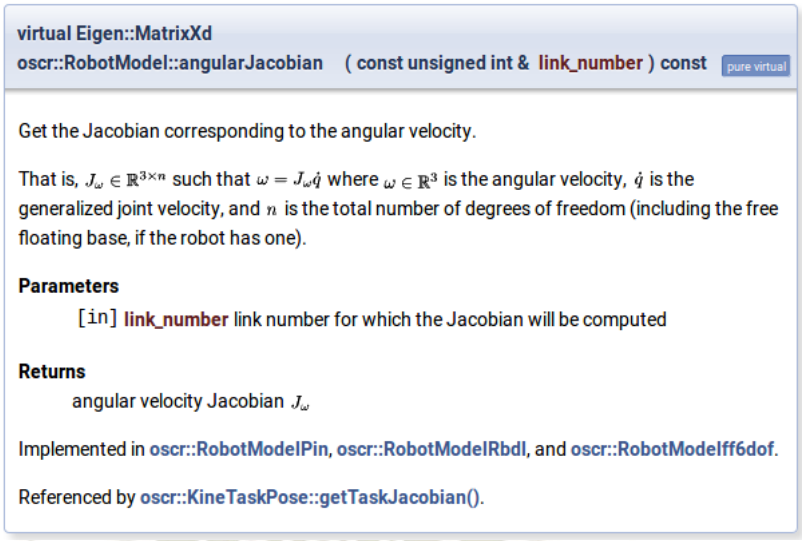
4.2.2. Generalidades de la Interfaz con Python

Como se ha mencionado, la base del sistema de generación de movimiento implementado en esta tesis usa C++. Así, el uso natural de estas funciones y clases implementadas puede ser realizado a través de ejecutables escritos en el mismo lenguaje. Sin embargo, la desventaja consiste en que C++

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA



virtual Eigen::MatrixXd
oscr::RobotModel::angularJacobian (const unsigned int & link_number) const pure virtual

Get the Jacobian corresponding to the angular velocity.

That is, $J_\omega \in \mathbb{R}^{3\times n}$ such that $\omega = J_\omega \dot{q}$ where $\omega \in \mathbb{R}^3$ is the angular velocity, $\dot{q}$ is the generalized joint velocity, and n is the total number of degrees of freedom (including the free floating base, if the robot has one).

Parameters

[in] **link_number** link number for which the Jacobian will be computed

Returns

angular velocity Jacobian J_ω

Implemented in oscr::RobotModelPin, oscr::RobotModelRbdl, and oscr::RobotModelff6dof.

Referenced by oscr::KineTaskPose::getTaskJacobian().

FIGURA 4.3: Ejemplo de documentación de una función (llamada *angularJacobian*, que calcula la parte angular del Jacobiano) usando Doxygen (elaboración propia).

no permite una interacción fácil con los resultados, es decir una experimentación o inspección de los valores obtenidos de manera interactiva. Además, para cada modificación, como por ejemplo una posición objetivo diferente, se debe recompilar todo el programa y ejecutarlo nuevamente, lo cual puede no ser muy práctico. Es posible desarrollar GUIs adicionales en C++ usando por ejemplo Qt, pero una interfaz gráfica no es el propósito de esta tesis. Alternativamente, se puede diseñar una interfaz con otro lenguaje que no tenga las limitaciones mencionadas. En esta tesis se optó por realizar una interfaz con Python.

La ventaja de Python es la interactividad que ofrece y el hecho que no necesita compilación cuando se realiza algún cambio dado que es un lenguaje tipo *script* que no se compila sino se interpreta. En particular, para la generación de movimiento en robótica donde típicamente se requiere la satisfacción de diferentes tareas cada una con diferentes especificaciones (cuyo objetivo muchas veces depende del estado actual del robot) es recomendable un lenguaje script que sea fácil de modificar, prototipar, y con el cual sea fácil interactuar. Además, Python es *open source*, y por tanto no se requiere la adquisición de licencias para su uso, lo cual es ventajoso a nivel de estudiantes y hasta de investigadores. La interfaz con Python fue desarrollada utilizando *swig*, y el procesamiento de matrices y vectores hace uso de *numpy*.

Swig. La interfaz del sistema desarrollado con Python fue implementada utilizando la herramienta llamada Swig (*Simplified Wrapper and Interface Generator*), disponible en `http://www.swig.org`, que es un compilador que conecta programas escritos en C y C++ con lenguajes de tipo script como Python. Swig usa las declaraciones que se encuentran en las cabeceras de C++ para automáticamente crear *wrappers* para el lenguaje destino, en este caso Python. Se utilizó Swig debido a la facilidad

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

de implementación, aunque otra alternativa pudo haber sido el soporte de Python que Boost ofrece. Para utilizar Swig es necesario crear un archivo con extensión `.i`, el cual especifica la configuración necesaria. En este caso se creó el archivo `oscr.i`, que se encuentra en `modules` del repositorio de github. Sin embargo, por defecto SWIG no soporta *Eigen* para el manejo de matrices, por lo cual se adaptó un archivo *eigen.i* (de *A. Barré. The Biomechanical Toolkit*) que expone las matrices Eigen de C++ a matrices NumPy en Python.

Numpy. Es una librería para Python que añade soporte para el manejo de grandes arreglos y matrices multi-dimensionales, así como una larga colección de funciones matemáticas, incluyendo operaciones de álgebra lineal, para operar con dichas matrices (`http://www.numpy.org`). En la interfaz con Python desarrollada se utilizó Numpy para manejar vectores y matrices, ya que su uso es fundamental en las especificaciones de las tareas cinemáticas descritas en esta tesis. Una de las ventajas de usar Numpy es que es casi la librería estándar de Python para este tipo de operaciones y existe una amplia documentación y tutoriales que explican su funcionamiento y uso.

4.2.3. Características Principales del Sistema Desarrollado

Algunas de las características que presenta la implementación realizada en esta tesis del sistema de generación de movimiento para robots redundantes y humanoides basado en control cinemático son las siguientes.

Distribución como Software Libre. Todo el código fuente desarrollado se encuentra disponible en un repositorio de github (`https://github.com/oeramosp/oscr`) y puede ser accedido por cualquier persona en cualquier lugar. El código se distribuye bajo la licencia permisiva *GNU General Public License v3* que cumple con los estándares de la *Free Software Foundation*.

Generalidad. El sistema desarrollado puede ser aplicado a cualquier tipo de robot poli-articulado compuesto por eslabones rígidos, siempre que las características del robot estén especificadas en un archivo con formato URDF. El sistema está orientado principalmente a robots humanoides, sin importar su complejidad o el número de grados de libertad, pero puede ser utilizado por otros tipos de robots redundantes con base flotante, como robots cuadrúpedos. Además, es también posible su uso con robots manipuladores de base fija, sean estos redundantes o no.

Independencia del *Middleware*. El sistema desarrollado no depende de ningún middleware específico y puede ser fácilmente integrado en cualquier arquitectura robótica compleja. Depende únicamente de algunas librerías independientes y open source (detalladas en la sección 4.1) que

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

implementan algoritmos del estado del arte en cinemática básica y resolución de problemas de optimización numérica. La actual arquitectura también permite el fácil desarrollo de plugins para otras librerías robóticas ya que el código se compila utilizando *CMake*, una herramienta común en entorno Linux pero extensible también a otras plataformas.

Extensibilidad. La forma modular en la que está implementado el sistema de control cinemático permite su fácil extensión a través del desarrollo de nuevos módulos. Por ejemplo, se puede desarrollar nuevas tareas que serán inmediatamente compatibles con el sistema. De igual manera, se puede incrementar los métodos de solución cinemática para probar nuevos algoritmos. Además de esto, es posible la extensión a control dinámico utilizando la misma estructura, aunque esto no es ya parte de la presente tesis.

Documentación del Código. Cada una de las clases y de los métodos utilizados se encuentran debidamente documentados utilizando *Doxygen*, una herramienta ampliamente utilizada para la generación de documentación en formato HTMl. Así, una vez compilado el código fuente, es posible generar automáticamente la documentación y tener una idea clara de qué es lo que hace exactamente cada parte del sistema.

Facilidad de Uso. Toda la librería desarrollada está implementada en C++, y por tanto su uso puede ser a través de programas escritos en C++ (se provee ejemplos en el código). Para facilitar más el uso, se desarrolló una interfaz para Python, un lenguaje script ampliamente utilizado en robótica y en otras áreas como aprendizaje de máquina o tratamiento de grandes cantidades de datos. La ventaja de Python es que no se requiere compilar el programa para ver el resultado, y se puede trabajar de manera interactiva, logrando una mejor comprensión y prototipado de cada etapa.

Eficiencia Computacional. El sistema se basa en librerías altamente eficientes y numéricamente estables implementadas en C++. Además, el desarrollo mismo del las funciones y clases que implementan el sistema se encuentra escrito en C++, siguiendo criterios eficaces de implementación computacional, lo cual permite ejecutar ciclos de control menores a 10 ms en pruebas experimentales.

4.3. Estructura Detallada del Sistema

El esquema general del sistema de control cinemático a nivel principalmente conceptual se muestra en la Figura 3.3, donde se puede observar de manera gráfica cómo la generación de movimiento se basa en las diferentes tareas que son especificadas como entradas al sistema de control cinemático.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Por claridad dicho esquema se concentra en la parte fundamental y omite algunos detalles de la implementación. Esta sección presenta dichos detalles, así como la relación con cada una de las librerías externas utilizadas y descritas en la sección 4.1.

4.3.1. Modelo del Robot y Cinemática Básica.

La descripción de los parámetros físicos del robot está dada mediante el formato URDF, como se mencionó en la sección 4.1.2. Así, cada robot tiene un archivo `*.urdf` asociado que lo describe por completo. Este archivo es leído por las librerías RBDL o Pinocchio, las cuales a su vez hacen uso de la librería `urdfdom`, como se muestra esquemáticamente en la Figura 4.4. Es posible utilizar cualquiera de estas dos librerías, y el uso de ambas se debe al hecho de poder verificar los resultados de una librería con la otra al momento de la implementación. En la práctica, con el sistema funcionando adecuadamente, resulta exactamente igual utilizar cualquiera de las dos. Estas librerías leen el modelo URDF y además requieren como entrada la configuración articular del robot correspondiente a las articulaciones actuadas $\boldsymbol{q}_a$. Luego, con base en la configuración actual $\boldsymbol{q}_a$, estas librerías realizan el cálculo de la cinemática directa de algún punto operacional con respecto a la base fija del robot, y retornan tanto su orientación y/o posición ${}^b\boldsymbol{x}$ como el Jacobiano geométrico ${}^bJ$. En esta notación, el superíndice a la izquierda ^{b} indica que estos valores se calculan con respecto a la base, que en estas librerías se asume como fija.

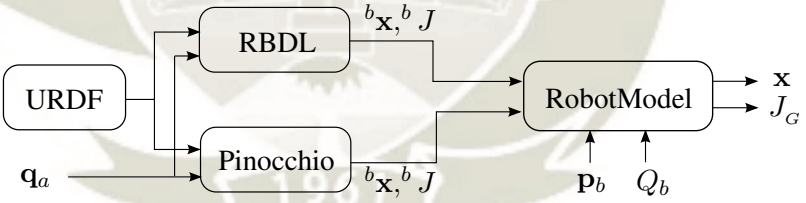


FIGURA 4.4: Esquema utilizado para la obtención de la cinemática directa y del Jacobiano, a partir del modelo URDF del robot usando RBDL y Pinocchio (elaboración propia).

Luego, la clase *RobotModel* (en realidad, sus clases derivadas) convierten estos elementos que se encuentran en el sistema de la base, al sistema inercial de referencia inercial utilizando la posición de la base $\boldsymbol{p}_b$ y la orientación de la base dada por un cuaternión Q_b. Así, la salida final es el valor de la cinemática directa de un punto operacional $\boldsymbol{x}$ y del Jacobiano geométrico asociado con dicho punto J_G con respecto al sistema inercial, considerando la posición y orientación de la base flotante. Esto es fundamental para generar movimiento para robots con base flotante. A continuación se presenta los detalles de la implementación de esta clase y de sus clases derivadas.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

a. Clase base *RobotModel*

Esta clase encapsula el cálculo de la cinemática directa y del Jacobiano del robot dado su modelo URDF como entrada y la configuración articular actual $\boldsymbol{q}$, que incluye la parte actuada $\boldsymbol{q}_a$ y la parte subactuada correspondiente a la base del robot $\boldsymbol{x}_b$. De manera conceptual, encapsula todo el diagrama mostrado en la Figura 4.4. Debido a que la implementación de la cinemática con respecto a la base del robot se realiza mediante dos diferentes librerías, se decidió implementar una clase base para uniformizar la interfaz que se utilizará, de tal modo que el uso de una u otra librería como *backend* sea transparente al momento de generar movimiento. Más aún, las clases que implementan las tareas y los esquemas de control cinemático utilizarán las funciones miembro de la clase base sin distinguir si se usa una librería o la otra, facilitando la generalidad de la implementación.
La clase *RobotModel* sirve de base para 2 clases derivadas principales de propósito general, denominadas *RobotModelRbdl* y *RobotModelPin*, las cuales utilizan las librerías RBDL y Pinocchio, respectivamente. Además, existe una tercera clase derivada llamada *RobotModelff6dof*, la cual calcula de manera analítica el modelo de un robot de seis grados de libertad de base flotante. Esta última clase solo se utiliza como prueba o verificación de los cálculos analíticos con los cálculos numéricos (se brinda mayor información de este modelo en la sección 5.2). La Figura 4.5 muestra un diagrama de la clase base con sus derivadas y fue generada usando Doxygen.

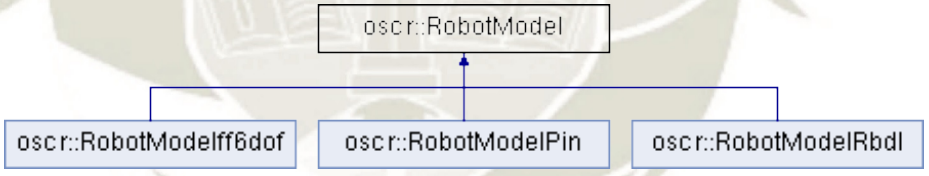


FIGURA 4.5: Diagrama de herencia para el modelo del robot y la cinemática directa, los cuales se implementan en la clase *RobotModel* y sus derivadas (elaboración propia).

El código completo correspondiente a la declaración de esta clase base puede verse en el apéndice C.1, y en el repositorio de github mencionado anteriormente se encuentra con el nombre de `include/oscr/model/robot-model.hpp` (la implementación de la clase se realiza en el archivo `src/robot-model.cpp`). Como se puede observar en el código, esta clase contiene varias funciones miembro virtuales, las cuales son implementadas en las clases derivadas. La declaración de algunas funciones miembro importantes se muestra en el Listado 4.3. Las tres funciones mostradas son puramente virtuales y por tanto su implementación se encuentra en cada clase derivada y no en la clase base. La función `loadURDF` toma como entrada una cadena de caracteres que especifica la ubicación del archivo que contiene el modelo urdf del robot, así como una indicación de si el robot posee una base flotante o no. De este modo, el sistema desarrollado permite también la generación de movimiento para robots manipuladores. La función `updateJointConfig` permite actualizar la

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

variable privada interna a la clase que contiene el valor articular del robot $\boldsymbol{q}$, el cual consta de la parte actuada y de la base flotante. Las funciones `linkPose` y `geometricJacobian` permiten obtener el valor de la cinemática directa $\boldsymbol{x}$ y del Jacobiano geométrico J_G para un punto operacional especificado mediante el nombre del eslabón o mediante su número de identificación. El parámetro `local_pos` es un vector que se usa para especificar cierto offset en la posición del punto deseado con respecto a la posición del eslabón, pero generalmente es considerado como cero. Existen más funciones que permiten obtener solo parte de la información anterior, es decir, solo la posición, o la orientación, o la parte correspondiente a la velocidad lineal o angular del Jacobiano.

Listado 4.3. Algunas funciones miembro de la clase RobotModel, las cuales cargan el modelo del robot y brindan información de su cinemática (elaboración propia).

```
virtual bool loadURDF(const std::string& model_name,
                      const bool& has_floating_base = true,
                      const bool& verbose = false) = 0;

virtual void updateJointConfig(const Eigen::VectorXd& q) = 0;

virtual Eigen::VectorXd linkPose(const std::string& link_name,
                                 const Eigen::Vector3d& local_pos =
                                       Eigen::Vector3d::Zero()) const = 0;

virtual Eigen::MatrixXd geometricJacobian(const unsigned int& link_number,
                                          const Eigen::Vector3d& local_pos =
                                                Eigen::Vector3d::Zero()) const = 0;
```

Esta clase también almacena información adicional del modelo del robot, como los límites mínimos y máximos de posición para cada una de las articulaciones, los cuales no deben ser sobrepasados. Además, se almacena información sobre los límites de velocidad de cada articulación, los nombres de las articulaciones y de los eslabones, así como sus identificadores numéricos (IDs).

b. Clase derivada *RobotModelRbdl*

Esta clase deriva de manera pública de *RobotModel* e internamente utiliza la librería RBDL para el cálculo de la cinemática con respecto a una base que se asume fija. La clase se encuentra definida en el archivo llamado `include/oscr/model/robot-model-rbdl.hpp` y su implementación en el archivo `src/robot-model-rbdl.cpp`. Internamente utiliza una instancia de tipo `Model`, propia de RBDL, para almacenar la información concerniente al modelo del robot. Esta instancia, que en realidad es un puntero, se denomina `rmodel_`, es privada a la clase, y su definición se realiza como: `RigidBodyDynamics::Model* rmodel_`. Al momento de utilizar la función `loadURDF` se realiza la lectura de los elementos del modelo URDF, los cuales son almacenados en `rmodel_`. El detalle de la implementación de esta función no se incluyó aquí debido a que es altamente dependiente de

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

la forma como funciona RBDL y no aporta mayor interés desde el punto de vista de esta tesis. Sin embargo, puede ser encontrado en el repositorio de github del sistema.

Luego de tener el modelo del robot, la librería RBDL proporciona la posición y orientación ${}^{b}\boldsymbol{x}$ con respecto a la base del robot, que se asume fija hasta este momento, y esta clase las convierte al sistema inercial o global agregando el efecto de la base flotante $\boldsymbol{x}_b$ haciendo uso de (3.68) y (3.72). La función miembro que implementa esta conversión, y que permite obtener $\boldsymbol{x}$ en el sistema inercial, se denomina `linkPose`, y en este caso depende de algunas funciones brindadas por RBDL. La implementación de esta función se muestra en el Listado 4.4 a continuación.

Listado 4.4. Función *linkPose*, implementada en esta tesis para el cálculo de la posición y la orientación de un eslabón del robot (elaboración propia)

```
Eigen::VectorXd RobotModelRbdl::linkPose(const unsigned int& link_number,
                                         const Eigen::Vector3d& local_pos) const
{
  Eigen::VectorXd lpose;
  lpose.resize(7);
  Eigen::Matrix3d Rlink;
  if (has_floating_base_)
  {
    lpose.head(3) = CalcBodyToBaseCoordinates(*rmodel_, q_.tail(ndofActuated()),
                                              link_number, local_pos, true);
    Rlink = CalcBodyWorldOrientation(*rmodel_, q_.tail(ndofActuated()),
                                     link_number, true).transpose();
    Eigen::Matrix3d Rbase;
    Rbase = quaternionToRotation(q_.segment(3,4));
    lpose.head(3) = Rbase*lpose.head(3) + q_.head(3);
    Rlink = Rbase*Rlink;
  }
  else
  {
    lpose.head(3) = CalcBodyToBaseCoordinates(*rmodel_, q_, link_number, local_pos, true);
    Rlink = CalcBodyWorldOrientation(*rmodel_, q_, link_number, true).transpose();
  }
  lpose.tail(4) = rotationToQuaternion(Rlink);
  return lpose;
}
```

En el código anterior, la función `quaternionToRotation` ha sido implementada usando (2.15), y la funciń `rotationToQuaternion` usando (2.16), como se muestra en el apéndice C.4. Puede notarse además, el uso exhaustivo que se hace de Eigen para el manejo de matrices y vectores. Se dispone además de funciones miembro que retornan solo la posición o solo la orientación de un punto operacional, normalmente asociado a un eslabón del robot, y su implementación es solo una fracción de la función mostrada. Estas funciones son útiles cuando la tarea consiste en solamente controlar la posición o la orientación, de tal manera que se evita realizar cálculos innecesarios.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Por otro lado, la librería RBDL provee el Jacobiano geométrico ${}^{b}J$ con respecto a la base del robot, que hasta este momento se asume fija, usando solo valores de las articulaciones actuadas. El sistema de referencia es luego cambiado con respecto al sistema inercial utilizando (2.37). Además del cambio de referencia, se añade el efecto del movimiento de la base flotante sobre el punto operacional en cuestión utilizando (3.21). Así, se obtiene finalmente el Jacobiano geométrico $J_G(\boldsymbol{q})$, y el siguiente fragmento de código muestra la función `geometricJacobian` que realiza el procedimiento descrito.

Listado 4.5. Función implementada para el cálculo del Jacobiano (elaboración propia)

```
Eigen::MatrixXd RobotModelRbdl::geometricJacobian(const unsigned int& link_number,
                                                  const Eigen::Vector3d& local_pos) const
{
  Eigen::MatrixXd J, Jtmp;
  if (has_floating_base_) {
    Eigen::Vector3d xlink; Eigen::Matrix3d Rbase, Rlink; Eigen::MatrixXd T;
    Rbase = quaternionToRotation(q_.segment(3,4));
    xlink = CalcBodyToBaseCoordinates(*rmodel_, q_.tail(this->ndofActuated()),
                                      link_number, local_pos, true);
    xlink = Rbase*xlink + q_.head(3);
    T.resize(3,4);
    T <<
      -2.0*q_(4),  2.0*q_(3), -2.0*q_(6),  2.0*q_(5),
      -2.0*q_(5),  2.0*q_(6),  2.0*q_(3), -2.0*q_(4),
      -2.0*q_(6), -2.0*q_(5),  2.0*q_(4),  2.0*q_(3);
    Jtmp.resize(6, this->ndofActuated()); Jtmp.setZero();
    CalcBodySpatialJacobian(*rmodel_, q_, link_number, Jtmp, false);
    Rlink = quaternionToRotation(linkOrientation(link_number));
    J.resize(6, this->ndof()); J.setZero();
    J.topLeftCorner(3,3) = Eigen::Matrix3d::Identity();
    J.block(0,3,3,4) = skew(q_.head(3)-xlink)*T;
    J.topRightCorner(3,this->ndofActuated()) = Rlink*Jtmp.bottomRows(3);
    J.block(3,3,3,4) = T;
    J.bottomRightCorner(3,this->ndofActuated()) = Rlink*Jtmp.topRows(3);
  }
  else {
    J.resize(6, this->ndof()); Jtmp.resize(6, this->ndof()); Jtmp.setZero();
    CalcBodySpatialJacobian(*rmodel_, q_, link_number, Jtmp, true);
    Eigen::Matrix3d R;
    R = quaternionToRotation(linkOrientation(link_number));
    J.topRows(3) = R*Jtmp.bottomRows(3);
    J.bottomRows(3) = R*Jtmp.topRows(3);
  }
  return J;
}
```

Existen, además, funciones que permiten obtener la parte correspondiente solamente a la velocidad angular y solamente a la velocidad lineal, y son útiles para no realizar cálculos innecesarios en el control cinemático. Para facilidad de uso también se almacena la cantidad de articulaciones que tiene el robot.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

c. Clase derivada *RobotModelPin*

Esta clase también hereda *RobotModel* de manera pública, pero internamente utiliza la librería Pinocchio para realizar el cálculo de la cinemática básica. La definición de esta clase se encuentra en la cabecera `include/oscr/model/robot-model-pin.hpp`, y su respectiva definición en el archivo `src/robot-model-pin.cpp`. Internamente utiliza dos estructuras de datos llamadas *Model* y *Data*, que se encuentran dentro del *namespace* llamado `se3` y que son declaradas como: `se3::Model* pmodel_`, y `se3::Data* data_`. Estas estructuras contendrán el modelo del robot y los datos asociados con su cinemática directa y Jacobiano, luego de llamar a la función `loadURDF` que lee los parámetros del robot a partir del modelo URDF. Se debe notar que al momento de cargar el modelo, se le indica a Pinocchio si se tiene una base flotante o no, dado que esta librería sí considera la base flotante (aunque con un sistema de referencia diferente).

Luego de tener el modelo del robot en formato de Pinocchio, la función `linkPose` permite el cálculo de la cinemática directa; es decir, de la posición y orientación de algún punto operacional a partir de la configuración articular. Esta función es de tipo `const` debido a que no modifica ningún miembro interno de la clase y solo retorna el valor calculado sin almacenarlo. La traslación se obtiene directamente de la librería y, por su parte, la rotación se convierte de matriz de rotación a cuaternión.

Listado 4.6. Función para obtener la posición y orientación (elaboración propia)

```
Eigen::VectorXd oscr::RobotModelPin::linkPose(const unsigned int& link_number,
                                              const Eigen::Vector3d& local_pos) const
{
  Eigen::VectorXd lpose(7);
  lpose.head(3) = data_->oMi[link_number].translation();
  lpose.tail(4) = rotationToQuaternion(data_->oMi[link_number].rotation());
  return lpose;
}
```

Para el cálculo del Jacobiano geométrico, se debe tener en cuenta que el Jacobiano retornado por la librería Pinocchio se encuentra expresado con respecto al sistema mismo del eslabón y debe ser transformado con respecto a la base. La estructura de este Jacobiano es:

$$J_p = \begin{bmatrix} A & B & C \\ 0 & D & E \end{bmatrix} = \begin{bmatrix} {}^{L}\boldsymbol{R}_{W}\,{}^{W}\boldsymbol{R}_{B} & {}^{L}\boldsymbol{R}_{W}\left({}^{W}\boldsymbol{p}_{B} - {}^{W}\boldsymbol{p}_{L}\right)^{\wedge}\,{}^{W}\boldsymbol{R}_{B} & {}^{L}J_{lin} \\ 0 & {}^{W}\boldsymbol{R}_{B} & {}^{L}J_{ang} \end{bmatrix} \tag{4.1}$$

asumiendo que la velocidad de la parte flotante está dada por ${}^{B}\boldsymbol{\xi} = \begin{bmatrix} {}^{B}\boldsymbol{v} & {}^{B}\boldsymbol{\omega} \end{bmatrix}^{T}$, con respecto a la base, que $\{\mathcal{W}\}$ representa el sistema inercial y que $\{\mathcal{L}\}$ representa el sistema del eslabón (*link*). Los valores A, B, C, D, E solo se muestran como conveniencia de notación. Para expresarlo con respecto al sistema de la base, en `RobotModelPin` a este Jacobiano se le aplica la siguiente

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

transformación

$$J = \begin{bmatrix} {}^{B}\boldsymbol{R}_{W}\,{}^{W}\boldsymbol{R}_{L}\,A & {}^{W}\boldsymbol{R}_{L}\,B\,{}^{B}\boldsymbol{R}_{W}\,T & {}^{W}\boldsymbol{R}_{L}\,C \\ 0 & T & {}^{W}\boldsymbol{R}_{L}\,E \end{bmatrix} \tag{4.2}$$

donde T es la matriz (3.15) obtenida a partir del cuaternión de la base flotante, y convierte la razón de cambio de este cuaternión en velocidad lineal y angular. El código que implementa lo anterior es el siguiente:

Listado 4.7. Implementación de la función para el cálculo del Jacobiano geométrico del robot que posee una base flotante (elaboración propia).

```
Eigen::MatrixXd oscr::RobotModelPin::geometricJacobian(const unsigned int& link_number,
                                                       const Eigen::Vector3d& local_pos) const
{
  se3::Data::Matrix6x J(6, pmodel_->nv); J.fill(0);
  se3::getJacobian<se3::LOCAL>(*pmodel_, *data_, link_number, J);
  if (has_floating_base_)
  {
    Eigen::MatrixXd Jg; Jg.resize(6, this->ndof()); Jg.setZero();
    Eigen::Matrix3d Rbase; Rbase = quaternionToRotation(q_.segment(3,4));
    Jg.topLeftCorner(3,3) =
      Rbase.transpose()*data_->oMi[link_number].rotation()*J.block(0,0,3,3);
    Jg.topRightCorner(3,ndofActuated()) =
      (data_->oMi[link_number].rotation())*J.block(0,6,3,ndofActuated());
    Jg.bottomRightCorner(3,ndofActuated()) =
      (data_->oMi[link_number].rotation())*J.block(3,6,3,ndofActuated());
    Eigen::MatrixXd T; T.resize(3,4);
    T <<
      -2.0*q_(4),  2.0*q_(3), -2.0*q_(6),  2.0*q_(5),
      -2.0*q_(5),  2.0*q_(6),  2.0*q_(3), -2.0*q_(4),
      -2.0*q_(6), -2.0*q_(5),  2.0*q_(4),  2.0*q_(3);
    Jg.block(0,3,3,4) =
      data_->oMi[link_number].rotation()*J.block(0,3,3,3)*Rbase.transpose()*T;
    Jg.block(3,3,3,4) = T;
    return Jg;
  }
  else
  {
    J.topRows(3) = (data_->oMi[link_number].rotation())*J.topRows(3);
    J.bottomRows(3) = (data_->oMi[link_number].rotation())*J.bottomRows(3);
    return J;
  }
}
```

4.3.2. Arquitectura de Tareas

Las tareas constituyen los bloques de construcción para el movimiento debido a que especifican los objetivos en el espacio operacional o espacio de la tarea. La Figura 4.6 muestra la arquitectura de una tarea cinemática genérica. Las entradas son tango la configuración deseada de la tarea $\boldsymbol{x}_d$,

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

como la configuración articular actual completa $\tilde{\boldsymbol{q}}$ medida por el robot. Esta configuración articular medida ingresa al bloque denominado *RobotModel*, que es el objeto descrito en la sección 4.3.1, y que en realidad contiene todo lo mostrado en la Figura 4.4. Usando este bloque de cinemática se obtiene la configuración actual de la tarea, $\boldsymbol{x}$, y el Jacobiano geométrico J_G, ambos considerando la base flotante del robot, en caso que la tuviera.

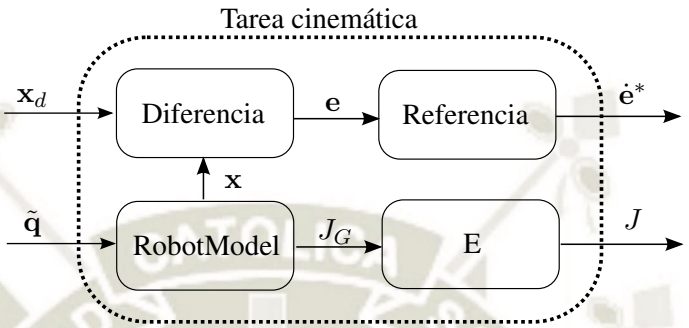


FIGURA 4.6: Arquitectura de una tarea cinemática genérica, donde la entrada es el valor deseado y la configuración actual del robot, y la salida es el comportamiento de referencia en velocidad, y el Jacobiano de la tarea (elaboración propia).

El valor actual de la tarea $\boldsymbol{x}$ junto con el valor deseado $\boldsymbol{x}_d$ se utilizan luego para calcular el error $\boldsymbol{e}$ que define la tarea. Este error puede ser una simple resta o puede ser una operación más compleja, por ejemplo para el error de orientación, y se describe con mayor detalle en la sección 3.5. Este error ingresa a un bloque que lo utiliza para generar la referencia deseada $\dot{\boldsymbol{e}}^*$ según la ley dada en (3.25). Es posible escoger la ganancia λ dentro de este bloque como constante o como una ganancia variable, según (3.26). Por otro lado, el Jacobiano de la tarea J, que también es salida de la tarea, es obtenido a partir de J_G de acuerdo con la sección 3.5. El valor de la matriz E, la cual depende del cuaternión de la base, es útil para esta relación en el caso de algunas tareas. A continuación se presenta el detalle de la clase base que lo implementa y sus clases derivadas.

a. Clase base *KineTask*

A nivel de implementación en C++, la estructura mostrada en la Figura 4.6 se encuentra encapsulada en una clase base llamada *KineTask*, debido a que se trata de tareas cinemáticas. Algunos miembros de esta clase son puramente virtuales mientras que otros no, ya que son comunes a todas las tareas. Esta clase sirve de base para las clases derivadas llamadas *KineTaskPose*, que es utilizada para las tareas de posición y/o orientación; *KineTaskPosture*, que se usa para las tareas de postura; y *KineTaskCoM*, que implementa tareas de control del centro de masa de un robot con base flotante. La Figura 4.7 muestra un diagrama de herencia de estas clases.

Si se desease añadir una nueva tarea cinemática para controlar algo específico, esta tarea debe heredar la clase base *KineTask*. Así, heredando de la clase base, es sencillo crear nuevas tareas

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

FIGURA 4.7: Diagrama de herencia para las clases que implementan tareas cinemáticas, basadas en *KineTask* y sus derivadas (elaboración propia).

que expandan la funcionalidad del sistema implementado, manteniendo toda la interfaz, siempre que se defina el error específico de la nueva tarea y su Jacobiano asociado. La definición completa de la clase se muestra en el apéndice C.2 y todo el código se encuentra disponible en `include/oscr/ik/kine-task.hpp` del repositorio de github. Algunas funciones importantes se muestran en el Listado 4.8.

Listado 4.8. Algunas funciones miembro importantes de KineTask (elaboración propia).

```
virtual void getSensedValue(Eigen::VectorXd& xsensed) const = 0;
virtual void getTaskJacobian(Eigen::MatrixXd& Jacobian) const = 0;
virtual void getDerivError(Eigen::VectorXd& de) = 0;
void updateJointConfig(const Eigen::VectorXd& q);
```

Estas funciones se utilizan para obtener el valor sensado de la tarea (que podría usar cinemática directa o alguna otra forma de cálculo), el Jacobiano de la tarea, y la derivada del error, respectivamente. La última función miembro se utiliza para actualizar la configuración articular interna de la clase. Además de las funciones mostradas, también se tiene funciones miembro para asignar y leer el nombre de la tarea, establecer la ganancia constante o adaptativa, de acuerdo con (3.26), establecer el valor deseado, obtener el valor deseado, establecer el peso de la tarea (útil cuando las tareas tienen ponderación), entre otras.

b. Clase derivada *KineTaskPose*

Esta clase deriva de *KineTask* y contiene la definición de una tarea de posición, de orientación, o de ambas a la vez. Se implementa en `src/kine-task-pose.cpp` del repositorio de github del sistema implementado en esta tesis. Para seleccionar cada opción con facilidad, se creó una enumeración, llamada `PoseType` que las define de manera clara como se muestra en el Listado 4.9.

Listado 4.9. Enumeración para indicar el tipo de tarea de posición, orientación, o ambas.

```
enum PoseType{ POSITION=0,
               ORIENTATION,
               FULLPOSE };
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Utilizando `PoseType` se puede determinar exactamente el tipo de tarea que se desea ejecutar. La implementación de la función que obtiene el valor sensado de la tarea, y que se basa en las clases derivadas de `RobotModel`, llamada `getSensedValue` se muestra en el Listado 4.10.

Listado 4.10. Implementación de la función, dentro de la clase KinePose, para obtener el valor sensado de posición y/o orientación (elaboración propia)

```
void KineTaskPose::getSensedValue(Eigen::VectorXd& xsensed) const
{
  // Tarea de posicion
  if (type_ == POSITION)
  {
    Eigen::VectorXd position = rmodel_->linkPosition(linkNum_, localpos_);
    xsensed.resize(selectedIndices_.size());
    for(unsigned int i=0; i<selectedIndices_.size(); ++i)
      xsensed(i) = position(selectedIndices_.at(i));
  }
  // Tarea de orientacion
  if (type_ == ORIENTATION)
    xsensed = rmodel_->linkOrientation(linkNum_);
  // Tarea de posicion y orientacion
  if (type_ == FULLPOSE)
    xsensed = rmodel_->linkPose(linkNum_, localpos_);
}
```

Se puede observar que según el tipo de tarea se extrae la posición, orientación, o ambas del modelo del robot, contenido en `rmodel_`, el cual es de tipo `RobotModel`. Así, la tarea es independiente de si el modelo usa RBDL o Pinocchio, y esa es una ventaja de usar el polimorfismo en este caso. Además, se puede notar que para la tarea de posición se puede escoger algunos de los ejes de movimiento de forma flexible.

La implementación de la función miembro que calcula la derivada del error, llamada `getDerivError` se muestra en el Listado 4.11. Esta función es importante para la determinación del comportamiento de referencia, diseñado para seguir una exponencial decreciente, el cual será utilizado luego por el controlador cinemático para generar la referencia deseada. Esta referencia será la entrada al robot controlado en posición o en velocidad. Se puede nuevamente observar que la implementación depende del tipo de tarea que se haya escogido: posición, orientación, o ambas a la vez. En esta función se implementa (3.25) como comportamiento de referencia usando ganancias adecuadas, y el error se calcula como se muestra en (3.67) para la posición, y como se muestra en (3.71) para la orientación. Debe notarse además, que la llamada a `getGain` puede devolver una ganancia constante o variable según como se haya definido ésta.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Listado 4.11. Implementación de la función getDerivError que calcula el comportamiento de referencia de la derivada del error (elaboración propia).

```
void KineTaskPose::getDerivError(Eigen::VectorXd& de)
{
  Eigen::VectorXd xsensed;
  getSensedValue(xsensed);
  // Tarea de posicion
  if (type_ == POSITION)
  {
    error_ = xdes_-xsensed;
    de = getGain(error_.norm())*error_;
  }
  // Tarea de orientacion
  else if (type_ == ORIENTATION)
  {
    error_(0) = xdes_(0)*xsensed(0) + xdes_.tail(3).transpose()*xsensed.tail(3) - 1.0;
    error_.tail(3) = xsensed(0)*xdes_.tail(3) - xdes_(0)*xsensed.tail(3)
                   - skew(xdes_.tail(3))*xsensed.tail(3);
    de = getGain(error_.norm())*error_;
  }
  // Tarea de posicion y de orientacion
  else if (type_ == FULLPOSE)
  {
    de.resize(7);
    error_.head(3) = xdes_.head(3)-xsensed.head(3);
    de.head(3) = getGain(error_.head(3).norm())*error_.head(3);
    error_(3) = xdes_(3)*xsensed(3) + xdes_.tail(3).transpose()*xsensed.tail(3) - 1.0;
    error_.tail(3) = xsensed(3)*xdes_.tail(3) - xdes_(3)*xsensed.tail(3)
                   - skew(xdes_.tail(3))*xsensed.tail(3);
    de.tail(4) = getGain(error_.tail(4).norm())*error_.tail(4);
  }
}
```

El cálculo del Jacobiano de la tarea de posición y orientación implica convertir el Jacobiano geométrico en un Jacobiano de la tarea, a través del uso de la transformación lineal adecuada. Una alternativa podría ser la derivación numérica, pero en esta implementación se prefiere el enfoque analítico. El código implementado para esta función, llamada `getTaskJacobian` se muestra en el Listado 4.12. Para el caso de la posición, dado que se usa coordenadas Cartesianas, la transformación es simplemente la identidad. Para el caso de la orientación, es necesario implementar (3.73), que implica el cálculo de la matriz T asociada a la orientación de la base del robot. Se puede observar además, que para el caso de posición y orientación se realiza la implementación de ambas partes a la vez. Se realizó la separación de los 3 casos mostrados para tener más eficiencia computacional evitando cálculos innecesarios.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Listado 4.12. Implementación de la función getTaskJacobian, la cual calcula el Jacobiano de la tarea de posición y/o orientación (elaboración propia).

```
void KineTaskPose::getTaskJacobian(Eigen::MatrixXd& Jacobian) const
{
  if (type_ == POSITION) {
    Eigen::MatrixXd J = rmodel_->linearJacobian(linkNum_, localpos_);
    Jacobian.resize(selectedIndices_.size(), this->ndof_);
    for(unsigned int i=0; i<selectedIndices_.size(); ++i)
      Jacobian.row(i) = J.row(selectedIndices_.at(i));
  }
  else if (type_ == ORIENTATION) {
    Eigen::MatrixXd Jang;
    Jang = rmodel_->angularJacobian(linkNum_);
    Eigen::VectorXd quat; getSensedValue(quat);
    Eigen::MatrixXd Tinv; Tinv.resize(4,3);
    Tinv << -0.5*quat(1), -0.5*quat(2), -0.5*quat(3),
             0.5*quat(0),  0.5*quat(3), -0.5*quat(2),
            -0.5*quat(3),  0.5*quat(0),  0.5*quat(1),
             0.5*quat(2), -0.5*quat(1),  0.5*quat(0);
    Jacobian = Tinv*Jang;
  }
  else if (type_ == FULLPOSE) {
    Eigen::MatrixXd Jg;
    Jg = rmodel_->geometricJacobian(linkNum_, localpos_);
    Eigen::VectorXd pose; getSensedValue(pose);
    Eigen::MatrixXd Tinv; Tinv.resize(4,3);
    Tinv << -0.5*pose(4), -0.5*pose(5), -0.5*pose(6),
             0.5*pose(3),  0.5*pose(6), -0.5*pose(5),
            -0.5*pose(6),  0.5*pose(3),  0.5*pose(4),
             0.5*pose(5), -0.5*pose(4),  0.5*pose(3);
    Jacobian.resize(7,ndof_);
    Jacobian.topRows(3) = Jg.topRows(3);
    Jacobian.bottomRows(4) = Tinv*Jg.bottomRows(3);
  }
}
```

c. Clase derivada *KineTaskPosture*

Esta clase implementa la tarea de postura y deriva de la clase base `KineTask` explicada anteriormente. Su definición se encuentra en `include/oscr/ik/kine-task-posture.hpp` y su implementación en `src/kine-task-posture.cpp`. Ambos archivos se pueden ver en su totalidad en el repositorio de github del sistema desarrollado en esta tesis.

En esta clase es importante poder escoger el número de la articulación que será controlada. Con este fin se tiene las dos funciones miembro llamadas `seJointNumbers` y `getJointNumbers`, las cuales permiten escoger los números de índices y obtener los índices actualmente seleccionados. Estas funciones se muestran en el Listado 4.13. De igual manera, el valor sensado de las articulaciones se obtiene directamente del vector de configuración articular, seleccionando solo aquellas

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

articulaciones de interés, con la función `getSensedValue`. El cálculo del error en la derivada del error se realiza de manera directa en este caso ya que ambas variables forman un espacio Euclideano, y por tanto, una simple sustracción es suficiente. La implementación de esta función, llamada `getDerivError` se muestra en el Listado 4.13.

Listado 4.13. Implementación de algunas funciones miembro de la clase KineTaskPosture, importantes para el sistema de control cinemático (elaboración propia).

```
void KineTaskPosture::setJointNumbers(const std::vector<unsigned int>& jnumbers)
{
  jointIndices_ = jnumbers;
}
std::vector<unsigned int> KineTaskPosture::getJointNumbers() const {return jointIndices_;}

void KineTaskPosture::getSensedValue(Eigen::VectorXd& xsensed)
  const
{
  Eigen::VectorXd q = rmodel_->getJointConfig();
  xsensed.resize(jointIndices_.size());
  for(unsigned int i=0; i<jointIndices_.size(); ++i)
  {
    xsensed(i) = q(7+jointIndices_.at(i));
  }
}

void KineTaskPosture::getDerivError(Eigen::VectorXd& de)
{
  Eigen::VectorXd xsensed;
  getSensedValue(xsensed);
  error_ = xdes_-xsensed;
  de = getGain(error_.norm())*error_;
}

void KineTaskPosture::getTaskJacobian(Eigen::MatrixXd& Jacobian)
  const
{
  Jacobian.resize(jointIndices_.size(), this->ndof_);
  Jacobian.setZero();
  for(unsigned int i=0; i<jointIndices_.size(); ++i)
  {
    Jacobian(i,jointIndices_[i]+7) = 1;
  }
}
```

El cálculo del Jacobiano de la tarea implementa lo descrito en la sección 3.5.4, y básicamente consiste en una matriz que contiene mayormente ceros, y solamente unos en las articulaciones cuyos valores se desea controlar. La implementación de este Jacobiano también se muestra en el Listado 4.13.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

4.3.3. Control Cinemático

Como se describe en la sección anterior, las salidas principales de cada tarea i-ésima son la trayectoria de referencia $\dot{e}_i^*$ y el Jacobiano de la tarea J_i. Estas señales, provenientes de cada una de las tareas que se especifique, ingresan al bloque de generación de movimiento basado en cinemática, como se muestra en la Figura 4.8. Dentro de todo este esquema, el bloque de control cinemático recibe estas entradas, además de la configuración articular actual $\tilde{\boldsymbol{q}}$ para implementar cada uno de los esquemas presentados en la sección 3.4. En el diagrama se hace evidente el uso de la librería QPOases para la solución a los programas cuadráticos necesarios para algunos esquemas de control. Este controlador cinemático luego genera la trayectoria de referencia de la velocidad articular $\dot{\boldsymbol{q}}^*$ que puede ser enviada directamente a un robot controlado en velocidad.

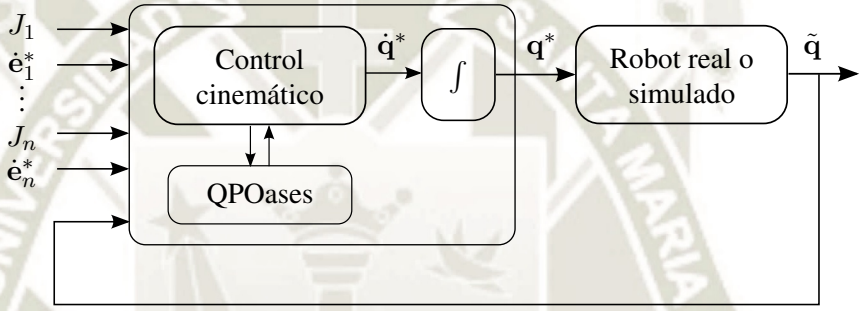


FIGURA 4.8: Diagrama de la generación cinemática de movimiento basada en tareas, donde las entradas son las referencias de velocidad de las n tareas, junto con sus Jacobianos, y la salida es la configuración articular $\mathbf{q}^*$ que se envía al robot (o a la simulación), asumiendo un robot controlado en posición (elaboración propia).

Sin embargo, debido a que los robots considerados en esta tesis son controlados en posición, la referencia de velocidad articular es integrada, como se muestra en la sección 3.3.3, para generar una trayectoria articular de referencia $\boldsymbol{q}^*$. Este vector de configuración variante en el tiempo es luego ingresado como señal de control a un robot real o a un robot simulado. Dicha señal debe ser obtenida cada periodo de control, el cual típicamente varía de 1 ms a 10 ms, y por tanto el cálculo debe ser bastante eficiente. Los valores articulares sensados $\tilde{\boldsymbol{q}}$ del robot real o del robot simulado dinámicamente son luego utilizados para el siguiente paso del controlador cinemático y para actualizar la configuración actual de la tarea. El detalle de la implementación de la clase base y de los esquemas de control cinemático se muestra a continuación.

a. Clase base *OSIKSolver*

La implementación de la Figura 4.8 en C++ se realiza principalmente a través de la interfaz provista por una clase base denominada *OSIKSolver* (OSIK=*Operational Space Inverse Kinematics*), la cual define las funciones básicas y genéricas del resolutor (*solver*). Luego, cada resolutor de la sección 3.4 es implementado en una clase que hereda a *OSIKSolver* y sus respectivos nombres son

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

OSIKSolverWQP, para el resolutor basado en ponderación de pesos, *OSIKSolverNS*, para el resolutor basado en proyecciones sobre los espacios nulos, y *OSIKSolverHQP*, para el enfoque basado en una jerarquía de QPs. La figura 4.9 muestra el diagrama de herencia de estas clases.

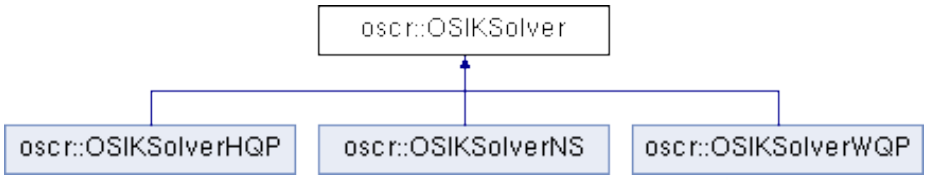


FIGURA 4.9: Diagrama de herencia para las clases del control cinemático, implementadas usando OSIKSolver y sus derivadas (elaboración propia).

Al igual que sucede con las clase base anteriores, en este caso también es posible la creación de nuevos algoritmos de solución heredando la clase base. La ventaja del uso de una clase base y de polimorfismo es que la interfaz no varía de resolutor a resolutor y resulta más fácil cambiar de uno a otro, o incluso implementar alguno diferente con el fin de realizar comparaciones. La definición de la clase base se encuentra en el archivo `include/osik/ik/osik-solver.hpp`, el cual se encuentra en el repositorio de github y también en el anexo C.3. La implementación de la clase se encuentra en `src/osik-solver.cpp`. Algunas de las definiciones de las funciones más importantes se muestran a continuación

Listado 4.14. Algunas funciones miembro de OSIKSolver (elaboración propia).

```
void pushTask(KineTask* task);
void popTask();
void removeTask(const std::string& taskName);
virtual void getPositionControl(const Eigen::VectorXd& q,
                                Eigen::VectorXd& qdes) = 0;
```

Las funciones `pushTask`, `popTask` y `removeTask` se utilizan para añadir una tarea, remover la última tarea, y remover una tarea arbitraria dado su nombre, respectivamente. Se puede observar que se hace referencia a las tareas solamente como un puntero a la clase base `KineTask`, sin especificar el tipo de tarea, motivo por el cual se puede utilizar cualquier tipo de tarea derivada, siempre que se use polimorfismo. La función `getPositionControl` permite obtener el vector de control cinemático que servirá como referencia para el robot controlado en posición.

b. Clase derivada *OSIKSolverWQP*

Esta clase deriva de la clase base `OSIKSolver` e implementa el esquema de control basado en ponderación de tareas descrito en la sección 3.4.1. Debido a la extensión del código que implementa la función `getPositionControl` para este caso, éste se muestra en su totalidad en el

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

apéndice C.3. En el repositorio del sistema implementado, la definición se encuentra en la cabecera `include/osik/ik/osik-solver-wqp.hpp`, y la implementación en `src/osik-solver-wqp.cpp`. En el código de esta función se puede observar que se implementa básicamente la ecuación descrita por (3.41), respetando los límites articulares y usando la librería *QPOases* para resolver el problema de optimización cuadrática. La cantidad de líneas de código se debe a que los datos se deben convertir al formato que utiliza *QPOases*, pero una vez realizado este proceso, la respuesta es obtenida de manera bastante rápida, como se verificará en el siguiente capítulo. Esta eficiencia es crucial para aplicaciones en tiempo real que corran en menos de 10 ms.

La resolución del problema de QP utiliza los límites articulares. Estos valores son dados a la clase utilizando la función miembro llamada `setJointLimits`, la cual siempre debe ser llamada antes de realizar cualquier operación. Estos valores pueden ser obtenidos directamente del modelo del robot; es decir, de alguna de las clases derivadas de `RobotModel`. Debido a que se está tomando en cuenta los límites articulares de posición y velocidad, este esquema de control puede ser aplicado en un robot real, ya que no se enviará referencias que estén fuera de las especificaciones de diseño del robot, garantizando su seguridad.

Por otro lado, dado que este esquema utiliza ponderación, se aprovecha el valor de peso que tiene cada una de las tareas. Como se mencionó en la sección 3.4.1, estos pesos indican la importancia relativa de cada una de las tareas en el resolutor, pero en este caso no son concluyentes; es decir, puede existir conflicto de tareas y se encuentra un punto medio entre éstas, sin llegar a satisfacer ninguna de las tareas por completo. Además, este esquema implementa un factor de regularización para obtener incrementos más pequeños y un movimiento más suave.

c. Clase derivada *OSIKSolverNS*

Esta clase deriva de la clase base `OSIKSolver` e implementa un resolutor basado en proyecciones sobre el espacio nulo de cada una de las tareas anteriores, las cuales se asume que tenían mayor prioridad. Este esquema es descrito con detalle en la sección 3.4.2. Así, es un esquema jerárquico donde una tarea de menor prioridad no interfiere con una tarea de mayor prioridad. La definición de la clase se encuentra en el repositorio github de esta tesis, en `include/osik/ik/osik-solver-ns.hpp`, y la implementación de la clase en `src/osik-solver-ns.cpp`. El código de la función llamada `getPositionControl`, que es la función más importante de esta clase ya que retorna el valor de referencia de la configuración articular, la cual será enviada como comando al robot, se muestra en el Listado 4.15.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Listado 4.15. Control cinemático basado en proyecciones sobre el espacio nulo (elaboración propia). De manera similar se puede usar los otros esquemas de control cinemático.

```
void OSIKSolverNS::getPositionControl(const Eigen::VectorXd& q, Eigen::VectorXd& qcmd)
{
  if (taskStack_.size()==0) {
    qcmd = q; return;
  }
  Eigen::MatrixXd J, pinvJ, P; Eigen::VectorXd dq, de;
  P = Eigen::MatrixXd::Identity(ndof_, ndof_);
  dq = Eigen::VectorXd::Zero(ndof_);
  Eigen::MatrixXd A1, A1p, A2, Pt;
  for (unsigned int i=0; i<taskStack_.size(); ++i)
  {
    taskStack_[i]->updateJointConfig(q);
    taskStack_[i]->getTaskJacobian(J);
    if(J.cols() != dq.rows()), return;
    taskStack_[i]->getDerivError(de);
    A1 = J*P;
    pinv(A1, A1p);
    A2 = de - J*dq;
    Pt = Eigen::MatrixXd::Identity(A1p.rows(), A1p.rows()) - A1p*A1;
    dq = dq + A1p*A2;
    P = P*Pt;
  }
  if (rmodel_->hasFloatingBase())
  {
    qdes_.tail(rmodel_->ndofActuated()) =
      qdes_.tail(rmodel_->ndofActuated()) + dt_*dq.tail(rmodel_->ndofActuated());
    qdes_.head(3) = qdes_.head(3) + dt_*dq.head(3);
    Eigen::Vector3d omega;
    Eigen::MatrixXd T; T.resize(3,4);
    T << -2.0*qdes_(4),  2.0*qdes_(3), -2.0*qdes_(6),  2.0*qdes_(5),
         -2.0*qdes_(5),  2.0*qdes_(6),  2.0*qdes_(3), -2.0*qdes_(4),
         -2.0*qdes_(6), -2.0*qdes_(5),  2.0*qdes_(4),  2.0*qdes_(3);
    omega = T*dq.segment(3,4);
    double angle = dt_*omega.norm()/2.0;
    Eigen::Vector3d k; k.setZero();
    if (omega.norm() > 1.0e-8)
      k = omega/omega.norm();
    else if (fabs(angle) > 1.0e-8)
      std::cout << "WARNING: axis for floating base orientation increment is zero" << std::endl;
    else {}
    Eigen::Vector4d dQ, Qf;
    dQ << cos(angle), k(0)*sin(angle), k(1)*sin(angle), k(2)*sin(angle);
    Qf = quaternionMult(dQ, qdes_.segment(3,4));
    qdes_(3) = Qf(0); qdes_(4) = Qf(1); qdes_(5) = Qf(2); qdes_(6) = Qf(3);
  }
  else
  {
    qdes_ = qdes_ + dt_*dq;
  }
  qcmd = qdes_;
  time_ ++;
}
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

A pesar que se realiza una integración de Euler de la velocidad obtenida, no se incurre en errores considerables debido a que el tiempo de control es bastante pequeño. Se podría realizar otro tipo de integración (como Runge-Kutta), pero en la práctica esto no representa una diferencia significativa. En el código se puede observar que se hace uso de cada una de las tareas añadidas a la pila de tareas (`stack`), de tal modo que la añadida en primer lugar es la más importante, y las siguientes van decreciendo en importancia. Esto implica implementar (3.51) de manera recursiva para cada una de las tareas, lo cual es realizado dentro de un bucle `for` que aplica esta ecuación. Nuevamente se puede observar que no interesa el tipo de tarea dado que se utiliza polimorfismo y por tanto solo se trabaja con la clase base. En este caso, dado que no hay un problema de optimización a resolver, no se hace uso de QPOases ni ningún otro paquete que brinde soluciones numéricas.

Es importante observar que existe una sección dedicada a la integración de la base flotante, tal como se define en la sección 3.3.3. Esta integración es importante para la correcta actualización de la parte de orientación de la base flotante, y para poder ser bien especificada cuando se realiza simulación. Para un robot real, su cálculo es necesario para poder tener una adecuada referencia interna de dónde se encuentra la base del robot, aunque esto podría ser reemplazado por un *IMU*, del cual se podría obtener la posición y orientación de la base. Por otro lado, la solución encontrada es la más cercana a la configuración actual. Este método es el más rápido de los implementados, pero por sí mismo no puede restringir los límites articulares debido a que no posee condiciones de desigualdad. Sin embargo, es posible introducir los límites utilizando una tarea de igualdad que se defina en términos de funciones potenciales repulsivas. Esto genera que el robot tienda a alejarse de configuraciones que se encuentran en el límite de lo permitido.

d. Clase derivada *OSIKSolverHQP*

Esta clase deriva de la clase base `OSIKSolver` e implementa un resolutor basado en programación cuadrática jerarquizada, como se describe en la sección 3.4.3. La principal función de esta clase (`getPositionControl`) no se colocó aquí debido a su complejidad, pero puede encontrarse en el apéndice C.3. En el repositorio de github, la definición se encuentra en la cabecera `include/osik/ik/osik-solver-hqp.hpp` y la implementación en `src/osik-solver-hqp.cpp`.

En este caso la implementación hace uso de *QPOases* dado que se tiene el problema formulado como una serie de problemas de optimización cuadrática. Debido a ello, los límites articulares pueden ser tomados en cuenta de manera directa como una restricción en el problema de optimización, y el movimiento es siempre viable en un robot real. Esta clase, y la función en cuestión, también realiza la integración de la base flotante del robot.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

4.3.4. Funciones de Ayuda.

Además de las clases mencionadas, se realizó la implementación de funciones para el manejo de operaciones relacionadas con rotaciones en 3D de cuerpos rígidos, descritos en la sección 2.1. Estas funciones se implementaron en el archivo `src/math-utils.cpp`, y su respectiva cabecera en `include/oscr/tools/math-utils.hpp`. Todo el código utilizado para estas funciones se muestra en el apéndice C.4. Se puede notar que se hace uso exhaustivo de la librería *Eigen* para el manejo de matrices y vectores. Se implementa operaciones para realizar transformaciones entre representaciones de eje/ángulo, matriz de rotación y cuaterniones, además de algunas operaciones con cuaterniones. Como en los casos anteriores, se usa el *namespace* `oscr` para evitar conflictos en los nombres de las funciones.

4.4. Interfaces de Uso

Como se muestra en la sección anterior, el sistema cinemático de generación de movimiento se implementa completamente en C++ para poder alcanzar eficiencia computacional. Por este motivo, la interfaz natural para el uso del sistema con cualquier robot es a través de programas escritos en este lenguaje. Existe además una interfaz para el uso de este sistema desde Python, por lo cual se puede aprovechar la facilidad de prototipaje de este lenguaje de tipo script. Esta sección abarca algunos ejemplos que muestran la forma de usar las interfaces desarrolladas. Debe notarse que la salida del sistema es una trayectoria de posición, que se envía luego a un robot o simulador de robot, pero un visualizador nativo del modelo, que podría ser realizado usando por ejemplo *openGL* o *unity*, está fuera del alcance de esta tesis. Sin embargo, para visualizar se puede usar algún sistema externo como ROS, lo cual se muestra en la sección 4.5.

4.4.1. Interfaz con C++

El uso nativo de las funciones implementadas se puede realizar con ejecutables escritos en este lenguaje. Sin embargo, la desventaja de utilizar el sistema propuesto desde aplicaciones netamente escritas en C++ es el hecho que C++ no permite interactuar con los resultados de manera natural. Además, cada vez que se desea modificar alguna posición objetivo o postura deseada, es necesario recompilar todo el programa y ejecutarlo, lo cual puede no ser muy práctico en algunos casos debido al tiempo de compilación, incluso si los cambios son muy pequeños.

Lectura del Modelo y Cinemática. Para leer el modelo de un robot y extraer tanto información básica del modelo, como la cinemática directa y el Jacobiano que incluye el efecto de la base

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

flotante, se utiliza lo descrito en la sección 4.3.1. La instanciación de la clase que utiliza la librería Pinocchio como backend se realiza de la siguiente manera:

Listado 4.16. Lectura del modelo en el sistema implementado (elaboración propia).

```
bool floating_base = true;
std::string urdf_model_name = "../models/ff6dof.urdf";
oscr::RobotModel *robot = new oscr::RobotModelPin(urdf_model_name, floating_base);
```

Se puede observar que es necesario proveer la ruta al archivo que contiene el modelo urdf, e indicar si el robot cuenta con una base flotante o no. En el ejemplo mostrado aquí, el modelo del robot se llama `ff6dof.urdf`, el cual ha sido diseñado solo para verificación, y cuenta con un brazo y una pierna, cada uno de 3 grados de libertad como se describirá con más detalle en la sección 5.2.1. La inicialización usando rbdl se realizaría de la siguiente manera

```
oscr::RobotModel *robot = new oscr::RobotModelRbdl(urdf_model_name, floating_base);
```

Como se ve, la estructura de declaración es similar, pero todas las funciones serán las mismas debido a que el puntero es de tipo `RobotModel`. Un ejemplo de salida de los datos de este modelo se muestran en la Figura 4.10, que se muestra en un terminal de Linux usando Ubuntu 16.04.

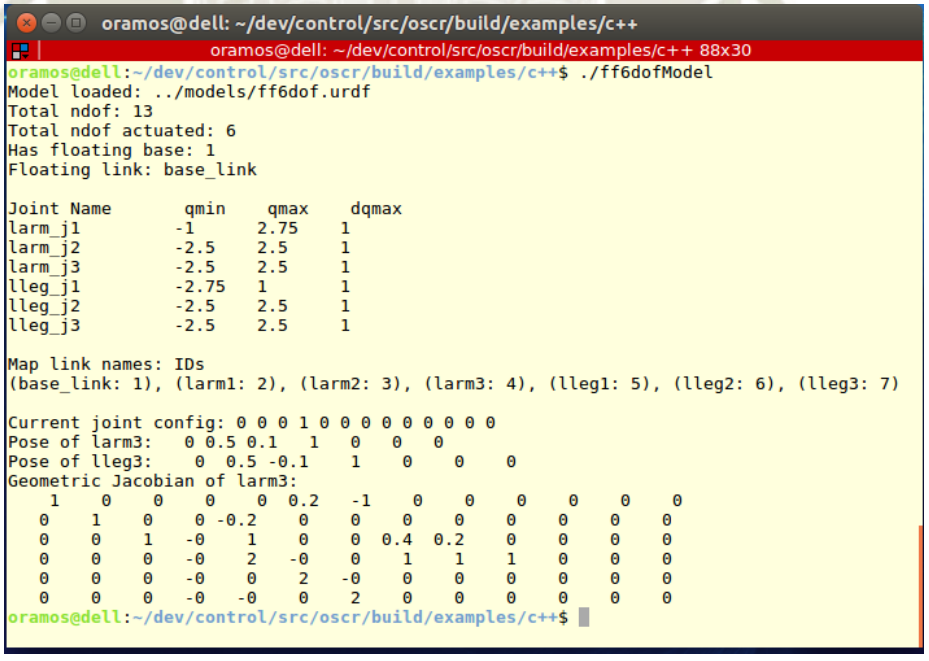


FIGURA 4.10: Salida de la lectura con C++ del modelo del robot llamado ff6dof, el cual posee 6 grados de libertad y base flotante, y su cinemática (elaboración propia).

Los valores obtenidos en la cinemática y en el Jacobiano serán luego verificados en la sección 5.2.2, donde se realizará el modelamiento de este robot de manera analítica solo a modo de verificación

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

y validación. El código en C++ que genera la salida mostrada en la figura anterior, se muestra a continuación, en el Listado 4.17.

Listado 4.17. Ejemplo de implementación de control cinemático (elaboración propia).

```
#include <oscr/model/robot-model-pin.hpp>
#include <oscr/model/robot-model-rbdl.hpp>
#include <oscr/tools/model-utils.hpp>

int main(int argc, char *argv[])
{
  bool fbase = true;
  std::string modelname = "../models/ff6dof.urdf";
  oscr::RobotModel *robot = new oscr::RobotModelPin(modelname, fbase);

  unsigned int ndof, ndof_actuated; bool has_floating_base;
  std::string floating_link; Eigen::VectorXd qmin, qmax, dqmax;
  std::vector<std::string> jnames;
  std::map<std::string, unsigned int> linkID;

  // Datos obtenidos de la lectura del modelo
  ndof = robot->ndof(); ndof_actuated = robot->ndofActuated();
  has_floating_base = robot->hasFloatingBase(); floating_link = robot->floatingLink();
  qmin  = robot->jointMinAngularLimits(); qmax  = robot->jointMaxAngularLimits();
  dqmax = robot->jointVelocityLimits();
  jnames = robot->jointNames();
  linkID = robot->mapLinkNamesIDs();

  // Informacion del modelo
  std::cout << "Total ndof: " << ndof << std::endl;
  std::cout << "Total ndof actuated: " << ndof_actuated << std::endl;
  std::cout << "Has floating base: " << has_floating_base << std::endl;
  std::cout << "Floating link: " << floating_link << std::endl;
  oscr::printJointLimits(jnames, qmin, qmax, dqmax);
  oscr::printLinkIDshort(linkID);

  // Configuracion articular con todos los valores en cero
  Eigen::VectorXd q(ndof), qcurrent(ndof); q.setZero();
  if (has_floating_base), q[3]=1.0;   // Cuaternion de la base
  robot->updateJointConfig(q);

  // Valores de cinematica y Jacobianos
  Eigen::VectorXd poselarm, poselleg; Eigen::MatrixXd Jlarm;
  qcurrent = robot->getJointConfig();
  poselarm = robot->linkPose(linkID["larm3"]);
  poselleg = robot->linkPose(linkID["lleg3"]);
  Jlarm = robot->geometricJacobian(linkID["larm3"]);

  std::cout << "\nCurrent joint config: " << qcurrent.transpose() << std::endl;
  std::cout << "Pose of larm3: " << poselarm.transpose() << std::endl;
  std::cout << "Pose of lleg3: " << poselleg.transpose() << std::endl;
  std::cout << "Geometric Jacobian of larm3:\n " << Jlarm << std::endl;

  return 0;
}
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Una vez declarado el puntero `*robot`, se extrae de él toda la información relevante que contiene el modelo urdf usando las funciones descritas en la sección 4.3.1, cuya declaración se puede ver con más detalle en el apéndice C.1. De igual manera, se define un vector de configuración inicial que en este caso es inicializado a cero (considerando un cuaternión unitario identidad, dado que se usa una base flotante). Esta configuración inicial se aplica al robot y a partir de ella se extrae la posición y orientación del extremo de la mano y de la pierna del modelo, así como el Jacobiano geométrico del brazo.

Control Cinemático. Para realizar control cinemático desde la interface de C++ es necesario definir las tareas usando instancias de `KineTask`, o sus derivadas como se describe en la sección 4.3.2, y definir un controlador cinemático usando alguna clase derivada de `OSIKSolver` como se describe en la sección 4.3.3. Debido a que la implementación del sistema no cuenta con un visualizador por defecto, la salida del control cinemático se mostrará solo como valores y se justificará usando gráficas del error. Un ejemplo, en un terminal de Linux (Ubuntu 16.04), se muestra en la Figura 4.11.

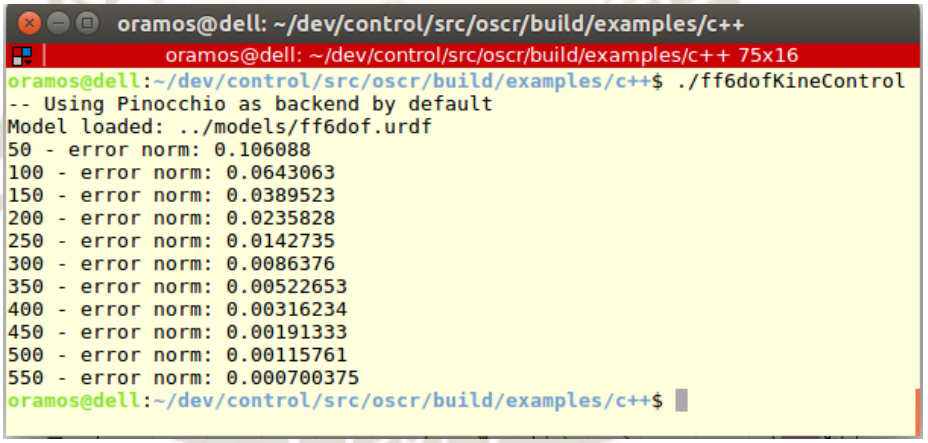


FIGURA 4.11: Salida del control cinemático con C++ para un robot de base flotante (elaboración propia). Se muestra que la norma del error disminuye con el tiempo.

A pesar de no haber un visualizador del modelo, se puede observar que el error de las tareas se reduce con el número de iteraciones y se detiene luego de 550 iteraciones debido a que el error es menor a la tolerancia arbitrariamente establecida en el programa. En este caso, el programa que genera esta salida mantiene en su lugar la parte terminal del brazo del robot (correspondiente a lo que sería el efector final) y mueve el extremo de la pierna del mismo usando todas las articulaciones y sacando ventaja del modelo con base flotante. El código de C++ utilizado para obtener esta salida se muestra en el Listado 4.18.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Listado 4.18. Ejemplo de generación de movimiento cinemático en C++ logrado en esta tesis (elaboración propia).

```
#include <oscr/model/robot-model-pin.hpp>
#include <oscr/model/robot-model-rbdl.hpp>
#include <oscr/ik/kine-task-pose.hpp>
#include <oscr/ik/osik-solvers.hpp>

int main(int argc, char *argv[])
{
  bool fbase = true;
  std::string modelname = "../models/ff6dof.urdf";
  oscr::RobotModel *robot = new oscr::RobotModelPin(modelname, fbase);
  std::map<std::string, unsigned int> linkID;
  linkID = robot->mapLinkNamesIDs();

  // Configuracion articular: todos los valores a cero y cuaternion identidad
  Eigen::VectorXd q(robot->ndof()); q.setZero(); q[3]=1.0;
  robot->updateJointConfig(q); q = robot->getJointConfig();

  // Definicion de una tarea para el brazo y una para la pierna
  oscr::KineTask *taskarm = new oscr::KineTaskPose(robot, linkID["larm3"], "position");
  oscr::KineTask *taskleg = new oscr::KineTaskPose(robot, linkID["lleg3"], "position");
  taskarm->setGain(1.0); taskleg->setGain(1.0);

  Eigen::VectorXd xarm, xleg, qdes, error;
  // Valor deseado para el brazo (mantenerlo sin movimiento)
  taskarm->getSensedValue(xarm);
  taskarm->setDesiredValue(xarm);
  // Valor deseado para la pierna (moverla 10 cm en x, y, z)
  taskleg->getSensedValue(xleg);
  xleg(0) = xleg(0) + 0.1; xleg(1) = xleg(1) + 0.1; xleg(2) = xleg(2) + 0.1;
  taskleg->setDesiredValue(xleg);

  // Solver: usando proyecciones en el espacio nulo
  double dt = 0.010;
  oscr::OSIKSolverNS solver(robot, q, dt);
  // Tareas usadas
  solver.pushTask(taskarm);
  solver.pushTask(taskleg);
  // Inicializacion del error
  taskleg->getDerivError(error);
  error = taskleg->getError();

  unsigned int i=0;
  // Bucle principal de control cinematico
  while (error.norm()>0.0005) {
    solver.getPositionControl(q, qdes);
    robot->updateJointConfig(q);
    q = qdes;
    error = taskleg->getError();
    if (++i%50==0), std::cout << i << " - error: " << error.norm() << std::endl;
  }
  return 0;
}
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

La Figura 4.12 muestra la posición del pie (parte final de la pierna) alcanzada (1 cm en x, y, z) así como el movimiento de todas las articulaciones para conseguir este efecto.

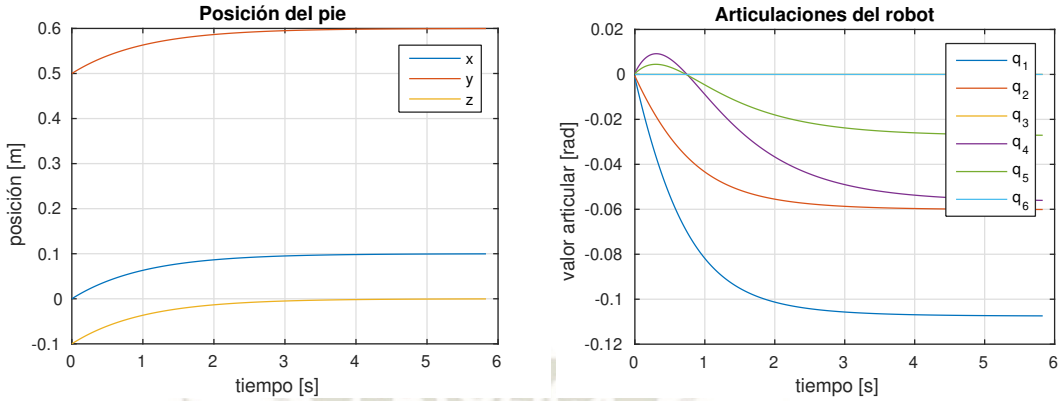


FIGURA 4.12: Posición alcanzada (izquierda) y articulaciones usadas (derecha) para un movimiento a modo de ejemplo (elaboración propia). Se puede observar la convergencia de los valores de posición al valor deseado (no mostrado en la imagen).

4.4.2. Interfaz con Python

A pesar de que es natural implementar programas usando C++ ya que todo el sistema está escrito en este lenguaje, a veces no es muy cómodo debido al tiempo de compilación que toma una simple modificación, y debido a que no es fácil la inspección interactiva de las variables. Por estos motivos se implementó una interfaz con Python, a través de SWIG, como se describió en la sección 4.2.2. En esta sección solo se mostrará ejemplos que brindan los mismos resultados que los mostrados en la sección anterior de C++.

Lectura del Modelo y Cinemática. Un ejemplo de cómo realizar la lectura del modelo de un robot, especificado mediante URDF, y de cómo obtener datos básicos de cinemática considerando la base flotante del robot, se muestra en el Listado 4.19. Si se compara este programa de Python con el programa de la sección anterior, hecho en C++, se puede observar que es prácticamente el mismo. Los nombres de las funciones son las mismas a excepción de algunos utilitarios utilizados para mostrar algunos valores en pantalla (como `printLinkNameIDShort` o `printJointLimits`), los cuales fueron escritos en Python mismo. Así, desde Python se utiliza las funciones de C++ sin realizar mayores cambios en las mismas, pero con las ventajas que un lenguaje script ofrece. En el ejemplo anterior se utilizó la librería *Pinocchio*. Para utilizar *RBDL*, se debe usar la clase `RobotModelRbdl`, como se describió anteriormente.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Listado 4.19. Ejemplo en Python de la lectura del modelo del robot ff6dof, en el archivo ff6dof_model.py (elaboración propia)

```
import numpy as np
from oscr.oscr import RobotModelPin, RobotModelRbdl
from oscr.utils import dictLink, printJointLimits, printLinkNameIDShort

if __name__ == '__main__':

    fbase = True
    modelname = "../models/ff6dof.urdf"
    robot = RobotModelPin(modelname, fbase)

    ndof  = robot.ndof(); ndof_actuated = robot.ndofActuated()
    has_floating_base = robot.hasFloatingBase(); floating_link = robot.floatingLink()
    qmin  = robot.jointMinAngularLimits(); qmax  = robot.jointMaxAngularLimits()
    dqmax = robot.jointVelocityLimits()
    jnames = robot.jointNames()
    linkID = dictLink(robot.mapLinkNamesIDs())

    # Datos obtenidos de la lectura del modelo
    print 'Total ndof: ', ndof
    print 'Total ndof actuated: ', ndof_actuated
    print 'Has floating base: ', has_floating_base
    print 'FloatingLink: ', floating_link, '\n'
    printJointLimits(jnames, qmin, qmax, dqmax)
    printLinkNameIDShort(linkID)

    # Configuracion articular con todos los valores en cero
    q = np.zeros(ndof)
    if (has_floating_base):
        q[3]=1.0   # Cuaternion de la base (identidad)
    robot.updateJointConfig(q)
    qcurrent = robot.getJointConfig()

    # Valores de cinematica y Jacobiano
    poselarm = robot.linkPose(linkID["larm3"])
    poselleg = robot.linkPose(linkID["lleg3"])
    Jlarm = robot.geometricJacobian(linkID["larm3"])

    print "Current joint config: ", qcurrent.T
    print "Pose of larm3: ", poselarm.T
    print "Pose of lleg3: ", poselleg.T
    print "Geometric Jacobian of larm3:\n ", Jlarm
```

En general, el código escrito en Python es más fácil de ser utilizado que el código en C++, lo cual permite realizar scripts y modificarlos con mayor facilidad. Además, al no necesitar compilar, es sencillo experimentar con las distintas opciones existentes.

La Figura 4.13 muestra la salida del programa anterior, en un terminal de Linux (Ubuntu 16.04). Se observa que la salida de la Figura 4.13 es similar a la mostrada en la Figura 4.10, donde se utilizó C++. La única diferencia es el formato de los vectores, dado que en este caso son de tipo *NumPy*, y

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

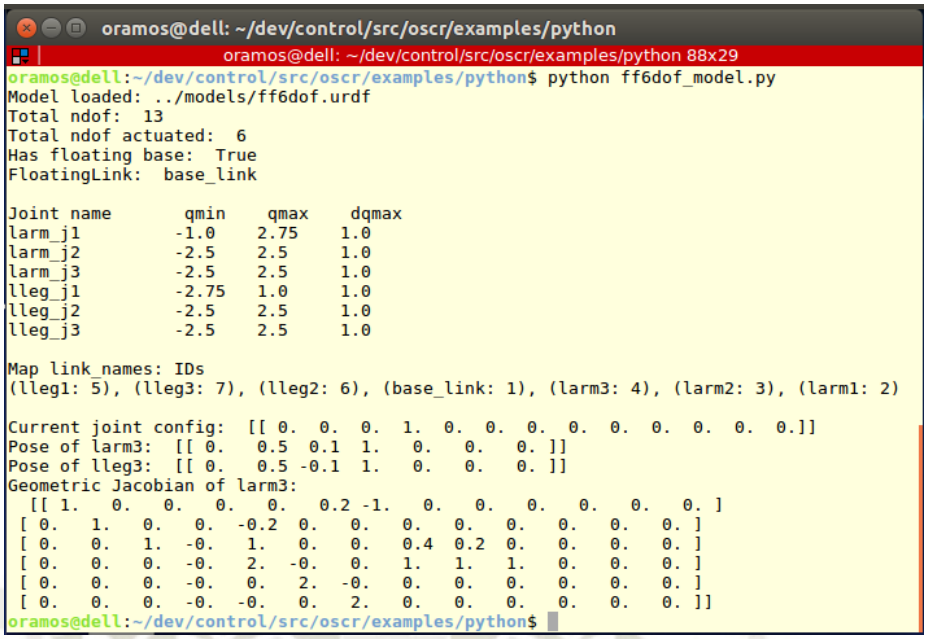


FIGURA 4.13: Salida de la lectura con Python del modelo del robot ff6dof y su cinemática (elaboración propia). Este robot posee 6 grados de libertad y una base flotante.

además en Python se muestra que los valores son de tipo flotante con un punto después del número, así sea cero. Aparte de estos valores, todo lo demás es similar.

Control Cinemático. Para generar movimiento utilizando control cinemático, es igualmente recomendable utilizar Python. Las mismas funciones que fueron usadas con C++ están disponibles en Python gracias a la interfaz provista. Así, el siguiente programa muestra cómo realizar un esquema básico de control cinemático en Python. La cinemática usa la clase `RobotModelPin`, pero ésta podría ser cambiada por `RobotModelRbdl` sin ningún problema y los mismos resultados serían obtenidos.

El programa mostrado en el listado 4.20, implementado en Python, realiza lo mismo que el programa equivalente de C++ mostrado anteriormente. Se puede observar que la estructura y las funciones son las mismas. Al igual que en ese caso, se utiliza la proyección en el espacio nulo para poder realizar las dos tareas previstas. Igualmente se podría usar los otros dos esquemas reemplazando `OSIKSolverNS` por `OSIKSolverWQP` o `OSIKSolverHQP` obteniendo resultados similares. La salida del programa es similar a la obtenida utilizando el código de C++, mostrado en la Figura 4.11, y por este motivo no se repite aquí. El uso de Python permite, por ejemplo, cambiar la referencia deseada de 10 cm a 5 cm e inmediatamente correr el programa sin necesidad de volver a compilar. También permite examinar paso a paso lo que sucede, cuando se ejecuta con `ipython -i`. Más aún, se puede hacer uso de interfaces como `Jupyter` que integran todo el código de manera amigable.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Listado 4.20. Ejemplo de control cinemático en Python (elaboración propia)

```
% ff6dof\_kine\_control.py}
import numpy as np
from oscr.oscr import RobotModelPin, RobotModelRbdl, KineTaskPose, OSIKSolverNS
from oscr.utils import dictLink, printJointLimits, printLinkNameIDShort

if __name__ == '__main__':

    fbase = True
    modelname = "../models/ff6dof.urdf"
    robot = RobotModelPin(modelname, fbase)
    linkID = dictLink(robot.mapLinkNamesIDs())

    # Configuracion articular: todos los valores a cero y cuaternion identidad
    q = np.zeros(robot.ndof())
    if (robot.hasFloatingBase()): q[3]=1.0
    robot.updateJointConfig(q)
    qcurrent = robot.getJointConfig()

    # Definicion de una tarea para el brazo y para una pierna
    taskarm = KineTaskPose(robot, linkID["larm3"], "position")
    taskleg = KineTaskPose(robot, linkID["lleg3"], "position")
    taskarm.setGain(1.0); taskleg.setGain(1.0)

    # Valor deseado para el brazo (mantenerlo sin movimiento)
    xarm = np.zeros((taskarm.getTaskDim(), 1))
    taskarm.getSensedValue(xarm); taskarm.setDesiredValue(xarm)
    # Valor deseado para la pierna (moverla 10 cm en x, y, z)
    xleg = np.zeros((taskleg.getTaskDim(), 1))
    taskleg.getSensedValue(xleg)
    xleg[0] = xleg[0]+0.1; xleg[1] = xleg[1]+0.1; xleg[2] = xleg[2]+0.1
    taskleg.setDesiredValue(xleg)

    # Solver: usando proyecciones en el espacio nulo
    dt = 0.010; solver = OSIKSolverNS(robot, q, dt)
    # Tareas usadas
    solver.pushTask(taskarm); solver.pushTask(taskleg)

    qdes = np.copy(q); error = np.zeros((taskleg.getTaskDim(), 1))
    taskleg.getDerivError(error); error = taskleg.getError()
    i=0; time = 0.0
    while (np.linalg.norm(error)>0.0005):
        solver.getPositionControl(q, qdes)
        robot.updateJointConfig(q)
        q = np.copy(qdes)
        error = taskleg.getError()
        if (i%50==0): print i, " - error norm: ", np.linalg.norm(error)
        i = i+1
```

Herramientas Desarrolladas en Python. Además del uso directo de las funciones de C++ en Python a través del uso de SWIG, en esta tesis se desarrolló algunas clases especializadas que facilitan el uso, pero que divergen un poco de la forma en la que se usa en C++. Se crearon clases que

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

internamente hacen llamado a las funciones de C++ pero que externamente tienen un funcionamiento más transparente para quienes vayan a utilizar el sistema desarrollado. Los detalles completos de las clases creadas puede verse en el repositorio github `https://github.com/oeramosp/oscr`, específicamente en `robot_base.py` y en `utils.py` que se encuentran en `oscr/modules/oscr`. Debido a que esta implementación es básicamente a nivel de Python y no aporta nada novedoso en la generación de movimiento más que funciones que encapsulan otras funciones, no se dará más detalles aquí.

4.5. Integración con ROS (*Robot Operating System*)

El sistema desarrollado en esta tesis por sí mismo no posee ningún visualizador o simulador, ya que ello no forma parte del objetivo de la tesis, el cual se enfoca principalmente en algoritmos de control cinemático para la generación de movimiento. Sin embargo, resulta importante visualizar el movimiento generado usando renderización de los modelos de los diferentes robots, y no solamente gráficos que muestren que el error tiende a cero. Con este fin se utiliza ROS (*Robot Operating System*): para poder visualizar los resultados y para poder simular robots reales. Además, ROS se ha convertido casi en un middleware estándar en robótica, y puede ser usado desde pequeños proyectos de robótica, hasta robots humanoides completos desarrollados con presupuestos millonarios.

4.5.1. Generalidades de ROS

ROS significa *Robot Operating System* y es una plataforma de desarrollo de aplicaciones en robótica que provee características como comunicación de mensajes, computación distribuida, reuso de código, entre otras. El objetivo es servir de plataforma que permita elaborar programas versátiles que puedan, a su vez, funcionar con diferentes robots realizando solo cambios menores al código. El proyecto ROS comenzó en el 2007 como parte del *Stanford AI Robot Project* del laboratorio de Inteligencia Artificial de la Universidad de Stanford. En el 2008 su desarrollo pasó a *Willow Garage*, un instituto e incubadora de investigación en robótica, pero con ayuda continua de reconocidas instituciones (de investigación e industriales) a nivel internacional. Desde el año 2013 hasta la actualidad, ROS es mantenido permanentemente por la OSRF cuyas siglas en inglés significan *Open Source Robotics Foundation*.

La comunidad de ROS ha experimentado un crecimiento muy rápido y actualmente cuenta con una gran cantidad de usuarios y desarrolladores a nivel mundial. La mayoría de las compañías y laboratorios de investigación en robótica están ahora portando su software a ROS. Esta tendencia también es visible en robots industriales, donde compañías están paulatinamente migrando de aplicaciones propietarias a ROS. El movimiento llamado *ROS Industrial* se ha incrementado en estos últimos

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

años y su objetivo, básicamente, consiste en extender las capacidades avanzadas de ROS a la manufactura. Las aplicaciones cada vez mayores de ROS pueden generar muchas oportunidades en esta área. Más aún, de acuerdo con las tendencias, el conocimiento de ROS se hace cada vez más importante en el área de robótica tanto en investigación como en industria.

Filosofía de ROS. En sentido general, ROS sigue la filosofía de Linux/GNU de desarrollo de software. Esto hace que ROS sea "natural" para desarrolladores familiarizados con Linux. La filosofía bajo la cual fue creado se basa en los siguientes aspectos.

- *Peer to peer.* Los sistemas en ROS están formados por pequeños programas (nodos) interconectados unos con otros a través del intercambio continuo de mensajes.
- *Multilenguaje.* Los lenguajes soportados por defecto en ROS son C++ y Python. Sin embargo, existen interfaces con LISP, Java, JavaScript, MATLAB, Ruby, Haskell, R, Julia, entre otros. Los módulos de ROS pueden correr y comunicarse entre sí independientemente del lenguaje en el que hayan sido escritos.
- *Modularidad.* Las convenciones de ROS motivan a crear librerías independientes y realizar wrappings para estas librerías de tal modo que puedan enviar y recibir mensaje a través de módulos de ROS. Esto permite mayor reusabilidad del software.
- *Open Source.* ROS se encuentra bajo la licencia BSD que permite uso comercial y no comercial. El paso de datos entre módulos usa comunicación interproceso (ICP), de tal modo que los sistemas pueden tener libertad sobre varios componentes.

Características Adicionales de ROS. Algunas de las características que hacen que recientemente exista preferencia por ROS sobre el uso de otras plataformas tradicionalmente usadas en robótica, como YARP, Orocos, son las siguientes:

- *Paquetes especializados.* Existen muchos paquetes en ROS fácilmente integrables con cualquier robot y con diversas funcionalidades (visualización, simulación, etc.).
- *Soporte para sensores y actuadores.* ROS viene con drivers y paquetes de interface para varios sensores y actuadores comunmente usados en robótica como lidars, escáneres láser, Kinect, servos Dynamixel, etc. Además, se puede realizar la interfaz de componentes con ROS de manera sencilla.
- *Manejo de recursos concurrentes.* El manejo de recursos de hardware por más de un proceso es normalmente un problema y se puede hacer complicado. La forma en la que ROS funciona,

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

a través de nodos, reduce la complejidad computacional y se incrementa la mantenibilidad del sistema.

Visualización y Simulación en ROS. En ROS existe un visualizador por defecto, llamado *RViz*, el cual muestra el modelo URDF del robot pero no realiza simulación. Es decir, solamente muestra los eslabones en la posición que se indica que deben estar, independientemente de las características dinámicas de los mismos. Para realizar simulación se utiliza un programa llamado *Gazebo*, el cual es un simulador dinámico que se basa en *ODE* (*Open Dynamics Engine*) y en *Bullet*, los cuales son a su vez librerías para el manejo de características dinámicas de los objetos. Así, usando Gazebo con parámetros adecuados se puede tener respuestas muy similares a las obtenidas con un robot real, además de poder simular controladores de posición, velocidad, torque, entre otros.

4.5.2. Generación de Movimiento en ROS

En esta tesis se decidió realizar una interfaz con ROS debido a la importancia que actualmente tiene esta plataforma en robótica, y debido a que muchos robots actualmente utilizan este recurso. Más aún, bajo la suposición que hace esta tesis, en la cual se trabaja solamente con robots controlados en posición, ROS permite realizar la simulación encargándose de los controladores de bajo nivel (PIDs de los motores) y brindando una interfaz de más alto nivel, tal como se realiza en un robot real. Así, lo importante del uso de ROS es principalmente la visualización usando *RViz* y la simulación usando *Gazebo*. Alternativamente se podría usar otros simuladores en ROS (como *VREP*) pero se decidió usar *Gazebo* dado que es el simulador por defecto de ROS. El siguiente capítulo muestra resultados usando este visualizador y este simulador, para algunos robots.

Debido a que la integración con ROS es un problema principalmente a nivel del uso mismo de la estructura de ROS (tópicos, servicios, nodos) y de la implementación de funciones y elementos que puedan interactuar adecuadamente con esta estructura, no se proveerá aquí más detalles. Esta integración es importante desde el punto de vista funcional, pero carece de importancia desde el punto de vista del propio sistema de control cinemático ya que no añade funcionalidades importantes sino solo interfaces. Es decir, encapsula funciones de tal modo que puedan ser fácilmente utilizadas desde ROS. Las principales clases e interfaces desarrolladas para ROS se encuentran en el repositorio de github `https://github.com/oeramosp/oscr`, específicamente en `oscr/modules/oscr`. En particular, los archivos implicados son `ros_robot.py`, `ros_kine_sim.py`, `ros_kine_tasks`, `ros_pubs.py`, y `ros_markers.py`. El lector interesado en estos detalles de implementación puede remitirse directamente a estos archivos.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Capítulo 5

Pruebas y Resultados

Como se ha mencionado en capítulos anteriores, el sistema desarrollado en la presente tesis no depende de complicados middlewares sino únicamente de algunas librerías de código abierto. Así, las pruebas pueden ser realizadas utilizando únicamente el sistema *per-se*. Sin embargo, debido a que el sistema no tiene embebido un visualizador, las pruebas sin elementos externos se restringirían a valores de salida en un terminal o al registro temporal de valores en un archivo (para su posterior gráfica). Sin embargo, un visualizador no es necesario cuando un sistema es ejecutado *on-board*, así que esta carencia no restringe la funcionalidad principal.

Debido a la falta de un visualizador embebido, resulta conveniente utilizar el sistema con algún middleware que sí lo posea. Por su independencia, el sistema puede ser fácilmente integrado en diversos middlewares o frameworks de robótica. En este capítulo se realizará la integración con ROS (*Robot Operating System*), como se describió en la sección 4.5, debido a que es un framework cada vez más utilizado en robótica, tanto que se ha convertido en el estándar *de facto* en robótica actualmente. La versión utilizada en esta tesis es *ROS Kinetic* y la distribución de Linux usada es Ubuntu 16.04. Sin embargo, el sistema es compatible con otras versiones ya que no hace uso de aspectos específicos de ninguna versión. Se iniciará mostrando la utilidad para robots manipuladores de base fija y luego se mostrará resultados para robots humanoides de base flotante.

5.1. Movimiento para Robots Manipuladores

Esta sección presenta el control cinemático de algunos robots manipuladores de base fija. Primero se mostrará el movimiento de un robot manipulador tradicional de seis grados de libertad, y luego de un robot de siete grados de libertad, donde se podrá apreciar la ventaja del enfoque de cinemática diferencial para el tratamiento de la redundancia.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

5.1.1. Robot KUKA KR-5 (seis grados de libertad)

Los robots manipuladores tradicionales utilizados en la industria poseen seis o menos grados de libertad. En esta sección se realizará la generación de movimiento para el robot industrial KR-5 de *KUKA*, el cual posee 6 grados de libertad. Todo el código usado para esta sección se encuentra disponible en https://github.com/oeramosp/kr5_oscr.

Modelo del Robot. Es un robot industrial de seis grados de libertad producido por el fabricante alemán de robots KUKA, y es típicamente utilizado en procesos de manufactura altamente automatizados. El modelo URDF de este robot puede ser recuperado a partir del siguiente repositorio git, el cual es provisto por el propio fabricante en https://github.com/ros-industrial/kuka_experimental. La Figura 5.1 muestra este robot en el mundo real así como la visualización de su modelo utilizando el visualizador RViz en ROS. El modelo mostrado en la derecha es obtenido a partir del modelo URDF del robot y no presenta un efector final.

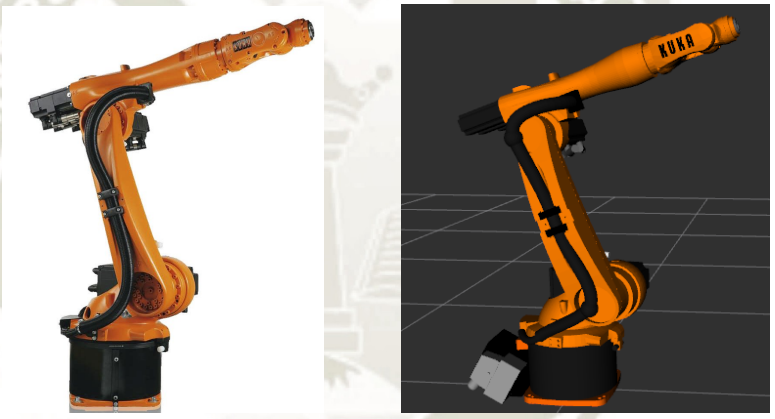


FIGURA 5.1: Robot KR-5 de KUKA. Izquierda: robot real (fuente: KUKA); derecha: robot en ROS (elaboración propia usando el modelo provisto por el fabricante).

Haciendo uso del sistema desarrollado en esta tesis se puede obtener de manera automática información importante de este robot. Como ejemplo, la Figura 5.2 muestra la salida en un terminal de Linux donde se muestra esta información utilizando la interface de Python. En dicha figura se puede observar el número de grados de libertad (ndof), el número de grados de libertad articulados, que en este caso es igual que el número anterior, la indicación que el robot no posee base flotante, el nombre de las articulaciones junto con sus límites articulares y de velocidad, así como el nombre de los eslabones junto con sus respectivos números.

Los mensajes mostrados son arbitrarios y pueden ser modificados. Se utilizó el inglés ya que esto es común en robótica, pero sin problemas se podría cambiar los mensajes informativos a español, excepto los nombres de eslabones o articulaciones ya que estos son provistos en inglés por el propio modelo URDF.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

```
Total ndof:  6
Total ndof actuated:  6
Has floating base:  False
floatingLink:  NONE
Joint name       qmin    qmax    dqmax
joint_a1        -2.70526034059  2.70526034059   2.68780704807
joint_a2        -3.14159265359  1.1344640138    2.68780704807
joint_a3        -0.261799387799         2.75762021815   3.97935069455
joint_a4        -6.10865238198  6.10865238198   5.98647933434
joint_a5        -2.26892802759  2.26892802759   6.70206432766
joint_a6        -6.10865238198  6.10865238198   12.5838239069
Link names and IDs (map):
  {'link_3': 3, 'link_2': 2, 'link_1': 1, 'link_6': 6, 'link_5': 5, 'link_4': 4}
Joint names and IDs (map):
  {'joint_a1': 0, 'joint_a2': 1, 'joint_a3': 2, 'joint_a4': 3, 'joint_a5': 4, 'joint_a6': 5}
```

FIGURA 5.2: Salida del programa que lee el modelo del robot, mostrando algunas características del robot KR-5 de KUKA (elaboración propia).

Control de Posición y Orientación. Para mostrar el funcionamiento de la tarea que controla la posición y orientación se comenzó con el robot en una configuración articular dada por $\boldsymbol{q}(0) = (0, -0.7,\ 1.5,\ 0,\ 0,\ 0)$, lo cual equivale a que el efector final se encuentra en la posición $\boldsymbol{x}(\boldsymbol{q}) = (1.16,\ 0,\ 0.43)$ m con una orientación dada por el cuaternión $\boldsymbol{Q}(\boldsymbol{q}) = (0.66, -0.27, -0.27,\ 0.66)$. El objetivo del movimiento consiste en alcanzar la posición final $\boldsymbol{x}_d = (0.66,\ 0.50,\ 0.93)$ incrementando en la orientación inicial 70 grados alrededor del eje y, lo cual equivale al cuaternión final $\boldsymbol{Q}_d = (0.67, -0.59,\ 0.16,\ 0.38)$. La Figura 5.3 muestra el resultado de posición y orientación en función del tiempo.

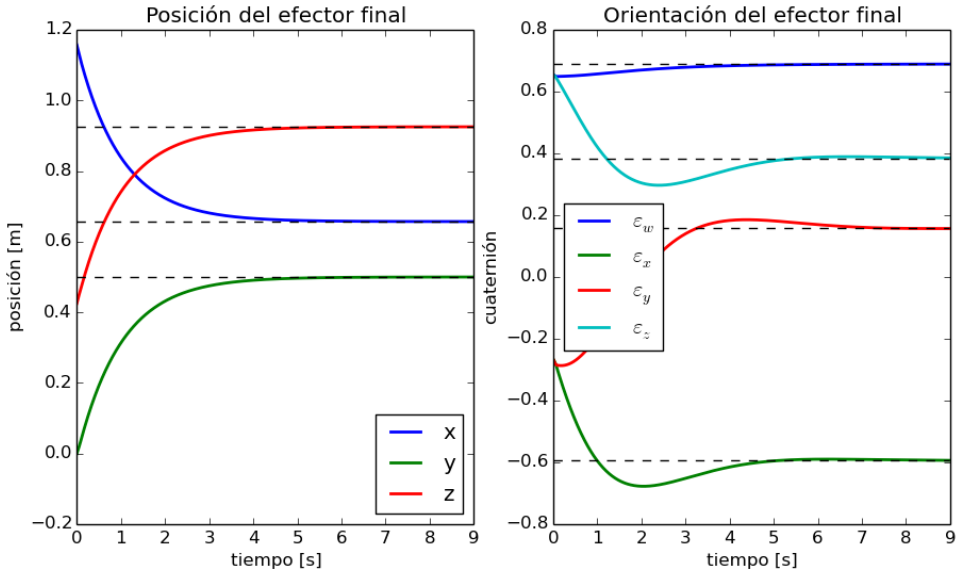


FIGURA 5.3: Posición cartesiana (izquierda) y orientación dada por el cuaternión (derecha) del efector final (elaboración propia). Las líneas punteadas son los valores deseados

Las líneas punteadas muestran los valores deseado para cada uno de los componentes. Se puede observar que se alcanza tanto la posición Cartesiana deseada, como la orientación deseada, y el tiempo de establecimiento es de aproximadamente 5 segundos utilizando el valor de ganancia $\lambda = 1.0$ en

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

(3.25). El tiempo de respuesta se puede mejorar incrementando el valor de ganancia (como se mostrará más adelante); sin embargo, en este caso solo es de interés el hecho de llegar al valor deseado. Se debe considerar además que cada uno de los componentes del cuaternión mostrados satisfacen en todo momento la restricción que el módulo del cuaternión es unitario. La figura 5.4 muestra la evolución temporal de cada una de las seis articulaciones de este robot para el movimiento descrito. Se puede observar un comportamiento temporal suave y sin saltos. Además, ninguna articulación sobrepasa los límites articulares.

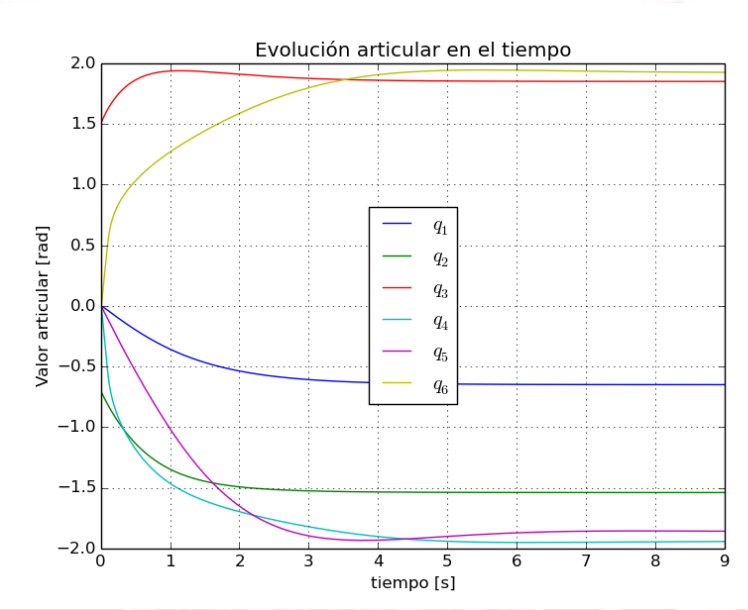


FIGURA 5.4: Valores articulares en función del tiempo para el movimiento que consiste en obtener una posición y orientación deseados del Robot KR-5 (elaboración propia).

La Figura 5.5 (izquierda) muestra la trayectoria cartesiana seguida por el efector final, así como el punto de inicio y el punto final, que es el punto deseado. Se puede observar que se describe casi una línea en el espacio Cartesiano, excepto por el inicio, donde hay una pequeña curva generada automáticamente para que el robot consiga una configuración apropiada. La figura de la derecha muestra la misma trayectoria (en verde) sobre el robot. Las coordenadas de la base del robot, que se utilizan como referencia, también se muestran. Como es convención en robótica, el eje x está representado por el color rojo, el eje y por el color verde y el eje z por el color azul (esta convención viene del agrupamiento de colores RGB). También se muestra el sistema de referencia en el efector final del robot. Las trayectorias en ambas figuras son equivalentes, siendo el ángulo de vista la única variación.

La Figura 5.6 muestra el tiempo de cálculo que consume el algoritmo de generación de movimiento en cada una de las iteraciones utilizadas. El resolutor utilizado en este caso es el de jerarquía de QPs (HQPs). Se observa que el tiempo de ejecución es en promedio alrededor de 0.2 ms con algunos picos eventuales que llegan a aproximadamente 0.4 ms. Este tiempo de cálculo permite una

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

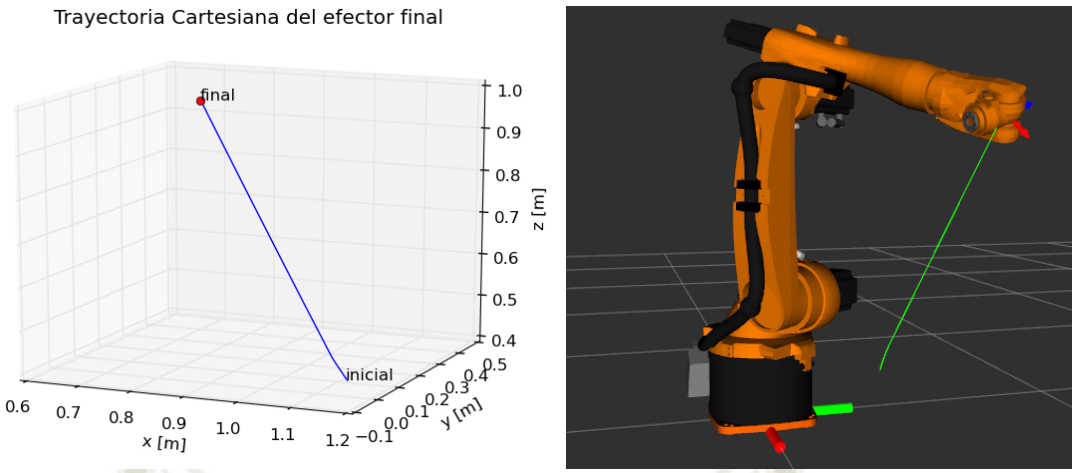


FIGURA 5.5: Trayectoria cartesiana para el movimiento que consiste en ir de la posición inicial a la final: gráfico 3D de la trayectoria (izquierda), movimiento del robot (derecha) donde la línea verde muestra la trayectoria seguida (elaboración propia).

aplicación en tiempo real ya que los robots industriales tienen ciclos de control de entre 5 ms a 10 ms, y el tiempo obtenido es muy inferior a dichos valores. El tiempo de cálculo obtenido con los otros dos resolutores (proyección en el espacio nulo y ponderación de pesos) es similar al mostrado en la Figura 5.6, variaciones significativas, por lo que se omite su gráfico aquí. Las diferencias temporales son más notorias en casos más complejos.

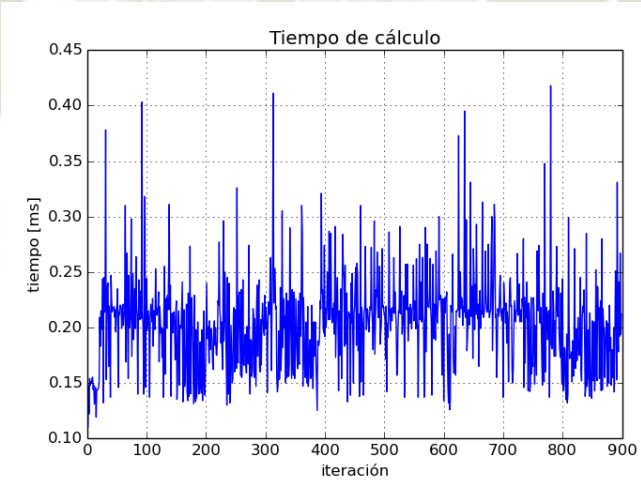


FIGURA 5.6: Tiempo de cálculo experimental para el robot KR-5, en función del número de iteraciones (tiempo discreto) usando el esquema HQP (elaboración propia).

Seguimiento de Trayectoria Lineal. A diferencia del caso presentado anteriormente donde se brindaba la posición y orientación finales, en este caso se brinda la trayectoria completa que tiene que ser seguida por el efector final, manteniendo su orientación. La trayectoria deseada es lineal comenzando en el punto $\boldsymbol{x}_o = (1,\ 0.5,\ 0.4)$ y desplazándose 1 m en la dirección del eje y negativo; es decir, terminando en el punto final $\boldsymbol{x}_f = (1, -0.5,\ 0.4)$. Los puntos dentro de esta trayectoria

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

lineal están descritos mediante

$$\boldsymbol{x} = \boldsymbol{x}_o + t\,(\boldsymbol{x}_f - \boldsymbol{x}_o)$$

donde $t = [0, 1]$. La Figura 5.7 muestra el resultado Cartesiano de esta trayectoria lineal, a la izquierda, y el resultado con el robot manipulador, a la derecha. La figura del robot muestra la posición final y la trayectoria seguida (en verde) por el efector final desde la configuración inicial hasta la final. Esto se realizó utilizando RViz y publicando cada uno de los puntos medidos del efector final en un tópico de camino (*path*).

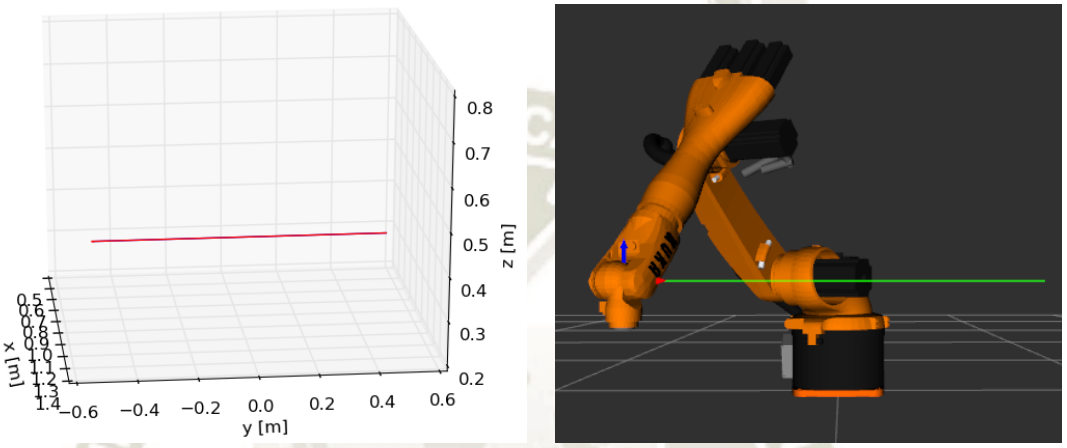


FIGURA 5.7: Movimiento de seguimiento de trayectoria lineal. Izquierda: gráfico 3D de la trayectoria seguida por el efector final del robot; derecha: robot en su posición final mostrando en verde la trayectoria seguida (elaboración propia).

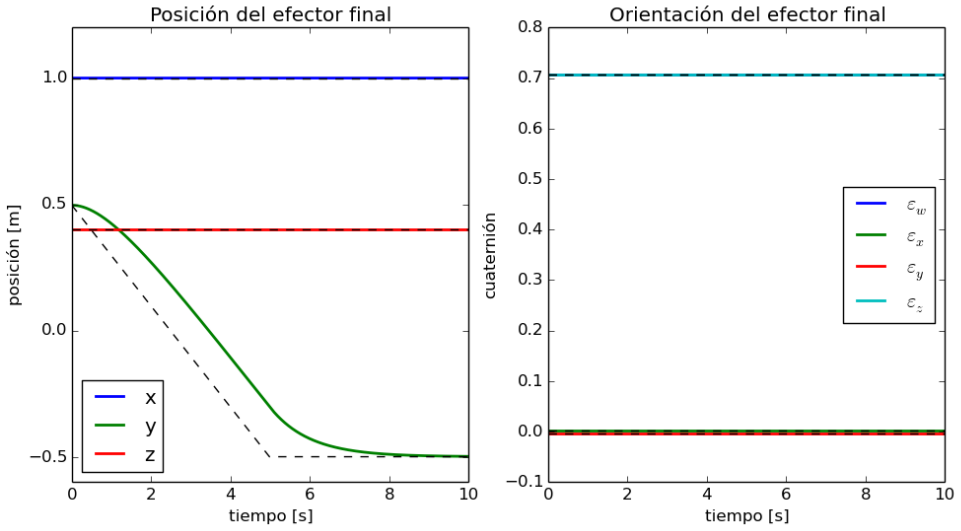


FIGURA 5.8: Posición cartesiana (izquierda) y orientación dada por un cuaternión (derecha) del efector final para el seguimiento de una trayectoria lineal, donde la línea punteada muestra la trayectoria de referencia (elaboración propia).

La Figura 5.8 muestra la evolución de la posición y de la orientación (mediante un cuaternión). Como se mencionó, en este caso la orientación se mantiene, por lo que el gráfico de la derecha

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

no muestra variaciones sino valores constantes para cada uno de los componentes del cuaternión. El gráfico de la izquierda muestra que los componentes x y z se mantienen constantes, como se desea, y es solamente el componente en y el que varía de 0.5 a -0.5. La línea punteada muestra la trayectoria deseada, y se observa que existe retardo en el seguimiento de dicha trayectoria. Esto es debido a la ganancia que se utilizó, la cual es $\lambda = 1$. Evidentemente al incrementar la ganancia se podrá seguir la trayectoria de manera más cercana; dicho análisis se mostrará posteriormente.

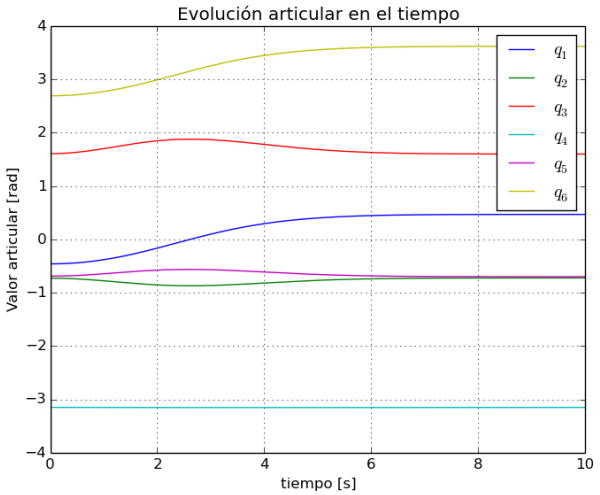


FIGURA 5.9: Valores articulares en función del tiempo para el Robot KR-5 cuando realiza el seguimiento de una trayectoria lineal (elaboración propia).

La trayectoria articular para este tipo de movimiento se muestra en la Figura 5.9. Esta trayectoria se obtiene de manera automática utilizando el resolutor basado en jerarquía de QPs y se satisface los límites articulares. Se puede además observar que esta evolución articular no es evidente ya que algunas articulaciones se incrementan continuamente, otras permanecen casi constantes, y otras se incrementan y luego decrementan, de una manera que sería difícil de controlar manualmente.

En este ejemplo, la trayectoria de interés se da únicamente en el eje y y como se vio en la Figura 5.8, para una ganancia unitaria, si bien se logra seguir la trayectoria deseada, existe un desfase temporal. La Figura 5.10 muestra la respuesta temporal en el eje y para distintos valores de ganancia λ en (3.25). La línea punteada muestra la trayectoria de referencia. Evidentemente, a medida que la ganancia se incrementa, el efecto es de seguir de manera más cercana la trayectoria de referencia. Para valores mayores a $\lambda = 5$ el error se hace bastante pequeño.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

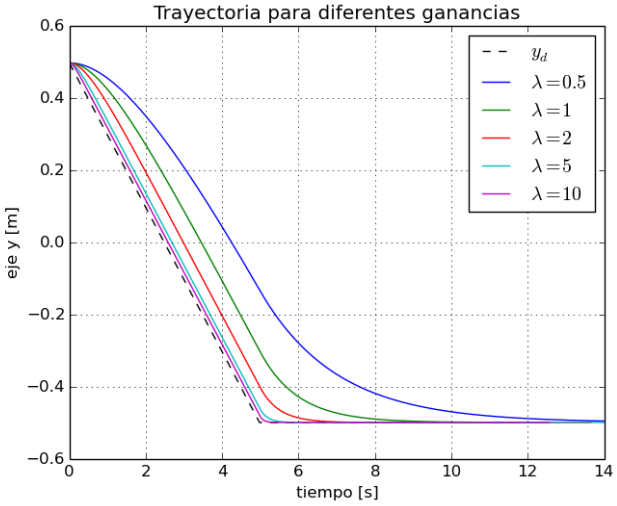


FIGURA 5.10: Coordenada y del efector final cuando se realiza el seguimiento de una trayectoria lineal en el espacio con distintos valores de ganancia λ (elaboración propia). La línea punteada muestra la trayectoria de referencia.

Seguimiento de Trayectoria Circular. En este caso se busca que el robot mantenga su orientación constante y siga una trayectoria circular definida paramétricamente como

$$x(t) = x_0$$
$$y(t) = y_0 + r\cos(\theta(t)) - r$$
$$z(t) = z_0 + r\sin(\theta(t))$$

con $\theta \in [0, 2\pi]$, donde r es una constante, y (x_0, y_0, z_0) es la posición inicial del efector final. Por diseño, la trayectoria circular empezará en la parte derecha y se moverá de manera antihoraria. Esto no representa limitación alguna y es solo un ejemplo de trayectoria. La evolución de la posición y orientación se muestra en la figura 5.11, donde se observa que la orientación es constante, como se desea, y las componentes x, y siguen muy de cerca la referencia (mostrada en línea punteada). Dicho seguimiento cercano de la trayectoria se debe a que se utilizó una ganancia $\lambda = 10$ en (3.25), y como se vio en la Figura 5.10, dicha ganancia mantiene un seguimiento de trayectoria adecuado. La Figura 5.12 muestra la evolución articular para esta trayectoria. Se observa que los valores articulares tienen un comportamiento ondulante debido a que de alguna forma están afectados por la forma sinusoidal de la trayectoria. Sin embargo, no es evidente determinar exactamente la función que describe cada articulación, pero esto no es necesario ya que su valor en el tiempo se obtiene de manera automática con el esquema de control cinemático.

La Figura 5.13 muestra la trayectoria en el plano Cartesiano, tanto en una gráfica que muestra la curva deseada y la curva seguida (en este caso no se nota la diferencia ya que el seguimiento es muy

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

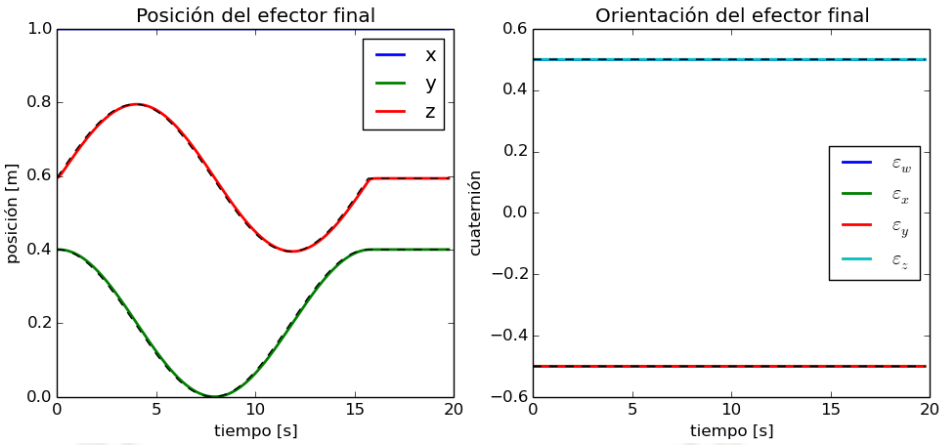


FIGURA 5.11: Posición cartesiana (izquierda) y orientación (derecha) del efector final del robot KR-5 para el seguimiento de una trayectoria circular que mantiene la orientación constante (elaboración propia).

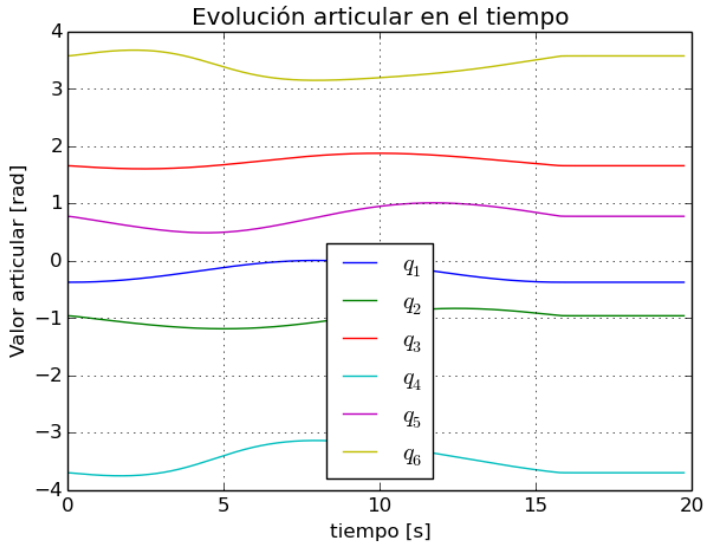


FIGURA 5.12: Valores articulares de KR-5 para el movimiento que consiste en el seguimiento de una trayectoria circular (elaboración propia).

cercano), como la curva obtenida con el efector final del robot. En ambos casos se observa que se obtiene una trayectoria circular.

5.1.2. Robot Sawyer (siete grados de libertad)

El robot Sawyer es fabricado por Rethink Robotics, una empresa Estadounidense, y es un ejemplo de robot colaborativo para aplicaciones industriales. El hecho que sea un robot colaborativo implica que está diseñado para poder desarrollar tareas en colaboración con humanos. Por este motivo tiene las esquinas redondeadas, como se observa en la Figura 5.14, para no causar lesiones, y en caso de detectar una obstrucción, el robot se detiene (ya que los sensores de corriente en las articulaciones detectan un pico inusual).

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

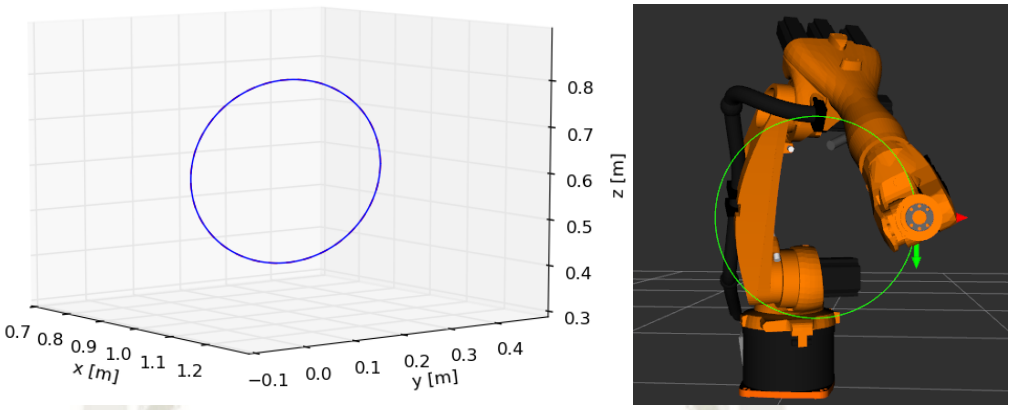


FIGURA 5.13: Trayectoria Circular obtenida en el espacio 3D (izquierda), y robot en su posición final (derecha) mostrando en verde la trayectoria seguida (elaboración propia).

FIGURA 5.14: Robot Sawyer. Izquierda: robot real (obtenido de Rethink Robotics); derecha: robot en RViz usando el modelo provisto por el fabricante (elaboración propia).

El modelo URDF del robot, mostrado en RViz en la Figura 5.14, se encuentra en el paquete `sawyer_description` del repositorio https://github.com/RethinkRobotics/sawyer_robot, el cual es provisto por el fabricante con todas las características del mismo. En esta sección se utiliza este robot para mostrar la generalidad de la solución propuesta en otro robot manipulador, pero con siete grados de libertad, lo cual hace que este robot sea redundante. En este caso, el robot tiene un efector final y es capaz de coger un objeto y desplazarlo, tal como sería realizado en actividades industriales. Todo el código para hacer el enlace entre el sistema implementado en esta tesis y el robot Sawyer, así como los scripts para la generación de movimiento de esta sección, se puede encontrar en el repositorio https://github.com/oeramosp/sawyer_oscr.

Uso del Espacio Nulo. Una de las ventajas del método propuesto e implementado en esta tesis con respecto a métodos tradicionales de robots manipuladores consiste en poder realizar más de una tarea en simultáneo. Cuando se cuenta con un robot redundante, como Sawyer, estas dos tareas

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

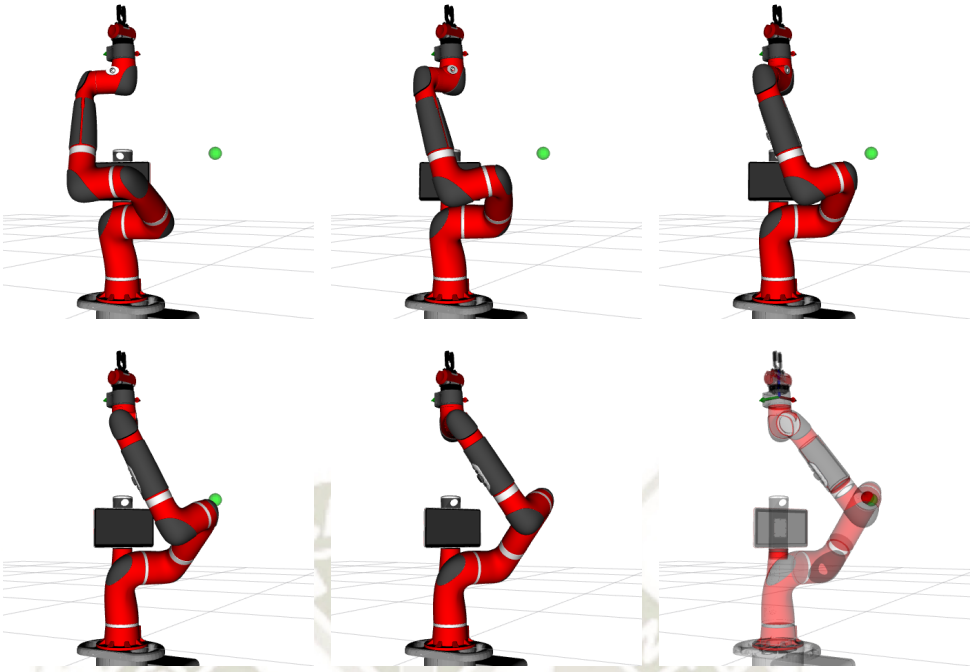


FIGURA 5.15: Movimiento del codo manteniendo la posición y orientación del efector final (elaboración propia). La secuencia es de arriba a abajo, y de derecha a izquierda.

corresponden al movimiento del efector final y al movimiento del codo; es decir, el codo constituye el espacio nulo del robot. Para verificar esto, la tarea que se dará al robot consiste en mantener el efector final en una posición fija mientras se comanda un movimiento al codo del robot de su posición actual hacia una posición deseada. La Figura 5.15 (de arriba a abajo y de izquierda a derecha) muestra este movimiento: se puede observar que el efector final no presenta movimiento, pero el codo del robot sí se desplaza. En este caso dado que el efector final no se mueve, la única forma en la que el codo puede moverse es girando. La esfera verde representa la posición deseada del codo, y al término del movimiento se observa una esfera roja (la posición actual del codo) cerca de la esfera verde (la posición deseada). No es posible llegar a la posición deseada dado que físicamente el robot no puede hacerlo mientras mantenga la posición fija del efector final, la cual es la tarea más importante. Esto muestra que la tarea secundaria (el codo) se satisface en la medida de lo posible sin alterar la tarea primaria (el efector final). De manera más específica, en el movimiento anterior la tarea principal consiste en mantener el efector final en la posición fija $(0.27,\ 0.16,\ 0.94)$, con orientación hacia arriba, y la tarea secundaria consiste en mover el codo desde su posición actual $(0.45,\ 0.15,\ 0.48)$ hasta $(0.03,\ 0.42,\ 0.58)$. La Figura 5.16 (izquierda) muestra la trayectoria del codo desde dicha posición inicial hasta la posición final. Como se mencionó, no se alcanza la posición deseada debido a restricciones físicas, y por esto existe cierto error de estado estacionario. Sin embargo, como se muestra a la derecha, el efector final conserva en todo momento su misma posición y orientación.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

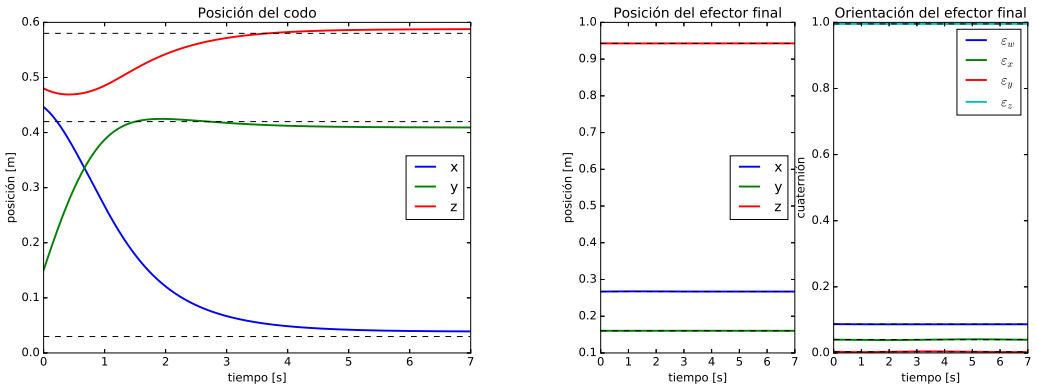


FIGURA 5.16: Posición del codo (izquierda), y del efector final (derecha) para el movimiento del codo que mantiene fijo al efector final (elaboración propia). El codo no llega exactamente a la posición deseada debido a restricciones impuestas por el efector final.

Debido a que se mantiene el efector final fijo, el movimiento realizado implica la coordinación de todas las articulaciones. Como se observa en la Figura 5.17, todas las 7 articulaciones del robot tienen movimiento que es automáticamente generado por el sistema propuesto, y que sería complicado de obtener de otro modo. En este caso no existe un valor de referencia en el espacio de las articulaciones dado que las referencias (para el efector final y el codo) se dan en el espacio Cartesiano, pero se observa que existe una convergencia hacia el final. Cabe resaltar que existen algunos aparentes sobre-impulsos, como el que se observa en la articulación q_5 entre los tiempos $t = 1$ y $t = 2$, pero que en realidad no constituyen un sobre-impulso. El motivo es que estos movimientos son automáticamente generados para alcanzar las especificaciones de movimiento tanto del efector final como del codo, no existiendo una referencia directa para los mismos.

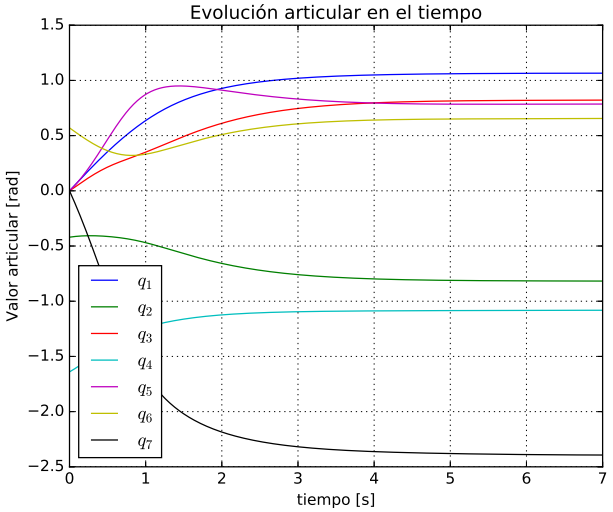


FIGURA 5.17: Evolución temporal de cada articulación del robot Sawyer cuando mueve el codo manteniendo fijo el efector final (elaboración propia).

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

La Figura 5.18 muestra la trayectoria Cartesiana que sigue el codo del robot desde su posición inicial hasta su posición final. El punto al final representa el punto de llegada deseado. La trayectoria del efector final no se muestra debido a que se mantiene constante a lo largo del movimiento del codo.

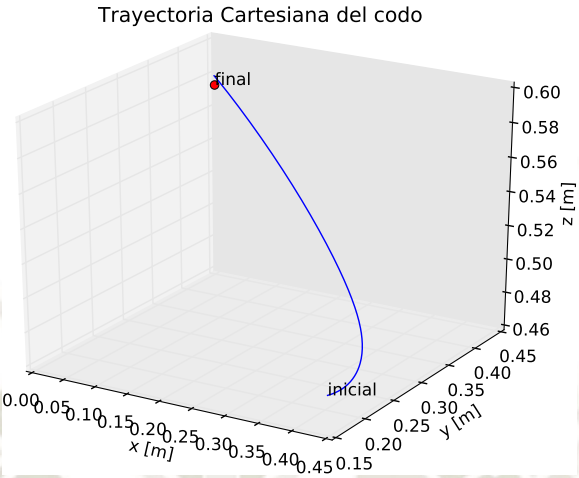


FIGURA 5.18: Trayectoria 3D seguida por el codo del robot mientras mantiene el efector final fijo (elaboración propia).

Es posible generar más movimiento de este tipo, donde se aproveche el espacio nulo en un robot manipulador redundante. No se muestran más ejemplos de esto debido a que el ejemplo anterior ilustra por completo esta idea, y otros movimientos tendrían un comportamiento similar.

Desplazamiento de Objetos en Gazebo. En esta sección se mostrará el uso del sistema desarrollado en esta tesis para generar movimiento del robot Sawyer en un simulador dinámico llamado *Gazebo*, el cual permite la interacción del robot con objetos de su entorno a través del uso de su efector final, como se muestra en la Figura 5.19.

La tarea consiste en desplazar tres cubos de diferentes colores, cada uno desde un punto inicial hasta un punto final. Esta tarea se muestra en la Figura 5.19, donde se puede observar que los bloques rojo, azul y verde son trasladados hacia la izquierda (desde el punto de vista de un observador que se encuentra delante del robot) manteniendo entre ellos sus posiciones relativas. El procedimiento para agarrar cada uno de los objetos consiste en: a) un movimiento hacia encima del bloque, b) un movimiento vertical hacia abajo hasta poner la garra a la altura del bloque seguido de un cierre de la garra, c) movimiento vertical hacia arriba, d) desplazamiento hacia la izquierda hasta quedar sobre la posición final, e) movimiento vertical hacia abajo hasta dejar el objeto sobre la mesa y abrir la garra, f) movimiento hacia el otro objeto. Esta secuencia se realiza para cada uno de los objetos y es especificada de forma manual. Una especificación automática podría ser realizada utilizando

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

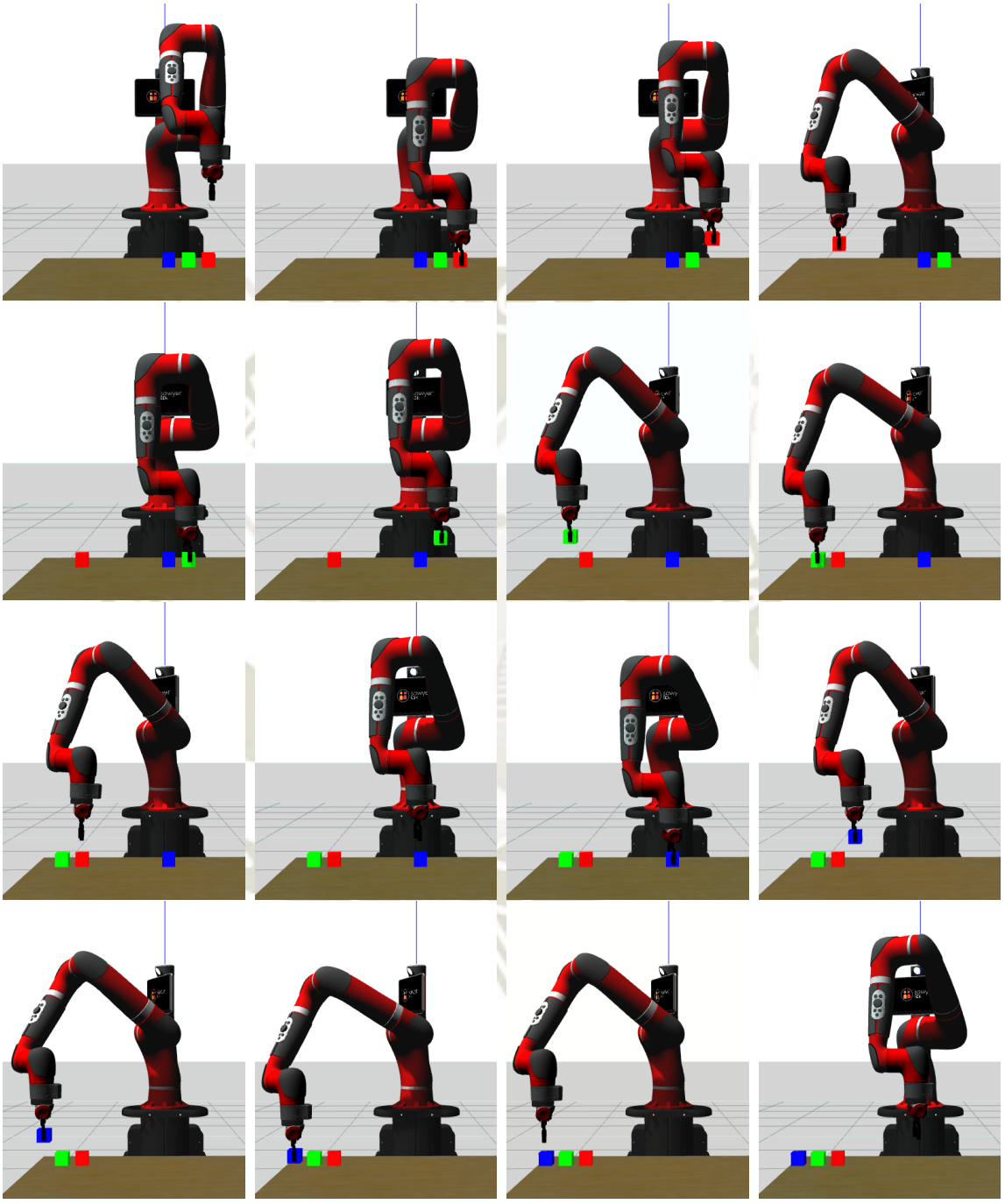


FIGURA 5.19: Robot Sawyer en el simulador dinámico Gazebo (derecha a izquierda, arriba a abajo) desplazando los tres cubos hacia la izquierda (elaboración propia).

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

técnicas de planificación de movimiento o de optimización de trayectorias, pero el uso de estos métodos que forman una especia de capa superior está fuera de los alcances de esta tesis.

La trayectoria de cada una de las articulaciones en estos movimientos no es obvia y no puede especificarse de forma manual sino que es obtenida de manera automática usando el método propuesto. Esta trayectoria se observa en la Figura 5.20 para cada una de las 7 articulaciones. Se puede observar que su evolución temporal es suave.

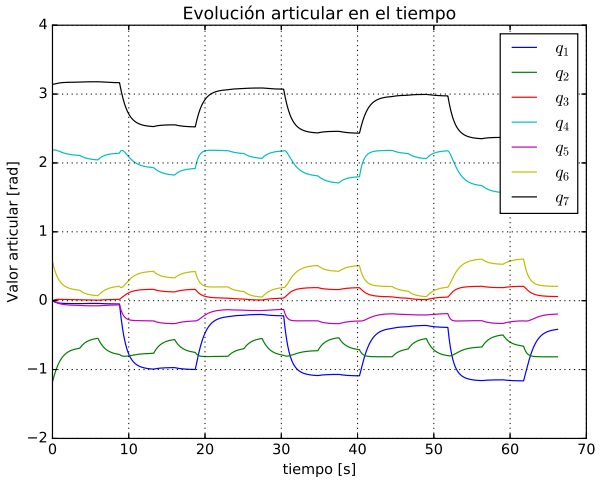


FIGURA 5.20: Trayectoria de cada articulación para el movimiento del robot Sawyer en el simulador dinámico Gazebo (elaboración propia).

Por otro lado, la Figura 5.21 muestra la evolución temporal del eje x, y, z del efector final del robot. Debido a que todos los bloques se encuentran al mismo nivel, el eje x (en azul) permanece constante. Para el eje z (en rojo) se puede observar que existe un movimiento repetitivo hacia abajo y hacia arriba, que se da para agarrar, trasladar y soltar el bloque. El movimiento del eje y (en verde) muestra claramente las posiciones iniciales de cada bloque y sus posiciones finales luego de haber sido trasladados. Las líneas punteadas muestran los valores de referencia, y las líneas sólidas los valores medidos de las articulaciones. Se puede observar que el sistema no presenta sobre-impulsos sino un comportamiento sobre-amortiguado debido a la elección que se hizo para el comportamiento del error, dado en (3.25).

5.2. Robot Simple de Base Flotante

Los resultados mostrados en la sección anterior demuestran que el sistema implementado puede ser utilizado para diversos robots manipuladores de manera adecuada. En esta sección se mostrará el uso del sistema para un robot con base flotante de solo 6 grados de libertad. Este robot fue creado desde cero para esta tesis, comenzando por la especificación de sus dimensiones y articulaciones,

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

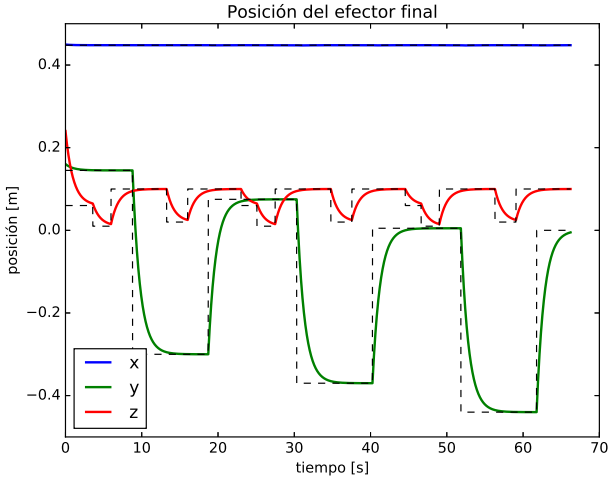


FIGURA 5.21: Trayectoria en x, y, z para el movimiento que consiste en desplazar los objetos en el simulador dinámico Gazebo (elaboración propia). Las líneas punteadas muestran los valores deseados para cada coordenada.

siguiendo con el cálculo analítico de su cinemática, y su implementación en un modelo URDF. El propósito de este robot es el de mostrar y validar el movimiento generado con el sistema propuesto en un robot arbitrario. El modelo del robot, así como todos los archivos completos que generan los movimientos presentados en esta sección se encuentran en el repositorio de github llamado: https://github.com/oeramosp/oscr_rosexample.

5.2.1. Especificaciones del robot

El robot desarrollado en esta sección tiene 6 grados de libertad articular: 3 grados en cada extremidad. Por facilidad a una extremidad se le denominará brazo y a la otra pierna (aunque podría dársele cualquier otro nombre sin pérdida de generalidad). La Figura 5.22 (izquierda) muestra el diseño de este robot y la localización del sistema de referencia de la base flotante $\{B\}$, que es el centro de la base, la cual tiene como dimensiones l_x, l_y, l_z. Se muestra además las dimensiones de cada eslabón (l_1 a l_6), así como el número de cada una de las articulaciones, donde las 3 primeras corresponden al brazo y las 3 últimas a la pierna.

Este modelo de robot fue implementado utilizando URDF (sección 4.1.2) y la especificación completa se encuentra en el apéndice B. La parte derecha de la Figura 5.22 muestra el modelo URDF en el visualizador RViz de ROS. Se puede ver que tanto el modelo previsto como el realmente implementado con URDF son similares. Así, cualquier modelo de robot puede ser descrito mediante este formato. Las dimensiones específicas que se utilizó para este robot son: $l_1 = l_2 = l_4 = l_5 = 0.2$m, $l_3 = l_6 = 0.05$m, y para el torso $l_y = l_z = 0.2$m, $l_x = 0.05$m. Todas las articulaciones de este robot son de revolución ya que actualmente casi no existen robots con articulaciones prismáticas.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

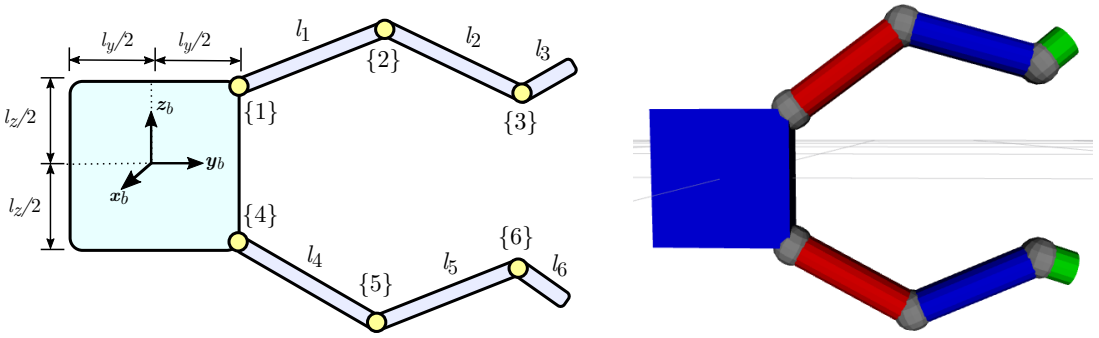


FIGURA 5.22: Modelo del robot de 6 gdl de base flotante: modelo diseñado (izquierda), modelo implementado con URDF y mostrado en RViz (derecha). Elaboración propia.

5.2.2. Análisis Cinemático Analítico

Para realizar el análisis cinemático de este robot se optó por un método geométrico que consiste en asignar arbitrariamente los sistemas de referencia al robot (alternativamente se pudo haber utilizado el método de Denavit-Hartenberg). Esta asignación de sistemas de referencia se muestra en la Figura 5.23 donde los ejes x se asumen perpendiculares y saliendo del plano de la hoja, ya que se sigue la regla de la mano derecha. No se muestra las dimensiones ya que estas se encuentran en la Figura 5.22. Los valores articulares van de q_1 a q_6 y especifican ángulos que van desde la dirección del eslabón anterior hasta la dirección del eslabón actual.

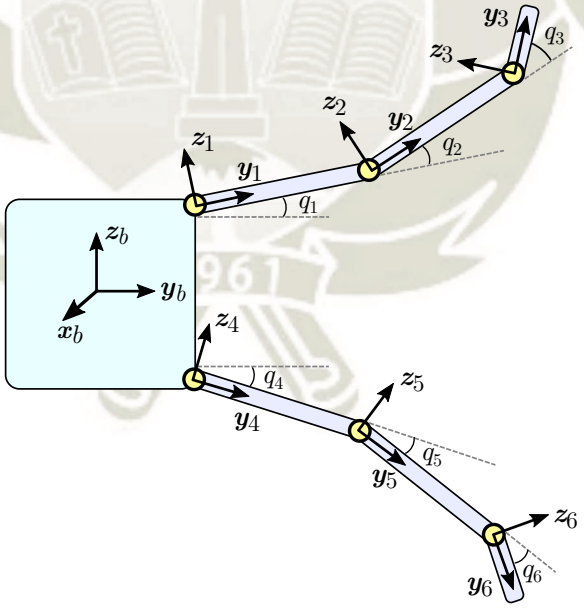


FIGURA 5.23: Asignación de coordenadas para el robot de 6 grados de libertad con base flotante, que se usará para su modelamiento cinemático (elaboración propia).

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

La posición y orientación del primer sistema de referencia, con respecto al sistema de la base $\{\mathcal{B}\}$ está dado por la siguiente coordenada Cartesiana y cuaternión:

$$ {}^{b}\boldsymbol{p}_1 = \left(0, \frac{l_y}{2}, \frac{l_z}{2}\right), \qquad {}^{b}Q_1 = \left(\cos\left(\frac{q_1}{2}\right), \sin\left(\frac{q_1}{2}\right), 0, 0\right) $$

donde para el cuaternión, el primer elemento es el componente escalar y los tres últimos elementos son el componente vectorial. Estos valores son obtenidos por inspección a partir del modelo mostrado. De la misma manera, la posición y orientación para el segundo eslabón es

$$ {}^{b}\boldsymbol{p}_2 = \left(0, \frac{l_y}{2} + l_1 \cos q_1, \frac{l_z}{2} + l_1 \sin q_1\right), \quad {}^{b}Q_2 = \left(\cos\left(\frac{q_1 + q_2}{2}\right), \sin\left(\frac{q_1 + q_2}{2}\right), 0, 0\right) $$

Para el último eslabón del brazo, el eslabón correspondiente a la articulación 3, se tiene:

$$ {}^{b}\boldsymbol{p}_3 = \left(0, \ \frac{l_y}{2} + l_1 \cos q_1 + l_2 \cos(q_1 + q_2), \ \frac{l_z}{2} + l_1 \sin q_1 + l_2 \sin(q_1 + q_2)\right) $$
$$ {}^{b}Q_3 = \left(\cos\left(\frac{q_1 + q_2 + q_3}{2}\right), \ \sin\left(\frac{q_1 + q_2 + q_3}{2}\right), \ 0, \ 0\right) $$

Para la primera articulación de la pierna se tiene:

$$ {}^{b}\boldsymbol{p}_4 = \left(0, \frac{l_y}{2}, -\frac{l_z}{2}\right), \qquad {}^{b}Q_4 = \left(\cos\left(\frac{q_4}{2}\right), \sin\left(\frac{q_4}{2}\right), 0, 0\right) $$

La segunda articulación de la pierna tiene la siguiente posición y orientación:

$$ {}^{b}\boldsymbol{p}_5 = \left(0, \frac{l_y}{2} + l_4 \cos q_4, -\frac{l_z}{2} + l_4 \sin q_4\right), \quad {}^{b}Q_5 = \left(\cos\left(\frac{q_4 + q_5}{2}\right), \sin\left(\frac{q_4 + q_5}{2}\right), 0, 0\right) $$

Finalmente la cinemática directa para el último eslabón de la pierna es:

$$ {}^{b}\boldsymbol{p}_6 = \left(0, \ \frac{l_y}{2} + l_4 \cos q_4 + l_5 \cos(q_4 + q_5), \ -\frac{l_z}{2} + l_4 \sin q_4 + l_5 \sin(q_4 + q_5)\right) $$
$$ {}^{b}Q_6 = \left(\cos\left(\frac{q_4 + q_5 + q_6}{2}\right), \ \sin\left(\frac{q_4 + q_5 + q_6}{2}\right), \ 0, \ 0\right) $$

Se puede notar en la descripción anterior que no se usó matrices de transformación homogénea debido a que en esta tesis se preferencia el uso de cuaterniones al uso de matrices de rotación. Sin embargo, a partir de la descripción anterior, y usando las conversiones entre parametrizaciones de la rotación, se podría llegar a la descripción con matrices de rotación. Por otro lado, la posición y orientación de cada eslabón ha sido calculada con respecto a la base flotante del robot. Para añadir el efecto del movimiento de la base flotante, se tendrá que realizar las transformaciones mencionadas en la sección 3.68 para la posición, y en la sección 3.72 para la orientación.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Los Jacobianos geométricos para este robot se pueden obtener por diferenciación de la parte correspondiente a la posición (ya que se utiliza coordenadas Cartesianas), y por inspección del eje de giro para la parte correspondiente a la velocidad angular, como se muestra en cualquier libro básico de robótica como [6, 7]. Los Jacobianos geométricos para las dos primeras articulaciones con respecto al sistema de la base son:

$$
{}^{b}J_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad {}^{b}J_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -l_1 \sin q_1 & 0 & 0 & 0 & 0 & 0 \\ l_1 \cos q_1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
$$

El Jacobiano geométrico del tercer eslabón (último eslabón del brazo) es:

$$
{}^{b}J_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -l_1 \sin q_1 - l_2 \sin(q_1 + q_2) & -l_2 \sin(q_1 + q_2) & 0 & 0 & 0 & 0 \\ l_1 \cos q_1 + l_2 \sin(q_1 + q_2) & l_2 \cos(q_1 + q_2) & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
$$

Para los dos primeros eslabones de la pierna del robot se tiene:

$$
{}^{b}J_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad {}^{b}J_5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -l_4 \sin q_4 & 0 & 0 \\ 0 & 0 & 0 & l_4 \cos q_4 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Finalmente, el Jacobiano geométrico del último eslabón de la pierna es:

$$
{}^{b}J_{6} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -l_4 \sin q_4 - l_5 \sin(q_4 + q_5) & -l_5 \sin(q_4 + q_5) & 0 \\
0 & 0 & 0 & l_4 \cos q_4 + l_5 \sin(q_4 + q_5) & l_5 \cos(q_4 + q_5) & 0 \\
0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Nuevamente en el caso de los Jacobianos geométricos no se incluye el efecto de la base flotante. Para añadir este efecto se puede añadir la parte correspondiente como se muestra en (3.21). Un reemplazo numérico de diferentes valores para cada una de las articulaciones (q_1 a q_6) brinda el modelo cinemático directo obtenido de manera analítica. Usando las ecuaciones de esta sección es posible verificar analíticamente que los valores del modelo, mostrados en la sección 4.4.1 y en la sección 4.4.2, los cuales fueron obtenidos usando la librería *Rbdl* y *Pinocchio*, son correctos. Como se mencionó anteriormente, este modelo analítico es ampliamente conocido en robótica y no brinda mayor novedad. Solo se realizó en esta sección para tener un modelo hecho desde cero y analizado analíticamente.

5.2.3. Generación de Movimiento

Lo que se desea mostrar en esta sección es la generación de movimiento usando el sistema propuesto para este robot de 6 grados de libertad con base flotante. Si se considerase la base como fija y se tratase a cada extremidad como solamente un manipulador, la posibilidad de movimiento sería reducida. Sin embargo, considerando la base como flotante se podrá brindar más movimiento al robot. Se debe tener en cuenta que este robot es completamente arbitrario (ficticio) y por tanto el movimiento generado solo servirá como una *prueba de concepto*.

El movimiento escogido consiste en mantener la posición de una extremidad fija y desplazar la otra hacia una posición y orientación deseadas. A la extremidad que se desplaza se le denominará *extremidad 1* y a la que mantiene su posición fija se le llamará *extremidad 2*. Para conseguir este movimiento se define 2 tareas, una asociada con cada extremidad. La tarea de la extremidad 1 considera tanto la posición como la orientación, como se describe en la sección 3.5.3, y la tarea de la extremidad 2 solamente considera la posición, como se describe en la sección 3.5.1. Ambas tareas son añadidas al sistema de control cinemático, especificando la extremidad 2 como primera tarea, y a la extremidad 1 como segunda tarea; es decir, la tarea de mantener fija la extremidad 2 tiene mayor prioridad que la de desplazar la otra extremidad. En este caso se utilizará un esquema

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

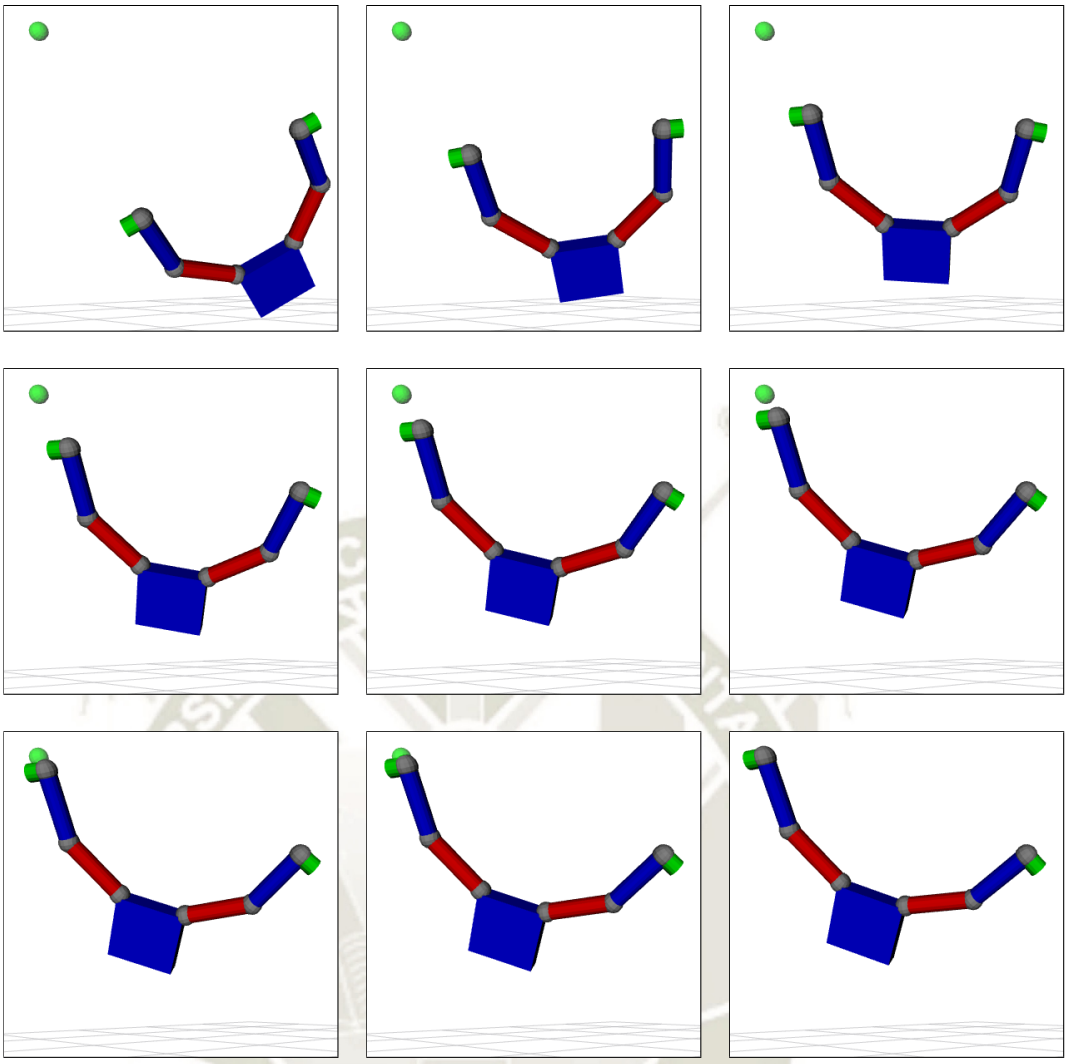


FIGURA 5.24: Movimiento del robot con base flotante para alcanzar la esfera verde del extremo superior izquierdo (elaboración propia). La base se mueve libremente.

de proyecciones sobre el espacio nulo (sección 3.4.2), pero se obtiene resultados similares con cualquiera de los otros métodos mencionados.

La Figura 5.24 muestra capturas de pantalla (de izquierda a derecha, y de arriba a abajo) del movimiento obtenido usando RViz en ROS. La extremidad de la derecha es la que se mantiene fija y la de la izquierda es la que se desplaza. El punto objetivo se muestra mediante una esfera verde a la izquierda. Se observa que el robot comienza abajo y poco a poco la extremidad izquierda sube aproximándose a la posición y orientación deseadas, pero manteniendo siempre posición de la extremidad derecha fija. Se puede observar además, que la base del robot (cuadrado azul) se desplaza libremente permitiendo el movimiento. Es este movimiento de la base flotante el que permite que el robot pueda realizar las tareas especificadas. Se puede observar además que tanto la posición como la orientación de la base flotante cambian en el tiempo.

La posición deseada (mostrada como la posición de la esfera verde) en este movimiento es $(0.97, -$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

$0.03,\ 1.06)$ m, y el cuaternión deseado es $(0.04,\ 0.93,\ 0.24,\ -0.29)$, cuya equivalencia en eje ángulo podría ser obtenida usando (2.14). La Figura 5.25 muestra la trayectoria obtenida para la extremidad 1 (la extremidad de la izquierda, la cual tiene movimiento), y las líneas punteadas muestran los valores deseados. Se puede observar que hay una convergencia desde los valores iniciales hasta los valores deseados, lo cual se observa también en la Fig 5.24. Adicionalmente puede verificarse que los cuaterniones obtenidos para la orientación son siempre unitarios.

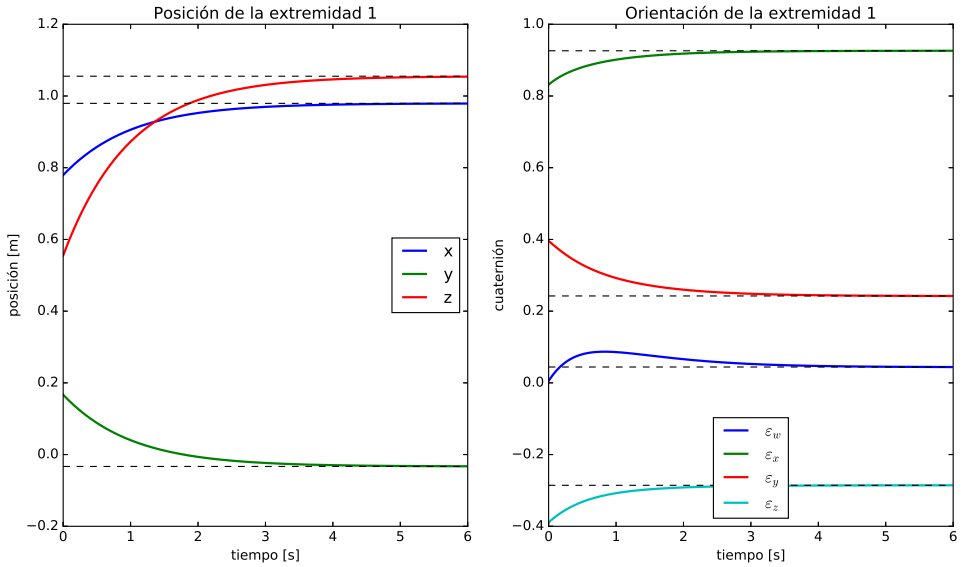


FIGURA 5.25: Posición y orientación de la extremidad izquierda, la cual se desplaza en el movimiento de ejemplo implementado (elaboración propia).

La extremidad derecha (la cual se mantiene fija) no varía su posición, como se muestra en la Fig 5.26. Se observa que x, y, z se mantienen constantes, tal como se desea. Sin embargo sí existe una variación en la orientación, debido a que la tarea solo mantiene fija la posición pero brinda libertad a la orientación. En este caso la orientación no se controla explícitamente, y los valores obtenidos son los que el sistema genera automáticamente para alcanzar las especificaciones del movimiento. Incluso si no se controla la orientación, se puede observar que no hay picos sino que el comportamiento es sobre-amortiguado.

El movimiento de la extremidad derecha (fija) podría implementarse en un sistema real si se contase con motores que tuviesen suficiente torque para soportar el peso de todo el resto del cuerpo, ya que el movimiento es similar al agarrarse con una mano de un pasamanos e intentar mover la otra mano hacia otro pasamanos, considerando que ambos pasamanos están a una distancia considerable. Dado que esta tesis se limita a la generación de trayectorias y asume la existencia de servomotores controlados en posición, no se realiza el análisis dinámico ni las implicancias que esto podría tener, suponiendo que los motores tendrán siempre suficiente torque (lo cual es normalmente el caso en

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

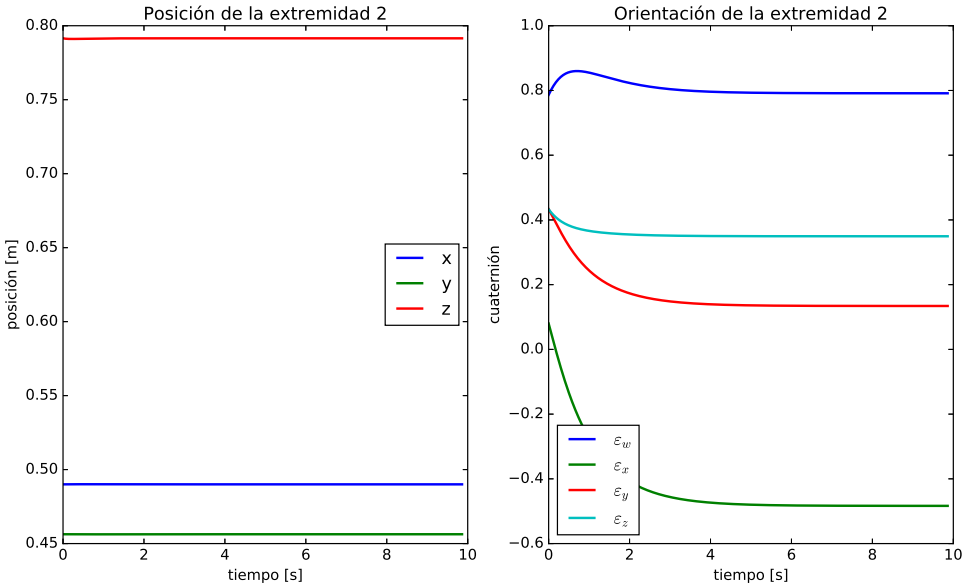


FIGURA 5.26: Posición y orientación de la extremidad derecha, cuya posición se mantiene fija a lo largo del movimiento, pero no la orientación (elaboración propia).

robots industriales y en robots que pueden hacer este movimiento, como *RoboSimian*, desarrollado por NASA-JPL).

Por otro lado, a pesar de que no existe un control directo sobre la posición y orientación de la base flotante, ésta se mueve para que la extremidad con movimiento alcance su posición y orientación deseadas. La Figura 5.27 muestra la evolución temporal de la posición y de la orientación de esta base flotante. Se observa un movimiento suave y generado de forma automática al considerar la base flotante en el desarrollo del sistema de control cinemático en esta tesis.

Finalmente, la Figura 5.28 muestra la variación de las 6 articulaciones en el tiempo para alcanzar el movimiento deseado. Como se ve en la Figura, las trayectorias articulares son suaves, sin picos y sin grandes variaciones, y podrían ser ingresadas como referencia a un robot controlado en posición, si este robot existiese. Además, sería casi imposible generar estas trayectorias articulares de forma manual para satisfacer los objetivos de este movimiento, siendo necesario el uso del sistema propuesto.

El código completo de este movimiento se encuentra en la carpeta `scripts` del repositorio `https://github.com/oeramosp/oscr_rosexample`. Usando este repositorio, es posible generar diferentes movimientos para este robot creado, pero no se mostrará más ejemplos aquí debido a que los resultados son similares a los mostrados.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

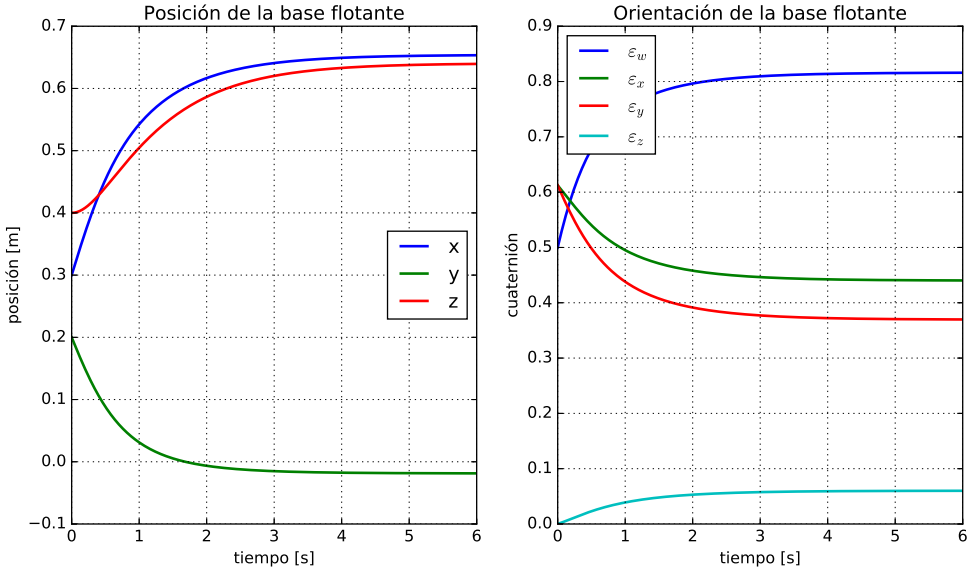


FIGURA 5.27: Posición y orientación de la base flotante para lograr el movimiento deseado, a pesar de que no existe un control directo sobre esta base (elaboración propia).

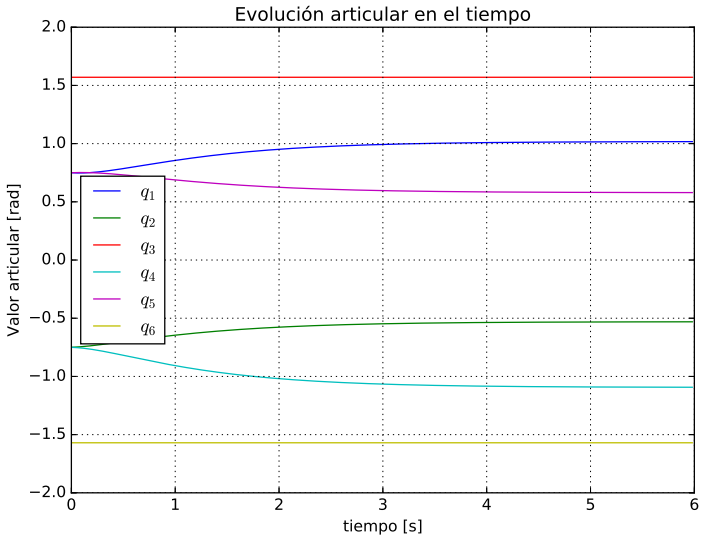


FIGURA 5.28: Configuración articular en función del tiempo para el movimiento mostrado (elaboración propia). Se observa la suavidad de la trayectoria generada.

5.3. Movimiento para Robots Humanoides

En esta sección se presentará la aplicación del sistema de generación de movimiento propuesto en esta tesis a dos ejemplos concretos de robots humanoides: el robot NAO y el robot Atlas. Se utilizó estos dos robots debido a que, desde un punto de vista cinemático, ambos tienen una estructura cinemática similar (una cabeza, dos brazos, dos piernas), siendo la principal diferencia el tamaño de los mismos: NAO es un robot pequeño, y Atlas es un robot grande. Además, Atlas posee algunos grados de libertad adicional que no posee NAO. Físicamente, esta diferencia de tamaños también

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

se ve reflejada en el tipo de motores que usan ambos robots, y las capacidades físicas de carga de los mismos. Sin embargo, en esta tesis se asume que cada robot tiene su sistema de control interno de motores y que éstos son capaces de seguir la referencia de posición indicada, lo cual se cumple para ambos robots. Es decir, el movimiento generado en esta sección, podría ser directamente enviado como referencia a alguno de estos robots de forma real. La versatilidad del sistema de control cinemático permite un esquema de control similar para ambos robots. En general, el sistema propuesto se podría aplicar a cualquier otro robot humanoide, siempre que se cuente con su modelo URDF.

5.3.1. Robot NAO

El robot NAO es un robot desarrollado por la compañía de origen francés, y ahora de capitales principalmente japonenes, llamada *SoftBank Robotics* (e inicialmente llamada *Aldebaran Robotics*). NAO es un robot humanoide pequeño, de 58 centímetros de alto, que tiene 26 grados de libertad y es controlado en posición; es decir, cada uno de sus motores tiene un lazo de control de posición interno que le permiten comportarse como un servomotor. Los grados de libertad de este robot se encuentran distribuidos de la siguiente manera: 6 por cada pierna, 5 en cada brazo, 2 en la cabeza, y 2 en cada mano (para abrir y cerrar la mano). La Figura 5.29 muestra al robot real (izquierda) y al robot en RViz de ROS (derecha).

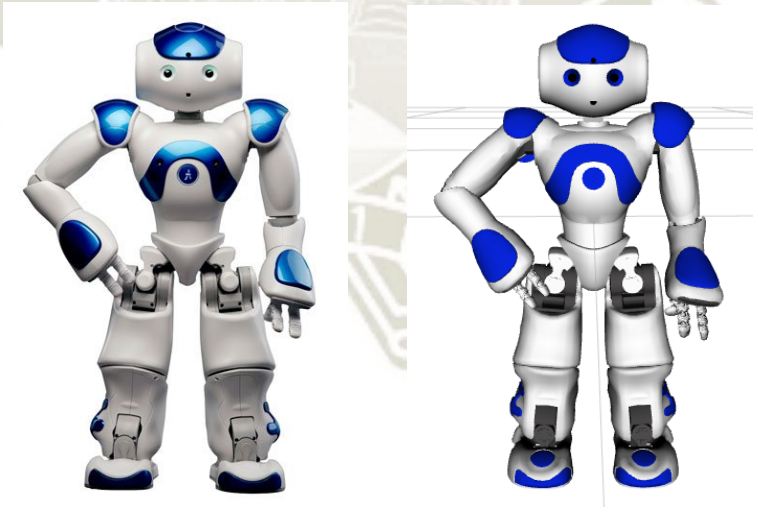


FIGURA 5.29: Robot NAO de SoftBank Robotics: real (izquierda, tomado de SoftBank Robotics), y en el visualizador RViz de ROS (derecha, elaboración propia).

Todos los scripts utilizados para el movimiento del robot NAO mostrados en esta sección se encuentran en el repositorio de git `https://github.com/oeramosp/nao_oscr`.

Movimiento de Ambos Brazos. Para mostrar la real efectividad del sistema implementado en esta tesis se comandará dos acciones simultáneas al robot NAO manteniendo ambos pies fijos sobre

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

el terreno. Estas acciones consistirán en mover tanto la mano derecha como la mano izquierda hacia una posición y orientación predeterminadas y elegidas arbitrariamente. De este modo se tiene una tarea para cada una de las manos. Además, se tendrá dos tareas adicionales para los pies, las cuales consisten en mantener tanto la posición como la orientación de cada pie; es decir, no mover el pie. Estas tareas de los pies se deben a que al no controlar alguna parte del cuerpo de manera explícita, esta parte tiene libertad para moverse de cualquier manera en favor del movimiento deseado (controlado). Así, si no se controlase los pies, éstos podrían presentar algún tipo de movimiento lo cual podría llevar al robot real a perder contacto con el piso, y el robot podría caer (una vez que el robot cae, el movimiento ya carece de sentido). Por este motivo, las tareas de ambos pies son las que tienen la mayor prioridad, y luego, en prioridad, vienen las tareas de las manos.

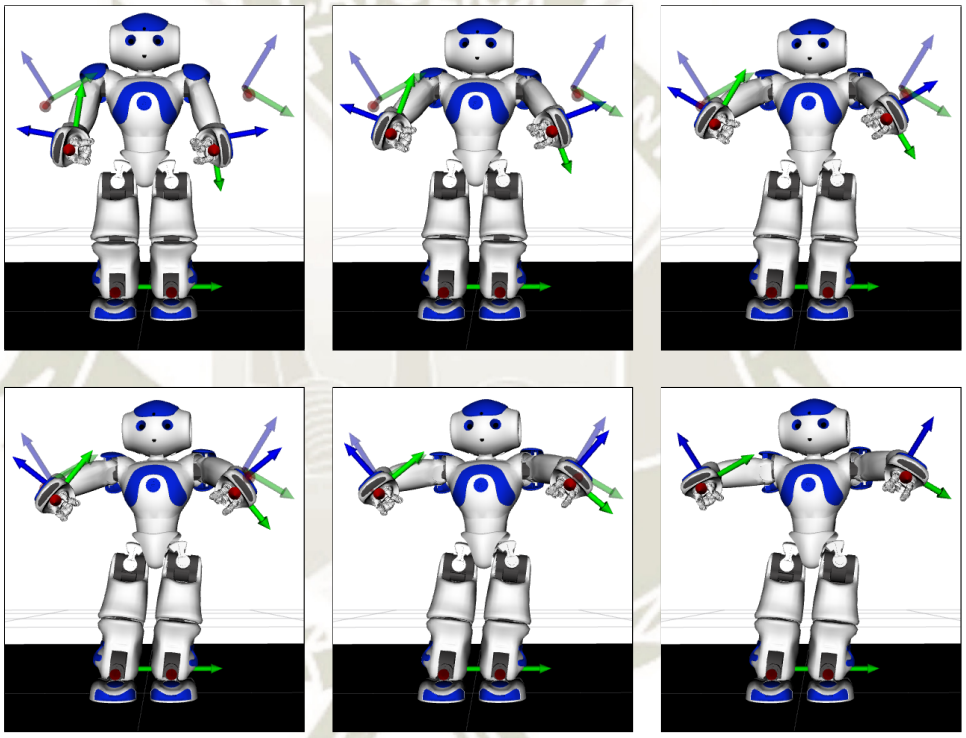


FIGURA 5.30: Movimiento de ambos brazos del robot humanoide NAO considerando una base flotante (elaboración propia). Notar el movimiento de todo el cuerpo del robot.

El movimiento generado se observa en la Figura 5.30 donde los sistemas de referencia (flechas roja, verde y azul) indican la posición y orientación actual de las manos y su valor deseado. Se observa que el robot comienza con ambas manos y piernas colocadas de manera simétrica, y conforme avanza el movimiento, ambas manos alcanzan sus valores deseados y además el cuerpo del robot también realiza algunos movimientos que ayudan a alcanzar este objetivo. Por ejemplo, se puede observar que las piernas presentan cierta inclinación hacia la derecha (desde el punto de vista del observador), y que el torso también presenta cierto movimiento. Este movimiento, llamado de cuerpo completo tiene la ventaja que permite alcanzar algunas configuraciones que de otro modo no

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

sería posible alcanzar, en especial para NAO ya que solo posee 5 grados de libertad por cada brazo; es decir, uno menos que los necesarios, considerando que el espacio tridimensional tiene 6 grados de libertad (3 de posición y 3 de orientación). El movimiento de cuerpo completo se puede verificar con la trayectoria de las articulaciones, mostradas en la Figura 5.31, donde se observa que todas las articulaciones varían con el tiempo, pero convergen a un valor especificado indirectamente por los objetivos de las tareas.

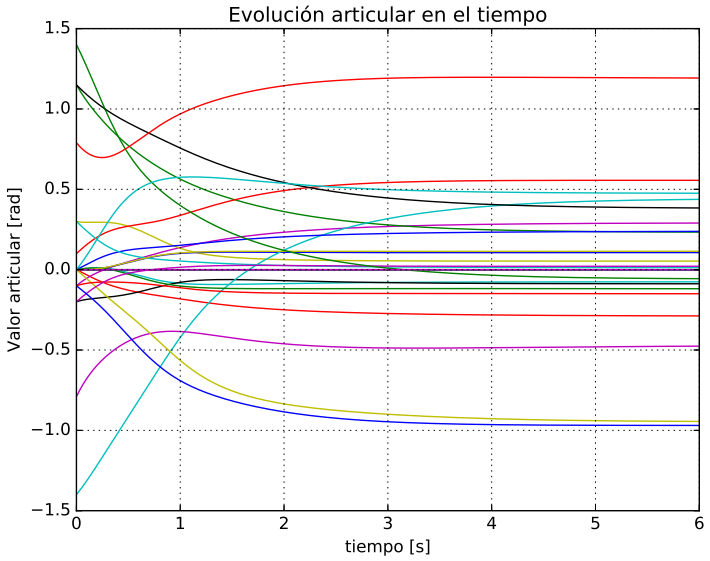


FIGURA 5.31: Trayectoria de las 26 articulaciones del robot NAO en función del tiempo (elaboración propia). Se observa una trayectoria suave para cada articulación.

La verificación de que las tareas asignadas a la mano derecha e izquierda se alcanzan de manera adecuada se muestra en la Figura 5.32 para la mano derecha, y en la Figura 5.33 para la mano izquierda. Se puede observar el comportamiento *exponencial* del sistema debido a la trayectoria de referencia especificada en (3.25).

Es importante además observar que, debido a que se genera movimiento considerando la base del robot como flotante, dicha base (en este caso el torso del robot) no permanece fija sino que se mueve para ayudar al alcance de la tarea especificada. Este movimiento se presenta en la Figura 5.34, donde se muestra que la base del robot se mueve ligeramente hacia el costado (y) y hacia atrás (x) pero mantiene su altura (z). La orientación también presenta pequeñas variaciones obtenidas de manera automática.

Por otro lado, la Figura 5.35 muestra el tiempo de cálculo del algoritmo para cada una de las iteraciones realizadas, en una computadora con 8Gb de RAM y un procesador Intel Core i7. Para el robot NAO, el control de posición debe ser enviado cada 10 ms para un adecuado funcionamiento, y los resultados muestran que el cálculo toma en promedio 1.5 ms, lo cual permite aplicar este sistema de control cinemático en tiempo real. Existen algunos picos, debido a que el Kernel en el

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

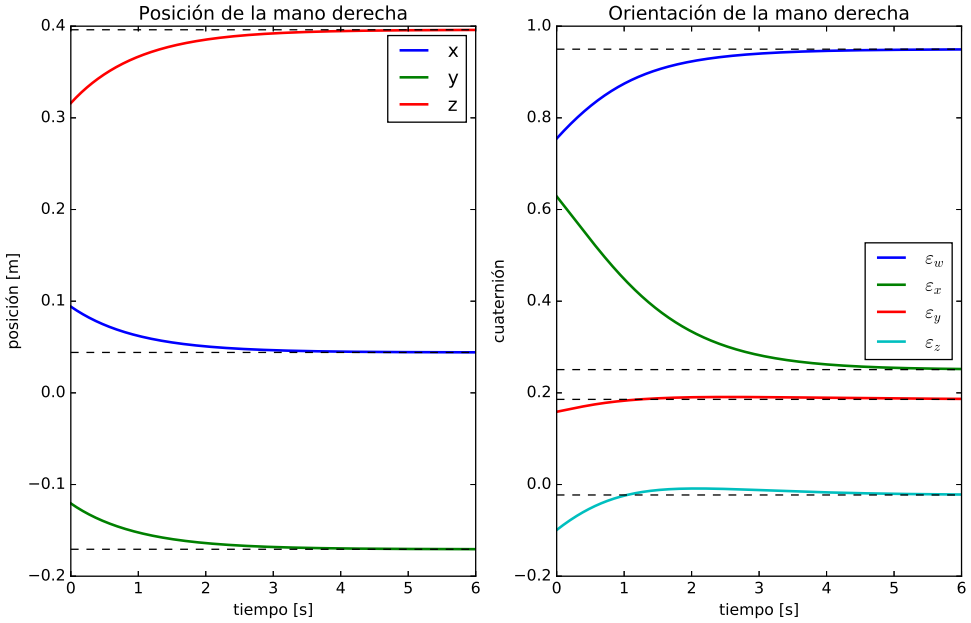


FIGURA 5.32: Posición y orientación de la mano derecha del robot NAO (elaboración propia). Las líneas punteadas muestran los valores de referencia.

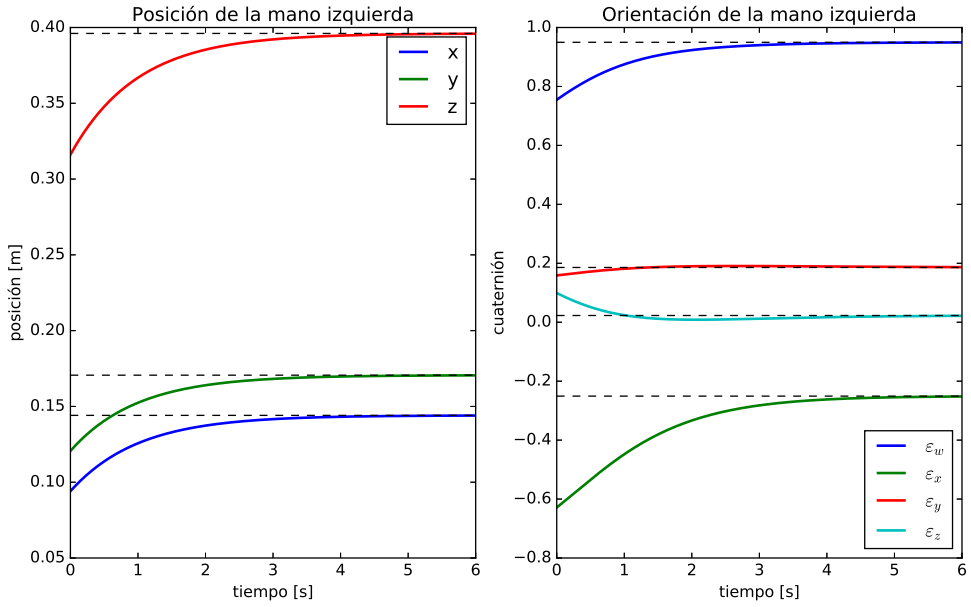


FIGURA 5.33: Posición y orientación de la mano izquierda del robot NAO (elaboración propia). Las líneas punteadas muestran los valores de referencia.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

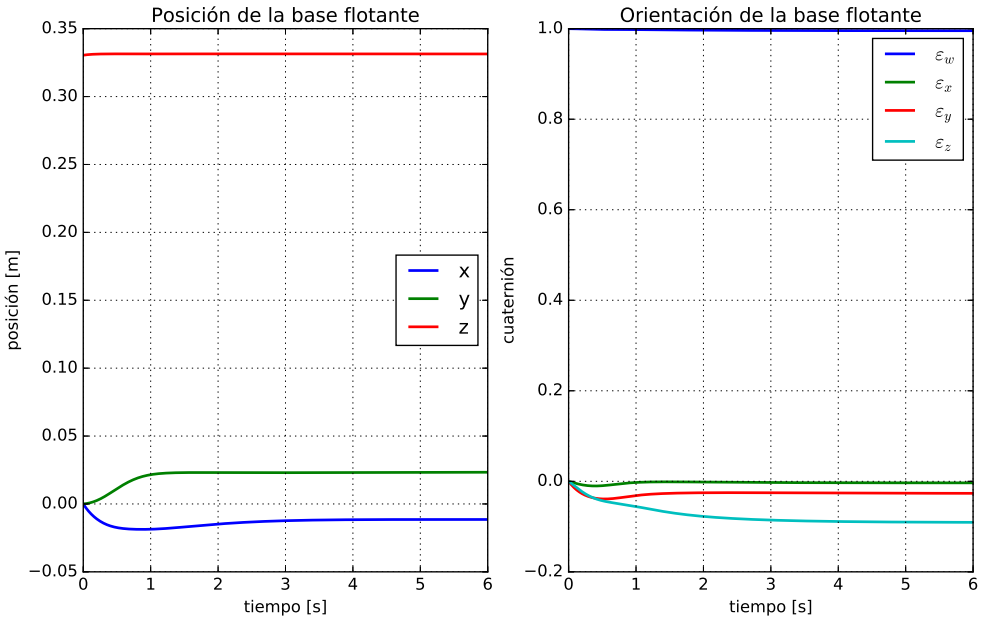


FIGURA 5.34: Posición y orientación de la base del robot, la cual presenta cierto movimiento para ayudar al objetivo principal (elaboración propia).

que se probó el sistema no es de tiempo real y por tanto no se está controlando directamente las actividades y prioridades del procesador, pero nunca superan los 3 ms. Evidentemente, el uso y control de un Kernel de tiempo real, como *Xenomai*, *RTLinux*, o algún otro, brindaría menos *jitter*. pero debido a que el tiempo obtenido con un Kernel tradicional de Linux (Ubuntu 16.04, en este caso) es menor a los 10 ms, dicha implementación está fuera de los alcances de esta tesis.

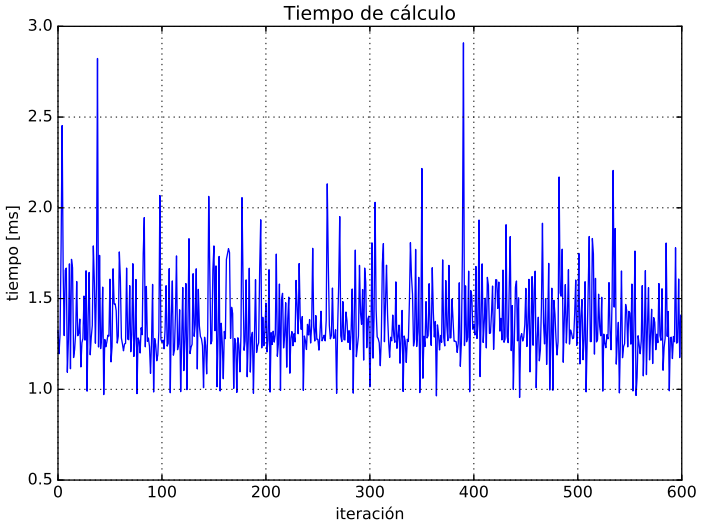


FIGURA 5.35: Tiempo de computación (en ms) para la generación de cada ciclo de control en el movimiento de ambos brazos (elaboración propia).

Como se mencionó, todo el código utilizado para generar el movimiento mostrado se encuentra en

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

el repositorio de github. Por facilidad, y para mostrar lo rápido del prototipaje en Python, el código se muestra en el Listado 5.1.

Listado 5.1. Programa en Python, usando el sistema implementado en esta tesis, para el movimiento de ambos brazos mostrado en las figuras anteriores (elaboración propia).

```
import numpy as np
import rospy
from oscr.ros_kine_sim import RosKineSim
from oscr.utils import incPosition, incPoseLocal

if __name__ == '__main__':
    # Frecuencia de actualizacion (100 Hz = 10 ms)
    f = 100.0
    # Instancia de simulacion del robot
    sim = RosKineSim('naoMotion', 'nao_oscr', '/urdf/naoV40red.urdf', f, True, 'pin')
    # Configuracion articular
    q = np.array([[0.0, 0.0, 0.33046, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                   0.0, 0.0, -0.1, 0.3, -0.2, 0.0, 1.15,  0.10, -1.4, -0.79, 0.0, 0.0,
                   0.0, 0.0, -0.1, 0.3, -0.2, 0.0, 1.15, -0.10,  1.4,  0.79, 0.0, 0.0]]).T
    # Enviar la configuracion articular al robot
    sim.update(q)
    # Inicializar el sistema de control cinematico
    sim.initKineSolver(q)
    # Tareas
    sim.initKineTaskPose('r_wrist','pose','RHand')
    sim.initKineTaskPose('l_wrist','pose','LHand')
    sim.initKineTaskPose('l_ankle','pose','LAnkle')
    sim.initKineTaskPose('r_ankle','pose','RAnkle')
    # Mantener los pies sin movimiento
    sim.task['LAnkle'].keep(10.0); sim.task['RAnkle'].keep(10.0)
    # Posicion y orientacion deseados para la mano derecha
    currentRHand = sim.task['RHand'].getSensedValue()
    desiredRHand = incPoseLocal(currentRHand, (-0.05, -0.05, 0.08), (-50., (1., 0., 0.)))
    sim.task['RHand'].setDesiredValue(desiredRHand)
    # Posicion y orientacion deseados para la mano izquierda
    currentLHand = sim.task['LHand'].getSensedValue()
    desiredLHand = incPoseLocal(currentLHand, (0.05, 0.05, 0.08), (50., (1., 0., 0.)))
    sim.task['LHand'].setDesiredValue(desiredLHand)
    # Adicion de tareas al sistema de control cinematico
    sim.pushTask(sim.task['LAnkle'])
    sim.pushTask(sim.task['RAnkle'])
    sim.pushTask(sim.task['RHand'])
    sim.pushTask(sim.task['LHand'])
    # Bucle principal en ROS
    while not rospy.is_shutdown():
        qdes = sim.getPositionControl(q)
        sim.update(qdes)
        q = qdes.copy()
        sim.rate.sleep()
```

Puede observarse que se define primero la frecuencia (100 Hz en este caso) y luego la clase `RosKineSim`, la cual está escrita puramente en Python y contiene a su vez las clases `RobotModel`,

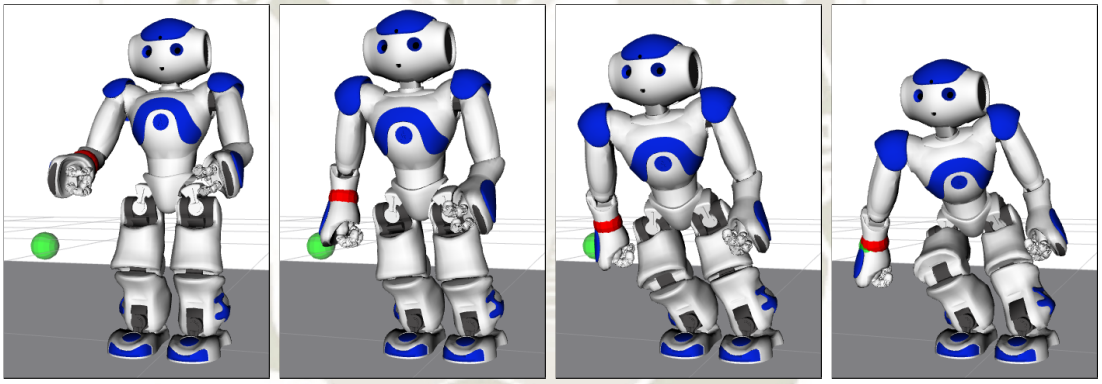
Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

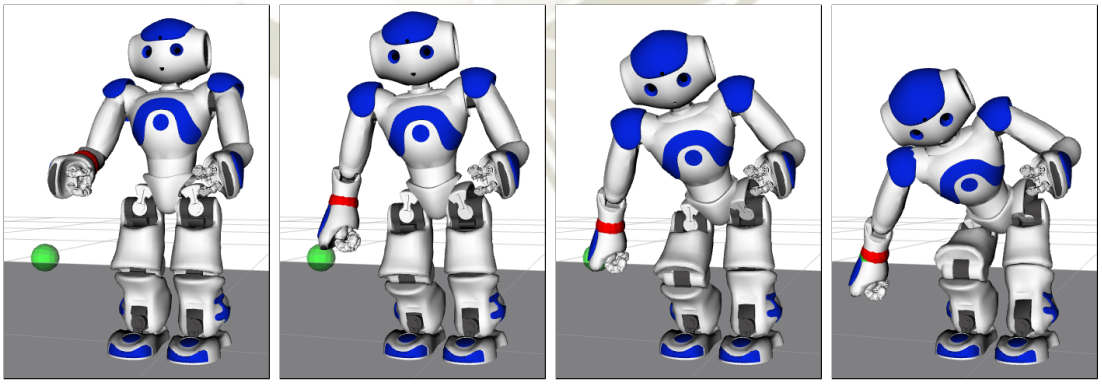
UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

KineTask y `OSIKSolver` descritas en el capítulo anterior. Luego se inicializa una configuración articular inicial, se inicializa el sistema de control cinemático, el cual en este caso usará el esquema HQP. Luego se crea las 4 tareas, para las manos y para los pies, haciendo referencia a los eslabones respectivos del modelo URDF del robot. Se indica luego que los pies no se moverán, y luego se brinda una posición y orientación incremental de referencia tanto para la mano derecha, como para la mano izquierda. Finalmente se añade las tareas al sistema de control cinemático y se realiza el bucle principal que genera las trayectorias deseadas.

Otros Ejemplos de Movimiento. Aprovechando la versatilidad del sistema de generación de movimiento, se mostrará algunos ejemplos de movimientos adicionales. En la Figura 5.36a se muestra un movimiento en el cual la mano derecha del robot debe moverse hacia abajo manteniendo la orientación de la cabeza constante y manteniendo ambos pies sobre el piso. La esfera verde muestra la posición deseada para la mano.



a) Movimiento de la mano derecha manteniendo la orientación de la cabeza



b) Movimiento de la mano derecha manteniendo fija la mano izquierda

FIGURA 5.36: Dos movimientos diferentes del robot NAO manteniendo fijas la posición y orientación de los pies para no perder contacto con el piso (elaboración propia).

Como se observa, el robot se agacha para poder alcanzar dicha posición ya que de otro modo no sería posible. Este es un claro ejemplo de la ventaja que tiene el uso de la base flotante, la cual permite, en este caso, alcanzar lo deseado. Se debe además notar que el movimiento de agacharse

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

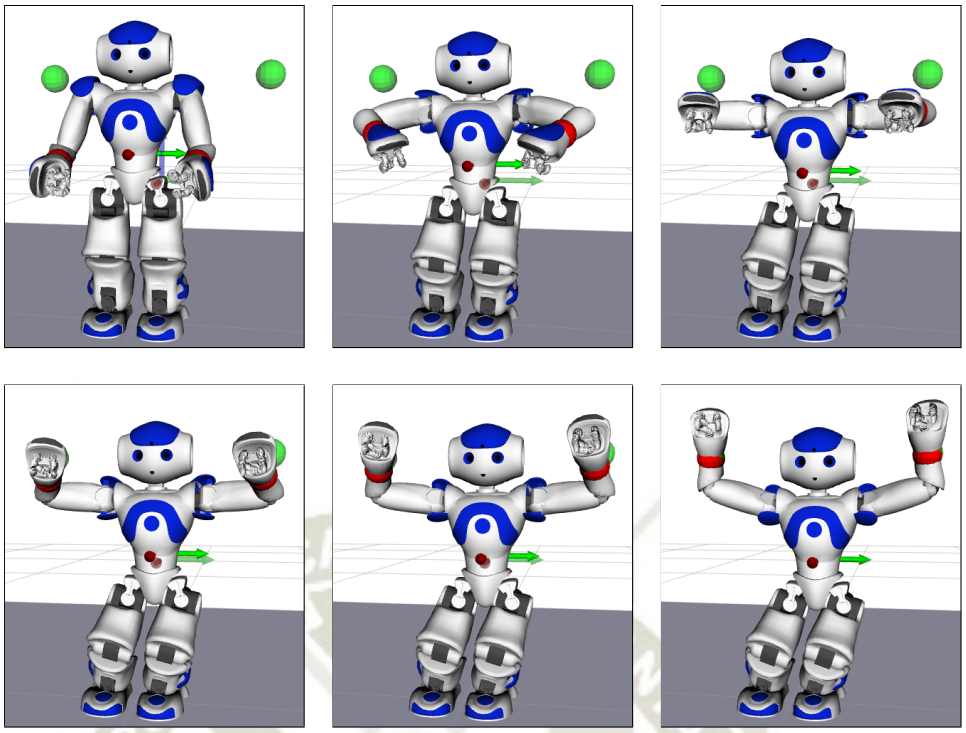


FIGURA 5.37: Movimiento simultáneo de la base del robot NAO hacia la izquierda y de ambos brazos hacia arriba (elaboración propia). Notar que no existe una articulación específica que mueva la base, sino una coordinación de ambas piernas.

se genera de manera automática: no existe ningún comando que diga explícitamente que el robot debe hacer eso. Sin embargo, el sistema de generación de movimiento coordina adecuadamente la trayectoria de todas las articulaciones para alcanzar el objetivo deseado. Se puede, además, notar que la mano izquierda no tiene ninguna tarea asociada, y su movimiento se puede considerar como residual, producto de las otras tareas.

La Figura 5.36b muestra un movimiento similar para la mano derecha, pero en este caso no existe ninguna tarea para la cabeza sino una tarea adicional para la mano izquierda, la cual debe mantener su posición y orientación fijas. Se puede observar que debido a esto la cabeza se mueve libremente y al final termina con cierta inclinación, mientras que la mano izquierda permanece con una posición y orientación constantes a lo largo de todo el movimiento. Estas múltiples tareas se pueden alcanzar debido a la redundancia que posee el robot NAO, al igual que cualquier robot humanoide.

Otro ejemplo de movimiento se muestra en la Figura 5.37. Aquí se tiene 5 tareas: una para la base flotante, una para cada mano del robot, y una para cada pie. Las tareas del pie tienen la más alta prioridad y los mantienen fijos sobre el piso. La tarea del movimiento de la base flotante tiene la siguiente prioridad y desplaza la base o la cadera del robot hacia la derecha y hacia abajo. Al mismo tiempo, ambas manos se desplazan hacia una posición determinada. Este es un claro ejemplo de la riqueza de movimiento que se puede obtener con el sistema implementado. Nuevamente se

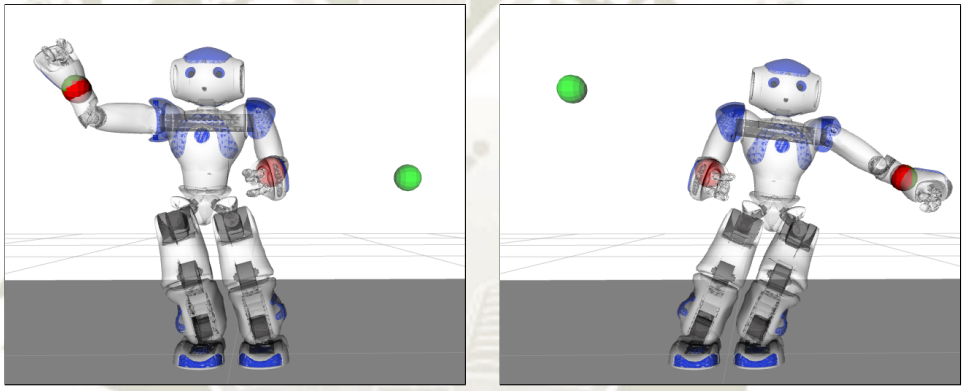
Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

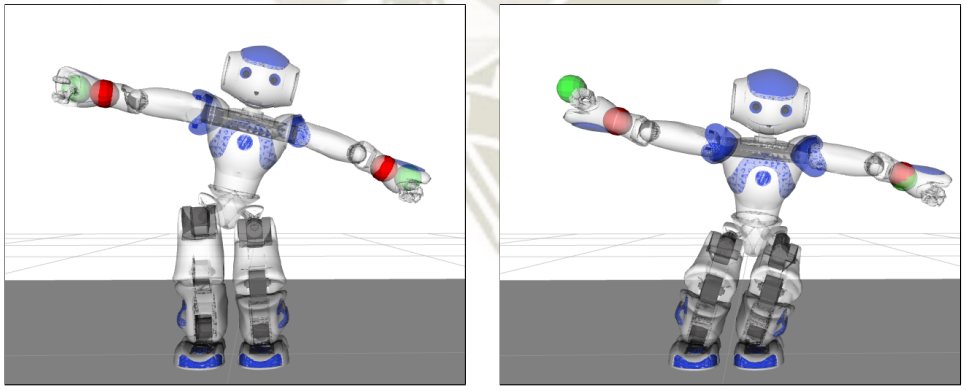
UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

puede observar que el movimiento de las piernas y de las demás articulaciones del robot es generado de manera automática. No se muestra las trayectorias articulares ni Cartesianas debido a que son similares a las mostradas para el primer movimiento del robot NAO y no agregarían mayor información.

Tareas Conflictivas. Los diferentes esquemas de control cinemático presentados en la sección 3.4 tienen, en general, comportamientos similares. La principal diferencia entre los esquemas con priorización y los esquemas con ponderación de tareas ocurre cuando existen tareas conflictivas. La Figura 5.38a muestra dos tareas, una para cada mano: la esfera verde es la posición deseada y la esfera roja es la posición actual (se muestra al robot con una cierta transparencia para hacer más visible el error entre lo actual y lo deseado). Por separado, cada tarea puede ser adecuadamente alcanzada por el robot usando el movimiento de la base flotante.



a) Movimiento independiente de la mano derecha y de la mano derecha



b) Ambas manos no pueden alcanzar el objetivo a la vez: WQP (izq.), HQP (der.)

FIGURA 5.38: Movimiento de ambos brazos con tareas conflictivas usando el esquema de control cinemático ponderado WQP, y el jerárquico HQP (elaboración propia).

Sin embargo, ambas tareas no pueden ser alcanzadas a la vez debido a restricciones físicas del robot. La Figura 5.38b muestra este caso. En la figura de la izquierda se usó un esquema basado en ponderación de tareas (WQP: *weighted* QP): se puede observar que existe un compromiso entre ambas tareas y ninguna se alcanza de manera completa; es decir, ninguna mano llega a su posición

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

deseada sino a una posición intermedia entre ambas tareas. Por otro lado, la figura de la derecha utiliza un esquema basado en priorización de tareas (HQP: *hierarchical* QP), donde la tarea de la mano izquierda del robot tiene mayor prioridad que la tarea de la mano derecha. En este caso se observa que efectivamente la mano izquierda alcanza su posición deseada y la mano derecha trata de alcanzar la suya de la mejor manera posible, a pesar de que nunca podrá hacerlo debido a restricciones físicas.

El resultado mostrado es típico de ambos esquemas de control cinemático. En el caso del uso de proyecciones sobre el espacio nulo, el resultado es similar al obtenido con HQP debido a que también es un esquema que tiene prioridad. Los tiempos de computación en todos los casos son similares y, debido a que se asemejan al presentado en la Figura 5.35, no se presentarán nuevamente aquí. El uso de uno u otro esquema de control dependerá de la aplicación específica y del resultado que se espera, no existiendo una diferencia considerable a menos que se tenga especificaciones de objetivos conflictivos para las tareas, como se presentó en este caso. Debido a que la mayoría de tareas no son conflictivas, no se hará más hincapié en la diferencia existente entre estos esquemas de control cinemáticos.

Movimiento Acrobático. Una de las ventajas del sistema implementado con respecto a sistemas de control cinemático tradicionales de brazos manipuladores, y donde realmente se puede apreciar la capacidad de generación de movimiento, es el poder mover varias partes del robot de manera simultánea. Así, es posible generar movimiento coordinado de todo el cuerpo, moviendo todas las articulaciones, mediante la especificación de diferentes tareas relativamente simples para cada una de las partes típicamente en el espacio Cartesiano. Para ejemplificar esta posibilidad de generación de movimiento para un robot humanoide, se realizará una secuencia de movimiento acrobático. Esta secuencia se muestra en la Figura 5.39 de izquierda a derecha y de arriba a abajo. Como se observa en la figura, este movimiento consiste en las siguientes partes:

- Desplazar la cadera del robot hacia la derecha con el fin de no perder el equilibrio estático en movimientos posteriores.
- Manteniendo la posición horizontal (en x, y) de la cadera, levantar el pie izquierdo. Se requiere mantener la posición horizontal con el fin de garantizar que el robot no caerá al levantar el pie.
- Estirar ambas manos hacia adelante y el pie izquierdo hacia atrás, siempre manteniendo la cadera sobre el pie de apoyo.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

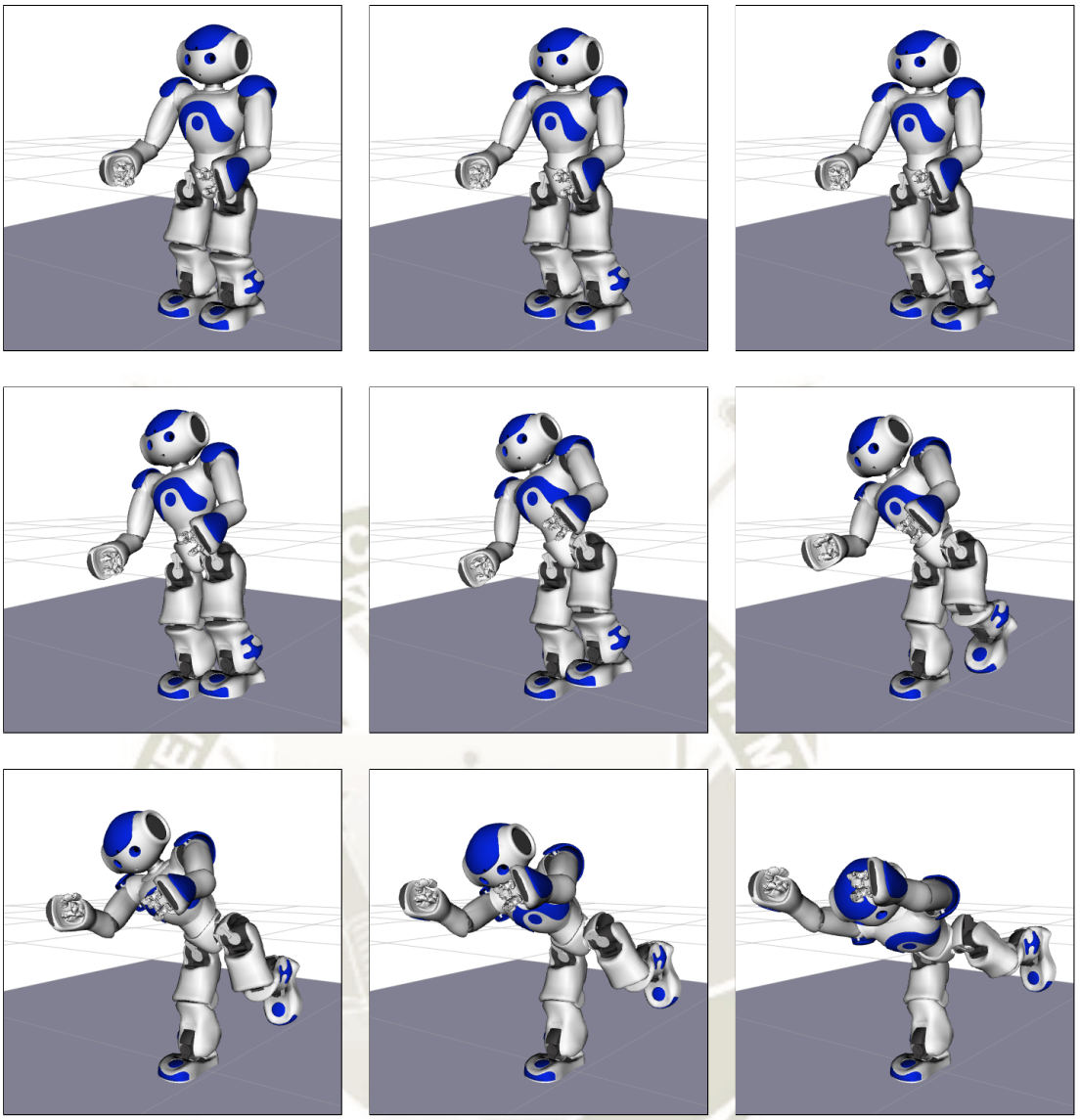


FIGURA 5.39: Movimiento acrobático del robot NAO: el robot se inclina hacia la derecha (sobre el pie derecho), levanta la pierna izquierda, y luego mueve ambos brazos hacia adelante mientras que el pie se mueve hacia atrás (elaboración propia).

- El movimiento culmina con el robot sostenido solo por su pie derecho, pero con las extremidades estiradas.

Luego de realizar este movimiento, el robot regresa a su posición inicial describiendo un movimiento exactamente opuesto al mostrado. Debido a que el movimiento de regreso al inicio es en esencia similar al descrito, pero en sentido opuesto, no se muestra en la figura. De esta manera se puede verificar visualmente los postulados del sistema de generación de movimiento: permitir mover las diversas partes del robot de manera simultánea. Se debe notar que, dado que se utiliza una base flotante, la base del robot puede comandarse como cualquier otra parte del cuerpo, a pesar de que no existe una articulación específica que desplace la base.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

La secuencia específica de tareas para alcanzar el movimiento mostrado se especifica en la Figura 5.40. La parte superior muestra las tareas más importantes y la parte inferior las tareas menos importantes, en orden jerárquico. El eje horizontal representa el tiempo: lo que se encuentra a la izquierda sucede primero, y la secuencia termina con lo que se encuentra a la derecha. Cada fila representa la tarea de un punto operacional, y los cuadros blancos indican que durante ese intervalo dicha tarea no existe (ha sido removida del sistema de control cinemático). Para mantener el esquema jerárquico se utilizó el sistema basado en HQPs.

<table>
<tr><td>Contacto pie izquierdo</td><td colspan="6"></td><td>Contacto pie izquierdo</td></tr>
<tr><td colspan="7">Contacto pie derecho</td></tr>
<tr><td>Base a la derecha</td><td colspan="5">Mantener la posición de la base (x,y)</td><td>Base a la izquierda</td><td></td></tr>
<tr><td></td><td>Levantar pie izquierdo</td><td>Pie izquierdo hacia atrás</td><td>Mantener pie izquierdo</td><td>Pie izquierdo hacia adelante</td><td>Bajar pie izquierdo</td><td colspan="2"></td></tr>
<tr><td colspan="2"></td><td>Mano derecha hacia adelante</td><td>Mantener mano derecha</td><td>Mano derecha hacia atrás</td><td colspan="3"></td></tr>
<tr><td colspan="2"></td><td>Mano izquierda hacia adelante</td><td>Mantener mano izquierda</td><td>Mano izquierda hacia atrás</td><td colspan="3"></td></tr>
</table>

FIGURA 5.40: Secuencia de tareas para el movimiento acrobático de NAO (elaboración propia). El eje horizontal representa el tiempo en el que se realiza cada movimiento.

Como se observa en la Figura 5.40, las tareas con más importancia son las de contacto del pie con el suelo. Dicho contacto es importante debido a que de otra manera la base flotante se desplazaría de manera irreal. El contacto del pie derecho con el piso permanece durante todo el movimiento, pero el contacto del pie izquierdo solo al inicio y al final del movimiento. Puede observarse que en algunos casos hay hasta cinco tareas que suceden en simultáneo, como hacia el centro del diagrama, donde se tiene todas las tareas excepto la del contacto del pie izquierdo. Esto significa que simultáneamente se controla ambos pies, la base, y ambas manos.

5.3.2. Robot Valkyrie

El robot Valkyrie, también llamado R5, es un robot desarrollado por el *Johnson Space Center* (JSC) de la NASA (`https://www.nasa.gov/feature/r5`). Tiene una altura de 1.88 m, un peso total de 136 Kg, y posee en total 44 grados de libertad, pero si no se considera los grados de libertad de sus manos, dispone solamente de 32 grados de libertad. Este robot se muestra en la Figura 5.41 a la izquierda, y en la parte derecha se muestra la visualización de su modelo URDF en RViz de ROS.

En esta sección se mostrará algunos ejemplos de movimiento generado para el robot Valkyrie. Los casos mostrados son similares a los desarrollados para el robot NAO con el fin de mostrar que el sistema propuesto es genérico y aplicable a cualquier tipo de robot humanoide, sea grade o pequeño, y es independiente del número de grados de libertad y de la configuración del robot. El modelo URDF del robot se encuentra disponible en el repositorio de NASA Robotics llamado: `https://`

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

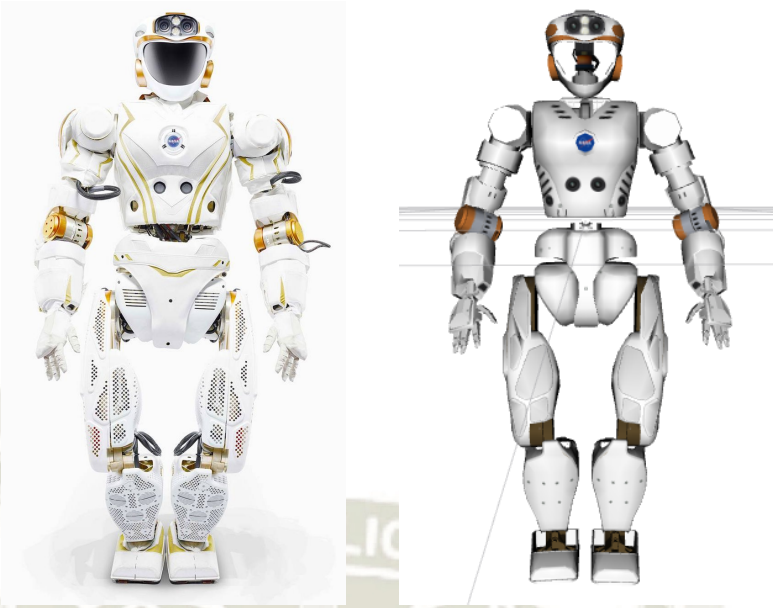


FIGURA 5.41: Robot Valkyrie de NASA-JSC: real (izquierda, tomado de NASA-JSC), en el visualizador RViz de ROS (derecha, elaboración propia).

gitlab.com/nasa-jsc-robotics/val_description. Todo lo desarrollado en esta sección utiliza dicho modelo URDF, y el código específico usado para el movimiento de esta sección se encuentra disponible en el repositorio de github: https://github.com/oeramosp/valkyrie_oscr.

Ejemplos de Movimientos. La Figura 5.42 muestra tres diferentes ejemplos de movimiento para el robot Valkyrie, en los cuales el robot mantiene en todo momento contacto con el piso a través de sus dos pies; es decir, en todo momento las tareas de mayor prioridad son aquellas que mantienen la posición y orientación de los pies fijas.

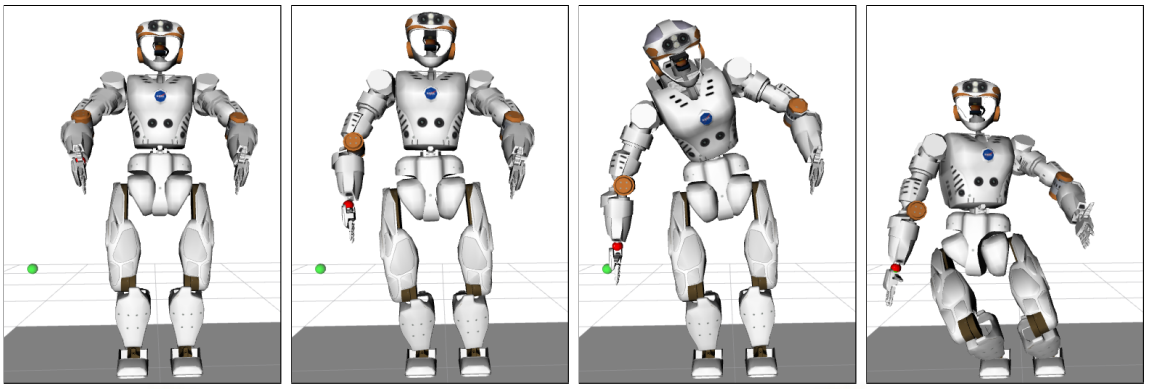
- En la Figura 5.42.a, la muñeca derecha del robot debe alcanzar la esfera verde que se encuentra más abajo del alcance máximo del brazo cuando el robot está completamente de pie. Además, se desea mantener la orientación de la cabeza (hacia adelante). Como se observa en la secuencia, para poder el robot llegar a la tarea de la mano derecha, de manera automática genera un movimiento que hace que se flexione las rodillas (como lo haría una persona) mientras mantiene la cabeza hacia adelante. Se observa que hacia el final del movimiento el torso gira un poco, pero esto es para ayudar al movimiento, y debido a que no existe ninguna restricción impuesta sobre el torso.

- En la Figura 5.42.b el robot debe alcanzar la esfera verde con la mano derecha sin mover la mano izquierda; es decir, la posición y orientación de la mano izquierda deben permanecer constantes durante todo el movimiento. En la secuencia se puede observar que esto es lo que pasa, y en este caso para que la mano derecha pueda llegar a su posición deseada, en lugar de

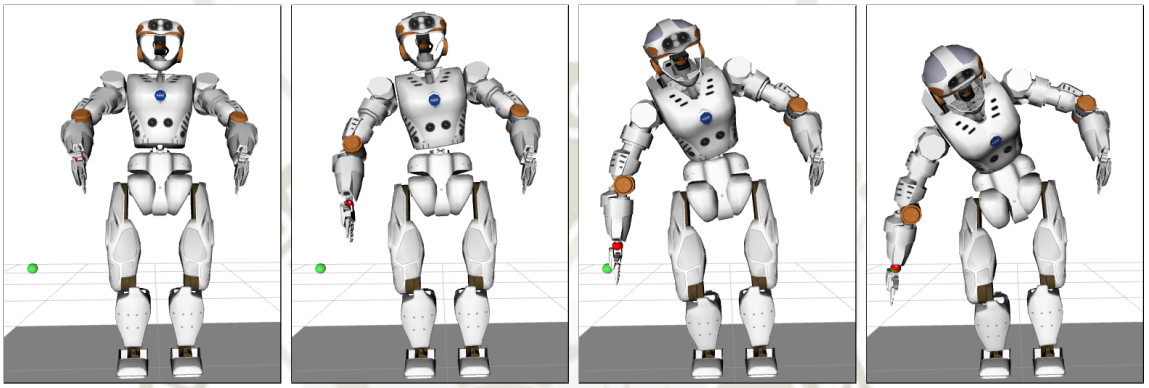
Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

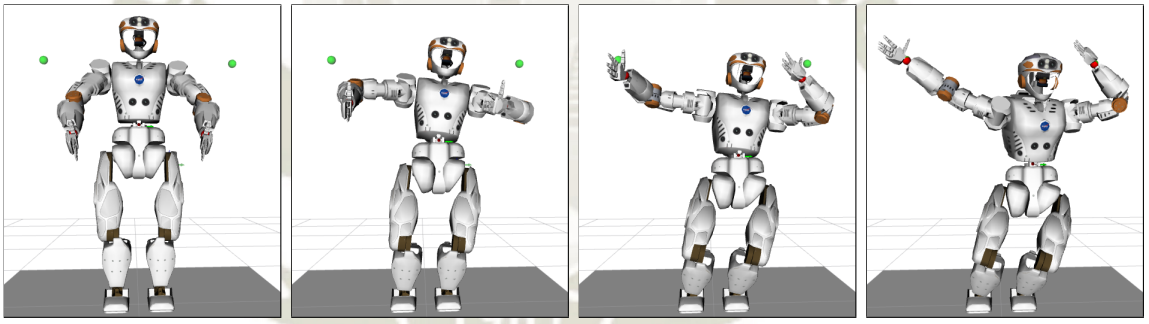
UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA



a) Movimiento de la mano derecha manteniendo la orientación de la cabeza



b) Movimiento de la mano derecha manteniendo fija la mano izquierda



c) Movimiento de la pelvis y de ambas manos

FIGURA 5.42: Movimiento del robot manteniendo fijas la posición y orientación de los pies para no perder contacto con el piso (elaboración propia).

flexionar las rodillas, el robot opta por inclinar el torso. Esto se obtiene debido a que el torso es libre de moverse ya que no hay tareas impuestas sobre él.

- En la Figura 5.42.c la pelvis del robot debe desplazarse hacia la izquierda y hacia abajo mientras ambas manos deben llegar a las posiciones deseadas, indicadas mediante esferas verdes. La secuencia muestra dicho movimiento. Existe cierto movimiento residual de la cabeza, por ejemplo, pero se debe a que no existe control sobre la misma. Igualmente, hacia el final existe cierto giro del torso para ayudar al movimiento deseado, dado que no existe ninguna tarea explícita sobre el mismo.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

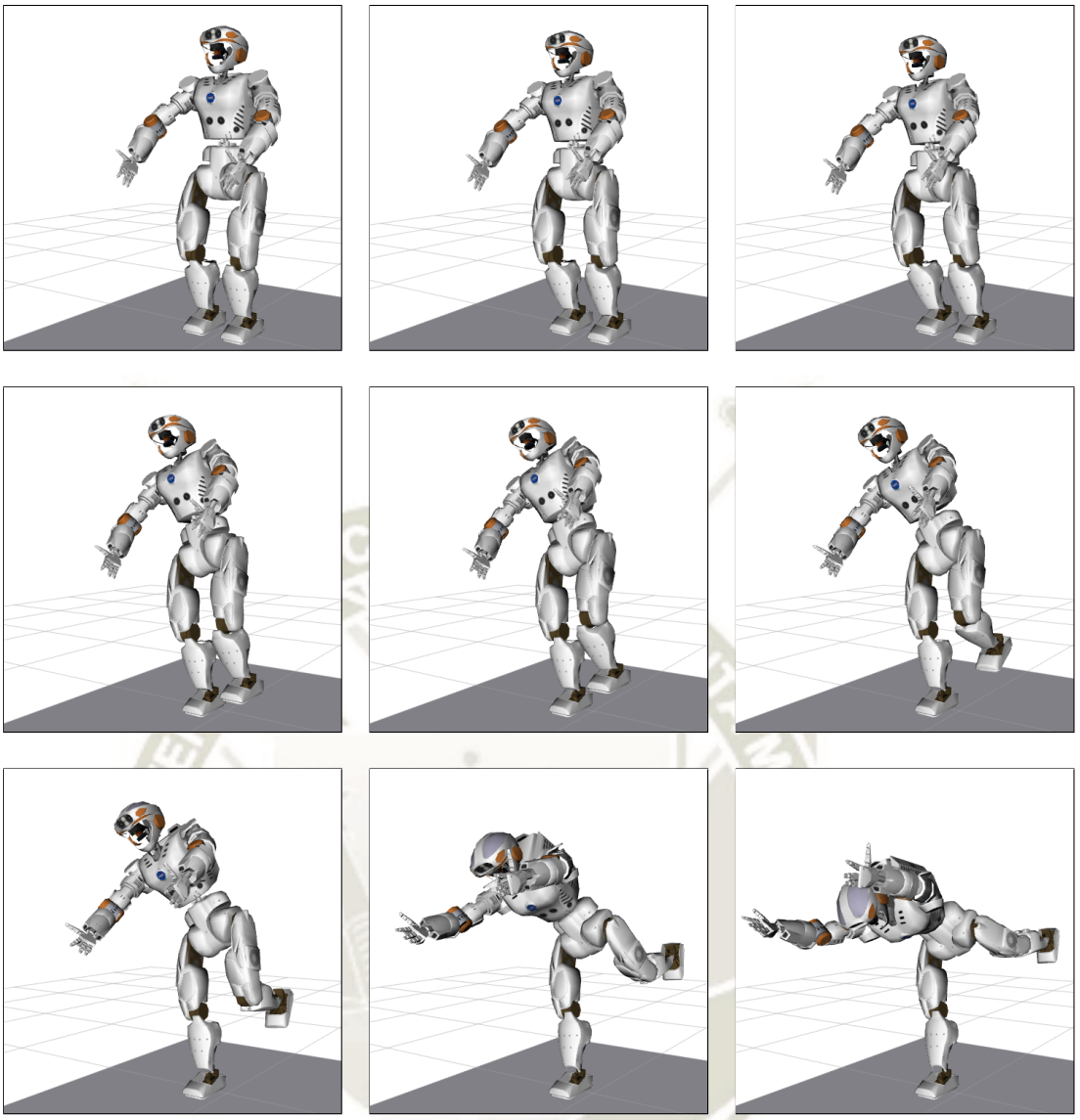


FIGURA 5.43: Movimiento acrobático del robot Valkyrie: el robot se inclina hacia la derecha (sobre su pie derecho), levanta la pierna izquierda, y luego mueve los brazos hacia adelante mientras que mueve el pie izquierdo hacia atrás (elaboración propia).

En los ejemplos anteriores se puede ver cómo el robot genera movimiento adicional de las rodillas, torso, pelvis, cabeza, etc. con el fin de alcanzar los objetivos. Es la redundancia del sistema la que permite alcanzar varios objetivos de manera simultánea, y mientras no se especifique una tarea para un punto operacional (por ejemplo, sobre el torso o la cabeza), dicho punto operacional es libre de escoger el movimiento que más convenga para la realización de las tareas controladas. No se muestra los gráficos de posición y orientación de cada parte debido a que muestran un comportamiento similar a los gráficos mostrados anteriormente. En este caso, el ciclo de control está alrededor de los 2 ms, y por tanto es factible la implementación en tiempo real en robots con una frecuencia de control de 100 o 200 Hz.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Movimiento Acrobático. Con el fin de mostrar que el sistema implementado es fácilmente aplicable a cualquier robot, se realizó un movimiento acrobático similar al desarrollado para el robot NAO (el cual fue mostrado en la Figura 5.39). El movimiento acrobático para el robot Valkyrie se muestra en la Figura 5.43, de izquierda a derecha y de arriba a abajo. Como se puede observar el movimiento es básicamente el mismo: desplazar la pelvis hacia la derecha, levantar el pie derecho, y mover el pie hacia atrás junto con las manos hacia adelante. Finalmente el movimiento se ejecuta en sentido inverso para terminar en la posición inicial, pero esto no se muestra en la figura anterior ya que el movimiento es mayormente simétrico y se observaría lo mismo pero en dirección opuesta. Toda la secuencia de tareas para este movimiento es la misma que la descrita para el robot NAO; es decir, está también dada por el diagrama mostrado en la Figura 5.40. En este caso, la única variación son los valores deseados ya que el robot Valkyrie es físicamente mucho más grande que el robot NAO. En este caso, solo a modo de verificación se mostrará la trayectoria de algunos puntos operacionales. La Figura 5.44 muestra las posiciones de la mano derecha y de la mano izquierda. Las líneas punteadas muestran los valores deseados, los cuales solamente existen en algunos instantes de tiempo. Por ejemplo, al inicio no existe ninguna restricción sobre las manos, y éstas presentan un movimiento libre. Las manos solamente son controladas explícitamente desde el tiempo $t = 10$ s hasta $t = 46$ s.

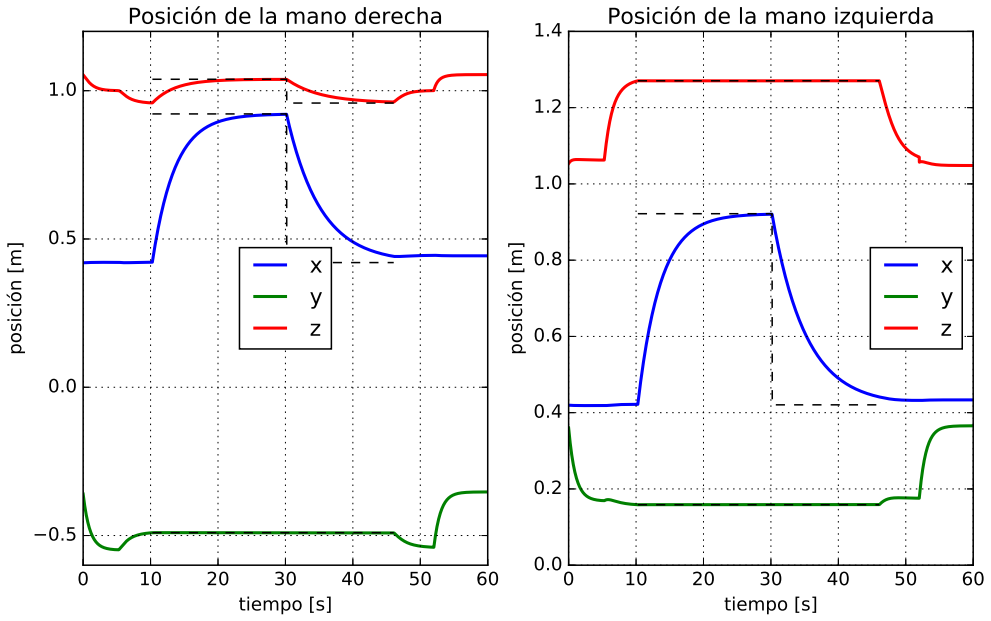


FIGURA 5.44: Evolución temporal de la posición de ambas manos en el movimiento mostrado (elaboración propia). Las referencias se muestran como líneas punteadas.

La posición de la pelvis junto con la posición del pie izquierdo pueden observarse en la Figura 5.45. Claramente la pelvis se desplaza en y hacia la derecha (-0.2 en y) hacia el inicio del movimiento, y

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

mantiene dicha posición durante la mayor parte del movimiento para solamente regresar a su valor inicial hacia el final. Por su parte, el pie izquierdo se desplaza un poco en z hacia arriba, luego de que la pelvis se ha desplazado hacia la derecha; y luego se mueve hacia atrás y mucho más arriba.

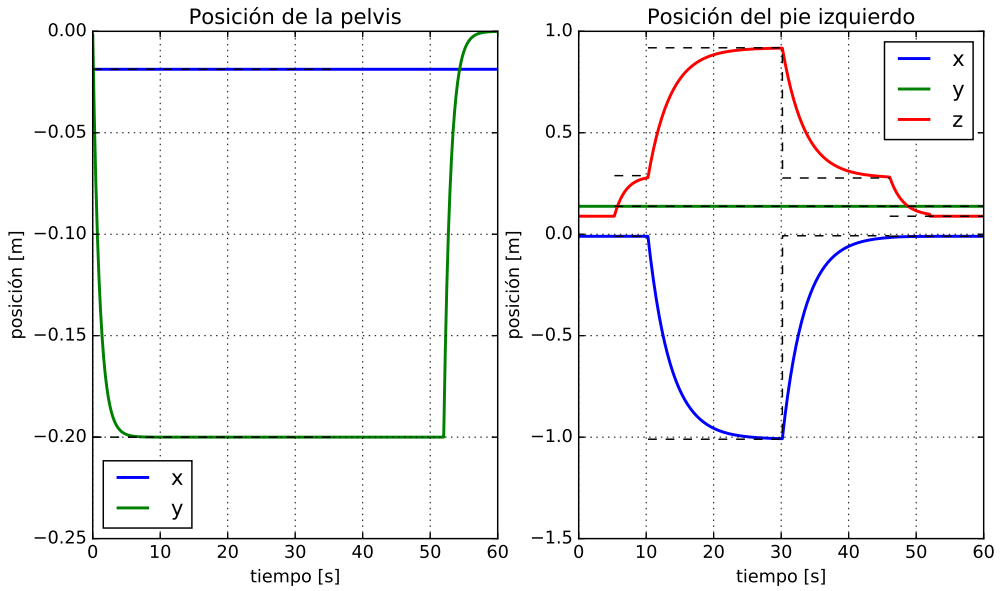


FIGURA 5.45: Evolución de la posición de la pelvis y del pie izquierdo para el movimiento mostrado (elaboración propia). La línea punteada es la referencia.

Como se puede observar, no es complicado generar un movimiento aparentemente complicado; basta con especificar la secuencia de tareas y los puntos a los cuales se desea llegar, en posición y/o orientación. El sistema genera automáticamente las trayectorias articulares que serán enviadas al robot controlado en posición. En este caso, la trayectoria de las 32 articulaciones que posee este robot se muestran en la Figura 5.46. Se muestra todas las trayectorias juntas sin una leyenda debido a la cantidad de las mismas. Se puede observar que no existen picos ni cambios bruscos en estas trayectorias, sino un movimiento suave generado de manera automática.

La obtención manual de las trayectorias mostradas sería, por decir lo menos, muy complicada y tediosa. Por otro lado, el tiempo computacional que toma resolver el programa cuadrático, ya que se utilizó el sistema de control cinemático basado en HQPs, se muestra en la Figura 5.47. Este tiempo varía entre aproximadamente 1.5 ms y 2.3 ms dependiendo de la complejidad del sistema; es decir, de cuántas tareas simultáneas se tiene. La parte intermedia muestra mayor complejidad debido a que en ese momento se realiza el control simultáneo de ambos pies, de ambas manos, y de la base, mientras que hacia el inicio y el final no se control a las manos. Esta figura muestra además que el tiempo de computación es mucho menor que 5 ms, el cual es un ciclo de control típico para este robot. Así, el sistema implementado puede ser usado en tiempo real para el robot Valkyrie. Se debe considerar, además, que las pruebas se han realizado en un sistema que no es de tiempo real,

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

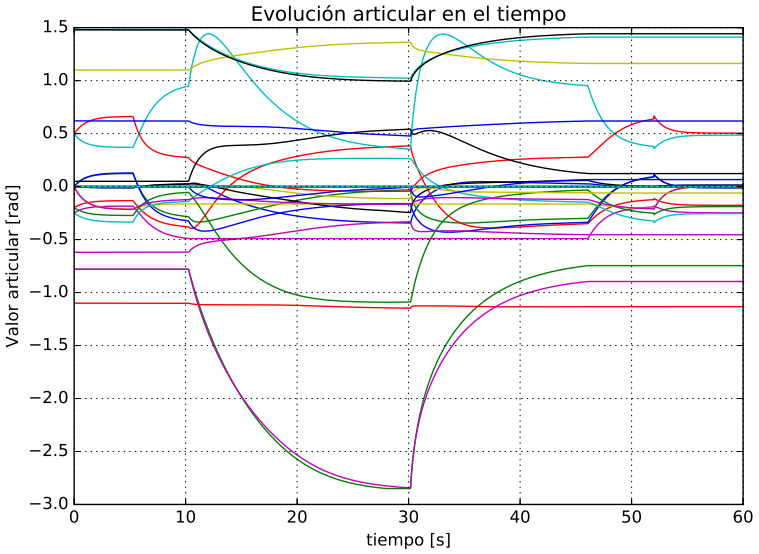


FIGURA 5.46: Evolución de la configuración articular (elaboración propia).

pero para la implementación en un robot real sería recomendable cambiar a un sistema operativo de tiempo real.

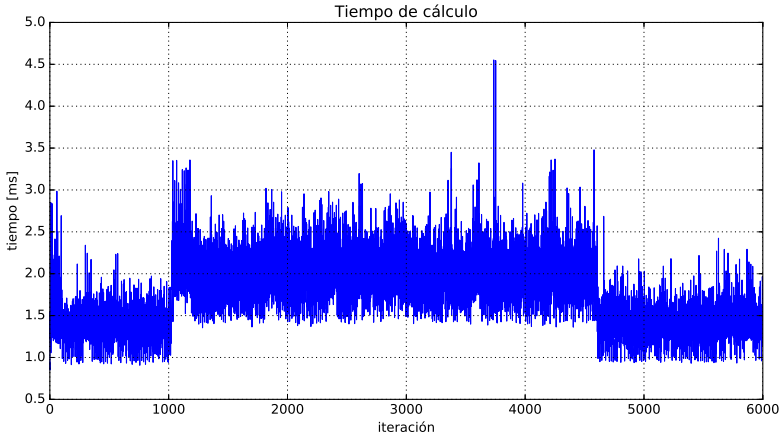


FIGURA 5.47: Tiempo de computación para el movimiento acrobático (elaboración propia). A pesar de ser un movimiento complejo, se tiene una media de 2.5 ms.

Así como se controló al robot Valkyrie y al robot NAO, es posible realizar el control cinemático de otros tipos de robots humanoides, siempre que se tenga su modelo URDF.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Capítulo 6

Conclusiones y Recomendaciones

Conclusiones

1. En esta tesis se mostró el desarrollo de un sistema de generación de movimiento cinemático para robots humanoides, el cual es aplicable también para robots manipuladores de base fija, o robots redundantes de base flotante. Se mostró los fundamentos teóricos subyacentes a esta implementación, así como el diseño completo del sistema basado en cinemática, y extendiendo el concepto tradicional de robot manipulador, a robot de base flotante. Se implementó el sistema utilizando C++ y creando una interfaz con Python por facilidad de uso. Se utilizó, además, ROS para la visualización de ejemplos de movimiento para diversos tipos de robot.

2. La especificación de objetivos de control a través de tareas cinemáticas mostró efectividad, permitiendo alcanzar adecuadamente varios objetivos de control de manera simultánea. Además, dado que esta especificación se realiza en el espacio de la tarea, que típicamente es el espacio Cartesiano, resulta intuitivo brindar el punto de referencia deseado, sea de posición o de orientación, o ambos a la vez. La única complejidad en el caso de la orientación consiste en el dominio del concepto del cuaternión.

3. El sistema de generación de movimiento propuesto fue implementado con base en el fundamento teórico para el diseño del sistema, prácticamente de cero. La base a nivel de software fue el uso de Eigen para el manejo de álgebra lineal en C++, de QPOases para la resolución de problemas de optimización cuadráticos, y el uso de dos librerías (RBDL y Pinocchio) que solamente proveen aspectos básicos de cinemática directa. Así, no se requirió de ningún otro tipo de middleware ni de ninguna estructura compleja de software. Por este motivo, el sistema desarrollado es independiente del middleware y podría ser adecuado a diversos sistemas

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

más específicos. Se mostró el uso del sistema propuesto con ROS, con fines de visualización, y debido a que ROS se ha convertido en el estándar *de facto* en robótica.

4. Mientras se disponga de un modelo de robot especificado mediante el formato URDF, es posible realizar el control cinemático del mismo haciendo uso del sistema desarrollado en esta tesis. El formato URDF es, en cierta forma, parte de ROS, y casi todos los robots de investigación poseen esta especificación. Además, la mayoría de fabricantes de robots industriales brindan actualmente las especificaciones de sus robots en este formato a través de repositorios de libre acceso.

5. Los resultados mostrados en el capítulo anterior demuestran que el sistema desarrollado puede no solamente ser aplicado a robots humanoides, sino también a robots manipuladores y otros tipos de robots de base flotante. En particular se demostró su uso con el robot KR5 de Kuka, Sawyer de Rethink Robotics, NAO de SoftBank Robotics y Valkyrie de la NASA. El sistema podría además aplicarse para otros robots humanoides sin mayor dificultad. De esta manera, se verificó que el sistema desarrollado permite mover de manera coherente y deseada diversos robots siempre que su modelo esté adecuadamente especificado utilizando el formato URDF.

6. El sistema desarrollado en esta tesis, llamado *OSCR*, se encuentra disponible como código abierto en un repositorio de github. Los programas para cada robot se encuentran también disponibles en sendos repositorios de github que hacen uso del sistema principal. Los modelos URDF usados en esta tesis son propiedad de sus respectivos fabricantes, pero se encuentran también disponibles de manera libre por los mismos. Para poder instalar este sistema, las instrucciones específicas se encuentran en el repositorio del sistema. Una ventaja importante es que todo el sistema desarrollado hace uso únicamente de herramientas de código abierto, y por tanto, para su uso no se requiere software privativo ni licencias, ya que además, tanto Python como C++ son libres.

7. Como trabajo futuro se desea desarrollar soporte para Octave, un programa OpenSource para la manipulación de matrices (muy semejante a MATLAB, con solo menores diferencias) pero que no requiere de licencia. Además se espera extender la generación de movimiento al uso de dinámica inversa especialmente para la optimización de trayectorias (control óptimo con condiciones iniciales dadas), y el uso de planificación de movimiento (*motion planning*).

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Recomendaciones

1. Para el uso del sistema de generación de movimiento cinemático desarrollado e implementado en esta tesis, se recomienda seguir los pasos indicados en el archivo `INSTALL.md` que se encuentra en el repositorio de `oscr`, el cual brinda paso a paso las instrucciones necesarias.

2. En esta tesis se utilizó bastantes conceptos de robótica. Por tanto, se recomienda la introducción de cursos de pregrado en esta área que se enfoquen en los fundamentos matemáticos de la robótica y no solamente en la programación básica de microcontroladores para aplicaciones de robots móviles. Además, sería recomendable también introducir cursos de robótica autónoma.

3. Para implementar esta tesis se hizo uso exhaustivo de programación principalmente en C++ y en Python, utilizando conceptos de clases, herencia, polimorfismo, y elementos de la librería estándar (stl) de C++. Así, se recomienda introducir con mayor profundidad cursos de programación orientada a objetos, usando stl y herramientas adicionales para el uso de matrices (como Eigen, por ejemplo).

4. Para el desarrollo de esta tesis se realizó el uso de herramientas de código abierto *open source*, las cuales tienen grandes capacidades y no requieren licencias. Por este motivo, sería recomendable introducir algunas de estas herramientas *open source* en cursos de pregrado como programación o simulación de sistemas.

5. Se recomienda exponer a los estudiantes al sistema operativo Linux, el cual es fundamental en robótica, tanto para correr programas en algunas micro computadoras comerciales (como Raspberry Pi, Jetson, etc), como para utilizar herramientas como ROS, que se han convertido en un estándar *de facto* en robótica.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Apéndice A

Cuaterniones

Los cuaterniones, introducidos por Hamilton [50], generalizan los números complejos y pueden ser utilizados para representar rotaciones en tres dimensiones al igual que los números complejos representan rotaciones planas en un círculo unitario. Así, brindan una parameterización global de $SO(3)$ al precio de usar cuatro números para representar la rotación. Un cuaternion es una tupla de 4 elementos reales $(w, \epsilon_x, \epsilon_y, \epsilon_z)$ con operaciones de adición y multiplicación que se representa como [6, 51]:

$$Q = w + \epsilon_x \boldsymbol{i} + \epsilon_y \boldsymbol{j} + \epsilon_z \boldsymbol{k} = (w, \boldsymbol{\epsilon})$$

con $\epsilon_x, \epsilon_y, \epsilon_z, w \in \mathbb{R}$. El elemento w es el componente escalar de Q y $\boldsymbol{\epsilon} = (\epsilon_x, \epsilon_y, \epsilon_z)$ es el componente vectorial. Los elementos de un cuaternión también se conocen como *parámetros de Euler* de una rotación finita [9].

El conjunto de cuaterniones $\mathbb{Q}$ es un espacio vectorial de cuatro dimensiones sobre el campo de los reales y forma un grupo con respecto a la multiplicación de cuaterniones denotada por $\circ$. La multiplicación es distributiva y asociativa pero no conmutativa, y satisface las siguientes relaciones, las cuales son comúnmente denominadas *ecuaciones fundamentales* de los cuaterniones:

$$\begin{aligned} &\boldsymbol{i} \circ \boldsymbol{i} = \boldsymbol{j} \circ \boldsymbol{j} = \boldsymbol{k} \circ \boldsymbol{k} = \boldsymbol{i} \circ \boldsymbol{j} \circ \boldsymbol{k} = -1 \\ &\boldsymbol{i} \circ \boldsymbol{j} = \boldsymbol{k}, \qquad \boldsymbol{j} \circ \boldsymbol{k} = \boldsymbol{i}, \qquad \boldsymbol{k} \circ \boldsymbol{i} = \boldsymbol{j} \\ &\boldsymbol{j} \circ \boldsymbol{i} = -\boldsymbol{k}, \quad \boldsymbol{k} \circ \boldsymbol{j} = -\boldsymbol{i}, \quad \boldsymbol{i} \circ \boldsymbol{k} = -\boldsymbol{j} \end{aligned}$$

A.1. Operaciones con cuaterniones

Para describir las operaciones con cuaterniones consideremos el cuaternión genérico $Q = (w, \boldsymbol{\epsilon})$ y los siguientes dos cuaterniones [51]: $Q_1 = (w_1, \boldsymbol{\epsilon}_1)$ y $Q_2 = (w_2, \boldsymbol{\epsilon}_2)$.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

1. *Suma.* La suma de dos cuaterniones Q_1 y Q_2 es semejante a una suma vectorial: $Q_1 + Q_2 = (w_1 + w_2,\ \boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_2)$.

2. *Producto escalar.* El producto de un escalar λ y un cuaternión Q es: $\lambda Q = (\lambda w,\ \lambda\boldsymbol{\epsilon})$.

3. *Producto.* El producto de un cuaternión Q_1 con un cuaternión Q_2 está dado por: $Q_1 \circ Q_2 = (w_1 w_2 - \boldsymbol{\epsilon}_1^T \boldsymbol{\epsilon}_2,\ w_1\boldsymbol{\epsilon}_2 + w_2\boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_1 \times \boldsymbol{\epsilon}_2)$, lo cual se puede demostrar algebraicamente realizando el producto término a término. En forma matricial, dado el cuaternión Q se puede definir la matriz $\hat{Q}$ siguiente:

$$\hat{Q} = \begin{bmatrix} w & -\epsilon_x & -\epsilon_y & -\epsilon_z \\ \epsilon_x & w & -\epsilon_z & \epsilon_y \\ \epsilon_y & \epsilon_z & w & -\epsilon_x \\ \epsilon_z & -\epsilon_y & \epsilon_x & w \end{bmatrix} = \begin{bmatrix} w & -\boldsymbol{\epsilon}^T \\ \boldsymbol{\epsilon} & wI + \hat{\boldsymbol{\epsilon}} \end{bmatrix}$$

   donde $\hat{\boldsymbol{\epsilon}}$ es la matriz antisimétrica obtenida a partir de $\boldsymbol{\epsilon}$. Así la multiplicación de cuaterniones es equivalente al producto matricial: $Q \circ Q_1 = \hat{Q}Q_1$. Notar que $\hat{Q}$ es una matriz ortogonal.

4. *Conjugada.* La conjugada de un cuaternión Q se define de manera análoga a la conjugada de un número complejo como: $Q^* = (w, -\boldsymbol{\epsilon})$.

5. *Norma.* La norma o magnitud de un cuaternión Q es un mapa $\|.\| : \mathbb{Q} \to \mathbb{R}$ y se define como: $\|Q\|^2 = Q \circ Q^* = w^2 + \epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2$. Algunas propiedades de la norma son: $\|Q_1 \circ Q_2\| = \|Q_1\|\|Q_2\|$, y $\|Q\| = \|Q^*\|$.

6. *Inversa.* La inversa de un cuaternión Q es: $Q^{-1} = \frac{Q^*}{\|Q\|^2}$, siempre que $Q \neq \mathbf{0}$. Además, se puede mostrar que $(Q_1 \circ Q_2)^{-1} = Q_2^{-1} \circ Q_1^{-1}$.

7. *Producto interno.* El producto interno entre cuaterniones $\circ : \mathbb{Q} \times \mathbb{Q} \to \mathbb{R}$ se define como: $Q_1 \cdot Q_2 = (w_1 w_2,\ \boldsymbol{\epsilon}_1^T \boldsymbol{\epsilon}_2)$. Usando esta definición se puede expresar la norma como $\|Q\|^2 = Q \cdot Q$.

Adicionalmente, el elemento identidad para la multiplicación de cuaterniones es $Q = (1, \mathbf{0})$. Usando la definición, es posible demostrar que la suma es asociativa y conmutativa, y que la multiplicación es asociativa pero no conmutativa [51].

A.2. Cuaterniones Unitarios y Rotaciones

Los cuaterniones unitarios son el subconjunto de todos los $Q \in \mathbb{Q}$ tales que su magnitud es unitaria; es decir: $\|Q\| = 1$. Una propiedad importante es que dados los cuaterniones unitarios Q, Q_1, Q_2,

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

se cumple que $\|Q_1 \circ Q_2\| = 1$ y $Q^{-1} = Q^*$. La importancia de un cuaternión unitario radica en que es equivalente a una rotación de $\theta \in \mathbb{R}$ alrededor de un eje unitario de rotación $\boldsymbol{r} \in \mathbb{R}^3$ como [11]:

$$Q = \left(\cos\frac{\theta}{2}, \boldsymbol{r}\sin\frac{\theta}{2}\right)$$

donde los componentes son $w = \cos\frac{\theta}{2}$ y $\boldsymbol{\epsilon} = \boldsymbol{r}\sin\frac{\theta}{2}$.

Composición de rotaciones. Un cálculo detallado muestra que si ${}^aQ_b$ representa una rotación entre los sistemas $\{\mathcal{A}\}$ y $\{\mathcal{B}\}$, y ${}^bQ_c$ representa una rotación entre los sistemas $\{\mathcal{B}\}$ y $\{\mathcal{C}\}$, luego la rotación entre $\{\mathcal{A}\}$ y $\{\mathcal{C}\}$ estará dada por el cuaternion ${}^aQ_c = {}^aQ_b \circ {}^bQ_c$. Así, la operación de grupo en cuaterniones unitarios corresponde directamente a la operación de grupo para las rotaciones. Además, el cuaternión que se extrae de $\boldsymbol{R}^{-1} = \boldsymbol{R}^T$ es directamente Q^{-1}.

Relación: matriz de rotación y cuaternión. Dado un cuaternión unitario $Q = (w, \boldsymbol{\epsilon})$, la matriz de rotación equivalente es $\boldsymbol{R} = e^{\hat{\boldsymbol{r}}\theta}$, donde

$$\theta = 2\cos^{-1} w, \qquad \boldsymbol{r} = \begin{cases} \frac{\boldsymbol{\epsilon}}{\sin\frac{\theta}{2}}, & \text{si } \theta \neq 0 \\ 0, & \text{si } \theta = 0 \end{cases}$$

Sin embargo resulta computacionalmente más eficiente utilizar $\theta = 2\,\text{atan2}(\|\boldsymbol{\epsilon}\|, w)$, $\boldsymbol{r} = \frac{\boldsymbol{\epsilon}}{\|\boldsymbol{\epsilon}\|}$. La ecuación para el eje de rotación no está bien condicionada cerca a $\theta = 0$, pero esto es un resultado de indeterminación fundamental del eje de rotación para una rotación nula. De manera explícita, los términos de la matriz de rotación dado un cuaternión se representan por (2.15). La relación inversa que permite obtener un cuaternión dada una matriz se obtiene usando (2.16).

Rotación de un vector mediante un cuaternión. Usando un cuaternión unitario se puede aplicar la rotación de un ángulo θ alrededor de un eje $\boldsymbol{r}$ a un vector $\boldsymbol{v} \in \mathbb{R}^3$. Para esto, primero se expresa el vector como un cuaternión con parte escalar nula $\tilde{\boldsymbol{v}} = (0, \boldsymbol{v})$, y luego la rotación de $\boldsymbol{v}$ se obtiene como $\tilde{\boldsymbol{v}}_{q_{rot}} = Q \circ \tilde{\boldsymbol{v}} \circ Q^*$, donde $\tilde{\boldsymbol{v}}_{rot} = (0, \boldsymbol{v}_{rot})$ es el cuaternión resultante que tiene parte escalar nula y la parte vectorial corresponde al vector $\boldsymbol{v}$ rotado. De manera explícita la expansión de $\tilde{\boldsymbol{v}}_{rot} = Q \circ \tilde{\boldsymbol{v}} \circ Q^*$ es

$$\begin{aligned}
\tilde{\boldsymbol{v}}_{rot} &= (w, \boldsymbol{\epsilon}) \circ (0, \boldsymbol{v}) \circ (w, -\boldsymbol{\epsilon}) \\
\tilde{\boldsymbol{v}}_{rot} &= (-\boldsymbol{\epsilon}^T\boldsymbol{v}, w\boldsymbol{v} + \boldsymbol{\epsilon} \times \boldsymbol{v}) \circ (w, -\boldsymbol{\epsilon}) \\
\tilde{\boldsymbol{v}}_{rot} &= \left(-w(\boldsymbol{\epsilon}^T\boldsymbol{v}) + (w\boldsymbol{v} + \boldsymbol{\epsilon} \times \boldsymbol{v})^T\boldsymbol{\epsilon},\ (\boldsymbol{\epsilon}^T\boldsymbol{v})\boldsymbol{\epsilon} + w(w\boldsymbol{v} + \boldsymbol{\epsilon} \times \boldsymbol{v}) - (w\boldsymbol{v} + \boldsymbol{\epsilon} \times \boldsymbol{v}) \times \boldsymbol{\epsilon}\right).
\end{aligned}$$

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

La parte escalar del cuaternión resultante se puede escribir como $-w(\boldsymbol{\epsilon}^T\boldsymbol{v})+w(\boldsymbol{v}^T\boldsymbol{\epsilon})+\boldsymbol{\epsilon}^T(\boldsymbol{\epsilon}\times\boldsymbol{v}) = 0$, que se simplifica a cero, y por tanto el cuaternión resultante no tiene parte escalar. Para la parte vectorial se tiene:

$$\begin{aligned}
\boldsymbol{v}_{rot} &= (\boldsymbol{\epsilon}\cdot\boldsymbol{v})\boldsymbol{\epsilon} + w^2\boldsymbol{v} + 2w(\boldsymbol{\epsilon}\times\boldsymbol{v}) - (\boldsymbol{\epsilon}\times\boldsymbol{v})\times\boldsymbol{\epsilon} \\
\boldsymbol{v}_{rot} &= (\boldsymbol{\epsilon}\cdot\boldsymbol{v})\boldsymbol{\epsilon} + w^2\boldsymbol{v} + 2w(\boldsymbol{\epsilon}\times\boldsymbol{v}) + (\boldsymbol{\epsilon}\cdot\boldsymbol{v})\boldsymbol{\epsilon} - \|\boldsymbol{\epsilon}\|^2\boldsymbol{v} \\
\boldsymbol{v}_{rot} &= 2(\boldsymbol{\epsilon}\cdot\boldsymbol{v})\boldsymbol{\epsilon} + (w^2 - \|\boldsymbol{\epsilon}\|^2)\boldsymbol{v} + 2w(\boldsymbol{\epsilon}\times\boldsymbol{v})
\end{aligned}$$

donde se utilizó $(\boldsymbol{a}\times\boldsymbol{b})\times\boldsymbol{c} = -(\boldsymbol{c}\cdot\boldsymbol{b})\boldsymbol{a}+(\boldsymbol{c}\cdot\boldsymbol{a})\boldsymbol{b}$, que es una propiedad del triple producto vectorial. Se puede demostrar que esta última expresión en realidad es una transformación lineal para $\boldsymbol{v}$, y a partir de la expansión de este operador de transformación se puede obtener (2.15). Reemplazando $w = \cos\frac{\theta}{2}$ y $\boldsymbol{\epsilon} = \boldsymbol{r}\sin\frac{\theta}{2}$ en la expresión anterior, y utilizando algunas identidades trigonométricas elementales, se llega a:

$$\begin{aligned}
\boldsymbol{v}_{rot} &= 2\sin^2\frac{\theta}{2}(\boldsymbol{r}\cdot\boldsymbol{v})\boldsymbol{r} + \left(\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2}\right)\boldsymbol{v} + 2\cos\frac{\theta}{2}\sin\frac{\theta}{2}(\boldsymbol{r}\times\boldsymbol{v}) \\
&= (1-\cos\theta)(\boldsymbol{r}\cdot\boldsymbol{v})\boldsymbol{r} + \cos\theta\,\boldsymbol{v} + \sin\theta(\boldsymbol{r}\times\boldsymbol{v}).
\end{aligned}$$

Esta última relación es llamada la *fórmula de Rodrigues* en forma vectorial, y representa la aplicación de una rotación a un vector. El uso de esta relación constituye una forma computacionalmente eficiente de rotación arbitraria.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Apéndice B

Modelo de un Robot de Base Flotante en URDF

El siguiente código corresponde al modelo de un robot de seis grados de libertad de base flotante (3 grados de libertad en cada extremidad) el cual es utilizado en las secciones 4.4.1, 4.4.2, 5.2. La sección 5.2 describe las características de este robot desde un punto de vista teórico. Aquí se muestra la implementación usando formato URDF.

En particular, para especificar el modelo y las características físicas del robot se utiliza *xacro* (`http://wiki.ros.org/xacro`), el cual es macro lenguaje XML que se sirve para construir URDFs de manera más sencilla ya que soporta la especificación de variables, cosa que URDF por defecto no soporta. En esta implementación, las variables corresponden a las longitudes y colores de diferentes partes del robot. Luego de crear el archivo `ff6dof.xacro`, el comando `xacro --inorder ff6dof.xacro` genera el archivo URDF equivalente.

Listado B.1. Modelo del robot usando xacro para generar un urdf (ff6dof.xacro)

```
<?xml version="1.0"?>
<robot name="ff6dof" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <!-- Definicion de colores para los eslabones -->
  <material name="Red">
    <color rgba="0.8 0.0 0.0 1.0"/>
  </material>
  <material name="Blue">
    <color rgba="0.0 0.0 0.8 1.0"/>
  </material>
  <material name="Green">
    <color rgba="0.0 0.8 0.0 1.0"/>
  </material>
  <material name="Gray">
    <color rgba="0.5 0.5 0.5 1.0"/>
  </material>

  <!-- Constantes -->
  <xacro:property name="M_PI" value="3.14159"/>
  <!-- Longitudes para el brazo -->
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

```
<xacro:property name="larm1_len" value="0.20" />
<xacro:property name="larm2_len" value="0.20" />
<xacro:property name="larm3_len" value="0.05" />
<xacro:property name="larm_width" value="0.025" />
<xacro:property name="larm_ball_width" value="0.03" />
<!-- Longitudes para la pierna -->
<xacro:property name="lleg1_len" value="0.20" />
<xacro:property name="lleg2_len" value="0.20" />
<xacro:property name="lleg3_len" value="0.05" />
<xacro:property name="lleg_width" value="0.025" />
<xacro:property name="lleg_ball_width" value="0.03" />
<!-- Longitudes para la base flotante (torso) -->
<xacro:property name="torso_xlen" value="0.05"/>
<xacro:property name="torso_ylen" value="0.2"/>
<xacro:property name="torso_zlen" value="0.2"/>

<!-- ============================================================= -->
<!--        TORAX: donde se ubican el brazo y la pierna            -->

<link name="base_link" />
<joint name="base_link_fixedjoint" type="fixed">
  <parent link="base_link"/>
  <child link="torso"/>
  <origin rpy="0 0 0" xyz="0 0 0"/>
  <axis xyz="0 0 0"/>
</joint>
<link name="torso">
  <visual>
    <origin xyz=" 0 0 0"  rpy="0 0 0"/>
    <geometry>
      <box size="${torso_xlen} ${torso_ylen} ${torso_zlen}"/>
    </geometry>
    <material name="Blue" />
  </visual>
</link>

<!-- ============================================================= -->
<!--        Brazos                                                 -->

<joint name="larm_j1" type="revolute">
  <parent link="torso"/>
  <child link="larm1"/>
  <origin xyz="0 ${torso_ylen/2} ${torso_zlen/2}" rpy="0 0 0" />
  <axis xyz="1 0 0" />
  <limit effort="300" velocity="1" lower="-1" upper="2.75"/>
</joint>
<link name="larm1" >
  <visual>
    <origin xyz="0 ${larm1_len/2} 0" rpy="${M_PI/2} 0 0" />
    <geometry>
      <cylinder radius="${larm_width}" length="${larm1_len}"/>
    </geometry>
    <material name="Red" />
  </visual>
</link>
<joint name="larm_j2" type="revolute">
  <parent link="larm1"/>
  <child link="larm2"/>
  <origin xyz="0 ${larm1_len} 0" rpy="0 0 0" />
  <axis xyz="1 0 0" />
  <limit effort="300" velocity="1" lower="-2.5" upper="2.5"/>
</joint>
<link name="larm2" >
  <visual>
    <origin xyz="0 ${larm2_len/2} 0" rpy="${M_PI/2} 0 0" />
    <geometry>
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

```
      <cylinder radius="${larm_width}" length="${larm2_len}"/>
    </geometry>
    <material name="Blue" />
  </visual>
</link>
<joint name="larm_j3" type="revolute">
  <parent link="larm2"/>
  <child link="larm3"/>
  <origin xyz="0 ${larm2_len} 0" rpy="0 0 0" />
  <axis xyz="1 0 0" />
  <limit effort="300" velocity="1" lower="-2.5" upper="2.5"/>
</joint>
<link name="larm3" >
  <visual>
    <origin xyz="0 ${larm3_len/2} 0" rpy="${M_PI/2} 0 0" />
    <geometry>
      <cylinder radius="${larm_width}" length="${larm3_len}"/>
    </geometry>
    <material name="Green" />
  </visual>
</link>
<!-- Esferas en las articulaciones (para fines de visualizacion) -->
<joint name="larm1_ball_joint" type="fixed">
  <parent link="larm1"/>
  <child link="larm1_ball"/>
  <origin rpy="0 0 0" xyz="0 0 0"/>
  <axis xyz="0 0 0"/>
</joint>
<link name="larm1_ball">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <sphere radius="${larm_ball_width}"/>
    </geometry>
    <material name="Gray" />
  </visual>
</link>
<joint name="larm2_ball_joint" type="fixed">
  <parent link="larm2"/>
  <child link="larm2_ball"/>
  <origin rpy="0 0 0" xyz="0 0 0"/>
  <axis xyz="0 0 0"/>
</joint>
<link name="larm2_ball">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <sphere radius="${larm_ball_width}"/>
    </geometry>
    <material name="Gray" />
  </visual>
</link>
<joint name="larm3_ball_joint" type="fixed">
  <parent link="larm3"/>
  <child link="larm3_ball"/>
  <origin rpy="0 0 0" xyz="0 0 0"/>
  <axis xyz="0 0 0"/>
</joint>
<link name="larm3_ball">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <sphere radius="${larm_ball_width}"/>
    </geometry>
    <material name="Gray" />
  </visual>
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

```
</link>

<!-- ============================================================= -->
<!--        Pierna                                                 -->

<joint name="lleg_j1" type="revolute">
  <parent link="torso"/>
  <child link="lleg1"/>
  <origin xyz="0 ${torso_ylen/2} -${torso_zlen/2}" rpy="0 0 0" />
  <axis xyz="1 0 0" />
  <limit effort="300" velocity="1" lower="-2.75" upper="1"/>
</joint>
<link name="lleg1" >
  <visual>
    <origin xyz="0 ${lleg1_len/2} 0" rpy="${M_PI/2} 0 0" />
    <geometry>
      <cylinder radius="${lleg_width}" length="${lleg1_len}"/>
    </geometry>
    <material name="Red" />
  </visual>
</link>
<joint name="lleg_j2" type="revolute">
  <parent link="lleg1"/>
  <child link="lleg2"/>
  <origin xyz="0 ${lleg1_len} 0" rpy="0 0 0" />
  <axis xyz="1 0 0" />
  <limit effort="300" velocity="1" lower="-2.5" upper="2.5"/>
</joint>
<link name="lleg2" >
  <visual>
    <origin xyz="0 ${lleg2_len/2} 0" rpy="${M_PI/2} 0 0" />
    <geometry>
      <cylinder radius="${lleg_width}" length="${lleg2_len}"/>
    </geometry>
    <material name="Blue" />
  </visual>
</link>
<joint name="lleg_j3" type="revolute">
  <parent link="lleg2"/>
  <child link="lleg3"/>
  <origin xyz="0 ${lleg2_len} 0" rpy="0 0 0" />
  <axis xyz="1 0 0" />
  <limit effort="300" velocity="1" lower="-2.5" upper="2.5"/>
</joint>
<link name="lleg3" >
  <visual>
    <origin xyz="0 ${lleg3_len/2} 0" rpy="${M_PI/2} 0 0" />
    <geometry>
      <cylinder radius="${lleg_width}" length="${lleg3_len}"/>
    </geometry>
    <material name="Green" />
  </visual>
</link>
<!-- Esferas en las articulaciones (solo para visualizacion) -->
<joint name="lleg1_ball_joint" type="fixed">
  <parent link="lleg1"/>
  <child link="lleg1_ball"/>
  <origin rpy="0 0 0" xyz="0 0 0"/>
  <axis xyz="0 0 0"/>
</joint>
<link name="lleg1_ball">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <sphere radius="${lleg_ball_width}"/>
    </geometry>
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

```
      <material name="Gray" />
    </visual>
  </link>
  <joint name="lleg2_ball_joint" type="fixed">
    <parent link="lleg2"/>
    <child link="lleg2_ball"/>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <axis xyz="0 0 0"/>
  </joint>
  <link name="lleg2_ball">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <sphere radius="${lleg_ball_width}"/>
      </geometry>
      <material name="Gray" />
    </visual>
  </link>
  <joint name="lleg3_ball_joint" type="fixed">
    <parent link="lleg3"/>
    <child link="lleg3_ball"/>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <axis xyz="0 0 0"/>
  </joint>
  <link name="lleg3_ball">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <sphere radius="${lleg_ball_width}"/>
      </geometry>
      <material name="Gray" />
    </visual>
  </link>

</robot>
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Apéndice C

Extractos de Código

Este apéndice muestra algunos extractos del código desarrollado para la implementación del sistema de generación de movimiento. No se muestran los comentarios para no hacer más largo el apéndice de lo necesario. Sin embargo, el código completo, incluyendo los comentarios de donde se extrae automáticamente la documentación, puede encontrarse en el repositorio de github: https://github.com/oeramosp/oscr.

C.1. Modelo del Robot

Listado C.1. declaración de la clase base RobotModel (robot-model.hpp)

```
#ifndef ROBOT_MODEL_HPP
#define ROBOT_MODEL_HPP

#include <string>
#include <vector>
#include <map>
#include <Eigen/Dense>

namespace oscr {

class RobotModel
{
public:
  RobotModel(const bool& has_floating_base = true);
  ~RobotModel();
  bool hasFloatingBase() const;
  std::vector<std::string> jointNames() const;
  Eigen::VectorXd jointMaxAngularLimits() const;
  Eigen::VectorXd jointMinAngularLimits() const;
  Eigen::VectorXd jointVelocityLimits() const;
  std::map<std::string, unsigned int> mapLinkNamesIDs() const;
  Eigen::VectorXd getJointConfig();
  virtual bool loadURDF(const std::string& model_name,
                        const bool& has_floating_base = true,
                        const bool& verbose = false) = 0;
  virtual unsigned int ndof() const = 0;
  virtual unsigned int ndofActuated() const = 0;
  virtual std::string floatingLink() const = 0;
  virtual void updateJointConfig(const Eigen::VectorXd& q) = 0;
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

```
  virtual Eigen::Vector3d linkPosition(const unsigned int& link_number,
                                       const Eigen::Vector3d& local_pos =
                                            Eigen::Vector3d::Zero()) const = 0;
  virtual Eigen::Vector3d linkPosition(const std::string& link_name,
                                       const Eigen::Vector3d& local_pos =
                                       Eigen::Vector3d::Zero()) const = 0;
  virtual Eigen::Vector4d linkOrientation(const unsigned int& link_number) const = 0;
  virtual Eigen::Vector4d linkOrientation(const std::string& link_name) const = 0;
  virtual Eigen::VectorXd linkPose(const unsigned int& link_number,
                                   const Eigen::Vector3d& local_pos =
                                         Eigen::Vector3d::Zero()) const = 0;
  virtual Eigen::VectorXd linkPose(const std::string& link_name,
                                   const Eigen::Vector3d& local_pos =
                                         Eigen::Vector3d::Zero()) const = 0;
  virtual Eigen::MatrixXd linearJacobian(const unsigned int& link_number,
                                         const Eigen::Vector3d& local_pos =
                                               Eigen::Vector3d::Zero()) const = 0;
  virtual Eigen::MatrixXd angularJacobian(const unsigned int& link_number) const = 0;
  virtual Eigen::MatrixXd geometricJacobian(const unsigned int& link_number,
                                            const Eigen::Vector3d& local_pos =
                                                  Eigen::Vector3d::Zero()) const = 0;
  Eigen::VectorXd setFeetOnGround(const std::string& foot1,
                                  const std::string& foot2,
                                  const double& sole_dist);
  virtual Eigen::VectorXd comPosition() = 0;
  virtual Eigen::MatrixXd comJacobian() const = 0;

protected:
  bool has_floating_base_;
  std::vector<std::string> jnames_;
  Eigen::VectorXd qmin_;
  Eigen::VectorXd qmax_;
  Eigen::VectorXd dqmax_;
  Eigen::VectorXd q_;
  std::map< std::string, unsigned int > link_id_;
  std::string floatingLink(const unsigned int& flink_id) const;
};

}

#endif
```

C.2. Tareas Cinemáticas

Listado C.2. Declaración de la clase base KineTask (kine-task.hpp)

```
#ifndef KINEMATIC_TASK_HPP
#define KINEMATIC_TASK_HPP
#include <Eigen/Dense>
#include <string>
#include <oscr/model/robot-model.hpp>

namespace oscr{
class KineTask
{
public:
  KineTask(RobotModel* rmodel, const std::string& taskName = "KineTask",
           const double& gain = 1.0);
  ~KineTask();
  void setName(const std::string& taskName);
  std::string getName() const;
  void setGain(const double& gain);
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

```
  void setAdaptiveGain(const double& max_gain, const double& min_gain, const double& rate_descent);
  double getGain(const double& error=0.0);
  void setDesiredValue(const Eigen::VectorXd& desiredValue);
  Eigen::VectorXd getDesiredValue() const;
  void setWeight(const double& weight);
  double getWeight() const;
  unsigned int getTaskDim() const;
  Eigen::VectorXd getError() const;
  void updateJointConfig(const Eigen::VectorXd& q);
  void keep(const double& gain);
  void getTaskBounds(Eigen::VectorXd& bound_min, Eigen::VectorXd& bound_max);
  virtual void getSensedValue(Eigen::VectorXd& xsensed) const = 0;
  virtual void getTaskJacobian(Eigen::MatrixXd& Jacobian) const = 0;
  virtual void getDerivError(Eigen::VectorXd& de) = 0;

protected:
  RobotModel* rmodel_;
  std::string name_;
  unsigned int ndof_;
  Eigen::VectorXd xdes_, esum_, error_, q_;
  bool adaptiveGain_;
  double lmax_, lmin_, k_, gain_, weight_;
};
}
#endif
```

C.3. Control Cinemático

Listado C.3. Declaración de la clase base OSIKSolver (osik-solver.hpp)

```
#ifndef OSIK_SOLVER_HPP
#define OSIK_SOLVER_HPP
#include <oscr/ik/kine-task.hpp>

namespace oscr{
class OSIKSolver
{
public:
  OSIKSolver(RobotModel* rmodel,
             const Eigen::VectorXd& qinit,
             const double& dt);
  ~OSIKSolver();
  void pushTask(KineTask* task);
  void popTask();
  void removeTask(const std::string& taskName);
  void getTaskStack(std::vector< KineTask* >& stack_tasks);
  virtual void getPositionControl(const Eigen::VectorXd& q,
                                  Eigen::VectorXd& qdes) = 0;
  unsigned int getTime();
  void printStack();

protected:
  RobotModel* rmodel_;
  double dt_;
  Eigen::VectorXd qdes_;
  std::vector< KineTask* > taskStack_;
  unsigned int ndof_;
  unsigned int time_;
};
}
#endif
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

Listado C.4. Declaración de la función de control (osik-solver-wqp.cpp)

```
void oscr::OSIKSolverWQP::getPositionControl(const Eigen::VectorXd& q, Eigen::VectorXd& qcmd)
{
  if (taskStack_.size()==0){
    std::cerr << "Cannot get control without tasks" << std::endl;
    qcmd = q;
    return;
  }
  if ( (qmin_.size()==0) || (qmax_.size()==0) ){
    std::cerr << "Joint limits are not set! ... keeping previous configuration" << std::endl;
    qcmd = q;
    return;
  }
  Eigen::VectorXd dq, de, p, eint;
  Eigen::MatrixXd J;
  Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic,Eigen::RowMajor> W;
  double weight;
  W = lambda_q_*Eigen::MatrixXd::Identity(q.size(), q.size());
  p = Eigen::VectorXd::Zero(q.size());
  for (unsigned int i=0; i<taskStack_.size(); ++i) {
    taskStack_[i]->updateJointConfig(q);
    taskStack_[i]->getTaskJacobian(J);
    taskStack_[i]->getDerivError(de);
    weight = taskStack_[i]->getWeight();
    W = W + weight*J.transpose()*J;
    p = p - 2*weight*J.transpose()*de;
  }
  unsigned int nV = rmodel_->ndof();
  double *H = W.data();
  double *g = p.data();
  double *lb = new double[nV];
  double *ub = new double[nV];
  double ltmp, utmp;
  for (unsigned int i=0; i<nV; ++i)  {
    ltmp = (1.0/dt_)*(qmin_[i]-q[i]);
    utmp = (1.0/dt_)*(qmax_[i]-q[i]);
    lb[i] = ltmp > -dqmax_[i] ? ltmp : -dqmax_[i];
    ub[i] = utmp < dqmax_[i] ? utmp : dqmax_[i];
  }
  qpOASES::QProblemB qp(nV);
  qp.setOptions(qpOptions_);
  int nWSR = 10;  // Max number of working set recalculations
  if (qp.init(H, g, lb, ub, nWSR, 0)) {
    std::cout << "OSIKSolverWQP: QP Problem ... returning previous values" << std::endl;
    qcmd = q;
    return;
  }
  double dqArray[nV];
  qp.getPrimalSolution(dqArray);
  dq.resize(nV);
  for (unsigned int i=0; i<nV; ++i)
    dq[i] = dqArray[i];
  if (rmodel_->hasFloatingBase())
  {
    qdes_.tail(rmodel_->ndofActuated()) =
      qdes_.tail(rmodel_->ndofActuated()) + dt_*dq.tail(rmodel_->ndofActuated());
    qdes_.head(3) = qdes_.head(3) + dt_*dq.head(3);
    Eigen::Vector3d omega;
    Eigen::MatrixXd T; T.resize(3,4);
    T << -2.0*qdes_(4),  2.0*qdes_(3), -2.0*qdes_(6),  2.0*qdes_(5),
         -2.0*qdes_(5),  2.0*qdes_(6),  2.0*qdes_(3), -2.0*qdes_(4),
         -2.0*qdes_(6), -2.0*qdes_(5),  2.0*qdes_(4),  2.0*qdes_(3);
    omega = T*dq.segment(3,4);
    double angle = dt_*omega.norm()/2.0;
    Eigen::Vector3d k; k.setZero();
    if (omega.norm() > 1.0e-8)
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

```
      k = omega/omega.norm();
    else if (fabs(angle) > 1.0e-8)
      std::cout << "WARNING: axis for floating base orientation increment is zero" << std::endl;
    else {}
    Eigen::Vector4d dQ, Qf;
    dQ << cos(angle), k(0)*sin(angle), k(1)*sin(angle), k(2)*sin(angle);
    Qf = quaternionMult(dQ, qdes_.segment(3,4));
    qdes_(3) = Qf(0); qdes_(4) = Qf(1); qdes_(5) = Qf(2); qdes_(6) = Qf(3);
  }
  else {
    qdes_ = qdes_ + dt_*dq;
  }
  qcmd = qdes_;
  time_ ++;
}
```

Listado C.5. Declaración de la función de control (osik-solver-hqp.cpp)

```
void oscr::OSIKSolverHQP::getPositionControl(const Eigen::VectorXd& q, Eigen::VectorXd& qcmd)
{
  if (taskStack_.size()==0)
  {
    qcmd = q;
    return;
  }

  unsigned int m, nV, nC;
  unsigned int n = rmodel_->ndof();
  unsigned int mprev = 0;
  double *dqArray;

  Eigen::MatrixXd Jprev, J;
  Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic,Eigen::RowMajor> Hm;
  Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic,Eigen::RowMajor> Am;
  Eigen::VectorXd dq, wopt, bminprev, bmaxprev, demin, demax;
  Eigen::VectorXd lbm, ubm, gm, bminm, bmaxm;
  Jprev.resize(0,n);
  bminprev.resize(0);
  bmaxprev.resize(0);

  for (unsigned int i=0; i<taskStack_.size(); ++i)
  {
    taskStack_[i]->updateJointConfig(q);
    taskStack_[i]->getTaskJacobian(J);
    taskStack_[i]->getTaskBounds(demin, demax);
    m  = demin.size();
    nV = m + n;
    nC = m + mprev;
    qpOASES::QProblem qp(nV, nC);
    qp.setOptions(qpOptions_);
    int nWSR = 10;
    Hm.setZero(nV, nV);
    Hm.topLeftCorner(m, m) = Eigen::MatrixXd::Identity(m, m);
    gm.setZero(nV);
    double *g = gm.data();
    double *H = Hm.data();

    Am.setZero(nC, nV);
    Am.topLeftCorner(m, m) = Eigen::MatrixXd::Identity(m, m);
    Am.topRightCorner(m, n) = J;
    Am.bottomRightCorner(mprev, n) = Jprev;

    bminm.resize(m+mprev);
    bminm.head(m) = demin;
    bminm.tail(mprev) = bminprev;
    bmaxm.resize(m+mprev);
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

```
    bmaxm.head(m) = demax;
    bmaxm.tail(mprev) = bmaxprev;

    double *A = Am.data();
    double *bmin = bminm.data();
    double *bmax = bmaxm.data();
    lbm.setConstant(nV, -INFTY);
    ubm.setConstant(nV, INFTY);
    double *lb = lbm.data();
    double *ub = ubm.data();
    if (has_joint_limits_)
    {
      double ltmp, utmp;
      for (unsigned int i=0; i<ndof_; ++i)
      {
        ltmp = (1.0/dt_)*(qmin_[i]-q[i]);
        utmp = (1.0/dt_)*(qmax_[i]-q[i]);
        lb[i+m] = ltmp > -dqmax_[i] ? ltmp : -dqmax_[i];
        ub[i+m] = utmp < dqmax_[i] ? utmp : dqmax_[i];
      }
    }

    if (qp.init(H, g, A, lb, ub, bmin, bmax, nWSR))
    {
      qcmd = q;
      return;
    }
    dqArray = new double[nV];
    qp.getPrimalSolution(dqArray);
    wopt.resize(m);
    for (unsigned int i=0; i<m; ++i)
      wopt[i] = dqArray[i];
    Jprev.conservativeResize(mprev+m, n);
    Jprev.bottomRows(m) = J;
    bminprev.conservativeResize(mprev+m);
    bmaxprev.conservativeResize(mprev+m);
    bminprev.tail(m) = demin-wopt;
    bmaxprev.tail(m) = demax-wopt;
    mprev = mprev + m;
  }

  dq.resize(n);
  for (unsigned int i=0; i<n; ++i)
    dq[i] = dqArray[i+m];
  if (rmodel_->hasFloatingBase())
  {
    qdes_.tail(rmodel_->ndofActuated()) =
      qdes_.tail(rmodel_->ndofActuated()) + dt_*dq.tail(rmodel_->ndofActuated());
    qdes_.head(3) = qdes_.head(3) + dt_*dq.head(3);
    Eigen::Vector3d omega;
    Eigen::MatrixXd T; T.resize(3,4);
    T <<
      -2.0*qdes_(4),  2.0*qdes_(3), -2.0*qdes_(6),  2.0*qdes_(5),
      -2.0*qdes_(5),  2.0*qdes_(6),  2.0*qdes_(3), -2.0*qdes_(4),
      -2.0*qdes_(6), -2.0*qdes_(5),  2.0*qdes_(4),  2.0*qdes_(3);
    omega = T*dq.segment(3,4);
    double angle = dt_*omega.norm()/2.0;
    Eigen::Vector3d k; k.setZero();
    if (omega.norm() > 1.0e-8)
      k = omega/omega.norm();
    else if (fabs(angle) > 1.0e-8)
      std::cout << "WARNING: axis for floating base orientation increment is zero"
                << std::endl;
    else {}
    Eigen::Vector4d dQ, Qf;
    dQ << cos(angle), k(0)*sin(angle), k(1)*sin(angle), k(2)*sin(angle);
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

```
    Qf = quaternionMult(dQ, qdes_.segment(3,4));
    qdes_(3) = Qf(0);
    qdes_(4) = Qf(1);
    qdes_(5) = Qf(2);
    qdes_(6) = Qf(3);
  }
  else
  {
    qdes_ = qdes_ + dt_*dq;
  }
  qcmd = qdes_;
  time_ ++;
}
```

C.4. Funciones para Posición y Orientación

Listado C.6. Declaraciones de las funciones para posición y orientación (math-utils.hpp)

```
#ifndef OSCR_MATH_UTILS_HPP
#define OSCR_MATH_UTILS_HPP
#include <Eigen/Dense>

namespace oscr{

void pinv(const Eigen::MatrixXd& input_mat,
          Eigen::MatrixXd& pseudo_inv_mat,
          const double& pinvtoler = 1.0e-6);
Eigen::Matrix3d skew(const Eigen::Vector3d& w);
Eigen::Matrix3d RPYToRotation(const Eigen::Vector3d& rpy);
Eigen::Vector3d rotationToRPY(const Eigen::Matrix3d& R);
Eigen::Vector3d rotationToAxisAngle(const Eigen::Matrix3d& R);
Eigen::Matrix3d axisAngleToRotation(const Eigen::Vector3d& w);
Eigen::Matrix3d rotationMatrix(const char& axis,
                               const double& angle);
Eigen::Matrix3d rotationMatrix(const Eigen::Vector3d& axis,
                               const double& angle);
Eigen::Matrix3d quaternionToRotation(const Eigen::Vector4d& quat);
Eigen::Vector4d rotationToQuaternion(const Eigen::Matrix3d& R);
Eigen::Vector4d quaternionMult(const Eigen::Vector4d& q1,
                               const Eigen::Vector4d& q2);
double sgn(const double& x);
const double INFTY = 1.0e20;

}
#endif
```

Listado C.7. Implementación de funciones para posición y orientación (math-utils.cpp)

```
#include <oscr/tools/math-utils.hpp>
#include <iostream>
#include <vector>

void oscr::pinv(const Eigen::MatrixXd& matrix_in, Eigen::MatrixXd& pseudo_inv,
                const double& pinvtoler)
{
  Eigen::JacobiSVD<Eigen::MatrixXd> svd(matrix_in, Eigen::ComputeThinU | Eigen::ComputeThinV);
  Eigen::VectorXd singular_values;
  Eigen::VectorXd singular_values_inv;
  singular_values = svd.singularValues();
  singular_values_inv.setZero(singular_values.size());
  for (int w = 0; w < singular_values.size(); ++w)
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM
UNIVERSIDAD CATÓLICA DE SANTA MARÍA

```
    if( singular_values(w) > pinvtoler)
      singular_values_inv(w) = 1/singular_values(w);
  pseudo_inv = svd.matrixV() * singular_values_inv.asDiagonal() * svd.matrixU().transpose();
  return;
}

Eigen::Matrix3d oscr::skew(const Eigen::Vector3d& w)
{
  Eigen::Matrix3d R;
  R(0,0) = 0.0;   R(0,1) = -w(2); R(0,2) = w(1);
  R(1,0) = w(2);  R(1,1) = 0.0;   R(1,2) = -w(0);
  R(2,0) = -w(1); R(2,1) = w(0);  R(2,2) = 0.0;
  return R;
}

Eigen::Matrix3d oscr::RPYToRotation(const Eigen::Vector3d& rpy)
{
  Eigen::Matrix3d res;
  return rotationMatrix('z',rpy(2))*rotationMatrix('y',rpy(1))*rotationMatrix('x',rpy(0));
}

Eigen::Vector3d oscr::rotationToRPY(const Eigen::Matrix3d& R)
{
  Eigen::Vector3d rpy;
  double m = sqrt(R(2,1)*R(2,1)+R(2,2)*R(2,2));
  rpy(1) = atan2(-R(2,0),m);
  if (fabs(rpy(1)-M_PI/2.0) < 0.001) {
    rpy(2) = 0;
    rpy(0) = atan2(R(0,1), R(1,1));
  }
  else if (fabs(rpy(1)+M_PI/2.0) < 0.001) {
    rpy(2) = 0;
    rpy(0) = -atan2(R(0,1), R(1,1));
  }
  else {
    rpy(2) = atan2(R(1,0), R(0,0));
    rpy(0) = atan2(R(2,1), R(2,2));
  }
  return rpy;
}

Eigen::Matrix3d oscr::quaternionToRotation(const Eigen::Vector4d& q)
{
  double normq = q.norm();
  if (fabs(normq-1.0)>0.001), return Eigen::Matrix3d::Identity();
  Eigen::Matrix3d res;
  res(0,0) = 2.0*(q(0)*q(0)+q(1)*q(1))-1.0;
  res(0,1) = 2.0*(q(1)*q(2)-q(0)*q(3));
  res(0,2) = 2.0*(q(1)*q(3)+q(0)*q(2));
  res(1,0) = 2.0*(q(1)*q(2)+q(0)*q(3));
  res(1,1) = 2.0*(q(0)*q(0)+q(2)*q(2))-1.0;
  res(1,2) = 2.0*(q(2)*q(3)-q(0)*q(1));
  res(2,0) = 2.0*(q(1)*q(3)-q(0)*q(2));
  res(2,1) = 2.0*(q(2)*q(3)+q(0)*q(1));
  res(2,2) = 2.0*(q(0)*q(0)+q(3)*q(3))-1.0;
  return res;
}

Eigen::Vector4d oscr::rotationToQuaternion(const Eigen::Matrix3d& R)
{
  double dEpsilon = 1e-6; Eigen::Vector4d quat;
  quat(0) = 0.5*sqrt(R(0,0)+R(1,1)+R(2,2)+1.0);
  if ( fabs(R(0,0)-R(1,1)-R(2,2)+1.0) < dEpsilon ), quat(1) = 0.0;
  else, quat(1) = 0.5*oscr::sgn(R(2,1)-R(1,2))*sqrt(R(0,0)-R(1,1)-R(2,2)+1.0);
  if ( fabs(R(1,1)-R(2,2)-R(0,0)+1.0) < dEpsilon ), quat(2) = 0.0;
  else, quat(2) = 0.5*oscr::sgn(R(0,2)-R(2,0))*sqrt(R(1,1)-R(2,2)-R(0,0)+1.0);
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

```
  if ( fabs(R(2,2)-R(0,0)-R(1,1)+1.0) < dEpsilon ), quat(3) = 0.0;
  else, quat(3) = 0.5*oscr::sgn(R(1,0)-R(0,1))*sqrt(R(2,2)-R(0,0)-R(1,1)+1.0);
  return quat;
}

Eigen::Vector4d oscr::quaternionMult(const Eigen::Vector4d& q1, const Eigen::Vector4d& q2)
{
  Eigen::Vector4d res;
  res(0) = -q1(1)*q2(1) - q1(2)*q2(2) - q1(3)*q2(3) + q1(0)*q2(0);
  res(1) =  q1(0)*q2(1) - q1(3)*q2(2) + q1(2)*q2(3) + q1(1)*q2(0);
  res(2) =  q1(3)*q2(1) + q1(0)*q2(2) - q1(1)*q2(3) + q1(2)*q2(0);
  res(3) = -q1(2)*q2(1) + q1(1)*q2(2) + q1(0)*q2(3) + q1(3)*q2(0);
  return res;
}

Eigen::Vector3d oscr::rotationToAxisAngle(const Eigen::Matrix3d& R)
{
  Eigen::Vector3d res;
  double sinTh = 0.5*sqrt((R(1,0)-R(0,1))*(R(1,0)-R(0,1)) + (R(2,0)-R(0,2))*(R(2,0)-R(0,2))
                          + (R(2,1)-R(1,2))*(R(2,1)-R(1,2)) );
  double cosTh = 0.5*(R(0,0)+R(1,1)+R(2,2)-1.0);
  double theta = atan2(sinTh,cosTh);   // angle btw 0 and PI (s>0)
  const double ANGLE_MIN = 0.0001;
  const double SINC_MIN  = 1e-8;
  const double COSC_MIN  = 2.5e-4;
  if ( (sinTh>ANGLE_MIN) || (cosTh>0.0) ) {
    double sinc = (fabs(sinTh) > SINC_MIN) ? (sinTh/theta) : 1.0;
    sinc = 1.0/(2.0*sinc);
    res(0) = sinc*(R(2,1)-R(1,2));
    res(1) = sinc*(R(0,2)-R(2,0));
    res(2) = sinc*(R(1,0)-R(0,1));
  }
  else {
    res(0) = theta*(sqrt((R(0,0)-cosTh)/(1.0-cosTh))); // Theta near PI
    if ((R(2,1)-R(1,2))<0), res(0) = -res(0);
    res(1) = theta*(sqrt((R(1,1)-cosTh)/(1.0-cosTh)));
    if ((R(0,2)-R(2,0))<0), res(1) = -res(1);
    res(2) = theta*(sqrt((R(2,2)-cosTh)/(1.0-cosTh)));
    if ((R(1,0)-R(0,1))<0), res(2) = -res(2);
  }
  return res;
}

Eigen::Matrix3d oscr::axisAngleToRotation(const Eigen::Vector3d& w)
{
  Eigen::Matrix3d R, S;
  Eigen::Vector3d axis;
  double angle = w.norm();
  axis = w.normalized();
  S = skew(axis);
  return Eigen::Matrix3d::Identity() + sin(angle)*S + (1-cos(angle))*S*S;
}

Eigen::Matrix3d oscr::rotationMatrix(const char& axis, const double& angle)
{
  double ca = cos(angle);
  double sa = sin(angle);
  Eigen::Matrix3d res;
  if (axis=='x'), res << 1, 0, 0, 0, ca, -sa, 0, sa, ca;
  else if (axis=='y'), res << ca, 0, sa, 0, 1, 0, -sa, 0, ca;
  else if (axis=='z'), res << ca, -sa, 0, sa, ca, 0, 0, 0, 1;
  else, res = Eigen::Matrix3d::Identity();
  return res;
}

Eigen::Matrix3d oscr::rotationMatrix(const Eigen::Vector3d& axis, const double& angle)
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

```
{
  Eigen::Vector3d w; double normaxis = axis.norm();
  w = angle/normaxis*axis;
  return axisAngleToRotation(w);
}

double oscr::sgn(const double& x)
{
  if (x>=0), return 1.0;
  else, return -1.0;
}
```


Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Apéndice D

Funciones y Documentación

Este apéndice presenta una parte de la documentación, donde se muestra las funciones utilizadas y lo que hacen. La documentación real es mucho más extensa y debido a los hipervínculos no está diseñada para ser imprimible. Sin embargo, se puede obtener en el CD adjunto de esta tesis, o compilando el paquete `oscr` con las indicaciones provistas en el respectivo repositorio (que consisten básicamente en ejecutar `make doc`, asumiendo que Doxygen está instalado).

RobotModel

Las siguientes son funciones y documentación (breve) de la clase RobotModel.

	RobotModel (const bool &has_floating_base=true) Constructor for the base model class. More...
	~RobotModel () Destructor. More...
bool	**hasFloatingBase** () const Get information about whether or not a floating base exists. More...
std::vector< std::string >	**jointNames** () const Get the joint names. More...
Eigen::VectorXd	**jointMaxAngularLimits** () const Get the maximum joint angluar limits. More...
Eigen::VectorXd	**jointMinAngularLimits** () const Get the minimum joint angluar limits. More...
Eigen::VectorXd	**jointVelocityLimits** () const Get the joint velocity limits. More...
std::map< std::string, unsigned int >	**mapLinkNamesIDs** () const Get the map between the link names and their IDs. More...
Eigen::VectorXd	**getJointConfig** () Get the current joint configuration. More...

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

virtual bool	**loadURDF** (const std::string &model_name, const bool &has_floating_base=true, const bool &verbose=false)=0 Load the URDF model. More...
virtual unsigned int	**ndof** () const =0 Get total number of degrees of freedom. More...
virtual unsigned int	**ndofActuated** () const =0 Get number of actuated degrees of freedom. More...
virtual std::string	**floatingLink** () const =0 Get the link corresponding to the floating base (link number 0) More...
virtual void	**updateJointConfig** (const Eigen::VectorXd &q)=0 Update the joint configuration. More...
virtual Eigen::Vector3d	**linkPosition** (const unsigned int &link_number, const Eigen::Vector3d &local_pos=Eigen::Vector3d::Zero()) const =0 Get the position of a point ($x \in \mathbb{R}^3$) in a link with respect to the world frame. More...
virtual Eigen::Vector3d	**linkPosition** (const std::string &link_name, const Eigen::Vector3d &local_pos=Eigen::Vector3d::Zero()) const =0 Get the position of a point ($x \in \mathbb{R}^3$) in a link with respect to the world frame. More...
virtual Eigen::Vector4d	**linkOrientation** (const unsigned int &link_number) const =0 Get the orientation of a link with respect to the world frame. More...
virtual Eigen::Vector4d	**linkOrientation** (const std::string &link_name) const =0 Get the orientation of a link with respect to the world frame. More...
virtual Eigen::VectorXd	**linkPose** (const unsigned int &link_number, const Eigen::Vector3d &local_pos=Eigen::Vector3d::Zero()) const =0 Get the pose of a link with respect to the world frame. More...
virtual Eigen::VectorXd	**linkPose** (const std::string &link_name, const Eigen::Vector3d &local_pos=Eigen::Vector3d::Zero()) const =0 Get the pose of a link with respect to the world frame. More...
virtual Eigen::MatrixXd	**linearJacobian** (const unsigned int &link_number, const Eigen::Vector3d &local_pos=Eigen::Vector3d::Zero()) const =0 Get the Jacobian corresponding to the linear velocity. More...
virtual Eigen::MatrixXd	**angularJacobian** (const unsigned int &link_number) const =0 Get the Jacobian corresponding to the angular velocity. More...
virtual Eigen::MatrixXd	**geometricJacobian** (const unsigned int &link_number, const Eigen::Vector3d &local_pos=Eigen::Vector3d::Zero()) const =0 Get the full geometric Jacobian. More...
Eigen::VectorXd	**setFeetOnGround** (const std::string &foot1, const std::string &foot2, const double &sole_dist) Set floating base for both feet on the ground (robot is vertical!). More...
virtual Eigen::VectorXd	**comPosition** ()=0 Get the robot Center of Mass with respect to the world frame. More...
virtual Eigen::MatrixXd	**comJacobian** () const =0 Get the Center of Mass Jacobian. More...

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

KineTask

Las siguientes son funciones y documentación (breve) de la clase KineTask.

Public Member Functions

	KineTask (**RobotModel** *rmodel, const std::string &taskName="KineTask", const double &gain=1.0) Constructor. More...
	~KineTask () Destructor. More...
void	**setName** (const std::string &taskName) Set the kinematic task name. More...
std::string	**getName** () const Get the kinematic task name. More...
void	**setGain** (const double &gain) Set a kinematic constant task gain. More...
void	**setAdaptiveGain** (const double &max_gain, const double &min_gain, const double &rate_descent) Set an adaptive kinematic gain. More...
double	**getGain** (const double &error=0.0) Get the kinematic task gain. More...
void	**setDesiredValue** (const Eigen::VectorXd &desiredValue) Set the desired value for the kinematic task. More...
Eigen::VectorXd	**getDesiredValue** () const Get the desired value for the kinematic task. More...
void	**setWeight** (const double &weight) Set the task weight (only used with KinematicSolverWQP) More...
double	**getWeight** () const Get the task weight (only used with KinematicSolverWQP) More...
unsigned int	**getTaskDim** () const Get the dimension of the task (this is inferred from the size of the sensed value) More...
Eigen::VectorXd	**getError** () const Get the task error e. More...
void	**updateJointConfig** (const Eigen::VectorXd &q) Update the joint configuration. More...
void	**keep** (const double &gain) Keep the task with the current values. More...
void	**getTaskBounds** (Eigen::VectorXd &bound_min, Eigen::VectorXd &bound_max) Get the bounds for kinematic tasks. More...
virtual void	**getSensedValue** (Eigen::VectorXd &xsensed) const =0 Get the sensed value for the kinematic task given a joint configuration. More...
virtual void	**getTaskJacobian** (Eigen::MatrixXd &Jacobian) const =0 Get the task Jacobian for a given configuration. More...

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

OSIKSolver

Las siguientes son funciones y documentación (breve) de la clase OSIKSolver. Las clases derivadas utilizan la misma interfaz.

	OSIKSolver (**RobotModel** *rmodel, const Eigen::VectorXd &qinit, const double &dt) Constructor. More...
	~OSIKSolver () Destructor. More...
void	**pushTask** (**KineTask** *task) Push a task into the stack of tasks. More...
void	**popTask** () Remove the last added task (pop) from the stack of tasks. More...
void	**removeTask** (const std::string &taskName) Remove a task from the stack of tasks using the task name. More...
void	**getTaskStack** (std::vector< **KineTask** * > &stack_tasks) Get the stack containing pointers to the currently existing tasks. More...
virtual void	**getPositionControl** (const Eigen::VectorXd &q, Eigen::VectorXd &qdes)=0 Solve the system and get the desired position control for the given configuration. More...
unsigned int	**getTime** () Get the internal time. More...
void	**printStack** ()

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Bibliografía

[1] C. Ruiz-Daza. Control kinect de un brazo robótico de 5 gdl a través de una interfaz intuitiva, 2014. Tesis de Pregrado, UCSM, Arequipa, Perú.

[2] W. Carera-Soria. Diseño y construcción del subsistema electrónico para el control de un brazo robot de 5 gdl, 2015. Tesis de Maestría, PUCP, Lima, Perú.

[3] A. Gonzáles-Chávez. Diseño del sistema de control de un robot tipo puma utilizando lógica difusa, 2013. Tesis de Pregrado, URP, Lima, Perú.

[4] C. Luyo-Gonzáles. Diseño e implementación de un sistema de control e interfaz para un brazo robótico de 5 gdl, 2015. Tesis de Pregrado, PUCP, Lima, Perú.

[5] R. Murray, Z. Li, and S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC press, 1994.

[6] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: Modelling, Planning and Control*. London: Springer London, 2009.

[7] M.W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, 2006.

[8] Y. Ma, S. Soatto, J. Kosecka, Y. Ma, S. Soatta, J. Kosecka, and S. Sastry. *An invitation to 3-D vision*, volume 6. Springer, 2004.

[9] Oussama Khatib. Advanced robotic manipulation, 2005.

[10] Roy Featherstone. *Rigid body dynamics algorithms*, volume 49. Springer New York, 2008.

[11] Kevin M Lynch and Frank C Park. *Modern Robotics*. Cambridge University Press, 2017.

[12] P-B Wieber. Holonomy and nonholonomy in the dynamics of articulated motion. In *Fast motions in biomechanics and robotics*, pages 411–425. Springer, 2006.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

[13] Shuuji Kajita, Sophie Sakka, Hirohisa Hirukawa, Kensuke Harada, and Kazuhito Yokoi. *Introduction à la commande des robots humanoïdes: De la modélisation à la génération du mouvement*. springer, 2009.

[14] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[15] L. Sentis. *Synthesis and Control of Whole-Body Behaviors in Humanoid Systems*. PhD thesis, Stanford University, 2007.

[16] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation*, 3(1):43–53, 1987.

[17] C. Samson, B. Espiau, and M. Le Borgne. *Robot control: the task function approach*. Oxford University Press, 1991.

[18] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. of the ASME. Journal of Applied Mechanics*, 22:215–221, 1955.

[19] J.J. Craig. *Introduction to robotics*. Addison-Wesley Reading, MA, 1989.

[20] D.L. Peiper. *The kinematics of manipulators under computer control*. PhD thesis, Stanford University, 1968.

[21] M. Raghavan and B. Roth. Inverse kinematics of the general 6r manipulator and related linkages. *Journal of Mechanical Design*, 115(3):502–508, 1993.

[22] S.R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *University of California, San Diego, Typeset manuscript*, 2004.

[23] Y. Nakamura. *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[24] K. L. Doty, C. Melchiorri, and C. Bonivento. A Theory of Generalized Inverses Applied to Robotics. *The International Journal of Robotics Research*, 12:1–19, 1993.

[25] A. Ben-Israel and T.N.E. Greville. *Generalized inverses: Theory and applications*. Springer, 2003.

[26] D.E. Whitney. The mathematics of coordinated control of prosthetic arms and manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 94:303, 1972.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

[27] B. Siciliano and J.J. Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *Fifth Intl. Conf. on Advanced Robotics (ICAR)*, pages 1211–1216, 1991.

[28] G. Tevatia and S. Schaal. Inverse kinematics for humanoid robots. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 1, pages 294–299, 2000.

[29] J. Salini, S. Barthelemy, and P. Bidaud. Lqp controller design for generic whole body motion. *Climbing and Walking Robots and the Support Technologies for Mobile Machines*, 2009.

[30] E. Yoshida, O. Kanoun, C. Esteves, and J-P Laumond. Task-driven support polygon reshaping for humanoids. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 208–213, 2006.

[31] David Galdeano, Ahmed Chemori, Sébastien Krut, Philippe Fraisse, et al. Task-based whole-body control of humanoid robots with zmp regulation, real-time application to a squat-like motion. In *SSD SAC 2014: 11th International Conference on Systems, Analysis and Automatic Control*, 2014.

[32] O. Kanoun, F. Lamiraux, P.B. Wieber, F. Kanehiro, E. Yoshida, and J.P. Laumond. Prioritizing linear equality and inequality systems: application to local motion planning for redundant robots. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2939–2944, 2009.

[33] Luis Sentis and Oussama Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(04):505–518, 2005.

[34] Howie M Choset, Seth Hutchinson, Kevin M Lynch, George Kantor, Wolfram Burgard, Lydia E Kavraki, and Sebastian Thrun. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[35] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 2, pages 995 –1001, 2000.

[36] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. Bagnell, and S. Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research (IJRR)*, 32(9-10):1164–1193, 2013.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE TESIS UCSM

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

[37] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 4569–4574, 2011.

[38] C. Park, J. Pan, and D. Manocha. High-dof robots in dynamic environments using incremental trajectory optimization. *International Journal of Humanoid Robotics*, 2014.

[39] L. Saab, O. Ramos, N. Mansard, P. Souères, and J-Y. Fourquet. Dynamic whole-body motion generation under rigid contacts and other unilateral constraints. *IEEE Transactions on Robotics (TRO)*, 29(2):346–362, April 2013.

[40] Michael Mistry, Jun Nakanishi, Gordon Cheng, and Stefan Schaal. Inverse kinematics with floating base and constraints for full body humanoid robot control. In *IEEE-RAS International Conference on Humanoid Robots*, pages 22–27, 2008.

[41] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35, 2006.

[42] Youngjin Choi, Doik Kim, Yonghwan Oh, and Bum-Jae You. Posture/walking control for humanoid robot based on kinematic resolution of com jacobian with embedded motion. *IEEE Transactions on Robotics*, 23(6):1285–1293, 2007.

[43] P. Baerlocher and R. Boulic. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *The visual computer*, 20(6):402–417, 2004.

[44] Oussama Kanoun, Florent Lamiraux, and Pierre-Brice Wieber. Kinematic control of redundant manipulators: generalizing the task priority framework to inequality tasks. *IEEE Transactions on Robotics*, 27(4):785–792, 2011.

[45] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research (IJRR)*, 5(1):90–98, 1986.

[46] Fabrizio Flacco and Alessandro De Luca. A reverse priority approach to multi-task control of redundant robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2421–2427, 2014.

[47] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[48] H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6 (4):327–363, 2014.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis

REPOSITORIO DE
TESIS UCSM

UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

[49] Scott Meyers. *Effective C++: 55 specific ways to improve your programs and designs*. Pearson Education, 2005.

[50] Simon L Altmann. Hamilton, rodrigues, and the quaternion scandal. *Mathematics Magazine*, 62(5):291–308, 1989.

[51] Matthew T Mason. *Mechanics of robotic manipulation*. MIT press, 2001.

Publicación autorizada con fines académicos e investigativos
En su investigación no olvide referenciar esta tesis